A Systematic Review of API Evolution Literature

MAXIME LAMOTHE, Department of Computer Engineering and Software Engineering, Polytechnique Montréal, Canada
YANN-GAËL GUÉHÉNEUC and WEIYI SHANG, Department of Computer Science and Software Engineering, Concordia University, Canada

Recent software advances have led to an expansion of the development and usage of application programming interfaces (APIs). From millions of Android packages (APKs) available on Google Store to millions of open-source packages available in Maven, PyPI, and npm, APIs have become an integral part of software development.

Like any software artifact, software APIs evolve and suffer from this evolution. Prior research has uncovered many challenges to the development, usage, and evolution of APIs. While some challenges have been studied and solved, many remain. These challenges are scattered in the literature, which hides advances and cloaks the remaining challenges.

In this systematic literature review on APIs and API evolution, we uncover and describe publication trends and trending topics. We compile common research goals, evaluation methods, metrics, and subjects. We summarize the current state-of-the-art and outline known existing challenges as well as new challenges uncovered during this review.

We conclude that the main remaining challenges related to APIs and API evolution are (1) automatically identifying and leveraging factors that drive API changes, (2) creating and using uniform benchmarks for research evaluation, and (3) understanding the impact of API evolution on API developers and users with respect to various programming languages.

CCS Concepts: • **Software and its engineering → Designing software**; **Software design tradeoffs**; **Software evolution**;

Additional Key Words and Phrases: SLR, APIs, API Evolution

ACM Reference Format:
Maxime Lamothe, Yann-Gaël Guéhéneuc, and Weiyi Shang. 2020. A Systematic Review of API Evolution Literature. 1, 1 (June 2020), 35 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Software application programming interfaces (APIs) allow their users to save time and effort by relying on pre-made functionality [121]. It is therefore not surprising that APIs are extensively used by software developers and that their usage is highly recommended to promote software quality while reducing development effort. For example, the Android API allows APKs, over 8 million in the Google Play store alone [56], to run on mobile devices across the world.

Authors' addresses: Maxime Lamothe, maxime.lamothe@polymtl.ca, Department of Computer Engineering and Software Engineering, Polytechnique Montréal, 2500 Chemin de Polytechnique, Montréal, Qc, Canada; Yann-Gaël Guéhéneuc, yann-gael.gueheneuc@concordia.ca; Weiyi Shang, shang@encs.concordia.ca, Department of Computer Science and Software Engineering, Concordia University, 1455 Boulevard de Maisonneuve O, Montréal, Qc, Canada.

XXXX-XXXX/2020/6-ART $15.00
https://doi.org/10.1145/1122445.1122456

APIs are by definition interfaces to be used as entry points to reusable software entities [145]. They are not independent software entities; they are instead packaged with the software libraries [40], frameworks [70], or Web services [161], that offer them.

The ease with which APIs can be discovered and used increased with the advent of Software-as-a-service [85] and the growth of open-source software repositories, e.g., GitHub. For example, JUnit, a popular unit-testing framework, has been used by over 20,000 applications in a 42,000 application sample [155] and is often adopted by users when migrating away from other testing frameworks, e.g., TestNG [37].

APIs are inherently software artifacts and are, thus, not immune to Lehman's laws [89]. To remain useful and competitive, APIs must evolve. They evolve to offer new functionalities, fix security issues, retire unsafe/no longer necessary functionalities, and, more generally, to increase the ease with which developers can use them. For example, JUnit was introduced in 2002 and its latest version released in September 2020. It grew from version 1.0 to version 5.7.0, from offering "plain old Java objects" (POJO, classes really) and using reflection to an annotation-based framework with filters, recorders, loggers, conditional testing, etc. However, API evolution can cause various issues for both their users and their developers [85, 103, 121, 147].

Due to their omnipresence and evolution, APIs greatly impact software development. Understanding, mitigating, and leveraging the impact of APIs and API evolution on software development is necessary to design and use software APIs [148].

In the last few decades, interest in APIs and API evolution has grown rapidly in the software-engineering research community. As it grew, so did the number of publications related to APIs and their evolution. We only identified one work published in 1994 related to API evolution but 36 works in 2020. In all the works that we studied, researchers explored a variety of aspects of APIs, from API usability and misuses, to API maintenance, migration, documentation, recommendation and more. Prior research has produced many empirical studies [15, 70, 74, 104], new tools and techniques [109, 145, 148], and datasets [3, 11, 155] to uncover and solve issues due to API evolution.

Due to both the breadth and depth of the research related to APIs and API evolution, it is difficult to determine the extent of prior research, for example, which problems were uncovered, and which solutions were proposed. The large number of existing publications hide advances and also cloak important, remaining challenges in APIs and API evolution. We need a comprehensive view of the state of the art on APIs and API evolution to help researchers and practitioners.

Therefore, a survey of prior work between 1994 and 2020 (i.e., 27 years) related to API evolution and its effects would benefit the software research community as well as software developers. It should highlight existing research into API evolution and issues affected by API evolution, present the current state-of-the-art solutions to uncovered challenges, and enumerate challenges that have yet to be solved.

In this survey paper, we compile the challenges of API evolution scattered in the literature through a systematic literature review. We uncover and describe publications trends as well as trending topics. We also compile common research goals, evaluation methods, metrics, and subjects. We summarise the current state-of-the-art and provide an overview of known existing challenges and new challenges uncovered during this review.

We conclude that the main remaining challenges related to APIs and API evolution are (1) to automatically identify and leverage factors that drive API changes, (2) create and use uniform benchmarks for research evaluation, and (3) understand the impact of API evolution on API developers and users with respect to various programming languages.

Section 2 defines APIs and API evolution. Section 3 describes the methodology used to find the works selected for this literature review. Section 4 highlights the various goals, tools, and evaluations used in API evolution research. Section 5 summarises the state-of-the-art in API evolution research.

Section 6 presents open API evolution challenges that remain either partially or completely unsolved by current research. Section 7 describes the threats to the validity of this paper. Finally, Section 8 concludes the paper.

2 PRELIMINARIES

This section briefly presents the concept of APIs and presents an introduction to API evolution.

2.1 Definition of an API

To the best of our knowledge, the term application programming interface appeared for the first time in 1968 within the context of providing a remotely accessed, interactive computer graphics system [32]. Application programming interfaces (API) are varied and can encompass different concepts. For example, when the concept of information hiding was first coined by Parnas [127] in 1972, it was based on interfaces among modules, which today would be called APIs.

Prior work has defined APIs as "the interface to a reusable software entity used by multiple clients outside the developing organization, and that can be distributed separately from environment code" [145]. Although the term 'API' can be used as a general term for an interface between software components, there exists nomenclature to refer to certain types of APIs. For example, software libraries [13, 20, 39, 40, 54, 57, 65, 77, 110, 115, 190, 195], software frameworks [33, 40, 42, 43, 70, 107, 117, 159, 183, 184, 188, 194], and Web services either RESTful [100, 137, 160, 161] or SOAP [160] have all been interchangeably been referred to as APIs because they all allow pieces of software to communicate, albeit in different ways. However, API terminology can sometimes be nuanced. For example, object-oriented languages, such as Java and C#, have specific keyword concepts to define interfaces [111, 126]. According to the definition of an API presented by prior research [145], these interfaces may only be considered APIs if they are used by multiple external clients. In this paper, we use this API definition but also consider interfaces that may be used by multiple clients within a developing organization as APIs.

2.2 API Evolution

Prior studies have shown that APIs evolve for various reasons such as increasing complexity [103], and continuous change [41, 89]. However, due to their nature as a connection point between software modules, API evolution is not without side-effects. Many studies have shown the effects of API changes not only on the API itself [41], but also on its clients [104]. APIs may therefore change differently from traditional software artifacts. For example, Sun Microsystem preferred introducing the new interface *java.awt.LayoutManager2* rather than change the *java.awt.LayoutManager* because changing the latter would have broken existing code [162].

The evolution of APIs induces a variety of problems and challenges for API users and API developers alike [86]. On the one hand, as predicted by Lehman, *continuing change* [89] means that API developers must determine ways to keep their APIs useful, cutting edge, and competitive with other pieces of software [85] and API users must adapt to these API changes and new API releases. On the other hand, *conservation of familiarity* [89], or existing API usages, constrain the evolution of an API to avoid breaking changes while improving the API (i.e., security or performance improvements). The evolution of APIs therefore involves a balancing act of constant improvement and maintaining existing functionality. Maintaining existing functionality requires in-depth knowledge of use cases and architectural foresight and flexibility, while keeping up with rapid release cycles requires modifications to user applications as well as learning about new APIs and changes to existing APIs. Therefore, when gathering literature for our systematic review, we not only concentrate on work that directly studies APIs and their evolution, but we also consider prior work that focuses on finding solutions to problems that are caused by API evolution.

3 METHODOLOGY

In this paper, we used a well-defined, structured, and systematic approach to produce a survey on API evolution. The approach followed was inspired by guidelines from Kitchenham et al. [81] and Petersen et al. [130].

3.1 Research Questions

The goal of this systematic literature review is to provide a structured and categorized aggregate of existing API evolution research to uncover the state of API research. This knowledge will hopefully allow insight into the current state-of-the-art research and provide a quick reference into existing practices and currently unsolved challenges for future research. To achieve this goal, we designed the following research questions (RQs):

– *RQ1: How has the field of API evolution research evolved?*
  We seek to explore published papers related to API evolution, we provide an overview of these paper, categorize them, identify their goals, and investigate strategies used by API evolution researchers to evaluate their findings and discuss evaluation trends. We present our findings for this RQ in Section 4.
– *RQ2: What is the current state-of-the-art in API evolution research?*
  We present state-of-the-art approaches and tools that have been proposed to deal with problems related to API evolution. We present our findings in Section 5.
– *RQ3: What are the current and future challenges related to API evolution?*
  Finally, we seek to uncover current and future challenges still left to solve for future API research. We present our findings in Section 6.

3.2 Literature Repository Selection

We used prior state-of-the-art software engineering literature reviews [72, 75] to obtain our selection criteria for online literature repositories. Our original selection of papers came from the following technical publishers:

- ACM Digital Library
- Elsevier Science Direct
- IEEE Xplore Digital Library
- Springer Online Library
- Wiley Online Library

We also augmented our paper selection by performing a search in the Google Scholar database by entering “API Evolution” as an exact search string and parsing the results. This was done to supplement the selection of papers from technical publishers and to ensure the widest possible search scope for our survey. Indeed, the exact search phrase “API Evolution” was as our search phrase for all technical publishers.[1] Furthermore, we manually mined the references of each of the papers in our original selection, using forward and backward snowballing (i.e., using Google Scholar to search for citations of, and in, a specific publication), to find cited works that appeared to present work within the scope of API evolution based on their abstracts.

3.3 Literature Search and Selection

Using our predefined literature repositories, we performed searches using the exact “API Evolution” search phrase.[2] The results obtained are presented in Table 1. The results highlight the absolute number of publications found in each library, as well as the number of publications that were

[1]Repository of our primary studies and classifications: https://github.com/senseconcordia/APIEvolutionSurveyPapers
[2]The latest search was conducted on Feb 12, 2020.

Table 1. Publications found by search engine

Search Engine	Unfiltered Publications	Cross-Referenced
ACM Digital Library	122	113
Elsevier Science Direct	22	10
IEEE Xplore Digital Library	38	33
Springer Online Library	81	74
Wiley Online Library	5	4
Google Scholar	1061	215
Total *(duplicates removed)*	1040	212

cross-referenced and available in multiple libraries. After accounting for all duplicate publications, we found a total of 828 publications. The first author then manually filtered the results of this search, keeping only results which met the following criteria:

(1) Studies must be written in English
(2) Studies must be related to computer science or software engineering
(3) Studies should have a relation to API evolution
(4) Studies must not be a Master or PhD thesis
(5) Studies must be fully available from one or more online library

A flowchart of our publication selection process can be found in Figure 1. Based on our filtering process, we obtained 179 publications. After checking the references of the chosen papers, we added a further 190 papers to the survey. These papers were likely missing in the initial library search due to nomenclature differences (e.g., *Framework* evolution instead of *API* evolution). Finally, using the results of our initial search, along with any references that matched our filtering criteria, we selected a total of 369 publications (or *primary studies*) with which to conduct this survey. The filtering was done by one author, with a test-retest reliability coefficient of 0.94, showing excellent reliability [81, 173]. The most common reasons for filtering out publications were: not related to

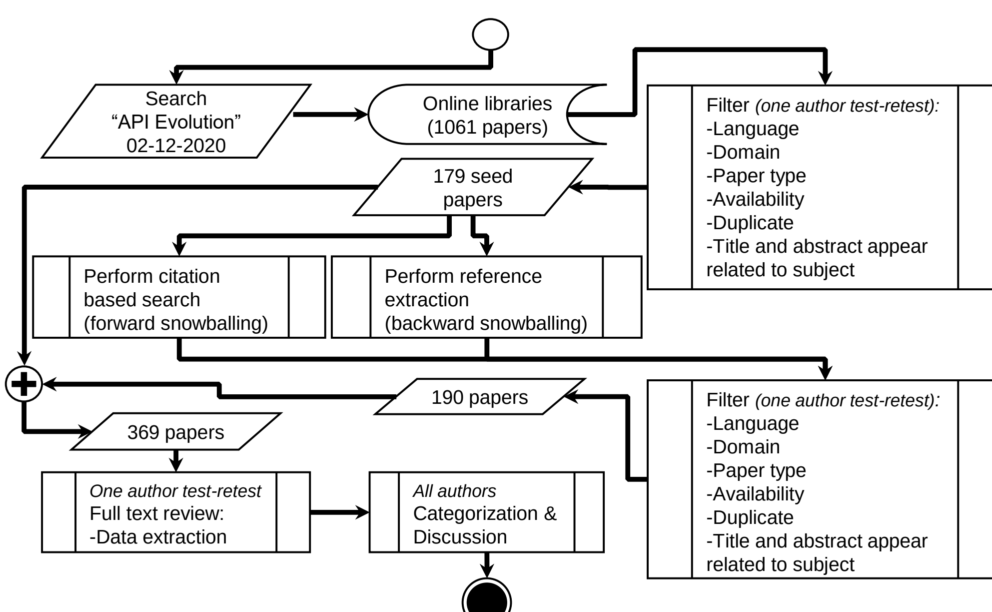


Fig. 1. Our paper selection process

Table 2. Data extracted for our research questions

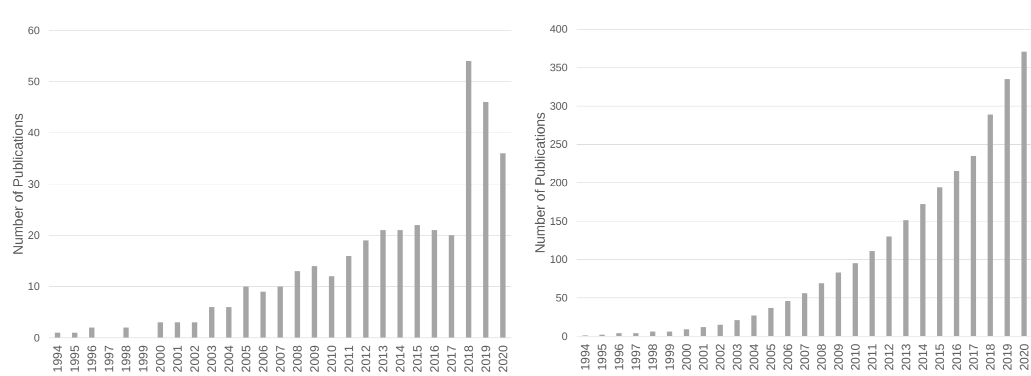
RQ	Type of Data Extracted
RQ1	Title, author information (names and affiliations), publication information (type, year, and location), names and sources of systems under test, types of evaluations performed, evaluation metrics, study motivation, methodology, and paper type
RQ2	Paper type, primary contribution, challenges uncovered and solved
RQ3	Unresolved questions, future research avenues

API Evolution (46%), not available from at least one online library (25%), thesis (17%), language (i.e., non-English) (9%), not related to software engineering (3%).

3.4 Data Extraction and Collection

To answer our research questions, one author carefully examined and extracted information from each of the 369 publications selected for this study. We paid particular attention to the motivation, contributions, methodologies and tooling, results, and challenges presented in the publications. To present concise and practical information, we categorized our findings into abstract categories whenever possible. The types of data extracted from each publication by one author and their relevance to each research question are presented in Table 2. The extracted categories were tested using test-retest reliability on a statistically significant sample of 189 publications (confidence level 95%, margin of error 5%). A test-retest reliability coefficient of 0.97 was obtained, showing excellent reliability [173]. All papers that met our filtering criteria and were officially published as of December 31st, 2020, were included in this study.

To answer RQ1 we categorize the topics of our selected publications, determine publication trends in API evolution, and uncover publication patterns in API evolution research. We look at which researchers and organizations publish most in the field, how often papers are published, in which type of venue they are published, and with which type of work they are most related.[3] We also categorize API evolution papers into five contribution types: **New Tools and Techniques**, **Empirical Studies**, **Tools and Techniques Proposals**, **Surveys**, and **Datasets**.



(a) Publications per year (b) Cumulative publications per year

Fig. 2. API Evolution publications from Sept 19th, 1994, to Dec 31st, 2020

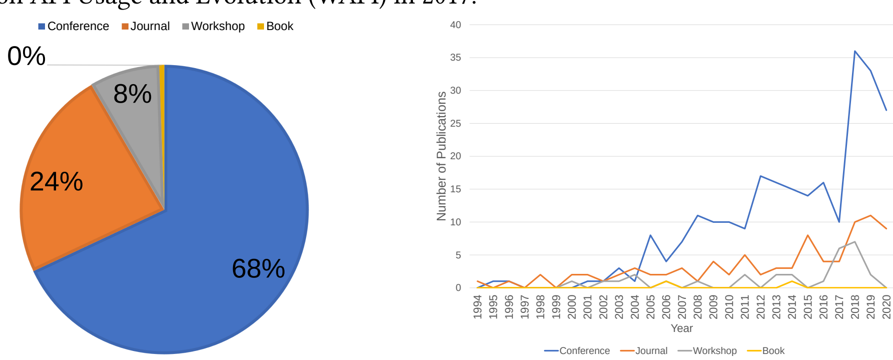
[3]More information is presented as part of an Appendix on: https://github.com/senseconcordia/APIEvolutionSurveyPapers

3.4.1 Publication Trends. Publications in API Evolution are trending upwards. As shown in Figure 2, the number of publications with topics related to API Evolution more than doubled from 2017 to 2018 and has continued to stay high. Furthermore, we can also observe an exponential increase in the number of cumulative publications per year. This tells us that API Evolution is not only an active research topic but is also a growing research field.

3.5 Overview of Primary Studies

3.5.1 Most Common Publication Venues. The publications studied in this paper are spread over a variety of venues, some are more popular than others. Amongst the reviewed publications, the most common venue for journal paper publications is IEEE's Transactions on Software Engineering (TSE) with eight journal publications, followed by Empirical Software Engineering with six API Evolution journal publications. The most common conference is the International Conference on Software Engineering (ICSE) with 42 publications. The most common workshop is the Workshop on API Usage and Evolution (WAPI) with 12 publications.

As shown in Figure 3 we can see that the majority of publications in API Evolution are conference papers, followed by Journal papers and workshops, with only a slim minority (two) books being published. We can also see that workshop papers appear to be increasing in numbers starting in 2017. This increase in workshop publications is likely due to the founding of the International Workshop on API Usage and Evolution (WAPI) in 2017.



(a) Overall venue distribution

(b) Yearly venue distribution

Fig. 3. API Evolution publication venues

3.5.2 Publication Topics. The three authors classified the 369 publications into various topics through the use of keywords provided by the authors within the papers themselves, keywords provided by the publisher (e.g., IEEE Keywords), or through the use of our judgment in cases where we could not recover relevant keywords.

We first employed closed card sorting to sort papers into three blanket categories, *API Maintenance* which contains 178/369 publications, *API Usability* which contains 161/369 publications, and *Other* which contains 19 publications. We then used a second round of closed card sorting to further subdivide each blanket category as shown in Figure 4. We identified three primary API research topics: *API Maintenance*, *API Usability*, and *Other.*

Since the *Other* category only contains 19 publications of various topics, it was not subdivided into subcategories. The evolution trend of the three categories and their subcategories can be observed in Figure 5. Figure 5a shows that both API usability and maintenance papers grew through the years. However, since 2011, the API research community has started to favor usability papers, with the exception of 2020 where maintenance papers dominated the field (24 vs. 7).

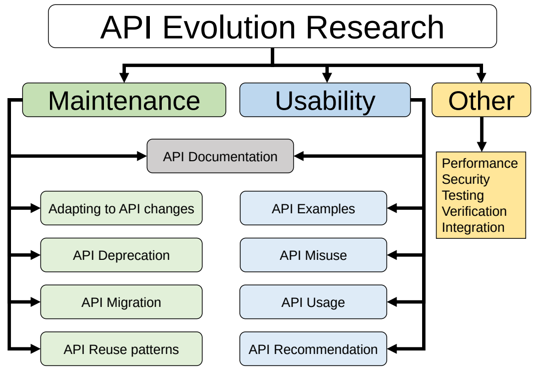


Fig. 4. API research topics

Looking at the subtopics for API maintenance and usability in Figure 5 we can see that the *API Usage* research subtopic appears to be growing rapidly in recent years. This growth can likely be attributed to tools and empirical research to uncover what makes API hard to use [202] and uncovering usage patterns to help developers [179]. The growth in popularity for these topics might be linked to the growth in available API usage data on open-source repositories and forums such as GitHub and Stack Overflow which were both launched in 2008. API migration research appears to be one of the more steadily growing research subtopics with a minimum of two publications per year since 2003, and 14 in 2020. Meanwhile, the API misuses and recommendation subtopics appear to be gaining popularity in recent years. Although the first API misuse paper in our sample was published in 2001 [50], recent years have shown a steady stream of papers related to the topic, with three papers published in 2018 [5, 9, 142], and two in 2019 [4, 177]. The topic of API recommendation started gaining recognition in 2009 [135] and has been steadily gaining ground ever since.

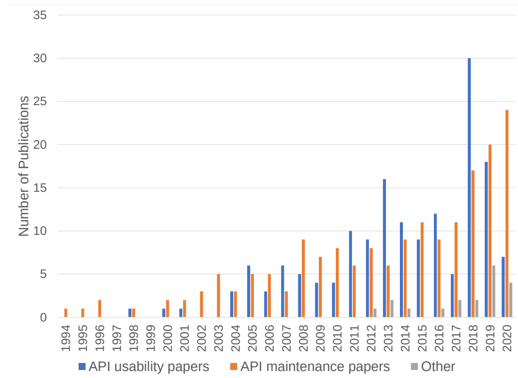
3.5.3 Publication Contribution Types. We also classified our sample of 369 publications into five publication contribution types using an open card sorting approach with all three authors. For this classification, we rely on the judgement of the authors of this paper to extract the primary contribution of each paper. It is possible for a paper to present more than one contribution, and we sometimes must rely on human judgement to identify the primary or main contribution. Similarly to the research topic classification in Section 3.5.2, we created the contribution type categories by using author and publisher keywords, while also relying on publication venue information when it was relevant. These sources of information were combined with our best judgement to classify each publication after reading it. We generated the following five contribution types: (1) *New Tools and techniques*: comprised of publications that showcase novel tools and techniques to aid with existing or unsolved API evolution challenges, (2) *Empirical studies*: publications that primarily present data analysis and findings based on empirical evidence, (3) *Tools and Technique Proposals*: publications that propose novel tools or techniques without implementation details or experimental

Table 3. API publication by type

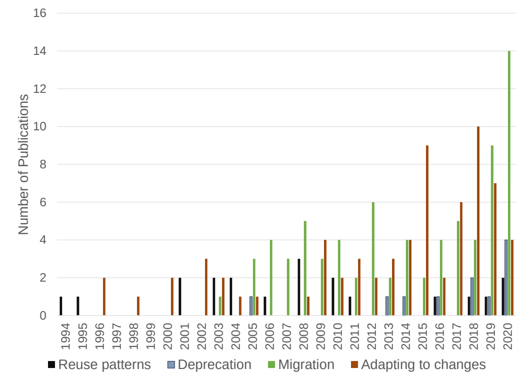
Publication Type	Papers
Conference	251
Journal	87
Workshop	29
Book	2

Table 4. API publication contribution types

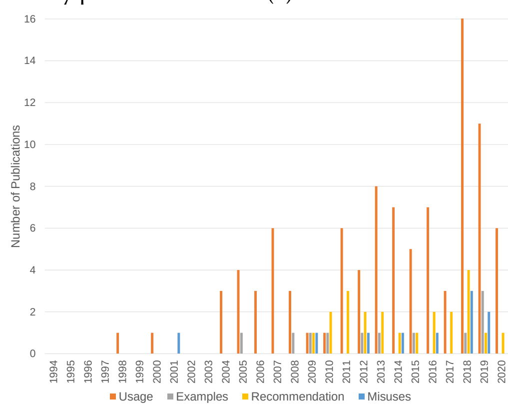
Main contribution	Papers
New Tools and Techniques	210
Empirical Studies	138
Tools and Technique Proposals	13
Surveys	5
Datasets	3



(a) API Maintenance and Usability publication trends



(b) API Maintenance subtopic publications per year



(c) API Usability subtopic publications per year

Fig. 5. API Maintenance and Usability subtopic publication trends

results, (4) *Surveys*: publications based on the systematic analysis of multiple existing works, and (5) *Datasets*: publications that showcase and share novel data to be used for future research. The overall classification of the publications we studied can be found in Table 4.

4 EVOLUTION OF API EVOLUTION RESEARCH

We now answer *RQ1: How has the field of API evolution research evolved?* We divide our answer in three parts on (1) API evolution research goals, (2) API evolution research evaluation, and (3) API evolution experimental subjects, which are the three main components of any research work on API evolution.

4.1 API Evolution Research Goals

We answer the first part of our RQ1 by presenting the various goals that we uncovered when surveying API evolution literature. API evolution presents various avenues for research. For example, it is possible to empirically observe the impact of API changes on API users [85], otherwise known as the effect of *perceived complexity* on users [89], these studies can then provide motivation and insight to develop software tooling [34]. To better understand the trends in API evolution research, we use our publication contribution classification of the 369 papers. We divide this section using the five contribution types identified in Section 3.5.3, namely, *Datasets*, *Empirical Studies*, *Tools and*

Technique Proposals, *Surveys*, and *New Tools and Techniques* to uncover which primary contributions align with which research goals.

4.1.1 New Tools and Techniques. As shown in Table 4, the majority of the papers present new tools and techniques to help with API evolution. These tools and techniques vary in scope and purpose. However, they all seek to resolve problems caused by API evolution through the intervention of either a tool or a new technique. For example, to deal with the challenge of breaking API changes, some papers attempt to help organize changes [89, 158], while others try to automate API migration [34, 64, 87, 184]. To help improve API usability, some attempt to reduce complexity [148], others attempt to help conserve familiarity [149]. To reduce API misuses, some papers propose misuse detectors [4]. We use existing surveys on API property inference techniques [145], recommendation systems [148], and software merging [109] as well as some of our own categorizations to label our dataset into API research topics presented in Section 3.5.2 of this survey (i.e, adapting to API changes, documentation, deprecation, examples, misuse, migration, recommendation, reuse patterns, usage, and other).

Answers of RQ1: New tools and techniques typically seek to help with API evolution by resolving problems that it can cause for API users (e.g., API migration tools) or to help reduce the development burden on API developer (e.g., automatic API documentation tools).

4.1.2 Empirical Studies. The second largest category of API evolution publications are presented as empirical studies. The empirical studies we observed within our dataset can largely be divided into three sub-categories. Data-mining empirical studies, that make use of data from several projects or non-human sources, empirical case studies, which target specific projects and often provide in-depth results for a few specific non-human sources, and finally user studies which make use of human participants.

Data-mining Studies: Data-mining studies concentrate on using large sources of data to provide evidence for the existence of problems and to determine their impact. These studies mainly explore challenges that deal with breaking API changes, for example, through studying the rapid evolution of mobile apps and the android API [28, 104], app categories [61] and ratings [15], and compatibility problems [63].

Case Studies: Case studies study a few (e.g., fewer than 10) systems. Comparatively to data-mining studies, the results of case studies are specific to the studied systems. These studies present a range of goals and deal with various challenges. From understanding braking API changes, by determining the impact of API evolution on API users [70], to improving API usability by determining whether IDEs influence the usability of dynamic and static APIs [131], or determining the factors that support the long term success of frameworks [117], and many more [9, 178, 179].

User Studies: We classified eight of the papers reviewed for this survey as user studies. These papers rely on human responses to answer their research questions, which have a strong usability component. Therefore, we surmise that user studies are particularly well suited for API usability studies, and particularly concentrate on the challenges of improving API usability. The papers determine learning barriers in end-user systems [74], analyze the API usage of an IDE [27], understand developers' deprecation needs [154] understand how API documentation fails [169], evaluate the usability of the factory pattern in APIs [48], determine what makes APIs hard to learn [144], explore the pitfalls of unfamiliar APIs [45], and study API usability [134].

Answers of RQ1: Empirical studies related to API evolution typically employ large data, case studies, or user studies to provide evidence of existing problems, the impacts of API evolution, or potential solutions to existing problems. These problems typically centre around the impacts of API evolution on API usability and API maintainability.

4.1.3 Tools and Technique Proposals. Tools and Technique Proposals within our dataset seek to highlight existing concerns in the field and provide potential approaches to resolving the problems. These papers are categorized separately because they present a particular paper structure. These papers concentrate on the same issues as New Tools and Techniques, however they only propose their solutions rather than actualize them. These proposals highlight issues that have been found in prior work (e.g., most API breaking changes are caused by refactoring [42]), and propose potential solutions to these problems (e.g., automatically detect API refactoring and replay them for clients in [42]). However, these papers are proposals and do not provide complete solution details and do not evaluate the proposed solution.

Answers of RQ1: Tools and techniques proposals related to API evolution typically seek to highlight existing concerns in the field, and provide potential approaches to resolving these problems.

4.1.4 Surveys. Like this research paper, surveys of existing literature seek to present a fair evaluation of a research topic by using rigorous methods [81]. The surveys presented in this paper typically start with a research topic and observe existing literature to provide a view of the topic at hand. Our dataset contains five surveys related to API evolution.

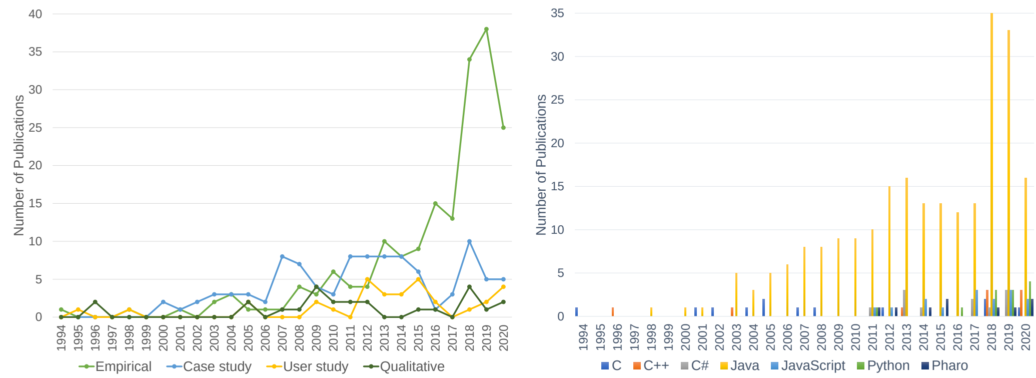
In his 2016 survey on software ecosystems research, Manikas [102] seeks to provide updated evidence to determine and document evolution in the field of software ecosystems. The survey shows evidence that the evolution of software ecosystems draws the attention of numerous papers [102].

As part of a book by Robillard [148], Mens and Lozano produced a chapter on Source Code-Based Recommendation Systems [108], and Kim and Meng [148], produced a chapter on Automating Repetitive Software Changes. These chapters can be independently obtained through the Springer archives, and we consider them to be two separate surveys of specific areas of recommendation systems since they are presented as such in *Recommendations Systems in Software Engineering*. Both of these chapters seek to provide state-of-the-art insight into specific recommendation Systems. Kim and Meng provide a general view of five source code-based recommendation systems and the in-depth design of one system to provide insight into the design decisions that are made when creating source-code based recommendation systems [148]. The chapter by Mens and Lozano seek to present state-of-the-art approaches that can be used to automate repetitive software changes [108].

In their survey of automated API property inference techniques, Robillard et al. [145], seek to provide an overview of API property inference techniques to present properties inferred, mining techniques, and empirical results of API property inference techniques [145].

In his survey on software merging, Mens [109], seeks to present a comprehensive analysis of available software merging approaches. The finding presented in this survey are directly applicable to API evolution topics such as API migration tools where merging techniques can be used to help automate API migration [106].

Answers of RQ1: Survey papers, like this systematic literature review, typically seek to present an overview of a subject using existing literature to provide clarity for their given subject and allow for effective stepping-stones for future research. The survey papers we reviewed consider subject matters related to API evolution without concentrating on API evolution itself.



(a) Type of evaluation used over time

(b) Language used for evaluation over time

Fig. 6. API evaluation trends

4.1.5 Datasets. Out of the 369 papers investigated for this study, we uncovered three papers that we labeled as dataset papers. Which concentrate on building a dataset related to some aspect of API evolution (e.g., Linux system calls [11]). The datasets are produced to conduct further studies [11], advance the state-of-the-art [3], and improve reproducibility of research [155].

4.2 API Evolution Research Evaluation

We seek to determine how API evolution research is typically evaluated. API evolution research often requires more than manually observing an API. Studies rely on distinct evaluation methods and make use of various software metrics to evaluate their results. Details for the various types of evaluations performed in API evolution can be found online.[4]

We identify four major means of evaluation used for API evolution research. Empirical evaluation, where quantitative metrics like LOC (lines of code) or precision and recall are used for evaluation over multiple subject systems. Case studies, where a single subject system is used to obtain subject related metrics and results. User studies, that employ survey techniques and interviews with developers or users. Finally, qualitative evaluation which relies on subjective interpretations. Figure 6a presents the evolution trends of these four evaluation means. We concentrate on the five paper types and identify the evaluation methods and the metrics that are used in these papers.

We identified 31 different evaluation metrics used in our publication sample. We assembled the metrics that occurred fewer than five times and were not known statistical properties (e.g., AUC, Confidence interval) into more global metric types, such as *absolute value metrics*, *qualitative metrics* and *other*. Thus, we obtained 9 metric types. Figure 7a shows their yearly trends.

Using the data we uncovered, we can see that although more rigorous evaluation metrics such as precision, recall, AUC, and F1 score appear to be gaining in popularity, a large percentage of papers still use a variety of non-standard absolute value metrics. A wide range of absolute value metrics are used to evaluate experiments and tools such as method parameter count, method changes, popularity, community size, project maturity, number of years active, fix rate, number of restarts [8, 59, 84]. None of these metrics are flawed but the lack of standardization makes it difficult to compare similar experiments and determine if progress is being made.

4.2.1 New Tools and Techniques. As presented in Section 4.1 the majority of the papers fall within the scope of new tools and techniques. A surprising number of API evolution tools and techniques do not formally evaluate their tool. In most of these cases, the tools appear to have been evaluated by

[4]https://github.com/senseconcordia/APIEvolutionSurveyEvaluation

the authors of the paper, however no formal evaluation is provided, e.g., when the tool is presented as part of a short paper and is evaluated as part of a second paper. This is the case for SemDiff by Dagenais et al. [34]. The reader must be vigilant to obtain the evaluation of a tool.

Most API evolution tools are evaluated for their accuracy. In older papers, this accuracy was simply reporting the true positive rate [19, 105, 140]. Recent papers reported precision, recall, F1-score, and the area-under-the-curve (i.e., AUC) [30, 71, 174, 175, 187]. Figure 7a shows that, in the last decade, papers begun using more standardized metrics for their experiments.

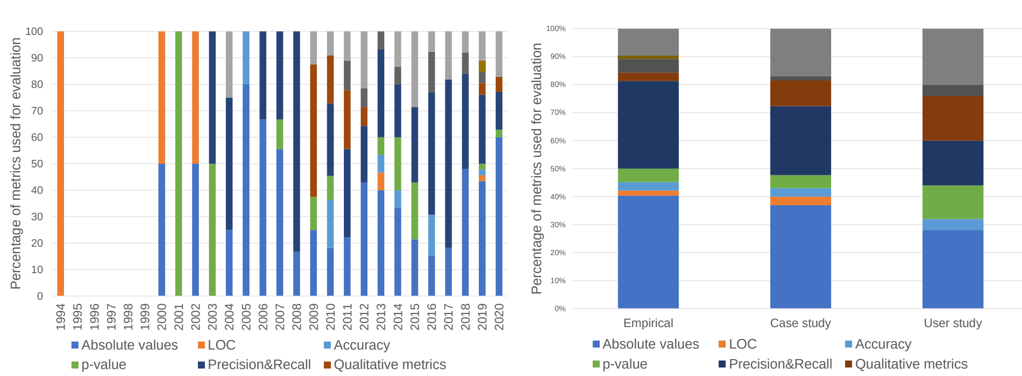
In some cases, it is not possible to ascertain the recall of a measure (e.g., in the case of mined framework usage changes [159]), then authors normally concentrate on providing precision metrics instead [33, 159, 165, 181, 189], which is particularly prevalent in data mined from large repositories for which it is impossible to manually determine if any instances were missed by the approach. It would be possible to remedy to this situation with high quality open-source datasets that have been manually vetted by experts.

4.2.2 Empirical Studies. All of the empirical studies relied on quantitative analysis to evaluate their results. The metrics evaluated depend on the study, ranging from changes in APIs (e.g., addition, modification, removal) [95], changes in lines of code [104], code smells [61], API popularity [22], errors [115]. The most pervasive API evaluation criteria is absolute changes in API methods (e.g., changes to numbers of deprecated APIs, APIs added, APIs removed, APIs modified).

As shown in Figure 7b, case studies present a variety of evaluations. Some papers [84] compare various metrics like added APIs, deprecated APIs, removed APIs. Quantifying API changes through added/modified/removed APIs [51, 84] appears to be common for API evolution case studies.

However, although most API case studies consider and quantify API changes, some also rely on qualitative evaluations [14, 36, 143]. For example, one study [36] identifies six promises and seven perils of ported visualization tools such as promising to provide feedback about errors. This qualitative information must be manually extracted by the authors.

Case studies appear to be well suited to uncover new evaluation metrics for APIs to uncover previously unknown information such as the promises and perils of ported visualization tools [36], the types of ripple effects caused by changes in software ecosystems [143], and API migration issues [14]. It is therefore expected that case studies present more uncommon absolute value metrics and other metrics, because these studies might be attempting to identify new metrics. The information uncovered through case studies can later be used in larger scale empirical studies of various APIs, for example to determine the impact of API migration issues on various APIs [196].



(a) Evolution of evaluation metrics over time

(b) Metrics used for different evaluation types

Fig. 7. API evaluation metrics

4.2.3 Tools and Techniques Proposals. Reports from talks or expert panels of API evolution concentrate on coarse grained issues and challenges that plague the field of software APIs. These papers concentrate on abstracted problems taken from existing literature, and rarely evaluate their methodology. Most papers that concentrate on future research avenues [16, 78] and paradigm shifts [149] do not present evaluation criteria.

However, some exceptions exist. A report on web APIs, concentrates on challenges in the field, but also suggests looking into metrics like latency to benchmark performance [180]. Similarly, papers on recommended practices [88] concern specific software metrics that could be improved through developer knowledge (e.g., reducing coupling) [88]. Finally, a tool proposal contains an evaluation for their tool through accuracy metrics, and a user study [42].

4.2.4 Surveys. We observe two types of survey papers related to API evolution. The first type concentrates on existing literature, for example a survey on automated API property inference techniques by Robillard et al. [145] surveys existing techniques and provides a summary of these techniques. Surveys of this type do not appear to rely on metrics to evaluate the papers presented in their findings. These papers instead rely on the evaluation presented in each of the papers surveyed. Furthermore, each survey of this type identifies a particular scope and specific criteria that must be respected throughout the study, criteria which are manually evaluated by the author(s). Similarly, in this systematic literature review, we also rely on the evaluations presented in our sampled papers. However, we also use quantitative information to uncover publishing and evaluation trends, as well as determine the emergence of API evolution sub-fields.

The second survey type provides the results of questions used to extract data from participants. These papers present quantifiable data that can be evaluated in various ways. For example, one paper [144] provides raw data for responses to survey questions within the related paper. Furthermore, the responses to the survey questions are quantitatively evaluated by the author [144]. Meanwhile, other works [48] survey the behavior of programmers to specific tasks. This behavior can be quantified through statistical measures such as standard deviation, Z-score, and p-values [48]. Current evaluation methodologies appear to be tailored to specific papers with no standardized dataset or evaluation methodology used for API evolution surveys. This lack of standardized evaluation methodology should be addressed by the community since it hampers research comparison and therefore makes it difficult to determine when and where progress has been made.

4.2.5 Datasets. We found three papers presenting empirical datasets. Datasets related to API evolution, are proposed to stimulate research [155], and to improve the state-of-the-art [3].

Datasets are not always fully evaluated because fully verifying large datasets can incur a heavy manual overhead. Therefore, some datasets do not present any immediate evaluation [155], some datasets are fully manually verified by multiple individuals [3], and some datasets are evaluated through manual verification of a statistically significant sample [11].

Answers of RQ1: Similarly to API property inference techniques [145], empirical evaluation in API evolution studies in general has not yet converged to specific styles and metrics. A surprising number of API evolution tools and techniques do not present any empirical evaluation while studies with similar tools and techniques evaluate precision, recall, f1-score, and AUC. Meanwhile, API evolution empirical studies rely on various metrics with absolute changes in API methods appearing most often, but not always. Survey papers, tools and techniques proposals, and dataset papers similarly present a variety of evaluation criterion with no clear standards. While some flexibility is indeed required to allow for various research goals, there is still work to be done to evaluate similar research goals using similar evaluations styles and metrics.

Table 5. APIs used most commonly as evaluation subjects

API	Frequency
Android	52
Java API	51
Toy systems	21
Eclipse	18
JHotDraw	12
Log4j	12
Proprietary systems	12
Struts (1&2)	9
JUnit	9

API	Frequency
Guava	8
Hibernate	8
JFreeChart	8
Lucene	8
Spring	8
Hadoop	6
Pharo	6
.Net API	5

4.3 API Evolution Experimental Subjects

We now finish answering our RQ1 by providing further insight into API evolution evaluation by presenting the APIs that are most commonly used as evaluation subjects. We concentrate on APIs that are used as evaluation systems in at least five different studies within our sample set. The frequency of API under evaluation in our sample set is presented in Table 5. While comparing the frequency of the various APIs used as evaluation subjects by prior work, we highlight benefits and reasons for choosing specific APIs as evaluation subjects.

We find that the majority (346) of the studies in our dataset employ at least one API to evaluate their hypotheses. 127 of these 346 studies employ multiple APIs to allow the generalization of results across multiple systems or multiple programming languages.

27 out of 369 publications either do not present or do not use an API to test hypotheses. The lack of test systems may be due to the nature of the publication. For example, survey papers concentrate on summarizing the state of prior work [102, 108, 109, 145]. Similarly, book chapters [148], papers about general programming practices [88], future research proposals [46, 149], and hypotheses about the future of software engineering [78] do not employ APIs. Some tool proposal papers do not provide any tests when simply presenting the tool [40, 42, 120, 163, 164, 186]. Similarly, exploratory research with theoretical findings does not always provide tests [1, 16, 180]. Finally, some research uses theoretical proofs to ascertain their results, and prove the validity of their approach without tests [29, 94, 150, 178].

When considering that the Android API is primarily Java, and that most toy systems (12 out of 21) used within our sample are created by using the Java programming language, we find that API evolution research is heavily skewed towards the Java programming language. As shown in Figure 6b, 236/369 papers (63.9%) are exclusively evaluated with Java systems between 1994 and 2020. However, this trend seems to be shifting in recent years. The second most common programming language is JavaScript with 15 papers using JavaScript APIs, since 2011, to evaluate their findings. Figure 6b presents the evolution trends of programming languages used in the evaluation of API research from 1994-2020. We only include programming languages that were used for more than two publications within our sample. Table 5 presents APIs that are used as test in more than five different research papers.

Answers of RQ1: API evolution evaluation is skewed towards the Java programming language: 236/369 papers (63.9%) used exclusively Java APIs. This presents opportunities for replication studies as well as potential avenues for future research with other programming languages that differ from Java, e.g., Python or JavaScript.

Answers of RQ1: How has the field of API evolution research evolved? API usability and maintainability are API researchers' main research goals and subjects of empirical studies. Proposals and surveys pertain to all aspects of API evolution. Empirical studies, surveys, and datasets on API evolution use various evaluation methods, criteria, and metrics. API evolution evaluation is heavily skewed towards the Java programming language.

5 SEMINAL AND RECENT PUBLICATIONS

To answer *RQ2: What is the current state-of-the-art in API evolution research?*, we first present publication trends within the state-of-the-art in API research. We then concentrate on the seminal and most recent concepts and research works. We chose these seminal works based on the novelty of their content, the number of works that present similar ideas and build on these seminal works. We divide this section by publication contribution type as in Section 4.1.

5.1 New Tools and Techniques

Over the years a variety of tools and techniques have been developed to ease the burdens caused by API evolution. In general, we find that tools and techniques appear to primarily concern themselves with Lehmans law of *Conservation of Familiarity* while other laws such as *Continuing Change*, *Increasing Complexity*, and *Invariant Work Rate* serve as challenges to the *Conservation of Familiarity* [89]. We separate API evolution tools and techniques into general topics such as *documentation* [35], *examples* [186], *misuse* [5], *migration* [43], *recommendation* [108], *usage* [145], and *other*. As presented in Section 3.5.2, these tool topics were either identified in prior surveys [102, 108, 109, 145, 148], or by using publication keywords, titles, abstracts as well as our own judgement. We provide a general overview of the state-of-the-art for each tool topic.

API Documentation Tools: API documentation has been described as large and cumbersome [38], lacking and difficult to produce [56], but instrumental to success [146]. State-of-the-art tools and techniques use Stack Overflow posts to supplement documentation for lexical queries [71], augment documentation by automatically detecting APIs in the documentation [171], employ dynamic specification mining to improve decaying documentation [2], identify misuses in documentation to warn users [92], generate high quality source code summaries [98], and employ neural networks to produce high-quality text-to-code [119], code-to-text, and code-to-code retrieval [123].

API Examples Tools: API examples have been touted as helpful to understand how APIs work [105, 113]. Approaches such as MAPO [199] and Jungloid [101] mine API examples from existing code. Approaches such as Examplore [58] employ relational topic models to produce API examples that span multiple files. Techniques using bytecode analysis [105], framework extension points [33], and software visualization [26] have also been used to identify API examples.

API Misuse Tools: API misuse tools primarily attempt to identify unfavorable API uses that could lead to future problems [4]. Approaches use machine learning [142], mutation analysis [177], specification mining [138], and API-usage-graphs [4], to attempt to detect misuses.

API Migration Tools: CatchUp! [40, 64] was one of the original approaches to deal with the problem of API migration. It captures API refactorings produced by API developers and synthesizes an edit script that can be replayed on API user code. Similar approaches were created where edit scripts could be manually created by the API developers [13] rather than recorded.

JDiff [6] is one of the first tools to synthesize a report of API changes between two versions of an application. It presents additions, removals, and modifications to any API. This information can be used to automatically track changes made to APIs. Similarly, ACUA [183] analyzes the binary code of both frameworks and client's programs written in Java to identify API changes, generating a report to estimate migration workload.

SemDiff [34] was one of the first approaches to use call dependency analysis to map APIs between two versions and determine a migration path between two or more migrated APIs. AURA [184] combines call dependency and text similarity analyses to identify API change rules between two versions. HiMa [107] uses revision control to create framework-evolution rules, which are then used to migrate user applications, outperforming both SemDiff and AURA. Approaches such as LASE [106] create a context-aware edit script from two or more examples and use the script to automatically identify edit locations and transform code. Recent API migration tools and techniques employ abstraction layers [57], knowledge extracted from API clients' evolution [158], and syntactic changes [20] to improve API migration techniques.

Tools and techniques also exist to migrate across programming languages rather than application version [24]. The state-of-the-art in this domain currently employs generative adversarial networks to produce high quality API mappings across languages such as Java and C♯ [24].

API Recommendation Tools: Identifying useful APIs can be a challenge for API users [118]. API recommendation attempt to ease the burden of selecting the most appropriate API by automatically recommending potentially useful API [141]. Various tools and techniques have been proposed to recommend useful API methods [30, 47, 135], and parameters [7]. Current state-of-the-art approaches rely on converting English text queries and documentation to API elements [12, 125], ranking existing API recommendations by leveraging API usage path features [99], version history and Stack Overflow posts [10, 190].

API Usage Mining Tools: Most of the tools and techniques are primarily targeted at API users. However, API usage mining tools are particularly suited to API researchers and API developers. These tools attempt to uncover various API usage metrics from API user projects and examples. These tools and techniques are meant to determine API usage for a variety of reasons. These reasons range from determining the most useful API methods [163], to improving API productivity [116]. We identified tools that automatically identify refactoring with high precision and recall [44, 168]. Tools that can automatically identify API that will be made public in the future [69], and tools that can extract fine-grained API usage [156].

Learning to use APIs appropriately is challenging [46]. Several attempts have been made at easing the learning curve of APIs by automatically improving online question/answer forums either through automatic answers [151], or by providing more information about the APIs themselves [132]. Other approaches use of machine learning approaches to extract and provide API tips to users [175]. There are also techniques that infer structured descriptions of web APIs from web examples [166].

Other API Tools: Not all tools fit in the categories presented above. Some tools present solutions to niche problems to help verify the impact of APIs on program correctness [164, 176, 181, 182], software security [73, 140, 200], and software quality [18]. We also found papers that detect deprecated APIs [201] and API reuse patterns or code clones [66, 122, 198] to identify useful patterns for API users and APIs to improve for API developers. Finally, tools have been created to apply standards to REST API [91], test cloud APIs [8], and develop adapters for web services [17].

Answers of RQ2: State-of-the-art tools and techniques related to API evolution seek to, in order of importance, improve API usage, help adapt to changes, provide automated API migration, provide API recommendations, reduce API misuse, and provide better API documentation and examples.

5.2 Empirical Studies

We uncover two main subjects in the state-of-the-art API evolution empirical studies, those that concentrate on API usability and those that concentrate on API maintenance.

API Usability: Many papers look into various aspects of API usability to reduce complexity [89]. These papers concentrate on issues such as breaking changes, integration problems, how API are used and what makes APIs hard to use, API standards, API misuses, and API documentation.

Current empirical studies in breaking API changes suggest that there is a growing need to document acceptable usages for APIs [179]. Furthermore, it is suggested that non-atomic refactoring patterns used by API developers can reduce API migration burdens [179]. Non-atomic refactorings in this case are defined as introducing a new API and changing the existing API piecemeal until there is no more use of the old API [179].

On average dominant topics on forums can cover at least of 50% of questions pertaining to web API integration [172]. It is possible to unbundle software APIs in different ways to vary the uniqueness of API bundles [103].

Finding good names, relations between API types, knowing the impact of API flexibility and accurate documentation are all needed for good API usability [134]. API users claim that discovering allowable types is difficult, thus tools to suggest allowable types could benefit users [45]. APIs do present meaningful local interaction patterns that can be used for future recommendations [65]. Developers have a hard time understanding reflections API, and only produce tests after a bug is reported [136]. Developers use examples to understand how APIs work. They also need to understand the general idea of how an API works [144]

Recent papers have uncovered 22 patterns that determine what makes an API less usable [202]. Programming language [197] as well as tools, information, and boundary resources such as community are very important when selecting an API [117].

Issues pertaining to API standards [92, 114, 178] affect the usability of web APIs [51]. Deprecation in particular has been found to vary mechanism, support, and implementation and fail to fully address the needs of developers [154]. Performance issues in mobile apps has been studied and carefully designing storage, limiting the MVC pattern, and limiting widgets are all factors that improve app performance [96].

Various works have studied API misuses [9, 79]. 11 different types of API fault cases have been identified [9]. Most cases have been attributed to missing data [9]. However, a lack of semantic awareness and correct usage examples lead to many false positives in API misuse detectors [5].

Many papers concentrate on API documentation motivated by incomplete documentation [49], the challenge of producing good documentation [129], and the shift of API documentation to more social sources [128]. A case study with Github and Stack Overflow to locate information from 10 popular APIs found that Github and Stack Overflow are often used by Google to document new functionalities [167]. An empirical study that combines API patterns extracted from GitHub projects to determine if Stack Overflow posts present faulty API code, found that up to 31% of posts may have potential API violations [193]. Languages with static typing and documentation are much easier to use than dynamic languages, with or without documentation [49]. Documentation incompleteness and ambiguity plague developers in a user-study to determine what causes developers to use other APIs [169]. Almost all usage constraints are present in API source code but not in documentation [152]. An empirical study of automatic knowledge extraction techniques to extract knowledge from API documentation found that SVM and deep-learning methods can be complementary when attempting automatic knowledge extraction [55].

Answers of RQ2: Empirical studies on API usability typically concentrate on how API are used and what makes APIs hard to use this includes challenges, such as, in order of importance, breaking changes, integration problems, and API standards. API usability studies also concentrate on API misuses, API examples, and API documentation.

API Maintainability: A large number of empirical studies related to API evolution concentrate on the maintainability of APIs to conserve familiarity as APIs evolve [89]. More precisely, papers mainly concentrate on aspects such as deprecation, reuse patterns, the speed at which APIs change, and the effects of propagating these changes.

A user study found that developers who use unstable Eclipse APIs often do not read documentation and therefore do not know which API are deprecated [27]. Empirical studies have been conducted to determine how effective documentation is at solving deprecation problems. Most documentation does not cover alternative APIs and code examples are very rarely documented [82]. However, in the case of the Android API, deprecated entities are removed in a timely manner, and the Android API recommends alternatives; yet most deprecated APIs in Android are in popular libraries [95]. Another empirical study determined that there is no major effort to update deprecation messages in most projects and that deprecated messages depend on the size and community of the project [23]. They found that only 64% of API elements that are deprecated have replacement messages, and that there is no effort to improve this over time [22]

Empirical studies have been conducted to detect reuse patterns and software clones to improve maintainability [76]. Patterns of API reuse have been identified in various code samples (e.g., opening and closing files) [110]. A decline of popularity appears to indicate that something is wrong with an API [112]. Some studies have shown that over 80% of breaking changes in API are due to refactoring [43], however other studies have since disputed this claim [31]. Refactoring APIs has however shown a tendency to increase the speed at which bugs are fixed [80].

Empirical studies that concentrate on the side effects of rapid API evolution found that using new APIs that are highly touted may be a counter-productive practice [139]. Although the potential for problems to occur due to developers updating to newer library versions without modifying any of their source code is high, these problems tend not to occur on a wide scale in practice [39]. 28% of android references are out of date. 22% of outdated API usages eventually upgrade to newer API versions but this takes about 14 months [104]. Mostafa et al. [115] found that most API incompatibilities are not well documented, and 67% of client bugs linked to backwards incompatibility can be fixed through simple client changes [115]. Furthermore, over 88% of Android apps follow the same workaround pattern to fix Android version issues, and this pattern can sometimes lead to incorrect behavior [63]. Studies have suggested that developers believe there is a direct relationship between adopted APIs and user ratings [15]. Web services follow a spike and calm cycle of maintenance, an empirical study into Amazon services determined recommendations to make the most of spike and calm cycles from a developer point of view [191].

API evolution empirical studies have been used to determine different patterns of evolution for web APIs [93]. APIs change due to needing more functionality and usability [60]. Most API developers appear to introduce breaking changes to simplify the API and introduce new functionality [21]. Meanwhile, library maintainers are less likely to break API classes used by many clients [83]. API users update API versions and only use deprecated entities less than 20% of the time. However, most users do not react to deprecation, but remove API references when something gets deleted [157]. 14.78% of API changes break compatibility and impact 2.54% of clients [185]. Systems with higher break frequencies are usually larger and more popular [185]. Another empirical study similarly finds that about half of API changes cause reactions in only 5% of clients and that the overall reaction time is slow [67].

Studies have shown that mobile development questions increase when new versions of Android are released, and these questions appear to concentrate on deleted methods [97]. Meanwhile, mobile devs rarely update their apps, and when they do, it is likely with respect to GUIs [153]. API updates are ignored due to poor awareness of benefits and high cost [153].

The results of empirical studies lead to the recommendation of semantic versioning, self-documenting APIs, publishing customized change-logs with discussion forums for changes [160]. Furthermore, web APIs should not change too often, old versions should not linger, API developers should keep usage data, blackout tests should be used, and providing examples is useful to users [53].

Answers of RQ2: Empirical studies on API maintainability typically concentrate on challenges, such as the speed at which APIs change, change impact, reuse patterns, and API deprecation.

5.3 Tools and Techniques Proposals

The tools and technique proposals primarily concentrate on highlighting an existing problem and proposing potential solutions for future work. API evolution tools and technique proposals concentrate on the future of API evolution research. The more recent proposals highlight the need to differentiate between web APIs and library APIs [180] and to develop digital assistants to map user intent to ever more numerous APIs [16]. Furthermore, one particular proposal concentrates on a vision of automated developer documentation [149]. It highlights challenges such as establishing precise links between artifacts, capturing document request context, and the summarizing and synthesis of documents [149]. These proposals are particularly useful to understand the current demands of researchers and developers.

Answers of RQ2: Tools and technique proposals discuss differences between Web and library APIs, automated documentation, and automated traceability between APIs and other software artifacts.

5.4 Surveys

Surveys highlight seminal concepts and state-of-the-art work by design. As previously mentioned in Section 4.1, we found five survey papers pertaining to API evolution using the methodology highlighted in Section 3.

These papers highlight the state-of-the-art in recommendation systems pertaining to API evolution [108, 148], software ecosystems [102], API property inference techniques [145], and software merging techniques [109]. We use metrics, classifications, and challenges uncovered by prior surveys [102, 108, 109, 145, 148] to reinforce our own findings, and to categorize tools and techniques employed in API evolution studies, and empirical studies into publication types in Sections 5 & 6.

The survey papers also highlight open problems and future research directions in their respective domain. Some open-questions have been solved since the publication of the surveys. However, some challenges are still open, and we re-iterate these along with our own findings in Section 6.

Answers of RQ2: Surveys associated to API evolution tend to highlight the state-of-the-art in research as well as current research challenges and future research directions.

5.5 Datasets

Papers that primarily concentrate on datasets are oriented towards replication, and future studies. In the three datasets in our study, the data presented is recent (2015-2018) and available online to be kept up to date and relevant to API evolution studies.

We identified a dataset constructed from the observation of a decade of Linux system calls [11]. This dataset presents 8,870 classified system call related changes. Another dataset presents 1,482,726

Challenge	Proposed State-of-the-Art Solution
Dealing with breaking API changes	Document acceptable usages [179]
	API bundles [103]
	Automated API migration [24, 57]
	Extracting migration knowledge from clients [86, 158]
Improving API usability	Local interaction patterns [65]
	Usability patterns [202]
	API selection criteria [117]
	Digital assistants [16]
	Automated API tips [175]
	Automated documentation [2, 71, 98, 149, 171]
	Example mining [58, 116, 168]
	API recommendation algorithms [12, 99, 125, 190]
Reducing API misuses	Misuse detectors [4, 5, 9]

Table 6. State-of-the-art solutions to existing API evolution challenges

method invocations related to 5 Java APIs (Guava, Guice, Spring, Hibernate, EasyMock) created by mining 20,263 projects on GitHub [155]. Both of these datasets target research in software APIs to improve the state-of-the-art in future API studies.

The final dataset specifically concentrates on API misuses [3]. This dataset contains 89 API misuses collected from 33 projects and a survey. The primary goal of the benchmark is to evaluate API-misuse detectors, which will then allow fair comparison between various approaches [3].

We consider three papers that present datasets as primary contributions. However, papers listed under different primary contributions (e.g., Empirical studies) could have a dataset as secondary contributions. For example, there are papers that contribute approaches [156], or empirical studies [61] but also include datasets. Making research datasets open-source is becoming more popular.

State-of-the-art datasets are vetted, open-source sources of data for replications: API invocations [155], Linux system calls [11], and API misuses [3] are available for API evolution research.

Answers of RQ2: What is the current state-of-the-art in API evolution research? We described seminal and recent API evolution works. Table 6 summarises their challenges and state-of-the-art solutions. They are concerned by breaking changes, usability, and misuses. They want to ease API usage, API changes, API migration. They also want to provide API recommendations, reduce API misuse, and document APIs. They suggest that future works should concern Web APIs, automated documentation, and automated traceability between APIs and other software artifacts.

6 CURRENT AND FUTURE CHALLENGES

To answer *RQ3: What are the current and future challenges related to API evolution?*, we manually identify existing API evolution research challenges and also uncover unsolved ones (presented in Table 7). Indeed, although API research has grown rapidly in the last decades, and several avenues of research have shown promising results and tools, there are still many unsolved challenges related to API evolution. Challenges in API evolution research are scattered in the literature, which hides advances and also cloaks important, remaining challenges.

While producing this literature review, we kept a record of challenges that are mentioned in publications. Using this record, if we find publications that attempt to resolve these challenges, we consider them existing challenges (EC). However, if a challenge is mentioned, but no solutions currently exist we consider the challenge to be an emerging or unsolved challenge (UC). We manually added further challenges to these unsolved challenges by using the insights that we gained throughout this literature review.

Table 7. Open challenges in API research

Challenge types	Paper types	Challenges
Existing challenges	New tools and techniques	*EC-1* Combining textual merging with syntactic and semantic approaches [109]
		EC-2 Providing a commercially viable API migration solution [20, 34]
		EC-3 Incorporating domain specific information into tools [109]
		EC-4 Using systematic evaluation methods in empirical evaluations [145]
		EC-5 Producing more specific and less abstract theories [102]
		EC-6 Reducing the variability of software API studies [102]
		EC-7 Finding input examples for API migration through examples [148]
		EC-8 Improving the granularity of API migration approaches [148]
		EC-9 Validation and correction of API migration edit scripts [148]
		EC-10 More tools to help with Web APIs [180]
		EC-11 Using existing library API research as stepping stones for Web APIs [180]
		EC-12 Combining both API side learning with client side learning [158]
		EC-13 Dealing with out-of-vocabulary problems [24]
	Empirical studies	*EC-14* Defining best fit APIs [192]
		EC-15 Automatically identifying factors that drive API changes [60, 70, 191]
		EC-16 Dealing with API semantics and dependencies [5]
		EC-17 Deploying bug fixes to multiple API versions [160]
Unsolved Challenges	New tools and techniques	*UC-1* Using uniform benchmarks for API tool evaluation
		UC-2 Supporting the context sensitivity of API migration tools
		UC-3 Improving performance of API tooling to allow user adoption
		UC-4 Dealing with fuzzy and ambiguous developer intent
		UC-5 Reducing the knowledge gap between API users and developers
		UC-6 Tools that mine usage data help API developers improve APIs
		UC-7 Keeping API users in the loop for API recommendation systems
		UC-8 Generalizing API tools to languages other than Java
		UC-9 Tools to help API developers deal with API migration, not just users
		UC-10 Reducing API misuse from the API development side
	Empirical studies	*UC-11* Understanding the coupling between APIs and programming languages
		UC-12 Determining API migration and API recommendation impacts
		UC-13 Generalizing API empirical studies to languages other than Java
		UC-14 Comparing the evolution of various APIs
	Datasets	*UC-15* Creating large scale API migration and recommendation datasets

We identify existing challenges for API evolution research on new tools and techniques, empirical studies. We uncover no challenges from proposals or surveys. Similarly, we did not uncover existing challenges from datasets, only unsolved challenges. Based on our findings, we believe that Lehman's 8th law, namely *Feedback System* [90], poses the largest hurdle to future API evolution research.

6.1 New Tools and Techniques

Existing Challenges: **Issue:** Most of the tools presented in this report concentrate on library APIs; little effort has been done on Web APIs [180] (*EC-10*). While Web APIs differ from library APIs, their users must concern themselves with quality of service, weak specifications, and a lack of comprehensive listings for Web APIs [180]. Web APIs do suffer from API migration, API documentation, and API example problems, but their research prevalence is sparse. **Propositions:** Researchers should use existing research, such as API migration approaches [6, 20, 24, 34, 57, 106, 158, 183, 184], high quality code summary generation [98], misuse identification [92], and using relational topic models for examples [58] as stepping stones to improve Web API tooling (*EC-11*).

State-of-the-art migration techniques should consider hybrid approaches (*EC-12*) to combine both API side learning with client side learning [158] and consider the use of domain adaptation methods (*EC-13*) to deal with out-of-vocabulary problems, a current API Evolution issue [24].

Issue: API migration, API recommendation, and API misuse detectors still have room for improvement. **Propositions:** These challenges require keeping the API users in the loop, because they are ultimately the ones most impacted by these problems. Furthermore, tools that attempt to aid with these problems should aim to support more programming languages and Web APIs.

Unsolved Challenges: **Issue:** Many tools and techniques have been created to deal with API evolution challenges. However, most tools concentrate on a small range of challenges and do not fully consider feedback loops involved in API evolution. Although individual tools show promising results [4, 34, 58], none can claim to be 100% effective at solving their target problem. Machine learning approaches are now emerging as potential solutions to key API evolution issues [25, 124], it remains unclear whether current approaches are good enough for user adoption, whether they can be applied to all issues, or if performance should still be improved before users can start using these tools (*UC-3*). Fuzzy and ambiguous intent (*UC-4*) as well as the rapid evolution of software services that employ APIs, such as IoT devices, are challenges that concern evolving APIs [16]. **Propositions:** Effective API engineering must find solutions to deal with technical problems caused by APIs, and to reduce ambiguity of APIs [132] and the knowledge gap between API developers and users (*UC-5*). New tools are needed to help API developers create APIs that are easy to use by API users [144] (*UC-6*), just like better techniques are required to help API users understand how to use APIs [147] (*UC-7*). Both of these challenges are dependent on researchers understanding what constitutes a "good" API, and why API users select one API over another.

Issue: Many tools want to expand to more programming languages [68, 71, 116, 175, 189]. However, most are still developed for Java. Figure 6b shows an emerging shift to other programming languages in recent years. However, it remains to be seen how effectively API evolution tools would translate to other programming languages (*UC-8*).

Issue: API migration received a great deal of attention in API evolution research. However, it is still an open problem. Most existing approaches concentrate on the client side, with the premise that API migration is the burden of API users [87, 106, 133]. **Propositions:** Research should be done to determine if it would be more efficient to transfer some of the burden to API developers (*UC-9*) (e.g., have API developers provide migration scripts like Python[5] versions 2 to 3), and develop tools to improve API engineering such that API migration efforts are reduced on the client side.

Issue: Several tools have been developed to extract API misuses [4] and API usage [105] (e.g., API call frequency). **Propositions:** Research should concentrate on using usage and misuse information to create a feedback loop to help API developers improve their APIs (e.g., using API workarounds as improvement areas [86]) (*UC-10*). Most of the API research conducted in the last two decades concentrated on API users rather than API developers.

6.2 Empirical Studies

Existing Challenges: **Issue/Proposition:** Studies uncovered the need for future work on API developers and API development for supporting the evolution of APIs [52, 143], defining best fit APIs [192] (*EC-14*), and automatically identifying factors driving API changes [60, 191] (*EC-15*).

Issue/Proposition: In their study on API misuse detectors, Amann et al. [5] highlight the need for future studies into program semantics and dependencies (*EC-16*), as well the need for tools that properly handle alternative patterns for the same API.

Issue/Proposition: The need for tools to deploy bug fixes to several versions of an API at once (*EC-17*) has been proposed by Sohan et al. [160].

Unsolved Challenges: **Issue:** Most (66%) API evolution empirical studies concentrate on APIs written in the Java programming language. Other languages such as C, C++, C#, JavaScript, Python are only covered by a small percentage (≤ 5% each) of empirical studies. **Proposition:** Future studies should generalize to languages other than Java (*UC-13*).

Issue: A great number (74%) of empirical studies do not rely on any statistical tests to evaluate their results. The majority of these studies present metrics such as lines-of-code (LOC) or the

[5] http://python-future.org/automatic_conversion.html

numbers of field/method/class changes, but there is no current way to normalize these results to compare them across studies or APIs (*UC-14*). **Proposition:** It remains an open challenge to compare the evolution of various APIs, particularly across programming languages.

6.3 Datasets

Unsolved Challenges: **Issue:** We identified three papers on datasets. Although it has become more popular in recent years to publish datasets (all four datasets presented in this paper were published after 2015), the field suffers from a lack of accepted and up-to-date datasets. For example, 13 papers concentrate on API migration tools and techniques, however, we could not identify any common dataset or API to directly compare migration tools or studies.

Proposition: API evolution research would greatly benefit from more datasets, particularly for API migration and recommendation (*UC-15*). Such datasets are challenging to create because some API migrations and recommendations are subjective and context sensitive.

6.4 Others

Other goals of research on API evolution, tools and technique proposals and surveys, are scarcer so we discuss them together in this section.

Existing Challenges: **Issue/Proposition:** In his survey on software merging [109], Mens highlights a need for tools that combine textual merging with syntactic and semantic approaches (*EC-1*). Since attempted in API migration tools like SemDiff [34] and APIDiff [20]. However, these tools have yet to provide a commercially viable solution (*EC-2*). Mens further highlights the need to incorporate domain specific information, which has also been attempted by various API migration tools, with various levels of success (*EC-3*). However, current solutions appear context sensitive.

Issue/Proposition: Robillard et al. [145] found that the empirical evaluation of API properties is lacking in systematic evaluation methodology (*EC-4*). Although their survey determines a foundation to compare API property inference techniques, this methodology has yet to rise. It is unclear why this foundation has yet to take hold. Perhaps due to a lack of exposure, or because there are hurdles imposed by the proposed systematic evaluation methodology. We hope to bring attention to this challenge amongst others, to improve the exposure of existing proposed evaluation methodologies, and guide future research into more systematic and comparable evaluations.

Issue/Proposition: Manikas et al. [102] posit that theories about software ecosystems and the APIs they involve can often be either too general (*EC-5*) or too abstract. Manikas highlights that it is difficult to study software ecosystems due to the high variability in the field, APIs which are part of these ecosystems are therefore similarly impacted by high variability (*EC-6*).

Issue/Proposition: Robillard et al. [148] highlight several open challenges with respect to automating repetitive software changes. Finding input examples to automate software changes remains an open problem (*EC-7*). Integrating testing with code recommendation and dealing with various levels of code granularity (*EC-8*) for API recommendations and migrations also remain open challenges. Current recommendation tools rely on human intervention to determine the correctness of the recommendation (*EC-9*). Tools such as MAPO [199] attempted to automate API example gathering, but no tool currently fully solved this challenge. Work remains to extract code examples relevant to user queries, and to determine whether multiple examples are similar.

Unsolved Challenges: **Issue:** Currently, API property inference techniques do not appear to use uniform benchmarks to test their performance. The results of these techniques are therefore at the mercy of the dataset and evaluation methodologies chosen by their authors which prevents comparisons between techniques. **Proposition:** Future research should seek to use a standard evaluation such as the one provided by Robillard et al. [145] to improve the ease of comparison between various approaches (*UC-1*). **Issue:** Current solutions are context sensitive, yet extracting code context remains a challenging problem [87]. **Proposition:** Incorporating domain specific information into tools could help remedy this problem [62]. Yet, it is unclear how to best support the context sensitive nature of API migration tools, how these approaches would perform on different datasets (*UC-2*), or how their usage might affect API evolution feedback loops.

Issue: We posit that although there are some studies that attempt to generate theories about software APIs [65, 96, 134, 202], most tools and studies appear to be either dependent on, or linked to, factors such as API ecosystems and programming languages of the API (*UC-11*). Few studies attempt to determine whether the severity of various API evolution problems such as API migration and API recommendation are present across all programming languages (*UC-12*). **Proposition:** Systematic studies to determine the impact of API migration and the helpfulness of API recommendation systems are required to understand whether such aid is universally required or language dependent.

Answers of RQ3: What are the current and future challenges related to API evolution? Table 7 [6] *summarises and labels existing challenges (EC-1 through EC-17) and unsolved challenges (UC-1 through UC-15) identified during this systematic literature review. It shows that existing and unsolved challenges concern new tools and techniques and empirical studies first. We also consider unsolved challenges with datasets. They are concerned first and foremost with API migration, including towards Web APIs, and the evaluation/validation of API tools and their results.*

7 THREATS TO VALIDITY

Construct validity. We do not claim that "API Evolution" presents a perfect search phrase. Different search sentences, and more search terms, could yield more results. However, to mitigate this threat, we include a large number of studies to accurately represent the field. Our taxonomy was produced in a mostly ad-hoc manner and may present some subjectivity bias [170]. We attempt to mitigate this threat by using classifications that can be found in existing papers, synonyms for that existing terminology, and the opinion of three authors.

External validity. While we concede that it is unlikely that we managed to find and present all of the papers linked to the topic of API evolution in this study, we believe that the sample of publications chosen for this study is representative of the state of the art in the field of API evolution. We are confident that the majority of published works in the field of API evolution are present in this study and that the trends and findings in this work are the state-of-the-art. We attempt to mitigate this threat by using six different publication search engines and by using forward and backward snowballing to obtain papers missed by out search.

Internal validity. The choice and categorization of the papers presented in this paper could present some biases on the part of the authors of this paper. We attempted to mitigate these biases by relying on the API experience of all three authors and by having all three authors agree on the selection procedure before papers were selected. Furthermore, the categories used to classify the papers were also agreed upon by all authors. Finally, although the majority of the selection and classification of the papers was done by a single author, these procedures were verified using a test-retest reliability to ensure that the results were internally consistent. Results showed excellent reliability.

[6] It shows the main references presenting existing challenges. Emerging unsolved challenges are indirectly referenced because they are recently emerging and have not yet been thoroughly discussed and addressed in the literature.

8 CONCLUSION

In this paper, we presented a systematic survey of the literature on API evolution between 1994 and 2020 (27 years). We uncovered the publication trends as well as questions and goals related to API evolution common in the literature. We answered three research questions: *RQ1: How has the field of API evolution research evolved?*, *RQ2: What is the current state-of-the-art in API evolution research?*, and *RQ3: What are the current and future challenges related to API evolution?*

We observed that there are five API evolution research goals, in Section 4.1: *new tools and techniques, empirical studies, tools and technique proposals, surveys, and datasets.* We summarised the various methods and popular subject APIs used to evaluate API evolution research. In Section 4.2, we observed a variety of evaluation metrics, with precision, recall, f1-score, and AUC being the most common. **We recommend that API evolution researchers develop/use more common benchmarks and systematic evaluation methodologies [145] to allow thorough comparisons against and systematic improvements to the state of the art.**

We collected information on the APIs used to perform evaluation in the literature and reported in Section 4.3 that the Java programming language is the language of the studied API in 70.4% of the analysed papers, with the Java API, the Java Android API, some toy systems in Java, the Eclipse platform, JHotDraw, and Log4J used in more than half the papers. **While we do recommend that API evolution research uses common benchmarks and, therefore, similar evaluation subjects, we also recommend considering different programming languages than Java to improve generalisability and to identify underlying common/different factors.**

We studied the tools and techniques proposed in the literature and observed that they mostly seek (1) to improve API usage, (2) to provide API recommendation, (3) to help with API migration, (4) to reduce API misuse, and (5) to create better API documentation and examples. **We recommend incorporating domain-specific information into tools [109], creating tools for Web APIs [180] and others that help API developers improve their APIs. We recommend exploring machine learning approaches to help with API evolution challenges, a currently emerging area of interest [25, 124]. We also recommend generalising API tools to programming languages other than Java.**

We reviewed works presenting empirical studies on API evolution and concluded that they focus mostly on API usability and API maintainability. Studies on API usability focus on breaking changes, integration problems, API usages, standards, misuses, and documentation. Studies on API maintainability concern change velocity and change impact, deprecation, and reuse patterns. **We recommend studies to understand the coupling between APIs and programming languages, to determine the impact of API migration and recommendations, and to compare the evolution of APIs.**

We reported that tools and technique proposals discuss differentiating between Web and library APIs, automated documentation, and automated traceability between APIs and other software artifacts while surveys highlight past, current, and future challenges. We also reported that datasets are available with Linux system calls, API misuses, and API invocations. While Web and library APIs are different and have different levels of difficulties due to the control available to API developers, API evolution research is on-going, which warrants a continuation of efforts related to survey papers and an increase in the number of datasets available for API evolution research. **We strongly encourage researchers whose work is related to API evolution to make their benchmarks and datasets openly available and to augment existing datasets when appropriate.**

Although we found that *continuing change*, *increasing complexity*, *conservation of familiarity*, *continuing growth*, and *declining quality* are all worthy challenges to API evolution, the next hurdle will be leveraging and mastering the *feedback systems* involved in API evolution [90]. Thus, we hope that this paper can act as a reference for existing work within the scope of API evolution, as well as present challenges to guide the future of API evolution research.

REFERENCES

[1] Alberto Abelló Gamazo, Claudia Martinez, Carles Farré, Cristina Gómez, Marc Oriol, and Oscar Romero. 2017. A Data-driven approach to improve the process of data-intensive API creation and evolution. In *CAiSE-Forum-DC 2017: Proceedings of the Forum and Doctoral Consortium Papers Presented at the 29th International Conference on Advanced Information Systems Engineering*. CAiSE, Essen, Germany, 1–8.

[2] Ziyad Alsaeed and Michal Young. 2018. Extending existing inference tools to mine dynamic APIs. In *Proceedings of the 2nd International Workshop on API Usage and Evolution - WAPI '18*. ACM Press, New York, New York, USA, 23–26.

[3] Sven Amann, Sarah Nadi, Hoan A. Nguyen, Tien N Nguyen, and Mira Mezini. 2016. MUBench A Benchmark for API-Misuse Detectors. In *Proceedings of the 13th International Workshop on Mining Software Repositories - MSR '16*. ACM Press, New York, New York, USA, 464–467.

[4] Sven Amann, Hoan Anh Nguyen, Sarah Nadi, Tien N Nguyen, and Mira Mezini. 2019. Investigating Next Steps in Static API-Misuse Detection. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR) (MSR '19)*. IEEE, Piscataway, NJ, USA, 265–275.

[5] Sven Amann, Hoan Anh Nguyen, Sarah Nadi, Tien N. Nguyen, and Mira Mezini. 2019. A Systematic Evaluation of Static API-Misuse Detectors. *IEEE Trans. Software Eng.* 45, 12 (2019), 1170–1188.

[6] Taweesup Apiwattanapong, Alessandro Orso, and Mary Jean Harrold. 2007. JDiff: A differencing technique and tool for object-oriented programs. *Automated Software Engineering* 14, 1 (mar 2007), 3–36.

[7] Muhammad Asaduzzaman, Chanchal K. Roy, Samiul Monir, and Kevin A. Schneider. 2015. Exploring API method parameter recommendations. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, USA, 271–280.

[8] Vaggelis Atlidakis, Patrice Godefroid, and Marina Polishchuk. 2019. RESTler: Stateful REST API Fuzzing. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE) (ICSE '19)*. IEEE, Piscataway, NJ, USA, 748–758.

[9] Joop Aué, Maurício Aniche, Maikel Lobbezoo, and Arie van Deursen. 2018. An exploratory study on faults in web API integration in a large-scale payment company. In *Proceedings of the 40th International Conference on Software Engineering Software Engineering in Practice - ICSE-SEIP '18*. ACM Press, New York, New York, USA, 13–22.

[10] Shams Azad, Peter C. Rigby, and Latifa Guerrouj. 2017. Generating API Call Rules from Version History and Stack Overflow Posts. *ACM Transactions on Software Engineering and Methodology* 25, 4 (jan 2017), 1–22.

[11] Mojtaba Bagherzadeh, Nafiseh Kahani, Cor-Paul Bezemer, Ahmed E. Hassan, Juergen Dingel, and James R. Cordy. 2018. Analyzing a decade of Linux system calls. *Empirical Software Engineering* 23, 3 (jun 2018), 1519–1551.

[12] Mehdi Bahrami, Junhee Park, Lei Liu, and Wei-Peng Chen. 2018. API Learning. In *Companion of the The Web Conference 2018 on The Web Conference 2018 - WWW '18*. ACM Press, New York, New York, USA, 151–154.

[13] Ittai Balaban, Frank Tip, and Robert Fuhrer. 2005. Refactoring support for class library migration. *ACM SIGPLAN Notices* 40, 10 (oct 2005), 265.

[14] Thiago Tonelli Bartolomei, Krzysztof Czarnecki, Ralf Lämmel, and Tijs van der Storm. 2010. Study of an API Migration for Two XML APIs. In *Software Language Engineering. SLE 2009*. Springer, Berlin, Heidelberg, 42–61.

[15] Gabriele Bavota, Mario Linares-Vasquez, Carlos Eduardo Bernal-Cardenas, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. 2015. The Impact of API Change- and Fault-Proneness on the User Ratings of Android Apps. *IEEE Transactions on Software Engineering* 41, 4 (apr 2015), 384–407.

[16] Boualem Benatallah and Fabio Casati. 2018. Panel on Cognitive Service Engineering. In *Companion of the The Web Conference 2018 on The Web Conference 2018 - WWW '18*. ACM Press, New York, New York, USA, 883–883.

[17] Boualem Benatallah, Fabio Casati, Daniela Grigori, Hamid R Motahari Nezhad, and Farouk Toumani. 2005. Developing Adapters for Web Services Integration. In *Advanced Information Systems Engineering*, Oscar Pastor and João e Cunha (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 415–429.

[18] David Bermbach and Erik Wittern. 2016. Benchmarking Web API Quality. In *Web Engineering. ICWE 2016*. Springer, Berlin, Heidelberg, 188–206.

[19] Salah Bouktif, Houari Sahraoui, and Faheem Ahmed. 2014. Predicting Stability of Open-Source Software Systems Using Combination of Bayesian Classifiers. *ACM Transactions on Management Information Systems* 5, 1 (apr 2014), 1–26.

[20] Aline Brito, Laerte Xavier, Andre Hora, and Marco Tulio Valente. 2018. APIDiff: Detecting API breaking changes. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Vol. 2018-March. IEEE, USA, 507–511.

[21] Aline Brito, Laerte Xavier, Andre Hora, and Marco Tulio Valente. 2018. Why and how Java developers break APIs. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, Vol. 2018-March.

IEEE, USA, 255–265. arXiv:1801.05198

[22] Gleison Brito, Andre Hora, Marco Tulio Valente, and Romain Robbes. 2016. Do Developers Deprecate APIs with Replacement Messages? A Large-Scale Analysis on Java Systems. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. IEEE, USA, 360–369.

[23] Gleison Brito, Andre Hora, Marco Tulio Valente, and Romain Robbes. 2018. On the use of replacement messages in API deprecation: An empirical study. *Journal of Systems and Software* 137 (mar 2018), 306–321.

[24] Nghi D. Q. Bui, Yijun Yu, and Lingxiao Jiang. 2019. SAR: learning cross-language API mappings with little knowledge. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2019*. ACM Press, New York, New York, USA, 796–806. arXiv:1906.03835

[25] Nghi D Q Bui, Yijun Yu, and Lingxiao Jiang. 2019. SAR: learning cross-language API mappings with little knowledge. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2019 (ESEC/FSE 2019)*. ACM Press, New York, New York, USA, 796–806.

[26] Raymond P. L. Buse and Westley Weimer. 2012. Synthesizing API usage examples. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, USA, 782–792.

[27] John Businge, Alexander Serebrenik, and M. van den Brand. 2013. Analyzing the Eclipse API Usage: Putting the Developer in the Loop. In *17th European Conference on Software Maintenance and Reengineering*. IEEE, USA, 37–46.

[28] Paolo Calciati, Konstantin Kuznetsov, Xue Bai, and Alessandra Gorla. 2018. What did really change with the new release of the app?. In *Proceedings of the 15th International Conference on Mining Software Repositories - MSR '18*. ACM Press, New York, New York, USA, 142–152.

[29] Joao Campinhos, Joao Costa Seco, and Jacome Cunha. 2017. Type-Safe Evolution of Web Services. In *2017 IEEE/ACM 2nd International Workshop on Variability and Complexity in Software Design (VACE)*. IEEE, USA, 20–26.

[30] Wing-Kwan Chan, Hong Cheng, and David Lo. 2012. Searching Connected API Subgraph via Text Phrases. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering (FSE '12)*. Association for Computing Machinery, New York, NY, USA, Article 10, 11 pages.

[31] Bradley E Cossette and Robert J Walker. 2012. Seeking the ground truth. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE '12 (FSE '12)*. ACM Press, USA, 1.

[32] Ira W. Cotton and Frank S. Greatorex, Jr. 1968. Data Structures and Techniques for Remote Computer Graphics. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I (AFIPS '68 (Fall, part I))*. ACM, New York, NY, USA, Article 1476661, 12 pages.

[33] Barthélémy Dagenais and Harold Ossher. 2008. Automatically locating framework extension examples. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering - SIGSOFT '08/FSE-16 (SIGSOFT '08/FSE-16)*. ACM Press, New York, New York, USA, 203.

[34] Barthelemy Dagenais and Martin P. Robillard. 2009. SemDiff: Analysis and recommendation support for API evolution. In *2009 IEEE 31st International Conference on Software Engineering*. IEEE, USA, 599–602.

[35] Barthélémy Dagenais and Martin P Robillard. 2010. Creating and evolving developer documentation. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering - FSE '10 (FSE '10)*. ACM Press, New York, New York, USA, 127.

[36] Marco D'Ambros, Michele Lanza, Mircea Lungu, and Romain Robbes. 2009. Promises and perils of porting software visualization tools to the web. In *2009 11th IEEE International Symposium on Web Systems Evolution*. IEEE, USA, 109–118.

[37] Fernando López de la Mora and Sarah Nadi. 2018. Which Library Should I Use? A Metric-Based Comparison of Software Libraries. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER '18)*. Association for Computing Machinery, New York, NY, USA, 37–40.

[38] Uri Dekel and James D. Herbsleb. 2009. Improving API Documentation Usability with Knowledge Pushing. In *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, USA, 320–330.

[39] Jens Dietrich, Kamil Jezek, and Premek Brada. 2014. Broken promises: An empirical study into evolution problems in Java programs caused by library upgrades, In 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE). *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)* 1, 64–73.

[40] Danny Dig. 2005. Using refactorings to automatically update component-based applications. In *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications - OOPSLA '05*. ACM Press, New York, New York, USA, 234.

[41] Danny Dig and Ralph Johnson. 2005. The role of refactorings in API evolution. *21st IEEE International Conference on Software Maintenance (ICSM'05)* 2005 (2005), 389–398.

[42] Danny Dig and Ralph Johnson. 2006. Automated upgrading of component-based applications. In *Companion to the 21st ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications - OOPSLA '06*, Vol. 2006. ACM Press, New York, New York, USA, 675.

[43] Danny Dig and Ralph Johnson. 2006. How do APIs evolve? A story of refactoring. *Journal of Software Maintenance and Evolution: Research and Practice* 18, 2 (mar 2006), 83–107.

[44] Danny Dig, Kashif Manzoor, Ralph Johnson, and Tien N. Nguyen. 2007. Refactoring-Aware Configuration Management for Object-Oriented Programs. In *Proceedings of the 29th International Conference on Software Engineering (ICSE '07)*. IEEE Computer Society, USA, 427–436.

[45] Ekwa Duala-Ekoko and Martin P Robillard. 2012. Asking and answering questions about unfamiliar APIs: An exploratory study. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, USA, 266–276.

[46] Anna Maria Eilertsen and Anya Helene Bagge. 2018. Exploring API. In *Proceedings of the 2nd International Workshop on API Usage and Evolution - WAPI '18*. ACM Press, New York, New York, USA, 10–13.

[47] Daniel S. Eisenberg, Jeffrey Stylos, and Brad A. Myers. 2010. Apatite. In *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*. ACM Press, New York, New York, USA, 1331.

[48] Brian Ellis, Jeffrey Stylos, and Brad Myers. 2007. The Factory Pattern in API Design: A Usability Evaluation. In *Proceedings of the 29th International Conference on Software Engineering*. IEEE Computer Society, USA, 302–312.

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

[50] M.D. Ernst, Jake Cockrell, W.G. Griswold, and David Notkin. 2001. Dynamically discovering likely program invariants to support program evolution. *IEEE Transactions on Software Engineering* 27, 2 (2001), 99–123.

[51] Tiago Espinha, Andy Zaidman, and Hans-Gerhard Gross. 2014. Web API growing pains: Stories from client developers and their code. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*. IEEE, USA, 84–93.

[52] Tiago Espinha, Andy Zaidman, and Hans-Gerhard Gross. 2015. Web API Fragility: How Robust is Your Mobile Application?. In *Proceedings of the Second ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft '15)*. IEEE Press, USA, 12–21.

[53] Tiago Espinha, Andy Zaidman, and Hans-Gerhard Gross. 2015. Web API growing pains: Loosely coupled yet strongly tied. *Journal of Systems and Software* 100 (feb 2015), 27–43.

[54] Darius Foo, Hendy Chua, Jason Yeo, Ming Yi Ang, and Asankhaya Sharma. 2018. Efficient static checking of library updates. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2018*. ACM Press, New York, New York, USA, 791–796.

[55] Davide Fucci, Alireza Mollaalizadehbahnemiri, and Walid Maalej. 2019. On using machine learning to identify knowledge in API reference documentation. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2019*. ACM Press, New York, New York, USA, 109–119. arXiv:1907.09807

[56] Jun Gao, Pingfan Kong, Li Li, Tegawende F. Bissyande, and Jacques Klein. 2019. Negative Results on Mining Crypto-API Usage Rules in Android Apps. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*. IEEE, USA, 388–398.

[57] Simos Gerasimou, Maria Kechagia, Dimitris Kolovos, Richard Paige, and Georgios Gousios. 2018. On software modernisation due to library obsolescence. In *Proceedings of the 2nd International Workshop on API Usage and Evolution - WAPI '18*. ACM Press, New York, New York, USA, 6–9.

[58] Elena L Glassman, Tianyi Zhang, Björn Hartmann, and Miryung Kim. 2018. Visualizing API Usage Examples at Scale. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems - CHI '18*. ACM Press, USA, 1–12.

[59] M.W. Godfrey and Lijie Zou. 2005. Using origin analysis to detect merging and splitting of source code entities. *IEEE Transactions on Software Engineering* 31, 2 (feb 2005), 166–181.

[60] William Granli, John Burchell, Imed Hammouda, and Eric Knauss. 2015. The driving forces of API evolution. In *Proceedings of the 14th International Workshop on Principles of Software Evolution - IWPSE 2015*, Vol. 30-Aug-201. ACM Press, New York, New York, USA, 28–37.

[61] Giovanni Grano, Andrea Di Sorbo, Francesco Mercaldo, Corrado A. Visaggio, Gerardo Canfora, and Sebastiano Panichella. 2017. Android apps and user feedback: a dataset for software evolution and quality improvement. In *Proceedings of the 2nd ACM SIGSOFT International Workshop on App Market Analytics - WAMA 2017*. ACM Press, New York, New York, USA, 8–11.

[62] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim. 2017. DeepAM: Migrate APIs with Multi-Modal Sequence to Sequence Learning. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*. AAAI Press, 3675–3681.

[63] Dongjie He, Lian Li, Lei Wang, Hengjie Zheng, Guangwei Li, and Jingling Xue. 2018. Understanding and detecting evolution-induced compatibility issues in Android apps. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering - ASE 2018*. ACM Press, New York, New York, USA, 167–177.

[64] J. Henkel and A. Diwan. 2005. Catchup! capturing and replaying refactorings to support API evolution. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.* IEEE, USA, 274–283.
[65] Robert Heumüller, Sebastian Nielebock, and Frank Ortmeier. 2018. Who plays with whom? ... and how? mining API interaction patterns from source code. In *Proceedings of the 7th International Workshop on Software Mining - SoftwareMining 2018.* ACM Press, New York, New York, USA, 8–11.
[66] Reid Holmes and Robert J. Walker. 2008. A newbie's guide to eclipse APIs. In *Proceedings of the 2008 international workshop on Mining software repositories - MSR '08.* ACM Press, New York, New York, USA, 149.
[67] André Hora, Romain Robbes, Marco Tulio Valente, Nicolas Anquetil, Anne Etien, and Stéphane Ducasse. 2018. How do developers react to API evolution? A large-scale empirical study. *Software Quality Journal* 26, 1 (mar 2018), 161–191.
[68] Andre Hora and Marco Tulio Valente. 2015. Apiwave: Keeping track of API popularity and migration. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, USA, 321–323.
[69] André Hora, Marco Tulio Valente, Romain Robbes, and Nicolas Anquetil. 2016. When should internal interfaces be promoted to public?. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016*, Vol. 13-18-Nove. ACM Press, New York, New York, USA, 278–289.
[70] Daqing Hou and Xiaojia Yao. 2011. Exploring the Intent behind API Evolution: A Case Study. In *2011 18th Working Conference on Reverse Engineering.* IEEE, USA, 131–140.
[71] Qiao Huang, Xin Xia, Zhenchang Xing, David Lo, and Xinyu Wang. 2018. API method recommendation without worrying about the task-API knowledge gap. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering - ASE 2018.* ACM Press, New York, New York, USA, 293–304.
[72] R. Huang, W. Sun, Y. Xu, H. Chen, D. Towey, and X. Xia. 2019. A Survey on Adaptive Random Testing. *IEEE Transactions on Software Engineering* 1, 1 (2019), 1–1.
[73] Shiyou Huang, Jianmei Guo, Sanhong Li, Xiang Li, Yumin Qi, Kingsum Chow, and Jeff Huang. 2019. SafeCheck: Safety Enhancement of Java Unsafe API. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, Vol. 2019-May. IEEE, USA, 889–899.
[74] Andrew J. Ko, B.A. Myers, and H.H. Aung. 2004. Six Learning Barriers in End-User Programming Systems. In *2004 IEEE Symposium on Visual Languages - Human Centric Computing.* IEEE, USA, 199–206.
[75] Z. M. Jiang and A. E. Hassan. 2015. A Survey on Load Testing of Large-Scale Software Systems. *IEEE Transactions on Software Engineering* 41, 11 (2015), 1091–1118.
[76] Johnson. 1994. Substring matching for clone detection and change tracking. In *Proceedings International Conference on Software Maintenance ICSM-94.* IEEE Comput. Soc. Press, USA, 120–126.
[77] Sukrit Kalra, Ayush Goel, Dhriti Khanna, Mohan Dhawan, Subodh Sharma, and Rahul Purandare. 2016. POLLUX: safely upgrading dependent application libraries. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016*, Vol. 13-18-Nove. ACM Press, New York, New York, USA, 290–300.
[78] R H Katz. 2000. The post-PC era. In *Proceedings of the 4th international workshop on Discrete algorithms and methods for mobile computing and communications - DIALM '00.* ACM Press, New York, New York, USA, 64.
[79] David Kawrykow and Martin P. Robillard. 2009. Improving API Usage through Automatic Detection of Redundant Code. In *2009 IEEE/ACM International Conference on Automated Software Engineering.* IEEE, USA, 111–122.
[80] Miryung Kim, Dongxiang Cai, and Sunghun Kim. 2011. An Empirical Investigation into the Role of API-Level Refactorings during Software Evolution. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE '11).* Association for Computing Machinery, New York, NY, USA, 151–160.
[81] B. Kitchenham and S Charters. Keele University, Keele, U.K., Tech. Rep. EBSE-2007-01, 2007. Guidelines for performing Systematic Literature Reviews in Software Engineering. (Keele University, Keele, U.K., Tech. Rep. EBSE-2007-01, 2007).
[82] Deokyoon Ko, Kyeongwook Ma, Sooyong Park, Suntae Kim, Dongsun Kim, and Yves Le Traon. 2014. API Document Quality for Resolving Deprecated APIs. In *21st Asia-Pacific Software Engineering Conference*, Vol. 2. IEEE, USA, 27–30.
[83] Raula Gaikovina Kula, Ali Ouni, Daniel M. German, and Katsuro Inoue. 2018. An empirical study on the impact of refactoring activities on evolving client-used APIs. *Information and Software Technology* 93, July 2016 (jan 2018), 186–199. arXiv:1709.09474
[84] Hobum Kwon, Juwon Ahn, Sunggyu Choi, Jakub Siewierski, Piotr Kosko, and Piotr Szydelko. 2018. An Experience Report of the API Evolution and Maintenance for Software Platforms. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, USA, 587–590.
[85] Maxime Lamothe and Weiyi Shang. 2018. Exploring the use of automated API migrating techniques in practice. *Proceedings of the 15th International Conference on Mining Software Repositories - MSR '18* 1 (2018), 503–514.
[86] Maxime Lamothe and Weiyi Shang. 2020. When APIs Are Intentionally Bypassed: An Exploratory Study of API Workarounds. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20).* Association for Computing Machinery, New York, NY, USA, 912–924.
[87] M. Lamothe, W. Shang, and T. P. Chen. 2020. A3: Assisting Android API Migrations Using Code Examples. *IEEE Transactions on Software Engineering* (2020).

[88] Craig Larman. 2001. Protected variation: the importance of being closed. *IEEE Software* 18, 3 (2001), 89–91.
[89] M. M. Lehman. 1980. Programs, life cycles, and laws of software evolution. *Proc. IEEE* 68, 9 (1980), 1060–1076.
[90] M. M. Lehman. 1996. Laws of Software Evolution Revisited. In *Proceedings of the 5th European Workshop on Software Process Technology (EWSPT '96)*. Springer-Verlag, Berlin, Heidelberg, 108–124.
[91] Grace A. Lewis and Dennis B. Smith. 2008. Service-Oriented Architecture and its implications for software maintenance and evolution. In *2008 Frontiers of Software Maintenance*. IEEE, USA, 1–10.
[92] Jing Li, Aixin Sun, Zhenchang Xing, and Lei Han. 2018. API Caveat Explorer – Surfacing Negative Usages from Practice. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval - SIGIR '18*. ACM Press, New York, New York, USA, 1293–1296.
[93] Jun Li, Yingfei Xiong, Xuanzhe Liu, and Lu Zhang. 2013. How Does Web Service API Evolution Affect Clients? *2013 IEEE 20th International Conference on Web Services* 1 (jun 2013), 300–307.
[94] Li Li and Wu Chou. 2015. Designing Large Scale REST APIs Based on REST Chart. In *2015 IEEE International Conference on Web Services (ICWS '15)*. IEEE, Washington, DC, USA, 631–638.
[95] Li Li, Jun Gao, Tegawendé F Bissyandé, Lei Ma, Xin Xia, and Jacques Klein. 2018. Characterising deprecated Android APIs. In *Proceedings of the 15th International Conference on Mining Software Repositories - MSR '18*. ACM Press, New York, New York, USA, 254–264.
[96] Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Rocco Oliveto, Massimiliano Di Penta, and Denys Poshyvanyk. 2014. Mining energy-greedy API usage patterns in Android apps: an empirical study. *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014* 1 (2014), 2–11.
[97] Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. 2014. How do API changes trigger stack overflow discussions? a study on the Android SDK. In *Proceedings of the 22nd International Conference on Program Comprehension - ICPC 2014*. ACM Press, New York, New York, USA, 83–94.
[98] Mingwei Liu, Xin Peng, Andrian Marcus, Zhenchang Xing, Wenkai Xie, Shuangshuang Xing, and Yang Liu. 2019. Generating query-specific class API summaries. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2019 (ESEC/FSE 2019)*. ACM Press, New York, New York, USA, 120–130.
[99] Xiaoyu Liu, LiGuo Huang, and Vincent Ng. 2018. Effective API recommendation without historical software repositories. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering - ASE 2018*. ACM Press, New York, New York, USA, 282–292.
[100] Maria Maleshkova, Carlos Pedrinaci, and John Domingue. 2010. Investigating Web APIs on the World Wide Web. In *2010 Eighth IEEE European Conference on Web Services*. IEEE, USA, 107–114.
[101] David Mandelin, Lin Xu, Rastislav Bodík, and Doug Kimelman. 2005. Jungloid Mining: Helping to Navigate the API Jungle. In *Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '05)*. Association for Computing Machinery, New York, NY, USA, 48–61.
[102] Konstantinos Manikas. 2016. Revisiting software ecosystems Research: A longitudinal literature study. *Journal of Systems and Software* 117 (jul 2016), 84–103.
[103] Anderson S. Matos, Joao B. Ferreira Filho, and Lincoln S. Rocha. 2019. Splitting APIs: An Exploratory Study of Software Unbundling. In *IEEE/ACM 16th International Conference on Mining Software Repositories*. IEEE, USA, 360–370.
[104] Tyler McDonnell, Baishakhi Ray, and Miryung Kim. 2013. An Empirical Study of API Stability and Adoption in the Android Ecosystem. In *2013 IEEE International Conference on Software Maintenance*. IEEE, USA, 70–79.
[105] Collin McMillan, Denys Poshyvanyk, and Mark Grechanik. 2010. Recommending Source Code Examples via API Call Usages and Documentation. In *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering (RSSE '10)*. Association for Computing Machinery, New York, NY, USA, 21–25.
[106] Na Meng, Miryung Kim, and Kathryn S. McKinley. 2013. LASE: Locating and Applying Systematic Edits by Learning from Examples. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, USA, 502–511.
[107] Sichen Meng, Xiaoyin Wang, Lu Zhang, and Hong Mei. 2012. A history-based matching approach to identification of framework evolution. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, USA, 353–363.
[108] Kim Mens and Angela Lozano. 2014. Source Code-Based Recommendation Systems. In *Recommendation Systems in Software Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg, 93–130.
[109] Tom Mens. 2002. A state-of-the-art survey on software merging. *IEEE Transactions on Software Engineering* 28, 5 (may 2002), 449–462.
[110] A. Michail. 2003. Data mining library reuse patterns in user-selected applications. In *14th IEEE International Conference on Automated Software Engineering*. IEEE Comput. Soc, USA, 24–33.
[111] Microsoft. 2019. interface c# reference 2019. *Docs.microsoft.com* (2019).
[112] Yana Momchilova Mileva, Valentin Dallmeier, and Andreas Zeller. 2010. Mining API Popularity. In *Testing – Practice and Research Techniques. TAIC PART 2010*. Springer, Berlin, Heidelberg, 173–180.

[113] Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Andrian Marcus. 2015. How Can I Use This Method?. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, USA, 880–890.
[114] Eduardo Mosqueira-Rey, David Alonso-Ríos, Vicente Moret-Bonillo, Isaac Fernández-Varela, and Diego Álvarez-Estévez. 2018. A systematic approach to API usability: Taxonomy-derived criteria and a case study. *Information and Software Technology* 97, December 2017 (may 2018), 46–63.
[115] Shaikh Mostafa, Rodney Rodriguez, and Xiaoyin Wang. 2017. Experience paper: a study on behavioral backward incompatibilities of Java software libraries. In *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis - ISSTA 2017*. ACM Press, New York, New York, USA, 215–225.
[116] Emerson Murphy-Hill, Caitlin Sadowski, Andrew Head, John Daughtry, Andrew Macvean, Ciera Jaspan, and Collin Winter. 2018. Discovering API usability problems at scale. In *Proceedings of the 2nd International Workshop on API Usage and Evolution - WAPI '18*. ACM Press, New York, New York, USA, 14–17.
[117] Varvana Myllärniemi, Sari Kujala, Mikko Raatikainen, and Piia Sevonn. 2018. Development as a journey: factors supporting the adoption and use of software frameworks. *Journal of Software Engineering Research and Development* 6, 1 (dec 2018), 6.
[118] Anh Tuan Nguyen, Michael Hilton, Mihai Codoban, Hoan Anh Nguyen, Lily Mast, Eli Rademacher, Tien N. Nguyen, and Danny Dig. 2016. API code recommendation using statistical learning from fine-grained changes. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM Press, USA, 511–522.
[119] Anh Tuan Nguyen, Peter C. Rigby, Thanh Van Nguyen, Mark Karanfil, and Tien N. Nguyen. 2017. Statistical Translation of English Texts to API Code Templates. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, USA, 331–333.
[120] Hoan Anh Nguyen, Tien N. Nguyen, Hridesh Rajan, and Robert Dyer. 2018. Towards combining usage mining and implementation analysis to infer API preconditions. In *Proceedings of the 1st ACM SIGSOFT International Workshop on Automated Specification Inference - WASPI 2018*. ACM Press, New York, New York, USA, 15–16.
[121] Phuong T Nguyen, Juri Di Rocco, Davide Di Ruscio, Lina Ochoa, Thomas Degueule, and Massimiliano Di Penta. 2019. FOCUS: A Recommender System for Mining API Function Calls and Usage Patterns. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, USA, 1050–1060.
[122] Thanh Nguyen, Peter C. Rigby, Anh Tuan Nguyen, Mark Karanfil, and Tien N. Nguyen. 2016. T2API: synthesizing API code usage templates from English texts with statistical translation. In *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM Press, USA, 1013–1017.
[123] Thanh Nguyen, Ngoc Tran, Hung Phan, Trong Nguyen, Linh Truong, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N. Nguyen. 2018. Complementing global and local contexts in representing API descriptions to improve API retrieval tasks. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE*. ACM Press, USA, 551–562.
[124] Trong Duc Nguyen, Anh Tuan Nguyen, Hung Dang Phan, and Tien N. Nguyen. 2017. Exploring API Embedding for API Usages and Applications. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. IEEE, USA, 438–449.
[125] Thanh V. Nguyen and Tien N. Nguyen. 2018. Inferring API elements relevant to an english query. In *Proceedings of the 40th International Conference on Software Engineering Companion Proceeedings - ICSE '18*, Vol. Part F1373. ACM Press, New York, New York, USA, 167–168.
[126] Oracle. 2019. what is an interface? (Java™ learning the java language object-oriented programming concepts. (2019).
[127] D. L. Parnas. 1972. On the Criteria to Be Used in Decomposing Systems into Modules. *Commun. ACM* 15, 12 (Dec. 1972), 1053–1058.
[128] Chris Parnin and Christoph Treude. 2011. Measuring API documentation on the web. In *Proceeding of the 2nd international workshop on Web 2.0 for software engineering - Web2SE '11*. ACM Press, New York, New York, USA, 25–30.
[129] Chris Parnin, Christoph Treude, Lars Grammel, and Margaret-Anne Storey. 2012. Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow. *Georgia Tech Technical Report* 1 (2012), 1–11.
[130] K. Petersen, S. Vakkalanka, and L. Kuzniarz. 2015. Guidelines for Conducting Systematic Mapping Studies in Software Engineering: An Update. *Inf. Softw. Technol.* 64, C (Aug. 2015), 1–18.
[131] Pujan Petersen, Stefan Hanenberg, and Romain Robbes. 2014. An empirical comparison of static and dynamic type systems on API usage in the presence of an IDE: Java vs. groovy with eclipse. In *Proceedings of the 22nd International Conference on Program Comprehension - ICPC 2014*. ACM Press, New York, New York, USA, 212–222.
[132] Hung Phan, Hoan Anh Nguyen, Ngoc M Tran, Linh H Truong, Anh Tuan Nguyen, and Tien N. Nguyen. 2018. Statistical learning of API fully qualified names in code snippets of online forums. In *Proceedings of the 40th International Conference on Software Engineering - ICSE '18*, Vol. 11. ACM Press, New York, New York, USA, 632–642.
[133] H. D. Phan, A. T. Nguyen, T. D. Nguyen, and T. N. Nguyen. 2017. Statistical Migration of API Usages. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*. 47–50.

[134] Marco Piccioni, Carlo A. Furia, and Bertrand Meyer. 2013. An Empirical Study of API Usability. In *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, USA, 5–14.

[135] David M Pletcher and Daqing Hou. 2009. BCC: Enhancing code completion for better API usability. *2009 IEEE International Conference on Software Maintenance* 1 (sep 2009), 393–394.

[136] Felipe Pontes, Rohit Gheyi, Sabrina Souto, Alessandro Garcia, and Márcio Ribeiro. 2019. Java reflection API: revealing the dark side of the mirror. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2019*. ACM Press, New York, New York, USA, 636–646.

[137] Ivan Porres and Irum Rauf. 2011. Modeling behavioral RESTful web service interfaces in UML. In *Proceedings of the 2011 ACM Symposium on Applied Computing - SAC '11 (SAC '11)*. ACM Press, New York, New York, USA, 1598.

[138] Michael Pradel, Ciera Jaspan, Jonathan Aldrich, and Thomas R. Gross. 2012. Statically checking API protocol conformance with mined multi-object specifications. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, USA, 925–935.

[139] Lutz Prechelt and D.J. Hutzel. 2003. The co-evolution of a hype and a software architecture: experience of component-producing large-scale EJB early adopters. In *25th International Conference on Software Engineering, 2003. Proceedings.*, Vol. 0. IEEE, USA, 553–556.

[140] Qi Xi, Tianyang Zhou, Qingxian Wang, and Yongjun Zeng. 2013. An API deobfuscation method combining dynamic and static techniques. *Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC)* 1 (dec 2013), 2133–2138.

[141] Mohammad Masudur Rahman, Chanchal K Roy, and David Lo. 2016. RACK: Automatic API Recommendation Using Crowdsourced Knowledge. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. IEEE, USA, 349–359.

[142] Anastasia Reinhardt, Tianyi Zhang, Mihir Mathur, and Miryung Kim. 2018. Augmenting stack overflow with API usage patterns mined from GitHub. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM Press, USA, 880–883.

[143] Romain Robbes and Mircea Lungu. 2011. A study of ripple effects in software ecosystems. In *Proceeding of the 33rd international conference on Software engineering - ICSE '11 (ICSE '11)*. ACM Press, New York, New York, USA, 904.

[144] Martin P Robillard. 2009. What Makes APIs Hard to Learn? Answers from Developers. *IEEE Software* 26 (2009), 27–34.

[145] Martin P Robillard, Eric Bodden, David Kawrykow, Mira Mezini, and Tristan Ratchford. 2013. Automated API Property Inference Techniques. *IEEE Transactions on Software Engineering* 39, 5 (may 2013), 613–637.

[146] Martin P Robillard and Yam B Chhetri. 2015. Recommending reference API documentation. *Empirical Software Engineering* 20, 6 (dec 2015), 1558–1586.

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

[148] Martin P. Robillard, Walid Maalej, Robert J. Walker, and Thomas Zimmermann. 2014. *Recommendation Systems in Software Engineering*. Springer Berlin Heidelberg, Berlin, Heidelberg. v–viii pages.

[149] Martin P Robillard, Andrian Marcus, Christoph Treude, Gabriele Bavota, Oscar Chaparro, Neil Ernst, Marco Aurelio Gerosa, Michael Godfrey, Michele Lanza, Mario Linares-Vasquez, Gail C. Murphy, Laura Moreno, David Shepherd, and Edmund Wong. 2017. On-demand Developer Documentation. *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)* 1 (sep 2017), 479–483.

[150] Thomas Ruhroth and Heike Wehrheim. 2009. Refinement-Preserving Co-evolution. In *Formal Methods and Software Engineering. ICFEM 2009*. Springer, Berlin, Heidelberg, 620–638.

[151] Chandan R Rupakheti and Daqing Hou. 2012. Evaluating forum discussions to inform the design of an API critic. In *2012 20th IEEE International Conference on Program Comprehension (ICPC)*. IEEE, USA, 53–62.

[152] Mohamed Aymen Saied, Houari Sahraoui, and Bruno Dufour. 2015. An observational study on API usage constraints and their documentation. *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)* 1 (mar 2015), 33–42.

[153] Pasquale Salza, Fabio Palomba, Dario Di Nucci, Cosmo D'Uva, Andrea De Lucia, and Filomena Ferrucci. 2018. Do developers update third-party libraries in mobile apps?. In *Proceedings of the 26th Conference on Program Comprehension - ICPC '18*. ACM Press, New York, New York, USA, 255–265.

[154] Anand Ashok Sawant, Maurício Aniche, Arie van Deursen, and Alberto Bacchelli. 2018. Understanding developers' needs on deprecation as a language feature. In *Proceedings of the 40th International Conference on Software Engineering - ICSE '18*, Vol. 11. ACM Press, New York, New York, USA, 561–571.

[155] Anand Ashok Sawant and Alberto Bacchelli. 2015. A Dataset for API Usage. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, Vol. 2015-Augus. IEEE, USA, 506–509.

[156] Anand Ashok Sawant and Alberto Bacchelli. 2017. fine-GRAPE: fine-grained APi usage extractor – an approach and dataset to investigate API usage. *Empirical Software Engineering* 22, 3 (jun 2017), 1348–1371.

[157] Anand Ashok Sawant, Romain Robbes, and Alberto Bacchelli. 2018. On the reaction to deprecation of clients of 4 + 1 popular Java APIs and the JDK. *Empirical Software Engineering* 23, 4 (aug 2018), 2158–2197.

[158] Simone Scalabrino, Gabriele Bavota, Mario Linares-Vasquez, Michele Lanza, and Rocco Oliveto. 2019. Data-Driven Solutions to Detect API Compatibility Issues in Android: An Empirical Study. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, Vol. 2019-May. IEEE, USA, 288–298.

[159] Thorsten Schäfer, Jan Jonas, and Mira Mezini. 2008. Mining framework usage changes from instantiation code. In *Proceedings of the 13th international conference on Software engineering (ICSE).* ACM Press, USA, 471.

[160] S.M. Sohan, Craig Anslow, and Frank Maurer. 2015. A Case Study of Web API Evolution. In *2015 IEEE World Congress on Services.* IEEE, USA, 245–252.

[161] S M Sohan, Craig Anslow, and Frank Maurer. 2015. SpyREST: Automated RESTful API Documentation Using an HTTP Proxy Server. In *30th IEEE/ACM International Conference on Automated Software Engineering.* IEEE, USA, 271–276.

[162] Brett Spell. 2015. *Pro Java 8 programming.* Apress, USA.

[163] Jeffrey Stylos and B.A. Myers. 2006. Mica: A Web-Search Tool for Finding API Components and Examples. In *Visual Languages and Human-Centric Computing (VL/HCC'06).* IEEE, USA, 195–202.

[164] Jingyi Su, Mohd Arafat, and Robert Dyer. 2018. Using consensus to automatically infer post-conditions. In *Proceedings of the 40th International Conference on Software Engineering Companion Proceeedings.* ACM Press, USA, 202–203.

[165] Siddharth Subramanian, Laura Inozemtseva, and Reid Holmes. 2014. Live API documentation. In *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014.* ACM Press, New York, New York, USA, 643–652.

[166] Philippe Suter and Erik Wittern. 2015. Inferring Web API Descriptions from Usage Data. In *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb).* IEEE, USA, 7–12.

[167] Christoph Treude and Maurício Aniche. 2018. Where does Google find API documentation? *Proceedings of the 2nd International Workshop on API Usage and Evolution - WAPI '18* 2 (2018), 19–22.

[168] Nikolaos Tsantalis, Matin Mansouri, Laleh M. Eshkevari, Davood Mazinanian, and Danny Dig. 2018. Accurate and efficient refactoring detection in commit history. In *Proceedings of the 40th International Conference on Software Engineering - ICSE '18.* ACM Press, New York, New York, USA, 483–494.

[169] Gias Uddin and Martin P. Robillard. 2015. How API Documentation Fails. *IEEE Software* 32, 4 (jul 2015), 68–75.

[170] Muhammad Usman, Ricardo Britto, Jürgen Börstler, and Emilia Mendes. 2017. Taxonomies in software engineering: A Systematic mapping study and a revised taxonomy development method. *Information and Software Technology* 85 (2017), 43–59. https://www.sciencedirect.com/science/article/pii/S0950584917300472

[171] Thanh Van Nguyen, Anh Tuan Nguyen, and Tien N. Nguyen. 2016. Characterizing API elements in software documentation with vector representation. In *Proceedings of the 38th International Conference on Software Engineering Companion - ICSE '16.* ACM Press, New York, New York, USA, 749–751.

[172] Pradeep K. Venkatesh, Shaohua Wang, Feng Zhang, Ying Zou, and Ahmed E. Hassan. 2016. What Do Client Developers Concern When Using Web APIs? An Empirical Study on Developer Forums and Stack Overflow. In *2016 IEEE International Conference on Web Services (ICWS).* IEEE, USA, 131–138.

[173] Gemma Vilagut. 2014. *Test-Retest Reliability.* Springer Netherlands, Dordrecht, 6622–6625. https://doi.org/10.1007/978-94-007-0753-5_3001

[174] Jue Wang, Yingnong Dang, Hongyu Zhang, Kai Chen, Tao Xie, and Dongmei Zhang. 2013. Mining succinct and high-coverage API usage patterns from source code. In *2013 10th Working Conference on Mining Software Repositories (MSR).* IEEE, USA, 319–328.

[175] Shaohua Wang, Nhathai Phan, Yan Wang, and Yong Zhao. 2019. Extracting API Tips from Developer Question and Answer Websites. In *IEEE/ACM 16th International Conference on Mining Software Repositories*, Vol. 1. IEEE, USA, 321–332.

[176] Lili Wei, Yepang Liu, and Shing-Chi Cheung. 2019. PIVOT: Learning API-Device Correlations to Facilitate Android Compatibility Issue Detection. In *IEEE/ACM 41st International Conference on Software Engineering.* IEEE, USA, 878–888.

[177] Ming Wen, Yepang Liu, Rongxin Wu, Xuan Xie, Shing-Chi Cheung, and Zhendong Su. 2019. Exposing Library API Misuses Via Mutation Analysis. In *IEEE/ACM 41st International Conference on Software Engineering*, Vol. 1. IEEE, USA, 866–877.

[178] Erik Wilde. 2018. Surfing the API Web. In *Companion of the The Web Conference 2018 on The Web Conference 2018 - WWW '18.* ACM Press, New York, New York, USA, 797–803.

[179] Titus Winters. 2018. Non-atomic refactoring and software sustainability. In *Proceedings of the 2nd International Workshop on API Usage and Evolution - WAPI '18.* ACM Press, New York, New York, USA, 2–5.

[180] Erik Wittern. 2018. Web APIs - challenges, design points, and research opportunities. In *Proceedings of the 2nd International Workshop on API Usage and Evolution - WAPI '18.* ACM Press, New York, New York, USA, 18–18.

[181] Erik Wittern, Annie T.T. Ying, Yunhui Zheng, Julian Dolby, and Jim A. Laredo. 2017. Statically Checking Web API Requests in JavaScript. In *IEEE/ACM 39th International Conference on Software Engineering*. IEEE, USA, 244–254.

[182] Erik Wittern, Annie T.T. Ying, Yunhui Zheng, Jim A Laredo, Julian Dolby, Christopher C Young, and Aleksander A Slominski. 2017. Opportunities in Software Engineering Research for Web API Consumption. *2017 IEEE/ACM 1st International Workshop on API Usage and Evolution (WAPI)* 1 (may 2017), 7–10. arXiv:1705.06586

[183] Wei Wu, Bram Adams, Yann-Gael Gueheneuc, and Giuliano Antoniol. 2014. ACUA: API Change and Usage Auditor. In *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*. IEEE, USA, 89–94.

[184] Wei Wu, Yann-Gaël Guéhéneuc, Giuliano Antoniol, and Miryung Kim. 2010. AURA. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, Vol. 1. ACM Press, New York, New York, USA, 325.

[185] Laerte Xavier, Aline Brito, Andre Hora, and Marco Tulio Valente. 2017. Historical and impact analysis of API breaking changes: A large-scale study. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, USA, 138–147.

[186] Tao Xie and Jian Pei. 2006. MAPO. In *Proceedings of the 2006 international workshop on Mining software repositories - MSR '06*. ACM Press, New York, New York, USA, 54.

[187] Deheng Ye, Zhenchang Xing, Chee Yong Foo, Jing Li, and Nachiket Kapre. 2016. Learning to Extract API Mentions from Informal Natural Language Discussions. *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)* 1 (oct 2016), 389–399.

[188] Reishi Yokomori, Harvey Siy, Masami Noro, and Katsuro Inoue. 2009. Assessing the impact of framework changes using component ranking. In *2009 IEEE International Conference on Software Maintenance*. IEEE, USA, 189–198.

[189] Ping Yu, Fei Yang, Chun Cao, Hao Hu, and Xiaoxing Ma. 2017. API Usage Change Rules Mining based on Fine-grained Call Dependency Analysis. In *Proceedings of the 9th Asia-Pacific Symposium on Internetware - Internetware'17*, Vol. Part F1309. ACM Press, New York, New York, USA, 1–9.

[190] Weizhao Yuan, Hoang H. Nguyen, Lingxiao Jiang, and Yuting Chen. 2018. LibraryGuru. In *Proceedings of the 40th International Conference on Software Engineering Companion Proceeedings*. ACM Press, USA, 364–365.

[191] Apostolos V. Zarras, Panos Vassiliadis, and Ioannis Dinos. 2016. Keep Calm and Wait for the Spike! Insights on the Evolution of Amazon Services. In *CAiSE*, Selmin Nurcan, Pnina Soffer, Marko Bajec, and Johann Eder (Eds.). Lecture Notes in Computer Science, Vol. 9694. Springer International Publishing, Cham, 444–458.

[192] Amir Zghidi, Imed Hammouda, Brahim Hnich, and Eric Knauss. 2017. On the Role of Fitness Dimensions in API Design Assessment - An Empirical Investigation. In *2017 IEEE/ACM 1st International Workshop on API Usage and Evolution (WAPI)*. IEEE, USA, 19–22.

[193] Tianyi Zhang, Ganesha Upadhyaya, Anastasia Reinhardt, Hridesh Rajan, and Miryung Kim. 2018. Are code examples on an online Q&A forum reliable?. In *Proceedings of the 40th International Conference on Software Engineering - ICSE '18*. ACM Press, New York, New York, USA, 886–896.

[194] Zhenchang Xing and Eleni Stroulia. 2007. API-Evolution Support with Diff-CatchUp. *IEEE Transactions on Software Engineering* 33, 12 (dec 2007), 818–836.

[195] Wujie Zheng, Qirun Zhang, and Michael Lyu. 2011. Cross-library API recommendation using web search engines. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering - SIGSOFT/FSE '11*. ACM Press, New York, New York, USA, 480.

[196] Hao Zhong, Suresh Thummalapenta, and Tao Xie. 2013. Exposing Behavioral Differences in Cross-Language API Mapping Relations. In *Fundamental Approaches to Software Engineering*, Vittorio Cortellessa and Dániel Varró (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 130–145.

[197] Hao Zhong, Suresh Thummalapenta, and Tao Xie. 2013. Exposing Behavioral Differences in Cross-Language API Mapping Relations. In *Fundamental Approaches to Software Engineering*. Springer, Berlin, Heidelberg, 130–145.

[198] Hao Zhong, Suresh Thummalapenta, Tao Xie, Lu Zhang, and Qing Wang. 2010. Mining API mapping for language migration. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering*, Vol. 1. ACM Press, USA, 195.

[199] Hao Zhong, Tao Xie, Lu Zhang, Jian Pei, and Hong Mei. 2009. MAPO: Mining and Recommending API Usage Patterns. In *Proceedings of the 23rd European Conference on ECOOP 2009 — Object-Oriented Programming (Genoa)*. Springer-Verlag, Berlin, Heidelberg, 318–343.

[200] Yibing Zhongyang, Zhi Xin, Bing Mao, and Li Xie. 2013. DroidAlarm. In *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security - ASIA CCS '13*. ACM Press, New York, New York, USA, 353.

[201] Jing Zhou and Robert J. Walker. 2016. API deprecation: a retrospective analysis and detection method for code examples on the web. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016*, Vol. 13-18-Nove. ACM Press, New York, New York, USA, 266–277.

[202] Minhaz F. Zibran, Farjana Z. Eishita, and Chanchal K. Roy. 2011. Useful, But Usable? Factors Affecting the Usability of APIs. In *2011 18th Working Conference on Reverse Engineering*. IEEE, USA, 151–155.