A Study of the Effectiveness of Usage Examples in REST API Documentation

Placeholder Author Name
Placeholder Department
Placeholder Affiliation
Placeholder Address
Placeholder Email

***Abstract*—Generating and maintaining REST API documentation with usage examples can be a time consuming and expensive process for evolving APIs. Most REST API documentation tools focus on automating the documentation of the API objects, but require manual effort for capturing usage examples. Consequently, REST API developers need to know the cost vs. benefit of providing usage examples in the documentation to prioritize the documentation efforts. To this end, we have performed a controlled study with 26 experienced software engineers to understand problems that REST API client developers face while using an API without usage examples. We found that REST API client developers face productivity problems with using correct data types, data formats, required HTTP headers and request body when documentation lacks usage examples. By following the REST API documentation suggestions from this paper, REST API developers can reduce the errors, improve success rate and satisfaction of API client developers.**

***Index Terms*—API; REST; Documentation; Usage Examples; Empirical Study; Controlled Study; Productivity;**

I. Introduction

REST APIs are used as the predominant application integration mechanism over the Internet. Generating and maintaining REST API documentation with usage examples can be an expensive process because most API documentation tools do not support effective usage examples. Researchers have emphasized API documentation as a key factor that impacts API usability both positively and negatively. To improve API documentation, researchers have recommended incorporating usage examples in the API documentation.

In a resource constrained environment, it is important to understand the value of usage examples on REST API usability to allocate sufficient attention and efforts towards incorporating examples in the API documentation. While it is expected that examples help developers, REST API developers need to know what to include in the examples and why.

The documentation of REST APIs has distinctive features compared to the documentation of local APIs such as Java libraries. For example, local API documentation commonly comprises the description of classes and interfaces with their methods. In contrast, REST API documentation needs to include information about HTTP headers, request and response body and the data representation formats such as JSON, XML. The existing research on API usability areas have primarily focused on local APIs. In our work we have focused on understanding the impact of usage examples within the realm of the distinctive REST API features.

We designed and performed a controlled study with experienced software engineers to understand how REST API client developers are affected while using an API documentation that describes the API elements but lacks usage examples. Participants were divided into two groups and given the same set of API tasks to complete. One group was given the official WordPress REST API documentation and another group was given an enhanced version of the documentation where three usage examples were added. Using a novel technique, we collected 539 API calls made by the participants. We analyzed the data and observed recurring obstacles faced by the participants while performing the API tasks using the official documentation that lacks usage examples. Our contributions on this paper are as follows:

- We provide a list of obstacles that REST API client developers face while performing API tasks using documentation that lacks usage examples.
- We provide a list of recommendations for REST developers to be used as a guideline to incorporate usage examples in API documentation.
- We provide empirical evidence that usage examples in REST API documentation help API client developers perform API tasks with higher developer satisfaction, less time, and better success rate.

The remainder of this paper is organized as follows: In Section II we present our research question. In Section III we discuss our methodology in terms of the study requirements, the selected API of the study and the participant selection, the study setup, data collection and analysis methods. In Section IV we discuss the results of our analysis. We discuss the threats to validity and provide a summary of the related work in Sections V and VI respectively.

II. Research Question

This research is aimed at answering the following question:

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

III. Methodology

To answer the aforementioned research question, 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 that the obstacles we observe in the study are the results of insignificant accidental problems symptomatic of an immature technology. We selected a familiar domain 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 a common usage area of the API.
- **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.
- **R3. Time bound.** We applied a maximum time constraint for each participant to measure the rate of success within a limited time frame. As a result, the study needs to be setup such that participants are able to focus on performing the tasks minimizing any overhead.
- **R4. Participant Selection.** 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.

A. 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 [1]. The API allows programmatic access to list, create, update, and delete WordPress data such as blog posts, comments, users, images, tags.

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 installation numbers between May 2015 and October 2016:[2]

- Total installations: $\approx 248K$ installs of the plug-in.
- Average daily installations: ≈ 500.

Starting January 2017, the WordPress REST API V2 is no longer required to be installed as a separate plug-in since it is bundled with WordPress installation. According to the code repository, there are a total of 99 and 46 contributors that had at least one commit to the code repository of the API and its documentation, respectively. These properties satisfy R1 AND R2, our requirements for using a representative API.

[1]https://wordpress.com/activity/

[2]https://wordpress.org/plugins/rest-api/stats/

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 in PHP

```
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` [3]. On Line 10, it defines `date`, one of the properties of `Post`, followed by a human readable description and type information. 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.

The WordPress team leverages this self-documenting feature to generate and publish the official API documentation as an HTML website. Fig. 1 shows a screenshot of the published documentation for the `date` attribute of the `Post` API element.[4]

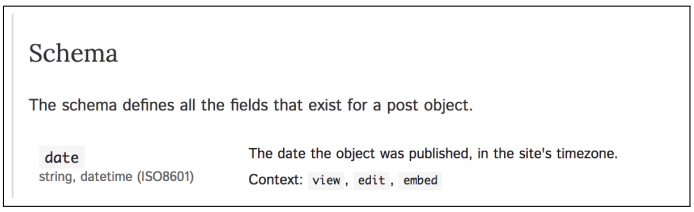


Fig. 1: Screenshot of documentation for Post/date

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

B. 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`[5]. The

[3]https://github.com/WP-API/WP-API

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

[5]A Post identifies a blog post within WordPress

tasks get progressively more 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 (R3). Participants are encouraged to proceed to the next task when they are either satisfied with their answer or feel stuck and unable to make progress.

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 its description and the study observation goals against the aforementioned API input and output information. For each of the tasks, the participants are required to the same WordPress REST API and one of the two variants of accompanying documentation.

T1: List all posts task. We asked the participants to use the WordPress REST API to get a list of the blog posts from a given 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: Filter posts by author task. The participants are asked to use the API to filter the list of posts obtained in T1 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: Exclude posts by IDs task. 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: Find total posts task. 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: Publish post task. 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 developer productivity with respect to the usage examples lacking details about the inputs I3-4.

T6: Update post task. 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 (R4), we used the following criteria for recruiting the study participants:

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

We recruited a total of 26 participants from sixteen different companies and six different countries (Canada, USA, Germany, Ireland, Brazil, and, Bangladesh) 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 in 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) Pilot Studies: We conducted two pilot studies 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 in-person or over video conferencing. 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 and a set of API tasks and online answer forms to complete using the API within an hour. The primary findings from this pilot are as follows: 1) participants required significant overhead time to setup a development environment with the proper API credentials, 2) the intermediate trial attempts of using the API are potentially

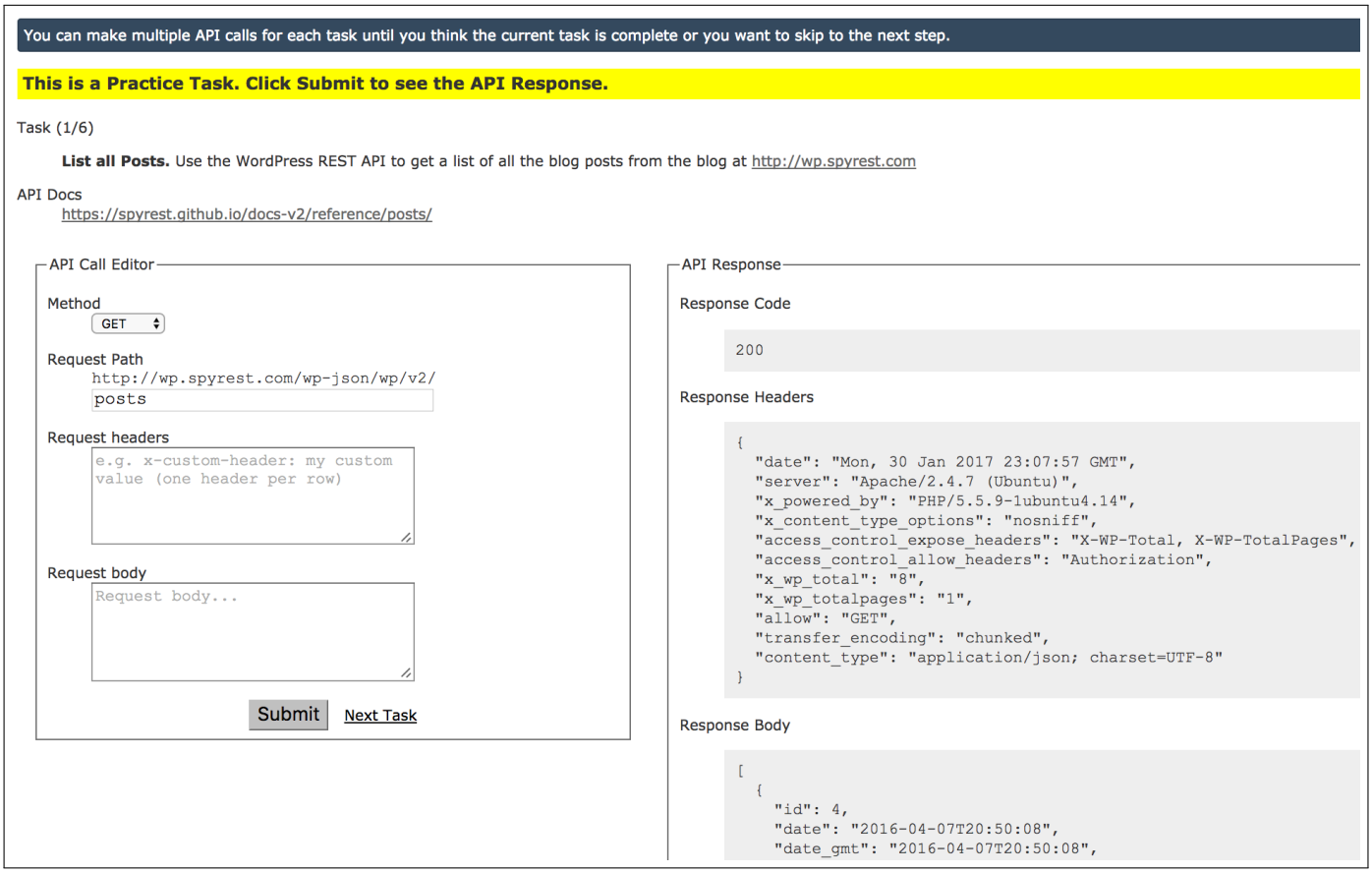


Fig. 2: Screenshot of the Custom-built Web-based REST API Explorer Used by Study Participants

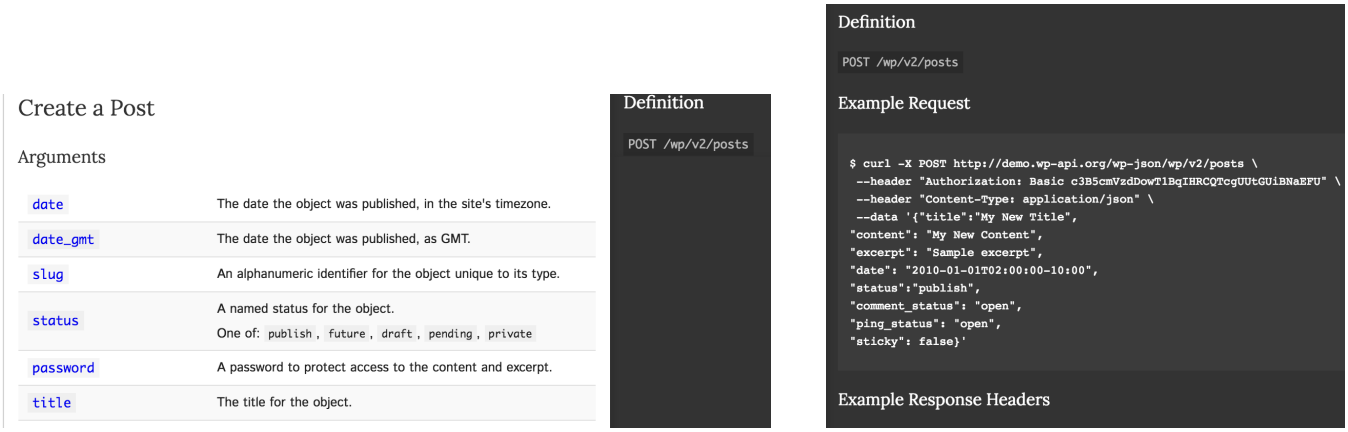
more valuable than the final answer as it allows us to study what obstacles the participants face, 3) the number of tasks for the study had to be reduced to fit within the one hour limit, and 4) for the GMail API, participants used up a large portion of their time on learning about how to use OAuth.

To overcome the shortcomings found from the first pilot study, we decided to develop a web-based REST API explorer as shown on Fig. 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 collect usability related feedback about the web-based REST API explorer. We only used WordPress since it did not require knowledge about OAuth. From this study, we observed patterns of mistakes that API client developers make that can be reduced by adding usage examples in the API documentation.

4) Protocol: Learning from the pilot studies, we designed the main study protocol as follows: we enhanced the original WordPress REST API documentation and added a total of 3 API usage examples 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. We used the WordPress API unit tests to find relevant data for these examples. Fig. 3a 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[6]. Fig. 3b shows a screenshot of the enhanced API documentation with a cURL[7] based usage example. cURL is used because it is used elsewhere in the original API documentation. In the enhanced 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.

We divided the participants 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 enhanced API documentation with usage examples. We designed the the

[6]http://v2.wp-api.org/reference/posts/
[7]https://curl.haxx.se/



(a) Original API Documentation

(b) API Documentation Enhanced with an Example

Fig. 3: Screenshots of Original vs. Enhanced API Documentation

web-based API explorer to allocate 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-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 except an overall limit of one hour for the entire study because we wanted participants to spend sufficient time on each task without forcing them to move to the next one. Participants were allowed to access the internet and external resources alongside the provided API documentation to perform the tasks as they would normally use on a typical work day. We used the web-based API explorer to also collect an experience rating on a scale of 0-10, 10 being the best possible, of using the given REST API documentation and a free-form feedback from each participant as an exit survey.

5) 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.

6) Data Analysis: 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. We exported a table of data from the web-based REST API explorer. For each row, the table contains the following columns: API task, participant identifier, timestamp, API trial number, I1-4, response headers and body. 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 information type that API client developers need in the API documentation to perform API tasks. The analysis started with an empty set of codes and new codes were introduced to describe scenarios that did not fall under existing codes. The first author applied the codes on the raw data artifact and provided a coding scheme comprising 11 codes to a co-author. The co-author applied the codes to 82/539 API calls (10% confidence interval with 95% confidence level). The two authors validated the codes against each other to provide consistency. The initial coding from the co-author mismatched 4 / 11 codes for 27 / 82 API call samples, which were resolved when the coding was repeated after a discussion.

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 complete a single task (T2), we marked it as partially successful. Otherwise, it's marked as unsuccessful. Based on these annotations, the following formulas were used to compute the quantitative results:

Success Rate (Task, Group) = (No. of participants that successfully completed Task) / (No. of participants in Group)

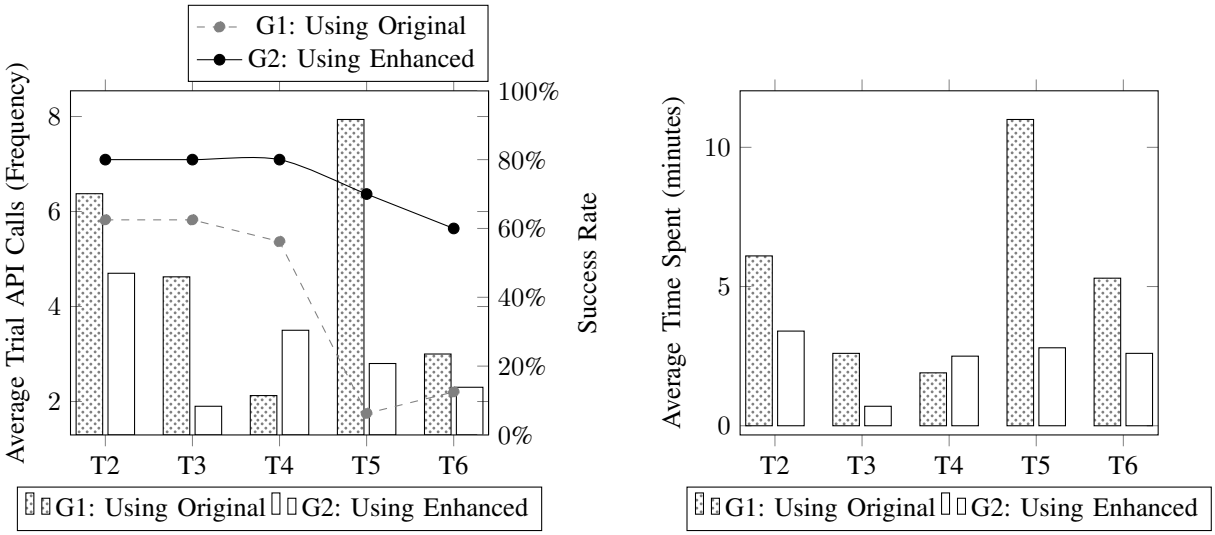
Average Trial API Calls (Task, Group) = (No. of API calls made by Group on Task) / (No. of participants in Group).

Average Time Spent (Task, Group) = (Time spent by Group on Task) / (No. of participants in Group)

IV. Results

A. Quantitative Analysis

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 Fig. 4. Fig. 4a 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 aforementioned formulas.



(a) Trial vs. Successful API Calls

(b) Average Time Spent

Fig. 4: Quantitative Comparison between G1 and G2

Even though it is expected that G2 would outperform G1 with an enhanced documentation, we were surprised to see the extent of the difference in effort vs. success of the two groups. In Fig. 4a and 4b we see that, for the simpler tasks, T2 and T3, G1 required more time and more trial API calls and reached successful answers with a lower percentage compared to G2. For T4, where the participants are required to inspect an HTTP response header, G2 executed more API calls on average, but also yielded 24% higher success rate. For the more complex 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 confirms the intuition that without usage examples developers spend more time and execute more trial and error attempts yet have lower success rates. And the situation gets worse for relatively complicated tasks.

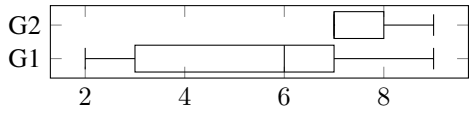


Fig. 5: Participation Satisfaction Ratings on a Scale of 0-10

In Fig. 5, 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.

The results provide an initial quantification of the impact of examples for typical REST-API usage tasks. This quantification is preliminary evidence that can help justify future investment in API documentation.

B. Qualitative Analysis Results

In their free-form feedback, 11 of the 16 G1 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 enhanced documentation. We identify usage examples as the most frequent and central topic of interest from our participants as per their feedback. This is also supported by our quantitative analysis results. In the remainder of this section, we provide our observations related to each task and their implications.

Observation 1. Use of Data Type. Without usage examples, we observed 10 of the 16 G1 participants used incorrect data type of `String` instead of `Integer`. 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 does not mention that the data type required is the numeric author ID, not the author name. 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 enhanced 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.

Recommendation 1. REST API documentation needs to include examples of data types such as Integer, String, and Array for each API field to satisfy API client developer information need.

Observation 2. Use of Data Format. We observed participants in G1 faced obstacles using correct data formats for `Date` and `Array` type data in API requests without usage 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."*. Seven of the 16 participants in G1 faced this error. The enhanced 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 the ISO8601 formatted date for publishing a post with a specific date. Even though the original documentation mentions ISO8601 format, it does not provide an example. Instead of using `2016-01-01T00:00:00-00:00`, participants from G1 used date formats as follows: `January 1, 2016`, `2016-01-01T12:00:00`, `20160101`. 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 did not provide the required `second` portion of the time. Eight 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.

Recommendation 2. REST API developers need to include examples showing the valid data format for the API elements.

Observation 3. Use of Request Headers. We were surprised to see that 14 of the 15 G1 participants that tried T5 and T6 faced problem using correct 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."* Even though G1 participants have years of experience with REST APIs, only one of them could use the `Content-Type` header correctly without example. In contrast, 2 out of the 10 participants from G2 failed to use this header correctly as indicated in the example in the enhanced documentation. Moreover, 1 of the 2 G2 participants that did not use the header the first time, later used it correctly on a subsequent attempt.

Recommendation 3. 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 4. Use of Interdependent API Calls. We observed the G1 participants had problem completing API tasks that require multiple API calls without usage examples. 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."*

Recommendation 4. 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.

V. Threats to Validity

The selection of the API, the tasks, and the participants may introduce selection bias. It is possible that there are obstacles we don't know about that could be more severe, but were not observed because the tasks did not involve anything related to these obstacles. The analysis of the raw data may introduce a bias. To reduce this, we have involved multiple researchers in the analysis process with a repeatable coding scheme.

VI. Related Work

In this section, we discuss the existing research on the relationship between API usability and usage examples. To this regard, we summarize the related work on APIs from the sub-areas of API learning obstacles, using crowd-sourced API examples, measuring API usability, and controlled studies on the impact of usage examples on API usability.

1) 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 many obstacles for API learnabilty were related to API documentation. They identified five impactful factors for API documentation as follows: documentation of intent, code examples, matching APIs

with scenarios, penetrability of the API, and format and presentation. Uddin and Robillard surveyed 323 professional software engineers to understand the different ways in which a piece of documentation can be unfit for purpose [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 and Robillard performed a study with 20 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. found that participants had limited success finding the relevant API elements to perform an API task using an enterprise API [5]. Among other factors, they recommended providing code examples in the API documentation to help API client developers. Our work fits in the general space of studies of API documentation obstacles. We focus on problems related to the absence of examples within the area of REST APIs.

2) Crowd-Sourced API Usage Examples: Wang et al. performed an exploratory study to understand the current state of API related knowledge available on the Internet [6]. They searched the web for usage examples related to the API of five popular Java libraries and found that on average API examples could be found for 77% of the 4,637 APIs from the libraries. Moreover, they found that the crowd-sourcing sites accounted for 93.7% of the usage examples compared to 6.13% that were officially published. An evaluation of Jadite showed that developers were three times faster to complete API tasks with access to auto-referred crowd-sourced usage examples [7]. Nasehi et al. analyzed StackOverflow threads to understand the characteristics of good and bad code examples [8]. They found that commonly down-voted API related answers lacked code examples, explanations, and shortcoming of solutions. 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 mostly focused on finding usage examples of local APIs. In this paper, we focused on REST APIs, where the API client development programming language is agnostic to API implementation language. This distinction with local APIs makes it hard to use the techniques proposed in the aforementioned papers for finding crowd-sourced REST API usage examples.

3) 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 [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. suggested using a combination of expert opinion and developer workshops to identify and collect feedback about API and its documentation related problems [14]. Farooq et al. recommended using peer reviews of API code in addition to API usability studies to uncover API related bugs and incorporate feedback [15]. In our work, instead of measuring API usability, we have focused specifically on the relationship of REST API usability with respect to usage examples.

4) Controlled Studies: 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]. 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. Endrikat et al. 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 found that the participant group using a statically typed API with explicit documentation could get the tests to pass in less time than the other groups. Ellis et al. performed a controlled study with two groups of 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 using factory method 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 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 local APIs; and, the participants are all professional software engineers.

VII. Conclusion

In this paper, we presented a set of problems that experienced REST API client developers face while performing API tasks using API documentation that lacks usage examples. We identified that, without examples, REST API client developers have trouble using the correct data types, correct data formats, and required HTTP headers and request body. We've also presented empirical evidence that by adding usage examples makes it possible to reduce mistakes, improve success rate and developer satisfaction of using the API. REST API developers can follow our recommendation as a set of guidelines while documenting REST APIs to improve API usability. REST API documentation tool developers can leverage our findings to improve reusable tool support for software developers.

Acknowledgment

We are grateful to our participants for their valuable time on this study. We thank the University of Calgary for funding this research.

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 *Proc. of the International Conference on Software Engineering*. IEEE Press, 2012, pp. 266–276.

[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.

[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.

[15] U. Farooq, L. Welicki, and D. Zirkler, "API usability peer reviews: A method for evaluating the usability of application programming interfaces," in *Proc. of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2010, pp. 2327–2336.

[16] S. M. Nasehi and F. Maurer, "Unit tests as API usage examples," in *Proc. of International Conference on Software Maintenance*. IEEE, 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 *Proc. of the International Conference on Software Engineering*. ACM, 2014, pp. 632–642.

[18] B. Ellis, J. Stylos, and B. Myers, "The factory pattern in API design: A usability evaluation," in *Proc. of International Conference on Software Engineering*. IEEE Computer Society, 2007, pp. 302–312.