API Deprecation: A Systematic Mapping Study

Leif Bonorden
Universität Hamburg
Hamburg, Germany
leif.bonorden@uni-hamburg.de

Matthias Riebisch
Universität Hamburg
Hamburg, Germany
matthias.riebisch@uni-hamburg.de

***Abstract*—Application Programming Interfaces (APIs) are the prevalent interaction method for software modules, components, and systems. As systems and APIs evolve, an API element may be marked as deprecated, indicating that its use is disapproved or that the feature will be removed in an upcoming version. Consequently, deprecation is a means of communication between developers and, ideally, complemented by further documentation, including suggestions for the developers of the API's clients.**

API deprecation is a relatively young research area that recently gained traction among researchers. To identify the current state of research as well as to identify open research areas, a meta-study that assesses scientific studies is necessary. Therefore, this paper presents a systematic mapping study on API deprecation to classify the state of the art and identify gaps in the research field. We identified and mapped 36 primary studies into a classification scheme comprising general and API-specific categories.

We identified five major gaps in previous research on API deprecation as opportunities for future studies: studying remote APIs, investigating a broader range of static APIs, joining suppliers' and clients' views, including humans in studies, and avoiding deprecation.

***Index Terms*—API Deprecation, Systematic Mapping Study, API Evolution**

I. Introduction

A. Motivation

As modern software architecture favors the decomposition of systems into more and smaller independent parts, the relations between these parts become more critical. The prevalent way to implement such a relation is an application programming interface (API). As software systems evolve, their APIs change as well. During the evolution of APIs, elements may be deprecated, indicating that their use is disapproved or that they will be removed in upcoming versions—posing the risk of breaking functionality in the API's clients.

Since research on API deprecation is still relatively young and gained traction recently, classification and coordination of research results and efforts is needed—which this meta-study addresses. First, we support planning and reporting on API deprecation research by providing a classification scheme. Second, we identify research gaps and highlight opportunities.

In addition, recent reporting on the state of the practice claimed that research on APIs does not reflect the diversity of APIs in practice. In particular, research on remote APIs is demanded [1]. In response, this meta-study evaluates the coverage of API types in research and validates the claim.

B. Contribution

This study investigates the state of research on API deprecation by performing a systematic mapping study identifying 103 unique studies. The main contributions are:

- A catalog of 36 relevant primary studies composing the state of research.
- A classification scheme for research on API deprecation.
- A map of current research and a gap analysis highlighting five major research gaps and opportunities.
- A replication package enabling further investigation [2].

The contribution is intended to benefit researchers conducting further studies on the topic, particularly supporting the direction of future research towards open issues.

II. Background

A. Application Programming Interfaces

APIs enable the cooperation of software systems using the means of programming languages. Since the term was introduced in the 1960s [3], it has gained importance through the division of complex software systems into modules [4]. Nowadays, APIs have become the prevalent interaction method to reuse software components, communicate via networks, and provide data for others [5].

Two essentially different types of APIs are distinguished [1]: **Static APIs** denote APIs in libraries, frameworks, or software development kits that may be statically linked—also referred to as *traditional APIs* or *programming language APIs*. In contrast, **remote APIs** use means of network communication to interact—also referred to as *web APIs*. With the advent of containerization and microservices, remote APIs are no longer limited to web applications but are becoming a ubiquitously used type of API—even if the system's components are deployed on the same machine or cloud service.

The main stakeholders concerned with APIs are its **suppliers**—*API developers*—and its **clients**—*API users*.

As software systems evolve, their APIs need to be adapted accordingly: Suppliers add, change or remove elements in the API, and clients need to adjust their code to the new version. Versioning of APIs may be used to avoid sudden changes. Furthermore, clients may be supported by providing suitable alternatives or by tools automatically updating code.

B. API Deprecation

API suppliers may mark elements of an API as *deprecated*, signaling its clients that the use is discouraged or that the

element will be removed in a future version. Depending on the type of API and the environment it resides in, an appropriate prebuilt deprecation feature exists. Otherwise or additionally, the supplier needs to find other means to document the deprecation and communicate it to its clients.

A common usage scenario for API deprecation is API evolution. Deprecation is used to warn about an upcoming change that could break the clients' code: The affected API element is marked as deprecated, and ideally, a schedule announcing the element's removal and a suggestion for an alternative solution are provided. The client is given a reasonable amount of time to adapt to the changing API element before the change breaks the client's code.

Additionally, API deprecation may also be used without the intention to change the API element: The element's usage may be discouraged, or caution needs to be exercised in its usage, even though it remains part of the API—e.g., the supplier does not guarantee thread safety for a method.

The Java Language Specification [6] provides a typical definition of deprecation:

> "Programmers are sometimes discouraged from using certain program elements (modules, classes, interfaces, fields, methods, and constructors) because they are considered dangerous or because a better alternative exists. The annotation interface `Deprecated` allows a compiler to warn about uses of these program elements."

The supplier of a Java API should use this annotation to mark an element as deprecated. Additionally, they should provide further explanation and advice in the accompanying documentation Thus, the client's Java compiler can now warn about the deprecation if the annotated element is used.

C. Related Work

A previous systematic review of API evolution by Lamothe et al. [7] mentions deprecation but does not consider it separately. The systematic review includes 369 publications, of which most deal with the Android ecosystem or study the Java programming language.

Further systematic studies on APIs cover usability [5], [8] or documentation [9], [10], but none of these include the topic of API deprecation.

In a study on API misuse, Bonifácio et al. [11] discuss related work on API evolution and identify two general directions for research: The first goal is to "help developers to migrate their systems in response to the evolution of APIs". The second focus lies on "understanding how developers evolve APIs and on characterizing the evolution of APIs." These directions relate to perspectives on clients and suppliers, respectively.

The state of the practice on API evolution has been studied by Raatikainen et al., focusing on the suppliers' perspective [1]: Remote APIs are identified as prevalent in industrial projects, while static APIs are still in use. Furthermore, API evolution itself is not considered a particularly challenging topic, but the manual effort to avoid the unintended breaking of APIs is high. Finally, it is argued that the diversity in API types is not reflected in current research.

III. Research Method

We conduct a *systematic mapping study* following the methodology for systematic mapping studies in software engineering by Petersen et al. [12], [13]. Additionally, we heed the empirical standard for *systematic reviews* from the ACM SIGSOFT Paper and Peer Review Quality Initiative [14].

A systematic mapping study method rigorously reviews primary studies on a research topic. It characterizes the state of the art of research on this topic and allows the identification of research gaps and opportunities for further primary studies.

Figure 1 provides an overview of the systematic mapping process and the interim numbers of studies in each step. Each study's assessment was performed by one researcher and the result was discussed among the authors as described by [15].

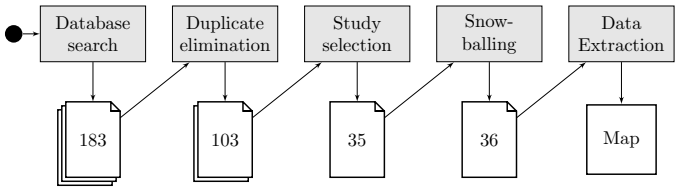

Fig. 1. Systematic mapping process

A. Research Questions

First, we aim to map the field of API deprecation research according to the Who-What-How research strategy framework for software engineering [16]:

RQ1 Who are the **beneficiaries** of research on API deprecation?

RQ2 What are the **types of contribution** in research on API deprecation?

RQ3 Which **research strategies** are used in research on API deprecation?

These criteria are derived deductively—i.e., originating in the Who-What-How framework—and allow for a general classification of software engineering research, e.g., distinguishing analyses describing the status quo from results proposing new solutions.

Second, we add classifications specific to API deprecation:

RQ4 What **types of APIs** are the subject of research on API deprecation?

RQ5 What **aspects of deprecation** are considered in research on API deprecation?

These criteria are constructed inductively—i.e., built from observations in this study—and provide a more specific, thus more detailed classification for research on API deprecation.

B. Search Strategy

We selected five academic databases to search for relevant studies: *ACM Digital Library*, *IEEE Xplore*, *Microsoft Academic*, *Elsevier ScienceDirect*, and *Web Of Science*. We

omitted *Springer* and *Wiley Intersciences* since their publications are included in *Microsoft Academic* and *Web of Science*, respectively.

We did not consider gray literature for this secondary study since the goal is to classify and map published research—in contrast to practitioners' experience and needs. Furthermore, the properties addressed in RQ1–3 are hardly applicable to gray literature; thus, the inclusion of gray literature would risk the study's validity and soundness. However, gray literature needs to be considered in future research on API deprecation to widen the view and connect academia and practice.

Since the topics of API deprecation and API evolution are closely related, we decided to pursue a relatively broad strategy and did not separate them. Instead, we set the appearance of "API deprecation" or "deprecated API" as the minimal requirement for relevance; thus, possibly including studies on the more general topic of API evolution. Consequently, we chose `api AND deprecat*` as the generic search string which ought to appear in title, abstract, or keywords. Depending on the databases' available search settings, we adapted the generic search string as shown in Table I. For databases that did not offer the desired settings exactly, we widened the search—including all metadata (including title, abstract, and keywords) or full text. The inquiries were last updated on October 1st, 2021.

In total, the results comprised 183 studies. We used the *EPPI-Reviewer* software [17] to collect these results and perform the following steps. Identifying 80 duplicates, 103 unique studies remained for screening.

C. Study Selection

To select only studies relevant to API deprecation, we defined *inclusion and exclusion criteria*, which we checked using adaptive reading depth: First, for each study, title, abstract, and keywords were checked. If no confident decision could be made, the full text was screened to avoid false negatives.

Any selected study must fulfill all of the following *inclusion criteria*:

- I1 **Publication type.** The study was published in a journal or at a workshop or conference.
- I2 **Research topic.** The study discusses API deprecation.
- I3 **Language.** The study is published in English.

Criterion *I1* subsumes any track at a conference—e.g., tool demonstration or doctoral symposium—and early access publications—i.e., ahead-of-time publication by the conference or journal. These inclusion criteria achieve the selection of all research contributions on API deprecation without further restrictions.

On the other hand, a study was excluded if any of the following *exclusion criteria* apply:

- E1 **Publication type.** The study is not peer-reviewed—e.g., a preprint or a masters' thesis.
- E2 **Research subject.** The study does not research API deprecation—e.g., deprecation is only mentioned as a related topic.
- E3 **Contribution.** The study does not present any new insights on deprecation—e.g., presentation of a research idea.
- E4 **Extension.** The study is included in another publication that is considered in this systematic mapping study instead—e.g., a conference article extended in a journal.

These exclusion criteria guarantee that only peer-reviewed results on API deprecation are included and ensure that no single investigation is included multiple times.

The selection excludes 68 studies following these criteria, yielding 35 studies for further classification.

D. Verification of the Search Process

To verify the choice of databases and search strings, we selected 5 studies we were already familiar with before conducting the systematic search [18]–[22]. The studies provide different views of API deprecation and are written by various authors. Our search strategy successfully identified all of these studies, indicating field coverage.

Furthermore, we applied forward and backward snowballing to the identified primary studies [23]: First, we collected the references and citations using the Semantic Scholar Academic Graph API [24]. Second, we eliminated duplicates and removed the studies already identified in the search process, yielding 1033 further publications. Again, we assessed these using adaptive reading depth, identifying 1 more study [25] to include. Finally, we repeated the snowballing process for this newly identified study, yielding 9 further publications—of which none met the criteria. In conclusion, we obtained a set of 36 relevant primary studies and a total of 1042 related works. A corresponding list is given in the data set [2].

E. Construction of the Classification Scheme

We used the generic Who-What-How research strategy framework for software engineering [16] to deductively outline a classification scheme along RQ1–3. Subsequently, we extended the classification scheme inductively for RQ4–5 by keywording of abstracts as proposed by Petersen et al. [12].

1) Beneficiaries ("Who?", RQ1): Software engineering research may benefit one or more of the following beneficiaries. If such information is given explicitly, we only consider the beneficiary stated by the study. Otherwise, we infer the data from the study's motivation or goal.

- **Human stakeholders**: The study claims a contribution for software developers or other people concerned with the development of software.
- **Technical systems**: The study's goal is the improvement of a tool or software system.
- **Researchers**: The study's results assist in further research.

2) Contribution ("What?", RQ2): Software engineering research may contribute different types of results. We consider studies aiming to understand the status quo, solve a problem, or both.

- **Descriptive Knowledge**: The study is an *analysis* of a phenomenon or problem yielding a better understanding.

TABLE I
INDIVIDUAL INQUIRIES

Source	Search String	Scope	# of Results
ACM Digital Library	"api deprecation"	Full text	22
ACM Digital Library	"deprecated api"	Full text	46
IEEE Xplore	api AND deprecation	All metadata	13
IEEE Xplore	api AND deprecated	All metadata	26
Microsoft Academic	"api deprecation"	Full text	18
Microsoft Academic	"deprecated api"	Full text	19
Science Direct	api AND deprecation	Title, abstract, keywords	4
Science Direct	api AND deprecated	Title, abstract, keywords	6
Web of Science	api AND deprecat*	Title, abstract, keywords	32

- **Solution**: The study introduces new means to solve a problem.

Furthermore, we inductively refined the *solutions* and found concrete **tools**, general **methods** and proposed **classification schemes**.

3) Strategy ("How?", RQ3): The research strategy model [16] identifies four empirical and one non-empirical category of research strategies. A study may apply multiple research strategies.

- **Field**: Researchers observe a socio-technical system without (*field study*) or with little (*field experiment*) intervention in its natural setting.
- **Lab**: Researchers let human actors perform activities in a highly controlled neutral setting (*lab experiment*) or setting mimicking a natural environment (*experimental simulation*).
- **Respondent**: Researchers gather insights from experts with a survey or interview (*sample survey*) or collect feedback on a new solution (*judgment study*).
- **Data**: Researchers study a phenomenon without involving human participants by analyzing data (*data mining study*) or performing benchmarks (*data experiment*).
- **Non-empirical**: Researchers analyze existing research contributions (*meta-studies*) or provide mathematical proofs (*formal methods*).

While meta-studies may be viewed as empirical investigations on a meta-level, we follow the terminology presented by the research strategy model and use the label 'non-empirical'.

4) API Type (RQ4): The two types of APIs—static APIs and remote APIs—are essentially distinct. We identified specific instances of these types from the studies' abstracts.

- **Static APIs**: The study is concerned with APIs that may be accessed within a single programming language, including libraries and frameworks, and elements provided by an SDK. Since we observed several studies on the Android ecosystem, we split it from general Java APIs.
  - *Java* (not Android-specific)
  - *Android*
  - *JavaScript*
  - *Python*
  - *other* (other language or language-agnostic)
- **Remote APIs**: The study is concerned with APIs implemented by another system or component and may be invoked over a network. We only identified one type of remote APIs:
  - *REST/OpenAPI*

5) Aspect of Deprecation (RQ5): To identify and distinguish various concerns in the context of API deprecation, we used keywording [12]: First, we read the studies' abstracts and noted keywords describing stakeholders, their points of view and their activities—e.g., 'client developers', 'fault mitigation', 'refactoring'. Second, we clustered the keywords into four categories:

- **Decision (Supplier)**
- **Documentation (Supplier)**
- **Information (Client)**
- **Reaction (Client)**

Section IV-A provides a more detailed description of the arising four-step API deprecation model.

F. Data Extraction

Following the construction of the classification scheme, we extracted the relevant data from the studies' full texts to sort them accordingly: The first author extracted the data using the data extraction form in EPPI-Reviewer [17], and the second author revised the classification. All disagreements were solved by discussion, and a consensus was reached.

The resulting classification is given in section IV-B.

IV. RESULTS

A. Four-step API Deprecation Model

The first result from the research process is the classification scheme itself—in particular, the four-step model of API deprecation:

- **Decision (Supplier)**: The supplier of an API investigates a situation and decides to deprecate an API element.
- **Documentation (Supplier)**: The supplier of an API implements, documents and communicates the deprecation.
- **Information (Client)**: The client obtains knowledge of the deprecation of an API element. They may research additional documentation or request assistance.
- **Reaction (Client)**: The client updates their code or chooses to ignore the deprecation.

The order of these steps may differ, and the steps may overlap—e.g., a supplier could add documentation after a client has already reacted to a deprecation.

Furthermore, research may take another point of view than a step itself indicates—e.g., researchers may reconstruct a supplier's decision process after a decision has been made and ask clients for information influencing their reaction after it has been carried out.

B. Classification and Map

We identified and classified 36 relevant primary studies. A histogram of the selected studies' publication years (see Figure 2) shows that research on API deprecation commenced in 2005 and was mainly published since 2016 (83% of identified primary studies). An overview of the studies' classification is presented in Table II. A more detailed version is provided in the data set for this paper [2].

TABLE II
CLASSIFICATION OF PRIMARY STUDIES

Beneficiary	
Human Stakeholders	[18], [21], [22], [26]–[52]
Technical Systems	[20], [25], [53]–[55]
Researcher	[18], [19], [22], [41], [48], [49], [51]
Contribution	
Analysis	[18], [19], [21], [22], [26]–[30], [32], [36]–[38], [40]–[42], [45], [48], [49], [51], [54]
Tool	[18], [20], [27], [31], [33], [34], [39], [43], [44], [47], [50], [52], [53], [55]
Method	[25], [30], [35], [45], [46]
Classification	[30]
Strategy	
Field	[29], [53]
Lab	—
Respondent	[22], [31], [32], [42], [51]
Data	[18]–[21], [26]–[28], [30]–[40], [42]–[52], [54], [55]
Non-empirical	[25]
API Type	
Java	[19], [22], [25], [28], [31], [33], [35]–[37], [39]–[42], [44], [50]–[52]
Android	[20], [21], [27], [29], [34], [43], [45], [46], [55]
JavaScript	[32], [54]
Python	[47]–[49], [51]
other (static)	[19], [26], [30], [38], [53]
REST/OpenAPI	[18]
Aspect	
Decision (Supplier)	[22], [30], [41]
Documentation (Supplier)	[18], [19], [25], [28], [31], [32], [37], [48], [54]
Information (Client)	[32], [47]
Reaction (Client)	[20]–[22], [26], [27], [29], [32]–[36], [38]–[40], [42]–[46], [48]–[55]

a) Beneficiaries of Research (RQ1): A large majority of studies (83%) directly support *human stakeholders*, while improvements of *technical systems* (14%) and guidance for future *research* (19%) occur rarely. Multiple beneficiaries were given in 6 studies.

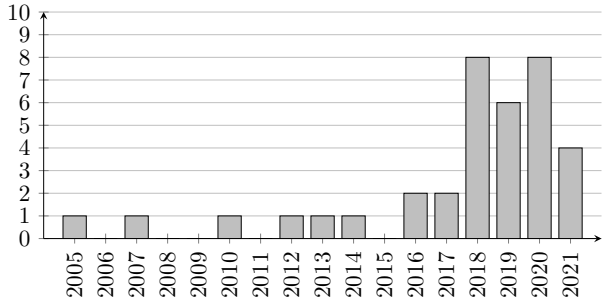

Fig. 2. Number of studies on API deprecation per year

b) Types of Contribution (RQ2): Descriptive knowledge (58%) and *solutions* (56%) are almost balanced. Within the solution providing studies, *tools* are the major contribution (39%). In 5 studies, we identified more than one contribution type.

c) Research Strategies (RQ3): The majority of studies apply a *data strategy* (86%). In contrast, *respondent strategies* (14%), *field strategies* (6%), and *non-empirical strategies* (3%) are rarely applied; *lab strategies* did not occur. Multiple research strategies were involved in 3 studies—combining *respondent* and *data strategies*. Additionally, 4 studies used *data experiments* together with *data mining studies*—two methods of the same strategy.

d) Types of API (RQ4): Research almost exclusively considered static APIs (97%); *remote* APIs are only considered in 1 study (3%). Specifically, the studies' primary focus are *Java* APIs in general (47%) and the *Android* ecosystem (25%). Other static APIs are studied less (31%).

e) Aspects of API Deprecation (RQ5): Clients' reactions to deprecation are the main subject of the selected studies (75%). On the other hand, significantly fewer studies are concerned with the suppliers of deprecated APIs (33%).

f) Map of Research on API Deprecation: We select the dimensions *aspect of deprecation*, *API type*, and *contribution type* for the research map on API deprecation, given in Figure 3. Both the left-hand and the right-hand sides of the map contain all 36 studies. Differences in numbers are attributable to the assignment of multiple categories to a single study.

V. DISCUSSION

A. State of Research on API Deprecation

Our systematic mapping study provides a high-level overview of the research area of API deprecation. Overall, we notice a focus on particular research topics and methods with few studies assessing issues outside this focus—which fits the characterization as a young and still evolving research area and provides opportunities to mature the research area by identifying and filling the gaps.

The majority of research on API deprecation addresses the reaction of clients to deprecation in Java and Android systems. These studies mostly use data research strategies to provide analyses and tools. This focus is reasonable as

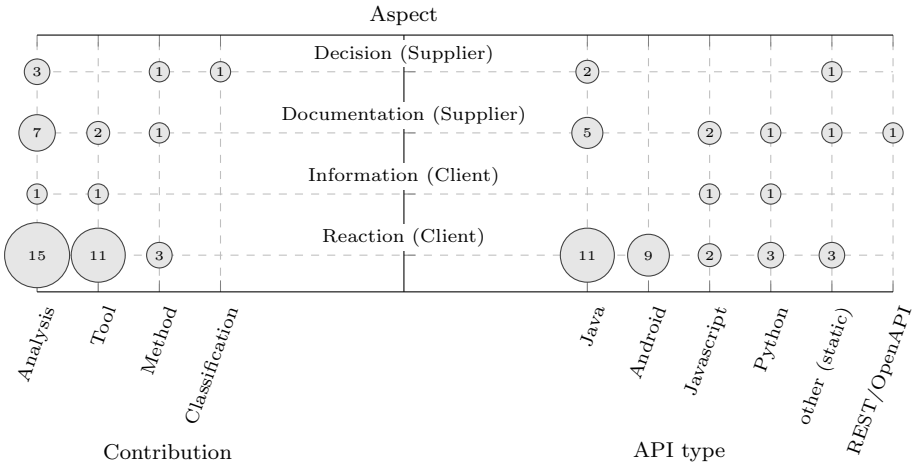

Fig. 3. Map of Research on API Deprecation

the Android ecosystem depends on the Android operating system, which follows a strict deprecation and removal policy—demanding swift reaction of all clients. In the case of Java, the mature deprecation feature allows for a thorough analysis of (open-source) projects.

Few studies address other static APIs. We attribute this to the less mature and less comprehensive deprecation features in other programming languages—including the popular Python and JavaScript [22]: Absence of a definitive way to deprecate elements may discourage developers from using deprecation and complicates research. **For remote APIs, research has just begun.** While the importance of remote APIs increases in practice [1], the variety of implementation options calls may hinder practical usage and scientific analysis of deprecation.

On the clients' side, research focuses the reaction—i.e., updating the clients' systems. **On the suppliers' side, research focuses on the documentation of deprecation**—i.e., activities after the decision to deprecate an element has been made. We attribute both of these observations to the potential of automation in practice, possibly motivating research. Furthermore, we assume that the feasibility of data research methods for these aspects facilitates research.

Most studies pursue data research strategies. These strategies offer a relatively objective and comparable measurement for the evaluation of tools and allow the inclusion of a variety of systems in data analyses. However, they lack human factors.

B. Research Gaps and Opportunities

Following the observations on the state of research, we identified gaps in previous research and opportunities for future studies. In the following, we focus on five significant observations:

a) ***Uncharted Territory****—Remote APIs:* Only 1 study researches deprecation of remote APIs [18], although remote APIs are the prevalent type of APIs in the software industry [1].

A tool for standardized specification and documentation of REST APIs exists in OpenAPI, and elements may be marked as deprecated in an OpenAPI specification. Future research needs to understand further suppliers' and clients' needs on deprecation of remote APIs as well as the current practice. Additionally, methods and tools to support the deprecation process are needed. Finally, the transferability of results on the deprecation of static APIs to remote APIs should be examined.

Furthermore, this supports the observation by Raatikainen et al., who noticed research on APIs to include remote APIs only rarely. [1]

b) ***Out of Focus****—Static APIs:* In the case of static APIs, most research has been conducted on the Java programming language and the Android ecosystem. In particular, primarily descriptive analyses have been performed for Java APIs; for the Android ecosystem, most publications present tools.

According to Sawant et al. [22], the Java deprecation feature comprises a broader range of functions than most other languages. Nonetheless, research on the deprecation of static APIs should consider more instances: More programming languages should be studied—especially those without a mature deprecation mechanism. Studies on individual programming languages or settings should be checked for transferability and generalizability.

c) ***Unbridged Gap****—Suppliers' and Clients' Views on API Deprecation:* Suppliers and clients of an API may pursue different goals in detail but share a common ground in general: The clients' software should interact correctly and efficiently with the API and the system to which it belongs.

We have identified 4 studies that regard both the suppliers' and the clients' perspectives—all of which are descriptive analyses. Future studies should better include the suppliers and their perspective in API deprecation research. In particular, both points of view should be combined for a more general understanding and assistance of the whole process.

d) ***Human-out-of-the-loop****—Methodical Triangulation of Research Strategies:* Research triangulation strategies confirm results using other research methods, including data and methodical triangulation. While data triangulation—the usage of multiple data sources—has been performed in several

studies on API deprecation, we could observe little methodical triangulation—the combination of different research strategies. In particular, most research on API deprecation is performed solely on data without the involvement of humans.

Additionally, if data research strategies are applied, studies typically inspect only open-source software.

*e) **Prevention Better Than Cure**—Avoiding API Deprecation:* Previous research focuses on understanding and reacting to API deprecation, but no studies on means to avert or simplify deprecation have been identified— i.e., no studies addressed situations before the first step of the API deprecation model. Future research should investigate which decisions lead to API deprecation and how deprecation may be avoided.

VI. Threats to Validity

Common threats to validity in software engineering secondary studies have been discussed in [56]. We address the applicable threats and our countermeasures.

A. Study Selection Validity

Completeness of study selection is the main threat for a systematic mapping study. To mitigate this threat, we decided to broaden the search as far as possible—narrowing the results relatively late after applying an adaptive reading strategy. This was possible since the number of identified studies was relatively low. We chose multiple databases to cover all major publication venues for software engineering research and verified the completeness by checking the inclusion of previously manually selected studies.

Inclusion and exclusion criteria were selected solely based on the study's intentions, and generic limitations (e.g., a restriction on the publication year) were not used. The decision to exclude gray literature has been motivated and complies with the study's intent. Furthermore, no sampling of relevant studies was needed as the number of results was low enough to select all results for further investigation. Decisions to select a study as an extended version of another were conducted with a consistent strategy.

Additionally, we performed forward and backward snowballing, identifying one further study. This study [25] was published at a small workshop in 2007 and is not indexed in any major database. Thus, we conclude that this addition does not indicate a general fault in our search process.

Only one author performing the study poses a threat to validity. This *researchers' bias* was mitigated by discussion between the authors before each step was conducted and revision of each step's results by the second author.

B. Data Validity

Data validity for systematic mapping studies is concerned with constructing the classification scheme and the data extraction. To mitigate this threat, we applied several strategies: We used a combination of deductive (the Who-What-How framework) and inductive scheme construction to preserve the advantages and avoid the disadvantages of both strategies. Furthermore, by using an adaptive reading strategy, no decisions have been generically limited to the contents of an abstract.

C. Research Validity

The research process for the systematic mapping study has been described in detail. The data has been made available to ensure reliability and enable replication or extension.

VII. Conclusions

A. Summary

This meta-study presents a systematic mapping study on API deprecation, including 36 relevant primary studies. We found studies to mainly address human stakeholders, to provide descriptive analyses as well as new solutions, and to base their insights mostly on (open-source) data. Furthermore, the studies strongly focus static APIs, particularly Java and the Android ecosystem, and clients' reactions to deprecation.

Subsequently, we identified five major gaps in current research on API deprecation:

1) **Uncharted Territory:** Deprecation of remote APIs has barely been considered.
2) **Out of Focus:** Research on the deprecation of static APIs includes only a few programming languages.
3) **Unbridged Gap:** Suppliers and clients of an API have rarely been considered jointly.
4) **Human-out-of-the-loop:** Research strategies have been focused on data and did not include human actors.
5) **Prevention Better Than Cure:** Investigations do not include causes or prevention of deprecation.

B. Future Work

First, future research on API deprecation needs to address the five major research gaps we identified.

Furthermore, future research needs to compare academic achievements with practitioners' needs on API deprecation to enhance the applicability of findings in practice. In this context, existing gray literature and knowledge among practitioners need to be included in future studies.

Finally, a general model of API deprecation—extending or replacing the basic four-step model presented in this study—should be developed and evaluated. In particular, both suppliers and clients should be included with their needs and contributions to the process.

C. Data Availability

The complete data set has been made available as open data via Zenodo [2]. The files include all unique search results:

- Included studies with their complete classification.
- Excluded studies with the decisive exclusion criteria.
- A list of studies identified through snowballing.

References

[1] M. Raatikainen, E. Kettunen, A. Salonen, M. Komssi, T. Mikkonen, and T. Lehtonen, "State of the Practice in Application Programming Interfaces (APIs): A Case Study," in *ECSA'21*, 2021, pp. 191–206.

[2] L. Bonorden and M. Riebisch, "API Deprecation: A Systematic Mapping Study [Data set]," Zenodo, 2022. [Online]. Available: https://www.doi.org/10.5281/zenodo.5650121

[3] I. W. Cotton and F. S. Greatorex, "Data structures and techniques for remote computer graphics," in *1968 Fall Joint Comp. Conf.*, ser. AFIPS Conf. Proc., vol. 33, part I, 1968, pp. 533–544.

[4] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Commun. ACM*, vol. 15, no. 12, pp. 1053–1058, 1972.

[5] I. Rauf, E. Troubitsyna, and I. Porres, "A systematic mapping study of API usability evaluation methods," *Comput. Sci. Rev.*, vol. 33, pp. 49–68, 2019.

[6] J. Gosling *et al.*, "The Java Language Specification, Java SE 17 Edition," 2021.

[7] M. Lamothe, Y.-G. Guéhéneuc, and W. Shang, "A Systematic Review of API Evolution Literature," *ACM Comput. Surv.*, vol. 54, no. 8, pp. 1–36, 2022.

[8] C. Burns, J. Ferreira, T. D. Hellmann, and F. Maurer, "Usable results from the field of API usability: A systematic mapping and further analysis," in *VL/HCC'12*, 2012, pp. 179–182.

[9] A. Cummaudo, R. Vasa, and J. Grundy, "What should I document? a preliminary systematic mapping study into API documentation knowledge," in *ESEM'19*, 2019, pp. 1–6.

[10] K. Nybom, A. Ashraf, and I. Porres, "A Systematic Mapping Study on API Documentation Generation Approaches," in *SEAA'18*, 2018.

[11] R. Bonifácio, S. Krüger, K. Narasimhan, E. Bodden, and M. Mezini, "Dealing with Variability in API Misuse Specification," in *ECOOP'21*, 2021, pp. 19:1–19:27.

[12] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, "Systematic Mapping Studies in Software Engineering," in *EASE'18*, 2008.

[13] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Inf. Softw. Technol.*, vol. 64, pp. 1–18, 2015.

[14] P. Ralph *et al.*, "Empirical Standards for Software Engineering Research," 2021.

[15] B. Kitchenham and S. Charters, "Guidelines for performing Systematic Literature Reviews in Software Engineering," Evidence-Based Software Engineering, Tech. Rep. EBSE-2007-01, 2007.

[16] M.-A. Storey, N. A. Ernst, C. Williams, and E. Kalliamvakou, "The who, what, how of software engineering research: A socio-technical framework," *Empir. Softw. Eng.*, vol. 25, no. 5, pp. 4097–4129, 2020.

[17] J. Thomas *et al.*, "Eppi-reviewer: advanced software for systematic reviews, maps and evidence synthesis," 2022, EPPI-Centre, UCL Social Research Institute, University College London.

[18] J. Yasmin, Y. Tian, and J. Yang, "A First Look at the Deprecation of RESTful APIs: An Empirical Study," in *ICSME'20*, 2020, pp. 151–161.

[19] G. Brito, A. Hora, M. T. Valente, and R. Robbes, "On the use of replacement messages in API deprecation: An empirical study," *J. Syst. Softw.*, vol. 137, pp. 306–321, 2018.

[20] S. A. Haryono, F. Thung, D. Lo, L. Jiang, J. Lawall, H. Jin Kang, L. Serrano, and G. Muller, "AndroEvolve: Automated Update for Android Deprecated-API Usages," in *ICSE'21: Demos*, 2021, pp. 1–4.

[21] L. Li, J. Gao, T. F. Bissyandé, L. Ma, X. Xia, and J. Klein, "CDA: Characterising Deprecated Android APIs," *Empir. Softw. Eng.*, vol. 25, no. 3, pp. 2058–2098, 2020.

[22] A. A. Sawant, M. Aniche, A. van Deursen, and A. Bacchelli, "Understanding developers' needs on deprecation as a language feature," in *ICSE'18*, 2018, pp. 561–571.

[23] C. Wohlin, "Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering," in *EASE'14*, 2014.

[24] W. Ammar *et al.*, "Construction of the literature graph in semantic scholar," in *NAACL-HLT: Industry*, 2018, pp. 84–91.

[25] S. A. Spoon, "Fine-grained api evolution for method deprecation and anti-deprecation," in *Int. Workshop on Foundations and Developments of Object-Oriented Languages*, 2007.

[26] A. Hora, R. Robbes, M. T. Valente, N. Anquetil, A. Etien, and S. Ducasse, "How do developers react to API evolution? a large-scale empirical study," *Softw. Qual. J.*, vol. 26, no. 1, pp. 161–191, Mar. 2018.

[27] H. Huang, L. Wei, Y. Liu, and S.-C. Cheung, "Understanding and detecting callback compatibility issues for Android applications," in *ASE'18*, 2018, pp. 532–542.

[28] D. Ko, K. Ma, S. Park, S. Kim, D. Kim, and Y. L. Traon, "API Document Quality for Resolving Deprecated APIs," in *APSEC'14*, 2014, pp. 27–30.

[29] M. Lamothe and W. Shang, "Exploring the use of automated API migrating techniques in practice: an experience report on Android," in *MSR'18*, 2018, pp. 503–514.

[30] A. Mirian, N. Bhagat, C. Sadowski, A. Porter Felt, S. Savage, and G. M. Voelker, "Web Feature Deprecation: A Case Study for Chrome," in *ICSE'19: SEIP*, 2019, pp. 302–311.

[31] L. Moreno, G. Bavota, M. D. Penta, R. Oliveto, A. Marcus, and G. Canfora, "ARENA: An Approach for the Automated Generation of Release Notes," *IEEE Trans. Softw. Eng.*, vol. 43, no. 2, 2017.

[32] R. S. d. Nascimento, E. Figueiredo, and A. Hora, "JavaScript API Deprecation Landscape: A Survey and Mining Study," *IEEE Softw.*, 2021, Early Access.

[33] H. A. Nguyen, T. T. Nguyen, G. Wilson, A. T. Nguyen, M. Kim, and T. N. Nguyen, "A graph-based approach to API usage adaptation," in *OOPSLA'10*, 2010, pp. 302–321.

[34] M. A. Nishi and K. Damevski, "Automatically identifying valid API versions for software development tutorials on the Web," *J. Softw. Evol. Proc.*, vol. 32, no. 4, p. e2227, 2020.

[35] J. H. Perkins, "Automatically generating refactorings to support API evolution," in *6th Workshop on Program analysis for software tools and engineering*. Lisbon, Portugal: ACM, 2005, pp. 111–114.

[36] D. Qiu, B. Li, and H. Leung, "Understanding the API usage in Java," *Inf. Softw. Technol.*, vol. 73, pp. 81–100, 2016.

[37] S. Raemaekers, A. van Deursen, and J. Visser, "Semantic versioning and impact of breaking changes in the Maven repository," *J. Syst. Softw.*, vol. 129, pp. 140–158, 2017.

[38] R. Robbes, M. Lungu, and D. Röthlisberger, "How do developers react to API deprecation?: the case of a smalltalk ecosystem," in *FSE'12*, 2012.

[39] M. Samak, D. Kim, and M. C. Rinard, "Synthesizing replacement classes," *Proc. ACM Program. Lang.*, vol. 4, no. POPL, 2020.

[40] A. A. Sawant, R. Robbes, and A. Bacchelli, "On the reaction to deprecation of clients of 4 + 1 popular Java APIs and the JDK," *Empir. Softw. Eng.*, vol. 23, no. 4, pp. 2158–2197, 2018.

[41] A. A. Sawant, G. Huang, G. Vilen, S. Stojkovski, and A. Bacchelli, "Why are Features Deprecated? an Investigation Into the Motivation Behind Deprecation," in *ICSME'18*, 2018, pp. 13–24.

[42] A. A. Sawant, R. Robbes, and A. Bacchelli, "To react, or not to react: Patterns of reaction to API deprecation," *Empir. Softw. Eng.*, vol. 24, no. 6, pp. 3824–3870, 2019.

[43] S. Scalabrino, G. Bavota, M. Linares-Vasquez, M. Lanza, and R. Oliveto, "Data-Driven Solutions to Detect API Compatibility Issues in Android: An Empirical Study," in *MSR'19*, 2019, pp. 288–298.

[44] R. Štrobl and Z. Troníček, "Migration from deprecated API in Java," in *SPLASH'13: Posters*, 2013, pp. 85–86.

[45] F. Thung, S. A. Haryono, L. Serrano, G. Muller, J. Lawall, D. Lo, and L. Jiang, "Automated Deprecated-API Usage Update for Android Apps: How Far are We?" in *SANER'10*, 2020, pp. 602–611.

[46] F. Thung, H. J. Kang, L. Jiang, and D. Lo, "Towards Generating Transformation Rules without Examples for Android API Replacement," in *ICSME'19*, 2019, pp. 213–217.

[47] A. Vadlamani, R. Kalicheti, and S. Chimalakonda, "APIScanner - Towards Automated Detection of Deprecated APIs in Python Libraries," in *ICSE: Demos*, 2021, pp. 5–8.

[48] J. Wang, L. Li, and A. Zeller, "Better code, better sharing: on the need of analyzing jupyter notebooks," in *ICSE'20: NIER*, 2020, pp. 53–56.

[49] J. Wang, L. Li, K. Liu, and H. Cai, "Exploring how deprecated Python library APIs are (not) handled," in *ESEC/FSE'20*, 2020, pp. 233–244.

[50] Y. Xi, L. Shen, Y. Gui, and W. Zhao, "Migrating Deprecated API to Documented Replacement: Patterns and Tool," in *11th Asia-Pacific Symposium on Internetware*, 2019.

[51] B. Xu, L. An, F. Thung, F. Khomh, and D. Lo, "Why reinventing the wheels? an empirical study on library reuse and re-implementation," *Empir. Softw. Eng.*, vol. 25, no. 1, pp. 755–789, 2020.

[52] J. Zhou and R. J. Walker, "Api deprecation: a retrospective analysis and detection method for code examples on the web," in *FSE'16*, 2016.

[53] S. Arvedahl, "Introducing Debtgrep: A Tool for Fighting Technical Debt in Base Station Software," in *2018 Int. Conf. on Technical Debt*, 2018.

[54] F. R. Cogo, G. A. Oliva, and A. E. Hassan, "Deprecation of packages and releases in software ecosystems: A case study on npm," *IEEE Trans. Softw. Eng.*, 2021, Early Access.

[55] S. A. Haryono, F. Thung, H. J. Kang, L. Serrano, G. Muller, J. Lawall, D. Lo, and L. Jiang, "Automatic Android Deprecated-API Usage Update by Learning from Single Updated Example," in *ICPC'20*, 2020.

[56] A. Ampatzoglou, S. Bibi, P. Avgeriou, M. Verbeek, and A. Chatzigeorgiou, "Identifying, categorizing and mitigating threats to validity in software engineering secondary studies," *Inf. Softw. Technol.*, vol. 106, pp. 201–230, 2019.