A preliminary GQM model to evaluate web API usability*

Ariel Machini[1][0000−0002−2589−8182] and Sandra Casas[2]

[1] Centro de Investigaciones y Transferencias Santa Cruz, Río Gallegos, Argentina
[2] GISP, Universidad Nacional de la Patagonia Austral, Río Gallegos, Argentina

Abstract. Web APIs allow easy access to a variety of resources and services, and it is because of that they have become essential for building modern applications. This generated a new business perspective, "the API economy", and for that reason, usability now turns into a key characteristic for the acceptance of a web API. Although web API usability is a researched topic, no studies proposing a usability model for web APIs have been found. For that reason, this study presents a preliminary model to help standardize and facilitate web API usability assessment. Our model, based on the GQM approach, has six goals, eight questions, and 32 metrics. We evaluated its usefulness through a survey directed to web API consumers and developers. Feedback suggests that the model is on the right track, and could have a positive impact on web API usability in the future.

Keywords: Web API usability · Model · Metrics · GQM

1 Introduction

An API (Application Programming Interface) is a software-to-software interface that defines the contract for applications to talk to each other over a network without user interaction. The term API may mean different things to different people, depending on the context: There are APIs for operating systems, applications, and the web. Today, when we talk about APIs, we probably refer to web APIs built using REST (Representational State Transfer) technologies. Hence, web APIs are synonymous to REST APIs [1].
Over the last decade, web APIs have become one of the pillars of modern application development [2]. Web APIs allow building different kinds of software, from simple web apps that display data from multiple external sources to more complex applications based on microservices, such as Netflix, that take advantage of web APIs to create new value. In addition to promoting software reuse, and access to resources and services among organizations, web APIs recently generated a new business perspective known as the API economy [3]. The web API market is very competitive and, because of that, the usability of a web API becomes a key characteristic to determine its value [4].

* Supported by CONICET.

Usability is considered one of the most important software quality attributes [5] and, even though it has many definitions, the definition given by ISO 9241-11 is probably the most popular one: "The extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use".
Through a literature review—carried out to identify and categorize metrics that influence web API quality and usability—, we identified 96 metrics, of which 86 are related to API usability. This analysis also evidenced a lack of standardization in the area of usability models for web APIs, and a lack of research in the area of metrics and web API usability (only 12 of the 86 usability-related metrics were explicitly linked to web APIs). Despite there being studies that propose different methods and models for the analysis, improvement and assessment of API usability, they are mainly focused on local APIs, and centered on API documentation. [6] also states that "the common problem with the metric approach is that the works are somewhat associated with a usability model, but the metrics derived are not related with a usability characteristic or are associated with only a few of these usability characteristics".
Because of that, in this work, we present a preliminary GQM (Goal-Question-Metric) [7] model with the aim to facilitate web API usability assessment. This artifact, containing six goals, eight questions and 32 metrics, was designed following the DSR (Design Science Research) approach [8]. We validated our work through a survey specifically directed to web API consumers and developers.
This paper is structured as follows: Section 2 describes the methodology followed; Section 3 details the proposed model; Section 4 shows how the model was evaluated; Section 5 presents threats to validity; and Section 6 discusses the results obtained and future work.

2 Methods

To design the artifact (the model), we followed the DSR approach [8]. This approach, generally applied in engineering and computer science fields, is used to build artifacts to provide useful and effective solutions to a problem in a given domain. These are the steps we followed to construct our artifact:

- **Step 1 – Identify research problem:** The lack of a web API usability model. This problem is detailed further in Section 1.
- **Step 2 – Define objectives of a solution:** Build a web API usability model (the artifact).
- **Step 3 – Design and development of the artifact:** Apply the GQM approach to construct the model, defining the conceptual (Goals), operational (Questions) and quantitative (Metrics) levels.
- **Step 4 – Demonstration and evaluation of the artifact:** The elements (metrics) of the model were evaluated through a survey directed to web API consumers and developers.
- **Step 5 – Communication:** Communication of results through this document.

Now, we will detail the third step, which consisted in the design and development of the model. We performed an iterative bottom-up process, consisting of the following activities:

1. **Collection of initial metrics:** The main source for API usability metrics was papers coming from the literature review [6,9,10,11,12,13]. A total of 81 metrics were collected in this step.
2. **Mapping of the metrics:** Usability metrics proposed only for local APIs were mapped to web APIs when needed and possible. Note that not all the metrics were designed for or convertible to this type of APIs.
3. **Collection of additional metrics:** Because only 35.80 % (29 out of the 81) of the metrics found were deemed applicable to web APIs, a search for additional metrics was performed. Sources consulted included blogs, reports, and design guides published by recognized organizations on the web API industry [14,15,16,17,18,19], and some of the metrics were author proposed.
4. **Grouping metrics by usability attribute(s):** Some of the sources classified their metrics into usability attributes. When this was not the case, metrics were classified using Nielsen's usability attributes [20], depending on their theoretical definition.
5. **Defining goals and questions:** Usability attributes' definitions and aims were considered to establish the model's Goals and Questions (part of the GQM model).
6. **Specification of the metrics:** Metrics from the set were distributed among the Questions, considering each metric's definition and the problems targeted by the questions. Description, type (subjective/objective) and the way to calculate their value was detailed.

Fig. 1 summarizes these activities in a graphical manner, and it also shows future steps planned for the model development process.

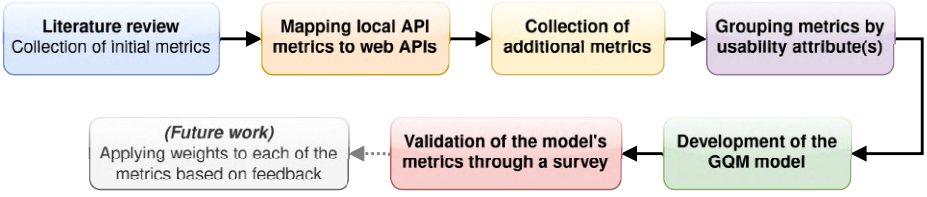

Fig. 1. Development process of the GQM model proposed.

2.1 The GQM approach

As we mentioned, the model presented in this article was built following the guidelines described by [7], in which the authors describe the GQM approach. According to the authors, "The result of the application of the Goal Question

Metric approach is the specification of a measurement system targeting a particular set of issues and a set of rules for the interpretation of the measurement data". They also explain that the model has three different, but related, levels:

- **Conceptual level (Goals):** Goals are defined for objects, which can be products (artifacts, deliverables, and documents that are produced during the system life cycle), processes (software related activities normally associated with time) and resources (items used by processes in order to produce their outputs).
- **Operational level (Questions):** Questions are used to characterize the way the achievement of a specific goal is going to be performed. Questions try to characterize the object of measurement with respect to a selected quality issue and to determine its quality from the selected viewpoint.
- **Quantitative level (Metrics):** A set of data is associated with every question in order to answer it in a quantitative way. The data can be objective (if they depend only on the object that is being measured) and subjective (if they depend on both the object that is being measured and the viewpoint from which they are taken).

In their paper, they describe a GQM model as a hierarchical structure starting with a goal (specifying purpose of measurement, object to be measured, issue to be measured, and viewpoint from which the measure is taken). The goal is refined into several questions, that usually break down the issue into its major components. Each question is then refined into metrics, some of them objective, some of them subjective.

3 Results

In this section, we will extensively detail all the Goals, Questions, and Metrics of the model we propose. Fig. 2 presents all of these components hierarchically. Note that most of the model's components came from the literature review mentioned in Section 2.

3.1 Goals

Here, we will detail all the components of the GQM model that we built. The six goals were constructed based on the definition of the usability attributes: **G1** responds to *Learnability*, *Understandability* and *Knowability*; **G2** responds to *Efficiency*; **G3** to *Memorability* and *Knowability*; **G4** to *Errors*, **G5** to *Robustness* and **G6** to *Satisfaction*. Note that we grouped attributes that have a similar definition into the same Goal. Each of the mentioned attributes come from the findings of the mapping, many of them being Nielsen's attributes [20]. For context purposes, we will elaborate the attributes considered by our model:

- **Learnability:** How easy is it for users to accomplish basic tasks the first time they encounter the design? (in this case, the web API). Found in [20].

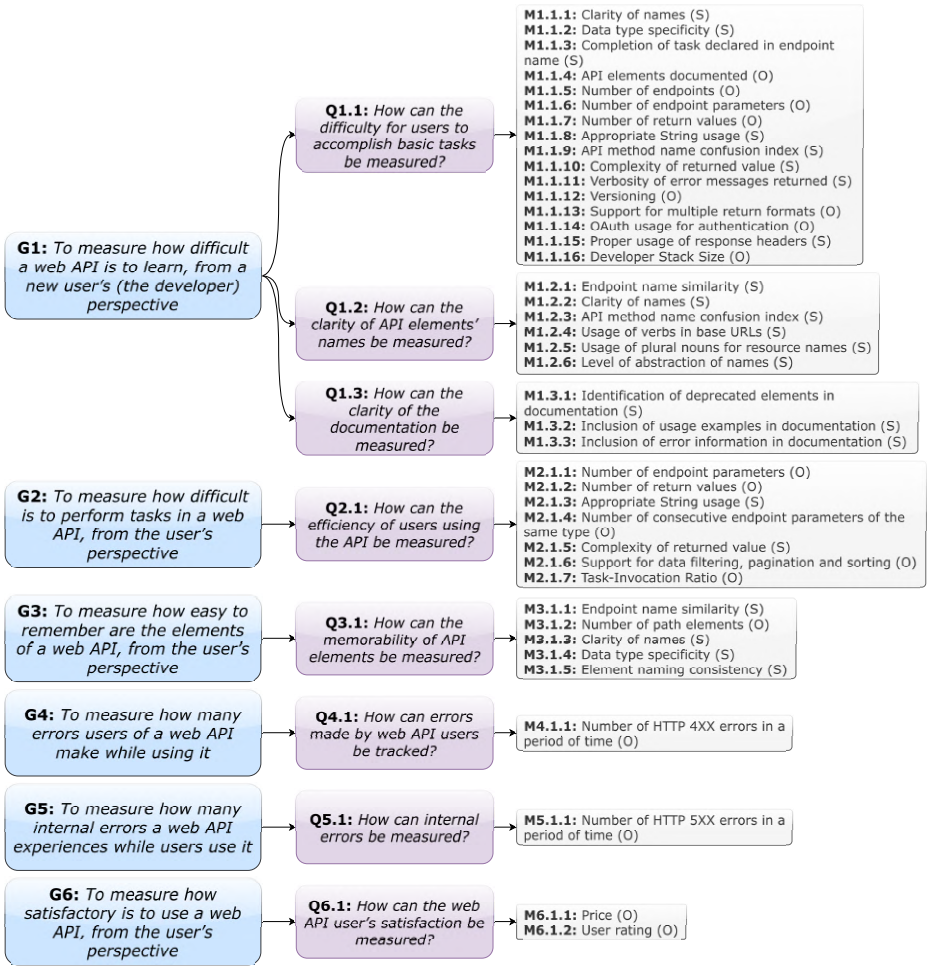

Fig. 2. The web API usability GQM model.

- **Understandability:** The effort required to understand the semantics of API features based on their names and documentation. Found in [21].
- **Knowability:** Implies the ease of understanding, learning and remembering the API. Among others, this attribute is mainly related to the naming of the API elements. Found in [9].
- **Efficiency:** Once users have learned the design, how quickly can they perform tasks? Found in [20].
- **Memorability:** When users return to the design (in this case, the web API) after a period of not using it, how easily can they reestablish proficiency? Found in [20].
- **Errors:** How many errors do users make, how severe are these errors, and how easily can they recover from the errors? Found in [20].
- **Robustness:** The capacity of the system to resist error and adverse situations. Found in [6].
- **Satisfaction:** How pleasant is it to use the design (in this case, the web API)? Found in [20].

3.2 Questions

Each Question in the model was designed with the above described usability attributes in mind. To ensure no detail was overlooked, we thoroughly analyzed the theoretical definition of each of these attributes.

3.3 Metrics

We distributed the metrics from the set among the Questions, considering each metric's definition and the problems targeted by the questions. To define the type of each metric (objective or subjective), we looked at the measurement process that has to be performed in order to get a value for the metric. We thought: "Is the process of picking a value for this metric as easy as picking between black and white, or is it as complex as picking between shades of gray?" – With this question, we meant that objective metrics should be easy to calculate regardless of the possible values, since there is no subjectivity tied to them, and in contrast, that subjective metrics need some kind of human intervention (from who wants to perform the measurement) to determine their value. Tab. 1 details all metrics encompassed by our model. In the second column of Tab. 1 (Metric), there are a few metrics with a (P) following their name. These metrics were proposed by the authors and were designed considering the different elements that a web API has. In the third column (Type), **S** represents subjective metrics, while **O** represents objective metrics.
Now, we will proceed to add supplementary information for certain metrics:

- **API elements documented (M1.1.4):** We suggest the formula $\frac{\text{Documented elements}}{\text{Total elements}} \times 100$ to calculate the percentage of documented elements. The API provider should aim for a value of 100 %.

Table 1. List of metrics (quantitative level).

Code	Metric	Type	Description	Attributes	Measurement
M1.1.1 M1.2.2 M3.1.3	Clarity of names [6]	S	How self-explanatory are API elements' names.	Learnability Understandability Knowability Memorability	Observation
M1.1.2 M3.1.4	Data type specificity [6]	S	How specific are the data types used. They should be as specific as possible, to make code more readable.	Learnability Understandability Knowability Memorability	Observation
M1.1.3	Completion of task declared in endpoint name [6]	S	If an endpoint does what its name suggests. An endpoint should only perform tasks described by its name.	Learnability Understandability Knowability	Observation
M1.1.4	API elements documented [6]	O	Number or percentage of API elements documented.	Learnability Understandability Knowability	Observation or automated
M1.1.5	Number of endpoints [11, 19]	O	The number of endpoints the API has.	Learnability Understandability Knowability	Observation or automated
M1.1.6 M2.1.1	Number of endpoint parameters [11, 12, 13]	O	The number of parameters an endpoint has.	Learnability Understandability Knowability Efficiency	Observation or automated
M1.1.7 M2.1.2	Number of return values [11]	O	The number of values an endpoint returns.	Learnability Understandability Knowability Efficiency	Observation or automated
M1.1.8 M2.1.3	Appropriate String usage [11]	S	If Strings are used only when needed. Strings should not be used if a better type exists, since they are cumbersome and error-prone.	Learnability Understandability Knowability Efficiency	Observation
M1.1.9 M1.2.3	API method name confusion index [13]	S	Name similarity between endpoints with akin functionality (generally, new versions of the same endpoint).	Learnability Understandability Knowability	Observation
M1.1.10 M2.1.5	Complexity of returned value (P)	S	Complexity of the response (for example, the amount of elements of a JSON).	Learnability Understandability Knowability Efficiency	Observation
M1.1.11	Verbosity of error messages returned [18, 19]	S	Level of detail error messages returned.	Learnability Understandability Knowability	Observation
M1.1.12	Versioning [14, 17, 18, 19]	O	If an API uses versioning.	Learnability Understandability Knowability	Observation or automated
M1.1.13	Support for multiple return formats [14, 17, 19]	O	If an API supports more than one return format.	Learnability Understandability Knowability	Observation or automated
M1.1.14	OAuth usage for authentication [19]	O	If an API uses OAuth for authentication. OAuth is a recognized standard, which can facilitate user experience.	Learnability Understandability Knowability	Observation
M1.1.15	Proper usage of response headers [14]	S	Whether response headers are properly defined or not.	Learnability Understandability Knowability	Observation
M1.1.16	Developer Stack Size [15]	O	Number of additional tools and libraries a developer need to install in order to use the API.	Learnability Understandability Knowability	Observation
M1.2.1 M3.1.1	Endpoint name similarity [9]	S	If an API has endpoints with similar names but different functionality.	Learnability Understandability Knowability Memorability	Observation
M1.2.4	Usage of verbs in base URLs [18, 19]	S	If verbs are used in base URLs (e.g., having an endpoint called "/createPerson").	Learnability Understandability Knowability	Observation
M1.2.5	Usage of plural nouns for resource names [14, 17, 18, 19]	S	If plural nouns are used for naming resources (where it makes sense).	Learnability Understandability Knowability	Observation
M1.2.6	Level of abstraction of names [19]	S	Level of specificity of resource names (e.g., naming a resource "videos" instead of "items", where "items" is the more general form).	Learnability Understandability Knowability	Observation
M1.3.1	Identification of deprecated elements in documentation [6]	S	If information about deprecated elements is included in the documentation.	Learnability Understandability Knowability	Observation
M1.3.2	Inclusion of usage examples in documentation [6]	S	If usage examples/tutorials are included in the documentation.	Learnability Understandability Knowability	Observation
M1.3.3	Inclusion of error information in documentation [6]	S	If information about possible errors is included in the documentation.	Learnability Understandability Knowability	Observation
M2.1.4	Number of consecutive endpoint parameters of the same type [11]	O	If an endpoint requires, consecutively, multiple parameters of the same type.	Efficiency	Observation or automated
M2.1.6	Support for data filtering, pagination and sorting [14, 15, 16, 17, 19]	O	If it is possible to filter, paginate and sort information returned by the web API.	Efficiency	Observation
M2.1.7	Task-Invocation Ratio [15]	O	Number of calls to the web API required to complete a task.	Efficiency	Observation
M3.1.2	Number of path elements [9, 19]	O	Path depth (for example, */analytics/events/aggregate/ID*, has three elements).	Memorability Knowability	Observation or automated
M3.1.5	Element naming consistency [6]	S	If some kind of pattern is followed for naming elements.	Memorability Knowability	Observation
M4.1.1	Number of HTTP 4XX errors in a period of time (P)	O	Number of HTTP errors from the 4XX group happening in a certain period of time.	Errors	Observation or automated
M5.1.1	Number of HTTP 5XX errors in a period of time (P)	O	Number of HTTP errors from the 5XX group happening in a certain period of time.	Robustness	Observation or automated
M6.1.1	Price [6, 10]	O	Cost to use the API.	Satisfaction	Observation
M6.1.2	User rating [10]	O	Rating given to the API by its users (if available).	Satisfaction	Observation

- **Number of endpoints (M1.1.5):** The number of endpoints could be counted automatically. As for a suggested value, [19] points out that a "manageable" value for this metric should be between 12 and 24, so less than 24 endpoints is acceptable.
- **Number of endpoint parameters (M1.1.6/M2.1.1):** We suggest the formula $\frac{\sum_{i=1}^{n} \text{Number of parameters}}{\text{Number of endpoint verbs}}$, where n is the number of endpoint verbs. Considering the suggestions in [11,13], the number of parameters should be, at most, 4 or 5. This metric can be used for each endpoint verb, or to get an average.
- **Number of return values (M1.1.7/M2.1.2):** We suggest the formula $\frac{\sum_{i=1}^{n} \text{Number of return values}}{\text{Number of endpoint verbs that return values}}$, where n is the number of endpoint verbs that return values. This metric can be used for each endpoint verb that returns, at least, one value, or to get an average.
- **Versioning (M1.1.12):** The value for this metric could be determined automatically simply by analyzing the web API's URL.
- **Support for multiple return formats (M1.1.13):** The value for this metric could be automatically computed by analyzing the web API's specification, if available.
- **Number of consecutive endpoint parameters of the same type (M2.1.4):** The value for this metric could be automatically computed using a counter to detect type streaks.
- **Number of path elements (M3.1.2):** The value for this metric could be automatically computed by counting the number of elements that an individual path has, which determine its depth. [19] suggests a value of 2 elements per resource.
- **Number of HTTP 4XX errors in a period of time (M4.1.1) and Number of HTTP 5XX errors in a period of time (M5.1.1):** Similar to [9], values for these metrics could be automatically computed using logs.

4 Evaluation

Initial model evaluation was done through a survey designed for web API consumers and developers. Respondents provided their opinion (in a scale from 1 to 5) on the influence of metrics encompassed by our model over web API usability. This survey, designed according to the guidelines in [24], was available between the 27th of September 2023 and the 1st of December of the same year. A total of 36 people from different countries, with varying levels of experience, responded to the survey. Most of the participants (94.6 %) are from Spanish-speaking countries, identify as expert software developers (63.9 %) and have between 5 and 10 years of experience with web APIs (38.9 %). Also, the majority of respondents are back-end developers (80.6 %), followed by these who work with databases (63.9 %) and front-end developers (55.6 %). In Fig. 3 and Fig. 4 we present, in a summarized way, the answers given by the respondents.

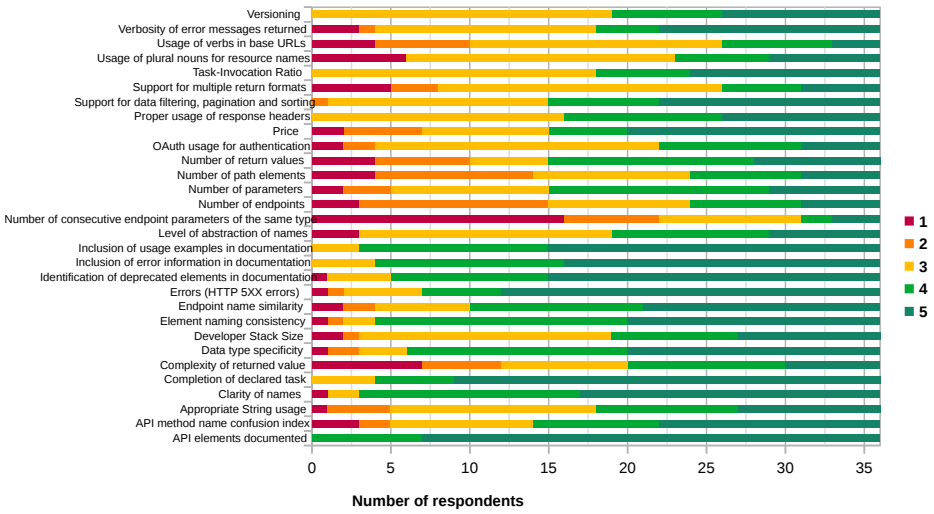

Fig. 3. Distribution of opinions given by respondents.

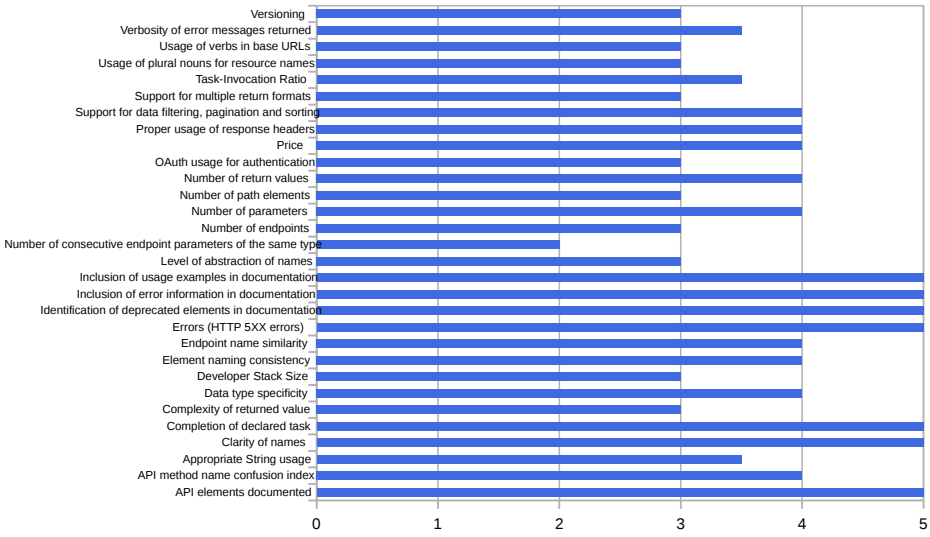

Fig. 4. Median opinions given by respondents.

5 Threats to validity and limitations

In the DSR approach, validity centers on whether the artifact solves the problem it was designed to address. [8] dedicates a step of this approach to artifact evaluation, which in this study consisted of a survey directed to web API professionals. Surveys are a reliable method to evaluate and validate DSR artifacts, as proposed by [22].
Additionally, in [23] two types of validity for DSR are presented: Summative and formative. Summative validity refers to the empirical evaluation of the artifact, while formative validity refers to correctly following an accepted procedure.
Regarding summative validity, we performed a survey on the importance of the elements belonging to the quantitative level of the model (the metrics), following the guidelines in [24]. In this sense, it remains to validate the full model, specifically the operational and conceptual levels. Regarding formative validity, as explained in Section 2.1, the model was built following the GQM approach. This approach, widely known and tested in the field of software engineering and other disciplines, has a layered structure that follows the most classical theory of usability for software products, proposed by Nielsen [5].

6 Discussion and conclusions

In this work, we have presented a preliminary model based on the GQM approach for the evaluation of web API usability, defining its three corresponding levels: conceptual (Goals), operational (Questions), and quantitative (Metrics). Because the majority of the metrics collected were not proposed in the context of a usability model (only [6] did, however, their work focuses on local APIs), and many of them were not linked to any usability attributes (only [6] and [9] linked their metrics to usability attributes), the main contribution of this work is the integration and organization of a wide range of metrics around usability attributes, in a model. With a set of six goals, eight questions, and 32 metrics, we consider that our model could help better understand the key aspects of web API usability assessment. The set of metrics, not definitive in this instance, is composed of 16 objective metrics and 16 subjective metrics. This fifty-fifty distribution was not intentional, as whether a metric is subjective or objective depends on its theoretical definition, and it indicates that using the model will require a variety of methods for collecting and evaluating values for these indicators.
The results of the survey presented in Section 4 indicate that except for the number of consecutive parameters of the same type an endpoint has, all metrics are meaningful since respondents have provided them with a median value of influence over web API usability of 3 or more. Particularly, metrics with a median value of 4 or 5 are of most importance: API elements documented (M1.1.4); Clarity of names (M1.1.1/M1.2.2/M3.1.3); Completion of task declared in endpoint name (M1.1.3); Errors (Number of HTTP 5XX errors – M5.1.1); Identification of deprecated elements in documentation (M1.3.1); Inclusion of error information in documentation (M1.3.3); Inclusion of usage examples in documentation

(M1.3.2); API method name confusion index (M1.1.9/M1.2.3); Data type specificity (M1.1.2/M3.1.4); Element naming consistency (M3.1.5); Endpoint name similarity (M1.2.1/M3.1.1); Number of endpoint parameters (M1.1.6/M2.1.1); Number of return values (M1.1.7/M2.1.2); Price (M6.1.1); Proper usage of response headers (M1.1.15); and Support for data filtering, pagination and sorting (M2.1.6). All of these metrics can be seen in Fig. 4. It is also important to point out that, even though *Number of consecutive endpoint parameters of the same type* (M2.1.4) received a median value of 2 out of 5, it is considered a significant usability factor by [11] and [13].

Regarding usability attributes, as seen in Fig. 2 and Tab. 1, the majority of the collected metrics belong to **G1** (To measure how difficult a web API is to learn, from a new user's perspective), more specifically, to **Q1.1** (How can the difficulty for users to accomplish basic tasks be measured?). This suggests that *Learnability*, *Understandability*, and *Knowability* are central to web API usability. As [4] stated, "Any good API should be easy to learn and use". **G2** (To measure how difficult is to perform tasks in a web API, from the user's perspective) and **G3** (To measure how easy to remember are the elements of a web API, from the user's perspective; both having a single Question) also have a sizable set of metrics, which suggests that *Efficiency* and *Memorability* are also important usability factors for web APIs. In contrast, **G6**, **G4**, and **G5** (To measure how satisfactory is to use a web API, from the user's perspective; To measure how many errors users of a web API make while using it; and To measure how many internal errors a web API experiences while users use it respectively; again, each with only one Question) have the least metrics, having **G4** and **G5** a single metric, which were proposed by the authors of this article. This suggests either that *Errors*, *Robustness*, and *Satisfaction* are not fundamental factors for web API usability, or that they are difficult to measure (this might be because of a variety of reasons).

In future investigations, we plan to (1) add weights to the metrics of the model, based on the feedback obtained from the survey, (2) contact survey respondents who agreed to be interviewed to gather additional information that might help improve the model, and (3) further validate the model with a group of experts in the field of web APIs. We also plan on developing a catalog of metrics organized by web API component, as a valuable contribution for web API consumers and developers.

References

1. De, B.: API Management. Apress, Berkeley, CA. (2017). https://doi.org/10.1007/978-1-4842-1305-6
2. Raemaekers, S., van Deursen, A., Visser, J.: Measuring software library stability through historical version analysis. 2012 28th IEEE International Conference on Software Maintenance (ICSM). (2012). https://doi.org/10.1109/ICSM.2012.6405296

3. Tan, W., Fan, Y., Ghoneim, A., Hossain, M.A., Dustdar, S.: From the Service-Oriented Architecture to the Web API Economy. IEEE Internet Computing. 20, 64–68. (2016). https://doi.org/10.1109/MIC.2016.74
4. Stylos, J., Myers, B.: Mapping the Space of API Design Decisions. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007). (2007). https://doi.org/10.1109/VLHCC.2007.44
5. Nielsen, J.: The usability engineering life cycle. Computer. 25, 12–22. (1992). https://doi.org/10.1109/2.121503
6. Mosqueira-Rey, E., Alonso-Ríos, D., Moret-Bonillo, V., Fernández-Varela, I., Álvarez-Estévez, D.: A systematic approach to API usability: Taxonomy-derived criteria and a case study. Information and Software Technology. 97, 46–63. (2018). https://doi.org/10.1016/j.infsof.2017.12.010
7. Basili, V. R., Caldiera, G., Rombach, H. D.: The Goal Question Metric approach. Encyclopedia of software engineering, 528-532. (1994).
8. Peffers, K., Tuunanen, T., Rothenberger, M.A., Chatterjee, S.: A Design Science Research Methodology for Information Systems Research. Journal of Management Information Systems. 24, 45–77. (2007). https://doi.org/10.2753/MIS0742-1222240302
9. Koçi, R., Franch, X., Jovanovic, P., Abello, A.: A Data-Driven Approach to Measure the Usability of Web APIs. 2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). (2020). https://doi.org/10.1109/SEAA51224.2020.00021
10. Ma, S.-P., Lan, C.-W., Ho, C.-T., Ye, J.-H.: QoS-Aware Selection of Web APIs Based on ϵ-Pareto Genetic Algorithm. 2016 International Computer Symposium (ICS). (2016). https://doi.org/10.1109/ICS.2016.0122
11. Scheller, T., Kühn, E.: Automated measurement of API usability: The API Concepts Framework. Information and Software Technology. 61, 145–162 (2015). https://doi.org/10.1016/j.infsof.2015.01.009
12. de Souza, C.R., Bentolila, D.L.: Automatic evaluation of API usability using complexity metrics and visualizations. 2009 31st International Conference on Software Engineering - Companion Volume. (2009). https://doi.org/10.1109/ICSE-COMPANION.2009.5071006
13. Rama, G.M., Kak, A.: Some structural measures of API usability. Software: Practice and Experience. 45, 75–110. (2013). https://doi.org/10.1002/spe.2215
14. Four Principles for Designing Effective APIs, https://www.mulesoft.com/api-university/four-principles-designing-effective-apis.
15. Riggins, J.: Why API Developer Experience Matters More Than Ever, https://nordicapis.com/why-api-developer-experience-matters-more-than-ever.
16. Levin, G.: Top 5 REST API Design Problems, https://blog.restcase.com/top-5-rest-api-design-problems.
17. Au-Yeung, J., Donovan, R.: Best practices for REST API design, https://stackoverflow.blog/2020/03/02/best-practices-for-rest-api-design.
18. Hämäläinen, O.: API-First Design with Modern Tools, https://www.theseus.fi/bitstream/handle/10024/226493/Hamalainen_Oona.pdf.
19. Mulloy, B.: Web API Design. (2012).
20. Nielsen, J.: Usability 101: Introduction to Usability, https://www.nngroup.com/articles/usability-101-introduction-to-usability.
21. Piccioni, M., Furia, C.A., Meyer, B.: An Empirical Study of API Usability. 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement. (2013). https://doi.org/10.1109/ESEM.2013.14

22. Cleven, A., Gubler, P., Hüner, K.M.: Design alternatives for the evaluation of Design Science Research artifacts. Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology - DESRIST '09. (2009). https://doi.org/10.1145/1555619.1555645
23. Lee, A.S., Hubona, G.S.: A Scientific Basis for Rigor in Information Systems Research. MIS Quarterly. 33, 237. (2009). https://doi.org/10.2307/20650291
24. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer Berlin, Heidelberg. (2012). https://doi.org/10.1007/978-3-642-29044-2