Automatic Summarization of Crowd-Sourced API Usage Scenarios

TABLE 1: Descriptive statistics of the dataset

Threads	Posts	Sentences	Words	Snippet	Lines	Users
3048	22.7K	87K	1.08M	8596	68.2K	7.5K
Average	7.5	28.6	353.3	2.8	7.9	3.9

TABLE 2: Descriptive statistics of the Java APIs in the database.

API	Module	Version	Link	AL	AM	AV
12,451	144,264	603,534	72,589	5.8	11.6	4.2

AM = # Avg Modules Per API, AL = #Avg Links per API, AV = #Avg Versions Per Module

1 Needs for API Usage Summaries

In this section, we further motivate the needs to summarize API usage scenarios from developer forums by conducting a case study on a dataset of 3048 Stack Overflow threads. Our study consisted of two steps: 1) We preprocess the dataset and link each code snippet in the dataset to an API about which the code snippet is provided (Section 1.1) and 2) We study the needs to summarize the code examples linked to an API in the dataset (Section 1.2).

1.1 Prepare Study Dataset

We collect all the code examples and textual contents found in the Stack Overflow threads tagged as 'Java+JSON', i.e., the threads contained discussions and opinions related to the json-based software development tasks using Java APIs. In Table 1 we show descriptive statistics of the dataset. There were 22,733 posts from 3,048 threads with scores greater than zero. We did not consider any post with a negative score because such posts are considered as not helpful by the developers in Stack Overflow. There were 8,596 *valid* code snippets (e.g., Java code) and 4,826 invalid code snippets (e.g., XML block)[1]. On average, each valid snippet contained at least 7.9 lines. The last column "Users" show the total number of distinct users that posted at least one answer/comment/question in those threads. On average, around four users participated in one thread. More than one user participated in 2,940 threads (96.4%). A maximum of 56 distinct users participated in one thread [13].

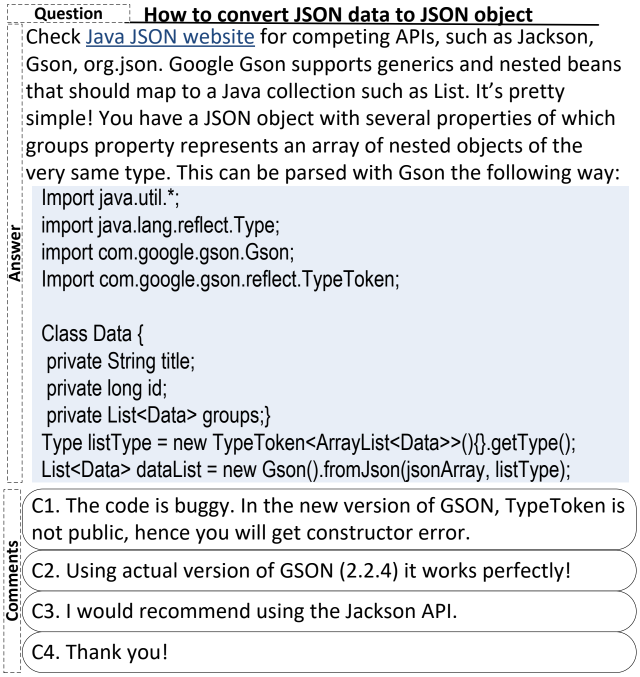
We associate each valid code snippet to an API about which the snippet is provided. To associate a code example to an API, we use a heuristic-based algorithm. The inputs to the algorithm are our API database, a code example, and the Stack Overflow thread where the code example is found. The output is an API name as mentioned in the thread with regards to the code example. We detect an API as an API name as mentioned in the textual contents in the forum posts. This design decision allows us to adopt the definition of an API as originally proposed by Martin Fowler, i.e., a "set of rules and specifications that a software program can follow to access and make use of the services and resources provided by its one or more modules" [30]. For example, in Figure 1, we consider the followings as APIs: 1) Google GSON, 2) Jackson, 3) java.util, and 4) java.lang. Each API thus can contain a number of modules (e.g., packages) and elements (e.g., API types, such as class, interface, etc.). This abstraction is also consistent with the Java official documentation. For example, the `java.time` packages are denoted as the Java date APIs in the new JavaSE official tutorial [12]). As we observe in Figures 1, **??** and **??**, this is also how APIs can be mentioned in online forum posts.

Our API database consists of 1) all the Java APIs collected from the online Maven Central repository [20],[2] and 2) all the Java official APIs from JavaSE (6-8) and EE (6 and 7). In Table 2, we show the summary statistics of our API database. All of the APIs (11,576) hosted in Maven central are for Java. We picked the Maven Central because it is the primary source for hosting Java APIs with over 70 million downloads every week [5].

To determine the association between a code example and an API name as mentioned in a forum post, we first create a Mention Candidate List (MCL) for each API mention by identifying all the API names in our API database that are similar to the API mention. For example, in Figure 2, we show a partial Mention Candidate List for the API mention 'Gson' in Figure 1. Each rounded rectangle denotes an API candidate with its name at the top and one or more module names at the bottom (if module names matched). In reality, this list contains 129 APIs. We associate a code example to an API mention by learning how API elements in the code example may be connected to a candidate API in the mention candidate lists of API mentions in the same post. We call this proximity-based learning, because we start to match with the API mentions that are closer to the code example in the forum before with the API mentions that are further away. For example, in Figure 1, we associate the code example to the API Google Gson, i.e., `com.google.code.gson` in Figure 2. We do this because Google Gson is mentioned in the post text before

[1] We detect valid code snippet using a technique previously proposed by Dagenais and Robillard [2], which uses Java class and method signatures to differentiate Java code examples from non-Java code examples.

[2] We consider a binary file (e.g., jar) of a project from Maven as an API



Question **How to convert JSON data to JSON object**

Answer

Check Java JSON website for competing APIs, such as Jackson, Gson, org.json. Google Gson supports generics and nested beans that should map to a Java collection such as List. It's pretty simple! You have a JSON object with several properties of which groups property represents an array of nested objects of the very same type. This can be parsed with Gson the following way:

```
Import java.util.*;
import java.lang.reflect.Type;
import com.google.gson.Gson;
Import com.google.gson.reflect.TypeToken;

Class Data {
 private String title;
 private long id;
 private List<Data> groups;}
Type listType = new TypeToken<ArrayList<Data>>(){}.getType();
List<Data> dataList = new Gson().fromJson(jsonArray, listType);
```

Comments

C1. The code is buggy. In the new version of GSON, TypeToken is not public, hence you will get constructor error.

C2. Using actual version of GSON (2.2.4) it works perfectly!

C3. I would recommend using the Jackson API.

C4. Thank you!

Fig. 1: How API usage scenarios are discussed in forum posts.

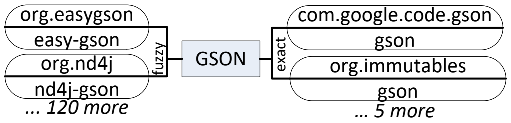


Fig. 2: Partial Mention Candidate List of GSON in Figure 1

the code example and the types (`Gson`, `TypeToken`) in the code example of Figure 1 can also be found in the Gson API. We parse a code snippet using a hybrid parser combining ANTLR [14] and Island Parser [10]. We detect API elements in code examples using Java naming conventions, similar to previous approaches (e.g., camel case for Class names) [2], [17].

1.2 Assessing Summarization Needs of Code Snippets

In Table 3, we show the distribution of code snippets matched to APIs. The 8596 code examples are associated with 175 distinct APIs.

Observation 1. The majority (60%) of the code examples were associated to five APIs for parsing JSON-based files and texts in Java: java.util, org.json, Google Gson, Jackson, and java.io. Some API types are more widely used in the code examples than others. For example, the `Gson` class from Google Gson API was found in 679 code examples out of the 1053 code examples linked to the API (i.e., 64.5%). Similarly, the `JSONObject` class from the org.json API was found in 1324 of 1498 code examples linked to the API (i.e., 88.3%). Most of those code examples also contained other types of the APIs. Therefore, if we follow the documentation approach of Baker [25], we would have at least 1324 code examples linked to the Javadoc of `JSONObject` for the API org.json. This is based on the parsing of our 3048 Stack Overflow threads. There are more than 14.9 million threads in Stack Overflow tagged as 'Java'. Clearly, such a Javadoc annotation approach to facilitate crowd-sourced API documentation might overwhelm the software developers who will want to use the documentation.

TABLE 3: Distribution of Code Snippets By APIs

Overall				Top 5			
API	**Snippet**	**Avg**	**STD**	**Snippet**	**Avg**	**Max**	**Min**
175	8596	49.1	502.7	5196	1039.2	1951	88

TABLE 4: Code clones per API in the mined code examples

	Overall				Top 5			
	API	**%**	**Avg**	**STD**	**Avg**	**Max**	**Min**	**STD**
Clones	96	54.9%	11	20.3	38.8	84	4	29.2

The code examples posted about an API in the forums can be similar to each other. For example, in Figure **??**, all the three code examples can be considered as clones. Given as input all the code examples associated to an API, we detect clones using the NiCad clone detection tool [18]. We use NiCad 3, which detects near-miss clones. Following [26], we set a minimum similarity of 60% and a block size of five lines.

Observation 2. We found on average 11 clones in the code examples of 54.9% of the APIs (see Table 4). Most of the clones are found in unrelated posts. Therefore, the prevalence of clones denote that similar feature of an API was consulted more than once, demonstrating the needs from multiple developers.

Indeed, any summarization efforts of the mined code examples to support API documentation may be required only if the code examples are not found in the API official documentation. To investigate how the different types (class, annotation, etc.) of an API are used in the code examples of our dataset as well as in API official documentation, we determine their coverage in the two resources. We focus on the coverage of API types, because developers can only use the different features of a Java API through its API types.

$$\text{Coverage (Resource)} = \frac{|\#\text{Types in Resource Code Examples}|}{|\#\text{All API Types}|} \tag{1}$$

Here 'Resource' denotes either the official documentation or our mined code examples of an API. We compute the coverage for the top five APIs, because our dataset mainly consists of Java API JSON parsing. As such, the mined code examples of the top five APIs should offer sufficient coverage of the different types of the APIs to learn and use JSON-based parsing using the APIs.

Observation 3. For the five APIs, the mined usage scenarios offer more coverage than the official documentation. For three APIs (java.util, java.io, and org.json), the official documentation web pages did not have any code examples and only contained standard Javadocs. However, the coverage is between 14.9% (java.util) - 62.5% (org.json) for those APIs in the mined code examples. In our mined code examples, the overall coverage is the highest for the Gson API (81.8%). In the code examples of official documentation, the coverage of API types was also the highest for the Gson API types (only 25%).

Summary: Most of the mined code examples in our dataset of top five APIs are not found in their official documentation. Hundreds and even thousands of code examples can be linked to an API type, and clones are prevalent in the code examples. Therefore, it can be difficult to improve API documentation by simply annotating the Javadoc of the API type with the code examples.

2 Evaluation

We answer the following three research questions using three separate user studies.

RQ1. How useful is the [TOOL] API usage scenario summarization engine to support development tasks?

RQ2. How useful are the [TOOL] usage summaries over API official and informal documentation?

The goal of the automatic summarization of APIs usage scenarios in [TOOL] is to assist developers in finding the right solutions to their coding tasks with relative ease than other resources. Therefore, to assess the effectiveness of [TOOL] API usage summaries in real-world coding tasks (RQ1), we conducted a user study involving programmers. A total of 31 programmers participated. The purpose of developing the usage summaries in [TOOL] is to add benefits over existing API documentation. Therefore, to examine the usefulness of [TOOL] over both API formal and informal documents (RQ2), we conducted a survey with the 31 the participants who completed the coding tasks. We asked them questions with regards to the observed value [TOOL] might have added over the API documents in their coding tasks.

The coding tasks (RQ1) were completed by a total of 31 participants. The survey for RQ2 was answered by 29 out of the 31 participants who completed coding tasks in RQ1. The surveys were conducted separately. The surveys were conducted after the participants completed their coding tasks. Among the 31 participants, 18 were professional developers. The rest of the participants were recruited from four universities.

Evaluation Corpus. We applied the four summarization algorithms on the datasets used in our empirical study in Section 1. The [TOOL] online web-site currently indexes the mined and summarized usage scenarios of the APIs found in this dataset. We used the [TOOL] online usage summaries to conduct the three user studies (RQ1-3). As we noted in Section 1, this dataset consists of a total of 3048 threads from Stack Overflow tagged as 'Java+JSON'. As such, we expect to see code examples discussing about Java APIs for JSON parsing in the posts. As we observed in Section 1, this dataset contains numerous usage scenarios from multiple competing APIs to support JSON-based manipulation in Java (e.g., REST-based architectures, microservices, etc.). JSON-based techniques can be used to support diverse development scenarios, such as, serialization of disk-based and networked files, lightweight communication between server and clients and among interconnected software modules, messaging format over HTTP as a replacement of XML, in encryption techniques, and on-the-fly conversion of Language-based objects to JSON formats, etc.

We describe the approaches we used to design the user studies. We introduce the participants involved in the studies in Section 2.2. We explain the metrics we used to assess the survey responses in Section 2.3. We present the results of the three studies in Section 3.

2.1 Study Design

2.1.1 How useful is the [TOOL] API usage scenario summarization engine to support development tasks? (RQ1)

The *goal* of this study was to analyze the *effectiveness* of [TOOL] to assist in coding tasks. The *objects* were the APIs and their usage summaries in [TOOL], Stack Overflow, and Official documentation. The *subjects* were the participants who each completed four coding tasks.

• **Analysis Dimensions.** Because developers often refer to API documentation (formal and informal) to assist them in completing tasks correctly and quickly [15], we compared the usefulness of [TOOL] over the other development resources (Stack Overflow, API formal documentation, and everything). We assessed the effectiveness of each resource to assist in the coding tasks along three dimensions:

Correctness ***of the provided coding solution***

We considered a solution as correct if it adhered to the provided specification and strived for functional correctness [3]. Because each coding solution in our study for a given task would require the usage of a given API, we define correctness based on two assessments: whether the correct API was used, and whether the required API elements were used.

Time ***taken to complete a coding task***

We defined the time spent to complete a coding task using a given development resource as the total time took for a participant (1) to read the task description, (2) to consult the given development resolution for a solution and (3) to write the solution in the provided text box of the solution.

Effort ***spent to complete a coding task***

We defined the effort spent to complete a coding task using a given development resource as how hard the participant had to work to complete the task. We analyzed the effort using the NASA TLX index [8] which uses six measures to determine the effort spent to complete a task. TLX computes the workload of a given task by averaging the ratings provided for six dimensions: mental demands, physical demands, temporal demands, own performance, effort, and frustration.

• **Design of the study.** Our design was a between-subject design [31], with four different groups of participants, each participant completing four tasks in four different settings. Our between-subject design allowed us to avoid repetition by using a different group of subjects for each treatment. To avoid potential bias in the coding tasks, we enforced the homogeneity of the groups by ensuring that: 1) no group entirely contained participants that were only professional developers or only students, 2) no group entirely contained participants from a specific geographic location and–or academic institution, 3) each participant completed the tasks assigned to him independently and without consulting with others 4) each group had same number of four coding tasks 5) each group had exposure to all four development settings as part of the four development tasks. The use of balanced groups simplified and enhanced the statistical analysis of the collected data.

• **Coding Tasks.** The four tasks are described in Table 5. Each task was required to be completed using a pre-selected API. Thus for the four tasks, each participant needed to use four different APIs: Jackson [4], Gson [7], Xstream [29], and Spring framework [19]. Jackson and Gson are two of the most popular Java APIs for JSON parsing [22]. Spring is one of the most popular web framework in Java [6] and XStream is well-regarded for its efficient adherence to the JAXB principles [24], i.e., XML to JSON conversion and vice versa.

All of the four APIs can be found in the list of top 10 most discussed APIs in our evaluation corpus. The APIs are mature and are fairly large and thus can be hard to learn. The largest API is the Spring framework 5.0.5 with a total of 3687 types (Class, Annotation, Exception, and Annotation), followed by Jackson 2.2 (467 types), XStream 1.4.10 (340 types), and Gson 2.8.4 (34 types). For Jackson API, we counted the code types that are shipped with the core modules (jackson-core, databind, and annotations). The Jackson API has been growing with the addition of more modules (e.g., jackson-datatype). The four APIs have a total of 4528 types. In comparison, the entire Java SE 7 SDK has a total of 4024 code types, and the entire Java SE 8 SDK has a total of 4240 code types.[3]

- **Task Settings.** For each coding task, we prepared four settings:

SO complete the task only with the help of Stack Overflow;

DO complete the task only with the help of API official documentation

OP complete the task only with the help of [TOOL]

EV complete the task with any resources available (i.e., SO, DO, [TOOL], and Search engines).

The participants were divided into four groups (G1, G2, G3, G4). Each participant in a group was asked to complete the four tasks. Each participant in a group completed the tasks in the order and settings shown in Table 6. To ensure that the participants used the correct development resource for a given API in a given development setting, we added the links to those resources for the API in the task description. For example, for the task TJ and the setting SO, we provided the following link to query Stack Overflow using Jackson as a tag: https://stackoverflow.com/questions/tagged/jackson. For the task TG and the setting DO, we provided the following link to the official documentation page of the Google GSON API: https://github.com/google/gson/blob/master/UserGuide.md. For the task TX and the setting PO, we provided a link to the summaries of the API XStream in the [TOOL] website. For the task TS with the setting EV, we provided three links, i.e., one from [TOOL] (as above), one from Stack Overflow (as above), an one from API formal documentation (as above). For the 'EV' setting, we further added in the instructions that the participants are free to use any search engine.

- **Task Selection Rationale.** The four tasks were picked randomly from our evaluation corpus of 22.7K Stack Overflow posts. Each task was observed in Stack Overflow posts more than once and was asked by more than one developer. Each task was related to the manipulation of JSON inputs using Java APIs for JSON parsing. The manipulation of JSON-based inputs is prevalent in disk-based, networked as well as HTTP-based message, file, and object processing. Therefore, the Java APIs for JSON parsing offer many complex features to support the diverse development needs. The solution to each task spanned over two posts. The two posts are from two different threads in Stack Overflow. Thus the developers could search in Stack Overflow to find the solutions. However, that would require those searching posts from multiple threads in Stack Overflow. All of those tasks are common using the four APIs. Each post related to the tasks was viewed and upvoted more than hundred times in Stack Overflow. To ensure that each development resource was treated with *equal fairness* during the completion of the development tasks, we also made sure that each task could be completed using any of the development resources, i.e., the solution to each task could be found in any of the resources at a given time, without the need to rely on the other resources.

TABLE 5: Overview of coding tasks

Task	API	Description
TJ	Jackson	Write a method that takes as input a Java Object and serializes it to Json, using the Jackson annotation features that can handle custom names in Java object during deserialization.
TG	GSON	Write a method that takes as input a JSON string and converts it into a Java Object. The conversion should be flexible enough to handle unknown types in the JSON fields using generics.
TX	Xstream	Write a method that takes as input a XML string and converts it into a JSON object. The solution should support aliasing of the different fields in the XML string to ensure readability
TS	Spring	Write a method that takes as input a JSON response and converts it into a Java object. The response should adhere to strict JSON character encoding (e.g., UTF-8).

TABLE 6: Distribution of coding tasks per group per setting. SO = Stack Overflow, DO = Javadoc, OP = [TOOL], EV = Everything. TJ, TG, TX, TS = Task Using Jackson, GSON, XStream, Spring, resp.

↓ Group | Setting →	SO	DO	OP	EV
G1	TJ	TG	TX	TS
G2	TS	TJ	TG	TX
G3	TX	TS	TJ	TG
G4	TG	TX	TS	TJ

- **Coding Guide.** A seven-page coding guide was produced to explain the study requirements. Before each participant was invited to complete the study, he had to read the entire coding guide. Each participant was encouraged to ask questions to clarify the study details before and during the study. To respond to the questions the participants communicated with the first author over Skype. Each participant was already familiar with formal and informal documentation resources. To ensure a fair comparison of the different resources used to complete the tasks, each participant was given a brief demo of [TOOL] before the beginning of the study. This was done by giving them an access to the [TOOL] web site.

- **Data Collection Process.** The study was performed in a Google Form, where participation was by invitation only. Four versions of the form were generated, each corresponding to one group. Each group was given access to one version of the form representing the group. The developers were asked to write the solution to each coding task in a text box reserved for the task. The developers were encouraged to use any IDE that they are familiar with, to code the solution to the tasks. Before starting the study, each participant was asked to put his email address and to report his expertise based on the following three questions:

Profession the current profession of the participant.

Experience software development experience in years.

Activity whether or not the participant was actively involved in software development during the study.

Before starting each task, a participant was asked to mark

[3] We used the online official Javadocs of the APIs to collect the type information. Our online appendix contains the list.

down the beginning time. After completing a solution the participant was again asked to mark the time of completion of the task. The participant was encouraged to not take any break during the completion of the task (after he marked the starting time of the task). To avoid fatigue, each participant was encouraged to take a short break between two tasks. Besides completing each coding task, each participant was also asked to assess the complexity and effort required for each task, using the NASA Task Load Index (TLX) [8], which assesses the subjective workload of subjects. After completing each task, we asked each subject to provide their self-reported effort on the completed task through the official NASA TLX log engine at nasatlx.com. Each subject was given a login ID, an experiment ID and task IDs, which they used to log their effort estimation for each task, under the different settings.

• **Variables.** The main independent variable was the development resource that participants use to find solutions for their coding tasks. The other independent variables were participants' development experience and profession. Dependent variables were the values of the following metrics: correctness, time, and effort.

2.1.2 How useful are the [TOOL] usage summaries over API documentation? (RQ2)

The *goal* of this study was to analyze whether [TOOL] can offer benefits over the API documentation and how [TOOL] can be complement the existing API documents. The *objects* were the APIs and their usage summaries in [TOOL], Stack Overflow, and Official documentation. The *subjects* were the participants who each responded to the questions.

We answer two research questions:

RQ2.1. Benefits over Formal Documentation.
How do the collected API usage scenarios offer improvements over formal documentation?

RQ2.2. Benefits Over Informal Documentation.
How do the collected API usage scenarios offer improvements over informal documentation?

For the first question (i.e., benefits over formal documentation), we asked three sub-questions to each developer based on his experience of using [TOOL]:

Improvements. For the tasks that you completed, did the summaries produced by [TOOL] offer improvements over formal documentation?

Complementarity. For the tasks that you completed, how would you complement formal API documentation with [TOOL] results? How would you envision for [TOOL] to complete the formal API documentation.

The first question had a five-point likert scale (Strongly Agree - Strongly Disagree). The second question had two parts, first part with a five-point likert scale and the second part with a open-ended question box.

For the second question (i.e., benefits over informal documentation), we asked two questions to each developer based on his experience of using [TOOL]:

Improvements. How do [TOOL] summaries offer improvements over the Stack Overflow contents for API usage?

Complementarity. How would you envision [TOOL] to complement the Stack Overflow contents about API usages in your daily development activities?

The survey was conducted in a Google doc form.

2.2 Participants

Each of the studies was conducted separately. The survey was conducted after the participants completed their coding tasks. The 31 participants in the coding tasks are divided into four groups (G1-G4). Each of three groups (G1, G3, G4) had eight participants. Among the 31 participants, 18 were recruited through the online professional social network site, Freelancer.com. The rest of the participants were recruited from four universities. Among the participants recruited from the universities, eight reported their profession as students, two as graduate researchers, and rest as software engineers and student. Among the 31 participants 88.2% were actively involved in software development (94.4% among the freelancers and 81.3% among the university participants). Each participant had a background in computer science and software engineering.

The number of years of experience of the participants in software development ranged between less than one year to more than 10 years: three (all of them being students) with less than one year of experience, nine between one and two, 12 between three and six, four between seven and 10 and the rest (nine) had more than 10 years of experience. Among the four participants that were not actively involved in daily development activities, one was a business analyst (a freelancer) and three were students (university participants). The business data analyst had between three and six years of development experience in Java. The diversity in the participant occupation offered us insights into whether and how [TOOL] was useful to all participants in general.

2.3 Study Data Analysis

We analyzed the survey data using statistical and qualitative approaches. For the open-ended questions, we applied an open coding approach [9]. As we analyzed the quotes, themes and categories emerged and evolved during the open coding process.

We analyzed each solution to a coding task using three metrics:

1) **Correctness.** How accurate is the provided solution to support the coding task?
2) **Time.** How much time was spent to code the solution?
3) **Effort.** How much effort was required to code the solution?

We computed the metrics as follows:

1) **Correctness.** To check the correctness of the solution for a coding task, we used the following process: a) We identified the correct API elements (types, methods) used for the coding task. b) We matched how many of those API elements were found in the coding solution and in what order. c) We quantified the correctness of the coding solution using the following equation:

$$\text{Correctness} = \frac{|\text{API Elements Found}|}{|\text{API Elements Expected}|} \tag{2}$$

An API element can be either an API type (class, annotation) or an API method. Intuitively, a complete solution should have all the required API elements expected for the solution. We discarded the following types of solutions: a) **Duplicates.** Cases where the solution of one task was copied into the solution of another task. We identified this by seeing the same solution copy pasted for the two tasks. Whenever this happened, we discarded the second solution. b) **Links.** Cases where developers only provided links to an online resource without providing a solution for the task. We discarded such solutions. c) **Wrong API.** Cases where developers provided the solution using an API that was not given to them.

2) **Time.** We computed the time taken to develop solutions for each task, by taking the difference between the start and the

end time reported for the task by the participant. Because the time spent was self-reported, it was prone to errors (some participants failed to record their time correctly). To remove erroneous entries, we discarded the following type of reported time: a) reported times that were less than two minutes. It takes time to read the description of a task and to write it down, and it is simply not possible to do all such activities within a minute. b) reported times that were more than 90 minutes for a given task. For example, we discarded one time that was reported as 1,410 minutes, i.e., almost 24 hours. Clearly, a participant cannot be awake for 24 hours to complete one coding task. This happened in only a few cases.

3) **Effort.** We used the TLX metrics values as reported by the participants. We analyzed the following five dimensions in the TLX metrics for each task under each setting: **(a)** ***Frustration Level.*** How annoyed versus complacent the developer felt during the coding of the task? **(b)** ***Mental Demand.*** How much mental and perceptual activity was required? **(c)** ***Temporal Demand.*** How much time pressure did the participant feel during the coding of the solution? **(d)** ***Physical Demand.*** How much physical activity was required. **(e)** ***Overall Performance.*** How satisfied was the participant with his performance? Each dimension was reported in a 100-points range with 5-point steps. A TLX 'effort' score is automatically computed as a task load index by combining all the ratings provided by a participant. Because the provided TLX scores were based on the judgment of the participants, they are prone to subjective bias. Detecting outliers and removing those as noise from such ordinal data is a standard statistical process [27]. By following Tukey, we only considered values between the following two ranges as valid: a) Lower limit: First quartile - 1.5 * IQR b) Upper limit: Third quartile + 1.5 * IQR . Here IQR stands for 'Inter quartile range', which is calculated as: IQR = $Q3 - Q1$. Q1 and Q3 stand for the first and third quartile, respectively.

3 Results

In this section, we present the results of the studies.

3.1 How useful is the [TOOL] API usage scenario summarization engine to support development tasks? (RQ1)

Summary: The participants completed four separate coding tasks, under four different settings (using Stack Overflow only, API official documentation only, [TOOL] only, or everything). While using [TOOL], the participants on average coded with more correctness (62.3%), spent less time (18.6 minutes per coding task) and less effort (45.8).

A total of 135 coding solutions were provided by the participants. We discarded 14 as invalid solutions (i.e., link/wrong API). Out of the 135 reported time, we discarded 23 as being spurious. 24 participants completed the TLX metrics (providing 96 entries in total), with each setting having six participants.

Table 7 shows descriptive statistics about the completed coding tasks along the three dimensions. The effort calculation is based on the TLX effort index as calculated by the TLX software. We now discuss the results in more details below.

3.1.1 Correctness of the Coded Solutions

Observation 4. **Participants showed better performance when using [TOOL] only, than when using every available resources (which included [TOOL], formal documentation, Stack Overflow, and Search Engines) during the coding tasks.** The participants reported that they were familiar with Stack Overflow and the official documentations, but that they needed time to learn [TOOL]. Thus, they may have started exploring those resources before resorting to [TOOL] for their coding solution when they were allowed to use all the resources. Several participants (including professional freelancers) mentioned that they found the coding tasks to be difficult. This difficulty contributed to the lower number of participants providing completely accurate solution.

Observation 5. **The correctness of the provided solutions were the lowest when participants used only Stack Overflow. Participants on average spent the highest amount of time and effort per coding solution when using only the formal documentation.** The lowest correctness while using Stack Overflow shows the difficulties that the participants faced to piece together a solution from multiple forum threads that may not be discoverable (e.g., they are not connected together). Since the formal documentation did have all the solutions, it was a matter of finding and piecing those solutions together. The longer time and effort highlights the need for a better presentation format of the contents of official documentations, as identified in our previous research [28].

Observation 6. **The participants showed more accuracy while using [TOOL] for two tasks (TJ and TX) than while using the other resources. For the task TG, they showed lower accuracy (μ = 0.83) while using [TOOL] than while using othe resources. In fact, they achieved almost perfect accuracy for the task TG, while using both Stack Overflow and API official documentation. For the task TS, while the participants showed the highest accuracy using every resources (i.e., EV setting with an average μ = 0.39 and median M = 0.45), [TOOL] was the second best with a slightly lower mean accuracy of 0.38 and a median of 0.4.**

- **Task TG.** Both Stack Overflow and API official documentation were better than [TOOL] to the participants for this task. For this task, the API was the Google Gson API for which the official documentation is well-regarded by developers [23]. The entire solution to the task can be found in the 'Getting Started' page of the API. The 'Getting Started' official documentation page of Gson follows the format of the single page documentation champion by the Python community. Therefore, finding the right information for this task using the formal documentation was very easy for participants; which explains why participants had the highest accuracy using formal documentation for this task. For the same task, participants also performed better while using Stack Overflow than while using only [TOOL]. This is because the participants were able to find the right post in Stack Overflow with keywords like (generic, json conversion). Five participants provided perfect answer using [TOOL] for this task. Compared to Stack Overflow and official documentation, the search for this solution in [TOOL] for this task was non-trivial. The participants needed to look at more than one summaries in [TOOL]. Instead of providing the solution, one participant simply wrote *"no solution found. There is no example code provided by [TOOL] for GSON"*. We considered that as a wrong answer (i.e., accuracy = 0).

- **Tasks TJ and TX.** The participants showed the most accuracy in their coding while using [TOOL] for both of the tasks. For these tasks, the search for the solution in Stack Overflow and official documentation was not trivial as the task TG. Instead, the search through the summaries in [TOOL] was more useful. For

TABLE 7: Summary statistics of the correctness (scale 0 - 1), time (in minutes) and effort (Scale 0-100) spent in the coding tasks

	Correctness				**Time**					**Effort**				
	Avg	**Std**	**Median**	**Perfect**	**Avg**	**Std**	**Median**	**Min**	**Max**	**Avg**	**Std**	**Median**	**Min**	**Max**
SO	0.46	0.45	0.33	33%	22.3	11.5	20	3	44	55.8	22.2	55	5	99
DO	0.5	0.44	0.33	33%	23.7	12.6	21	5	58	63.9	18.9	64	24	99
[TOOL]	**0.62**	0.39	**0.73**	37%	**18.6**	12.5	**15**	2	**55**	**45.8**	26.6	**55**	9	98
EV	0.55	0.41	0.6	33%	19.4	14.3	14.5	2	55	54.8	25.5	62.5	4	94

example, for the task TJ, the solution could be found under one concept that grouped four relevant usage scenarios.

• **Task TS.** [TOOL] was the second best performer (behind EV setting) for this task. The participants seemed to have the most difficulty while completing this task, because the overall average coding accuracy for this task is the lowest among all the tasks. This task involved the usage of the Spring framework in Java. The usage of the API for this task required the participants to find the right annotation types in Spring that can be used to inject customized values into JSON inputs in a Java class. Two of the participants using [TOOL] for TS were students, both had the lowest accuracy among all the participants using [TOOL] for the task. One of the freelancers mentioned that he found this task to be quite hard (he had the lowest accuracy among all the freelancers for this task using [TOOL]). While Spring framework is widely used in the Industry, it is a big framework and thus learning how to use it takes time and effort (e.g., 'why is learning spring framework so hard' [16]). Thus, when discoverability of a solution is easier in [TOOL] compared to the other two resources, developers showed more accuracy using [TOOL].

3.1.2 Time Spent to Code Solutions

Observation 7. **The participants spent less time while using [TOOL], than the other resources for the two tasks (TG, TS). For the task TX, participants spent on average 19.4 minutes using [TOOL] was the second best behind (17.3 minuties) only Stack Overflow (the lowest completion time). Out of all tasks, the participants spent the longest time (on average) while completing the task TJ across all the settings.**

• **Task TJ.** Even though the participants spent the most amount of time (on average) to complete the task TJ, they achieved the best accuracy for the task while using [TOOL]. The task TJ involved the API Jackson. This task required participants to use the annotation features in Jackson to deserialize a Java object to JSON. Understanding the annotation feature in Jackson can be a non-trivial task and it may not appeal to everyone (ref. the blog post 'Jackson without annotation' [11]). Therefore, it may take more time for a developer to be familiar with this feature.

• **Task TG.** For the task TG, participants spent the lowest time while using [TOOL]. However, that number is low due to the one participant who complained that he could not find a solution using [TOOL]. He only spent four minutes on this task (that involved reading the task description, opening [TOOL], and searching for the solution in [TOOL]).

• **Tasks TX and TS.** While using the formal documentation, participants spent the most amount of time for the task TX, but they also showed the second best accuracy among all the four tasks while using formal documentation (behind only [TOOL]). Therefore, it may be that participants who coded with more accuracy were more cautious to explore the resources. Thus it took them a longer time. Nevertheless, it is encouraging that participants spent the lowest time while using [TOOL] for the two tasks (TS and TX), but still achieved the highest accuracy (*average*) for the tasks.

3.1.3 Effort Spent to Code Solutions

Observation 8. **The overall effort across all the tasks was the lowest for [TOOL], across the four settings (see Table 7). The participants spent the least amount of effort while using [TOOL] for one task (TS). For two tasks (TG, TX), [TOOL] was the second best. For the other task (TJ), developers spent more effort when using the official documentation in comparison to when they used [TOOL] only.**

• **Task TG.** For the task TG, developers showed the lowest frustration and best satisfaction while using the formal documentation. The solution to this task was easily found in the 'Getting Started' page of the Gson formal documentation. Recall that, participants had the best accuracy using the formal documentation for this task. The participants, however, felt more time pressure while completing the task using [TOOL]. This is perhaps due to their relative less familiarity with the [TOOL] interface and their expectation of finding a quicker solution while using [TOOL].

• **Tasks TJ, TS, and TX.** For TJ, participants spent almost similar effort while using [TOOL] and Stack Overflow. However, their developed code had more accuracy while using [TOOL]. Task TJ required using the Jackson API. The official documentation of Jackson is considered to be complex [21]. The summaries in [TOOL] did lessen the effort by more than 10% on average.

For the task TS involving the Spring Framework, [TOOL] considerably outperformed the other settings for this task (less than 40% for [TOOL] to more than 70% for formal documentation).

While analyzing the TLX values along the reported dimensions (frustrations, time pression, etc.), we found that participants expressed the lowest frustration while using [TOOL] for two tasks (TS, TX). Participants, however, felt more satisfaction about their coding solution while using [TOOL] for three task (TJ, TS, TX).

3.2 How useful are the [TOOL] usage summaries over API official and informal documentation? (RQ2)

Summary: More than 80% of the participants considered the usage summaries in [TOOL] as an improvement over API official documentation and Stack Overflow by offering an increase in productivity, confidence and time spent.

RQ2.1 How do the collected usage scenarios offer improvements over formal documentation?

In Figure 3, we show the responses of the participants [TOOL] offered improvements whether in their four development tasks over formal documentation and if so which summary.

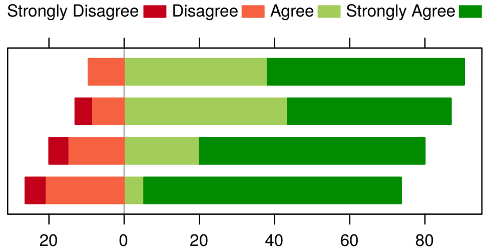


Fig. 3: Developers' preference of [TOOL] summaries over formal documentation

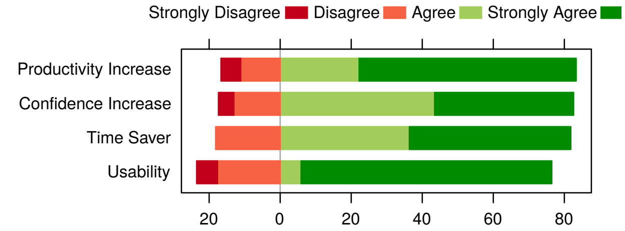


Fig. 4: Benefits of [TOOL] over informal documentation

Observation 9. **The participants mostly (more than 90%) agreed that the [TOOL] usage summaries did indeed offer improvements over formal documentation. The participants appreciated the following improvements in [TOOL] over API official documentation: 1) Uptodateness of the scenarios, and 2) Presence of sentiments to validate the effectiveness of usage scenarios.** The participants suggest to combine the strengths of each summary into their decision making process, *"Statistical summary would help me to find the users decisions about the API. I would chose conceptual summary and over other summaries rather than formal documentation because these provides the usage example with positive and negative reaction."*

Observation 10. **One particular reason of moving to [TOOL] from official documentation is that [TOOL] can stay up to date, while API official documentation can be often obsolete [1].** For example, one participant still considers API official documentation as the starting point to learn an API, but wants to leverage [TOOL] more over time as they want to learn more about the API: *"Formal documentation will almost always be the starting point for new APIs. But as the APIs start to grow, [TOOL] will serve as the most useful tool to find correct solutions to the problems in less time, become familiar with the trends and compare different alternatives."*

Observation 11. **Because formal documentation in Java are mainly javadocs with a single page view for each class, the type-based summary was the most relevant for such documentation. Therefore, it is intuitive that participants thought that the [TOOL] type-based summaries were a considerable upgrade over the formal javadocs.** The concept-based summaries while not present in any formal documentation, came in second position, just slightly below the type-based summary, showing the needs for recommendations based on clustering of similar API usage scenarios, in the formal documentation.

Observation 12. **When we asked the participants how they would like [TOOL] to introduce such benefits to the formal documentation, a majority of developers participants (79%) wanted [TOOL] results to be integrated into the formal documentation, rather than replacing the formal documentation (65.5%) altogether with the summaries of [TOOL].** According to one participant, the integration of such informal viewpoints into API official documentation can suit both naive and experienced developers *"The formal API documentations are complete and are perfectly tuned now a days to suit the naivest developer. One thing that these miss on are the problems and discussion forum, for which StackOverflow was created. [TOOL] could complement the formal documentation, if the conceptual summaries are incorporated within the API documentations."*

RQ2.2 How do the collected usage scenarios offer improvements over informal documentation?

In Figure 4, we show how developers responded to our questions on whether in their coding tasks [TOOL] offered benefits over the informal documentation.

Observation 13. **More than 80% of the participants agreed that [TOOL] summaries offered increases in productivity, confidence, and saved time over the informal documentation.** According to one developer *"there are some sections like conceptual summary and code example would be useful for developer on daily basis"*.

Observation 14. **The participants considered learning an API using [TOOL] could be quicker than while using Stack Overflow, because [TOOL] synthesizes the information from Stack Overflow by APIs using both sentiment and source code analyses.** According to one participant *"It is quicker to find solution in [TOOL] since the subject is well covered and useful information is collected."* **The participants mentioned that [TOOL] offers more features than Stack Overflow to learn an API**: *"[TOOL] has more feature set to help the developers find out the better API for a task. The evaluation and nice presentation of positive and negative sentiments help them decide an API."*

Observation 15. **The participants recommended to facilitate a better search feature in [TOOL], so that they can search for API tasks**. According to one participant, such feature can improve the learning of an API: *"Just for learning any new API, it is great. Because, the most challenging thing I faced in tech based domains is the collection of my vocabulary. To search or ask something we need to first know the proper question. If we do not know with what terms I would ask something, then no only a tool but not even a human can answer me. "*

References

[1] B. Dagenais. *Analysis and Recommendations for Developer Learning Resources*. PhD in Computer Sscience, McGill University, 2012.

[2] B. Dagenais and M. P. Robillard. Recovering traceability links between an API and its learning resources. In *Proc. 34th IEEE/ACM Intl. Conf. on Software Engineering*, pages 45–57, 2012.

[3] D. D. Dunlop and V. R. Basili. A comparative analysis of functional correctness. *ACM Computing Surveys*, 14(2):229–244, 1982.

[4] FasterXML. *Jackson*. https://github.com/FasterXML/jackson, 2016.

[5] B. Fox. *Now Available: Central download statistics for OSS projects*, 2017.

[6] fryan. *Language Framework Popularity: A Look at Java, June 2017*. http://redmonk.com/fryan/2017/06/22/language-framework-popularity-a-look-at-java-june-2017/, 2017.

[7] Google. *Gson*. https://github.com/google/gson, 2017.

[8] S. G. Hart and L. E. Stavenland. Development of NASA-TLX (Task Load Index): Results of empirical and theoretical research. pages 139–183, 1988.

[9] M. Miles and A. Huberman. *Qualitative Data Analysis: An Expanded Sourcebook*. SAGE Publications, 1994.

[10] L. Moonen. Generating robust parsers using island grammars. In *Proc. Eighth Working Conference on Reverse Engineering*, pages 13–22, 2001.

[11] M. Nikolaidis. *Jackson without annotations*. https://manosnikolaidis.wordpress.com/2015/08/25/jackson-without-annotations/, 2017.

[12] Oracle. *The Java Date API*. http://docs.oracle.com/javase/tutorial/datetime/index.html, 2017.

[13] S. Overflow. http://stackoverflow.com/questions/338586/, 2010.

[14] T. Parr. *The Definitive ANTLR Reference: Building Domain-Specific Languages*. Pragmatic Bookshelf, 1st edition, 2007.

[15] L. Ponzanelli, G. Bavota, M. Di Penta, R. Oliveto, and M. Lanza. Prompter: Turning the IDE into a self-confident programming assistant. *Empirical Software Engineering*, 21(5):2190–2231, 2016.

[16] Quora. *Why is learning Spring Framework so hard?* https://www.quora.com/Why-is-learning-Spring-Framework-so-hard, 2017.

[17] P. C. Rigby and M. P. Robillard. Dicovering essential code elements in informal documentation. In *Proc. 35th IEEE/ACM International Conference on Software Engineering*, pages 832–841, 2013.

[18] C. K. Roy and J. R. Cordy. NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In *Proc. 16th IEEE International Conference on Program Comprehension*, pages 172–181, 2008.

[19] P. Software. *Spring Framework*. https://spring.io/, 2017.

[20] Sonatype. *The Maven Central Repository*. http://central.sonatype.org/, 22 Sep 2014 (last accessed).

[21] StackOverflow. http://stackoverflow.com/q/2378402/, 2010.

[22] StackOverflow. *Jackson Vs. GSON*. http://stackoverflow.com/q/2378402/, 2013.

[23] StackOverflow. *Post-18601384*. http://stackoverflow.com/q/18599591/, 2013.

[24] StackOverflow. *XStream or Simple*. http://stackoverflow.com/q/1558087/, 2017.

[25] S. Subramanian, L. Inozemtseva, and R. Holmes. Live api documentation. In *Proc. 36th International Conference on Software Engineering*, page 10, 2014.

[26] J. Svajlenko and C. K. Roy. Evaluating modern clone detection tools. In *IEEE International Conference on Software Maintenance and Evolution*, pages 321–330, 2014.

[27] J. W. Tukey. *Exploratory Data Analysis*. Pearson, 1st edition, 1977.

[28] G. Uddin and M. P. Robillard. How api documentation fails. *IEEE Softawre*, 32(4):76–83, 2015.

[29] J. Walnes. *Xstream*. http://x-stream.github.io/, 2017.

[30] Wikipedia. *Application programming interface*. http://en.wikipedia.org/wiki/Application_programming_interface, 2014.

[31] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.