What is a Good API? A Survey on the Use and Design of Application Programming Interfaces

Natalie Kiesler(✉) and Daniel Schiffner

DIPF Leibniz Institute for Research and Information in Education, Frankfurt am Main, Germany
{kiesler,schiffner}@dipf.de

Abstract. In the Internet of Everything context, relevant and valuable connections between people, data, processes, and things are core elements. Machine-to-Machine (M2M) connections can be achieved, for example, by using application programming interfaces (APIs). However, investigating quality criteria of APIs has not yet gained significant traction in computing, or computing education research. In this work, we present a study of quality criteria for APIs to identify which factors developers rate as important when it comes to an API's quality. We then discuss how we can possibly quantify these factors. To achieve these goals, an online survey with experienced developers was conducted (n = 19). The results reveal that developers seem to appreciate established, stable solutions, the availability of examples, and a community. Developers also consider it important that an API can be extended, and integrated into an existing framework. However, strong trends among the criteria were not identified, as many factors play a role when choosing an API. Therefore more research is required to provide guidance to (future) software developers on how they can improve the design of their APIs.

Keywords: Internet of Everything · machine-to-machine · M2M · Application Programming Interface · API use · API design · API criteria

1 Introduction

The Internet of Everything (IoE) can be defined "as a distributed network of connections between people, smart things, processes, and data, whereas these components interact and exchange real-time data." [4] Valuable connections between these four pillars include people-to-people (P2P), machine-to-machine (M2M), and people-to-machine (P2M) systems.

To create a connection between products or services, especially in the M2M context, application programming interfaces (APIs) are the common basis for this exchange, enabling that products and services can communicate. APIs also

T. Pereira et al. (Eds.): IOECON 2023, LNICST 551, pp. 45–55, 2024.
https://doi.org/10.1007/978-3-031-51572-9_4

allow programmers to create new services and provide necessary insights to interact with existing methods. They can help ease a service's administration, its use, make it more flexible, and serve as a basis for innovation. Thus, APIs are a crucial technology within the context of the IoE, where connections between people, data, processes, and things are a core element.

Yet there is surprisingly little work on how to design usable APIs and how API designers may evaluate their usability prior to publishing them. When it comes to a definition of a "good" API, no clear criteria seem to exist [12]. Even though some best practice examples are identified in the literature [11, 14], and challenges developers encounter while working with an API have been investigated [9], a concrete, hands-on characterization of a high-quality API still requires research. In fact, there is little work on the design of usable APIs and how developers can evaluate their usability [3,13]. Some even resume that "Due to its nature, classification of API research is daunting for researchers" [14].

In the present work, we begin with an evaluation of good practices and how developers actually consume APIs developed by others. We, therefore, designed an online questionnaire to answer the following research question:

– RQ: *What are important factors for developers when it comes to an API's quality?*

The **goal** of this work is to identify whether and how we can classify the quality of APIs by means of quality criteria and other factors important to developers using APIs. We will then discuss how we can possibly quantify these factors. These insights, in turn, will potentially lead to better APIs and guidance for developers to increase the usability of existing and new APIs and services. The results further have implications for computing education and how to prepare software engineers for the IoE.

The structure of this paper is as follows. Related research on APIs, best practices, and challenges are presented in the following section. Section 3 outlines the research design before the results of the conducted survey are introduced in Sect. 4. Their discussion in Sect. 5 is followed by conclusions and future work.

2 Background

We reviewed several related studies and technical guidelines that define either the maturity of an API or focus on usability for developers. Looking at technical guidelines, one can find the CESSDA Technical Guidelines [1] which demand extensive resources, and thus time and effort to achieve high-quality software. For design purposes, requirement analysis is one of the key factors that is mentioned throughout the literature, as pointed out by [15]. In a technical report, Charest and Rogers [2] derive a list of methodologies when just focusing on data exchange alone. They explain the benefits and drawbacks when utilizing these methodologies as part of an API.

Meng et al. [11] use a different approach, identifying the topic from a learning perspective and how to structure documentation most effectively. They conducted two studies, one interview, and one questionnaire. They found evidence of two personas looking at APIs, who have different learning strategies. They

claim that documentation is an important asset and requires more than just expertise from software developers, but also information design and communication professionals. An interesting note is that the majority of participants in the studies (106 out of 110) voted for the integration of code examples into API documentation.

Maalej and Robillard [9] evaluated the content of Java SDK and .NET API references in 2013. At this point in time, the second largest contributor was represented by information units classified as "non-information", often found in methods and fields. The authors define the so-called "non-information" as "a section of documentation containing any complete sentence or self-contained fragment of text that provides only uninformative boilerplate text" [9]. Thus, the sole documentation of all fields and methods is not improving quality. Instead, it actually hinders readability and hence can reduce the perceived quality of an API's documentation.

Ofoeda et al. [14] provide an overview of API research using a socio-technical perspective. They argue that this additional perspective is important as developers, i.e. users of an API, are the target audience. In their study, they identify a number of research topics in the context of API design, whereas API development, documentation, usage, and security are predominant. They also state that: "Nonetheless, there remains a clear gap in the adoption and use of APIs since peer-reviewed articles are lacking." [14] They conclude, among other things, that a theoretical understanding of APIs is required and further insights into APIs are needed. The authors further conclude that a poor API design and bad implementation can lead to significant challenges for companies [14].

According to Murphy et al., "prior research has shown that APIs commonly suffer from significant usability problems" and "little attention has been given to studying how APIs are designed and created in the first place" [12]. Murphy et al. further resume about best practices, such as following API design guidelines, which might be locally defined or adapted from other companies. An API should be defined by an interdisciplinary team, while a designer is responsible for taking care of all perspectives. If the goal is to create a public API, i.e. customers or users are relying on it in their own code, the burden is even higher: "To avoid "breaking changes", it is important that the API designers get core abstractions and core methods of the API correct the first time". [12] To ensure this, Murphy et al. propose getting regular feedback from API users by using Human-Computer-Interaction methods, such as user studies or personas. However, the challenge of how to generate a good and reliable design remains, as only some additional hints on how to improve the quality are provided.

In the context of computing education and research, it seems as if there is still a lack of conscientiousness for software and its contribution to the epistemological process in general [8]. The development of tools as a research area is described as challenging "both for designing and reporting research" [10]. However, we need to invest more effort into the investigation and reporting of software (including, e.g., APIs) as part of the research process [5,6]. This is especially true in the IoE context, where improving the quality of an API may have a tremendous impact on future services, data, things, and eventually people [7].

3 Research Design

The present work aims at the identification of factors developers rate as important when it comes to the quality of APIs. To achieve this and better understand the developer's perspective, the authors conducted an online survey focusing on the use of APIs. It contained open and closed questions, clustered within three sections:

1. General questions and prior experience (3 open and 6 closed questions)
2. Criteria of good APIs (2 closed questions)
3. Criteria for using an API (5 open 2 and closed questions)

The full list of questions is summarized in the Appendix. Answering all questions was expected to last between 15 and 20 min. Participation in the survey was voluntary. Anyone could withdraw from the participation at any time without giving any reasons and without experiencing any disadvantages. No personal information was collected, and the data cannot be attributed to an individual. Study participants further received information on their legal rights granted by the General Data Protection Regulation. Moreover, the study was approved by the ethics board of the authors' institution.

The survey was distributed among teams containing software developers and computing researchers at several German research institutes within the Leibniz community via e-mail and internal chats. Due to this interdisciplinary, international work environment with English as *lingua franca*, the survey was made available in both German and English. It was online for four weeks, with one reminder message being sent to the addressed target audience after two weeks.

4 Results

In this section, we summarize the results beginning with the general questions and prior experience (see Appendix, Section I) to characterize the sample. The main sections (II and III) of the survey are summarized in alignment with the research question stated before. We thus point out important API characteristics from the developer's perspective.

4.1 Characteristics of the Sample

In total, 19 responses have been collected that almost completely answered the questionnaire. The majority of the 19 respondents is either a software developer (n = 8), a researcher in the context of computer science (n = 8), or employed in a coordinating or leading position (n = 3). Most respondents are experienced programmers, with 12 of them having more than 10 years of programming experience, two have 6 to 10 years of experience, four have 3 to 5 years of experience, and one person has 0 to 2 years of experience.

When asked about their prior experience in using APIs in years, seven persons replied as having more than 5 years, four have 4 to 5 years, four others have 2

to 3 years, and yet another four have 0 to 1 year of experience. We also gathered details from respondents on previous experience with using APIs from other developers. Five persons have used more than 10, seven have used 5 to 10, and the remaining seven have used 1 to 5 APIs from other developers. Hence, we can conclude all respondents have experience with using APIs.

For the question of how they learned using APIs, candidates could select multiple answers. Here, 16 of 19 respondents stated that they developed their competencies related to APIs via informal education (e.g., own research, literature, etc.). As this was a multiple choice question, 11 replies indicate it was due to professional practice, eight due to higher education, and seven due to further training (on-the-job). Three responses provided vocational training as educational background.

The survey further asked for the types of APIs respondents are using. In this multiple-choice question, open/public APIs were selected 16 times, and internal APIs 13 times. Both, external APIs and authentification APIs were each selected by 12 respondents. When asked for the context of their API use, a variety of topics was provided, such as education (n = 7), research (n = 5), Natural Language Processing (n = 5), research data center (n = 4), repositories (n = 3), library services (n = 2), as well as information gathering and management (n = 2). Other contexts mentioned by the participants were social media (n = 1), knowledge graphs (n = 1), identity management (n = 1), tools (n = 1), and infrastructure in general (n = 1).

We further asked for the programming language preferably used for working with APIs (as an open question), and received the following replies: Python (n = 10), Java (n = 7), Javascript (n = 6), C# (n = 2), as well as PHP (n = 1) and Perl (n = 1). Interestingly, data formats (JSON, n = 1, and XML, n = 1) were among the replies. Respondents mostly use APIs for the backend (n = 17), but also for frontend development (n = 13), whereas 11 persons use APIs for both areas.

4.2 Important Factors for Developers

The second and third section of the survey (see Appendix) focused on crucial API criteria from a developer's perspective. The responses will be presented next to answer the research question.

As part of that, survey participants were asked to rate the importance of several API features, assuming they are having a choice of which API to use. We then offered five answer pairs on a bipolar, seven-point Likert scale. The five answer pairs containing opposites comprise:

(a) Stability vs. Up-to-dateness
(b) Support offers vs. Extendability
(c) Integration in an existing framework vs. Required duration for getting started
(d) Documentation vs. Focus on solutions
(e) Established Solution vs. Innovative solution

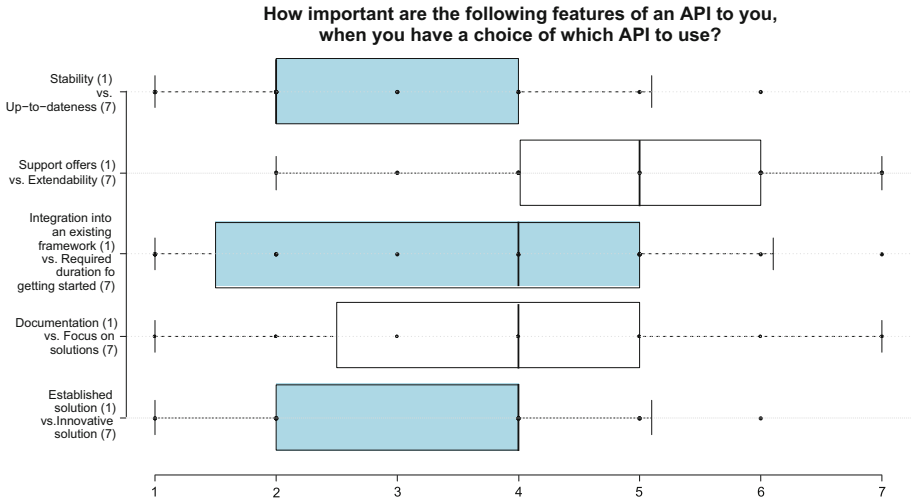

Fig. 1. Box plot of the responses to the question on feature preferences of APIs (n = 19).

The responses to these five answer pairs are summarized in Fig. 1. They indicate a median of 2 leaning towards the stability of an API (Mean 2,7 and SD 1,4) as an important feature. The surveyed developers favored the extendability of an API over available support offers (Mean 4,8 and SD 1,6). The integration into an existing framework seems more important than the required duration it takes to get started (Mean 3,4 and SD 1,9). The median, however, is 4. The same median applies to the remaining two answer pairs (d) and (e). Rating the importance of available documentation versus a focus on solutions also resulted in a mean of 4, and a standard deviation of 1,8 indicating that developers are undecided. For the last answer pair, the mean is 3,2 (SD 1,5), so developers seem to favor established solutions somewhat more than innovative ones.

The next question focused on important factors when actually using an API. Figure 2 represents how the respondents rated eight factors on a five-point unipolar Likert scale ranging from *absolutely essential* to *not important at all*. In addition, *I can't say* was offered as an answering option. All of the 19 surveyed persons replied to the question. The highest level of agreement (i.e., the addition of the ratings as *absolutely essential* and *very important*) were found related to the existence of examples of the API's use (n = 16), and a rigorous naming of functions, variables, and methods (n = 16). Similarly, an existing community (e.g., on GitHub, Stack Overflow, etc.) is rated as absolutely essential or very important by 14 respondents. The effort required for familiarizing one's self with how the API actually works (n = 12), as well as the sound embedding in the debugging process (n = 10) further play a crucial role for many API users. Nonetheless, it seems as if all of the surveyed factors do play a role when using an API, but with slightly varying weight.

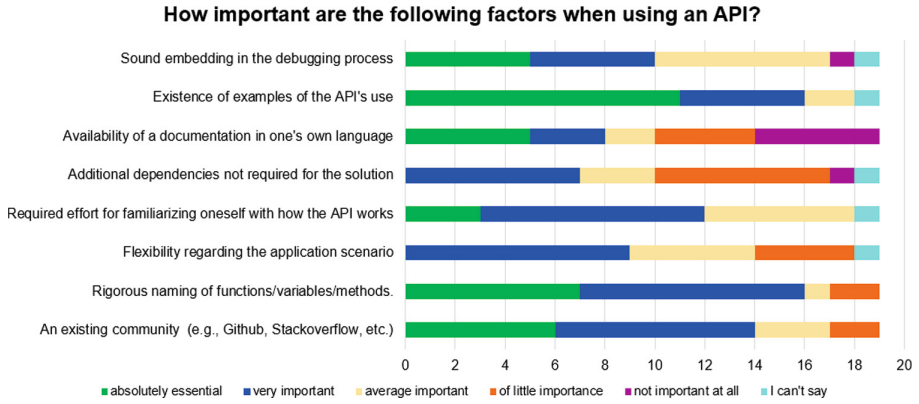

Fig. 2. Responses to the question on important features when using an API (n = 19).

Asking the developers about the importance of flexibility that usually comes with REST APIs, 14 participants answered that it is one of their driving factors to select them. Only four participants stated otherwise, while one did not answer the question. When explaining their selection, the developers stated in the negative case that usually no other option would be available, or that flexibility can cause security issues.

We further asked for the developers' own criteria as an open question. It contains, among other aspects, "ease of use", documentation, dependencies, or time needed for integration as important criteria. One developer expressed it in a very philosophical manner:

> In the best case, the API integrates well into the framework and stands on the shoulders of the giant instead of fighting it.

When we asked whether or not the developers have always adhered to their own criteria, the answers were almost evenly distributed between adhering to their rules (n = 9) and breaking them (n = 7).

The open question on important factors for the integration of APIs made the surveyed developers more conservative. Several participants responded with answers favoring a clear structure and modularization. However, no clear trend can be identified and seven persons did not even answer this question.

The final question set focused on the challenges when implementing APIs and how to overcome them. Among bad documentation (n = 6), and lack of availability if provided by a third party (n = 3), there were concerns regarding long-term maintenance and compatibility (n = 3). Seven participants did not answer the question. When asked how to overcome these issues, some participants (n = 8) responded with common terms among *Developer User Experience*, such as "easy to learn", "clear contracts" or "graceful version transition". One answer actually included the active use of test cases to learn the API:

> [...], I learn by examining the test cases how to utilize the API.

Some respondents were stating that the bigger picture needs to be involved and automatic tooling could help simplify some of the tasks. One participant fell out of this pattern, by mandating not to use third-party APIs.

5 Discussion

In this section, we briefly discuss some of the most interesting aspects of the survey responses, while considering possibilities to quantify the factors considered important by the developers.

One of the most repeated inputs was the community-building aspect. Having an (a)synchronous method for communication was considered the most helpful resource when using an API. Similarly, developers repeatedly referred to documentation, but well beyond the classical text document. Short examples on how to use the API together with concise code documentation were considered to be most helpful. One participant also stated that they like to utilize test cases to better understand an API. This underlines that the examples do not need to be artificial, but can utilize existing methods for code quality.

In the bipolar Likert scale, we asked for different trade-offs. Stability, extendability, and established solutions were rated as being more important. Given their long experience in programming, we assume the developers had their fair share of breaking APIs and incompatibilities. Support for them is not the key factor when selecting an API in the first place. This is also reflected in the answer to innovative solutions, which tend to be in a state of flux and thus change more often. The other two dimensions, however, have not been as clear. Integration was considered to be more relevant than the duration it takes to get started. This can be interpreted as the reuse of code and patterns being very important, especially in long-term maintenance projects. Documentation, on the other hand, was not a clear favorite, mainly due to the fact that the other dimension might not be as contradictory as initially thought. Good documentation might focus on the solution of an actual problem and hence contribute to the solution.

While not stated explicitly, a lot of the answers are aimed at the Developer User Experience (DevUX). “Ease of use”, “easy to learn” or “community” are elements supporting a better DevUX. As numerous APIs exist, sole functionality is not a deciding factor anymore. Developers tend to favor other factors over pure innovation or problem-solving. These factors should also be taken into account when “designing” an API.

Quantifying the results factors remains somewhat challenging. Measuring when an API is “easy to use” certainly requires more research before this factor can be evaluated. Criteria where a quantification seems reasonable comprise:

- the duration of time required for developers to familiarize themselves with the API,
- the availability of examples, and use cases,
- the API’s complexity,
- the extent to which an API’s documentation contains methods which are explained with reference to their objective,

– the deliverable of an API,
– an API's availability,
– an API's security.

Some of these criteria have also been reported in prior work. Maalej and Robillard [9] have already criticized the level of "non-information" within API documentation. Therefore, explaining an API's method with regard to its objective seems to be a more promising quality criterion. Murphy et al. [12] have also discussed the impact of APIs on security. It thus seems reasonable to include this aspect as a quality criterion.

Limitations of the present work concern the small sample size. With 19 respondents from several German research institutes, the survey is limited to one country and the context of educational technology, and infrastructure research. Moreover, the nature of the survey was somewhat exploratory due to the lack of a great body of prior research. Hence, it is not yet possible to define strict, or clear-cut criteria for APIs.

6 Conclusion and Future Work

Application Programming Interfaces (APIs) are crucial to achieve machine-to-machine connections within the Internet of Everything. In this study, we explored and identified important criteria for developers when using APIs, and how we can possibly quantify them. The starting point of this research was the lack of both research and common approaches toward a good API's design, usability, or other criteria. We, therefore, conducted a study with software developers and computing researchers who have experience in using APIs. The results of the online survey revealed that developers value stable, established solutions with rigorous naming and a community or examples they can relate to when making the effort to familiarize themselves with the API. Similarly, the possibility to extend an API, and integrate it into an existing framework is appreciated, whereas the required effort itself plays an important role. In summary, all of the surveyed factors do play a role when using an API, while their degree of importance and related challenges slightly vary. Moreover, it is still challenging to quantify some of the criteria indicating an API's quality, and increasing its use by others. Nonetheless, the gathered insights can be a reference for educating future software engineers, who will likely design services and products for or in the context of the IoE.

As the results of this work have implications for research, practice, and educators, there are several pathways for future work. One of them is to continue our efforts to identify and quantify the quality criteria of APIs and conduct a follow-up study with developers. Another approach is to expand the scope of the survey to other institutions and industry or to qualitatively address the developer's perspective. The latter could lead to an increased understanding of the conditions for certain preferences and tendencies. Yet another future research approach is to analyze popular, publicly available APIs with regard to the criteria investigated in this study.

Appendix

Survey Questions

Section I - General Questions and Prior Experience

1. Current job title
2. Your programming experience in years
3. Your experience in using APIs in years
4. How many APIs from other developers have you used so far?
5. How did you develop your competencies related to APIs?
6. Which types of APIs are you using?
7. In which context are you using APIs (e.g., education, research data center, natural language processing, etc.)?
8. Which programming language(s) do you prefer to use for working with an API?
9. For which interfaces do you use APIs?

Section II - Criteria of Good APIs

1. Assume there are several APIs available to choose from. How important are the following features of an API to you, when you have a choice of which API to use?
2. How important are the following factors when using an API?

Section III - Criteria for Using an API

1. Open API architectures (such as REST) offer creative freedom in API design. Is this flexibility a factor that drives you to use them?
2. Please explain your selection regarding flexibility as a factor (see previous question).
3. Which criteria do you use to evaluate the complexity of integrating a new API into an existing system/framework?
4. Have you ever encountered cases where the criteria you mentioned regarding the complexity of an API did not play a role in your choice?
5. What factors in your system architecture make it easier for you to integrate existing APIs?
6. Which challenges (due to the API and the system) have you already experienced when integrating third-party APIs?
7. Do you have ideas on how to overcome these challenges?

References

1. CESSDA Technical Guidelines (2023). https://docs.tech.cessda.eu/
2. Charest, G., Rogers, M.: Data exchange mechanisms and considerations. https://enterprisearchitecture.harvard.edu/data-exchange-mechanisms

3. Horvath, A., Nagy, M., Voichick, F., Kery, M.B., Myers, B.A.: Methods for investigating mental models for learners of APIs. In: Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems, CHI EA 2019, pp. 1–6. ACM, New York (2019). https://doi.org/10.1145/3290607.3312897
4. Kiesler, N., Impagliazzo, J.: Perspectives on the internet of everything. In: Pereira, T., Impagliazzo, J., Santos, H. (eds.) IoECon 2022. LNICST, vol. 458, pp. 3–17. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-25222-8_1
5. Kiesler, N., Schiffner, D.: On the lack of recognition of software artifacts and IT infrastructure in educational technology research. In: Henning, P.A., Striewe, M., Wölfel, M. (eds.) 20. Fachtagung Bildungstechnologien (DELFI), pp. 201–206. Gesellschaft für Informatik e.V., Bonn (2022). https://doi.org/10.18420/delfi2022-034
6. Kiesler, N., Schiffner, D.: Why we need open data in computer science education research. In: Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education, ITiCSE 2023, vol. 1. Association for Computing Machinery, New York (2023). https://doi.org/10.1145/3587102.3588860
7. Kiesler, N., Thorbrügge, C.: Socially responsible programming in computing education and expectations in the profession. In: Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education, ITiCSE 2023, vol. 1, pp. 443–449. Association for Computing Machinery, New York (2023). https://doi.org/10.1145/3587102.3588839
8. Kinnunen, P., Meisalo, V., Malmi, L.: Have we missed something? Identifying missing types of research in computing education. In: Proceedings of the Sixth International Workshop on Computing Education Research, ICER 2010, pp. 13–22. ACM, New York (2010). https://doi.org/10.1145/1839594.1839598
9. Maalej, W., Robillard, M.: Patterns of knowledge in API reference documentation. IEEE Trans. Softw. Eng. **39**, 1264–1282 (2013). https://doi.org/10.1109/TSE.2013.12
10. Malmi, L.: Tools research-what is it? ACM Inroads **5**(3), 34–35 (2014). https://doi.org/10.1145/2655759.2655768
11. Meng, M., Steinhardt, S., Schubert, A.: Application programming interface documentation: what do software developers want? J. Tech. Writ. Commun. **48**(3), 295–330 (2018). https://doi.org/10.1177/0047281617721853
12. Murphy, L., Kery, M.B., Alliyu, O., Macvean, A., Myers, B.A.: API designers in the field: design practices and challenges for creating usable APIs. In: 2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pp. 249–258 (2018). https://doi.org/10.1109/VLHCC.2018.8506523
13. Myers, B.A., Stylos, J.: Improving API usability. Commun. ACM **59**(6), 62–69 (2016). https://doi.org/10.1145/2896587
14. Ofoeda, J., Boateng, R., Effah, J.: Application programming interface (API) research: a review of the past to inform the future. Int. J. Enterp. Inf. Syst. (IJEIS) **15**(3), 76–95 (2019)
15. Zowghi, D., Coulin, C.: Requirements elicitation: a survey of techniques, approaches, and tools. In: Aurum, A., Wohlin, C. (eds.) Engineering and Managing Software Requirements, pp. 19–46. Springer, Heidelberg (2005). https://doi.org/10.1007/3-540-28244-0_2