A Data-Driven Approach to Measure the Usability of Web APIs

Rediana Koçi, Xavier Franch, Petar Jovanovic, Alberto Abelló
Universitat Politècnica de Catalunya, BarcelonaTech
{koci, franch, petar, aabello}@essi.upc.edu

***Abstract*—Application Programming Interfaces (APIs) are means of communication between applications, hence they can be seen as user interfaces, just with different kind of users, i.e., software or computers. However, the very first consumers of the APIs are humans, namely programmers. Based on the available documentation and the "ease of use" perception (sometimes led by corporate decisions and/or restrictions) they decide to use or not a specific API. In this paper, we propose a data-driven approach to measure web API usability, expressed through the predicted error rate. Following the reviewed state of the art in API usability, we identify a set of usability attributes, and for each of them we propose indicators that web API providers should refer to when developing usable web APIs. Our focus in this paper is on those indicators that can be quantified using the API logs, which indeed reflect the actual behaviour of programmers. Next, we define metrics for the aforementioned indicators, and exemplify them in our use case, applying them on the logs from the web API of District Health Information System (DHIS2) used at World Health Organization (WHO). Using these metrics as features, we build a classifier model to predict the error rate of API endpoints. Besides finding usability issues, we also drill down into the usage logs and investigate the potential causes of these errors.**

***Index Terms*—API usability, API logs, log mining, web API.**

I. Introduction

Application programming interfaces (APIs) represent the abstraction layer built upon sets of low-level methods and functions, in order to make them easily reused by third parties [6]. They are a means of communication between applications. Thus, we can say that APIs are user interfaces, just with different users in mind, meaning software or computers [22]. But we should not exclude from API users the human dimension.

Actually, the very first consumers of the APIs are humans, namely programmers. They are the ones who decide to use or not a specific API in their applications (sometimes under some corporate decisions and restrictions, e.g., pricing). If developers want to build a mobile application, and one of the features that they want to add to their application is the user location, usually it is up to them which location API to choose: Google Map API, OpenStreetMap API, Bing Maps API, Foursquare API, etc. Usually, API consumers decide to use an API by reading its documentation and by trying to perform different small tasks with it [22]. So, if API providers want to increase their costumer outreach or the number of users of their API, they should focus on improving the API documentation and its easy-to-use interface.

In this paper, we propose a data driven approach to measure API usability, based on how API consumers perceive and use APIs. Throughout our work we focus on web APIs, which differ from the traditional ones (statically linked APIs) mostly in the way providers and consumers are connected (via internet, typically by HTTP protocol) or how consumers adapt when APIs change (if web API providers decide to disconnect an older version of web API, its consumers are forced to upgrade to the newer versions) [19], [25]. We use the API usability taxonomy proposed by Mosqueira-Rey et al. [4], which is based on the work of Alonso-Rios et al. [3]. We adopt this taxonomy to describe the usability of web APIs. As we explain in more detail in Section III, there are some usability sub-attributes, which can not be investigated from the logs (they are related to the API source code and not to the interface). Therefore, for (almost) each of the usability sub-attributes, we propose some indicators that API providers should refer to. Based on these indicators, we define some metrics to measure each of the attributes. Then, we assess the relevance of these metrics in evaluating the usability of web API (reflected in the error rate of API endpoints), by building a classifier model that predicts the kind of error rate of endpoints based on the computed metrics.

We aim to not only find usability issues, but also to investigate about their root causes by drilling down into the usage logs. Consumers' behaviour is imprinted in these logs, so their monitoring and analysis is crucial as they can play an important role in revealing usability issues. Since log data are semi structured and often noisy, we explain how to perform the pre-processing step before doing the analysis.

We introduce and further use the following concepts:

- API resources - the data that API provides.
- API endpoints - the location where the resources can be found.
- API elements - resources, parameters, schema attributes.

Here we focus only on the interface level of the APIs. In other words, we evaluate API usability abstracting from the implementation code and the functionalities that API offers. Putting all together, our study is driven by the following research questions:

RQ1: Which are the usability sub-attributes that mostly influence the API consumers experience?

RQ2: Can we find issues impacting these usability sub-attributes by analyzing the API usage logs?

RQ3: Can the usability issues found in the API usage logs be measured in a meaningful way from the API consumer point of view?

DOI 10.1109/SEAA51224.2020.00021

RQ3.1: How to carry out the pre-proccessing of API log data before performing usability analysis?

The main contributions of the paper are as follows:

- We perform an empirical study in measuring the web API usability, by monitoring and analyzing the API usage data.
- We define and adapt a set of measurable indicators for web API usability attributes, and quantify in terms of API usage log traceability the indicators of one of the usability attributes, know-ability.
- We perform an experimental validation of the importance of these metrics, by applying our approach in a real-world case study.

The remaining of this paper is organized as follows: in Section II we present the related work on API usability evaluation. In Section III we frame our approach. We describe the API log data pre-processing phase in Section IV and present our case study in Section V. We discuss our findings in section VI, and conclusions and future work in Section VII.

II. Related Work

The evaluation of APIs usability has gained a lot of attention in the recent years [2], [14]. Based on the methods used, we can see two main categories: works that analyze the APIs design and their structural metrics, not taking into account how the API is being used (analytic methods [5], [9], [10]), and works that study how the API users are using the API (empirical methods [1], [13], [7]). We give a general overview of these works here, while the relevant usability attributes that we use in our study are explained in Section III.

Rama et al. [5] presented a set of structural metrics that can be evaluated on the API interface, or on the API implementation source code. Scheller and Kühn [9] also provided several metrics to measure the usability of APIs. They conducted a literature review to identify factors that affect API usability, and investigated more in depth few of these factors. De Souza and Bentolila [10] provided visual representation of APIs complexity based on complexity metrics of Bandi et al. [15].

Gerken et al. [1] on the other hand, used the concept map method to study and evaluate API usability. But, their longitudinal approach can only be applied on long time segments. McLellan et al. [11] used the think aloud protocol. They gave API code examples to four programmers and asked them to analyze and understand. They found usability testing very effective, based on the usability issues identifies by the participants. Thus, they suggested iterative API redesign and testing phases. Piccioni et al. [7] designed an empirical usability study by interviewing 25 programmers, and giving them a concrete task to accomplish using the API under study, in order to compare the participants' expectations with their actual performance. Grill et al. [13] evaluated API usability by applying a methodology of three phases: a heuristic evaluation (based on Zibran et al. [21]), a developer workshop and interviews, following this way both analytic and empirical methods.

Actually, as Rauf et al. [2] stated in their work on systematic mapping of API usability studies, the most used methods to evaluate APIs usability were empirical ones: usability tests, controlled experiments, surveys, etc. In fact, these methods are the most consolidated in software usability or human computer interaction (HCI) studies. As APIs differ from regular traditional software (can be used in scenarios that even API designers have not thought before, are prone to frequent changes, etc.), their usability evaluation requires more automated, scalable and time-efficient methods [2].

Few studies focus on mining repositories. Zibran et al. [21] studied five different bug repositories to identify the most reported API usability issues (by the API consumers). More than one third (37.14%) of the bug-reports in their study were related to API usability. From these, the most frequent ones were about missing features, correctness, and documentation. Macvean et al. [8] analyzed specifically the usability of web APIs. They took data from Google API Explorer to identify APIs, with which developers struggle more and spend more time and effort in learning and using. The metric they used to measure API usability was API request error rate (client side erroneous requests (4XX) per total requests to the API). Nonetheless, they recognized that other metrics can also be considered to better evaluate the usability, like API consumer satisfaction measured by API surveys or the number of erroneous requests consumers make until they achieve a successful call (known also as time to first *hello world* or *ah ha moment* [17]). Although their results were still preliminary and, as they stated, early in nature, the methodology used seems promising and opens a lot of areas for future research. Murphy-Hill et al. [12] developed a tool that analyzed consecutive snapshots saved by developers to derive problems of the API. They checked the API methods that developers changed between snapshots, but more than the 'worst' usability problems, these changes reflected the most used API. They suggested the use of other heuristics, like analyzing consumers experience, comparing the experience of novice API consumers with the more familiar ones, etc.

Mosqueira-Rey et al. [4] used both analytic and empirical methods. They adopted a general usability framework from Alonso-Rios et al. [3]. They derived from it a set of heuristics and guidelines for traditional APIs, and used these to evaluate the usability of a given API. Nevertheless, they pointed out the need to expand their work towards web API. Indeed, we use and follow their taxonomy that comprises six top-level attributes, refined into 21 more specific sub-attributes, to give and define a set of indicators that web API provider should use to offer usable web APIs. As Wittern et al. [19] stated in their work, there is still a lack of research on the consumption of web API (strongly related to the usability). As already annotated, web APIs differ from the traditional ones, hence, new methods need to be used in evaluating their usability, as well as other assumptions need to be taken into account. To understand and analyze the web API consumers behaviour, we do not conduct observational or controlled experiments, but instead monitor and analyze the web API log data, which indeed contain the interaction between API and its consumers.

TABLE I
API USABILITY

Usability Attributes	Sub-attributes	Indicators	Traceability in the logs
1.Know-ability	1.a Clarity	API elements' name clearness, descriptiveness, unambiguity, similarity [21], [12], [9], [5], [7]	✓
	1.b Consistency	Uniformity in naming API elements [21]	✓
	1.c Memorability	The number of API endpoints' parameters [21], [5], [9], [15]	✓
	1.d Helpfulness	The use of different status codes for different situations [21], [5]	✓
2. Operability	2.a Completeness	Consumers workaround solutions for the missing features [23]	✓
	2.b Precision	Proper data types to avoid loss of precision and unnecessary type-casting [3], [21], [13], [5], [9]	✗
	2.c Universality	The use of universally recognized names, formats, etc., for API elements [4]	✓
	2.d Flexibility	Multiple ways to do the same thing [21], [7]	✓
3. Efficiency	3.a In human effort	The balance between the flexibility of having different ways of doing a task and the complexity of having too many options [7]	✓
	3.b In task execution	API response time [17]	✓
	3.c In tied up resources	The excessive use of shared resources made by API [4], [21]	✗
	3.d To economic costs	The excessive costs required for the API use [4]	✓
4. Robustness	4.a To internal error	The proper handling of internal errors [9], [21]	✓
	4.b To improper use	The proper handling of consumers errors [9], [21]	✓
	4.c To third party abuse	The handling and mitigation of abusive behaviour of third party	✓
	4.d To environment problems	The handling of errors coming from environment problems	✓
5. Safety	5.a User safety	The use of safe HTTP methods to change resources	✓
	5.b Third party safety	The confidentiality protection of the users' personal information [4]	✓
	5.c Environment safety	The security of the users' assets [4]	✗
6. Subjective satisfaction	6.a Interest	The trend of API users over time (API new consumers, API churn rate) [17]	✓
	6.b Aesthetic	The aesthetic of API elements' name (no weird names or special characters used in an inappropriate way) [4]	✓

III. The proposed approach

In this section we describe the key elements of our approach. We start with explaining the different types of API logs and which usability attributes can be evaluated using each of them. Then we introduce the usability attributes, sub-attributes and the indicators inferred and adapted from the literature review. Last, we interpret these indicators in terms of API usage log traceability, where possible.

A. Measuring web API usability in web API logs

Web API usage logs can be collected at the provider side, at the consumer side, or at proxy servers [20], [26]. The usage data collected at each case reflect different aspects of the API usage. Data collected from the consumer side have all the requests made to the API, but only from that consumer. Log data from different consumers should be gathered to have more generalizable results from the analysis. On the other hand, the data logged at the API provider side contain information about all the consumers of the API, but if consumers have adopted API response caching, they do not record the requests for which the responses are cached. Moreover, development logs cannot be distinguished from production logs [8].

Nevertheless, the information transmitted through development logs differs from those in production logs. The former are created while the developers are creating and testing their applications. We can see the developers learning curve as well as their difficulties while using the APIs, only on development logs [8]. On the other hand, production logs are created during the post-development phase of the applications, after the applications are launched and used by their end users. The analysis of these logs can give us insights about API new usage scenarios, not evident even to API providers. Here we can address issues related to all the other usability aspects.

All in all, we cannot evaluate all the usability attributes in one type of API logs: different API logs are needed to evaluate different attributes. For example, we cannot measure the completeness of an API, if we have development logs from one API consumer. We need production logs from different API consumers to conclude if the API in study lacks in some features, forcing this way its consumers to come up with different workarounds.

B. API usability aspects

API usability represents a qualitative characteristic [2]. As such, there exist different interpretations, different terminologies, and different definitions [2], [3]. After choosing the usability taxonomy to expand for web APIs, we performed a literature review in order to quantify each of the sub-attributes into indicators.

Initially, we searched for works on API usability assessment, for both traditional and web APIs. As in our study we measure the usability based on the API interface, we filtered out the works that focused their analysis in the API implementation code. Next, we consolidated into indicators for each sub-attribute, all the information gathered. Then, considering that most of the information was for traditional APIs, we adapted it for web APIs. For example, Rama et al. [5] mentioned in their work as structural metric the one related to exception classes "Using exception throwing classes that are too general with respect to the error conditions that result in exceptions". We link this with the status codes returned, and give as indicator the use of different status codes in different situations, so that web API consumers would know the exact error that happened. We classify this as an indicator for the helpfulness of the API. Finally, there were some sub-attributes for which we could not find any pertinent information in the literature review, thus we extend the current state of the art with additional indicators. Table I summarizes our findings and reports main usability attributes with their sub-attributes, their indicators, and points out if such indicators can be traced in API logs.

C. Usability issues detected in API usage logs

As already stated, we evaluate API usability by analyzing API usage logs. Due to different server setting parameters, logs may comply to different formats, but typically each log entry has information about the client's IP address, the request time, the request method (GET, POST, etc.), the request body, the protocol, the time needed to respond to the request, the status code, and the size of the object returned.

We evaluate from the API logs those attributes and sub-attributes that can be mapped to the information in the log entries (see Table I). Thus, we focus on indicators consisting of the information about the interface of the API (naming of API elements, number of parameters) and indicators about the interaction consumer - API (status codes, duration, request sequences).

As a matter of fact, not all the usability sub-attributes of the taxonomy can be evaluated based on the information in the logs. For example, the precision of the API, an operability sub-attribute, is mostly related to the precision of the data types used. Data types selection is seen as too critical. API consumers should not perform type casting when it is not necessary, as this will not only increase their effort but also affect the precision [3], [21], [13], [5], [9]. However, in the log entry, requests are just strings, so we cannot analyze the parameters' data types (part of the implementation code of the functions and methods under the APIs).

Know-ability implies the ease of understanding, learning and remembering the API. Among others, this attribute is mainly related to the naming of the API elements. To properly evaluate the naming, it is essential to take into account the purpose and the functions of the API elements, and then to perform semantic analysis of their names. In automated solutions this is almost infeasible [9]. Hence, the controls that we perform for clarity, consistency, and memorability sub-attributes are channeled in names' similarity, the style of naming, path/query length, query complexity features, etc. On the other hand, helpfulness is related mostly to accurate documentation and detailed error messages. In the logs this can be manifested in the erroneous repeated requests.

Operability is mostly associated to the API consumers' needs fulfillment. Tracking workarounds built by consumers when API is not offering them a direct solution, is quite difficult. Anyway, consumers interaction with the API is imprinted in the API logs, so from there we can infer behaviours that address this issue. Part of operability is also universality, which implies the use of universal names and symbols. Next, flexibility is a double-edged sword: for experienced programmers, it is considered beneficial, but for novice ones, it increase the complexity of APIs (as explained below).

Efficiency can be evaluated in terms of human effort, time and resources spent while using the API. Regarding human effort, usually, the more complex the API is, the more effort is spent from the consumers' side. Complexity is a very general concept, and comprises several usability attributes. As having several ways to do the same thing sometimes confuses the programmers [7], we consider this as an indicator in evaluating efficiency according to human effort. Efficiency in task execution is reflected in the time that an API needs to respond to a request, which in the logs is stored as duration. For the costs of using the API, we look at the API endpoints, that can be optimized regarding number of calls. Consumers have to pay for number of calls for non-free APIs. So, if there are endpoints that are always called one after the other, their merging can reduce the consumers' expenses. We do not have a log-based indicator for the efficiency in tied up resources sub-attributes, as this is not related to the API interface, and is not reflected in any log entry fields.

Robustness, defined as the property of an API to handle errors and adverse situations, it is strongly related to the status codes that APIs send to their consumers. Even thought applications that consume the API should also be robust and treat properly the error situations, APIs should not fail in front of incorrect or even improper use. Therefore, APIs have to handle both the errors that come from their side (5xx errors), and the errors from the consumers' side (4xx errors).

Safety deals with the challenge of mitigating risks or damage while the consumers are using the API. Consumers typically access web APIs using HTTP requests. APIs should not allow the consumers to change resources using safe HTTP methods (methods that do not modify resources, e.g. GET). The safety of the APIs is also associated with the third party safety, as well as API environment safety. For the last one, we do not have a log-based indicator.

Subjective satisfaction is the capacity of the API to engage their users and preserve their interest. API providers can evaluate this by monitoring the trend of new API consumers. Additionally, the aesthetic of API is related to the aesthetic of API elements' name.

IV. API LOG DATA PRE-PROCESSING

An API log file contains the requests made to the API. This information is raw, so before applying any analysis technique, the data should undergo a pre-processing phase (see Figure 1), typically counted as the most difficult task in the Usage Mining process [18], [20]. While it usually consists of three steps, namely data fusion, data cleaning, and data structuring, we include a fourth step, data generalization [18]. See the final steps in Figure 1.

Data fusion. We already explained part of the data fusion step discussing different types of API log data (see Section III-A). Based on where we collect the data (consumers' or providers' side), we extract the log files and merge them (if they are in different servers). Before proceeding with the pre-processing steps, the data might be anonymized, because log files might contain information considered sensitive (identifiable personal information). One way of handling this concern is by masking the sensitive data (i.e., IDs, or IP addresses) with surrogate identifiers and then proceed with the next steps [18].

Data cleaning. This step mainly consists of removing irrelevant and noisy data from the files, and correcting the data by means of adding or completing missing values. When fusing data from several sources, the logs from different sources can have different formats. Thus, when merging them, we may need to adapt their formats: adding/removing certain fields, dealing with quoted/unquoted fields, etc. On the other hand, whether to keep or remove certain log entries depends much on the purpose of the further analysis [18]. For example, if one wants to discover user session profiles in web log data, he/she should filter out: (i) the log entries that result in errors, (ii) the log entries that have a request method different from GET, and (iii) the ones that accesses image files [16]. If the purpose of the analysis is to support caching or pre-fetching, then log entries for accessing images should not be excluded from the analysis [18]. Since we aim to measure the usability of the APIs, the status code is one of the most valuable fields in the log entries. The error rate of each API endpoint can reveal usability issues that can highly affect API consumers. Therefore, for purpose of measuring the usability of the APIs, we keep the erroneous requests, and filter out from the file, the log entries that are not API requests, and the ones that do not imply API resources manipulation. These are some typical examples, but we certainly do not exclude the possibility of having to remove other log entries, depending on how the information is logged. After this step, the number of log entries will be highly reduced [24], [18], [20].

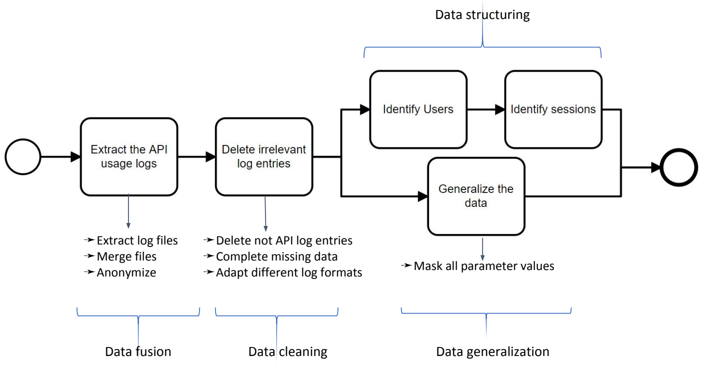

Fig. 1. Data pre-processing.

Data structuring. This step includes user and session identification. By users here we mean the applications that are consuming the API, so this step should be performed when working with API logs from the provider side. Ideally, when an application uses an API, the logs generated should have an ID that identifies their pathway with the API. Usually, the log file contains only the device's address (i.e., IP) and the user agent (i.e., software agent like a browser or an email reader). When applications' identifiers are not in the API logs, each IP might be counted as a user [18]. On the other hand, log data are usually not completed with the session ID, and session identification results also specially challenging, due to several reasons (i.e., caching, proxy servers, same device used by several users, etc.) [18]. Thus, session must be inferred combining available user identification and approximate time-out simulating the time spent by a single user using the API.

Data generalization. This step is considered an advanced pre-processing step, comprising one of the most complex tasks in API log data analysis [19], [24]. It consists of extracting general API specifications from the requests in the log files. For instance, if we have "`https://.../api/country/Spain/regions`", the challenge would be to detect "`/country/`" and "`/regions/`" as resources' name (fixed part of the path) and "`/Spain/`" as a parameter value (dynamic part of the path). Synthesising general API description from API usage is a hard problem, and existing solution are hampered by API logs nature (noisy, incomplete) and also API design and implementation problems [24], [19].

V. Case study design

A. DHIS2 Web API

We analyzed the log data of the District Health Information Software 2 (DHIS2) web API. DHIS2 is an open source, web-based health management information system platform used worldwide for data entry, data quality checks and reporting. It has an open REST API, used by more than 60 native applications. External software can make use of the open API, by connecting directly to it or through an interoperability layer.

DHIS2 is as well instantiated as WHO Integrated Data Platform[1] (WIDP) at World Health Organization (WHO), and is used by several departments for routine disease surveillance and country reporting. For the analysis, we use API log data from the development instance of WIDP, with more than 50 applications installed, core or built in-house. The logs date from September 2018 to November 2019.

B. Data pre-processing

We instantiate the data pre-processing workflow previously introduced in Section IV, and further discuss the challenges encountered in different steps of the process.

Data fusion. As previously mentioned, we had the log data from WIDP, which is a DHIS2 API consumer. But as several applications are installed on this platform, it partly behaves as provider. The logs were recorded using a customized Apache log format, which contained the following information: client IP, request long date (date, time, timezone), duration (time needed to send the response), keepalive, request (method + resources URL + protocol), response code, the size of the object returned and the user agent.

Data cleaning. We discarded the log entries with request method HEAD or OPTIONS, as they do not imply any resource manipulation or resource retrieval [24]. We filtered out also the log entries that were not related to APIs, thus

[1]http://mss4ntd.essi.upc.edu/wiki/index.php?title=WHO_Integrated_Data_Platform_(WIDP)

not containing the "/api/" entry point, predefined to be in the API request. These are usually log entries for loading of style files, fonts, or graphics. But at the same time we were careful not to remove key log entries (i.e., log entries about the login, logout and applications' first access) that, even though did not contain the "/api/" entry point, played an important role in data structuring. After data cleaning, our log file contained 2,268,291 log entries (i.e., requests), out of 5,936,203 that it had initially.

Data structuring. For user identification, we used the information under "client IP" in the log entry to identify different API consumers (i.e., users). We considered as a session all the requests made by one user (client IP), with time difference no more than 15 minutes. We assigned incrementally a number to each session as an identifier, ending up with 40,067 sessions, from 849 users. As we were analyzing the know-ability of the API, our focus was not on the applications that were using the API (i.e., the real consumers of API), but on the programmers (i.e., the first consumers of the API). However, if measuring, for example, the completeness (investigate about workarounds) of an API, one should identify which application submitted each request, in order to be able to build an exact sequence of the requests for each application. As already mentioned, in our case, the log files contained log entries from each of the applications installed in the platform. But the log entries did not have information about which application submitted each requests. The only information we had was the log entry specifying the opening of any application: "`GET /dhis2-dev/{nameOfTheApplication}/index.action`". But the same user could open several applications at the same time. So, from the moment the same user (ClientIP1) opened more than one application, we could not be sure about the origin of the next log entries:

```
ClientIP1 "GET /{App1}/index.action"
ClientIP1 "requests from App1"
ClientIP1 "GET /{App2}/index.action"
ClientIP1 "requests from App1 or App2"
```

In these analysis scenarios, we might need to make approximations and combine IP information, current app(s) opened and the timeout. The lack of applications' identifiers can impede not only the user identification, but also hinder the data cleaning.

Apart from user and session identification, we performed also "target endpoint" identification. In order to be able to aggregate statistical information for each endpoint (total number of requests, requests with client side error, etc.) we assigned a "target endpoint" to each request that got a client side error response code. For example, if we want to compute statistical metrics for the endpoint "`/api/organizationUnitGroupSets`", then we should somehow match it with the endpoints: "`/api/organization-unit-group-sets`" or "`/api/organizationsUnitsGroupsSets`", which have a wrong syntax, but the programmer intent was the first endpoint. We started by extracting all the requests that got a status code not related with syntax errors. Using the Levenshtein distance algorithm [27], we computed the similarity of these endpoints with each real endpoint in the requests body, and grouped together the most similar ones.

Data generalization. During this step we had to define which parts of paths were fixed (i.e., resources) and which were dynamic (i.e., parameter values). That is, for several endpoints with path body like "`api/userGroups/uNJOBaIw/users/H4atNsEr`", or "`api/userGroups/BzbYRSpk/users/D78WJM8J`", we inferred from them a general one, with generic API specifications "`api/userGroups/ID/users/ID`". We applied some ad hoc procedures, and masked all the parameter values with the same string "ID".

C. Data Analysis

We assumed that an API that suffers from poor know-ability, will have a high error rate. For each sub-attribute, based on the defined indicators, we computed the below metrics and created a model to predict the kind of error rate.

- **Clarity:** We analyzed the endpoints' names and the similarity between them, assuming that similar names confuse users and increase the chances of making errors. We expect that endpoints with higher similarity will have more client side errors response codes. We split each "target endpoint" into its elements. For each of them, we found the most similar one, by computing their similarity. For example, the `account` resource had the highest similarity of 0.71 with `count`, `constants` 0.82 with `constraints`, and so on. Using this information, we then computed two metrics for each endpoint: the **average similarity** and the **maximum similarity**. For example, `userDataStore/gridColumns/eventCaptureGridColumns` has three elements, `userDataStore` with similarity 0.69, `gridColumns` with similarity 0.48, and `eventCaptureGridColumns` with similarity 0.72. So the endpoint average similarity coefficient will be 0.63, while the maximum similarity will be 0.72, coming from `eventCaptureGridColumns`.
- **Consistency:** We focus on the syntactical aspects of naming to evaluate the consistency. Thus, we analyzed the **naming style** of endpoints (names that contain only lower case or numbers, upper cases, underscores, hyphens, special characters, or more than one of these "styles"). Actually, we do not expect a specific naming style to be the cause of errors from client side. We will investigate the impact that the existence of several naming styles, can have on API consumers. Clearly, the semantic aspects (use of synonyms, homonyms, etc.) are also very important when we analyse the consistency of an API and can be evaluated using different tools for natural language processing. We will include this in our future work.
- **Memorability:** We analyzed the path part as well as the query part of the requests. First we computed the **path length** as the overall number of characters;

and stored under **path elements** the information about the number of elements in the path (e.g., "`analytics/events/aggregate/ID`" has three elements in the path body). Then, we examined the query part to quantify its complexity. We measure the **query length** as the number of characters; we analyze the query syntax and impute the **transformation** metric as the number or transformation functions in the query; we represent the **logical operators** as the number of logical operators used; we defined the **query depth** as the number of nested objects, giving to each nested object the same weight, despite the number of fields it contains, as we are analyzing the complexity of the query part from the programmer point of view (amount of code to be written) and not the machine point of view (amount of time to compute the results of the query); we counted the **query parameters**; and also the **schema attributes** used in each request. Both are part of the query length, commonly used to evaluate query complexity, but as they represent different ways of reducing the result size, we choose to count them separately. As we realised that consumers often use the same query parameter or schema attribute several times in a single request, we decided to reflect this also in different metrics: **unique query parameters** and **unique schema attributes**.

- **Helpfulness:** To measure this sub-attribute, we analyzed the endpoints that got repeated client side error codes from the same user. We assume that the lack of details in the error messages and the lack of examples in the documentation can lead consumers to repeat the same mistakes. Therefore, we grouped all the requests with the same "target endpoint", from the same client IP, and analyzed those that had 2 or more client side error. We imputed the **error repetition rate** as the number of error per total number of request for the same endpoint for each programmer.

Besides the metrics per each request, we computed the error rate as the number of erroneous requests per all requests. We first selected only those endpoints that were in requests with error rate greater than zero and divided them in two classes: endpoints with error rate higher than 0.3 were considered with poor usability, and the ones with error rate lower than 0.3 with no usability issues. There were 1128 endpoints with certain values of the metrics. In order to balance the classes distribution to 40% for poor usability and 60% for no usability issues, we randomly extracted endpoints with error rate zero to obtain the desired proportion.

First, in order to reduce over-fitting and facilitate interpretability of results, we performed attribute (i.e., metrics) selection, keeping only those metrics that were more relevant for predicting the class. We run the CorrelationAttributeEval with Rank method on WEKA[2], which ranks the attributes by measuring the correlation between them and the class. From all the aforementioned metrics, maximum and average similarity, query depth, logical operators, and path length were the more relevant ones.

[2]https://www.cs.waikato.ac.nz/ml/weka

Then, to see if we could predict the class based on these metrics, we built a decision tree using WEKA implementation of J48, with the default parameterization and 10-fold cross-validation. We used a classification model because we were interested in finding if the API had or not usability issues, more than predicting the exact error rate (as a regression model would imply). Thus, using the selected attributes from the information in the logs, we obtained a model able to predict the class of the endpoints with an accuracy of 72.25%. Considering that in our analysis we do not take into account other factors in APIs usability like: APIs functionality, semantic of API names, API programmers experience, etc., we aimed at a high precision, more than at a high recall. But, even though we were not aiming to find all the endpoints with usability problems only from the data in the API logs, our model performed quite good with a recall of 0.722 (Table II, III).

TABLE II
CONFUSION MATRIX

a	b	classified as
272	280	a
103	725	b

a=poor usability
b=no usability issues

TABLE III
J48 RESULTS

Correctly Classified	72.25%
Incorrectly Classified	27.75%
Kappa statistic	0.389
Mean absolute error	0.399
Weighted Avg Recall	0.722
Weighted Avg Precision	0.723

VI. DISCUSSION

To get finer insights for the gained classification, we analyzed in more detail the branches of the decision tree. We found that requests that did not have a query part tend to have lower error rate, even when the path had an average similarity higher than 0.61. The error rate was also low for those endpoints with both average and maximum similarity low, respectively lower than 0.61 and 0.75. On the other hand, the error rate was high for those endpoints that even though had low average similarity, they had at least one element in their path with very high similarity. Endpoints with high average similarity and a query part, had also high error rate.

Additionally, we examined closer the endpoints with higher error rate, to see which were the most common mistakes done by the programmers. We point out the following issues:

- Consumers try different name styles until they find the right one. The fact that in the same API, different resources are named using different styles, confuses them. For example, before typing `userRoles/ID`, one of the consumers tried with `userrole/ID` and `user-roles/ID`.
- We encountered another consistency issue in the naming of API, plural and singular form of resources' name. As some of the resources' name were in plural, and others in singular, consumers tried their different forms. For objects with composed names (i.e., multi-word names), the error rate increased. For example, before typing `organisationUnitGroups`, one consumer tried three different versions: `organisationUnitGroup`, `organisationUnitsGroups` and `organisationUnitsGroup`. The lack of consistency in naming resources decreases the memorability of the API. Different naming conventions (pascal or camel case, underscore, etc.) and

the semantic nature of the analysis needed were the main reasons why these two aspects (plural/singular form and multi-word names), were not reflected in any of the metrics.

- When analyzing the repeated errors for the same endpoints from the same consumers, to measure the helpfulness, we noticed low memorability perception from consumers. From 1,283 cases of repeated client side errors, 659 of them had a non client side error response code for the first request. This implies that even after understanding the API and submitting correct requests, consumers fall in mistakes.

Threats to validity. The main threat to construct validity involve the arbitrary chosen threshold of 0.3 for the error rate class. To mitigate this, we plan to re-define the threshold by conducting other use cases (i.e., independent datasets). This way, we will also minimize the external validity threat, whose main concern is related to the one API we have as use case.

VII. Conclusion and Future Work

We reviewed the current state of the art in API usability evaluation in order to identify those usability attributes that mostly affect the programmers experience in using the APIs. We combined the gathered information and for each usability attribute, we specified an indicator that web API providers should use to provide usable web APIs. We embodied the indicators for the know-ability attribute into several metrics, which we later computed using the web API usage data from our case study. In order to assess the significance of these metrics, we built a classifier to predict the kind of error rate based on the endpoints' specifications. Know-ability issues that more influenced the API consumers experience were more related to similar API elements' names, multi-word names, and lack of consistency in naming convention.

As future work, we plan to define more metrics for the attributes under study and expand our analysis to other usability attributes. In order to re-define the threshold and the generalizability of our model, we will take under study other use cases. On the other hand, to evaluate the analysis results we plan to conduct a supplementary empirical analysis, directly asking the API users for their opinion on whether they encountered usability issues while using the API or no.

Acknowledgment

This work is supported by GENESIS project, funded by the Spanish Ministerio de Ciencia e Innovación under project TIN2016-79269-R. We especially thank Dr. Lise Grout and Dr. Pedro Albajar-Viñas from the Neglected Tropical Diseases (NTD) department at WHO, for providing the use case.

References

[1] Gerken, Jens, Hans-Christian Jetter, Michael Zöllner, Martin Mader, and Harald Reiterer. "The concept maps method as a tool to evaluate the usability of APIs." In Proceedings of the SIGCHI conference on human factors in computing systems. ACM, 2011.

[2] Rauf, Irum, Elena Troubitsyna, and Ivan Porres. "A systematic mapping study of API usability evaluation methods." Computer Science Review 33, 2019.

[3] Alonso-Ríos, David, Ana Vázquez-García, Eduardo Mosqueira-Rey, and Vicente Moret-Bonillo. "Usability: a critical analysis and a taxonomy." International Journal of Human-Computer Interaction 26, 2009.

[4] Mosqueira-Rey, Eduardo, David Alonso-Ríos, Vicente Moret-Bonillo, Isaac Fernández-Varela, and Diego Álvarez-Estévez. "A systematic approach to API usability: Taxonomy-derived criteria and a case study." Information and Software Technology 97, 2018.

[5] Rama, Girish Maskeri, and Avinash Kak. "Some structural measures of API usability." Software: Practice and Experience 45, 2015.

[6] Pautasso, Cesare, and Erik Wilde. "Why is the web loosely coupled? A multi-faceted metric for service design." In Proceedings of the 18th international conference on World wide web, 2009.

[7] Piccioni, Marco, Carlo A. Furia, and Bertrand Meyer. "An empirical study of API usability." In 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, IEEE, 2013.

[8] Macvean, Andrew, Luke Church, John Daughtry, and Craig Citro. "API Usability at Scale." In PPIG, 2016.

[9] Scheller, Thomas, and Eva Kühn. "Automated measurement of API usability: The API concepts framework." Information and Software Technology 61, 2015

[10] de Souza, Cleidson RB, and David LM Bentolila. "Automatic evaluation of API usability using complexity metrics and visualizations." In 2009 31st International Conference on Software Engineering-Companion Volume, IEEE, 2009.

[11] McLellan, Samuel G., Alvin W. Roesler, Joseph T. Tempest, and Clay I. Spinuzzi. "Building more usable APIs." IEEE software 15, 1998.

[12] Murphy-Hill, Emerson, Caitlin Sadowski, Andrew Head, John Daughtry, Andrew Macvean, Ciera Jaspan, and Collin Winter. "Discovering API usability problems at scale." In Proceedings of the 2nd International Workshop on API Usage and Evolution, ACM, 2018.

[13] Grill, Thomas, Ondrej Polacek, and Manfred Tscheligi. "Methods towards API usability: a structural analysis of usability problem categories." In International conference on human-centred software engineering. Springer, Berlin, Heidelberg, 2012.

[14] Myers, Brad A., and Jeffrey Stylos. "Improving API usability." Communications of the ACM 59, 2016.

[15] Bandi, Rajendra K., Vijay K. Vaishnavi, and Daniel E. Turk. "Predicting maintenance performance using object-oriented design complexity metrics." IEEE transactions on Software Engineering 29, 2003.

[16] Joshi, Anupam, Karuna Joshi, and Raghu Krishnapuram. "On mining web access logs." UMBC Computer Science and Electrical Engineering Department, 1999.

[17] Gilling, Derric. "13 API Metrics That Every Platform Team Should be Tracking" Available: https://www.moesif.com/blog/technical/api-metrics/API-Metrics-That-Every-Platform-Team-Should-be-Tracking/ Accessed [04/12/2019]..

[18] Tanasa, Doru, and Brigitte Trousse. "Advanced data preprocessing for intersites web usage mining." IEEE Intelligent Systems 19, 2004.

[19] Wittern, Erik, Annie Ying, Yunhui Zheng, Jim A. Laredo, Julian Dolby, Christopher C. Young, and Aleksander A. Slominski. "Opportunities in software engineering research for web API consumption." In Proceedings of the 1st International Workshop on API Usage and Evolution. IEEE Press, 2017.

[20] Srivastava, Jaideep, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan. "Web usage mining: Discovery and applications of usage patterns from web data." Acm Sigkdd Explorations Newsletter 1, 2000.

[21] Zibran, Minhaz F., Farjana Z. Eishita, and Chanchal K. Roy. "Useful, but usable? factors affecting the usability of APIs." In 2011 18th Working Conference on Reverse Engineering. IEEE, 2011.

[22] Berlind, David. ProgrammableWeb "What Are APIs and How Do They Work?" Available: https://www.programmableweb.com/api-university/what-are-apis-and-how-do-they-work Accessed [06/12/2019]

[23] Bush, Thomas. "How Workarounds Reveal the True Needs of Your API Consumers" Available: https://nordicapis.com/how-workarounds-reveal-the-true-needs-of-your-api-consumers Accessed: [02/12/2019].

[24] Suter, Philippe, and Erik Wittern. "Inferring web API descriptions from usage data." In 2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb). IEEE, 2015.

[25] Espinha, Tiago, Andy Zaidman, and Hans-Gerhard Gross. "Web API growing pains: Loosely coupled yet strongly tied." Journal of Systems and Software 100, 2015.

[26] Kosala, Raymond, and Hendrik Blockeel. "Web mining research: A survey." ACM Sigkdd Explorations Newsletter 2, 2000.

[27] Levenshtein, Vladimir I. "Binary codes capable of correcting deletions, insertions, and reversals." In Soviet physics doklady, 1966.