Inappropriate Usage Examples in Web API Documentations

***Abstract*—Application Programming Interfaces (APIs) are common in software development to reuse other products. Although the documentation allows API consumers to learn about API usages, it can be unreliable. Here, we investigate the characteristics of inappropriate usage examples in web API documentation by extracting and comparing OpenAPI Specifications from usage example-response pairs. About 65.5% of the endpoints have some form of inappropriate usage examples. Furthermore, mismatches are classified into four categories: undocumented keys pattern, dynamic keys pattern, unreturned keys pattern, and type mismatched pattern. Our results suggest that the number of keys in the response is correlated with the number of mismatches. These findings should assist both API providers and consumers who deal with unreliable documentation in web APIs.**

***Index Terms*—Web API, REST, Documentation, Usage Example, Usability, Learnability, Reliability**

I. Introduction

Various studies have reported that more usable software such as that with simple easy to understand documentation or comments in the source code promote reuse instead of developers writing original code [1] [2] [3]. One reuse method is to apply Application Programming Interfaces (APIs) to develop the application [4].

Web APIs, which are invoked via a network with the REST protocol, have become popular in recent software developments [5]. Web APIs often have more complex interfaces than conventional APIs such as libraries or frameworks. This is because the HTTP, which is used to invoke web APIs, accepts many kinds of arguments and data. Thus, it is more difficult to understand how to use web APIs compared to other types of APIs. If web APIs are used incorrectly, serious bugs and defects may occur in the products [6]. Consequently, for API consumers to improve the reliability of their products, documentation to improve API's usability, especially learnability, is invaluable.

Documentation with usage examples is the most influential factor to web API's learnability [7]. Thus, we focus on these examples to understand the characteristics of inappropriate usage examples. Insufficient usage examples may confuse API consumers. Consequently, software implemented based on such documentation may not work correctly. For example, according to the issue #2366 of GitHub [8], the service returns floats although its documentation says the type is int64.

Specifically, we examine three research questions:

RQ1. What is the ratio of the APIs with inappropriate usage examples? Web API's documentation with incorrect usage examples exists. However, its prevalence is unclear. RQ1 will answer this question. This question aims to quantify the current situation.

RQ2. What kinds of mismatches are observed between the responses and the usage examples? Although RQ1 will provide the ratio of unreliable usage examples, it does not discriminate the mismatch patterns between the responses and the usage examples. To use APIs more effectively, mismatches should be categorized. RQ2 aims to classify mismatch types.

RQ3. Are metrics correlated with the number of observed mismatches? To predict whether the documentation is reliable, the characteristics of inappropriate usage examples must be elucidated. This research question should help to make such a prediction possible. In our study, we treated APIs and their documentation from the viewpoint of API consumers. Therefore, all metrics used to answer this question can be measured by API consumers.

The answers to these research questions should provide insight to address unreliable documentation of web APIs. To provide the answers, we compared OpenAPI Specifications (OASs) [9] extracted from a usage example-response pair of the APIs. OAS is a way to notate the specifications of the REST APIs. Because it is a simple, standard, and language-agnostic interface for APIs, both humans and computers can understand APIs without accessing the source code, documentation, or network traffic inspections [9]. API providers can define various API specifications such as a title, version, base url, request parameters, and responses. These specifications are well-structured and written in the JSON or YAML format.

In this paper, we investigate the reliabilities of usage examples in the web API documentation. Section II introduces related works. Then section III describes the metrics used in the research. Section IV shows the methodology and the results of this study. Finally, section V provides conclusions and future works.

II. Related Works

Martin P. Robillard et al. examined obstacles for developers when learning usages of APIs [10] [11]. Through questionnaires they asked developers about the challenges when learning the usages of APIs [10] and usefulness of different types of API documentation [11]. The responses indicated that unreliable documentation led to issues. Unreliable documentation was composed of insufficient or inadequate examples and the lack of references for the specific usages. These results support our motivation as knowledge about unreliable documentation should aid in efficient learning of API usages.

The effectiveness of usage examples in the documentation is supported by the research of S. M. Sohan et al. [7] Their study focused on obstacles for API consumers when learning the

usages from documentation without the usage examples. They demonstrated that participants understood documentation with usage examples faster than that without usage examples. In addition, usage examples provided higher satisfaction ratings. Thus, appropriate usage examples help improve the learnability of APIs.

Unfortunately, usage examples sometimes contain defects. Lin Shi et al. [2] found that Java library developers added examples to explain API usages, but later had to fix them due to defects. In addition, usage examples for web APIs had similar behaviors. Jun Li et al. [12] investigated the evolution of web APIs. They found that API changes could be summarized into 16 patterns. They also revealed six new challenges in migrating web APIs and several unique characteristics in the web API evolutions. For example, changes such as adding, removing, and changing parameters in both the requests and responses could affect the usage examples. Furthermore, the response format sometimes changed from XML to JSON. Such modifications may force usage examples to change boldly.

These studies have contributed to the knowledge of appropriate documentation for API's learnability. However, knowledge about common documentation failures of web APIs does not exist.

III. Metrics

Here we describe the metrics used in this research. Each one can be measured from API consumers' point of view.

ResponseTime is the time the web service takes to respond to data [13]. It is measured in milliseconds. The count begins before sending a request to a service and stops when a response is returned. Values are the average of five measurements.

ResponseLength is the length of the responses as the number of characters. We expect it to reflect the complexity of the service. In other words, more complex services should return longer responses. This metric does not distinguish between a large response with a lot of keys and that with a single large element.

NumberOfKeysInResponse represents the number of the keys in the response a service provides to the client. Unlike ResponseLength, this metric distinguishes between a large response with a lot of keys and that with a single large element.

In addition, we define **MismatchDensity** as the number of mismatches per OAS's rows. This metric explains the degree of correctness/incorrectness of usage examples in the documentation. This value is calculated based on two OASs. Figure 1 shows an example of an OAS. This index is from a well-structured, equivalent format with the original OASs.

MismatchDensity is defined as the following normalized edit distance of two dictionaries, dictionary A and B, which are equivalent in the original OASs. Since it is normalized by the number of OAS's rows, they are comparable to each other.

$$MismatchDensity(A, B) = \frac{D}{\max(len(A), len(B))} \quad (1)$$

In this definition, D represents the edit distance of two input dictionaries, while $len(X)$ represents the function that returns the number of lines of the OAS X. A value of 0.0 means that there is no mismatch between the two OASs. As shown in section IV, inputs are extracted from both the responses and the usage examples in the documentation. Examples (e.g., the last line in Fig. 1) are removed before calculating.

```
openapi: 3.0.0
servers:
- url: http://petstore.swagger.io/v2
paths:
  /pet/{petId}:
    get:
      parameters:
      - name: petId
        in: path
        ...
      responses:
        200:
          description: ''
          content:
            ...
                    example: 0
```

Fig. 1. Example of an OAS

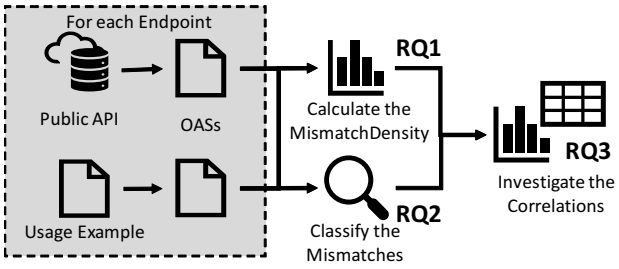

Fig. 2. Outline of our study

IV. Empirical Study

A. Methodology

Our research is composed of five steps: extracting OASs, calculate MismatchDensity, classify mismatches, investigate correlations, and building a prediction model (Fig. 2).

For each endpoint of the APIs, a pair of OASs is extracted to calculate MismatchDensity. To extract OASs, we referenced Hamza Ed-douibi et al. [14] Each pair includes the OAS from the usage example and that from the response. The OAS extractor produces a request-response pair as the input and output OAS. All the information used to extract the OAS is from the documentation and the responses. Thus, APIs are treated as black boxes.

After extracting the OAS pairs, MismatchDensity is calculated. As described in the section III, it explains the degree of correctness/incorrectness of the usage examples in the documentation. The results of this step answer RQ1.

While calculating the MismatchDensity, lists of mismatches for each pair are generated. In the classification step, these lists are categorized into patterns. The answer for RQ2 is derived for each cluster.

The final step investigates the correlations and provides an answer for RQ3. To discuss the correlations, we compute the correlation coefficient matrix.

B. Dataset

To investigate the reliabilities of the web API documentation, numerous usage example-response pairs are required.

TABLE I
LIST OF APIS IN THE DATASET

Domain	Name (# of Endpoints)
Animals	Dogs(8), Shibe.Online(1)
Anime	Studio Ghibli(10)
Books	BookNomads(1), Open Library(5)
Business	Domainsdb.info(1)
Calendar	Church Calendar(6), Hebrew Calendar(3)
Cryptocurrency	CoinDesk(3),CoinMarketCap(4),CryptoCompare(14),Nexchange(8)
Data Validation	PurgoMalum(1)
Development	APIs.guru(2),Bored(1),DigitalOcean Status(8),Genderize.io(310),GitHub(40)

These pairs should be collected from various web services. In our study, we collected 119 endpoints from 18 APIs. We considered the following conditions:

- There is at least one usage example in the API documentation.
- The API call is successful.
- The name of the API is listed in a repository called "toddmotto/public-apis" on GitHub [15].
- The response is in the JSON format.
- Documentation is written in either English or Japanese.
- The API is free to use.
- The API does not require an api key or authorization.

To extract the OAS, a request-response pair is needed. The first two conditions are based on this requirement. Usage examples are used to extract the OASs from the documentation, and the results of API calls are used to extract from their implementations. Because the OAS extractor only accepts the JSON format as a response, APIs with XML responses are excluded.

The last two conditions are for practical reasons. Due to the number of required APIs, paying API royalties is a hurdle. Furthermore, signing up for different service is another obstacle.

From the top to the bottom of the list on the repository named "toddmotto/public-apis" on GitHub [15], we selected APIs meeting the aforementioned conditions. Although there are a lot of APIs on the list, we picked up 18 APIs due to the various documentation formats and difficulties automating usage example extraction. As discussed later, this limitation may be a threat to validity. Table I lists the APIs[1]. By executing the OAS extractor, a pair of OASs is generated for each endpoint.

C. Results and Discussion

1) RQ1: Figure 3 plots the distribution of MismatchDensity in a histogram. Up to 41 of the 119 endpoints have a value of 0.0. In other words, about 65.5% of the endpoints have some mismatch between their two OASs. This is the answer for the RQ1. The results indicate that documentation should be checked more carefully.

Because each OAS is extracted from a request-response pair, more than half of the endpoints have differences between the usage example and the response. If a practitioner employs these APIs by reading the documentation and implementing the code, the product may contain defects.

[1]Details of the APIs are available at https://gofile.io/?c=yh22rE

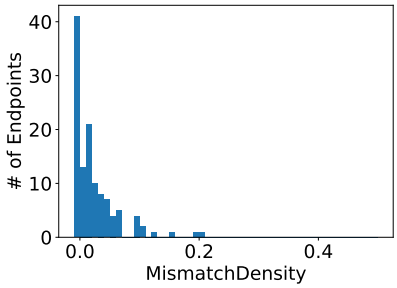

Fig. 3. Histogram of MismatchDensity

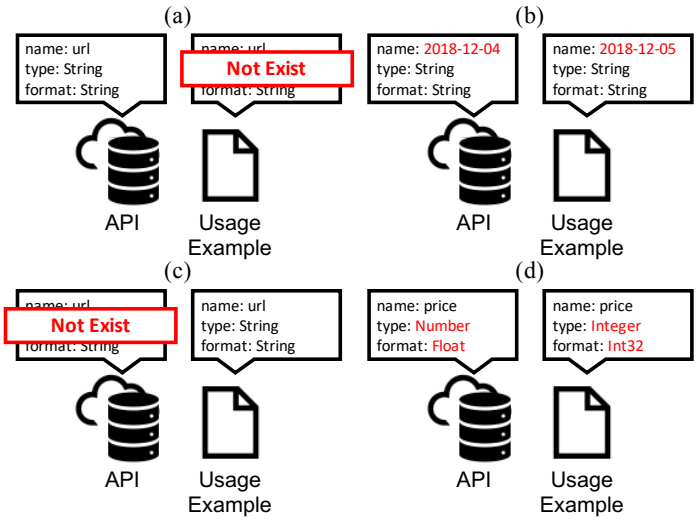

Fig. 4. Types of mismatches: (a) Undocumented Keys Pattern, (b) Dynamic Keys Pattern, (c) Unreturned Keys Pattern, and (d) Type Mismatched Pattern

2) RQ2: We checked all the differences between the two OASs. Four types of differences are observed (Fig. 4). Figure 5 show example responses of these four patterns, where "+ means that the corresponding row only appears the responses, and "- means the opposite. This is the answer for RQ2.

Undocumented Keys Pattern: This kind of difference occurs when the response contains a key that is not in the usage example. For instance, the response of the endpoint named "Return all films of the Studio Ghibli API (v1.0.1) contains a key called "url. However, this key is not in the usage example in the documentation (Fig. 5(a)).

```
  "release_date": "1986",                                    (a)
  "rt_score": "95",
+ "url": "https://ghibliapi.herokuapp.com/films/..."
},
```

```
{                                                            (b)
  "bpi": {
  + "2018-07-19":7470.825,
  - "2013-09-01":128.2597,
```

```
"domains":[                                                  (c)
  {
  - "name": "google.com, facebook--de.com",
    "hasWhois": 0,
```

```
  "rank": 1,                                                 (d)
+ "circulating_supply": 17217812,
- "circulating_supply": 17008162.0,
```

Fig. 5. Examples of Mismatch Patterns

Dynamic Keys Pattern: This kind of difference occurs when the response contains a key that dynamically changes its name. For example, the response of the endpoint named "Historical BPI data" of the CoinDesk Bitcoin Price Index API contains a key of dates (Fig. 5(b)). Thus, its name dynamically changes based on the date it is called. This type of the mismatch is not critical. However, the existence of such keys should be acknowledged.

Unreturned Keys Pattern: This kind of difference occurs when the response does not contain a key that is in the usage example. Responses of the endpoint "search" of the domainsdb.info API matches this pattern. The key "name" is in the usage example but not in the responses (Fig. 5(c)). This is the opposite pattern of Undocumented Keys. This is an issue. Developers may think that the response contains such keys and use it in their product, leading to serious defects.

Type Mismatched Pattern: This kind of difference occurs when the type of value in the response differs from that in the usage example. A typical example is the endpoint called "Ticker (Specific Currency)" of the CoinMarketCap Public API version 2. In the documentation, the value identified by the key "circulating_supply" is a float, although the value in the response is an integer (Fig. 5(d)). Many of the differences classified into this type will not cause serious defects because recent programming languages automatically cast the value into the best type. However, it might cause defects if it is the difference between specific pairs such as an array-immediate pair.

According to Jun Li et al. [12], there are various change patterns in the web API documentations. Some are considered to be the cause of mismatches. For example, mismatches classified into the undocumented keys pattern may occur when some parameters are added into the response but its usage example is not updated. Furthermore, mismatches classified into the unreturned keys pattern may occur when parameters are removed from the response but its usage example is not updated. The type mismatched pattern may occur when changing the parameter types.

In our study, we set a condition in the dataset as "The API call is successful." Actually, the calls failed for some endpoints. Similar to the observed patterns of the mismatches, this phenomenon may occur when the method but not its documentation is deleted.

Figure 6 shows the distributions of each type of mismatch. Undocumented keys pattern and unreturned keys pattern are the two most frequent. To deal with these mismatches, API consumers should ensure that their products have redundancies. For example, the effect of mismatches belonging to the undocumented keys pattern occurring by deprecation of the endpoint can be mitigated by implementing null-checks because such mismatches are due to the deletion of the keys from the usage examples before they are deleted from the responses. If API consumers implement null-checks into their products, they will become aware of key deletions. Furthermore, this measure will also mitigate the effects by an unreturned keys pattern. Taking measures against the mismatches belonging to the type mismatched pattern, API consumers should always be casting types, even if it is redundant.

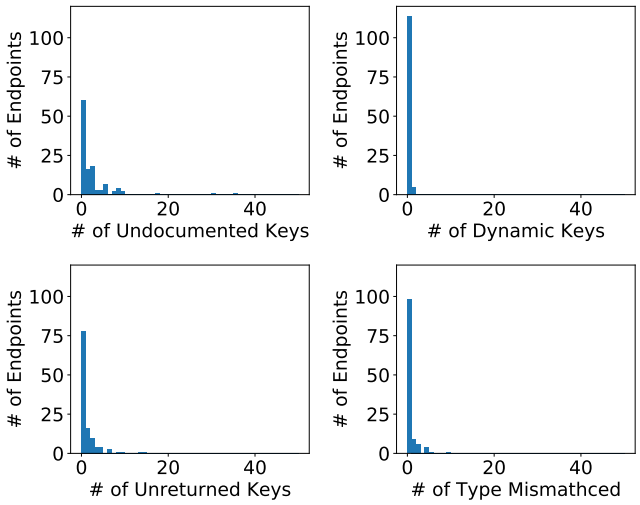

Fig. 6. Histogram of the number of mismatches by category

3) RQ3: Table II shows the matrix of the correlation coefficients. A value with "*" denotes a correlation coefficient with the significance level of 5%, while a value with "**" is that with a significance level of 1%. MismatchDensity (IX in Table II) is not correlated with the other metrics (VI-VIII). The answer to RQ3 is "No. On the other hand, the number of mismatches (V) is strongly correlated with the NumberOfKeysInResponse (VIII). NumberOfKeysInResponse is correlated with the number of OAS's rows because it depends on the amount of API definitions such as keys. Hence, the number of mismatches is correlated with the number of OAS's rows.

In the definition of MismatchDensity (Equation 1), the number of mismatches is normalized by the number of OAS's rows. However, NumberOfKeysInResponse should be valuable if the reliability of documentation is evaluated as the number of mismatches.

Documentation with a high reliability but not a MismatchDensity value of 0.0 is problematic. Thus, we separated the endpoints into two groups: those with MismatchDensity of 0.0 and all others. Then we analyzed the relationships.

Three extra figures represent the distributions of the metrics explained in section III for the two groups. According to the result of the Shapiro-Wilk test, these metrics do not follow a normal distribution. Therefore, the p-values of the significant differences are based on the Wilcoxon rank sum test where the null hypothesis is, "There is no significant differences between the two groups."

Figure 7a shows the boxplot for ResponseTime. There is no significant difference between the two groups for this metric (The p-value is 0.513). Hence, ResponseTime does not reflect the reliabilities of the documentation.

Figure 7b shows ResponseLength, and Fig. 7c is that for NumberOfKeysInResponse. Both distributions have significant differences (The p-value is 1.71e-5 in Fig. 7b and 2.12e-10 in Fig. 7b) Furthermore, a threshold clearly appears for

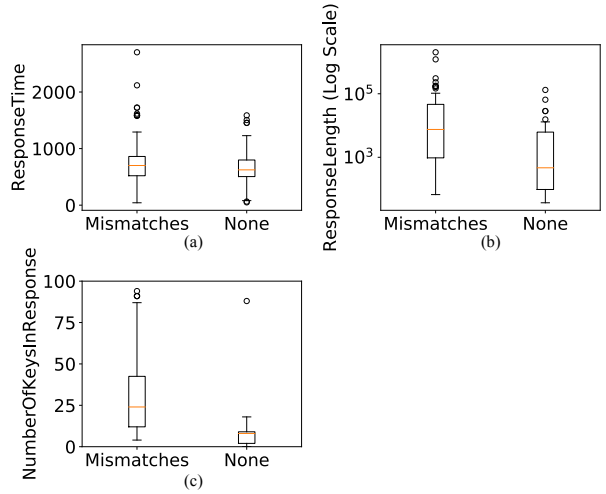

Fig. 7. Boxplots: (a) Distribution of ResponseTime, (b) Distribution of ResponseLength, and (c) Distribution of NumberOfKeysInResponse

NumberOfKeysInResponse, and its value is probably around 10. On the other hand, ResponseLength lacks a clear threshold. Thus, NumberOfKeysInResponse is more suitable than ResponseLength to predict whether the documentation is reliable.

Our findings will help API consumers improve their product's reliabilities. They can be summarized as follows. **Finding 1)** Over half of the endpoints somehow have unreliable usage examples. **Finding 2)** Mismatches can be classified into four types: undocumented keys pattern, dynamic keys pattern, unreturned keys pattern, and type mismatched pattern. **Finding 3)** NumberOfKeysInResponse may have a threshold around ten for the existence of mismatch.

These findings should be useful for API consumers. For instance, findings 1 and 3 demonstrate that API consumers should focus when reading documentation. If there are more than 10 keys, API consumers should read more carefully. Finding 2 should provide solutions to develop countermeasures to mitigate the effects of mismatches. Furthermore, API providers benefit from these findings. For example, finding 3 should help maintain sufficient and reliable documentation. API providers should polish their API documentation from larger OASs.

D. Threats to Validity

Although GitHub API v3 was used in our research, most APIs in our dataset are minor ones in industry. In most cases, industry uses famous web services with plenty of functionalities such as AWS and Google Cloud Platform. However, huge services require a usage fee to access, which limits our access. Our results may not be practical for industry.

A numerous APIs exist in the world, yet we collected endpoints only from 18 APIs. However, API documentation has a variety of formats. Hence, it is difficult to automate data collection, which is the biggest challenge in this research.

V. Conclusion and Future Works

We investigated the characteristics of inappropriate usage examples in the web API documentation by extracting the mismatches between the usage examples and the responses in an effort to verify the usability, especially the learnability of web APIs. About 65.5% of the endpoints have some form of inappropriate usage examples (RQ1). The mismatches can be divided into four types: undocumented keys pattern, dynamic keys pattern, unreturned keys pattern, and type mismatched pattern (RQ2). The value of NumberOfKeysInResponse may be correlated with the number of mismatches, and a threshold around ten may indicate the existence of mismatch (RQ3). Our results should help API providers and consumers deal with inappropriate documentation in web APIs.

In the future, we plan to implement some case studies based on this research. These studies will provide insights on the causes of the mismatches, which should prevent mismatches from the API provider's point of view. We also plan to interview developers to evaluate their experiences of defects due to unreliable API documentation. Furthermore, we will address the challenge of predicting fault-prone endpoints. Eventually, we intend to establish guidelines for API providers to enhance the reliability of their APIs.

TABLE II
Correlation Coefficients Matrix

	b	c	d	e	f	g	h	i
a	0.39**	-0.01	0.06	0.97**	-0.03	0.51**	0.82**	0.20*
b	-	-0.10	0.30**	0.41**	-0.16	0.58**	0.61**	0.12
c	-	-	-0.07	0.19*	0.01	-0.02	-0.03	0.13
d	-	-	-	0.16	-0.10	0.06	0.05	0.43**
e	-	-	-	-	-0.04	0.51**	0.80**	0.27**
f	-	-	-	-	-	-0.09	-0.18*	0.11
g	-	-	-				0.83**	-0.02
h								-0.04

a) Undocumented Keys, b) Dynamic Keys, c) Unreturned Keys, d) Type Mismatched, e) All Mismatches, f) ResponseTime, g) ResponseLength, h) NumberOfKeysInResponse, i) MismatchDensity

References

[1] S.G. McLellan, et al., "Building more usable APIs," IEEE Software, 15(3), pp. 78-86, 1998.

[2] L. Shi, et al., "An Empirical Study on Evolution of API Documentation, 14th International Conference on Fundamental Approaches to Software Engineering (FASE), pp. 416-431, 2011.

[3] J.C. Zanoni, et al., "A semi-automatic source code documentation method for small software development teams, 15th International Conference on Computer Supported Cooperative Work in Design, pp. 113-119, 2011.

[4] W. Maalej, et al., "Patterns of Knowledge in API Reference Documentation," IEEE Transactions on Software Engineering, 39(9), pp. 1264-1282, 2013.

[5] E. Wittern, et al., "Opportunities in Software Engineering Research for Web API Consumption," 1st International Workshop on API Usage and Evolution (WAPI), pp. 7-10, 2017.

[6] B.A. Myers, J. Stylos, "Improving API usability," Communications of the ACM, 59(6), pp. 62-69, 2016.

[7] S.M. Sohan, et al., "A study of the effectiveness of usage examples in REST API documentation," IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 53-61, 2017.

[8] https://github.com/Azure/azure-rest-api-specs (accessed 10-Jan-2019)

[9] https://swagger.io/specification/

[10] M.P. Robillard, "What Makes APIs Hard to Learn? Answers from Developers," IEEE Software, 26(6), pp. 27-34, 2009.

[11] G. Uddin, M.P. Robillard, "How API Documentation Fails," IEEE Software, 32(4), pp. 68-75, 2015.

[12] J. Li, et al., "How Does Web Service API Evolution Affect Clients?" 20th IEEE International Conference on Web Services, pp. 300-307, 2013.

[13] D.A. Menasce, "QoS issues in Web services," IEEE Internet Computing, 6(6), pp. 72-75, 2002.

[14] H. Ed-douibi, et al., "Example-Driven Web API Specification Discovery," European Conference on Modelling Foundations and Applications (ECMFA), pp.267-284, 2017.

[15] https://github.com/toddmotto/public-apis (accessed 10-Jan-2019)