State of the Practice in Application Programming Interfaces (APIs): A Case Study

Raatikainen, Mikko

Springer-Verlag
2021

Raatikainen , M , Kettunen , E , Salonen , A , Komssi , M , Mikkonen , T & Lehtonen , T 2021 , State of the Practice in Application Programming Interfaces (APIs): A Case Study . in S Biffl , E Navarro , W Lowe , M Sirjani , R Mirandola & D Weyns (eds) , Software Architecture. ECSA 2021 : 15th European Conference, ECSA 2021, Virtual Event, Sweden, September 13-17, 2021, Proceedings . Lecture Notes in Computer Science , vol. 12857 , Springer-Verlag , pp. 191-206 , European Conference, ECSA , Sweden , 13/09/2021 . https://doi.org/10.1007/978-3-

http://hdl.handle.net/10138/345538
https://doi.org/10.1007/978-3-030-86044-8_14

acceptedVersion

State of the Practice in Application Programming Interfaces (APIs): A Case Study

Mikko Raatikainen[1][0000−0002−2410−0722], Elina Kettunen[1], Ari Salonen[2], Marko Komssi[3], Tommi Mikkonen[1][0000−0002−8540−9918] and Timo Lehtonen[4][0000−0001−8833−1725]

[1] University of Helsinki, Helsinki, Finland
first.last@helsinki.fi
[2] Digia Plc, Turku, Finland
ari.salonen@digia.com
[3] F-Secure Plc, Helsinki, Finland
marko.komssi@f-secure.com
[4] Solita Ltd, Tampere, Finland
timo.lehtonen@solita.fi

Abstract. Application Programming Interfaces (APIs) have become prevalent in today's software systems and services. APIs are basically a technical means to realize the co-operation between software systems or services. While there are several guidelines for API development, the actually applied practices and challenges are less clear. To better understand the state of the practice of API development and management in the industry, we conducted a descriptive case study in four Finnish software companies: two consultancy companies developing software for their customers, and two companies developing their software products. As a result, we identified five different usage scenarios for APIs and emphasize that diversity of usage should be taken into account more explicitly especially in research. API development and technical management are well supported by the existing tools and technologies especially available from the cloud technology. This leaves as the main challenge the selection of the right technology from the existing technology stack. Documentation and usability are practical issues to be considered and often less rigorously addressed. However, understanding what kind of API management model to apply for the business context appears as the major challenge. We also suggest considering APIs more clearly a separate concern in the product management with specific practices, such as API roadmapping.

Keywords: Software engineering · Application programming interface · API · API management · API management · Case study

1 Introduction

Application Programming Interfaces (APIs) have become prevalent in today's software systems and services. APIs play an integral role in enabling software systems to interact with each other and in allowing applications to be built on the

data or functionality of other systems. In fact, companies have been opening their products to third-party developers through various APIs for several years [4].

Despite being a central concept, the definitions for API are quite broad and diverse [11]. For instance, on one hand, the term API includes libraries, frameworks, and software development kits (SDKs). We refer these traditional APIs as *static APIs*. On the other hand, APIs are accessible over a network, such as SOAP or REST-based APIs from another component, system, or service. We refer to to these as *web APIs*. Diversity exists also in the goals of using APIs. Rauf *et al.* [24] see three main tasks for APIs. The first task is to enable software reuse via defining the interfaces of software components. The second task is to provide interfaces to software services that are available over a network. The third task is to enable the publication of open data through APIs.

Various API development and management guidelines have been summarized in textbooks (e.g., in [8]). In addition, many specific concerns of APIs have been addressed in research, such as API learnability [25], evolution [13], and usability evaluation [24].

However, a holistic overview of the API development and management state of the practice has not been presented. Thus, our goal in this study was to capture an overview of the API development and management in an industrial context from the API owners' and developers' view rather than API users' or client developers' point of view. We limit the investigation broadly to technical concerns so that, e.g., API business or monetizing models are not our primary concerns. The research problem we are addressing is:

- *How are APIs technically developed and managed in the software industry and what are the main challenges?*

The study was carried out as a descriptive case study [27] in four Finnish software product and consultancy companies. We interviewed one API expert from each company. The case study method was selected as the research method because we desired to construct broad and holistic understanding about practices related to APIs in the industry.

The rest of the paper is organized as follows. Section 2 covers the background and related work. Section 3 presents the research method and in Section 4 are the results of the case study. Section 5 discusses the results and Section 6 considers the validity of the study. Conclusions are in Section 7.

2 Background and previous work

Murphy *et al.* [19] conducted a study in which they interviewed professional software engineers and managers from seven different companies. In the study, the focus is on how API developers learned or received training on API design and what processes are included in API design. In comparison to Murphy *et al.*, our study is more focused on technical concerns of API properties and API development and management practices.

In contrast to our focus on API developers viewpoint, empirical studies of API client developers have been carried out revealing that often APIs are difficult to learn due to insufficient documentation [11,25]. In addition to being difficult to learn, APIs also often have issues with usability [20]. APIs have been also discovered to form from the API user's perspective broader boundary resources [3]. APIs have been found to be prone to evolve over time, and often changes result in breaking the code of API clients [5,10,13,15]. Especially refactoring is one major cause for API breaking changes [10]. Thus, API evolution is considered one major challenge in API development. Moreover, there may be issues with APIs changing without a warning [11]. In research the literature, the recommendations for API developers often include providing useful examples of the interaction with the API, versioning system, API usage monitoring, and API usability testing [6,11].

In addition to empirical studies, different types of API design guidelines have been published by many companies that provide REST APIs to client developers. In fact, Murphy *et al.* [18] conducted a study comparing 32 industrial REST API style guideline sets and discovered how these guides handle some topics differently and emphasize different concepts in API design.

Several academic papers have been written about API design and management also give recommendations for API developers. In a paper on API design, Henning [12] gives several recommendations on designing good APIs and emphasizes considering the needs of API callers in the design process. Espinha *et al.* [11] base their recommendations on interviews of API client developers and the recommendations include avoiding changes in the API, having a versioning system, and including examples of the interaction with the API in the documentation.

Despite the popularity of APIs in the practice and research literature, there are only a few systematic literature reviews that focus and summarize concerns specifically on APIs. By adapting the search protocol of an earlier study ([23]), we found four systematic literature reviews or mapping studies that focus on APIs and their properties [6,7,22,24]. However, these reviews — as summarized below — can be characterized as mapping studies aiming at finding proposed practices or guidelines rather than summarizing empirical evidence.

Burns *et al.* [6] have reviewed papers that have empirical justification for their recommendations regarding API development divided into three categories: design, documentation, and methodology recommendations. Design recommendations include technical recommendations such as preferring simple architecture and forms of object creation, using constructors instead of factories, avoiding unnecessary inheritance and configuration files, and not forcing developers to set all parameters. Documentation recommendations mention having useful examples in the documentation, documentation integration with the IDE, and using unit tests as a form of documentation. Methodology recommendations suggest using actual developers to evaluate API usability and finding out the participants' assumptions about an API. The recommendations focus on static APIs.

Cummaudo *et al.* [7] have synthesized a five-dimensional taxonomy for constructs needed to create good API documentation:

1. Usage Description (how the developer should use the API).
2. Design Rationale (when the API is the right choice for a particular use case).
3. Domain Concepts (why choose this API for a particular domain).
4. Support Artefacts (what additional documentation is provided by the API).
5. Documentation Presentation (visualization of the above information).

Each class of the taxonomy is further divided into 12, 7, 3, 6, and 6 different, more concrete categories, respectively. Of these categories, the primary studies mention most often code snippets, step-by-step tutorials, and low-level reference documentation for the usage description; purpose or overview of the API as a low barrier to entry to the design rationale; and the consistency in the look and feel of the documentation for the documentation presentation. While the taxonomy provides a long list of options for the documentation, as a systematic mapping study, the level of empirical evidence is not covered.

Nybom *et al.* [22] have studied generative approaches available for creating and improving API documentation. Generation and its tool support make creating and maintaining API documentation easier, as without proper tools code updates may render the documentation at least in some parts obsolete or inadequate. The study lists tools in its primary studies and constructed taxonomies for different sources for the generation, generation outputs, and quality properties of generative approaches. They conclude that many approaches contribute to API documentation in the areas of natural language documentation and code examples. All primary studies focus on new tool proposals rather than study the existing, well-known tools, such as Javadoc or Swagger/OpenAPI. Therefore, the tools appear to be mostly research prototypes or at least detailed empirical evidence is not provided, but the taxonomies provide an overview of different possibilities in generation. The focus is at least implicitly mostly on static APIs.

Rauf *et al.* [24] have studied different methods for evaluating API usability outlining and categorizing the different methods. The majority of studies use empirical methods about *API use* to evaluate the usability of an API. A less used category is analytical methods about *API specification* that includes reviews and metrics (the least addressed). Many research articles using empirical evaluation methods aim to provide guidelines, recommendations, or general principles that could be useful to API designers and developers in evaluating API usability. Only a few studies aim to provide tools for usability evaluation and most of these tools are not available for the public. The study differentiates usability evaluation phases to be design, development, and post-development, where the last category is dominant. They conclude that usability is inherently subjective and there are several factors or concepts but a lack of synthesis or agreement between the studies about the usability concepts.

Finally, there are also systematic literature reviews or mapping studies where APIs are covered shortly as a part of a specific context or application domain, such as in microservices or software ecosystems [1,9,17,26]. Notably, API versioning or evolution are mentioned as recurring challenges in three of these reviews [9,17,26]. The problem of API portability can also be seen similarly as a

challenge of different versions from the user perspective although the versions do not typically have drastic differences.

To summarize the previous research in terms of the objectives of this paper, we argue — in parallel with Rauf *et al.* [24] — that research needs also focus on the development concerns rather than the post-deployment phase and API users' or client developers' perspectives. Another notable shortcoming is that much of the research appears to be still done on static APIs rather than on web APIs. At least explicit differentiation between different types of APIs is not usually made. Finally, research often proposes guidelines or practices rather than characterizes industrial state-of-the-practice in terms of challenges and tested solutions.

3 Research method: Case study

This study was carried out as a descriptive case study [27] that studied APIs in four Finnish software companies. The primary selection of the companies for the study was two-fold: First, we selected consultancy companies that develop software for their customers both of them being large-sized and having several different customers. Second, we selected product companies that develop their own software being more focused on long-term development and maintenance of APIs. All products or projects were related to information systems, rather than, e.g., embedded systems, games, or mobile apps. We also practically knew that the selected companies considered APIs being important for their business. Otherwise, the case selection was based on convenience sampling, i.e. the companies that we had easy access to based on ongoing or earlier research collaboration.

The consultancy companies in the study are Digia Plc[5] and Solita Ltd[6]. Both of them are medium-sized having several projects and over 1000 employees. Both consultancy companies mainly operate on the Nordic or northern European markets. The third company, Vertex Systems Ltd[7], is a product-oriented company with around 100 employees. Vertex provides information management solutions for the industry, especially for industrial products and building construction design. The fourth company, F-Secure Plc[8], is a product- and service-oriented company with over 1700 employees providing cybersecurity solutions directly to their end-users and end-customers but also often through different partnership channels. Both these product companies operate on global markets.

The data was primarily collected by audio-recorded interviews that each took between one to two hours. Two university researchers acted as main interviewers with company representatives interested in the topic. One interview was carried out on the premises of the responding company while others were carried out by partly telecommuting due to the ongoing COVID-19 situation. Afterward, we asked a few clarification questions and let the respondents review and comment

[5] http://www.digia.com
[6] http://www.solita.fi
[7] https://www.vertex.fi
[8] https://www.f-secure.com

Table 1. The typical API technologies and practices in the case companies.

	Digia	**Solita**	**Vertex**	**F-Secure**
Use case	Integrations	Integrations, open APIs	Partner APIs	Partner APIs, internal APIs
Language	Java most common	Java most common	Java	Many
Framework	Spring Boot	Spring Boot	Spring Boot	Many
Cloud provider	Amazon, Azure, Google	Azure, Amazon, Google	None	Amazon
Container	Docker	Docker	No[1]	Docker
CI & CD	Yes	Yes	Partial	Yes
CI Tool	Jenkins	Jenkins	Bamboo	Jenkins
API Gateway	Yes	Yes	No	Yes
Monitoring	Yes	Yes	None	Yes
Logging	Elastic Stack, Splunk	Splunk, Elastic Stack	None	Splunk, proprietary
API Documentation	Annotation-driven (OpenAPI/ Swagger)	Annotation-driven (OpenAPI/ Swagger)	Postman	Endpoint reference, Open API, Github examples

[1] Container technology is currently being investigated.

on a draft of this paper. In addition, company representatives present in interviews co-author this paper but they are not the same representatives that were interviewed.

The interviews were semi-structured focusing on the projects or products that the respondent was familiar with. The questions in the interviews aimed to cover APIs holistically from the technological rather than business perspective. The interviews were structured by the following themes: technologies used for API development, deployment, and run-time; API testing; API monitoring and management; API documentation; API versioning practices; and API quality attributes. Thus, the themes covered roughly first the general life-cycle phases of API development and management, and then a few selected specific concerns. The SQuaRE product quality model of ISO25010:2011 [14] was used and shown to the respondent as a reference model for quality attributes during the study. The model covers eight main quality characteristics — functional suitability, performance efficiency, compatibility, usability, reliability, security, maintainability, and portability — that have further sub-characteristics.

4 Case study findings

We summarize a set of typical technologies and practices related to API development in the case companies in Table 1 that are elaborated in more depth in the following sections.

4.1 API usage scenarios

We identified four different usage scenarios for the concept of API. First, the consultancy companies developed most often APIs that exchange data between two or a few systems to integrate the systems for their customers — although there can be a few systems involved, we refer this to as *one-to-one integration APIs*. Second, all companies also developed *partner APIs* that their — or, in the case of consultancy companies, their customers' — partner organizations or customers can use. A partner API is similar to the one-to-one integration API but available to a broader audience of partners. Specifically, F-Secure has several partner APIs that their integration partners use for end-customer solutions. Third, especially Solita has projects to develop *open APIs* mainly to publish the data of public sector systems. The idea of such open APIs is that anyone can at least read data through the API. One example of open APIs are maritime traffic APIs (see [16]). Fourth, *internal APIs* are a means of reuse between products at F-Secure. F-Secure has many existing systems, some of which have been acquired by business mergers and acquisitions. These systems integrate through internally accessible APIs. However, some internal APIs are opened for trusted partner organizations, thus widening the scope of reuse to partner APIs.

All above APIs rely on web APIs except that F-Secure also applies static APIs for some of its internal APIs. Therefore, web APIs, especially in terms of REST-architecture and JSON-messages, have become prevalent industry-standard for all API usages. However, legacy systems still use SOAP or XML-based data and APIs, which must be considered especially in the consultancy companies when building APIs. Finally, other technologies for web APIs, such as GraphQL as an example of emerging technology, are not common although getting increasing interest in the industry.

4.2 Technologies and adoption of new technologies

While Table 1 highlights typical technologies and practices, there is a large selection of technologies that are used depending on the context of the project. Especially in the consultancy business, different technologies can be selected depending on the specific project. The product companies have a more focused selection of technologies but even F-Secure uses many different technologies, some of which pertain to the long history of the different systems, and some are inherited from business mergers and acquisitions. However, Java and Spring Framework are very common for all companies and projects.

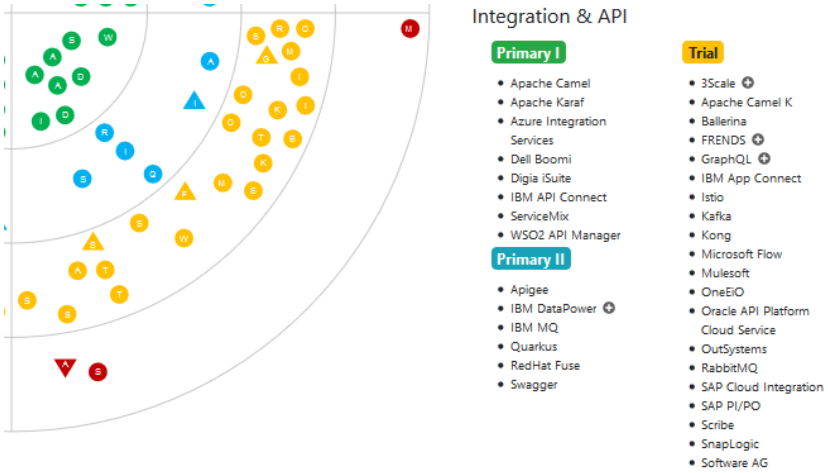

Fig. 1. A screenshot of the technology radar of Digia (https://techradar.digia.online).

The variety of technologies is well displayed by Digia's Tech Radar, which is used to understand and represent the ever-growing field of technological options both internally and to the customers. A screenshot of one quadrant in the Tech Radar is presented in the following[9]. The quadrant in Figure 1, Integration & API, is the most relevant in the context of this paper. Tech Radar contains four prioritized classes of the most relevant technologies represented by four rings: Primary I (actively used and preferred), Primary II (widely used), Trial (emerging and fading away), and Hold (not yet used or avoided). Plus signs are used to indicate recent new entries and arrows the direction within the rings (not visible in the screenshot). The first two rings contain the most important technologies. For instance, in Integration & API quadrant, one of the primary technologies used is Apache Camel. In the trial ring, there is Camel K mentioned as an emerging technology. Technologies that are fading away in the Integration & API quadrant, are, for instance, Scribe/Tibco. The other quadrants also contain some relevant technologies for API development. For instance, in DevOps & Tools quadrant, commonly used technologies for infrastructure-as-code approach [2], e.g., Ansible and Docker, are mentioned.

The majority of preferred technologies are similar and alternative to each other although some variations exist between companies. For example, both consultancy companies, Digia and Solita, make extensive use of the open source Apache Camel framework as the main integration tool. Digia, as noted also above in Tech Radar, was also interested in newer Camel K, which can run integrations on Kubernetes. Solita, however, does not use Kubernetes as much and expressed no interest in adopting Camel K to their technology stack for the

[9] For readability and space reasons, we can only show one quadrant. Full Tech Radar is available at https://techradar.digia.online.

time being. There were also certain excluding selections, such as one company did not practically use IBM's tool stack at all. One key reason for not using a certain technology is license costs. A commercial stack may have remarkable license costs already at the beginning of the project, whereas a stack based on Open Source products is free from license costs. Consultancy companies make a profit by selling man-months, i.e., their business model is often based on providing consultancy services in long-term customer relationships without license costs.

Especially among the personnel of the consultancy companies, technologies are concentrated around interest groups with preferred technologies. For instance, the two consultancy companies have *tribes* or *competence communities* gathering around selected technologies that can complement each other, such as Amazon and Microsoft Azure cloud technologies. The groups of people share knowledge on the existing technologies and actively adopt new emerging technologies from the market or open source communities.

4.3 Deployment

Cloud deployment has become a de facto practice. A cloud readily offers some essential services, such as an API gateway, a developer portal, and federated user management. Of the cloud services, Amazon Web Servers (AWS) and Microsoft's Azure are the two most commonly used especially in the projects of the consulting companies. Google Cloud is used significantly less. Azure is very widely used in organizations in the public sector in Finland. A reason for the popularity was assumed to be the existing licenses to Microsoft products in the public sector customers in Finland, which makes Azure an easy add-on option to the existing technology stack of the organization. Containers, especially Docker, also are usually used. Serverless architecture is also becoming increasingly common but by no means prevalent. There are still server-based installations: Vertex had their software and APIs still running on their servers but transferring to a cloud environment and container technology is under investigation and planned in the near future.

Continuous integration and delivery (or deployment) (CI/CD) features have also become the status quo in all companies involved in the study. The differences are in the environments used. The consultancy companies have typically separate environments for development, test, and production with deployments by a modern CI tool, most often Jenkins. F-Secure has an additional partner-specific deployment and test environment where new versions are released. Vertex had currently only the development and production environments in addition to a playground server. The speed of CI/CD is the main challenge rather than CI/CD itself. Some software frameworks, such as Spring, require inconveniently long compiling time for today's CI/CD needs.

4.4 API versioning

API versioning was considered a significant maintenance effort in the worst case, but not especially technically challenging per se. As probably the most rigorous

example of versioning, F-Secure applies semantic versioning[10] in their APIs consisting of the major, minor, and patch version values (major.minor.batch). A new major version can break backward compatibility but the end-of-life policy of F-Secure guarantees a transition period of at least 12 months. F-Secure aims to retain backward-compatibility so that the older API versions can be kept online together with the newer versions. The consultancy companies likewise apply API versioning although there are no such explicit guidelines or commitment as in F-Secure because versioning is also the responsibility of their customers. It was noted that especially integration APIs can be changed without versioning if backward compatibility is guaranteed, such as in the case of adding endpoints. Vertex, which in the early stages with APIs, had not yet adopted versioning practices. Although several versions of APIs can be available simultaneously, old versions are taken down when they are not used anymore. Monitoring the API usage is used as an indicator when the old version can be taken down.

4.5 API documentation

As REST is the dominant architectural style for APIs, the endpoints in the consultancy companies are documented using OpenAPI Specification[11] (or Swagger as it was earlier known). The consultancy companies also sometimes set up a developer portal for their client companies. Solita mentioned using API catalogs that can contain also other written documentation, such as tutorials and guides, in addition to generated technological specifications. Vertex uses Postman to produce similar endpoint specifications as OpenAPI. F-Secure maintains a reference guide to its API endpoints that is constructed using a proprietary tool. Each endpoint has a description and types of its data, and one or a few simple examples of the calls and responses. In addition, there are a few more complex use cases. The APIs have also OpenAPI specifications. API usage code examples are given publicly in Github[12].

However, in all companies, the documentation consists mainly of technical endpoint specifications and code examples, and there are usually very few other kinds of documentation, such as tutorials or getting started guidelines. The companies cited the lack of resources as the main reason for not having more examples of the API usage in the documentation. For the consultancy companies, much of the documentation was considered more of the responsibility of their customers. The operations of one-to-one integration APIs were noted to be quite similar create, read, update and delete (CRUD) operations or other equivalent obvious functionality that a domain expert can learn from technical endpoint documentation without the need of more thorough documentation. Similarly, in the case of F-Secure, the users of APIs are often partner companies who build long collaboration relationships and have gained understanding about the domains and products of the APIs. Being partner companies, F-Secure can also give necessary guidance directly.

[10] https://semver.org/
[11] https://swagger.io/specification/
[12] E.g., https://github.com/F-Secure/atlant-api

4.6 API quality attributes

The companies did not report major challenges related to quality attributes in APIs based on the SQuaRE [14] reference model. The existing standards, such as OpenID Connect for security, and technologies, such as cloud servers and API gateways for performance and scalability, provide typically sufficient solutions for the majority of quality characteristics.

At the moment, Vertex has no issues with quality attributes despite relying on server-based deployment. However, in the future, the company has plans for more cloud-based solutions as a means to ensure dependability and scalability with a more extensive user base. Challenges with quality attributes, especially run-time quality such as reliability and performance, pertain typically to the problems in the backend systems, such as databases, rather than APIs in all companies. In a similar manner in the case of large data, network bandwidth can also become a consideration.

The only issues concerning quality attributes were related to usability, portability, and maintainability. From the usability point of view, specific API testing was not conducted in any company. For example, F-Secure considers API testing as a part of product testing and usability is a part of general usability that the product management is responsible for. In terms of portability, the consultancy companies reported that portability — or adaptability as its sub-characteristic — may require additional effort although it is not problematic per se. A typical example is a deployment to a different cloud environment, or both to a cloud and a local server. Finally, reusability was seen as the main point of API development, but developing an API that is as reusable as possible and not difficult to use was noted to be an endless issue for the design.

4.7 API management

The interest in API management among the customers of the consultancy companies was mentioned being increasing considerably during the past few years. Much like above, the existing technologies mostly provide sufficient tools: For instance, Azure was mentioned having a good tool for API management or dedicated tools can be used, such as WSO2, IBM API Connect, TreeScale, Kong, and ServiceMix. However, the respondents noted that despite the technologies exist, the concept of API management is still vague in terms of what API management should cover, and when and to what extend dedicated API management is needed. In addition, the business models based on APIs were perceived as immature. For instance, monetizing the APIs was not really considered by the consultancy companies, while Vertex and F-Secure apply a subscription model for their API for the paying customers.

As an example of technical API management, API monitoring appeared particularly mature. Both consultancy companies used designated tools for API monitoring and logs. For example, Solita strives to use a centralized log management system and a uniform log format. The Camel framework offers components for sending API log data. Digia mentioned using also Splunk, Elastic, and Kibana

in real-time usage monitoring. As a cybersecurity company, F-Secure uses extensive API monitoring and multiple levels of logging: They have an internal monitoring system and logging with Splunk and they use the alert system of Amazon's cloud. The aim is to monitor the whole flow. Vertex is not yet using comprehensive API monitoring, but logs are used for basic access monitoring. However, they are very interested in developing their API management system and more detailed monitoring.

5 Discussion

Our findings show that the concept of API has become, and is getting broader and more diverse. We identified four usage scenarios for APIs: one-to-one integration APIs, internal reuse, partner APIs, and open APIs. All of these are widely based nowadays on the REST architectural style making web APIs prevalent in industrial software projects for information systems. However, more traditional static APIs of SDKs, libraries, or frameworks are still used within the code as the fifth usage scenario as evidenced by the use of various open source and cloud technologies. Similar static APIs are also still used as a means of intra-organizational reuse. While this presented classification is quite straightforwards, it is often implicit. Moreover, the classification can be refined and other classifications from other viewpoints can be proposed. However, rather than the classification alone, we emphasize that the diversity of API usage scenarios in practice should be more explicitly taken into account in research since much of the research appears to be implicit about the nature of API.

Existing tools and technologies make most of the technical tasks in API development, deployment, and management straightforward and ensure sufficient runtime quality of APIs. The main technological challenges are, in fact, related to the backend systems or computing infrastructure rather than to APIs themselves. The major decision for APIs is the selection of the desired technology from the existing technology stacks of multiple similar and comparable options for which many technologies are inherited from the backend technology selections. Mature techniques might be in some cases old-fashioned and inefficient, but on the other hand, newer fashionable technologies may suffer from immaturity. Similarly, personal preferences and familiarity can favor certain technologies. However, the technologies appear to be quite comparable, and advance and mature rapidly.

Despite being often presented as challenges in the literature (e.g., [11,13]), API evolution and versioning were not considered to be especially technically challenging by the practitioners. However, API evolution can be more problematic to the client developers, and the companies we studied mainly provide APIs, so the viewpoint is most likely different. API evolution is not either always visible to consultancy companies as they do not necessarily maintain the API for longer time. Nevertheless, API evolution, even if not being a challenge, requires considerable manual effort and rigor in order not to break the applications us-

ing the API. A research challenge is to reduce the manual work and help with ensuring continuity of APIs.

In contrast to evolution, documentation and usability appeared more as a practical issue. First, API documentation appeared often to be a somewhat under-addressed practice. Similar results were presented in [19]. Although API endpoint specifications are constructed using OpenAPI or a similar tool, more detailed API user guidelines are still often lacking. That is, a *developer journey* [21], which roughly means attracting and supporting new developers for an API, keeping them motivated as they learn to use the API, and provide specialized and detailed support and tools also for experts, is not actively considered. Second, API usability is not necessarily designed or it is treated as a part of the product quality cursorily. API usability might sometimes need reconsideration because the users of APIs are software developers that can be different from the end-users. Interestingly for example at F-Secure, the same product manager was responsible for a product and its APIs although the users of products and APIs are quite different. Who should be responsible for APIs is a topic that deserves further investigation.

However, it is important to take into account that even though API usability and documentation are not always extensively addressed in industrial practice, the rigor of addressing these disciplines depends on the API context and use cases. API usability and documentation become more important or even relevant when APIs are used in a broader context with external developers or even unknown developers rather than only internally or for integration.

API management appeared clearly as the vaguest and most challenging concept. API management was mentioned as being increasingly important for customers of the consultancy companies. As the discussion and information on APIs have become more widespread, some customers have become concerned about whether they too should have an API and API management system even though they might not have an exact idea of what an API actually is or if a full API management system is even completely necessary for their business. Likewise, for API development, there are several individual tools available for API monitoring and management, and the technical aspects of the process are not as challenging as deciding what to do and handling the received data. That is, besides using API monitoring for the availability, performance, and functional correctness of an API, monitoring could be used for understanding customers and API users. Thus, the greatest challenges pertain to better understanding the needs of API management such as what kinds of policies or models to use for API management in different situations from the business perspective.

Overall, APIs were always considered an integral part of products (or services) and managed as a part of the products. However, APIs have certain inherent characteristics that make them different from products. For example, the user and customer of an API can be different from that of a product. Furthermore, as several other systems and stakeholders can rely on APIs, the APIs cannot be developed quickly to prototype different features as in agile and DevOps practices for product development. Therefore, it could be beneficial to consider

APIs at least more clearly a separate concern in the product management having certain dedicated practices. For instance, having a dedicated API roadmap planning could help in diminishing the technical effort required in API versioning. As APIs have become such a quintessential part of software systems and services, APIs could be considered at least a more separate concern in product management and business planning.

Finally related to APIs being a separate concern of a product, one interviewed API expert mentioned how API design and use are not taught at educational institutions similarly to software design in general. Therefore, all API developers have to be taught at work, at least to some extent. Similar notions of learning on the job were brought up by API developers interviewed in [19]. This raises questions on the need to emphasize API development and management in computer science curricula at educational institutions. Courses covering API development and design could help to prevent some of the API usability issues and help with technology transfer from research to practice. This is becoming more relevant as APIs are increasingly important in many software systems.

6 Study validity

A threat to construct validity is that only limited sources of evidence were used relying on one or a few interviewees. However, to mitigate the threats to construct validity, we let the respondents read and comment on a version of this manuscript, and other representatives from companies co-author this paper. Another threat to construct validity relates to the respondents in the consultancy companies that described practices in several projects. Their selection of examples could have been unintentionally biased or they might have done generalizations across projects. To mitigate biases in cases, we purposefully selected different kinds of cases that represent product- or service-oriented business and consultancy business. We did not either initially limit our selection to four cases but considered during data collection that we had quite saturated understanding of the state of the practice and decided not to need additional cases.

A threat to external validity in qualitative studies pertains to the selection of cases. This study focused on companies and projects that develop information systems and are geographically limited. Therefore, the results may not be generalizable especially to other technological domains, such as mobile apps or embedded systems.

7 Conclusions

We presented a case study on four Finnish companies to characterize API development and management state-of-the-practice in the context of information systems. The study shows that APIs and API management are important and prevalent practices in the software industry. We identified internal reuse, partner APIs, one-to-one integration APIs, and open APIs as web-based complementing usage scenarios to more traditional static APIs in, e.g., SDKs, libraries, or

frameworks. While these usage scenarios identify some diversity, we argue that the diversity in the nature of APIs — and especially web APIs — should be better and more explicitly taken into consideration in research and practice.

API development and management are technically well supported in the practice by existing tools and technologies especially related to cloud services and provided by cloud vendors. API usability and documentation as design problems are most challenging to be properly addressed. The main technical challenge remains to be the selection of the right technologies from the existing technology stacks. Although API management is supported technically, the challenges in practice include what kinds of policies or models to use for API management in different business contexts. We suggest considering APIs more clearly at least as a separate concern in the product management, which might require some novel API specific practices, such as API roadmapping. In terms of future research, solutions for API roadmapping could be proposed and a more extensive empirical investigation could be carried out even by quantitative methods.

Acknowledgements

We acknowledge the financial support of Business Finland as a part of 4APIs project.

References

1. Alshuqayran, N., Ali, N., Evans, R.: A systematic mapping study in microservice architecture. In: IEEE International Conference on Service-Oriented Computing and Applications. pp. 44–51 (2016)
2. Artac, M., Borovssak, T., Di Nitto, E., Guerriero, M., Tamburri, D.A.: Devops: introducing infrastructure-as-code. In: IEEE/ACM International Conference on Software Engineering (Companion volume). pp. 497–498 (2017)
3. dal Bianco, V., Myllärniemi, V., Komssi, M., Raatikainen, M.: The role of platform boundary resources in software ecosystems: A case study. In: IEEE/IFIP Conference on Software Architecture. pp. 11–20 (2014)
4. Bosch, J.: From software product lines to software ecosystems. In: AMC International Conference Software Product Lines. pp. 111–119 (2009)
5. Brito, A., Valente, M.T., Xavier, L., Hora, A.: You broke my code: understanding the motivations for breaking changes in APIs. Empirical Software Engineering **25**(2), 1458–1492 (2020)
6. Burns, C., Ferreira, J., Hellmann, T.D., Maurer, F.: Usable results from the field of API usability: A systematic mapping and further analysis. In: IEEE Symposium on Visual Languages and Human-Centric Computing. pp. 179–182 (2012)
7. Cummaudo, A., Vasa, R., Grundy, J.: What should I document? a preliminary systematic mapping study into API documentation knowledge. In: ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (2019)
8. De, B.: Api Management. Springer (2017)

9. De Lima Fontao, A., Dos Santos, R., Dias-Neto, A.: Mobile software ecosystem (MSECO): a systematic mapping study. In: International Computer Software and Applications Conference. vol. 2, pp. 653–658 (2015)
10. Dig, D., Johnson, R.: How do APIs evolve? A story of refactoring. Journal of software maintenance and evolution: Research and Practice **18**(2), 83–107 (2006)
11. Espinha, T., Zaidman, A., Gross, H..: Web API growing pains: Loosely coupled yet strongly tied. Journal of Systems and Software **100**, 27–43 (2015)
12. Henning, M.: API design matters. ACM Queue **5**(4), 24–36 (2007)
13. Hora, A., Robbes, R., Valente, M.T., Anquetil, N., Etien, A., Ducasse, S.: How do developers react to API evolution? A large-scale empirical study. Software Quality Journal **26**(1), 161–191 (2018)
14. ISO/IEC: 25010:2011, Systems and software engineering — Systems and software quality requirements and evaluation (SQuaRE) — system and software quality models (2011)
15. Jezek, K., Dietrich, J., Brada, P.: How Java APIs break - an empirical study. Information and Software Technology **65**, 129–146 (2015)
16. Joutsenlahti, J., Lehtonen, T., Raatikainen, M., Kettunen, E., Mikkonen, T.: Challenges and governance solutions for data science services based on open data and apis. In: IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI of International Conference on Software Engineering (2021)
17. Manikas, K.: Revisiting software ecosystems research: A longitudinal literature study. Journal of Systems and Software **117**, 84–103 (2016)
18. Murphy, L., Alliyu, T., Macvean, A., Kery, M.B., Myers, B.A.: Preliminary analysis of REST API style guidelines. Ann Arbor **1001**, 48109 (2017)
19. Murphy, L., Kery, M.B., Alliyu, O., Macvean, A., Myers, B.A.: API designers in the field: Design practices and challenges for creating usable APIs. In: IEEE symposium on visual languages and human-centric computing. pp. 249–258 (2018)
20. Myers, B.A., Stylos, J.: Improving API usability. Communications of the ACM **59**(6), 62–69 (2016)
21. Myllärniemi, V., Kujala, S., Raatikainen, M., Sevón, P.: Development as a journey: factors supporting the adoption and use of software frameworks. Journal of software engineering research and development **6**(1) (2018)
22. Nybom, K., Ashraf, A., Porres, I.: A systematic mapping study on API documentation generation approaches. In: Euromicro Conference on Software Engineering and Advanced Applications. pp. 462–469 (2018)
23. Raatikainen, M., Tiihonen, J., Männistö, T.: Software product lines and variability modeling: A tertiary study. Journal of Systems and Software **149**, 485–510 (2019)
24. Rauf, I., Troubitsyna, E., Porres, I.: Systematic mapping study of API usability evaluation methods. Computer Science Review **33**, 49–68 (2019)
25. Robillard, M.P.: What makes APIs hard to learn? Answers from developers. IEEE Software **26**(6), 27–34 (2009)
26. Soldani, J., Tamburri, D., Van Den Heuvel, W.J.: The pains and gains of microservices: a systematic grey literature review. Journal of Systems and Software **146**, 215–232 (2018)
27. Yin, R.K.: Case study research: Design and methods. Sage (2014)