A systematic mapping study of API usability evaluation methods

A Systematic Mapping Study of API Usability Evaluation Methods

Irum Rauf[a,*], Elena Troubitsyna[b], Ivan Porres[c]

[a]*Centre for Research in Computing, The Open University, United Kingdom*
[b]*KTH – Royal Institute of Technology, Stockholm, Sweden*
[c]*Åbo Akademi University, Dept. of Information Technologies,Turku, Finland*

Abstract

An Application Programming Interface (API) provides a programmatic interface to a software component that is often offered publicly and may be used by programmers who are not the API's original designers. APIs play a key role in software reuse. By reusing high quality components and services, developers can increase their productivity and avoid costly defects. The usability of an API is a qualitative characteristic that evaluates how easy it is to use an API. Recent years have seen a considerable increase in research efforts aiming at evaluating the usability of APIs. An API usability evaluation can identify problem areas and provide recommendations for improving the API. In this systematic mapping study, we focus on 47 primary studies to identify the aim and the method of the API usability studies. We investigate which API usability factors are evaluated, at which phases of API development is the usability of API evaluated and what are the current limitations and open issues in API usability evaluation. We believe that the results of this literature review would be useful for both researchers and industry practitioners interested in investigating the usability of API and new API usability evaluation methods.

Keywords: API Usability, Usability Evaluation Methods, API Developers, Usability factors, Cognitive Dimensions

*Corresponding author
Email addresses: `irum.rauf@open.ac.uk` (Irum Rauf), `elenatro@kth.se` (Elena Troubitsyna), `ivan.porres@abo.fi` (Ivan Porres)

1. Introduction

An Application Programming Interface (API) provides a programmatic interface to a software component or service. APIs play a key role in software reuse: they allow programmers besides the original designers to use a certain component or service.

Although the concept of API is not new, there is currently a renewed interest in creating, publishing and consuming APIs as well as improving methods for carrying out these tasks. APIs have a prominent role in software engineering because they support three main tasks.

First, APIs enable software reuse by defining the interfaces of software components. This has traditionally been the main use of APIs. It supports the well-established concept of software modularity [70].

Second, APIs are also interfaces to software services offered over a network. They facilitates an interoperability of computing systems. In recent years, we have seen an exponential growth of the use of APIs to connect web applications and services over the Internet. Currently, many web applications use authentication or storage services from Google, Facebook or Dropbox thanks to their public APIs. Cloud providers such as Amazon Web Services (AWS) offer self-provision and management interfaces through an API. Additionally, mobile applications for smartphones connect and interoperate with web applications using APIs. Currently, there are more than 18,000 public web APIs[1].

Finally, APIs enable publication of open data. Many public organizations from all around the world are publishing vast amounts of data that can be accessed through the Internet using an API. As an example, the Finnish Ministry of Finance currently offers 1766 different datasets[2].

An API is a result of a design process performed by humans. *Designers* of an API are programmers who decide which features should be included in it, and how to use these features and document them. API *users* are also program-

[1]http://www.programmableweb.com/apis/directory
[2]http://www.avoindata.fi/

mers. They must study the API documentation and, in some cases, its source code, and use the API to create new programs or interconnect existing ones. APIs have many different quality attributes such as functionality, reliability, and availability [19]. However, the focus of this study is an often neglected but equally important characteristic of an API: its *usability*.

The usability of an API is a qualitative attribute that indicates us how easy it is, for developers, to use an API in a certain context and learn it. APIs with good usability can increase programmers' productivity and satisfaction [59]. Over the years, developers have realized the importance of creating usable APIs to attract and retain their users. However, once an API is published, it is difficult to change it since there can be many programmers who are using it already. Therefore, it becomes necessary to evaluate and review the usability of an API before it is offered publicly.

In this systematic mapping study, we summarize the body of knowledge created in the field of evaluating API usability. We evaluate and categorize different approaches, systematically analyze their features and discuss the future trends in the area of evaluating API usability. This work targets both researchers from academia and practitioners who are interested in evaluating the usability of APIs.

The paper is organized as follows. Section 2 discusses API usability and how it is defined in the literature. Section 3 presents our review protocol and the research questions. Section 4 presents answers to the research questions. Section 5 discusses previous systematic literature reviews. Section 6 and 7 discuss the challenges of existing works and conclude our findings correspondingly.

2. API Usability

APIs offer software services to wider audiences who can use the functionality offered by APIs in ways beyond the imaginations of the original developers. For example, API offered by Google Maps [1] is being used today by thousands of applications in a variety of unique and innovative ways. AirBnB [2] has effectively used Google Maps API to revolutionize the lodging market. The cab hiring companies like Uber [3], with the help of Google Map API, have completely changed the way cab hiring is done today. APIs that are developed with its users in mind and provide good usability encourage users to continue using them. On the other hand, users easily leave the APIs that they are not comfortable with [79].

Well-designed APIs encourage programmers to use them in productive and satisfying way in using them [39]. However, not all APIs are equally usable, and some APIs may not offer ease-of-use to its users as other APIs do. Similarly to the graphical user interfaces that are convenient to use, provide easy access to information and increase user's productivity by easing out the cognitive load, the API users also select and continue the use of an API that provides them easy access to information and improves their *time-to-develop*.

API users choose an API by reading its documentation and analysing how to use it by doing small tasks [4]. If API user does not perceive ease-of-use and satisfaction with it, they will either not choose it or discontinue its use after an initial trial [79]. For an API provider, loss of API users incurs a loss of revenue and reputation [54]. For an API user, a choice of a badly designed API results in an increase in time, efforts and money to build applications using such an API [54]. It thus becomes essential for API providers to evaluate the usability of their APIs before releasing it because once an API is released it becomes difficult to change it since there may be many developers using it.

Usability is a qualitative attribute of the design of a product. The importance of usability lies in the fact that it significantly contributes to the acceptance of the product by its users. However, since usability is a subjective term, it has been defined and used differently by different usability standards and researchers. For example, we can find three different definitions of usability in three different ISO standards. *ISO/IEC 9126 Software engineering (Product quality)* [45] defines usability as "a quality characteristic defined as the capability of the software product to be understood, learned, used and to be attractive to the user when used under specified conditions".

ISO/IEC 25010 defines usability as characteristic of having the following attributes: "learnability, flexibility, accessibility, and context conformity".

ISO 9241 [44] provides a relatively broad view on usability. It terms software usable "when it allows the user to execute his task effectively, efficiently and with satisfaction in the specified context of use".

Other usability experts have also proposed different usability models. The work by Nielsen and Nielsen [66] and Loranger [67] define usability as five quality components: "learnability, efficiency, memorability, errors and satisfaction". Abran et al. [5] took ISO 9241-11 as a baseline for their consolidated model and proposed extensions to it with the following two attributes: "learnability" and "security". The work of Quesenbery [75] defines "effectiveness, efficiency, engagement, error tolerance, and ease of learning" as quality attributes of a usable product.

Seffah et al. [88] proposes a consolidated model built upon the analysis of the usability measurement literature. This model includes ten usability factors: efficiency, effectiveness, productivity, satisfaction, learnability, trustfulness, usefulness, safety, accessibility and universality.

Figure 1 presents a word cloud formed from the key definitions of usability used often in the literature (given above). A word cloud provides a composition of words in which the size of the words shows the frequency of their occurrence. It demonstrates that usability can be referred to as a collection of different quality attributes with different experts giving one attribute preference over the other. Learnability, however, comes out to be the most common attribute selected by experts for defining usability, followed by satisfaction and efficiency.

Although there is a large body of knowledge that defines usability and provides methods to evaluate usability of information artefacts [53], the usability definitions, models and evaluation methods do not cover the unique nature of API and its usability. An API offers programmatic interfaces that exhibit information in a different manner than the interfaces of software designed for non-programmers, e.g., GUIs (Graphical User Interfaces). The concerns that

Figure 1: Word cloud formed from key usability definitions in literature

contribute to the usability of an API may not be applicable to that of a GUI. APIs offer non-graphical interfaces that provide additional cognitive load to its users compared to GUIs [36]. In addition, API users have different needs from an API than that from a GUI by a non-programmer. For example, programmers need to reuse codes, easily write new software applications using an API and quickly find errors when compiling their codes. These and other such characteristics of APIs make them different from traditional software with GUIs.

With the widespread adoption of APIs, there is a considerable increase in efforts to evaluate usability of APIs as traditional approaches do not fill in the gap adequately. It thus becomes necessary to conduct a systematic study to analyse how the usability of API is studied in the current literature, and identify the limitations of the existing work and define the directions of future work in the area of API usability evaluations.

3. Systematic Mapping Approach

We present a systematic mapping study of the existing API usability evaluation methods. We conduct systematic mapping study (SMS) following the rigorous protocols developed for such studies [71], [50]. An SMS analyses an evidence in a domain at a high level of granularity [50] with an aim to study research trends. The research questions in SMS sought answer to the general questions like a trend of publications over the time and topics studied in a specific research area [71]. On the other hand, systematic literature review (SLR) studies the research questions in detail with very specific goals, e.g., answering questions like whether a particular intervention is of practical use to the industry [71]. The details of differences between SMS and SLR can be found in [71] and [50].

We found SMS more appropriate to systematically study and analyse existing literature relevant to API usability evaluation methods, since our research objective for such a study was of a broad nature. Our research objective was to discover research trends over time by posing general research questions that would classify the primary studies in different categories like publication trend over time, topics covered in the literature, etc. [71].

Both SLR and SMS require a rigorous selection process which is transparent and exhaustive in order to identify as many as possible relevant research articles within the scope of the review. Our review protocol is based on the guidelines proposed by Kitchenham et al. [50]. The review protocol consists of three main phases for a systematic mapping study: *Planning, Conducting* and, *Reporting the Review*. The first phase starts with the review planning stating the motivation presented in Section 2 and formulating the research questions based on the motivation to do an SMS (presented in Section 3.1). The second phase of conducting the review consists of identifying and selecting the primary studies, assessing the quality of the studies, extracting data and synthesizing it (presented in Sections 3.2 - 3.5). The review is concluded by writing this review report based on the results of the study.

3.1. Research Questions

We have identified the following four research questions with an aim to provide a picture of the current research efforts in the area of API usability evaluation in the interests of researchers and practitioners:

- RQ 1:What are the different methods used to evaluate the usability of an API? A wide range of usability evaluation methods exists ([53] [38]). This question aims at identifying which usability evaluation methods are used in the existing literature to evaluate the usability of APIs.
- RQ 2: What is the aim of the existing research efforts on API usability evaluation? This research question identifies and categorizes the existing articles based on their research aims. Answering this research question will help us to analyze the state of the art in this area.
- RQ 3: In which phase of API development does the API usability evaluation method apply? The usability evaluation can be done at various stages of development. By answering this research question we can help researchers and API developers by directing them to appropriate studies that address usability evaluation at a particular phase and learn the applicability of different API usability evaluation methods at different development phases from them.
- RQ 4: What are the usability factors identified and evaluated by the existing studies? The term usability encompasses a rather broad set of concepts. Various researchers use different perspectives of usability to evaluate an API. With this research question, we study which usability factor is addressed by the researchers when evaluating the usability of an API.

3.2. Study Selection Strategy

The study selection strategy is critical to find and select complete and relevant literature. Therefore, this step has to be carefully designed and validated in order to provide rigor to the systematic review.

3.2.1. Identify Search Terms and Define Search String

We built our search string by 1) First identifying major keywords based on our research questions. The major keywords that we identified were: API, usability and evaluation. 2) As a second step, we identified the alternative keywords by finding the synonyms. 3) In the third step, we combined the major terms with the Boolean AND operator and the alternative keywords with the Boolean OR operator.

The search string was refined through the iterative process along with deepening our understanding of existing work during a search of different repositories. The final search string was:

```
{( (API OR framework) AND (usability OR usable) AND
   (evaluat* OR assess* OR measur* OR tool OR test* OR automat*))
< in abstract, keywords, and title > }
```

3.2.2. Identify Repositories

The search repositories that we selected were: IEEE Xplore, ACM Digital Library, Web of Science(WS), SpringerLink(SL) and ScienceDirect(SD).

We used IEEE Xplore and ACM Digital Library since they cover almost all important workshops, conferences and journal papers that are published by either IEEE or ACM. The other three repositories were selected since they show journal papers from leading software engineering journals. We also manually searched the conferences and journals publishing work on human-computer interaction and human-centric computing. We selected the following journals: *International Journal of Human-Computer Interaction*, *ACM Transactions on Computer Human Interaction* , *Advances in Human-Computer Interaction*, *IEEE Transactions on Human-Machine Systems*, *Interacting with Computers*, *International Journal of Human-Computer Interaction* and *Journal of Usability Studies*. We also manually searched the proceedings of *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC).* We selected these peer-reviewed venues based on their relevance to our topic. We also searched http://apiusability.org to find relevant research articles that we might have missed and searched profiles of researchers actively working on API usability.

3.2.3. Study Selection

Our study selection process is based on steps presented in Table 1. Initially, we searched electronic repositories using our search string. The found papers were filtered by reading their titles, keywords, and abstracts using our inclusion and exclusion criteria (presented later in this section). We then manually searched some of the journals and conferences mentioned above. From [99], we adapted the snowball strategy in our selection process. We adopted the last three steps (out of 5) of the snowball strategy since we had already selected an initial set of papers with automatic search and manual search. We adopted and applied the following three steps of the snowball strategy:

1. The initial set of papers consisting of 76 research articles was taken as an input to perform *backward snowballing.* The references of these articles were thoroughly looked at to identify any potential primary studies in addition to authors' own previous works that they may have conducted in this area. The results of this step were also used as an input in the following steps.
2. The selected studies became a subject to *forward snowballing* by looking at the papers that cited these selected studies. As recommended [99], we used Google scholar because it indexes a collection of databases.
3. If no new papers were found by iterating steps 1-2, we searched personal homepages of authors or databases of their research institutes and in some cases, some active authors in this area were contacted via email to identify any potential missing articles. If new papers were found, then we looped back to Step 1.

Step: 3

Table 1 shows summary of our selection process.

Table 1: Summary of Selection Process

	WS	IEEE	ACM	SL	SD	**Total**
Search results	801	73	164	4446	2041	**6487**
After reviewing titles/keywords	32	12	14	9	4	**70**
After manual search	34	13	15	10	4	**76**
Snowballing	38	17	21	12	4	**92**
After reading abstracts	32	15	15	9	3	**75**
After skimming/ reviewing	29	12	12	4	2	**59**
After Discussion-Final	21	10	11	3	2	**47**

3.2.4. Study Quality Assessment (SQA)

Assessing the quality of a research article is an important part of the selection process. We selected two criteria to classify papers based on their quality. Our first criterion was to select papers that were more than 5 pages in IEEE double-column format or more than 7 pages in LNCS single-column format. The second criterion was to select papers based on their research impact, i.e., the citation of the paper. This criterion is justified by the argument that the research articles that were cited less or not at all, after many years, did not have much research impact [65]. We adopted this approach and ranked papers for their quality by computing the difference between the Google Scholar(GS) citation number of the paper and the value obtained from the formula, *2n-2*, where *n* is the number of years since that paper had been published. This formula, taken from the work of Nguyen et al. [65], takes into consideration the fact that a paper may get cited less often in the first two years of its publication. Hence, such an approach becomes less restrictive on the study selection. The papers published in 2017 and 2018 are not evaluated based on this citation criteria as papers have fewer citations in the first two years. Thus, the formula adopted to calculate study quality assessment (SQA) score is calculated by subtracting (2n-2) from GS citation number, i.e. *GS citation number - (2n-2)*.

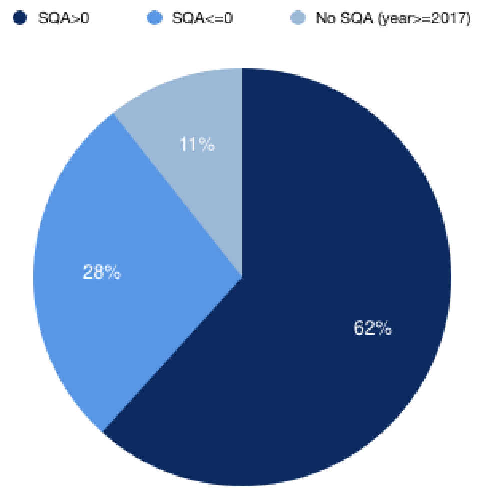

Figure 2: Study Quality Assessment for Primary Studies

Figure 2 shows that most of the primary studies have good research impact, although most of them are recent researches implying API usability evaluation is an active research area. The detailed tables are attached in Appendix A. Table A.2 and Table A.3 show the SQA assessment results of all the primary studies along with the comparison of the article length. We did not find any research article less than 7 pages long in LNCS single-column format so in Table A.2 and Table A.3 the column less than 5 pages suggest the articles only in IEEE double-column format.

A SQA greater than 0 implies that the article has a significant research impact over the years and an SQA score less than 0 implies the opposite. The classification serves to show significant API usability evaluation studies and less common or emerging API usability evaluation studies.

3.2.5. Inclusion Criteria

In our systematic literature review, we have included works published before and in the year 2018. In section 1, we have discussed what we infer from API usability. The usability of an API has a diverse definition and hence, we explicitly defined our inclusion and exclusion criteria as follows:

- the papers that explicitly discussed evaluating the usability of APIs.
- the papers that used both the words *API* and *Usability* as keywords for paper identification
- the papers that provided or used approaches that evaluate the usability of an API.

3.2.6. Exclusion Criteria

Initially, we excluded papers that were duplicates of the papers already found in the other repository/ repositories. We have, then, excluded the works according to the following criteria:

- the papers not written in English
- in case more than one paper addressed the same approach, we included the latest version of the approach that contained the complete description of the work. For example, we selected the extended journal version [87] over the shorter conference papers ([84], [86], [85])
- the papers that focused on how to improve API documentation in itself but did not provide information on how their study assesses the usability of the API. The detailed study of existing tools that are used for API documentation generation can be found at [68]
- the papers that provided a usage analysis of APIs to study usage trends, e.g., [82].
- the papers that studied the use of a program by its users in order to study non-functional properties other than usability, e.g., maintainability ([74],[51], [73])

3.3. Extract Data

A total of 47 primary studies were selected.

The papers from the final list of primary studies were read in full to understand the state of the art in evaluating API usability. At this phase, we collected

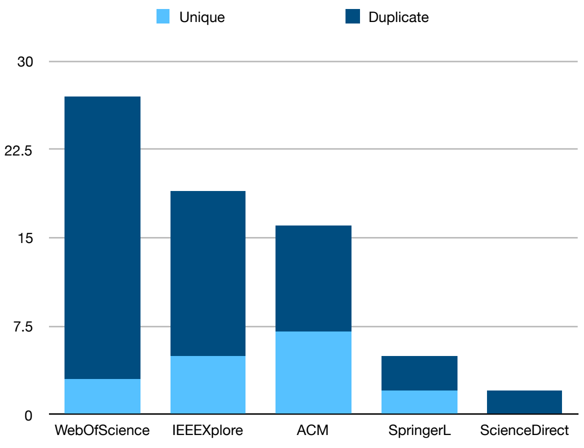

Figure 3: Selection from repositories

the relevant data from primary studies that would help us in answering our research questions.

The data we collected from each paper consisted of the following items: the year of publication, the number of citations, the method used, tool availability, the aims of usability evaluation, factors evaluated for usability, and the development phase at which the usability evaluation was done.

3.4. Phase: 3 - Data Analysis

After extracting the relevant data from the primary studies, we analyzed the collected data to find answers to our research questions. The articles were analyzed and classified by the first and last authors of this review.

Figure 3 depicts the selection of studies from the repositories. It shows that the majority of the studies were found in Web of Science and the smallest amount were found in ScienceDirect. However, many of the studies found in Web of Science can also be found in the other repositories. A search in the ACM database yielded the most unique studies. Figure 4 shows the ratio between Journal and Conference articles in the primary studies.

Figures 5 and 6 present the trend of research studies in the area of API usability over the years. Figure 5 shows the trend of publishing in the conferences and journals separately, and Figure 6 shows the trend of publishing

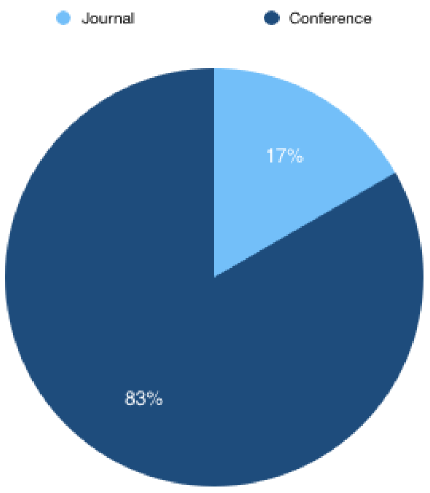

Figure 4: Journal and Conference Ratio in Primary Studies

over the years for the total number of publications (including both journal and conference publications). The figures show that research on API usability has gained momentum only after 2007. The work of Bore and Bore [16] appeared in 2005, but does not score well for its quality as shown in Table A. 2 - was only 2 pages long and had a negative citation score (defined above). At the beginning, more research publications can be found in conferences, which showed the initial research efforts. Gradually, as the research efforts matured, the journal publications also gained momentum. Some of the journal articles contained a more detailed and mature version of other previous studies of the same authors that appeared previously as conference/ workshop papers. We did not include previous conference/ workshop papers of such journal papers in the primary studies. However, they are included in trend analysis shown in Figures 5 and 6 to get a better view of how research in API usability evaluation progressed over time.

3.5. Threats to Validity

One fundamental issue that can affect the validity of our results is the selection of right keywords. In addition to our initially selected keywords, we have

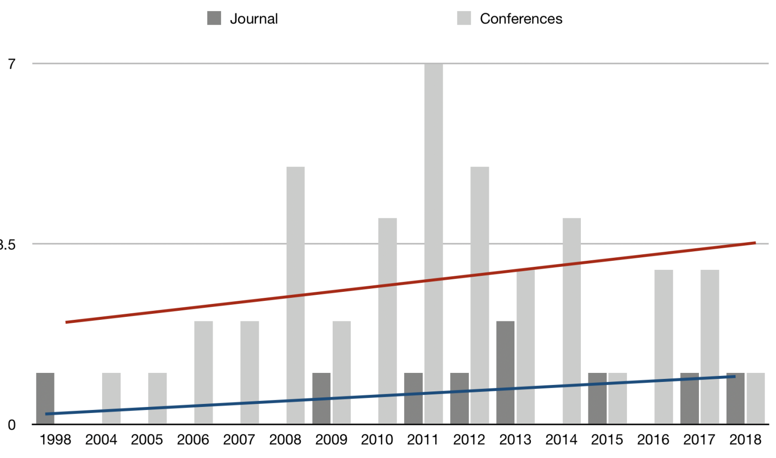

Figure 5: Trend Analysis of Primary Studies - Conferences and Journals Separately

also used their synonyms and alternate keywords in order to find as many studies as possible. When our understanding of the existing work improved, we also iteratively refined the keywords. The search string has to be comprehensive to result in finding as many relevant studies in the electronic repositories as possible. However, defining a search string that is comprehensive enough is difficult in this field of study due to the variety of terms used in different papers. We tried to address this issue by also analyzing the references given on the selected papers to find any additional papers that we might have missed. It resulted in finding some additional papers (explained in 3.2.3). There could still be some relevant papers that we might have missed, however, we used our best efforts to identify all the relevant literature.

The choice of the electronic repositories that we searched is also critical. We used the repositories mentioned above because they cover almost all important workshops, conferences and journal papers in our field of study. We also searched journal papers from the leading software engineering journals.

In addition, we also manually searched some of the conferences and journals since they publish work on human-computer interaction and human-centric computing. This was done to ensure, to the best of our efforts, that we did not miss

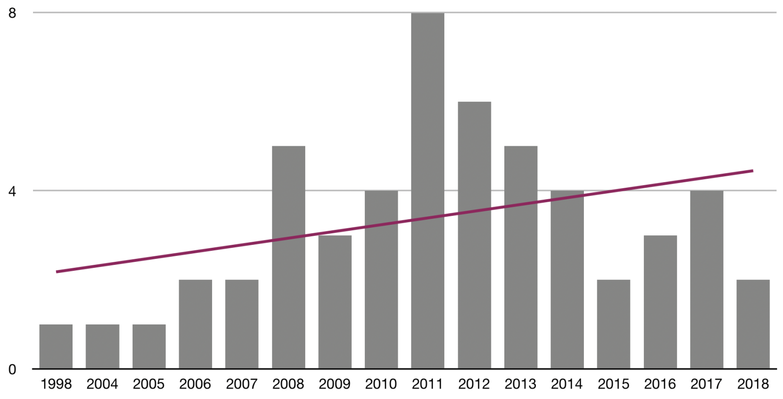

Figure 6: Trend Analysis of Primary Studies - Total Number of Publications

out any important work that did not appear with our search query. We also searched the references of the papers and web pages of the authors manually in order to find any relevant papers that we might have missed.

The classification of the research studies in different categories has been done by the first and last authors of this review. The classification is subjective, however, it has been done independently and the results compared between the two reviewers. In case of any conflict of opinion, the classifications were discussed until a common agreement was reached.

3.6. Research Results

Based on our inclusion and exclusion criteria, 47 articles are included in our SMS(listed in Appendix B - Tables B.4, B.5, and B.6).

4. Answering Research Questions

4.1. RQ 1: What are the different evaluation methods used to evaluate the usability of an API?

We can find in the literature two main categories of API Usability Evaluation methods: the *analytic* methods that have as an object of study an *API specification* itself and the *empirical* methods that focus on studying how developers *use*

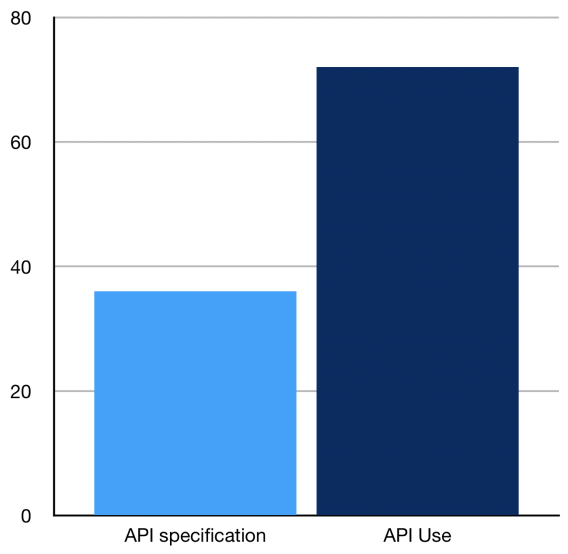

Figure 7: Object of Study

an API. This categorization is done to better understand the existing literature and identify different evaluation methods used to evaluate the usability of an API.

The literature review has revealed the following *analytic* methods:

- Metrics: An API design is measured using different software metrics and the measurements are compared against predefined thresholds.
- Reviews: Experts study an API design and its documentation in order to assess its usability and provide feedback on how to improve it. The reviews are usually conducted following a review protocol containing the heuristics or design guidelines and take as input the original API and its requirements. However, it is possible to conduct reviews that do not follow any specific guidelines and are solely based on the judgment and experience of the reviewers.

The *empirical* methods focus on studying how developers use an API in practice. We have identified mainly three empirical approaches to evaluate API usability in the literature :

- Task-Based Usability Tests: In a usability test, the subjects perform one or more predefined tasks using the API under evaluation. The subjects are observed while performing the tasks or questioned/surveyed after their completion. The output of their tasks may also be examined.

- Controlled experiments: The subjects perform one or more predefined tasks using different treatments such as different APIs. The effects of applying systematically the different treatments to each task are then compared. Ideally, the study contains a rigorous experiment design with clear hypothesis formulation, randomization or blocking of undesired factors, statistical hypothesis testing and analysis of threats to validity.

- Surveys and Repository Mining focus on the study of developer's past experiences using an API. This can be accomplished by inquiring developers directly via surveys or by mining issue tracking and defect databases and source code repositories that document the results of previous projects using the API.

Figure 7 shows that majority of the studies evaluate the usability of an API using empirical methods, i.e., 72%(n=31) of studies evaluate the usability of an API by studying how the users of the API are using it and 36%(n=17) of studies evaluate the usability of an API using API specifications in the absence of its users. The two categories are not mutually exclusive as some of the studies use both *empirical* and *analytic* methods to evaluate the usability of an API (Grill et al. [37], Scheller and Kühn [87], Murphy-Hill et al. [62], Mosqueira-Rey et al. [60]). Also, some empirical studies have been carried out to validate the analytic methods. Appendix C contains tables that summarize the primary studies in these categories. Table C.7 shows a summary of papers that use API specifications as the object of the study and Table C.8 presents a summary of papers that evaluate the usability of an API based on how they are used by the users.

Figure 8 shows how primary studies have used these evaluation methods to evaluate the usability of APIs. The first two columns (from left) show usage

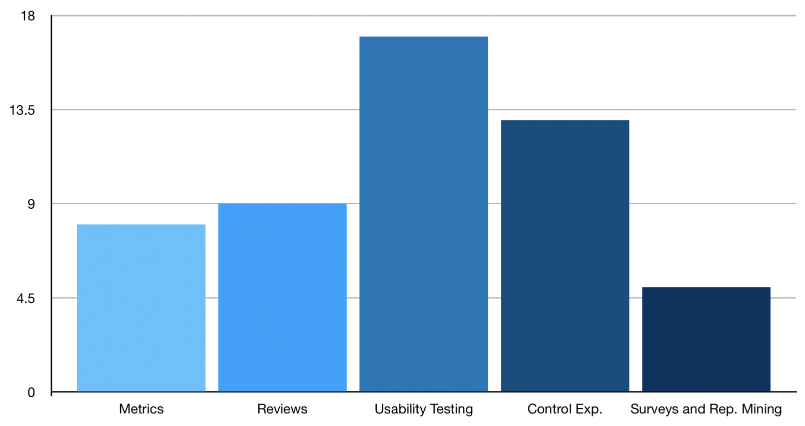

Figure 8: Evaluation Methods

of analytical methods in the primary studies, i.e., metrics and reviews, and the last three columns show usage of empirical methods in primary studies, i.e., usability tests, control experiments and surveys and repository mining.

The most used methods are usability testing and control experiments. This is because usability is often envisioned and studied with real users in mind. Human computer interaction (HCI) studies are rich with literature that studies how real users (in laboratory settings or outside) use and perceive an information or a software artefact. It then becomes quite obvious that researchers use the well-practiced usability evaluation methods to evaluate the usability of APIs as well. However, API differs from traditional software in many ways that makes the use of traditional empirical approaches like control experiments and user studies less-optimal for the API evaluation. APIs mostly have wide user-base, can be used in circumstances beyond the initial intent of the designers, are extensible, i.e., functionalities can be removed/ added throughout its lifecycle and are mostly open-source, hence, inviting developers from unknown backgrounds to update the code. These features require automated, scalable and time-efficient methods to evaluate the usability of an API. The least addressed evaluation methods, as shown in Figure 8, i.e., metrics (for analytical methods) and repository mining (for empirical methods) have the most potential to address these unique features of API and facilitate automated, scalable and time-efficient usability evaluation of APIs.

We do not discuss articles in details here. The articles in each of these categories are further categorized according to the aims of the studies, in RQ 2 (next), and further discussed therein.

4.2. RQ 2: What is the aim of the existing research efforts on API usability evaluation?

We categorized the selected studies according to their main research aims. We believe that this will help researchers and practitioners to understand the existing work and identify unexplored areas. The aims fall under four categories:

- *New Approach:* The article proposes and uses a new approach, that was not used previouly in literature, to evaluate the usability of an API.
- *Fundamental Results:* The main aim of the article is either a set of usability recommendations, guidelines or concepts that have been generalized beyond the concrete examples mentioned in the article.
- *API Evaluation Study:* The article evaluates the usability of a particular API with an aim to study how usable the API under study is. These articles can be of great value when planning new usability studies.
- *Software Tool:* The main aim of the article is a presentation of a software tool that is used to evaluate the usability of an API in an automated manner.

4.2.1. New Approach

The articles in this category add a new approach to evaluate the usability of an API in the research literature. The work of Ratiu [78], Scheller and Kühn [87], Rama and Kak [76], Hou et al. [43] and Stylos and Myers [92] propose new metrics to evaluate the usability of an API and use them to evaluate the usability of API/ APIs to demonstrate their applicability. While the works of

Stylos and Myers [92],Rama and Kak [76] and Scheller and Kühn [87] define metrics using programming constructs of API specifications, Ratiu [78] and Hou et al. [43] take a different approach to define metrics. Ratiu and Jurjens [77] use ontological analysis of API under study to define metrics. They map the ontology of the domain to an ontology of API that represents it and then define metrics that explicitly link the domain concepts to the program elements in the API. This explicit representation captures the complexity of concepts implemented in API.

Hou et al. [43] conducted scattered concern analysis focusing on the conceptual units that can help in understanding program code. In this method, the concerns that are scattered in the design are identified and either refactored or redocumented. They did a pilot study of the Java Swing JTree, identified concerns in it and evaluated them using the model-view-controller (MVC) architecture. Their work found that most of the concerns conformed to the MVC architecture. However, four design flaws negatively affected the usability of JTree that can be fixed to improve its usability.

The work of Watson [95], Farooq and Zirkler [30], Faulring et al. [31], Grill et al. [37], Lee et al. [55], Macvean et al. [57] and Mosqueira-Rey et al. [60] provided new approaches to review APIs in order to evaluate their usability.

Watson [95] applied technical communication in the initial design of an API in order to improve its usability. Based on heuristics taken from design guidelines by Cwalina and Abrams [23], he evaluated the usability of an API from a documentation perspective and analysed the API elements for their consistency. The suggestions were incorporated by the development team and corrections were made to make APIs more usable. Farooq and Zirkler [30] presented a usability assessment method based on peer reviews in order to evaluate API usability. The review included four interdisciplinary roles: “feature owner“, “feature area manager“, “usability engineer,“ and 3-4 reviewers. They selected reviewers to categorize defects in the API according to the cognitive dimension framework. The result showed that compared to usability tests, API peer reviews provided a more extensive usability test coverage.

Faulring et al. [31] performed two usability evaluations: one based on an inspection and the other based on a cognitive walk-through. They used a heuristic evaluation technique by involving experts who examined the user interfaces and gave their opinions based on an initial set of heuristics. They also conducted a study in which a SAP team used the SAP NetWeaver gateway service to develop common business use cases and did a cognitive walk through of their development activity. An important aspect of this study was to integrate HCI techniques with agile development. The work done by Grill et al. [37] used 16 design guidelines taken from factors identified by Zibran [100]. Their usability study comprised a heuristic-based inspection, a developer workshop and interviews. During inspection, reviewers were recruited and asked to find, analyse and categorize problems in an API according to a list of heuristics. Lee et al. [55] presented an API design process that included API reviewers as a part of the design process. API reviewers reviewed APIs and its documentation based on the API guidelines and documentation guidelines. Macvean et al. [57] provided a lightweight, distributed and scalable API design review process that aimed at improving the usability of Google's APIs. The key stakeholders of the review process were defined to be the API owner, design reviewer, design review team, shadow design reviewer and moderator. The reviews by each of the stakeholders were documented in a collaborative online document that allowed feedbacks from each stakeholders on each others review. This facilitated open dialogue and a single source for discussion. Their work was evaluated on 43 APIs with high satisfaction results from all the stakeholders of the API.

Mosqueira-Rey et al. [60] presented a set of heuristics and guidelines based on a usability model and context-of-use. These are used to analyse the usability of a sleep medicine API by integrating them in the heuristic evaluation with usability experts and subjective analysis by the users of the API. The main characteristic of their approach is the explicit integration of *context-of-use* into API usability evaluation.

Murphy-Hill et al. [62] presented a new analysis technique along with a tool for API usability evaluation. Compared to previous works that cover API usability issues at scale using surveys [80], online forums [42] or defect repositories [101], Murphy-Hill et al. [62] analysed the snapshots of successive edits (saved) by the developer and based on collective analysis of snapshots of different developers analysed the problem areas of the API.

Controlled experiments and usability studies are carried out to observer directly the use of an artefact (API in our case) by its users. Although, these techniques are not new when studying Human-Computer Interaction (HCI), some researchers have approached them in a novel way to study API usability.

Nanz et al. [63] presented a design of the study that can be used to compare ease of learning of two different concurrent programming languages by its users. They present a study template for learning a new concurrent language, set of test questions and evaluation scheme for interpreting their answers. The study design can be used to analyse the two programming languages side-by-side without losing the control over bias introduced due to variation in developers programming skills.

The work of Rauf et al. [79] introduced the analysis of users' perceptions about API and its usage by combining the use of Expectation-Confirmation theory [11] with control experiments.

The aim of the API usability study by Grill et al. [37] is to explore different HCI methods used for usability evaluation and compare the applicability of these methods in finding usability problems in an API. The usability issues discovered in each phase were classified in a number of categories that ranged from documentation to structural problems in an API.

Clarke 2004 was the first to use cognitive analysis in conjunction with usability studies aiming at making developers aware of their APIs and knowing the difficulties which the users face when using an API. O'Callaghan 2010 evaluated how well an API matches the user mental model using the API walkthrough method in order to study understanding of the code by its users.

Gerken et al. 2011 studied and identified usability issues and learning barriers that surface over time with a longitudinal study. The longitudinal study was carried out considering the learning of an API as a continuous process which cannot be effectively measured in few hours of study.

The work of Picconi et al. 2013 studied the effort required to understand semantics of API features and how an API can be learnt easily and incrementally.

Bhaskar et al. [10] investigated the usability of an API by combining the use of agile methodologies with the concepts of cognitive dimension framework.

4.2.2. Fundamental Results

The articles in this category evaluate the usability of an API using different methods like controlled experiments, usability studies, surveys, repository mining, etc. and based on their analysis provide recommendations or guidelines that can be applied to the development of other APIs to improve their usability.

Nasehi and Maurer 2010 proposed to use unit tests as API usage examples. Authors showed that writing good unit tests with common usage scenarios in mind can ease the learning of difficult APIs. They conducted a controlled experiment to study the API of Apache POI and showed that their unit tests were helpful in learning the API. Before starting the actual experiment, pilot study was performed on two students in order to determine the level of difficulty of the task descriptions.

Robillard [81] survey aimed to find problems that developers face when they are learning an API and present the important questions that come in developers' mind when they are using unfamiliar APIs. Their work identifies missing areas in documentations that come with APIs. By answering the questions raised in the study, developers can create more usable APIs. Zibran et al. [101] studied 1,513 defects reports across five different defect repositories and analysed them quantitatively and qualitatively. The aim of the study was to identify API usability issues that API users reported in the defect repositories and find significance of factors identified in his earlier work [100]. Using the results of the quantitative studies they ranked the factors based on how frequently they appear in the defect repositories.

Hou and Li [42] manually collected and analysed 172 newsgroup discussions from the Swing forum. They studied each case in detail and, based on their analysis, presented a list of different obstacles programmers face when using an API.

Gerken et al. 2011 studied and identified usability issues and learning barriers that emerged over time with a longitudinal study. Their work was motivated by the concern that the usability of an API cannot be grasped in few hours of study, as learning an API is a continuous process.

Duala-Ekoko and Robillard [27] conducted usability studies with 20 programmers and observed problems that programmers face when they encounter unfamiliar APIs. Based on their studies, they identified 20 different types of questions that developers ask when they are learning to use a new API.

4.2.3. API Evaluation Study

When it comes to evaluating usability of specific APIs, we found two types of studies. The first type evaluated a particular API giving information only about the usability of that particular API (e.g., [47], [16]) or evaluating a particular usability concept proposed by the authors (e.g., [52], [93]). The results of the former studies to evaluate API usability are not applicable to other APIs (e.g., [16], [9], [47], [21], [17], [52], [56], [83]) or result in validating a fundamental concept that can be used to improve the usability of APIs in the future (e.g., [93], [91], [72], [52]). Such studies are useful in studying how API usability can be evaluated.

The second type of studies evaluates the usability of a specific type of API, e.g., security APIs [35, 6] and static type systems[89], and then generalize their findings that are applicable to other APIs of that particular type. Such studies are included in two categories. One is under *API Evaluation Study* and the second category is one of the other three categories defined above, depending on the aims of that article. For example, the work of Mayer et. al [58], Endrikat et, al [29] and Spiza and Hanenberg [89] evaluate the usability of example APIs from static type systems against those of dynamic type systems and generalized their findings for all APIs of static type systems. The work of Cave et. al [21] compares library approach with a language approach. Both these approaches are used in task-parallel programming models. The library-based approach does not require any change in the tool-chain, e.g., in compilers or IDEs. However, they they may lead to code which is hard to understand. On the other hand, the language-based approaches may *require the standardization of new language constructs but are easier to understand and express intent of programmers* [21]. Their work concludes that common users with little knowledge on parallelism were more comfortable with a language approach, whereas advanced users found library approach to be more readable. The work of Acar et al. [6] evaluates the usability of security APIs and concluded that usability of APIs is a major hindrance to the adoption of security APIs.

The work of Jugel [47] conducts a theoretical discussion aiming at demonstrating the usability of the smart wrapper libraries generated through their model-driven approach. An informal review by the end-users for its usability assessment was also mentioned. However, real evaluation of it usability was not carried out. Cavé et al. [21] discuss and compare the usability of two programming models, library-based approach and language-based approach, for task parallelism. The discussion is built upon the authors' expert opinion and experience in teaching both the models.

4.2.4. Software Tool

Software tools exist only for evaluation methods that use metrics to evaluate API usability. In their work, de Souza and Bentolila [24] presented a tool, *Metrix*, that parses API definition and provides a visual representation of APIs with respect to their complexity. They used complexity metrics proposed by Bandi et al. [8] to evaluate API usability. Rama and Kak [76] provided nine API usability metrics based on the design structure. They presented different mathematical formulas to measure values of these metrics. The metrics can be computed automatically using their proprietary tool. They measured and analysed seven software systems to see how these systems scored for various metrics. The work of Scheller and Kühn [87], provides a promising direction in the auto-

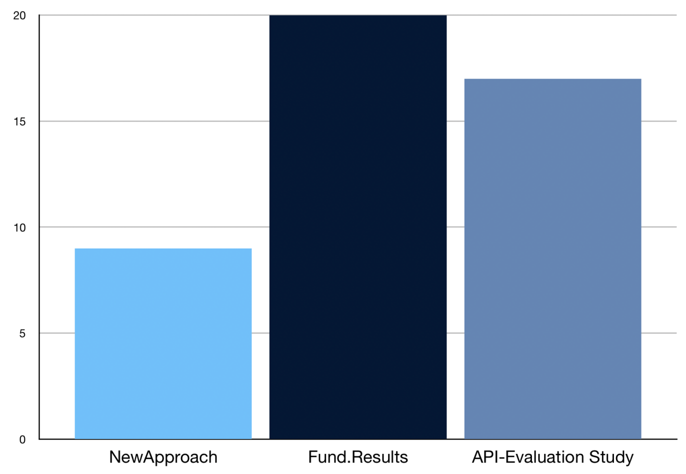

Figure 9: Aim of the Studies (API Use)

mated measurement of API usability, as they provide a number of metrics and studies that evaluate API usability objectively. However, an automated tool that measures API usability based on these metrics is not available.

Another notable tool, StopMotion Murphy-Hill et al. [62], analyses the file history of developers using a particular API. The tool looks for similar patterns in the snapshots saved during the work history of the developer. Based on the analysis of similar patterns of method changes, i.e., what methods are users changing often, it identifies API usability issues.

The categories discussed above are not mutually exclusive as some articles have more than one aim. Figures 9 and 10 show the number of primary studies according to their research aims categorized according to the objects of their study, i.e., API use or API specifications. A summary of this classification is attached in Appendix C, in Tables C.7 and C.8.

Figure 9 shows the aim of the research articles that use empirical methods to evaluate API usability. Their goal is to provide fundamental results, i.e., recommendations, guidelines or general principles, that can be used by API designers and developers to improve or evaluate API usability. These approaches mostly learn from API users in (relatively) small settings and then generalize

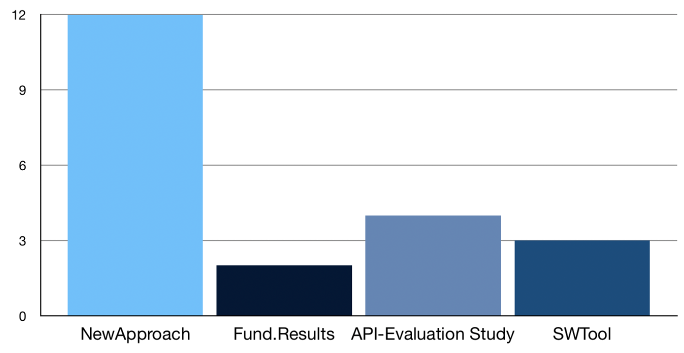

Figure 10: Aim of the Studies (API Specifications)

their results. Figure 10 shows that articles employing API specifications as an object of study, mostly do not aim to generalize their results but rather to provide novel ways to evaluate API usability in the absence of its users. This trend shows a pressing need for API usability evaluation approaches that scale up, without needing users, as engaging users for such activities can be expensive and time-consuming. However, among all the studies, only a few aimed at providing usability evaluation tools, out of which most are proprietary and not available for general use.

4.3. RQ 3: In which phase of API development does the API usability evaluation method apply?

Just like any software, an API goes through typical development phases of a software development lifecycle: analysis & design phase, development phase and post-development phase. Each phase of development feeds next phases with a set of constraints and directions of development upon which the next process of development is carried out. Researchers and practitioners advocate 'best practices' that consider incorporating an usability feature into API development from initial phases, since incorporating it as an afterthought may not yield the best possible results [39]. In addition, evaluating the usability of an API after it is developed, though useful, can be a time-consuming and expensive task since the results of the evaluation may require changes that can be costly to fix once

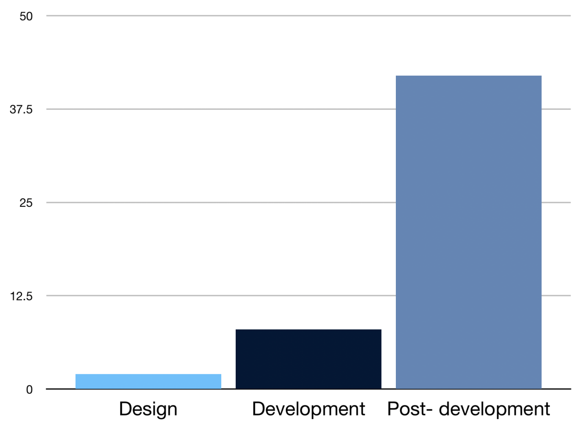

Figure 11: API Usability Evaluation Methods Applied at Development Phases

it is developed. Moreover, if an API is published already, changing an API due to its usability concerns can affect its reputation and other applications that use the API. Although researchers have provided tools and techniques that can help the developers to write code with relative ease [68], in this research question we study what approaches are available that developers can use to evaluate the usability of their API during different phases of development. This can help developers to identify the usability issues at an appropriate time and take appropriate measures when needed.

Figure 11 shows the number of primary studies according to the phase of API development they address when evaluating the usability of API. It is evident from Figure 11 that most of the usability evaluation methods used for APIs target the post-development phase. The summary of primary studies classified according to phases of API development is presented in Appendix D.

Only two approaches are available in the current literature that address the evaluation of API usability from the design phase. The work of Lee et al. [55] and Macvean et al. [57] address incorporating API usability evaluation strategies starting from the design phase, so that its usability is continuously monitored and addressed throughout the development phase. These approaches can be of advantage in a team environment, where APIs are developed collaboratively.

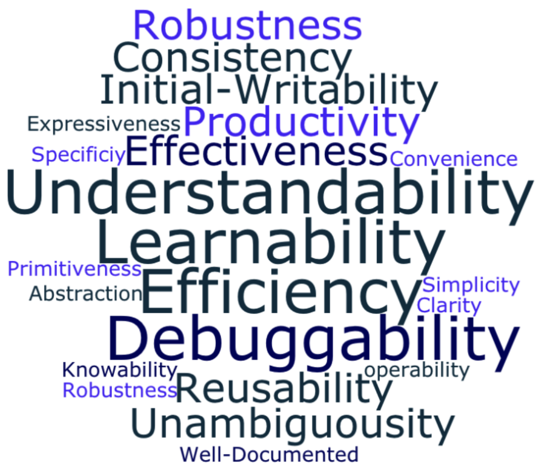

Figure 12: Word Cloud for Usability factors considered in Primary studies

However, for small teams where roles within teams are not clearly defined, adaptation or iteration of such approaches would be needed.

In the development phase, the work of researchers and practitioners can learn from the works of Clarke [22], O'Callaghan [69], Farooq and Zirkler [30], Ratiu and Jurjens [77], Watson [95], Faulring et al. [31] and Bhaskar et al. [10]. Most of these approaches can be used iteratively between the development and post-development phases.

The most concentrated efforts to evaluate API usability are done in the phase of post-development, showing a general trend towards API usability evaluation as an afterthought. Although an increase in research efforts to evaluate API usability is encouraging, it also points out that research in this phase is still at an early stage and more efforts are needed to help the developers evaluate the usability of their API when they design and develop their APIs.

4.4. RQ 4 :What are the usability factors addressed by the existing API usability evaluation studies?

We discussed in section 2 that usability is a subjective term and different usability standards and researchers have defined it differently. We investigate existing literature to study what are the usability factors that are considered by the researchers when evaluating API usability. Our study reveals that usability is defined and evaluated differently by different researchers. Figure 12 a shows word cloud of different terms used in primary studies to define and evaluate API usability. Appendix E contains tables that summarize research studies according to the usability factors they address. We have selected these usability factors based on the terms used by the authors. However, in some studies the authors use synonym words to define the same usability factor. Such papers were classified under one term. For example, Mayer et. al [58] defines *the time required to fulfill a certain task* as *Effort in programming*, whereas in other papers([28, 87, 61, 94]) it is mostly defined as *efficiency.* We categorized this API usability factor identified by Mayer et. al [58] as efficiency. Productivity is considered to be different from efficiency. While efficiency measures the time required to fulfill a task, *productivity* is defined differently by different authors. Similarly, Ellis et. al [28] identified debuggability as a usability factor. It is also called *better debugging* in the work of Nanz et al. [63]. In our review, we categorize these views under the same category: usability factor *debuggability.* There is also contradiction regarding how different authors define the same usabiltity factor, for example, Farooq and Dieter[30] defined productivity as *ratio of the mean number of bugs per test/session iteration* [30], while Fisher and Stefan [33] defined productivity as a number of tasks completed in a given time. Below, we give a brief summary of usability factors addressed in the literature and shown in Appendix E. These usability factors are not mutually exclusive, as usability factors are subjective in nature.

- Learnability: measures the capability of software to be learnt by its developers with ease [88].
- Efficiency: defines how much software enables its users to use the right amount of resources to complete a task in a specific context [88].
- Understandability: measures how well a user can understand the code without confusion.

- Effectiveness: defines how much software enables its users to complete their tasks correctly [88].
- Readability: measures the level to which a code written with the API is readable by its users, such that they are able to follow it logically.
- Satisfaction: provides subjective response of the users about their comfort with software and positive feelings after using it [88].
- Debuggability: measures the level to which a user is able to find errors in code and debug it.
- Productivity: defines to what level software enables its users to complete their tasks accurately with respect to the resources expended [88].
- Reusability: measures the level to which parts of code written with the API can be reused in other contexts.
- Abstraction: Lopez et. al [56] defines it as: *Abstraction is the ability of the API to guarantee that programmers can use the API with proficiency without requiring specific knowledge or assumptions in relation to its implementation details.*
- Expressiveness: Lopez et. al [56] defines it as: *the ability of inferring readily the purpose of an entity.*
- Initial-Writability: measures how easy it is for the developers to write a simple initial code using the API [91].
- Unambiguousity: Lee et. al [55] defines it as: *The function of an API is figured out by its name. The name conforms to the coding conventions.*
- Primitiveness: Lee et. al [55] defines it as: *An API provides a single function. If the function can be provided by the combination of other APIs, it does not meet primitiveness.*

- Convenience: Lee et. al [55] defines it as: *The parameters in an API are minimally defined, and order and type of parameters keep consistency.*

- Specificity: Bore and Bore [16] defines it as: *percentage of API elements that address application functionality.*

- Simplicity: Bore and Bore [16] defines it as: *how easily a developer can translate their required application functionality into API elements.*

- Clarity: Bore and Bore [16] defines it as: *how obvious is purpose from the name.*

- Layered-APIs: Bore and Bore [16] defines it as: *measuring collective effect of different layers in an API (components, interfaces in the component, functions in interfaces etc.).*

Cognitive analysis of information artefacts and software products measure the mental process of how the users of the system understand the system. There is a range of studies that use different cognitive theories to analyze the API usability. These are:

- Zibran's GL: Guidelines (GL) to create usable APIs are given by Zibran et. al [101] and then used by others in their work. They are: *1) easy to learn 2) easy to remember 3) easy to write client code 4)easy to interpret client code* and *5) difficult to misuse.*

- Bloch's Guidelines (GL): Guidelines to create usable APIs are given by Bloch [14] and then used by others in their work. They are: *1) easy to learn, 2) easy to use, 3) hard to misuse 4) easy to read and maintain the code that uses it, 5) sufficiently powerful to satisfy requirements 6) easy to extend* and *7) appropriate to audience.*

- Cognitive Dimension Framework (CDF) [36]: The work of Green et. al [36] presents the *cognitive dimension framework* for analyzing usability of visual programming environments. It captures the psychology of the programming by providing a set of vocabulary that can capture Cognitve relation among different aspects of programming structure. These terms are: *Abstraction Gradient, Closeness of mapping, Consistency, Diffuseness, Error-proneness, Hard mental operations, Hidden dependencies, Premature commitment, Progressive evaluation, Role-expressiveness, Secondary notation, Viscosity* and *Visibility.*

- Cognititive Dimension Framework (CDF) [22]: The work of Clarke [22] adapted CDF by Green and Petre [36] to suit API usability. These include: *Abstraction level, , Learning style, Working framework, Work-step unit, Progressive evaluation, Premature commitment, Penetrability, API elaboration, API viscosity, Consistency, Role expressiveness* and *Domain correspondence.*

- Conceptual F/W: Fischer's conceptual framework [32] outlines reuse and redesign strategies to describe different software tools. His framework identifies design tasks as translation of users' problems to a formal system model supported by programmer's cognitive perspective of the problem.

- Cognitive Walkthrough (CWT) [12]: Blackmon et. al [12] presents steps for conductive cognitive walkthrough of the web.

- ECT: Expectation-Confirmation Theory [11] measures *perceived ease of use, perceived ease of learning, perceived confirmation, intention to continue,* and *satisfaction* of API users.

The cognitive dimension framework presented by Green and Petre [36] has been widely used in the usability analysis of different software systems. Interestingly, it has also been used much for analyzing the usability of an API.

Clarke [22] used CDF by Green and Petre [36] as a data analysis method in usability tests and adapted them to suit API usability. These works ([22] and [36]) on cognitive dimensions for API usability have been used alternatively by different researchers when using cognitive dimension frameworks for studying API usability (e.g., [94] and [91], [72], [10], etc.) (See Table 5).

However, Diprose et. al [25], [10],[9], [93] used the original work on Cognitive Dimensions framework by Green and Petre [36] to analyze usability of an API in his work.

The work of Hou and Li [42] used Fisher's conceptual framework [32] to organize their results into cognitive stages.

Faulring et. al [31] used work of Blackmon et. al [12] for conductive cognitive walkthrough of the tool under study to evaluate its usability.

The work of Rauf et. al [79] used expectation-confirmation theory (ECT) and adapted it to suit the evaluation of API usability. Expectation-Confirmation Theory (ECT) theory is widely accepted in the information systems research [41]. It shows a strong correlation between users' intentions to continue and users' satisfaction and other cognitive beliefs [11].

Although different cognitive analysis methods have been used to study API usability, the work of Green and Petre [36] and its adaptation to API usability by Clarke [22] have been used widely in the existing literature in this area. The work of Wijayarathna et al. [96] also builds on the work of Clarke [22] and presents a generic questionnaire for evaluating the usability of security APIs. However, this work is not included in our primary studies since the empirical evaluation of the usability of security APIs based on this questionaire is a part of their future work at this point in time.

When comparing the word cloud generated for the terms used in the primary studies (Figure 12) to the word cloud generated for the terms used in existing non-API specific usability literature to define usability, we see some distinct similarities and differences. Learnability is considered the most emphasized factor for measuring usability. 49% of primary studies studied learnability of developers in order to evaluate the usability of an API, followed by efficiency and understandability. More focused research efforts are needed to study other factors to evaluate API usability. The word 'context' is also very important when defining usability in standards and models. However, the primary studies do not consider ‘context’ when evaluating API usability, except the latest work by Mosqueira-Rey et al. [60], which uses the concept and taxonomy of ‘context-of-use’ for evaluating API usability.

It is interesting to note that while the work of Mosqueira-Rey et al. [60] and Gorski and Iacono [35] have attempted to summarise the existing body of knowledge into existing usability models and taxonomies, we see new terms emerging from API usability literature which may not be adequately represented by placing them under general models to define usability, for example, *debuggability*, *initial-writability*, *expressiveness*, etc. Many of the terms from existing usability taxonomies used in the work of Mosqueira-Rey et al. [60] do not represent usability of an API, for example, *safety, aesthetics,* etc. Similarly, the work of Gorski and Iacono [35] does not mention how physical interface and logical interface from the adopted usability model of Winter et al. [97] will differ for an API.

5. Previous Literature Reviews

The systematic studies are conducted to perform a methodical assessment of a subject using some predefined plan, i.e., focusing on some particular research questions [46]. In the field of API analysis, we found systematic mapping study by Burns et al. [18]. He published a summary of 28 papers in the field of API usability. Their goal was to identify different methodologies used in these papers and the kinds of recommendations they provide. Keeping in mind the continuously evolving nature of research in APIs and its usability for programmers, there is a need for renewed analysis in this area. Our work differs from their work in answering previously unanswered questions and presenting quality assessment of primary studies. In addition, our detailed approach includes new research articles that were not included previously either due to the limited scope of their review or being published afterwards. Our work, thus, covers the state of the art in the area of API usability evaluation.

The work of Mosqueira-Rey et al. [60] and Gorski and Iacono [35] also attempted to analyze existing literature. Although we acknowledge a positive impact of their work, we believe that our work differs from their in the following key areas. They do not conduct a formal systematic mapping study based on a formal protocol and do not answer the research questions answered in our work. Both the approaches adopt existing usability models, Gorski and Iacono [35] adopts work of Winter et al. [97], and Mosqueira-Rey et al. [60] adopts usability taxonomy from the work of Alonso-Ríos et al. [7]. Instead of adopting existing usability taxonomies and models, our work learns from existing literature and emphasizes the need for unique terms and concepts that are specific to API usability. In addition, many unique features of API cannot be adequately captured by generalized usability factors and may lose their meaning and significance to the developer when classified under usability factors that are too general.

6. Discussion

The earliest work that explicitly highlights the importance of evaluating API usability from the perspective of programmers comes from the work of McLellan et al. [59]. After that, we see a range of research efforts in this area.

We observed an increasing trend (Figure 5) in the research efforts to evaluate API usability using multiple methods often aiming at different usability factors. 32 out of 47 studies reviewed in this article appeared from 2010 to 2018, i.e., in the span of the last 8 years, compared to the other 14 studies that appeared from the year 1998 to 2009, i.e., a span of more than 10 years. The trend of increased efforts to evaluate API usability can be attributed to the exponential growth of APIs due to advancements in the Internet and digital technologies.

Our systematic analysis of the existing body of knowledge not only shows an increase in efforts to evaluate API usability but also demonstrates that as the research in API usability gets relatively mature over a period of more than a decade, researchers are building upon this literature and extending it for adoption in other specific aspects, e.g., security APIs. Earlier research on API usability evaluation, researched the evaluation of API usability with results that can be generalized or novel methods to evaluate all kinds of APIs. Some research studies under the category of *API usability evaluation*, applied API usability evaluation approaches to study the usability of APIs for different scenarios, e.g., task parallelism [21], static type systems [58], etc. or for different domains, like interactive public display applications [20], concurrent programming languages [63], etc. However, in recent years we see increasing interest in evaluating API usability for security APIs, i.e., APIs that provide security features to protect data confidentiality, integrity and availability [15]. We see a steep increase in efforts to evaluate usability of security API, i.e., work of Gorski and Iacono [35] in 2016, Acar et al. [6] and Wijayarathna et al. [96] in 2017 and work of Murphy-Hill et al. [62] in 2018. Wijayarathna et al. [96] adapt cognitive dimension framework from API usability literature to evaluate usability of security APIs. Similarly, the work of Gorski and Iacono [35] builds on a consolidated view of the existing research on API usability and based on that, proposes eleven characteristics to evaluate usability of security APIs. We attribute this trend firstly to relative maturity of API usability such that it can be used as a baseline for the new areas of research. Secondly, misuse of APIs leads to security flaws that can be crucial for the users of applications developed using these APIs and also for the API providers and the application providers who may suffer from security flaws in the applications. We consider this a good use of API usability evaluation work.

However, our analysis of current literature shows that research on API usability evaluation still has many limitations and open research avenues that need further exploration. We thus found the following limitations and research opportunities in the existing literature.

6.1. Software Tools to Evaluate API Usability

The analysis of the current body of knowledge domain revealed five types of evaluation methods that we have categorized into *analytical* and *empirical* methods. Most work in recent years has focused on the *empirical* methods.

The research efforts to evaluate API usability using metrics, i.e., an analytical method, started only a decade ago and has gained the attention of researchers aiming at an objective evaluation of API usability without involving its users. The use of the analytical methods provides efficient ways to evaluate API usability in a fast-pace digital world. Metrics, compared to other empirical methods, can be time and cost efficient provided that proper tools are available. We, however, observed that there is a lack of actual tools that can calculate the metrics automatically. Only three tools were mentioned in the literature, two of them have not been released and the other one is not being maintained.

In their work, de Souza and Bentolila [24] took API usability as a function of its complexity and presented a tool, *Metrix*, that parses API definition and provides a visual representation of APIs with respect to their complexity.

An evaluation tool for measuring API usability was also presented by Doucette [26]. However, as this work is not presented in a peer-reviewed scientific venue, it is not part of our primary studies. The tool is created in Java and gives statistics on Java APIs based on 12 metrics. These metrics were created from existing guidelines, heuristics and qualitative studies about API usability present in the literature.

A tool to increase API usability has been presented by Stylos [90]. Although the studies on which this tool is built are included in our primary studies, i.e., [92, 91, 93], the tool [90] is not a part of our primary studies as it does not evaluate API usability, and only helps in API discovery and writing a better code.

Rama and Kak [76] provided nine API usability metrics based on the design structure. They presented different mathematical formulas to measure values of these metrics. These metrics can be computed automatically using their proprietary tool. They measured and analysed seven software systems to see how these systems scored for various metrics.

The work of Scheller [87] presented metrics that can be used to build tools and calculate the usability of APIs in an automated manner. It builds on his previous works ([84], [86], [85]). Although the work of Scheller et. al [87] is a promising direction in the automated measurement of API usability, an automated tool that measures API usability based on these metrics is not available.

The current software tools, briefly mentioned in this section, only use software metrics to quantify and evaluate the usability of API. The methods and tools for evaluating API usability can greatly benefit from integrating data driven approaches, such as machine learning [98]. Nowadays there are large volumes of data that can be collected about different aspects of using APIs. By relying on machine learning techniques, it would be possible to identify certain trends in using APIs. For instance, machine learning can shed a light on the most preferential architectural and documentation style of APIs, differences between using APIs in different application domains and development platforms, programmers preferences based on their experience etc. This opens promising perspective towards building the adaptive APIs, i.e., the APIs that are can be continuously improved and easier adapted to task at hand. Adaptive software development [40] presents the concept of continuous adaptation of task-in-hand using a collaborative approach. Hence, this would allow the API developers to be better informed about the user's needs and consequently cater to them already during the API development.

6.2. Defining API Usability Factors

Our analysis of existing literature shows an inconsistent use of terms to define usability. Specifically, literature shows confusion in terms like factors, characteristics and guidelines. Henning [39] presented 8 guidelines to design good APIs. Bloch [14] defined five characteristics of a good API. He also provided 39 design recommendations to achieve these factors.

Zibran [100] defined five characteristics for a usable API. He further identified 22 factors that affect these API characteristics based on his literature review. In [101], Zibran studied 1,513 defect reports across five different defect repositories based on characteristics identified in his previous work [100].

Grill et. al [37] used 16 heuristics from the work of Zibran [100] to categorize different problem areas in APIs identified during heuristic evaluation of APIs.

Although Zibran [100] terms these heuristics as *factors*, Grill et al. [37] terms them as *heuristics.* For preparing the interview questions, the authors use five characteristics of a good API defined by Bloch [14] and term them usability *factors.*

Bore and Bore [16] also provided factors that contribute to API usability. They built their work on the basic programming guidelines by Kernighan et al. [49], [48].

This confusion between researchers on using a standard set of usability factors and using different terms like *factors, characteristics, guidelines,* etc. interchangeably to evaluate API usability, makes understanding of existing literature difficult.

In addition, different usability evaluation studies have used either different terminologies to define the same parameters, e.g., Mayer et al. [58] defined *the time required to fulfill a certain task* as *effort in programming* whereas the work of Ellis et al. [28] defined it as *efficiency.* Ellis et al. [28] identified debuggability as a usability factor and Nanz et al. [63] terms it as *better debugging.* Some studies have used the same term with different meanings, e.g., Farooq and Zirkler [30] defined *productivity* as a *ratio of the mean number of bugs per test/session iteration* and Fischer and Hanenberg [33] define *productivity* as number of tasks completed in a given time.

Interestingly, we also observed that the correspondence between different features of API and API usability factors is often based on authors' own opinions lacking any evidence for such a correspondence based on empirical studies or expert reviews.

For example, de Souza and Bentolila [24] assumed that APIs that are complex are hard to use and then presented the tool that provided a visual representation of the complexity of an API and Jugel [47] assumed that smaller number of choices for methods usually improves readability.

This inconsistency in usage of terms to define and evaluate usability stems from the inherent subjective nature of the term usability that encompasses a broad set of concepts. A distinct divide in the work of researchers in the areas of software engineers and behavioral sciences also adds to the ambiguity in defining API usability. There is a wealth of literature that defines usability better and strategies to evaluate it using empirical and non-empirical studies. The evaluation of API usability can be studied better by studying the inter-relationship between the behavioral sciences and modern technologies in software engineering.

7. Conclusion

The provision of usable APIs is an important area since with the advent of ubiquitous computing and proliferation of APIs, the development of software systems does not remain the job of a selected few who are expert in programming and technical skills. It has, therefore, become important to develop APIs with good usability. APIs with poor usability may reduce programmers' productivity and can lead to an increased number of defects. On the other hand, APIs with good usability can increase programmers' productivity and satisfaction.

In this systematic mapping study, we attempted to analyze state of the art in the area of evaluation methods for API usability. We were interested in studying how the research in evaluating API usability evolved and provided an analysis of the existing literature. Our study shows that research on evaluating API usability is gaining momentum and is getting mature with more and more researchers contributing towards evaluating API usability in novel ways. Based on our analysis of existing literature, we identified certain limitations in the current body of knowledge and identified the following open issues and future research directions in this domain that can improve the way API usability is evaluated.

- Usability has an inherently subjective nature since it involves human interpretation. We have identified a lack in the synergy of efforts that define and evaluate API usability. This is evident by a large number of API usability factors addressed by the researchers and inconsistent use of usability terminologies by the different authors. Some studies review API usability literature and attempt to model it with general usability models. However, in doing so some unique characteristics of API are compromised and many redundant factors are introduced that are not applicable to the evaluation of API usability. In answering RQ4, we have shown emergence of new terms from our study of the literature. There is a need for consolidated approaches to define usability models for API to provide a standard approach to evaluate API usability.

- Usability is a broad term with many interpretations. The analysis of research in API usability evaluation shows that most of the research studies focused on evaluating *learnability* of an API. This points towards an important aspect of API usability evaluation, i.e., researchers are interested in evaluating how quickly or easily different types of users are able to learn and use the API. It also points at the interest in knowing how API is learnt in different contexts. With widespread adoption of APIs, APIs are being used by a wide variety of users in several different and well-known contexts. Using empirical methods with limited number of users in lab settings to evaluate API usability does not adequately reveal the real picture of how API is learnt by the 'masses', i.e., a large number of users from different backgrounds. Novel approaches are required to evaluate learnability of an API using technological advances in the areas of big data, adaptive software engineering and machine learning approaches.

- Although learnability contributes significantly to the usability of an API, other usability factors also play an important role in API usability but have received considerably less attention. There is a need for research efforts that focus on other factors of API usability, besides learnability, in order to evaluate API usability with greater confidence.

- The usability evaluation of API requires evaluation of the cognitive load on human users of API which is usually done in limited settings with a relatively small number of users, on the other hand, industrial objective of faster-to-market and catering to large number of users pose new challenges for research and industry alike. We believe that developing APIs with good usability requires interdisciplinary knowledge that combines cognitive pychology with software engineering in an efficient manner that meets industrial objectives of *faster-to-market*.

- An API, once released, is available for public use and maybe used in scenarios not evident to API developers. It becomes difficult to change it since there can be many programmers and programs using it. It is, therefore, important to evaluate API usability before a public release. The current literature on API usability shows that API usability is mostly evaluated after the development of an API. More research efforts are needed to provide approaches that can evaluate API usability during the design and development phases.

We conclude that although there exist many publications proposing API evaluation methods, this subject still has many open questions for research. In particular, we believe that the evaluation of API usability in a time and cost efficient manner poses many interesting research challenges that have not been addressed in the existing literature. In addition, APIs are increasingly being used by users of different capabilities and skills. Thus, we believe that developing APIs with good usability requires interdisciplinary knowledge that combines cognitive psychology with software engineering. Combining this with the need to develop efficient approaches that evaluate API usability to meet faster-to-market software development cycles, makes it a challenging research area. We hope that this review provides both the researchers and practitioners interested in API usability with the useful ideas about how to develop this subject further.

Appendix A. Study Quality Assessment

Table A.2 and Table A.3 give Study Quality Assessment (SQA) for Primary Studies.

Table A.2: Study Quality Assessment (SQA) for Primary Studies

Type		**SQA>0**	**SQA<=0**	**No SQA(year<=2017)**
Journals		Robillard and Deline [81],	Jugel [47],	López-Fernández et al. [56]
		McLellan et al. [59],		Santos and Myers [83],
		Grill et al. [37],		Mosqueira-Rey et al. [60]
		Scheller and Kühn [87],		
		Nanz et al. [63],		
		Rama and Kak [76]		
Conferences	**p> 5**	Beaton et al. [9],	O'Callaghan [69],	Murphy-Hill et al. [62],
		Zibran et al. [101],	Cardoso and José [20],	Acar et al. [6],
		Macvean et al. [57],	Ratiu and Jurjens [77]	
		Spiza and Hanenberg [89],	Lee et al. [55]	
		Fischer and Hanenberg [33],	Watson [95],	
		Cavé et al. [21],	Diprose et al. [25]	
		Cardoso and José [20],	Stylos et al. [93],	
		Piccioni et al. [72],	Hou et al. [43]	
		Endrikat et al. [29],		
		Brunet et al. [17],		

Table A.3: Study Quality Assessment (SQA) for Primary Studies (continued)

Type		**SQA>0**	**SQA<=0**	**No SQA(year<=2017)**
Conferences	**p> 5**	Mayer et al. [58],		
		Duala-Ekoko and Robillard [27],		
		Nasehi and Maurer [64],		
		Gerken et al. [34],		
		Hou and Li [42]		
		Stylos and Clarke [91],		
		Ellis et al. [28],		
		Stylos and Myers [92]		
	p <= 5	Faulring et al. [31]	Bhaskar et al. [10],	
		Clarke [22], Farooq and Zirkler [30], Ko and Riche [52], Stylos et al. [94], de Souza and Bentolila [24]	Rauf et al. [79], Bore and Bore [16], Murphy-Hill [61]	

Appendix B. Selected Papers

Table B.4, Table B.5 and Table B.6 list the primary studies.

Appendix C. Articles Classified according to Subject of Study, Evaluation Method and Aim of the Study (UQ= Unique Papers)

Table C.7 and Table C.8 classify the primary studies according to object of study, i.e., API specification or API use, respectively.

Appendix D. Articles Classified according to Phases of Development

Table D.9 classify primary studies according to Phases of Development.

Appendix E. Usability Factors Addressed in Primary Studies

Table E.10 and Table E.11 classify primary studies according to usability factors addressed.

References

[1] Google Maps Platform -Google Developers. `https://developers.google.com/maps/documentation/`. (Accessed on 10/24/2018).

[2] Holiday Homes & Condo Rentals - Airbnb. `https://www.airbnb.co.uk/`. (Accessed on 02/11/2019).

[3] Uber — Sign Up to Drive. `https://www.uber.com/`. (Accessed on 02/11/2019).

[4] What Are APIs and How Do They Work? — ProgrammableWeb. `https://www.programmableweb.com/api-university/what-are-apis-and-how-do-they-work`. (Accessed on 02/11/2019).

Table B.4: Selected Papers

#	**Author(s)**	**Title**	**Year**
1	McLellan et al. [59]	Building more usable APIs	1998
2	Clarke [22]	Measuring API usability	2004
3	Bore and Bore [16]	Profiling software API usability for consumer electronics	2005
4	Stylos et al. [93]	Comparing API design choices with usability studies: A case study and future directions	2006
5	Murphy-Hill [61]	Improving usability of refactoring tools	2006
6	Stylos and Clarke [91]	Usability implications of requiring parameters in objects' constructors	2007
7	Ellis et al. [28]	The factory pattern in API design: A usability evaluation	2007
8	Stylos and Myers [92]	The implications of method placement on API learnability	2008
9	Hou et al. [43]	Documenting and evaluating scattered concerns for framework usability: A case study	2008
10	Stylos et al. [94]	A case study of API redesign for improved usability	2008
11	Ratiu and Jurjens [77]	Evaluating the reference and representation of domain concepts in APIs	2008
12	Beaton et al. [9]	Usability challenges for enterprise service-oriented architecture APIs	2008
13	de Souza and Bentolila [24]	Automatic evaluation of API usability using complexity metrics and visualizations	2009
14	Watson [95]	Improving software API usability through text analysis: A case study	2009
15	Jugel [47]	Generating smart wrapper libraries for arbitrary APIs	2010
16	Cavé et al. [21]	Comparing the usability of library vs. language approaches to task parallelism	2010
17	O'Callaghan [69]	The API walkthrough method: a lightweight method for getting early feedback about an API	2010
18	Nasehi and Maurer [64]	Unit tests as API usage examples	2010

Table B.5: Selected Papers (continued)

#	**Author(s)**	**Title**	**Year**
19	Farooq and Zirkler [30]	API peer reviews: a method for evaluating usability of application programming interfaces	2010
20	Gerken et al. [34]	The concept maps method as a tool to evaluate the usability of APIs	2011
21	Hou and Li [42]	Obstacles in using frameworks and APIs: An exploratory study of programmers' newsgroup discussions	2011
22	Ko and Riche [52]	The role of conceptual knowledge in API usability	2011
23	Brunet et al. [17]	Structural conformance checking with design tests: An evaluation of usability and scalability	2011
24	Zibran et al. [101]	Useful, but usable? factors affecting the usability of APIs	2011
25	Faulring et al. [31]	A case study of using HCI methods to improve tools for programmers	2012
26	Grill et al. [37]	Methods towards API usability: a structural analysis of usability problem categories	2012
27	Duala-Ekoko and Robillard [27]	Asking and answering questions about unfamiliar APIs: an exploratory study	2012
28	Mayer et al. [58]	An empirical study of the influence of static type systems on the usability of undocumented software	2012
29	Nanz et al. [63]	Design of an empirical study for comparing the usability of concurrent programming languages	2013
30	Rama and Kak [76]	Some structural measures of API usability	2013
31	Piccioni et al. [72]	An empirical study of API usability	2013
32	Cardoso and José [20]	Evaluation of a programming toolkit for interactive public display applications	2013
33	Nanz et al. [63]	Design of an empirical study for comparing the usability of concurrent programming languages	2013
34	Lee et al. [55]	An API Design Process in Terms of Usability: A Case Study on Building More Usable APIs for Smart TV Platform	2014
35	Endrikat et al. [29]	How do API documentation and static typing affect API usability?	2014

Table B.6: Selected Papers (continued)

#	Author(s)	Title	Year
36	Spiza and Hanenberg [89]	Type names without static type checking already improve the usability of APIs (as long as the type names are correct): An empirical study	2014
37	Diprose et al. [25]	A human-centric API for programming socially interactive robots	2014
38	Fischer and Hanenberg [33]	An empirical investigation of the effects of type systems and code completion on API usability using typescript and javascript in ms visual studio	2015
39	Scheller and Kühn [87]	Automated measurement of API usability: The API concepts framework	2015
40	Bhaskar et al. [10]	Developing usable APIs with XP and cognitive dimensions	2016
41	Rauf et al. [79]	Perceived obstacles by novice developers adopting user interface APIs and tools	2016
42	Macvean et al. [57]	API Design Reviews at Scale	2016
43	Santos and Myers [83]	Design annotations to improve API discoverability	2017
44	López-Fernández et al. [56]	Designing and evaluating the usability of an API for real-time multimedia services in the Internet	2017
45	Acar et al. [6]	Comparing the usability of cryptographic apis	2017
46	Murphy-Hill et al. [62]	Discovering API Usability Problems at Scale	2018
47	Mosqueira-Rey et al. [60]	A systematic approach to API usability: Taxonomy-derived criteria and a case study	2018

Table C.7: Studies included in the review - Part 1 (UQ = Unique Papers)

Object of Study	**Evaluation Method**	**n**	**Aim of the Study**	**Papers**
API Spec.	Metrics	5	New Approach	Ratiu and Jurjens [77],Scheller and Kühn [87], Rama and Kak [76], Hou et al. [43], Stylos and Myers [92]
(UQ=17)	(UQ=8)	3	SW Tool	Rama and Kak [76], de Souza and Bentolila [24],Murphy-Hill et al. [62]
36%		2	API Evaluation Study	Scheller and Kühn [87], Bore and Bore [16]
	Reviews (UQ=9)	7	New Approach	Watson [95], Farooq and Zirkler [30], Faulring et al. [31],Grill et al. [37], Lee et al. [55] ,Macvean et al. [57] Mosqueira-Rey et al. [60]
		2	Fund. Results	Faulring et al. [31], Cavé et al. [21]
		2	API Evaluation Study	Jugel [47],Cavé et al. [21]

[5] Alain Abran, Adel Khelifi, Witold Suryn, and Ahmed Seffah. Usability meanings and interpretations in ISO standards. *Software Quality Journal*, 11(4):325–338, 2003.

[6] Yasemin Acar, Michael Backes, Sascha Fahl, Simson Garfinkel, Doowon Kim, Michelle L Mazurek, and Christian Stransky. Comparing the usability of cryptographic APIs. In *Security and Privacy (SP), 2017 IEEE Symposium on*, pages 154–171. IEEE, 2017.

[7] David Alonso-Ríos, Ana Vázquez-García, Eduardo Mosqueira-Rey, and Vicente Moret-Bonillo. Usability: a critical analysis and a taxonomy. *International Journal of Human-Computer Interaction*, 26(1):53–74, 2009.

[8] Rajendra K. Bandi, Vijay K. Vaishnavi, and Daniel E. Turk. Predicting maintenance performance using object-oriented design complexity metrics. *Software Engineering, IEEE Transactions on*, 29(1):77–87, 2003.

[9] Jack Beaton, Sae Young Jeong, Yingyu Xie, Jeffrey Stylos, and Brad A Myers. Usability challenges for enterprise service-oriented architec-

Table C.8: Studies included in the review - Part 2 (UQ = Unique Papers)

Object of Study	**Evaluation Method**	**n**	**Aim of the Study**	**Papers**
API Use (UQ=34) 72%	Usability Testing,	7	New Approach	Grill et al. [37], Clarke [22], O'Callaghan [69], Gerken et al. [34], Piccioni et al. [72], Bhaskar et al. [10], Mosqueira-Rey et al. [60]
	Walkthrough (UQ=17)	7	Fund. Results	McLellan et al. [59], Ko and Riche [52], Duala-Ekoko and Robillard [27], Piccioni et al. [72], Clarke [22], Stylos and Clarke [91], Stylos et al. [93]
		9	API Evaluation Study	Stylos et al. [93],Beaton et al. [9], Stylos and Clarke [91], Piccioni et al. [72], Brunet et al. [17], Ko and Riche [52], Cardoso and José [20], Stylos et al. [94], Diprose et al. [25]
	Controlled	2	New Approach	Rauf et al. [79], Nanz et al. [63]
	Experiments	9	Fund. Results	Mayer et al. [58], Scheller and Kühn [87], Endrikat et al. [29], Spiza and Hanenberg [89], Fischer and Hanenberg [33],Ellis et al. [28],Rauf et al. [79], Stylos and Myers [92], Nasehi and Maurer [64]
	(UQ=13)	7	API Evaluation Study	Murphy-Hill [61],Mayer et al. [58], Scheller and Kühn [87] ,Endrikat et al. [29], Spiza and Hanenberg [89], Santos and Myers [83], Acar et al. [6]
	Surveys and Repository	1	API Evaluation Study	López-Fernández et al. [56]
	Mining (UQ=5)	4	Fund. Results	Robillard and Deline [81], Hou and Li [42], Zibran et al. [101],Murphy-Hill et al. [62]

Table D.9: API usability Evaluation methods applied at Development Phases

Phase	Papers	#
Design	Lee et al. [55], Macvean et al. [57]	2
Development	Clarke [22],O'Callaghan [69],Farooq and Zirkler [30],Ratiu and Jurjens [77],Watson [95],Farooq and Zirkler [30], Faulring et al. [31],Bhaskar et al. [10]	8
Post-development	Watson [95],Clarke [22],Farooq and Zirkler [30],Stylos et al. [93],McLellan et al. [59],Bore and Bore [16],Murphy-Hill [61],Stylos and Clarke [91],Ellis et al. [28],Stylos and Myers [92],Hou et al. [43],Beaton et al. [9], Stylos et al. [94],de Souza and Bentolila [24],Jugel [47],Cavé et al. [21],Farooq and Zirkler [30],Nasehi and Maurer [64],Gerken et al. [34],Hou and Li [42], Brunet et al. [17],Zibran et al. [101],Ko and Riche [52],Robillard and Deline [81],Mayer et al. [58],Grill et al. [37],Duala-Ekoko and Robillard [27],Nanz et al. [63], Piccioni et al. [72],Rama and Kak [76],Cardoso and José [20],Diprose et al. [25],Spiza and Hanenberg [89],Endrikat et al. [29],Fischer and Hanenberg [33], Scheller and Kühn [87],Rauf et al. [79],Santos and Myers [83],López-Fernández et al. [56],Acar et al. [6],Murphy-Hill et al. [62], Mosqueira-Rey et al. [60]	42

Table E.10: Factors Addressed in Primary Studies

Factors	%	Papers
Learnability	49%	McLellan et al. [59],Clarke [22],Stylos and Myers [92],Hou et al. [43],Beaton et al. [9],Watson [95], Farooq and Zirkler [30],Nasehi and Maurer [64],O'Callaghan [69], Ko and Riche [52], Gerken et al. [34] , Robillard and Deline [81], Zibran et al. [101],Hou and Li [42], Faulring et al. [31] Duala-Ekoko and Robillard [27], Grill et al. [37], Rama and Kak [76], Piccioni et al. [72] , Cardoso and José [20], Scheller and Kühn [87], Rauf et al. [79], López-Fernández et al. [56],
Efficiency	23%	Murphy-Hill [61], Ellis et al. [28],Stylos et al. [94], Farooq and Zirkler [30], Mayer et al. [58], Grill et al. [37], Endrikat et al. [29], Spiza and Hanenberg [89], Fischer and Hanenberg [33], Scheller and Kühn [87] ,Santos and Myers [83]
Understandability	16%	Ratiu and Jurjens [77], O'Callaghan [69],Ko and Riche [52],Piccioni et al. [72], Cardoso and José [20],Nanz et al. [63], López-Fernández et al. [56], de Souza and Bentolila [24]
Effectiveness	8%	Brunet et al. [17],Cardoso and José [20],Bhaskar et al. [10],Acar et al. [6]

Table E.11: Factors Addressed in Primary Studies

Factors	%	Papers
Readability	8%	Stylos and Clarke [91],Jugel [47],Cavé et al. [21] ,Grill et al. [37],
Satisfaction	8%	McLellan et al. [59], Scheller and Kühn [87], Macvean et al. [57] , Rauf et al. [79],Acar et al. [6]
Memorability	4%	Rama and Kak [76],Mosqueira-Rey et al. [60]
Debuggability	4%	Stylos and Clarke [91], Nanz et al. [63]
Productivity	4%	Farooq and Zirkler [30], Fischer and Hanenberg [33]
Reusability	4%	López-Fernández et al. [56], Piccioni et al. [72]
Zibran's GL	4%	Zibran et al. [101],Grill et al. [37]
Bloch' GL	2%	Grill et al. [37]
Cognitive Analysis:		
- CDF [22]	9%	Clarke [22],Stylos and Clarke [91],Stylos et al. [94], Piccioni et al. [72],
- CDF [13]	9%	Stylos et al. [93], Beaton et al. [9], Diprose et al. [25], Bhaskar et al. [10],
- Conceptual F/W [32]	2%	Hou and Li [42]
- CWT [12]	2%	Faulring et al. [31]
- ECT [11]	2%	Rauf et al. [79]

ture APIs. In *Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on*, pages 193–196. IEEE, 2008.

[10] Rahul Kamal Bhaskar, Craig Anslow, John Brosz, and Frank Maurer. Developing usable APIs with XP and cognitive dimensions. In *Visual Languages and Human-Centric Computing (VL/HCC), 2016 IEEE Symposium on*, pages 101–105. IEEE, 2016.

[11] Anol Bhattacherjee. Understanding information systems continuance: an expectation-confirmation model. *MIS quarterly*, pages 351–370, 2001.

[12] Marilyn Hughes Blackmon, Peter G Polson, Muneo Kitajima, and Clayton Lewis. Cognitive walkthrough for the web. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 463–470. ACM, 2002.

[13] Alan F Blackwell and Thomas RG Green. A Cognitive Dimensions questionnaire optimised for users. In *Proceedings of the Twelfth Annual Meeting of the Psychology of Programming Interest Group*, pages 137–152, 2000.

[14] Joshua Bloch. How to design a good API and why it matters. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 506–507. ACM, 2006.

[15] Michael K Bond. *Understanding Security APIs.* PhD thesis, University of Cambridge, 2004.

[16] C Bore and S Bore. Profiling software API usability for consumer electronics. In *Consumer Electronics, 2005. ICCE. 2005 Digest of Technical Papers. International Conference on*, pages 155–156. IEEE, 2005.

[17] Joao Brunet, Dalton Serey, and Jorge Figueiredo. Structural conformance checking with design tests: An evaluation of usability and scalability. In *Software Maintenance (ICSM), 2011 27th IEEE International Conference on*, pages 143–152. IEEE, 2011.

[18] Chris Burns, Jennifer Ferreira, Theodore D Hellmann, and Frank Maurer. Usable results from the field of API usability: A systematic mapping and further analysis. In *Visual Languages and Human-Centric Computing (VL/HCC), 2012 IEEE Symposium on*, pages 179–182. IEEE, 2012.

[19] Cinzia Cappiello, Florian Daniel, and Maristella Matera. A quality model for mashup components. In *Web Engineering*, pages 236–250. Springer, 2009.

[20] Jorge Cardoso and Rui José. Evaluation of a programming toolkit for interactive public display applications. In *Proceedings of the 12th International Conference on Mobile and Ubiquitous Multimedia*, page 6. ACM, 2013.

[21] Vincent Cavé, Zoran Budimlić, and Vivek Sarkar. Comparing the usability of library vs. language approaches to task parallelism. In *Evaluation and Usability of Programming Languages and Tools*, page 9. ACM, 2010.

[22] Steven Clarke. Measuring API usability. *Doctor Dobbs Journal*, 29(5): S1–S5, 2004.

[23] Krzysztof Cwalina and Brad Abrams. *Framework design guidelines: conventions, idioms, and patterns for reusable. net libraries.* Pearson Education, 2008.

[24] Cleidson RB de Souza and David Leonardo M Bentolila. Automatic evaluation of API usability using complexity metrics and visualizations. In *Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, pages 299–302. IEEE, 2009.

[25] James P Diprose, Beryl Plimmer, Bruce A MacDonald, and John G Hosking. A human-centric API for programming socially interactive robots. In *Visual Languages and Human-Centric Computing (VL/HCC), 2014 IEEE Symposium on*, pages 121–128. IEEE, 2014.

[26] Andre Doucette. On API usability: An analysis and an evaluation tool. *CMPT816-Software Engineering, Saskatoon, Saskatchewan, Canada: University of Saskatchewan*, 2008.

[27] Ekwa Duala-Ekoko and Martin P Robillard. Asking and answering questions about unfamiliar APIs: an exploratory study. In *Proceedings of the 2012 International Conference on Software Engineering*, pages 266–276. IEEE Press, 2012.

[28] Brian Ellis, Jeffrey Stylos, and Brad Myers. The factory pattern in API design: A usability evaluation. In *Proceedings of the 29th international conference on Software Engineering*, pages 302–312. IEEE Computer Society, 2007.

[29] Stefan Endrikat, Stefan Hanenberg, Romain Robbes, and Andreas Stefik. How do API documentation and static typing affect API usability? In *Proceedings of the 36th International Conference on Software Engineering*, pages 632–642. ACM, 2014.

[30] Umer Farooq and Dieter Zirkler. API peer reviews: a method for evaluating usability of application programming interfaces. In *Proceedings of the 2010 ACM conference on Computer supported cooperative work*, pages 207–210. ACM, 2010.

[31] Andrew Faulring, Brad A Myers, Yaad Oren, and Keren Rotenberg. A case study of using HCI methods to improve tools for programmers. In *Cooperative and Human Aspects of Software Engineering (CHASE), 2012 5th International Workshop on*, pages 37–39. IEEE, 2012.

[32] Gerhard Fischer. Cognitive view of reuse and redesign. *IEEE Software*, 4 (4):60, 1987.

[33] Lars Fischer and Stefan Hanenberg. An empirical investigation of the effects of type systems and code completion on API usability using type-

script and javascript in ms visual studio. In *ACM SIGPLAN Notices*, volume 51, pages 154–167. ACM, 2015.

[34] Jens Gerken, 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 2011 annual conference on Human factors in computing systems*, pages 3373–3382. ACM, 2011.

[35] Peter Leo Gorski and Luigi Lo Iacono. Towards the Usability Evaluation of Security APIs. In *In Proceedings of the Tenth International Symposium on Human Aspects of Information Security and Assurance (HAISA)*, 2016.

[36] Thomas R. G. Green and Marian Petre. Usability analysis of visual programming environments: a cognitive dimensions framework. *Journal of Visual Languages & Computing*, 7(2):131–174, 1996.

[37] Thomas Grill, Ondrej Polacek, and Manfred Tscheligi. Methods towards API usability: a structural analysis of usability problem categories. In *Human-Centered Software Engineering*, pages 164–180. Springer, 2012.

[38] H Rex Hartson, Terence S Andre, and Robert C Williges. Criteria for evaluating usability evaluation methods. *International journal of human-computer interaction*, 13(4):373–410, 2001.

[39] Michi Henning. API design matters. *Queue*, 5(4):24–36, 2007.

[40] Jim Robert Highsmith. *Adaptive software development: a collaborative approach to managing complex systems.* Addison-Wesley, 2013.

[41] Mohammad Alamgir Hossain and Mohammed Quaddus. Expectation–confirmation theory in information system research: A review and analysis. In *Information systems theory*, pages 441–469. Springer, 2012.

[42] Daqing Hou and Lin Li. Obstacles in using frameworks and APIs: An exploratory study of programmers' newsgroup discussions. In *Program*

Comprehension (ICPC), 2011 IEEE 19th International Conference on, pages 91–100. IEEE, 2011.

[43] Daqing Hou, Chandan Raj Rupakheti, and H James Hoover. Documenting and evaluating scattered concerns for framework usability: A case study. In *Software Engineering Conference, 2008. APSEC'08. 15th Asia-Pacific*, pages 213–220. IEEE, 2008.

[44] ISO. Ergonomic requirements for office work with visual display terminals (VDTs). 1998.

[45] ISO/IEC. Software engineering - Product quality - Part 3: Internal metrics. 2003.

[46] Tina Jones and David Evans. Conducting a systematic review. *Australian Critical Care*, 13(2):66–71, 2000.

[47] Uwe Jugel. Generating smart wrapper libraries for arbitrary APIs. In *International Conference on Software Language Engineering*, pages 354–373. Springer, 2010.

[48] Brian W Kernighan. *The practice of programming.* Addison-Wesley Professional, 1999.

[49] Brian W Kernighan and Phillip James Plauger. The elements of programming style. *The elements of programming style, by Kernighan, Brian W.; Plauger, PJ New York: McGraw-Hill, c1978.*, 1, 1978.

[50] Barbara A Kitchenham, David Budgen, and O Pearl Brereton. Using mapping studies as the basis for further research–a participant-observer case study. *Information and Software Technology*, 53(6):638–651, 2011.

[51] Sebastian Kleinschmager, Romain Robbes, Andreas Stefik, Stefan Hanenberg, and Eric Tanter. Do static type systems improve the maintainability of software systems? an empirical study. In *Program Comprehension*

(ICPC), 2012 IEEE 20th International Conference on, pages 153–162. IEEE, 2012.

[52] Andrew J Ko and Yann Riche. The role of conceptual knowledge in API usability. In *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on*, pages 173–176. IEEE, 2011.

[53] Panayiotis Koutsabasis, Thomas Spyrou, and John Darzentas. Evaluating usability evaluation methods: criteria, method and a case study. *Human-Computer Interaction. Interaction Design and Usability*, pages 569–578, 2007.

[54] Arnaud Lauret. *The Design of Everyday APIs.* Manning Publications Co., 2019. ISBN 9781617295102.

[55] Sunghoon Lee, Sanghee Lee, Sumi Lim, Jiyoung Jung, Sangho Choi, Neunghoe Kim, and Jung-Been Lee. An API Design Process in Terms of Usability: A Case Study on Building More Usable APIs for Smart TV Platform. In *Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International*, pages 567–571. IEEE, 2014.

[56] Luis López-Fernández, Boni García, Micael Gallego, and Francisco Gortázar. Designing and evaluating the usability of an API for real-time multimedia services in the Internet. *Multimedia Tools and Applications*, 76(12):14247–14304, 2017.

[57] Andrew Macvean, Martin Maly, and John Daughtry. API Design Reviews at Scale. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, pages 849–858. ACM, 2016.

[58] Clemens Mayer, Stefan Hanenberg, Romain Robbes, Éric Tanter, and Andreas Stefik. An empirical study of the influence of static type systems on the usability of undocumented software. In *ACM SIGPLAN Notices*, volume 47, pages 683–702. ACM, 2012.

[59] Samuel G McLellan, Alvin W Roesler, Joseph T Tempest, and Clay I Spinuzzi. Building more usable APIs. *IEEE software*, 15(3):78–86, 1998.

[60] Eduardo Mosqueira-Rey, 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*, 97:46–63, 2018.

[61] Emerson Murphy-Hill. Improving usability of refactoring tools. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pages 746–747. ACM, 2006.

[62] Emerson Murphy-Hill, Caitlin Sadowski, Andrew Head, John Daughtry, Andrew Macvean, Ciera Jaspan, and Collin Winter. Discovering API Usability Problems at Scale. 2018.

[63] Sebastian Nanz, Faraz Torshizi, Michela Pedroni, and Bertrand Meyer. Design of an empirical study for comparing the usability of concurrent programming languages. *Information and Software Technology*, 55(7): 1304–1315, 2013.

[64] Seyed Mehdi Nasehi and Frank Maurer. Unit tests as API usage examples. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–10. IEEE, 2010.

[65] Phu H Nguyen, Max Kramer, Jacques Klein, and Yves Le Traon. An extensive systematic review on the Model-Driven Development of secure systems. *Information and Software Technology*, 68:62–81, 2015.

[66] Jakob Nielsen. *Usability engineering.* Elsevier, 1994.

[67] Jakob Nielsen and Hoa Loranger. *Prioritizing web usability.* Pearson Education, 2006.

[68] Kristian Nybom, Adnan Ashraf, and Ivan Porres. A Systematic Mapping Study on Tools for API Documentation Generation. Technical Report 1180, 2017.

[69] Portia O'Callaghan. The API walkthrough method: a lightweight method for getting early feedback about an API. In *Evaluation and Usability of Programming Languages and Tools*, page 5. ACM, 2010.

[70] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058, December 1972. ISSN 0001-0782.

[71] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1–18, 2015.

[72] Marco Piccioni, Carlo A Furia, and Bertrand Meyer. An empirical study of API usability. In *Empirical Software Engineering and Measurement, 2013 ACM/IEEE International Symposium on*, pages 5–14. IEEE, 2013.

[73] Lutz Prechelt and Martin Liesenberg. Design patterns in software maintenance: An experiment replication at freie universität berlin. In *Replication in Empirical Software Engineering Research (RESER), 2011 Second International Workshop on*, pages 1–6. IEEE, 2011.

[74] Lutz Prechelt, Barbara Unger-Lamprecht, Michael Philippsen, and Walter F Tichy. Two controlled experiments assessing the usefulness of design pattern documentation in program maintenance. *IEEE Transactions on Software Engineering*, 28(6):595–606, 2002.

[75] Whitney Quesenbery. Balancing the 5Es of Usability. *Cutter IT Journal*, 17(2):4–11, 2004.

[76] Girish Maskeri Rama and Avinash Kak. Some structural measures of API usability. *Software: Practice and Experience*, 2013.

[77] Daniel Ratiu and Jan Jurjens. Evaluating the reference and representation of domain concepts in APIs. In *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on*, pages 242–247. IEEE, 2008.

[78] Daniel Petrica Ratiu. *Intentional meaning of programs.* PhD thesis, Technical University Munich, 2009.

[79] Irum Rauf, Pekka Perälä, Jouni Huotari, and Ivan Porres. Perceived obstacles by novice developers adopting user interface APIs and tools. In *Visual Languages and Human-Centric Computing (VL/HCC), 2016 IEEE Symposium on*, pages 223–227. IEEE, 2016.

[80] Martin P Robillard. What makes APIs hard to learn? answers from developers. *Software, IEEE*, 26(6):27–34, 2009.

[81] Martin P Robillard and Robert Deline. A field study of API learning obstacles. *Empirical Software Engineering*, 16(6):703–732, 2011.

[82] Martin P Robillard, Eric Bodden, David Kawrykow, Mira Mezini, and Tristan Ratchford. Automated API property inference techniques. *Software Engineering, IEEE Transactions on*, 39(5):613–637, 2013.

[83] André L Santos and Brad A Myers. Design annotations to improve API discoverability. *Journal of Systems and Software*, 126:17–33, 2017.

[84] Thomas Scheller and Eva Kühn. Measurable concepts for the usability of software components. In *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, pages 129–133. IEEE, 2011.

[85] Thomas Scheller and Eva Kühn. Influencing factors on the usability of API classes and methods. In *Engineering of Computer Based Systems (ECBS), 2012 IEEE 19th International Conference and Workshops on*, pages 232–241. IEEE, 2012.

[86] Thomas Scheller and Eva Kühn. Usability Evaluation of Configuration-Based API Design Concepts. In *Human Factors in Computing and Informatics*, pages 54–73. Springer, 2013.

[87] Thomas Scheller and Eva Kühn. Automated measurement of API usability: The API concepts framework. *Information and Software Technology*, 61:145–162, 2015.

[88] Ahmed Seffah, Mohammad Donyaee, Rex B Kline, and Harkirat K Padda. Usability measurement and metrics: A consolidated model. *Software Quality Journal*, 14(2):159–178, 2006.

[89] Samuel Spiza and Stefan Hanenberg. Type names without static type checking already improve the usability of APIs (as long as the type names are correct): An empirical study. In *Proceedings of the 13th international conference on Modularity*, pages 99–108. ACM, 2014.

[90] Jeffrey Stylos. *Making APIs more usable with improved API designs, documentation and tools*. ProQuest, 2009.

[91] Jeffrey Stylos and Steven Clarke. Usability implications of requiring parameters in objects' constructors. In *Proceedings of the 29th international conference on Software Engineering*, pages 529–539. IEEE Computer Society, 2007.

[92] Jeffrey Stylos and Brad A Myers. The implications of method placement on API learnability. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 105–112. ACM, 2008.

[93] Jeffrey Stylos, Steven Clarke, and Brad Myers. Comparing API design choices with usability studies: A case study and future directions. In *Proceedings of the 18th PPIG Workshop*, 2006.

[94] Jeffrey Stylos, Benjamin Graf, Daniela K Busse, Carsten Ziegler, Ralf Ehret, and Jan Karstens. A case study of API redesign for improved

usability. In *Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on*, pages 189–192. IEEE, 2008.

[95] Robert B Watson. Improving software API usability through text analysis: A case study. In *Professional Communication Conference, 2009. IPCC 2009. IEEE International*, pages 1–7. IEEE, 2009.

[96] Chamila Wijayarathna, Nalin AG Arachchilage, and Jill Slay. A Generic cognitive dimensions questionnaire to evaluate the usability of security APIs. In *International Conference on Human Aspects of Information Security, Privacy, and Trust*, pages 160–173. Springer, 2017.

[97] Sebastian Winter, Stefan Wagner, and Florian Deissenboeck. A comprehensive model of usability. In *IFIP International Conference on Engineering for Human-Computer Interaction*, pages 106–122. Springer, 2007.

[98] Ian H Witten, Eibe Frank, Mark A Hall, and Christopher J Pal. *Data Mining: Practical machine learning tools and techniques.* Morgan Kaufmann, 2016.

[99] Claes Wohlin and Rafael Prikladniki. Systematic literature reviews in software engineering. *Information and Software Technology*, 55(6):919–920, 2013.

[100] M Zibran. What makes APIs difficult to use. *IJCSNS International Journal of Computer Science and Network Security*, 8(4):255, 2008.

[101] Minhaz F Zibran, Farjana Zebin Eishita, and Chanchal K Roy. Useful, but usable? factors affecting the usability of APIs. In *Reverse Engineering (WCRE), 2011 18th Working Conference on*, pages 151–155. IEEE, 2011.