How Important are Usage Examples in REST API Documentation?

S M Sohan, Frank Maurer*
* Dept. of Computer Science
University of Calgary
Canada
{smsohan, frank.maurer}@ucalgary.ca

Craig Anslow
School of Eng. and Computer Science
Victoria University of Wellington
New Zealand
craig@ecs.vuw.ac.nz

Martin Robillard
School of Computer Science
McGill University
Canada
martin@cs.mcgill.ca

***Abstract*—[Brain-dump mode]**

What is the the problem? Effort and cost associated to REST API documentation with usage examples. Why is this a problem? Cost vs. benefit of usage examples is not known. What to include in the examples? What have we done? Performed an empirical study with 26 professional software engineers with REST API experience to understand how their productivity and types of problems faced changes with respect to usage examples. What have we learned? (Hoping) examples help developer perform task with fewer problems with greater success, with fewer errors, and faster in terms of time. What does it mean? (expected) API developers should include usage examples in REST API documentation that include specific examples based on the problems faced.

I. Introduction

II. Methodology

A. Research Goals

This research is aimed at answering the following research questions:

- **RQ**. What obstacles API client developers face when using a REST API documentation that lacks usage examples?

B. Requirements

To answer the aforementioned research questions, the study has the following requirements:

- **R1. Representative API.** We had to choose an existing REST API that is currently used by API client developers. Selecting a mature REST API for this study reduces the possibility of an underdeveloped solution that may be found if a new or a seldom used API is selected. A familiar domain needed to be selected so that participants are able to relate to the API features without requiring upfront training. In addition to selecting the API, we had to select tasks that are related to the core features provided by the API to represent reality.
- **R2. Open source.** To be able to understand the impact of usage examples, we needed to select an open-source API where we can add new examples to the documentation for performing this study. For proprietary APIs, it won't be possible to retrofit usage examples on existing API documentation without separating the examples from the documentation.
- **R3. Time bound.** The total time spent by each participant is required to be time bound to measure productivity. As a result, the study needs to be setup such that participants are able to focus on performing the tasks minimizing any overhead.
- **R4. Experienced developers.** Developers with prior experience on REST APIs need to be recruited as study participants to perform the study within a limited amount of time and in a realistic setup. Furthermore, to reduce a learning bias, only participants with no prior experince of using the WordPress REST API V2 are accepted for this study.

C. Study API

We selected the WordPress REST API V2 for this study. WordPress is a blog-like open-source (R2) framework used by over 409 million people to visit 23.6 billion pages each month. The API allows programmatic access to list, create, update, and delete WordPress data such as blog posts, comments, users, images, tags, etc.

The WordPress REST API V2 has been published and maintained since May 2015. Before January 2017, the WordPress REST API V2 was distributed as a plug-in where WordPress users could optionally install the API component. The following statistics are for the plug-in install numbers between May 2015 and October 2016:

- Total installs: aprox. 248K installs of the plug-in.
- Average daily installs: approx. 500.

Starting January 2017, the WordPress REST API V2 is no longer required to be installed as a separate plug-in since it's pre-bundled with every WordPress install. As per the code repository on GitHub, there are a total of 99 and 46 contributors that had at least one commit to the code repository behind the API and and it's documentation, respectively. These properties satisfy R1, our requirements for using a representative API.

By selecting an open-source project we are able to access the source-code to inspect the implementation and documentation technique of the WordPress REST API. The API implements a self-documenting feature where API developers expose API endpoints over HTTP OPTIONS verb to explain

the API elements. To implement this feature, the API developers describe the API elements in the code. For example:

Listing 1: Example of self-documenting API Code

```
1  public function get_item_schema() {
2    $schema = array(
3      '$schema'    => 'http://json-schema.org/draft
           -04/schema#',
4      'title'      => $this->post_type,
5      'type'       => 'object',
6      /*
7       * Base properties for every Post.
8       */
9      'properties' => array(
10         'date'           => array(
11             'description' => __( "The date the
                   object was published, in the
                   site's timezone." ),
12             'type'        => 'string',
13             'format'      => 'date-time',
14             'context'     => array( 'view', 'edit
                   ', 'embed' ),
15         ),...
```

On Listing 1, line 4 specifies that this is a schema definition for the API element $Post$. Then, on Line 10, it defines $date$, one of the properties of $Post$, followed by a human readable description and and type information. Then, on line 14, the context of this property is mentioned as $view, edit, embed$, meaning that this property will be returned when the $Post$ object is returned, embedded, or can be used as an input for editing.

This self-documenting feature is leveraged to generate and publish the official API documentation as an HTML website. Figure 1 shows a screenshot of the published documentation for the $date$ attribute of the $Post$ API element[1].

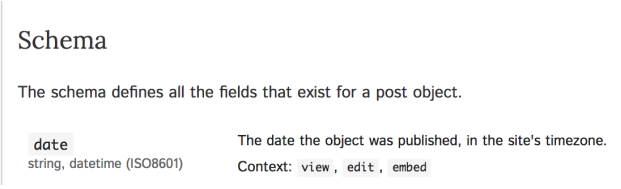


Fig. 1: Screenshot showing auto-generated documentation for Post/date

In addition to the auto-generated documentation of the the schema and API actions, custom content is added by the contributors to the API documentation project to provide prosaic overviews and usage examples.

D. Study Design

1) Tasks: The participants are requested to perform a total of six tasks using the API including one practice task. All of the tasks are related to a single API element, $Post$. The tasks get progressively difficult, and all but the last task can be performed independently of each other. Participants are requested to limit the total time on the study to a maximum of one hour. Participants are encouraged to proceed to the next task when they are either satisfied about or stuck on the right answer.

To use a REST API, the API client developers need to work with the following four inputs over HTTP:

- I1. Request method
- I2. Request URL
- I3. Request headers
- I4. Request body

To verify the response of an API call, the API client developers can use HTTP response headers and/or HTTP response body. To perform the tasks, the participants are required to use one or more of the inputs I1-4. In the following paragraphs, we describe each task with it's description and the intended test scenario against the aforementioned API input and output information. For each of the tasks, the participants are required to use the same WordPress REST API and its documentation.

T1: ListAllPostsTask. We ask the participants to use the WordPress REST API following its documentation to get a list of the blog posts from a live WordPress site. This is the practice task, and the inputs to answer this task are pre-filled for the participants. It allows the participants to understand the tools used for this study as well as get familiarity with the Post API. The answer for this task makes use of I1 and I2.

T2: FilterPostsByAuthorTask. The participants are asked to use the API to filter the list of posts by an author given the author's user name. To answer this correctly, the participants are required to first make an API call to get the numeric ID of the author given the string based user name. Then, the ID needs to be used on the Post API to filter posts by the author. This task allows us to understand the impact of usage examples on API client developers when multiple API calls need to be made to perform a task using the API. Inputs I1-2 are required to complete this task successfully.

T3: ExcludePostsByIdsTask. We ask the participants to use the API to get a list of all posts excluding posts with IDs 1 and 4. Participants need to use the inputs I1-2, and use a desired format on I2 to pass an array of IDs as a parameter. This task allows us to understand how API client developers identify the format for using an array within the URL with respect to the usage examples in the API documentation.

T4: FindTotalPostsTasks. This task requires the participants to use the API to find a total number of posts. Participants need to use the inputs I1-2 and inspect the HTTP response headers to successfully complete this task. This task allows us to understand how API client developer productivity is affected with respect to missing examples about HTTP response headers in the API documentation.

T5: PublishPostTask. We ask the participants to use the API to publish a blog post with a specific title, content, and a published date. To successfully complete this task, the participants are required to use all four input types and inspect both the HTTP response header and the response body. Additionally, the participants are required to use a specific date format that the API accepts as a valid format for date specification. Answers to this task allows us to study API client

[1]http://v2.wp-api.org/reference/posts/

developer productivity with respect to the usage examples lacking details about the inputs I3-4.

T6: UpdatePostTask. We ask the participants to use the API to update a blog post that they published in T5 with a new excerpt. Similar to T5, this task requires the use of inputs I1-4, but with different values for the inputs. This task allows us to understand API client developer productivity on inter-dependent tasks with respect to usage examples.

To summarize, the tasks allow us to understand how REST API client developers approach API tasks of different complexity levels involving various input types and available output information with respect to the usage examples in the API documentation.

2) Participant Selection: To satisfy the requirement of developers with REST API experience (R3), we have used the following criteria for recruiting the study participants:

- Currently working as a software engineer.
- At least 1 year of industry experience as a software engineer.
- At least 1 year of industry experience with REST APIs.
- No prior experience with WordPress REST API.

A total 26 participants from sixteen different companies and six different countries (Canada, USA, Germany, Ireland, Brazil, and, Bangladesh) were recruited through online announcements posted on Twitter, Facebook, and software developer focused mailing lists. Table I shows a summary of the experience level of the participants on each group.

TABLE I: Participant Profile

	Group 1	Group 2
Number of Participants	16 (P1.1-P1.16)	10 (P2.1-P2.10)
Industry Experience		
1-5 years	5	1
5-10 years	6	5
10+ years	5	4
Average	9.1	10.6
REST API Experience		
1-3 years	5	3
3-5 years	7	4
5+ years	4	3
Average	4.5	4
Number of Companies	10	8

3) Process: Two pilot studies were performed to evaluate and understand an effective process for performing this study. The first pilot study involved four participants that were invited to join the first author on this paper in-person or using a video conferencing software. The study involved tasks using two APIs, the WordPress REST API V2 and the GMail REST API. Each participant was given one of the two APIs and a set of tasks to complete using the API within an hour. Participants were given an online answer form to record the answers to the tasks. The primary findings from this pilot are as follows: 1) asking participants to use an API to perform the tasks required significant overhead time for them to setup a development environment with the proper API credentials, 2) the intermediate trial attempts of using the API are potentially more valuable than the final answer as it allows us to understand API client developer information needs that may are not answered by the documentation, 3) the number of tasks for the study had to be reduced so the participants could complete the tasks within the one hour limit, and 4) for GMail API, participants used up a large portion of their time on setting up their API credentials that requires understanding of OAuth.

To overcome the shortcomings found from the first pilot study, we decided to develop a web-based REST API explorer as shown on Figure 2 that allows participants to use their browser to make the API calls without setting up any development environment. The web-based API explorer only requires the inputs I1-4, and displays the HTTP response headers and body on the click of a button. Thus the participants could focus on using the right input and verifying the output without having to write any code. The web-base REST API explorer also allowed us to automatically collect all the trial API calls that the participants make for each API task. A second pilot study involving seven new participants was performed to understand the features of the web-based REST API explorer in practice and to improve the user interface based on feedback from the participants. Participants completed the study on their own without having to meet in-person or over video conferencing. Only the WordPress REST API was used to focus on REST APIs without the required learning curve associated with OAuth. We found encouraging results from this pilot study as the collected data showed patterns of mistakes that API client developers make that can be reduced by adding usage examples in the API documentation.

Based on the lessons learned from the pilot studies, we designed the actual study process as follows: the original WordPress REST API documentation was forked and a total of 3 API usage examples were added to show listing of blog posts with query parameters for filtering, a request to create a blog post and a request to update a blog post. Figure 3 shows a screenshot of the original API documentation related to T4 where the API client developers are provided with a reference table describing the different properties that can be used to create a $Post$ object[2]. Figure 4 shows a screenshot of the forked API documentation with a cURL based usage example. cURL is used because it's used by elsewhere in the original API documentation. In the forked API documentation, the example shows one possible API call with realistic values for the data that is described in the reference table and associated API response headers and body.

Participants were divided into two groups, G1 and G2. G1 participants were provided with a link to the official API documentation on the web-based API explorer, and G2 participants were provided with a link to the forked API documentation with usage examples. The web-based API explorer allocated more participants to G1 compared to G2 because we wanted to better understand the impact of the lack of usage examples on API client developer productivity. However, each individual participant was randomly assigned to a group by the web-

[2]http://v2.wp-api.org/reference/posts/

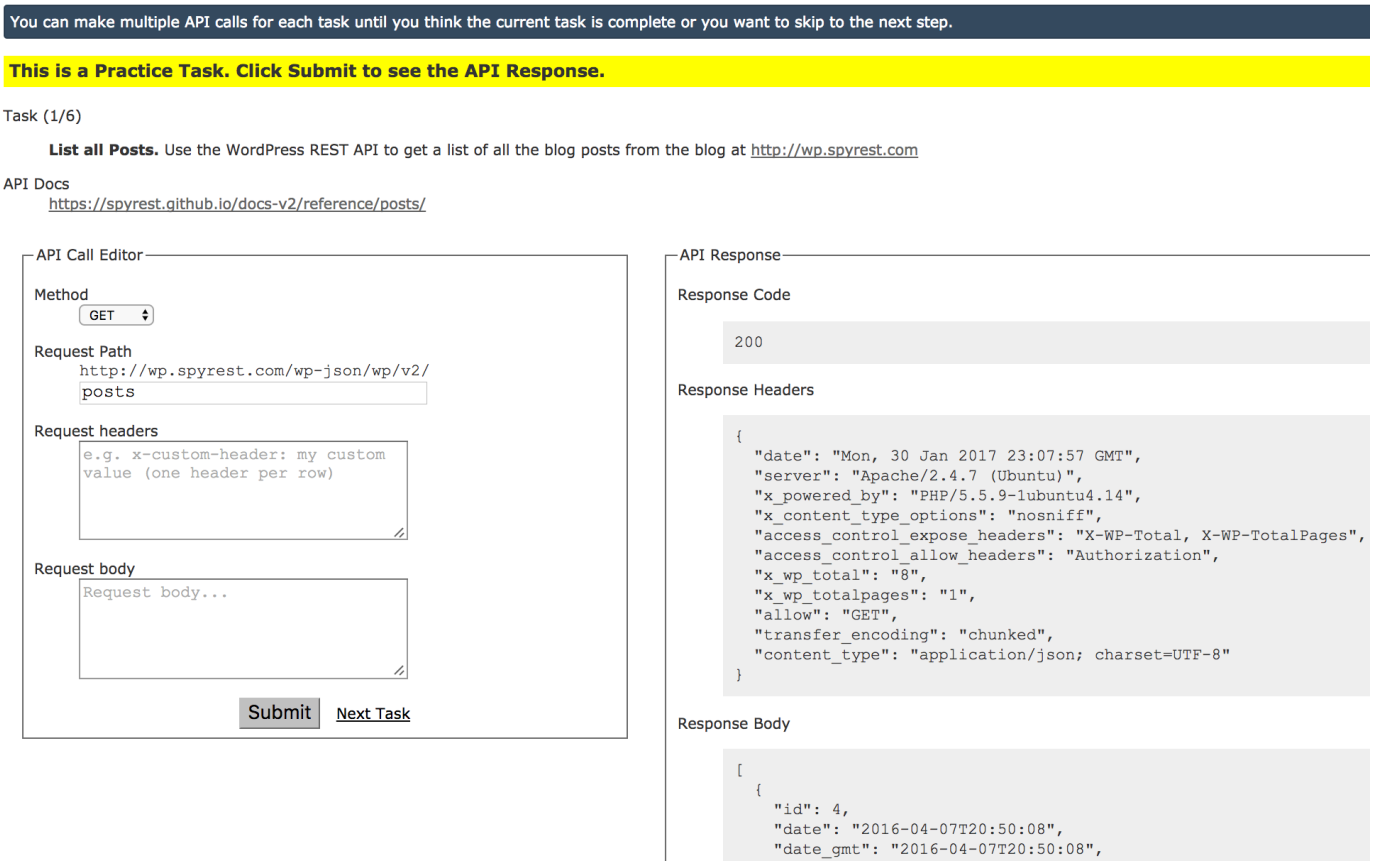


Fig. 2: Screenshot of the web-based REST API explorer

based API explorer. All participants were given the same set of API tasks and were requested to limit their participation time to a maximum of one hour. No task specific time limit was imposed because we wanted participants to spend sufficient time on each task without forcing them to move the next one. Participants were allowed to access the internet and external resources alongside the provided API documentation to perform the tasks as they'd normally use on a typical work day.

4) Data Collection: The data collected by the web-based REST API explorer for each participant is exported into a text file as the raw data artifact. For each API task and each participant that attempted the task, the exported text file contains the request inputs I1-4 and associated API response headers and body for each API call made by the participant. For each participant, we recorded all trial API calls with timestamps. Additionally, the demographic information, experience rating and the free-form feedback for each participant is also included in the artifact.

5) Data Analysis: The same data artifact is used to analyze both quantitatively and qualitatively to answer the aforementioned research question.We analyzed the data artifact qualitatively to understand the obstacles that REST API client developers face when using an API documentation that lacks usage examples. The data artifact exported from the web-based REST API explorer included a table with raw data. For each row, the table contains the following columns: API task, participant identifier, timestamp, API trial number, I1-4, response headers and body. It allowed us to observe and categorize the different input values that the participants used to perform the given API tasks. The values for I1-4 used by the participants were manually coded to group the participant responses into categories. The categories help us determine the types of information that API client developers need in the API documentation to perform API tasks successfully. The analysis started with an empty set of codes and new codes were introduced to describe scenarios that didn't fall under codes that were already applied. The first author on this paper applied the codes on the raw data artifact and the co-authors verified the codes randomly. To resolve disagreements, the authors discussed and updated the codes accordingly. The codes were then further categorized to higher level grouping to represent common problems faced by the study participants for each group.

For each API task attempted by each participant, we annotated the artifact with one of the following labels: successful, partially successful, and unsuccessful. Task participations are marked successful when I1-4 matches the required values for performing the given task. If a participant is able to use the correct I1-4 for one of the two tasks required to

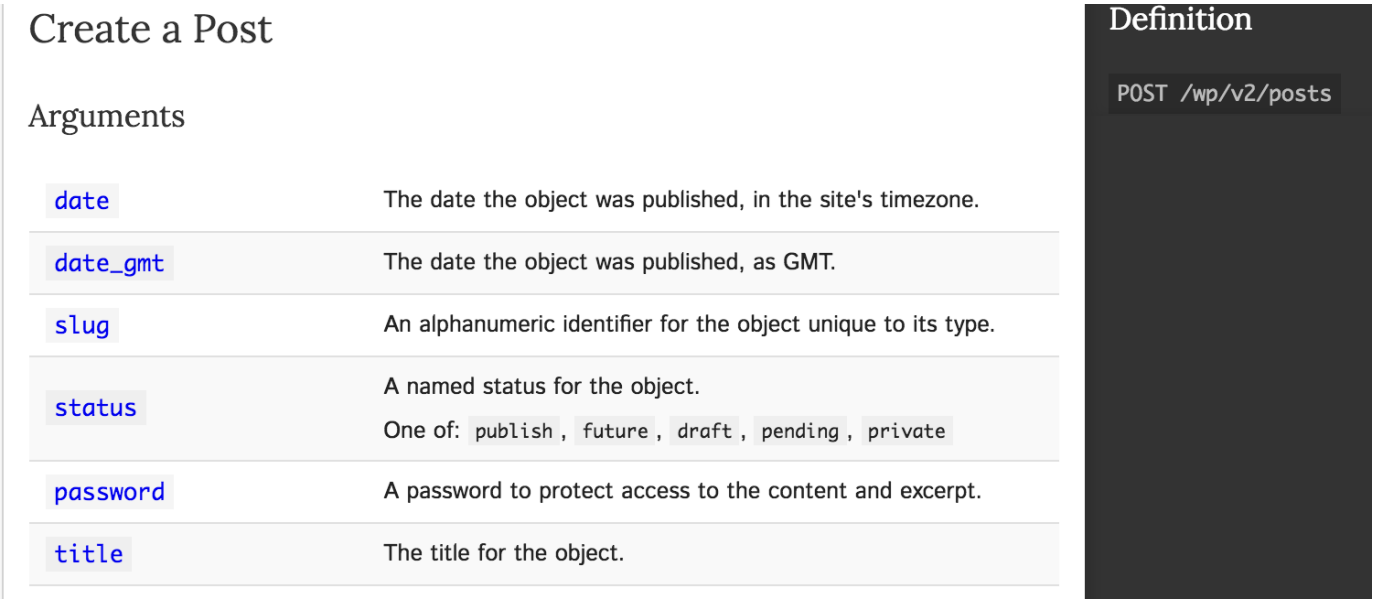


Fig. 3: Screenshot of the original WordPress REST API documentation

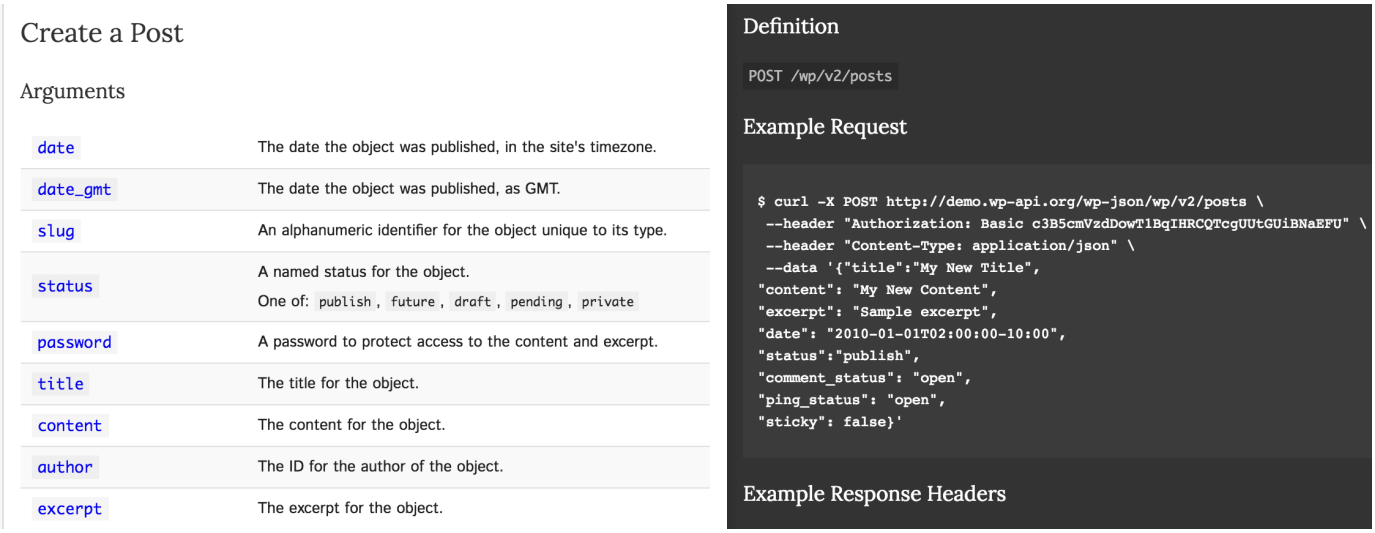


Fig. 4: Screenshot of the forked WordPress REST API documentation with a usage example

complete a single task (T2), we marked it as a partially successful. Otherwise, it's marked as unsuccessful. Based on these annotations, the answers for RQ1 was computed using the following:

success rate of group = (total number of successful tasks by group) / (number of participants in group * number of API tasks)

API calls per success of group = (number of API calls annotated as successful) / (total number of trial API calls made by group)

average time taken for success = (number of successful API tasks by group) / (total time spent by all participants in the group)

III. Results

A. Qualitative Analysis Results

Observation 1. REST API client developers want usage examples along with the definition of API elements. From analyzing the participant feedback, we've observed usage examples mentioned as the most frequent and central topic of interest. 11 of the 16 G2 participants mentioned the lack of examples with HTTP headers, request and response as the primary problem with the API documentation. On the other hand, 7 out of the 10 G2 participants mentioned the examples being helpful from the forked documentation. This is also supported by our quantitative analysis as discussed later in this section.

Implication 1. REST API developers need to provide usage examples showing sample requests and responses with HTTP headers and body for API actions to help API usability.

Observation 2. API client developers face problem with using right data type without examples. To successfully complete T1 participants had to use an integer ID representing an $Author$ object given the author name. In the original API documentation, the following is mentioned about the author parameter: "author: Limit result set to posts assigned to specific authors." The documentation doesn't mention that the data type required is the numeric author ID, not the author name. We found 10 of the 16 participants from G1 made API calls with the author name instead of the ID. P1.12 mentioned the following: **"...to find posts for the author, I had to inspect the response to see that indeed the $author\_id$ was in there"**. The forked documentation showed one example of using numeric ID. 3 of the 10 participants from G2 ran into the same error. When faced with this error, participants used trial and error to eventually complete the task.

Implication 2. REST API documentation needs to include examples of data types for each API field to satisfy API client developer information need.

Observation 3. API client developers face problem with using the right data format without examples. Tasks T3 required the use of a correct format to represent an array of numeric post IDs to be excluded using the API. The original documentation mentions the following about this API query parameter: "exclude: Ensure result set excludes specific IDs." Participants attempted to solve this task using multiple different formats for specifying the array. For example: they tried with $exclude = [1, 4]$, $exclude = 1\&4$, $exclude = 4\&\&exclude = 1$, $id! = 1\&id! = 4$ and other alternatives before eventually finding the right format as follows $exclude = 1, 4$. Participant P1.6 mentioned the following: **"It was difficult to figure out whether some of the inputs needed to be arrays or just a comma-separated list."** 7 out of the 16 participants in G1 faced this error. The forked documentation showed one example of using multiple IDs and all G2 participants could use the right format.

Task T5 also required the participants to use an ISO8601 formatted date for publishing a post with a specific date. Even though the documentation mentions ISO8601 format, it doesn't provide an example. So, participants had to use online search. P1.8 mentioned the following: **"Got hung up trying to figure out the formatting of the date (which doesn't appear to be ISO8601, despite what it says)"**. After analyzing the response, we found that P1.8 used the ISO8601 formatted date in the API calls, but didn't provide the $second$ portion of the time as required by the API. 8 of the 16 participants in G1 had an error in the date formats. With an example, 1 of the 10 participants had the same error from G2.

Implication 3. REST API developers need to include examples showing the valid data format for the API elements.

Observation 4. REST API client developers face problem with using request headers without usage examples. Tasks T5 and T6 required the use of a HTTP request header named $Content - Type$. In the original API documentation, the following was mentioned: "The API uses JSON exclusively as the request and response format, including error responses.". Experienced software engineers still had problem using this HTTP header as we found from the participants in G1. P1.12 mentioned the following: **"I didn't know how to specify application/json so that I could send POST data via the body instead in the URL parameters... Examples would really help - with a sample request and a sample response."** We were surprised to see that 14 of the 15 participants that attempted T5 and T6 from G1 had the error of not specifying the $Content - Type$ header even though they have years of experience with REST APIs. In contrast, 2 out of the 10 participants from G2 failed used this header correctly as indicated in the example in the forked documentation. Moreover, 1 of the 2 G2 participants that didn't use the header first time, later used it correctly on a subsequent attempt.

Implication 4. If API requests need to use HTTP Request headers, in addition to the request method and body, REST API develpers need to include examples of the HTTP headers.

Observation 5. API client developers face problem with performing API tasks that require multiple API calls. Tasks T2 and T6 required the use of multiple API calls to complete successfully. Both tasks require an initial API call to find data using the API that is needed make a second API call. For example, P1.5 mentioned the following about T6, **"The post I created is not visible and not editable, though I got 201. Not sure why"**. After analyzing the response, we observed that P1.5 was able to create a blog post, but failed to publish it because it didn't have the $publish$ status. As a result, the post was not returned via the API for T6. P2.2 mentioned the following about T2: **"In case of search posts by author, it's not clear how to find the author_id. Although, I realized that users and authors are same in this case."**

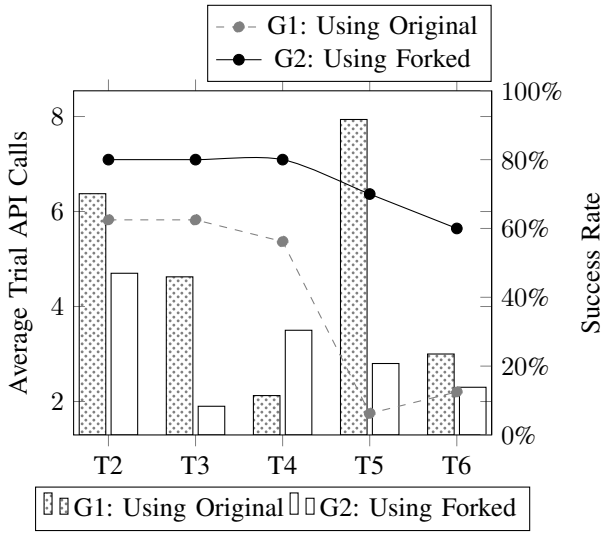
Implication 5. If there are prerequisites for making an API call, REST API developers need to provide examples showing how to get the prerequisites in the API documentation.

To summarize, we recommend API developers to consider usage examples as an essential requirement for REST API documentation. As such, sufficient resources and a high priority should be devoted to generate and maintain the usage examples. Based on our observations, we recommend REST API documentation tools to provide first-class support for including usage examples with realistic data.

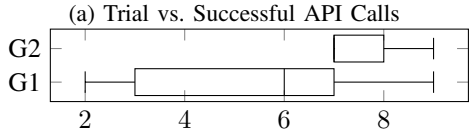
B. Quantitative Analysis Results

We present a summary of the quantitative results as found by analyzing a total of 539 API calls (385 from G1, 152 from G2) made by the participants from this study in Figure 5. Figure 5a juxtaposes the average number of trial API calls made by each group against their rate of success in performing the given tasks T2-T6. The average number of trial API calls and success rate for each group is computed using the following formulas:

AverageTrialAPICall(task, group) = Total number of API calls made by group on task / Number of participants in group



(a) Trial vs. Successful API Calls



(b) Participation Satisfaction Ratings

Fig. 5: Comparison Between G1 and G2

SuccessRate(task, group) = Number of participants that correctly completed task / Number of participants in group

In Figure 5a we see that, for the simpler tasks T2 and T3, G1 had used more trial API calls on average and reached successful answers with a lower percentage compared to G2. For T4, where the participants are required to inspect a HTTP response header, G2 needed more API calls on average, but also yielded 24% higher success rate. For the more complicated tasks T5 and T6, we see that participants in G1 had very low success rate and very high number of average API trials compared to G2. The data here clearly depicts that without usage examples REST API client developers require more trial and error attempts yet resulting in lower success rates, and the situation gets worse for relatively complicated tasks.

In Figure 5b, we see that the G2 participants provided higher satisfaction rating compared to G1 participants. P1.1 provided a satisfaction rating of 9, and analyzing the participant feedback against the success, we see a false positive case, where the participant perceived that the tasks were successfully completed even though only 2 of the 5 tasks were correctly answered. Other than this case, the satisfaction ratings provided by G1 participants consistently fall behind the G2 participants' ratings.

Based on these results, we recommend REST API developers to study the usability of their API documentation with prospective API client developers and provide required usage examples in the documentation to improve API client developer productivity and satisfaction.

IV. Threats to Validity

API Selection Task Selection Participant Selection Experiment Design

V. Related Work

1) API Learning Obstacles: Researchers have published several papers on the topic of API learning obstacles. Robillard et al. used a mixed approach of surveys and in-person interviews with professional software engineers to understand the API learning obstacles [1] [2]. They found that API documentation was related to one of the most severe obstacles for API learnabilty and recommended considering five factors for API documentation as follows: documentation of intent, code examples, matching APIs with scenarios, penetrability of the API, and format and presentation. Gias et al. surveyed 323 professional software engineers to understand the factors that fail a API documentation [3]. They identified ambiguity, incompleteness, and incorrectness as the three severest problems that lead API documentations to fail to answer the information needs of developers. Duala-Ekoko et al. performed a study with twenty programmers to understand the types of questions that developers have while using an unfamiliar API [4]. They categorized the API related information needs into twenty generic questions that can be used to analyze the quality of API documentation. Myers et al. performed a study of enterprise SOA API usability and found that the study participants had limited success finding the relevant API elements to perform an API task [5]. Among other factors, they recommended providing code examples in the API documentation to help API client developers with sample solutions to different patterns of API tasks.

In this paper, we focused on the usage examples part of API documentation to further understand the obstacles that API client developers face when the documentation contains relevant information about the API, it's intent, description of the API actions and elements, but lacks code examples.

2) Crowd-Sourced API Usage Examples: Several existing research has focused on sourcing API usage examples from the Internet. Wang et al. performed an exploratory study to understand the current state of API related knowledge available on the Internet that can be leveraged by API client developers [6]. They searched the web for usage examples related to the API of five popular Java libraries and found that on an average API examples could be found for 77% of the 4,637 APIs included in the selected libraries. Moreover, they found that the crowd-sourced API documented accounted for 93.7% of the usage examples compared to 6.13% that were published on the official documentation. Incorporating a corpus of crowd-sourced API examples from the web and using placeholders for commonly related API elements, an evaluation of Jadite found that developers were three times faster to complete API tasks [7]. Nasehi et al. analyzed StackOverflow threads to understand the characteristics of good and bad code examples based on user provided votes [8]. They found that commonly down-voted API related answers lacked code examples, explanations, and shortcoming of solutions. They

recommended API developers to evolve the documentation with usage examples to answer API client developer questions that were not anticipated in the existing documentation. Treude et al. proposed a machine learning based approach to find relevant API documentation from StackOverflow using both the text and metadata found from the questions and answers [9]. Kim et al. presented a technique to automatically augment code examples from the web using code search tools to Java APIs [10]. Jiau et al. observed a severe inequality within the context of crowd-sourced API usage examples where most of the content were related to the popular APIs [11].

Existing research on crowd-sourced API documentation has mostly focused finding crowd-sourced usage examples of local APIs. In this paper, we focused on the impact of usage examples for REST APIs, where the API client development programming language is agnostic to the language used to implement the API. This important distinction with local APIs makes it hard to leverage the techniques proposed in the aforementioned papers to find relevant REST API usage examples.

3) Measuring API Usability: Several papers have been published on the topic of measuring API usability. Rama et al. presented a set of formulas for computing a measure of API usability based on the API's structural components such as classes, methods, parameters, return values, thread-safety, etc [12]. Scheller et al. presented a framework for automatically measuring the complexity of an API [13]. They identified a list of measurable API properties and provided formulas to compute complexities related to the interfaces, implementation and setup of an API. Grill et al. proposed an HCI based approach that can be used by API developers to understand and improve on the problem areas related to an API [14]. They suggested using a combination of expert opinion and developer workshops on APIs to identify and collect feedback about API and its documentation related problems. To improve API usability, Farooq et al. proposed using peer reviews of API code in addition to API usability studies to uncover API related bugs and incorporate feedback regarding the API elements [15]. In our work, instead of measuring overall API usability, we have focused specifically on the relationship of REST API usability with respect to usage examples.

4) Controlled Studies: Several authors published the results of controlled studies on the topic of API usability. Nasehi et al. performed a controlled study to understand if API unit tests can be used to provide as usage examples to facilitate API client developers [16]. They grouped participants into two groups, and one of the groups was provided with API unit tests in addition to the documentation. The two groups performed the same set of API tasks. The researchers found that the examples from the unit tests helped understanding the API concepts better but it was challenging for the participants to locate relevant examples from the corpus of unit tests. They recommended automated extraction of relevant high-level API usage examples from API unit tests. Endrikat performed a controlled study with four groups of participants to understand the impact of API documentation on APIs that are implemented using programming languages with static and dynamic type systems [17]. Participants were given a set of failing unit tests and were asked to make them pass by writing code using the studied API. They defined the API client developer productivity in terms of the time to get the tests passing. They found that the participant group using a statically typed API with explicit documentation were more productive than the other groups. Ellis et al. performed a controlled study with two groups of Java developers to understand the impact of using a constructor vs. a factory method design pattern on API client development time [18]. They found that API client developers needed more time to complete API tasks when the API requires the use a factory method to instantiate objects compared to using a constructor.

Our work in this paper is based on a controlled study and shares part of the setup that were used by the aforementioned API related controlled studies with the following differences: our goal is to understand the obstacles faced by API client developers without usage examples; we focus on REST APIs instead of a local API; and, the participants are professional software engineers.

VI. Conclusion

Acknowledgment

References

[1] M. Robillard, "What makes APIs hard to learn? the answers of developers," *Software, IEEE*, vol. PP, no. 99, pp. 1–1, 2011.

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

[3] G. Uddin and M. P. Robillard, "How api documentation fails," *IEEE Software*, vol. 32, no. 4, pp. 68–75, July 2015.

[4] E. Duala-Ekoko and M. P. Robillard, "Asking and answering questions about unfamiliar apis: An exploratory study," in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 266–276. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337223.2337255

[5] B. Myers, S. Jeong, Y. Xie, J. Beaton, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and D. K. Busse, "Studying the documentation of an api for enterprise service-oriented architecture," *J. Organ. End User Comput.*, vol. 22, no. 1, pp. 23–51, 2010.

[6] L. Wang, Y. Zou, L. Fang, B. Xie, and F. Yang, "An exploratory study of api usage examples on the web," in *2012 19th Asia-Pacific Software Engineering Conference*, vol. 1, Dec 2012, pp. 396–405.

[7] J. Stylos, A. Faulring, Z. Yang, and B. A. Myers, "Improving api documentation using api usage information," in *2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Sept 2009, pp. 119–126.

[8] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns, "What makes a good code example?: A study of programming Q&A in StackOverflow," in *Proc of. IEEE International Conference on Software Maintenance*, 2012, pp. 25–34.

[9] C. Treude and M. Robillard, "Augmenting api documentation with insights from stack overflow," in *Proc. of ACM International Conference on Software Engineering*, ser. ICSE '16, New York, NY, USA, 2016, pp. 392–403.

[10] J. Kim, S. Lee, S.-w. Hwang, and S. Kim, "Adding examples into java documents," in *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 540–544. [Online]. Available: http://dx.doi.org/10.1109/ASE.2009.39

[11] H. Jiau and F.-P. Yang, "Facing up to the inequality of crowdsourced api documentation," *ACM SIGSOFT Software Engineering Notes*, vol. 37, no. 1, pp. 1–9, 2012.

[12] G. M. Rama and A. Kak, "Some structural measures of api usability," *Software: Practice and Experience*, vol. 45, no. 1, pp. 75–110, 2015.

[13] T. Scheller and E. Kühn, "Automated measurement of api usability: The API concepts framework," *Information and Software Technology*, vol. 61, pp. 145–162, 2015.

[14] T. Grill, O. Polacek, and M. Tscheligi, *Methods towards API Usability: A Structural Analysis of Usability Problem Categories*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 164–180. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-34347-6_10

[15] U. Farooq, L. Welicki, and D. Zirkler, "Api usability peer reviews: A method for evaluating the usability of application programming interfaces," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '10. New York, NY, USA: ACM, 2010, pp. 2327–2336. [Online]. Available: http://doi.acm.org/10.1145/1753326.1753677

[16] S. M. Nasehi and F. Maurer, "Unit tests as api usage examples," in *2010 IEEE International Conference on Software Maintenance*, Sept 2010, pp. 1–10.

[17] S. Endrikat, S. Hanenberg, R. Robbes, and A. Stefik, "How do api documentation and static typing affect api usability?" in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 632–642. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568299

[18] B. Ellis, J. Stylos, and B. Myers, "The factory pattern in api design: A usability evaluation," in *Proceedings of the 29th International Conference on Software Engineering*, ser. ICSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 302–312. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2007.85