Developer Perspectives on REST API Usability: A Study of REST API Guidelines

Developer Perspectives on REST API Usability: A Study of REST API Guidelines

Sven Peldszus
Chalmers | University of Gothenburg
Sweden
Ruhr University Bochum
Germany
IT University of Copenhagen
Denmark
sven.peldszus@gu.se

Jan Rutenkolk
Marcel Heide
Jan Sollmann
Ruhr University Bochum
Germany
jan.rutenkolk@rub.de
marcel.heide@rub.de
jan.sollmann@rub.de

Benjamin Klatt
Frank Köhne
viadee AG
Cologne, Germany
benjamin.klatt@viadee.de
frank.koehne@viadee.de

Thorsten Berger
Ruhr University Bochum
Germany
Chalmers | University of Gothenburg
Sweden
thorsten.berger@rub.de

Abstract

REST is today's most widely used architectural style for providing web-based services. In the age of service-orientation—a.k.a. Software as a Service (SaaS)—APIs have become core business assets and can easily expose hundreds of operations. While well-designed APIs contribute to the commercial success of a service, poorly designed APIs can threaten entire organizations. Recognizing their relevance and value, many guidelines have been proposed for designing usable APIs, similar to design patterns and coding standards. For example, Zalando and Microsoft provide popular REST API guidelines. However, they are often considered as too large and inapplicable, so many companies create and maintain their own guidelines, which is a challenge in itself. In practice, however, developers still struggle to design effective REST APIs. To improve the situation, we need to improve our empirical understanding of adopting, using, and creating REST API guidelines.

We present an interview study with 16 REST API experts from industry. We determine the notion of API usability, guideline effectiveness factors, challenges of adopting and designing guidelines, and best practices. We identified eight factors influencing REST API usability, among which the adherence to conventions is the most important one. While guidelines can in fact be an effective means to improve API usability, there is significant resistance from developers against strict guidelines. Guideline size and how it fits with organizational needs are two important factors to consider. REST guidelines also have to grow with the organization, while all stakeholders need to be involved in their development and maintenance. Automated linting provides an opportunity to not only embed compliance enforcement into processes, but also to justify guideline rules with educational explanations.

CCS Concepts

• **Software and its engineering** → **Software design engineering**; **Software usability**; **Software design techniques**; • **Information systems** → **RESTful web services**.

Keywords

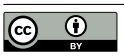
REST API, usability, design, guidelines

ACM Reference Format:
Sven Peldszus, Jan Rutenkolk, Marcel Heide, Jan Sollmann, Benjamin Klatt, Frank Köhne, and Thorsten Berger. 2026. Developer Perspectives on REST API Usability: A Study of REST API Guidelines. In *34th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (FSE Companion '26), July 05–09, 2026, Montreal, QC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3803437.3805267

1 Introduction

Web service APIs are critical drivers for internal operations and the monetization of software services. Internal APIs connect an organization's IT systems, and external APIs allow other organizations to access services. According to hundreds of developers surveyed [51], web service APIs are intensively used by organizations of any size. Large organizations with more than 10,000 employees often use over 250 APIs, while smaller organizations already use up to 50.

Today, Representational State Transfer (REST) is the dominant architectural style for delivering web services [51]. Already in 2017, REST was used in 18,400 APIs (82 % of all known APIs) [55], growing to 24,471 by 2022 [22]. RESTful services are stateless and well separated from their clients via a unified interface, typically using HTTP endpoints. Their APIs can be relatively large, with 62 % of REST APIs providing between 11 and 100 operations, and 18 % even more [7].



FSE Companion '26, Montreal, QC, Canada

ACM ISBN 979-8-4007-2636-1/2026/07
https://doi.org/10.1145/3803437.3805267

While RESTful services enable loose client-server coupling and independent service development, this flexibility introduces new challenges. Most importantly, for a service to be actually used, its API must be easy to use, especially when competing with similar services of other organizations. Usability often suffers from inconsistent naming or usage across API parts, sometimes to the extent that the service no longer adheres to REST principles [27].

In practice, organizations try to address API usability through developer guidelines that describe best practices for API design. These are similar to design patterns [20] or coding standards, such as those by the SEI CERT [10]. While there are many well-documented REST API guidelines available, inconsistencies in REST APIs persist. HTTP traffic analyzed in 2016 revealed that *"implementation and usage of REST APIs—as well as that of Web services more in general—is still far from being a stable and consolidated discipline."* [54]. According to preliminary discussions with industry experts, this has been a recurring theme over the past decade and has not improved since 2016. REST API usability remains a major pain point. It is unclear what social or technical factors explain these observations, whether unclear definitions, disagreements, or random errors hinder the creation of high-quality REST APIs, or whether the observed inconsistency is an accepted byproduct of the development process. To address this gap, three research questions must be answered.

RQ1: *What is the developers' understanding of REST API usability?* To determine the impact of guidelines, it is essential to capture developers' general knowledge and understanding of RESTful services, with a focus on API usability. We investigated what developers consider a usable REST API and what factors influence usability.

RQ2: *What obstructs the successful adoption of API guidelines?* While REST API guidelines are considered and important means to develop usable APIs, they do not seem to be having their intended effect. We study the reasons, specifically how the quality and organizational aspects influence the adoption of guidelines.

RQ3: *What challenges creating and tailoring API guidelines?* Since existing guideline catalogs are often large, generic, and difficult to adopt, organizations often need to create custom guidelines, or tailor existing ones to the organization or concrete project.

We present an interview study with 16 REST experts with substantial experience developing and maintaining REST APIs for different companies and domains. Our online appendix [46] contains the interview guide. We transcribed and analyzed the interviews using open-coding to answer our research questions, determining the notion of API usability, challenges of adopting and designing guidelines, and best practices to develop usable RESTful services.

On a final note, according to one of our experts, REST API guidelines are *"a giant topic under which you can summarize anything you want."* This underscores the complexity and diversity of REST API guidelines. In this light, our results help clarify the role and success factors of REST API guidelines. We hope they facilitate an informed discussion of challenges, potential mitigation strategies, and the appropriate balance between developer freedom and API consistency.

2 Background and Motivation

We now briefly introduce REST and motivate our study by discussing API usability and the guidelines landscape.

Representational State Transfer (REST). REST is an architectural style for web services [18] that encompasses six properties. (1) REST architectures realize the client-server model [56] to enforce separation of concerns, (2) but optionally, servers can provide code on-demand [8, 9] for client-side execution to minimize traffic. (3) RESTful services are stateless, avoiding to maintain sessions or store request-specific data on the server, but requiring each client request to include all necessary information. (4) This allows caching of responses and improves scalability and performance. (5) Clients cannot detect intermediaries, e.g., layers enforcing security, between themselves and the server. (6) Services are provided via a uniform API that may not reflect internal server data structures, allowing clients and services to evolve independently.

REST API uniformity includes a Uniform Resource Identifier (URI) [39] for all accessible information, and a Uniform Resource Locator (URL) [38] for each service. URLs for follow-up requests are provided on demand in responses (cf. hypermedia as the engine of application state (HATEOAS) [3]). All responses use a consistent data format across the REST API, which may differ from internal structures but is detailed enough to support data manipulation.

How to realize REST architectures is not standardized [27]. In practice, services are mostly provided via HTTP endpoints, i.e., URLs targetable via HTTP requests, that return JSON data. However, REST does not specify concrete technologies to be used, so developers could use XML instead of JSON for data representations. APIs are either provided from provider to consumer, client developers in this case, or through public API directories [14].

Several REST API frameworks are used to implement RESTful services [37]. Among them, OpenAPI/Swagger [57] is the most widely used framework for HTTP-based REST APIs and allows, among other things, generating server or client code from a REST API specification in OpenAPI format or generating an OpenAPI specification from annotated source code [35]. The OpenAPI format is a JSON-based, human-readable and machine-processable structure, thus facilitating understanding of services without requiring access to source code, documentation, or network traffic analysis.

REST API Usability. With the success of RESTful services, corresponding APIs have grown large [16], and maintaining multiple, long-lived APIs has become common. This suggests similar effects on REST API quality, i.e., usability, similar to software aging in traditional software [43], i.e., the steady decay of software quality with changes. This can negatively impact the usability and security of a REST service. Furthermore, since REST is no official standard [27] like SOAP, with specific rules and syntax, this leads to an unfocused field of API usability and deviation from REST principles is common in practice. Often, REST becomes a synonym for any Web API utilizing URIs and HTTP [27]. In particular, API consistency is an issue, since REST provides no standard for naming conventions, data formats, or similar concepts. Thus, collaborative development results in inconsistencies that impact usability.

To illustrate the challenges of inconsistent REST APIs, Fig. 1 shows three consecutive API calls in a shop system. The frontend must connect to APIs provided by different teams. The superscripts illustrate inconsistencies in the API designs: a) versioning schemes, i.e., major version only vs. minor version, b) inconsistent parameter types, i.e., path parameter vs. query parameter, and c) inconsistent

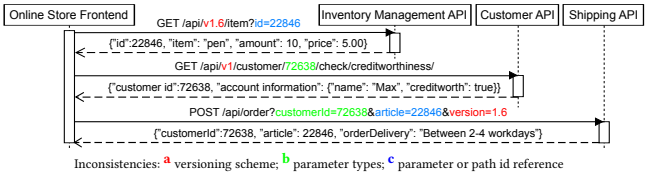


Figure 1: Illustration of inconsistently designed REST APIs

wording of semantically identical entities, i.e., id vs. name. This challenges comprehension, maintenance, and evolution of APIs.

REST API Guidelines. To mitigate API quality problems, companies started creating API design guidelines to foster consistent API landscapes. Guidelines provide examples and best practices for developers, and are often the basis for creating custom guidelines [36]. Such guidelines are typically a set of rules, each regarding different aspects in REST API development, i.e., one rule in Zalando's guideline in the category *"REST Basics - Security"* is that REST APIs *"MUST secure endpoints"* [67]. To this end, various security features can be used [25]. Guidelines, however, can vary widely because they are based on the experience of developers and the institutions in which they are created and there are many guidelines—a GitHub search for "REST Guideline" in September 2025 returned 102 results. More systematically, Wilde collected 27 guidelines and two compilations of guidelines [64], including the API Stylebook [30]. Murphy et al. analyzed 32 public REST API guidelines and defined 27 aspect categories, e.g., naming conventions for URIs [36].

However, many REST APIs and their associated guidelines exhibit inconsistencies, concerning varying and sometimes conflicting approaches to key aspects such as documentation, error handling, and overall design standards [36]. Such inconsistencies may partly arise from linguistic antipatterns and poor documentation practices [42]. Implementing advanced REST concepts such as HATEOAS is not a priority for many developers [27]. Before, developers were more familiar with SOAP, which provides more structured guidance compared to the more flexible and loosely interpreted REST guidelines, resulting in low adoption of best practices and ideas of REST API development as captured in REST guidelines [54]. However, the current state of practice is unclear. To enhance the standardization and usability of REST API guidelines, one suggestion is to develop more machine-readable resources [63].

Overall, API guidelines are widely used, but it is questionable whether they achieve their goals. It is unclear how developers perceive and use API guidelines, which needs to be understood to identify improvements and best practices.

3 Related Work

Several studies address challenges in REST API design. Bogner et al. experimentally confirm that adhering to RESTful API design rules significantly improves the understandability of APIs, and that violations clearly reduce comprehension regardless of user experience [5]. Yamamoto et al. emphasize the need for automation to resolve behavioral discrepancies between web-based and local REST APIs, supporting consistency [66]. Daughtry et al. focus on improving API usability through clear and accessible documentation using tools like Swagger [15]. In this context, Coblenz et al. show that while OpenAPI and static analysis tools support syntactic validation of REST APIs, developers face challenges in automating usability, context-aware design, and real-time feedback, underscoring the need for combining automated checks with manual, human-centered processes [12]. The Richardson Maturity Model has become a widely referenced standard framework for assessing REST APIs, from basic HTTP usage (Level 0) to advanced HATEOAS (Level 3) [19]. Robles et al. analyze REST API evolution, showing that breaking changes decrease in newer versions, enhancing client-side stability [53].

Nguyen et al.[41] highlight REST API security concerns, noting the absence of standardized specifications compared to SOAP. Iacono et al. underscore this need for robust security frameworks, especially in message-oriented architectures [32]. Abdulghani et al. propose security guidelines for IoT, pointing out gaps in standardized frameworks for REST API security, especially in resource-constrained environments [1]. Collectively, these studies call for comprehensive REST API standards that balance complexity, usability, and security. Work attempting to mitigate these challenges includes a proposed set of REST API design guidelines based on existing Web standards [63]. Koci et al. propose metrics for web API usability, expressed through the predicted error rate, and trained a classifier model to predict the error rates of API endpoints [28]. Verborgh and Dumontier[62] explore feature-based reuse in web APIs, enabling shared client/server code, documentation, and tools to promote cross-API compatibility. Daugthry et al. describe eight ways developers use interactive explorers for web APIs, grounded in empirical analyses of the Google APIs Explorer [15].

Petrillo et al. survey the literature to extract a catalog of 73 best practices for REST API design and study well-known REST APIs (e.g., Google Cloud Platform) to determine how they are offered and accessed [47]. Similar to our study, Zhang et al. interview 23 API designers from 6 companies and 11 open-source projects to understand their practices and needs, but they focus on user feedback through bug reports and peer reviews [68]. In addition, Piccioni et al. present an API usability study on 25 programmers, which combines interview questions with systematic observations of programmer behavior while solving token-based programming tasks. So, their scope is limited to API design [48].

While related works demonstrate the advantages of adhering to REST API guidelines and address their specifics, the factors necessary for successfully adopting them remain unexplored, which is one of the aspects our study addresses.

4 Methodology

Our study required an exploratory, qualitative methodology able to uncover relevant factors. Hence, we employed a respondent strategy [59], specifically an interview study. We conducted semi-structured interviews [2] with knowledgeable experts.

Participant Selection. We sampled interviewees via an industrial partner (a consulting company whose experts work for many companies) and its network, following a purposive sampling strategy [44]. Sampling criteria included professional expertise with REST APIs, diverse backgrounds, and different relations with REST APIs. We ensured a range of seniority, from younger software developers to experienced software architects, as well as specialists and experts involved in creating popular guidelines and enterprise architecture tooling. We selected both providers and consumers of REST

APIs to incorporate perspectives on creating and using REST API guidelines. We ensured a diverse range of backgrounds, including domains such as finance, insurance, telecommunication, commerce, and healthcare. Participants were recruited continuously until we reached saturation. This way, we recruited 16 knowledgeable experts, whose experience, current role, relation to REST APIs, and professional focus (e.g., as consultants for different companies) is shown in Table 1. This number is within the expected range of 16 to 24 interviews for saturation in interview studies [23].

Interview Design. Following the concept of intensive interviewing [11], we created an interview guide with broad, open-ended questions to facilitate dynamic and detailed discussions.

1) Background & Project Context: First, asking general questions about the participant, i.e, the role in the job, experience with REST, and the project context. *2) REST API Usability:* Second, about the developers' perception of what makes up a well-usable REST API (RQ1). *3) Experiences with REST Guidelines:* Only thereafter, to not bias the developers in their perception of REST API usability, we asked them about their experiences with REST guidelines to complete the collection of background information. *4) Guideline adoption:* We continued with questions on their experiences in using guidelines and asking about challenges in everyday use (RQ2). *5) Creating guidelines:* Lastly, we asked about experiences in tailoring existing guidelines or creating new ones (RQ3).

These broad questions were designed to leave the interviewer room for detailed follow-up questions and allowing to steer the interview into directions that have not been uncovered in previous interviews [11, 29]. This design aligns with qualitative research principles, where not every participant needs to answer the same questions [34]. Instead, the interviewer adapts follow-up questions based on the participant's responses and emerging themes, allowing for flexibility and depth in exploration. Our appendix [46] provides the interview guide and the questions asked in each interview. Due to privacy concerns and the confidential nature of the interviewees' and companies' information, transcripts cannot be released.

Interview Execution. Each interview was conducted in German and lasted an average of 30 minutes. Earlier interviews tended to be longer, while later ones tended to be shorter due to response saturation. Before the main interviews, a trial interview was executed to refine the interview guide, test clarity and relevance of the questions, and ensure a smooth interview process [11, 29]. After each interview, its recording was transcribed and analyzed before the next interview to account for new factors. Analysis followed a rigorous coding process using MAXQDA [61]. First, the 1st, 2nd, and 3rd authors of this paper individually performed open coding to extract all relevant aspects [6], with the 1st author validating all codes after coding. In the next step, these code proposals were then aggregated and refined by the 1st and 4th authors based on collaborative discussions between the two authors [13, 65]. Through multiple iterations, the two authors identified key aspects, refined them, and validated them with supporting interview snippets, ensuring a thorough and transparent analysis capturing nuanced insights.

Ethnography and Context. To contextualize this study, we captured the interviewees' ethnographies and their work with REST APIs based on the responses in interview parts 1) and 3). We also used these dimensions to identify differences in the perception of senior and junior developers, consumers and providers of APIs, and consultants and developers from their company customers.

Table 1: Overview of the Interviewees

ID	Exp.[1]	Role(s)	Relation to REST	Focus[2]
I_1	8	developer, architect	provider	consultant
I_2	-[3]	consultant	provider	consultant
I_3	2	data scientist	provider	consultant
I_4	4	developer, architect	provider	consultant
I_5	3	developer	consumer	consultant
I_6	10	architect	consumer & provider	consultant
I_7	2	developer	consumer & provider	consultant
I_8	1.5	architect	consumer & provider	consultant
I_9	5	developer, consultant	consumer	consultant
I_{10}	10	consultant, guideline author	provider	consultant
I_{11}	8	architect, guideline author	provider	company
I_{12}	20	manager	provider	company
I_{13}	12	architect	provider	company
I_{14}	5	developer	provider	company
I_{15}	-[3]	developer, architect	provider	company
I_{16}	0.5	developer	provider	company

[1] experience with REST in years; [2] consultant: works for multiple companies (broad domain experience) / company: works for one company (focused domain experience); [3] experience exists, but was not quantified by the participant

As shown in Table 1, the interviewees are primarily developers and software architects. The majority has 5–10 years of experience with REST APIs, five less than five years, and two more than 10 years, while two did not specify their experience with REST. Most use or develop REST APIs. In addition, four interviewees manage or sell API endpoints without developing them, two are migrating legacy systems to REST, and one is using REST in process automation, internal training, and providing support services and software for API management. Two interviewees did not specify their context for using REST. Half of the interviewees are familiar with popular REST API guidelines, i.e., of Zalando [67] or Microsoft [33]. One even mentioned having used about 15 different guidelines. In addition, three interviewees only knew their internal guidelines, three knew general best practices, and two did not know any guidelines.

5 Usability of REST APIs (RQ1)

To better understand the usability of REST APIs, we first surveyed the interviewees for their perspective on what makes a usable API, and then how to create usable REST APIs.

5.1 Factors Influencing REST API Usability

To obtain a first picture, we asked the interviewees about their general understanding of REST API usability and what impacts it. We identified 8 factors that influence REST API usability (see Fig. 2).

Adherence to Conventions. For the majority (9 out of 16 interviewees), it is essential that usable REST APIs build upon existing conventions (I_1, $I_{4\text{-}5}$, $I_{7\text{-}8}$, I_{10}, $I_{12\text{-}13}$, I_{15}), i.e., that *"conventions are used wherever possible"* (I_4). Particularly, hard to use APIs are often *"not a structured API but simply a web service that also uses JSON but is not defined according to the REST pattern"* (I_8). Four interviewees emphasized the resource-orientation of REST (I_1, $I_{12\text{-}13}$, I_{15}), stating that *"R [in REST] stands for resource"* (I_1) and to expect *"smaller services that work on a resource basis and do not deliver everything (...), but that I can work on a resource basis (...)"* (I_{15}). In particular, I_1 states that HTTP and JSON are mandatory—although not specified in REST, using HTTP and JSON is the most common practice [40].

Documentation. A frequently mentioned factor is the clear and comprehensive documentation of REST APIs ($I_{1\text{-}3}$, I_5, $I_{7\text{-}8}$, $I_{11\text{-}12}$), notably mentioned by less experienced interviewees. Among others,

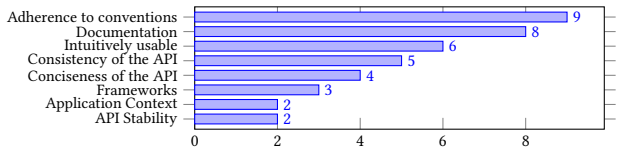


Figure 2: Factors influencing the usability of a REST API

formulated as an obligatory requirement: *"It needs to be documented so that someone can use it"* (I_1). The documentation should consist of resources being provided in a clear and convenient way (e.g., not nesting resource paths). To this end, I_8 stated that *"it has always annoyed me when APIs are poorly documented. There are enough of them, then you get a Word document thrown over the fence and then it's not a structured API"*. Ideally, as mentioned by five interviewees ($I_{1\text{-}3}$, I_8, I_{12}), documentation uses Swagger/OpenAPI, i.e., *"maybe also OpenAPI, there are also documentation tools like Swagger, (...), provide visual documentation for consumers"* (I_3).

Intuitively usable. Intuitively usable REST APIs are a major concern of six interviewees (I_1, I_4, I_7, I_{11}, $I_{15\text{-}16}$), primarily providers of REST APIs. While $I_{4\text{-}5}$ and $I_{15\text{-}16}$ phrase it as being straight-forward or easy to use, i.e., *"I want it to be easy to use"* (I_{15}), further two interviewees ($I_{4\text{-}5}$) explicitly mention intuition, i.e., *"that everything is as intuitive as possible"* (I_4). Five interviewees (I_5, I_7, I_{11}, $I_{15\text{-}16}$) also stated that an API should be self-explanatory and request that *"with common sense I can find my way around"* (I_7), that *"I can look at them [the APIs] and understand them directly without having to read through documentation"* (I_{16}), or *"that the interface is clear and simple, and understandable (...)"* (I_{15}). The optimal usage of a REST API should be apparent from its design, particularly *"that I can discover them without documentation. (...) when I access a resource, I should see links to things that I can do or query"* (I_{11}). Finally, I_1 focused on APIs and data formats, emphasizing that they should be *"human-readable, i.e., a REST API with XML or JSON (...)."*

Consistency of the API. Particularly consultants (I_2, $I_{4\text{-}5}$, I_7), but also one customer (I_{12}), emphasized that different APIs and single service API endpoints should be written in a consistent manner. I_5 highlights to *"simply get difficulties when using it [an API], because I have the feeling that it is not consistent."* Particularly, there should be *"uniform rules for resource naming"* (I_2) and that if *"I have an endpoint somewhere that I can use, and I get an endpoint [in the response of the latter one] that is used in the same way"* (I_7).

Conciseness of the API. Four interviewees (I_1, I_6, $I_{14\text{-}15}$) emphasized API conciseness in terms of separation of services, i.e., *"to ensure separation of systems in my REST API"* (I_1). To that end, each API endpoint should have a precise and well-focused purpose, because *"what often makes it [a REST API] more usable now is that teams have been exposed to how to slice and dice them, and the focus has been sharpened to make APIs smaller"* (I_{14}).

Frameworks. Three interviewees (I_1, I_3, I_{10}) emphasized the benefits of dedicated tools supporting the design of REST APIs, since *"REST is very native in that it can be connected to tools."* (I_3) To this end, all three interviewees explicitly mentioned OpenAPI and noted that related REST API frameworks come with several features to enhance the usability of APIs, e.g., *"documentation tools like Swagger (...) provide visual documentation for consumers"* (I_3).

Application Context. I_4 and I_5 emphasized that REST API usability depends on the usage context of the API. For example, APIs can be consumer- or developer-oriented, and therefore, it *"depends a bit on the context how simple or complicated an API is"* (I_5).

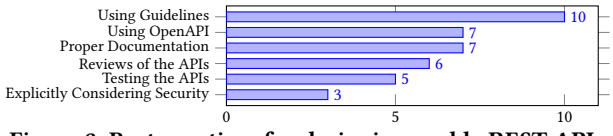


Figure 3: Best practices for designing usable REST APIs

API Stability. Finally, I_6 and I_{12} stated that APIs should be stable, expecting *"stability from the API even if the application behind it may change"* (I_6)—one of the key concepts of REST [18]. While not explicitly mentioned in the interviews, standards such as semantic versioning [49] can help to clearly indicate braking changes.

5.2 Best Practices for Usable REST APIs

Having identified the factors that influence usability, we asked how to support the creation of usable REST APIs (see Fig. 3).

Guidelines. With 10 out of 16 interviewees, the majority stated some form of guidelines as a best practice to help developers design usable APIs. However, only I_4, $I_{7\text{-}8}$, I_{12}, I_{14}, and I_{16} explicitly named guidelines, while I_3, I_5, and I_{11}, I_{13} stated that *"there are enough standards that I wanted to adhere to"* (I_3), i.e., recommendations such as *"how are URLs structured (...)"* (I_{13}).

OpenAPI. The systematic API specification using OpenAPI is an essential best practice for six consultants ($I_{1\text{-}4}$, I_6, I_9) and one customer (I_{12}). I_1 stated that in *"Swagger or in newer OpenAPI 3.0, I find excellent support"*. Among others, allowing them *"to generate the specification immediately during coding"* (I_6), which can then be given to consumers. This is seen as useful because *"when I have an OpenAPI specification (...) at hand, I can generate my client"* (I_9). It also serves as public documentation, so an OpenAPI specification *"is something that one should have (...)"* (I_{12}).

Documentation. Seven interviewees ($I_{5\text{-}7}$, $I_{11\text{-}12}$, $I_{14\text{-}15}$) consider proper REST API documentation a best practice, but only I_6 and I_{12} named OpenAPI as an essential means to generate documentation. The rest (I_5, I_7, I_{11}, $I_{14\text{-}15}$) states that one should create a *"good and simple documentation"* (I_5), not specifically mentioning any technology. That REST APIs are documented is assumed, i.e., in statements like *"he will probably have provided documentation on this"* (I_7).

Reviews. Six interviewees (I_2, I_{10}, $I_{12\text{-}13}$, $I_{15\text{-}16}$), five of them experienced customers, see reviews as important to improve quality aspects such as usability, i.e., to *"look over the API design (...) to build uniformity in the company on how REST APIs are built"* (I_{13}). For three of them, adherence to guidelines is an essential aspect of the reviews. It is essential to give developers feedback, e.g., by *"noting any irregularities [related to guidelines] and then make an appointment with the developers"* (I_{16}).

Tests. The testing of the REST APIs is described by five interviewees as a necessary step in the development process (I_1, I_4, I_7, I_{12}, I_{16}), i.e., *"the entire API has to be tested"* (I_1). Thereby, it is essential to have different test levels, among others *"unit and integration tests"* (I_{12}).

Security. Finally, three interviewees ($I_{1\text{-}2}$, I_{15}) emphasized the need to explicitly consider API security, stating that *"security must be ensured"* (I_1). *"Uniform security in the application landscape"* (I_2) is needed with respect to usability. Explicitly considering security in API guidelines is crucial, as developers are typically not security experts and require additional security guidance [24].

The main concerns on API usability are adherence to conventions and proper API documentation. APIs should be designed intuitively usable, consistent, and concise. Guidelines are seen as essential for designing usable APIs. Specifically, OpenAPI should be used to specify APIs, also since it allows to generate documentation and client code. While consultants focus more on such technical solutions, customers seem to be more concerned with processes.

RQ1: REST API Usability

6 Adoption of REST API Guidelines (RQ2)

Institutions have created various REST API guidelines, which (as confirmed above) are an essential factor in implementing usable REST APIs. Given that REST API usability remains suboptimal, we examined how guidelines are used, what factors influence their successful adoption, and best practices for guideline adoption.

6.1 REST API Guideline Adoption in Practice

First, we assessed the current state of guideline adoption. Fourteen of the 16 interviewees have used internal or public guidelines.

Provision of Guidelines. Half of the interviewees (I_2, I_6, I_8, $I_{12\text{-}16}$) access guidelines through internal wikis, i.e., Confluence, while interviewees (I_1, $I_{4\text{-}5}$, I_{12}, I_{16}) use a website. I_1, I_3, and I_5 (also) use PDFs, which according to one interviewee are more like cheat sheets. I_3 notes that interactive media, such as websites, is better than static media such as PDFs. I_5 stated that they use the website of an external guideline and the media provided there.

Adherence to Guidelines. Eleven interviewees ($I_{1\text{-}4}$, $I_{6\text{-}7}$, $I_{11\text{-}15}$), four of them mentioned guidelines as best practice, indicated that guidelines are partially avoided or rarely used. $I_{1\text{-}2}$ and I_8 do not utilize guidelines for release, but document their functionality via Swagger/OpenAPI, while one states usage of guidelines as templates when publishing APIs. One interviewee complained about a lack of automation for guidelines and respective checks. Another interviewee reports to use guidelines for release documentation and specification, when deploying new APIs.

Enforcement. The subliminal wish in most interviews ($I_{1\text{-}2}$, $I_{4\text{-}8}$, $I_{10\text{-}12}$, $I_{14\text{-}15}$) was that developers should not be forced to follow a guideline, i.e., *"you should not force it on people, but you (...) have to teach people"* (I_{10}). I_8, I_{10} and I_{12} wish for training or support for questions and problems. In contrast, I_{12} stated that "*the more specific it [a guideline] is, the more likely it is to be violated and if we do not force it, then it is kind of pointless.*" Finally, one interviewee wishes for an external controlling authority. While these are more ideas and wishes, they reflect the state reported by four interviewees (I_2, $I_{13\text{-}15}$) that guidelines are generally poorly enforced. $I_{3\text{-}5}$ and I_{15} stated that their guideline cannot be enforced all the time because it contains too many or specific rules. One interviewee regularly checks APIs and applications for compliance. Three interviewees (I_3, I_5, I_8) only looked at guidelines from a catalog once, i.e., used it to implement a new API or learn about internal standards. After that, the guidelines were intuitive enough to remembered them.

6.2 Factors for Successful Guideline Adoption

While adherence to guidelines improves qualities such as understandability [5], as I_{15} stated, good guidelines are useless if unused.

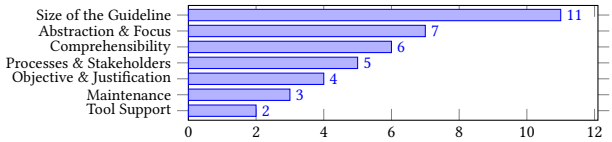


Figure 4: Factors influencing REST API guideline adoption

To determine the reasons for the observed opposition to strict guidelines, we identify factors that influence the successful adoption of and adherence to REST API guidelines (see Fig. 4).

Guideline Size. For the majority (11 of 16 interviewees (I_1, $I_{3\text{-}6}$, $I_{11\text{-}16}$)), including all customers, guideline size is the biggest concern in adopting it, stating that *"the less text you have the higher the chance of compliance"* (I_6). According to them, *"the size of the guideline is the biggest challenge. These are guidelines that are extremely large, and you have to work through them to find out what is relevant for you."* (I_1), For example, *"a known guideline is the one of Zalando (...) They have, (...) 120 to 140 rules, but such a bunch is hard to remember and comply with"* (I_5). Almost all interviewees who were aware of popular guidelines (I_1, $I_{3\text{-}5}$, $I_{12\text{-}14}$, I_{16}) noted that they are too extensive, so they use smaller ones in their organization. When asked about the size of known guidelines, seven interviewees (I_1, I_3, I_6, $I_{11\text{-}12}$, $I_{14\text{-}15}$) stated that a guideline needs to be concise, and thus be fast digestible by developers, i.e., one interviewee prefers a two page document. However, according to four interviewees ($I_{1\text{-}2}$, I_6, I_{10}), guideline size is also determined by the technical depth and detail included. When asked about appropriate size, seven interviewees (I_1, I_3, I_6, I_{10}, $I_{12\text{-}13}$, I_{16}) consider a minimal guideline as adequate in an internal or self-contained environment, as opposed to larger public guidelines. In practice, there is no fixed size limit above which guidelines are ignored, but it depends on the context of the REST API, i.e., *"I try to evade the big guidelines when writing my APIs, but if I create publicly available ones, I have no other (...)"* (I_1).

Abstraction & Focus. Seven interviewees ($I_{1\text{-}3}$, $I_{5\text{-}6}$, I_{10}, I_{16}) described a good guideline as focused on implementable rules, while too broad guidelines will not be read. Therefore, I_2 prefers selections of rules, because *"you have to see what situation you want to solve and if you need a rule for that."* Another interviewee states that *"you should concentrate on important rules and may continue to extend with lesser prioritized rules"* (I_5). One interviewee states that this is done through individual guidelines for each department, while another wants just one guideline. According to I_1 and I_5, bad guidelines combine rules that are adverse or unrelated to their context. Particularly, I_1 requests that *"my guideline should help me describe my interface and not contain too many details, which are irrelevant here."* Related to this, I_5 and I_{16} state, too many technical details or performance enhancements bog down a guideline, since *"the guideline may describe with an example of how to return data, but if I encounter more complicated situations, this example may not be applicable"* (I_{16}). Finally, one interviewee remarked that the level of detail or stage of development determines a guideline's acceptance.

Comprehensibility. Six interviewees (I_1, $I_{5\text{-}6}$, $I_{11\text{-}12}$, I_{15}) state, *"guidelines should be easily comprehensible"* (I_{11}). To this end, a guideline *"should be concise (...) you don't have much time to read through"* (I_{11}). Guidelines should include keywords for every rule, *"at best in headlines"* (I_{11}). However, according to I_4, guidelines that use buzzwords for explanations without proper sentences, or clear prioritization

of rules (another interviewee), are less usable. Six consultants ($I_{1\text{-}2}$, I_4, $I_{5\text{-}6}$, I_9) emphasize to support rules by explanations, examples and providing technical depth to be informative (I_1, I_6).

Processes & Stakeholders. For five of six customers (I_{10}, $I_{13\text{-}16}$), the processes within an institution and involved stakeholders are an essential factor for the successful adoption. I_{15} emphasized the challenge of integrating guidelines into workflows, and another emphasized the challenge of choosing an appropriate guideline. Particularly concerning stakeholders, *"acceptance is most important. I can write wonderful things in my guideline, but if developers don't accept it, it's useless. I have to connect to the people, consider their different skill levels, and create consensus of the problem"* (I_{15}). According to four interviewees (I_2, $I_{13\text{-}14}$, I_{16}), one must consider that *"everyone has their own level of knowledge, you can definitely see, more with older developers, that they develop APIs the same way they did before with SOAP"* (I_{16}). Particularly, *"there is always a problem of old thought patterns—after working for 30 years, knowing your data models and functions, one may not see the problem, that others don't understand them in the same way"* (I_{14}). It is essential to maintain a community with appropriate culture, since it is *"problem of willingness to learn. Developers who are interested are taking part in meet-ups and do look up rules in forums while others do not."* (I_{13}) I_{10} and I_{13} emphasize the need to enforce guidelines in some way, especially to senior developers who tend to be reserved. Focusing on processes, I_{10} states *"If you want to create APIs according to guidelines, you need to make sure, that all developers are able to"*. One interview emphasized the appropriate choice for the guideline medium as factor.

Objective & Justification. According to I_1, I_3, I_5, and I_{10}, guidelines must serve a clear purpose and *"explain why rules exist"* (I_{10}). To this end, plans and explanations for the aimed at maturity level (cf. Richardson Maturity Model [19]), i.e., which concepts of REST are targeted to be implemented, should be included. A guideline should not be self-purpose. It must enhance the whole workflow, in which enforcement is handled (I_6, I_{10}).

Maintenance. For three interviewees ($I_{1\text{-}2}$, I_7) it is essential that guidelines are up-to-date. One interviewee noted that regardless of the scope, guidelines must be kept up to date, and one interviewee emphasized that an outdated guideline is bad. Overall, the maintenance of old guidelines is an important factor that affects their practical adoption for three interviewees (I_1, I_7, I_{15}). In addition, one interviewee emphasizes that any examples given in a guideline should work. One interviewee suggests a REST API guideline should promote feedback and revisions from users and developers.

Tool Support. Two consultants remarked that a usable guideline provides tool support, or is at least machine-readable, i.e., OpenAPI (I_1, I_5). Guideline size matters less, if creators, such as Zalando, provide sufficient support (one interviewee) or machines and tools can provide feedback to developers (one interviewee). Lastly, one interviewee remarked that a guideline should also have human support.

6.3 Best Practices for Adopting Guidelines

We identified six best practices for adopting guidelines.

Community Work. When adopting guidelines, it is essential to convince all stakeholders of its benefit. It is essential *"to proactively promote guidelines"* (I_{12}) and *"to support teams that have not worked with it"* (I_{14}). The goal is to build *"a community on this topic"* (I_{15}). Instead of management strictly enforcing a guideline, it is essential *"to convince why this way [guideline rule] is the standard or simpler way"* (I_{13}). Courses, e.g., on API design, can help. Compliance may decline if developers observe peers disregarding guidelines, reflecting the broken window theory of technical debt [31].

Tools and Automation. Although continuous integration has automated many software development tasks, compliance with API guidelines is still primarily checked manually, i.e., by integration teams (I_{16}) or during certifications and code reviews (I_{12}). However, interviewees see automated checking for compliance with REST API guidelines as highly desirable, i.e., *"if you not only invest in your guideline but also automation, then developers wouldn't need to read and remember all rules"* (I_{10}). I_{11} reports positive experiences with automation—*"we had hundreds of microservices, and you just cannot have a process to check the APIs. That is why we developed an API linter to support the use of guidelines"* (I_{11}). This is in line with current trends, since *"typically today, teams are working on automating processes, mostly by automating reviews over time. Step by step more rules get checked against guidelines"* (I_{10}).

Examining the detection rules of static analyzers such as SonarQube [4, 58] reveals that REST API quality rules mostly assume the use of tools such as OpenAPI. Besides using OpenAPI being considered a best practice (I_1, I_5), this further encourages its use. However, this reveals also a gap in the current landscape of API linters: While tools excel at syntactic and schema validation [60], they struggle with usability and context-dependent guidelines that require human judgment [52]. Usability testing remains a manual or semi-automated process, essential for ensuring APIs are intuitive and user-friendly. In this context, REST API guidelines can be categorized into general principles (e.g., statelessness, caching), which are difficult to automate, and fine-grained rules (e.g., naming, schema validation), which are more amenable to automated checking [12].

While tool support can reduce cognitive load, it may also reduce commitment to guidelines, raising questions such as *"is there already something implemented in the build pipeline?"* (I_4). To address this, organizations should complement automation with educational scaffolding [50], such as interactive feedback, that reinforces the purpose of guidelines. For example, linters could not only flag violations but also explain the impact of non-compliance (e.g., "This naming convention improves discoverability for client developers"). A general problem with linters is them typically finding too many issues, requiring the identification of the most urgent ones [45]. Here, API issues compete with all other types of bugs and vulnerabilities.

Interactive Guidelines. A key recommendation for guideline provision is to utilize interactive media. Ten interviewees ($I_{1\text{-}2}$, I_6, I_8, $I_{11\text{-}16}$) effectively access guidelines through internal wikis, such as Confluence, i.e., *"our own guideline for our REST APIs, documented in a wiki format"* (I_2). Interactive media is seen as superior to static documents, providing an engaging, user-friendly experience. In particular, interactive navigation *"where I can jump back and forth and search better"* (I_1) is important. It enhances accessibility and usability, facilitating better compliance with guidelines. Thereby, priority keywords for every rule in the guideline can provide a quick fist overview, since *"in everyday use, you don't have much time to read through"* (I_{11}). Using priority keywords, as in the Zalando guideline [67], helps users understand and adhere to guidelines.

Appropriate Rule Selection. All interviewees agreed on have an appropriate guideline for the organization is most important. To ensure optimal results, it is recommended to adopt a concise guideline. The interviewees stressed that the size of a guideline is the biggest concern. Recall that I_6 noted, that *"the less text you have, the higher the chance of compliance"*. Consequently, organizations should use smaller guidelines internally to enhance adherence. A best practice for ensuring adherence to guidelines is to design them to be intuitive and focus on organization relevant rules.

> *The biggest challenge in adopting guidelines is the acceptance by developers, since, as I_5 summarized, guidelines are extra work for developers. The primary guideline-specific factors impacting their adoption are their size and closely related to this their provided abstraction and focus. The best way to ensure adoption is to convince developers of the benefits. This suggests that institutional culture and processes are key factors in improving adoption. Tooling can particularly help embed guidelines into development processes and automate compliance checking to some extent.*
>
> RQ2: Adoption of REST Guidelines

7 Creating Custom Guidelines (RQ3)

The factors for successful adoption of REST API guidelines indicate multiple organization-specific factors and that public guidelines are often too broad. Thus, we inspected the motivations, challenges, and best practices behind custom guidelines.

7.1 Motivations for Custom Guidelines

To understand the motivations for custom guidelines in the presence of public guidelines, we further asked for reasons.

Minimize Size. As the above discussion shows, guideline size is a major factor in successful adoption, and public guidelines are seen as too big. Therefore, a main motivation for custom guidelines is to minimize size to increase efficiency and effectiveness. Still, public guidelines are often considered, but they want to *"delete what is not relevant"* (I_1) to *"keep it simple"* (I_{12}). Consequently, organizations often create *"a slimmed down version"* (I_1) of guidelines. The factors below play an essential role in determining which rules to include.

Domain-specific Context. A custom guideline is described as necessary by five interviewees (I_1, $I_{6\text{-}7}$, I_{10}, I_{13}), because it must fit company and industry context. One has to consider that each *"interface is unique. Therefore, my guideline should reflect all relevant aspects"* (I_1). In particular, rules of public guidelines do not apply to every context, so organizations want to remove them since *"irrelevant rules reduce acceptance"* (I_6). Context dependent, organizations often *"notice that there are some aspects that one does not have to treat (...)"* (I_{13}) in their application context, allowing to reduce the size while increasing the chances of compliance. Additionally, I_1, I_3 and I_4 mention the concrete technologies used as context, especially when changed, the guideline needs to be tailored to stay relevant.

Organizational Priorities. Seven interviewees (I_1, I_4, I_6, $I_{11\text{-}12}$, $I_{14\text{-}15}$) noted that organizational priorities shape API guidelines. In the end, *"this is a matter of taste, with our [the organization] own flavor"* (I_{12}). Organizations want to reflect custom API styles that stem from their custom priorities and unique selling points. Particularly, a guideline should reflect the organizations priorities, i.e., *"we then narrowed it down [the Zalando guideline] to 10 points and described them using [organizational] wording and limited ourselves to it"* (I_{15}).

Organizational Processes. Another essential motivation for customization is that a guideline has to *"fit processes"* (I_{10}) of the adopting organization. One interviewee states that the Zalando guideline is not flexible enough and another that guidelines need an appropriate *"balance between governance and team autonomy"* (I_{12}). How a guideline reflects this is specific to each organization and therefore requires customization. To increase autonomy, organizations customize guidelines by deciding what to regulate, i.e., they *"agreed on that [guideline rules] and left the rest to the particular situation"* (I_7).

Reflect Discussions. An essential factor for a successful guideline is the *"mutual agreement on what is good"* (I_{10}). Often there are *"different possible solutions"* (I_{10}), but only *"one must be selected"* (I_{10}). Selection is often the result of internal discussions that *"involve developers"* (I_6). E.g., *"when you face a problem or question (...) we search for a solution in the team"* (I_{16}). The guidelines should document the outcome to support future decisions, therefore guidelines should *"reflect discussions"* (I_2), automatically leading to customization. Furthermore, there is often *"feedback of development teams"* (I_{11}) that should be addressed in the guideline, i.e., on what to describe better or how well rules can be realized in practice. Particularly, customization allows to *"address that rules are not followed"* (I_{14}).

Control over Changes. Finally, I_2, I_{12}, and I_{16} indicate that having control over changes can be a motivation. Firstly, organizations want to keep their guidelines up to date, because *"you cannot just develop according to the same guidelines for ten years. That is why you always have to see what the current state of science and practice is"* (I_{16}). Still, organizations do not want to rely on external guidelines being maintained. Secondly, they need stable guidelines and want to limit changes, where guideline size is a factor to consider, *"because it is manageable and stable due to the few rules"* (I_{12}).

7.2 Guideline Creation in Practice

Of the 16 interviewees, seven helped create a single guideline, one contributed to several guidelines, six had minimal involvement, and two were not involved at all. Thus, we asked the 14 involved interviewees how guidelines are created in their organizations.

Reference Guidelines. When asked why they create a custom guideline instead of using an existing public guideline, I_1, I_4, and I_{10} agreed that *"people already thought about it and in the best case, I don't have to figure it out myself"* (I_4). Therefore, the creation itself is based on other guidelines utilized as templates (I_1, I_{10}, I_{15}), acquired through web research or other companies. Four interviewees raised criticism in the way how custom guidelines are created (I_1, I_5, I_7, I_{11}), i.e., that *"it is a bad process to create your guideline by randomly copying and crafting together a wild bunch of rules, since they can contradict one another"* (I_5). Existing guidelines should not just be copied, but definitely help by giving inspiration. Some rules may be copied if they seem helpful, but rules can conflict with other rules. To the point of inspiration, one statement was that *"for some time the Zalando guideline was (...) a baseline for tailoring an individual guideline, but lastly I heard that this trend shifted to the Google guideline"* (I_{10}).

Contributors. While four interviewees stated that software architects, quality assurance, and developers of the API ($I_{1\text{-}2}$, I_6, I_{11}) are involved in creating guidelines, three interviewees (I_1, I_{12}, I_{14})

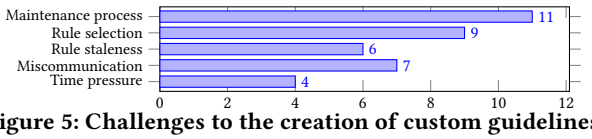


Figure 5: Challenges to the creation of custom guidelines

stated to ensure that everyone in the project is integrated. Other stakeholders involved in guideline creation include technical leads as stated by I_1 and I_{11}. In addition, single interviewees indicated to involve also API experts, security experts, API consumers, the distribution team, and team leads. One interviewee remarked that any guideline creation or modification is done by the core team itself. I_5 and I_{14} mentioned that any changes made to the guideline need to be approved by the developers before changes are made.

Data Collection Process. Regarding the process of creating guidelines, I_2, I_6, I_{10}, and I_{12} refer to their agile work environment, which ensures a feedback loop with the participants. Interviewees I_2, I_{11} and I_{15} mention that they first acquire knowledge through research, and after gathering some guidelines, modify them according to their needs. I_{12} reported they have an explicit *"RFC process [Request For Comments], so if someone has a request, you can put it in the wiki"*.

Communication. When guidelines change, 9 out of 15 interviewees actively notify users of their guideline or API (I_1, $I_{5\text{-}7}$, $I_{12\text{-}16}$). To this end, they use Blog posts, emails, or internal communication channels, e.g. *"we have an announcement mechanism that uploads to our website (...) We also tried Teams channels (...) and serial emails"* (I_6). I_1 and I_{11} reported that emails work well. I_1 uses a personal channel to *"inform them [the guideline consumers] actively through email, about changes and when they apply"* (I_1) and I_{11} uses *"newsletters to announce changes"* (I_{11}), particularly stating that this works well. One conclusion was to *"always over-communicate over several channels"* (I_{11}). Lastly, two interviewees (I_7, I_{11}) communicate any alterations via tool support, either through REST endpoints or linters. In contrast, the guideline version is silently updated by four participants (I_1, I_{13}, $I_{15\text{-}16}$). I_4 and I_{16} do not communicate changes.

Continuous Maintenance. Three interviewees (I_1, $I_{5\text{-}6}$) conclude that a guideline is never finished, e.g., due to the need for addition and revision of new rules. In this regard, two I_1 and I_5 emphasized that guideline development needs to be actively terminated, either by stopping or completing the corresponding API development. I_6 stressed the importance of maintaining a guideline throughout its life-cycle. Particularly, guidelines reflect the decisions in the project, and therefore, organizations usually *"start with a small set [of rules] that everyone can quickly agree on and then see how well they work and gradually expand them. That's why they are never finished."* (I_6)

7.3 Challenges

To better understand why current practices of tailoring guidelines to the organization to increase adoption are not resulting in optimal compliance, we asked the interviewees about the challenges of creating custom REST API guidelines (see Fig. 5).

Maintenance process. The maintenance process is seen as a challenge by most interviewees (11 of 14 interviewees that create guidelines: $I_{1\text{-}2}$, $I_{5\text{-}6}$, I_{10}, $I_{11\text{-}13}$, $I_{15\text{-}16}$). Since a guideline cannot include everything, two interviewees (I_7, I_{11}) mention that surprising or new situations need be included after occurence. $I_{10\text{-}11}$ pointed out that the alteration process itself poses challenges, specifically regarding the consensus and implementation. Backwards compatibility has to be ensured, and no breaking changes should be made.

Rule selection. Rule selection is seen as a challenge by nine interviewees (I_1, $I_{5\text{-}7}$, $I_{10\text{-}11}$, $I_{14\text{-}16}$). Three interviewees (I_1, I_7, I_{10}) stated that it is important but challenging to keep and provide the context of the API, particularly in guidelines with manageable size. Recall, there is considerable resistance to reading through long texts; however, it is a challenge to select a set of appropriate rules to include in the guideline. In relation to this, two interviewees mentioned, that preventing inconsistencies is challenging (I_5, I_{11}). Also, deciding on an appropriate level of abstraction and including good and appropriate examples is challenging. This is further challenged by the circumstances that *"there are many points where there is simply no clear best solution (...)"* (I_5). This usually leads to discussions of which solution to adopt in an organization., i.e., adding it to the guideline.

Rule Staleness. Six interviewees ($I_{1\text{-}2}$, I_7, I_{12}, $I_{15\text{-}16}$) see keeping rules up to date as a challenge. Guideline initiatives can fail from one stale rule alone, considering that *"if a developer encounters outdated guideline rules, he or she will not read through the rest, thinking those rules will be outdated too"* (I_{15}). However, guidelines do not evolve as fast as the technology stacks involved, i.e., I_{12} stated that he has not *"seen substantial changes to the guideline in the last three years. But there are other guidelines where it is different, especially, testing strategies where discussions and changes occur more frequently"*.

Miscommunication. While seen as essential, seven interviewees identified good communication of a guideline and its changes as a significant challenge (I_1, $I_{5\text{-}6}$, $I_{10\text{-}11}$, $I_{15\text{-}16}$). Organizations must avoid situations in which *"someone uses the old guideline"* (I_6), but providing a sensible versioning scheme is major challenge for three participants. The involved communication is dependent on the organization, especially size, and what software or method reaches most people. While providing active notice of new guidelines or changes, two interviewees (I_6, I_{11}) note that used channels tend to be spammed and notifications are lost. For example, I_6 reports that they *"have an announcement mechanism that uploads to our website, but we want to change that. There is always someone who does not see it. We also tried channels or emails with the same result and it developed a spam characteristic"*.

Time pressure. According to $I_{5\text{-}7}$ and I_{11} one factor is the time-consuming nature of creating guidelines. When asked about workload or costs, I_{11} answered that *"especially in the start-up field, you need to think about time-to-market. (...) If there is no time to create guidelines, you just copy other guidelines like Zalando."* This challenge was also phrased as an investment decision, stating that *"creating a guideline is unpleasant work (...). Agreeing on the same language costs time initially, but you become faster in the future"* (I_7).

7.4 Best Practices

From the answers presented and further in-depth questions, we derived best practices for custom guidelines that improve REST APIs and facilitate compliance while keeping costs low.

Start Small. Given the size of many guidelines and the difficulty of grasping and remembering all rules, starting with a smaller or focused subset is beneficial. It is important because *"those responsible have to read and internalize the guidelines first. Depending on the length of the document, this is a major criterion"* (I_5). In the end, the

interviewees' experience that *"if you start with a small guideline, you will have good acceptance because the developers are directly involved and if there are specific questions, you can go into more detail"* (I_{15}). This allows to reflect situations and challenges observed during REST API development and keeps rules relevant. Since the interviewees emphasized the consistency of REST APIs as an important quality aspect, although not explicitly stated by them, this has to be considered in the rule selection. While improving only changed code is best practice [17], ensuring API consistency may require refactoring the entire REST API when adding a new rule.

Collective Ownership. Involving users in the guideline creation prior to its widespread adoption is seen as an best practice to improve guideline acceptance (I_2, I_6, I_{12}, I_{13}, and I_{14}). A key benefit is that *"with involvement we get the advantage to say, you did accept this, if you do not comply we need to change it. So we are not seen as the team that dictates rules."* (I_{14}). It also allows to deal with the challenge that *"not everyone agrees on guideline rules, just because we implemented them. We are not total experts and developers have their own knowledge on the topic"* (I_{16}). Therefore, organizations should involve developers to leverage existing knowledge and as they have to commit to the guideline afterward. As a positive example, I_6 particularly pointed out that *"in the past, we organized the guidelines centrally. Now, we try building it with a community approach and integrate people earlier."* While the approach to integration differs due to the development teams' sizes, the basic idea is to give guideline users the opportunity to participate. Changes to the guideline can be suggested or required by different sources, i.e., new situations can be encountered or developers suggest changes based on newly gathered experience. When asked why and when guideline rules have to be changed, I_{14} stated that *"when rules are not complied with, there is reason to talk about. The legal framework can change, and we need to see how it affects us. Or simply someone mentions the need for change."* In line with the interviewees' perception, collective ownership has effective in improving code quality [21]. However, different factors impact effectiveness of collective software artifact ownership [26], which may have to be considered also for REST API guidelines.

> *Guidelines are typically customized by organizations to tailor them to their needs, trying to increase developer compliance. The main challenge is to actively maintain the guideline to keep it relevant. To scope a guideline it is essential to identify the rules that should be added to the guideline—at an appropriate level of abstraction for the targeted stakeholders, while having limited resources. We identified the iterative growth of guidelines combined with the active involvement of all stakeholders as best practices to keep guidelines focused and embedded in the developers minds.*
>
> RQ3: Creating and Tailoring REST API Guidelines

8 Threats to Validity

Internal Validity. The validity of the interviews could be threatened by factors, such as subject expectancy bias, observer bias, and selection bias. Especially, subject expectancy bias could have affected the interviews, since the interviewer was known among some interviewees. The interviewer itself had significant experience in working with REST APIs and guidelines, among others from working for the organization from which the interviewees were recruited. While this provides him with necessary background, this could affect interviews due to an internal bias. To mitigate this threat, we involved further authors that are not related to the organization in the analysis of the interview transcripts. It is important to note that both the questions and answers were translated, since the interviews were conducted in German. This may affect the internal validity due to potential bias in the exact translation of terms. However, as the full context was always considered, the potential impact is minimal.

External Validity. A threat to the external validity is the limited number of interviewees due to focusing on a smaller group of experts instead of a broader group of general developers. Nevertheless, saturation was reached in later interviews, confirming all key challenges and practices. Another potential threat is the large number of participants who work as consultants for the same company. Yet, these consultants bring diverse experiences from various clients and do not share REST APIs or guidelines, minimizing bias. As shown above, the individual experience of developers can influence how they view guidelines. Although saturation was clearly evident in the data analysis, the details of challenges may be different in other organizations. Cultural and contextual parameters are likely to play a role. However, the general challenges and best practices identified are unlikely to change.

9 Conclusion

REST APIs are core business assets, yet their usability often impedes their usage. While API guidelines aim to improve usability, their practical application and effectiveness remain understudied. To fill this gap, our study examines the factors influencing successful adoption and organizational tailoring of REST API guidelines.

While we found that API usability depends on consistent conventions, intuitive design, and thorough documentation, we also learned that REST API guideline adoption faces developer resistance, as extra rules are often perceived as burdensome. Successful adoption thus requires demonstrating clear benefits, tailoring guideline scope and abstraction, and embedding compliance into workflows through supportive culture and tooling. We found that REST API guidelines are typically tailored to specific organizations to reduce complexity and improve relevance, but maintaining their usefulness demands active upkeep, stakeholder collaboration, and iterative refinement to balance practicality and resource limits—often complicated by conflicts between valid but competing practices. Shared ownership and incremental growth emerged as best practices to keep guidelines focused, practical, and integrated into developers' daily work. Linting, in particular, offers opportunities not only for enforcing compliance but also for educating developers.

Our results give rise to the following research directions. First, advancing automated guideline compliance checks is crucial, as it appears to be the only scalable strategy for governing numerous, evolving rules across a growing number of API endpoints in a team. Additionally, while we considered REST API guidelines holistically, analyzing individual rules and their taxonomy could benefit both research and practice to address the problem of limited cognitive capacity. Finally, recall that guidelines are often very specific to organizations. While work that analyze REST API guidelines and categorize their rules already exists [36], further work is needed for a unified catalog of universally valid rules.

References

[1] Abdulghani, H. A., Nijdam, N. A., Collen, A., and Konstantas, D. A Study on Security and Privacy Guidelines, Countermeasures, Threats: IoT Data at Rest Perspective. *Symmetry 11*, 6 (2019).

[2] Adams, W. *Handbook of Practical Program Evaluation*. Wiley, 2015, ch. Conducting Semi-Structured Interviews.

[3] Alarcón, R., Wilde, E., and Bellido, J. Hypermedia-Driven RESTful Service Composition. In *International Conference on Service-Oriented Computing* (2010).

[4] ApiAddicts. Sonar OpenAPI. https://github.com/apiaddicts/sonar-openapi, 2024.

[5] Bogner, J., Kotstein, S., and Pfaff, T. Do RESTful API Design Rules Have an Impact on the Understandability of Web APIs? *Empirical Software Engineering (EMSE) 28*, 6 (2023), 132.

[6] Boyatzis, R. E. *Transforming Qualitative Information*. Sage, 1998.

[7] Buelthoff, F., and Maleshkova, M. RESTful or RESTless – Current State of Today's Top Web APIs. In *European Semantic Web Conference (ESWC)* (2014).

[8] Carzaniga, A., Picco, G. P., and Vigna, G. Designing Distributed Applications With Mobile Code Paradigms. In *International Conference on Software Engineering (ICSE)* (1997).

[9] Carzaniga, A., Picco, G. P., and Vigna, G. Is Code Still Moving Around? Looking Back at a Decade of Code Mobility. In *International Conference on Software Engineering (ICSE)* (2007).

[10] CERT Coordination Center. SEI CERT Coding Standards. www.securecoding.cert.org, 2024.

[11] Charmaz, K. *Constructing Grounded Theory: A Practical Guide through Qualitative Analysis*. Sage, 2006.

[12] Coblenz, M., Guo, W., Voozhian, K., and Foster, J. S. A Qualitative Study of REST API Design and Specification Practices. In *Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (2023), IEEE, pp. 148–156.

[13] Corbin, J., and Strauss, A. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage publications, 2014.

[14] Cruz, M. Puplic APIs – A Collaborative List of Public APIs for Developers. https://github.com/public-apis-dev/public-apis, 2024.

[15] Daughtry, J., Macvean, A., and Church, L. The Uses of Interactive Explorers for Web APIs. In *Workshop on Evaluation and Usability of Programming Languages and Tools* (2017).

[16] Di Lauro, F., Serbout, S., and Pautasso, C. A Large-Scale Empirical Assessment of Web API Size Evolution. *Journal of Web Engineering 21*, 6 (2022).

[17] Digkas, G., Chatzigeorgiou, A., Ampatzoglou, A., and Avgeriou, P. Can Clean New Code Reduce Technical Debt Density? *Transactions on Software Engineering (TSE) 48*, 5 (2022).

[18] Fielding, R. T. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[19] Fowler, M. Richardson Maturity Model. https://martinfowler.com/articles/richardsonMaturityModel.html, 2010.

[20] Fraser, S., Gamma, E., Helm, R., and Johnson, R. E. Design Patterns: Beginnings and Futures. In *Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)* (2006).

[21] Greiler, M., Herzig, K., and Czerwonka, J. Code Ownership and Software Quality: A Replication Study. In *Working Conference on Mining Software Repositories (MSR)* (2015).

[22] Harrison-Adcock, K. API Roundup for 7 September 2022. *Programmable Web* (2022). Accessible via the Internet Archive: https://web.archive.org/web/20221002225348/https://www.programmableweb.com/news/api-roundup-7-september-2022/brief/2022/09/04.

[23] Hennink, M. M., Kaiser, B. N., and Marconi, V. C. Code Saturation Versus Meaning Saturation: How Many Interviews Are Enough? *Qualitative Health Research 24*, 4 (2016).

[24] Hermann, K., Peldszus, S., Steghöfer, J., and Berger, T. An exploratory study on the engineering of security features. In *International Conference on Software Engineering (ICSE)* (2025), IEEE, pp. 2470–2482.

[25] Hermann, K., Schneider, S., Tony, C., Yardim, A., Peldszus, S., Berger, T., Scandariato, R., Sasse, M. A., and Naiakshina, A. A Taxonomy of Functional Security Features and How They Can Be Located. *Empirical Software Engineering (EMSE) 30*, 5 (2025), 117.

[26] Koana, U. A., Le, Q. H., Raman, S., Carlson, C., Chew, F., and Nayebi, M. Examining Ownership Models in Software Teams. *Empirical Software Engineering (EMSE) 29*, 6 (2024).

[27] Kotstein, S., and Bogner, J. Which RESTful API Design Rules Are Important and How Do They Improve Software Quality? A Delphi Study with Industry Experts. In *Symposium and Summer School on Service-Oriented Computing*, vol. 1429. 2021.

[28] Koçi, R., Franch, X., Jovanovic, P., and Abelló, A. A Data-Driven Approach to Measure the Usability of Web APIs. In *Euromicro Conference on Software Engineering and Advanced Applications* (2020).

[29] Kvale, S., and Brinkmann, S. *InterViews: Learning the Craft of Qualitative Research Interviewing*. Sage, 2009.

[30] Lauret, A. API Stylebook – Design Guidelines. http://apistylebook.com/design/guidelines/, 2017.

[31] Levén, W., Broman, H., Besker, T., and Torkar, R. The Broken Windows Theory Applies to Technical Debt. *Empirical Software Engineering (EMSE) 29*, 73 (2024).

[32] Lo Iacono, L., Nguyen, H. V., and Gorski, P. L. On the Need for a General REST-Security Framework. *Future Internet 11*, 3 (2019).

[33] Microsoft. Microsoft REST API Guidelines. https://github.com/microsoft/api-guidelines, 2024.

[34] Miles, M. B., Huberman, A. M., and Saldana, J. *Qualitative Data Analysis: A Methods Sourcebook*. Sage, 2014.

[35] Miller, D., Whitlock, J., Gardiner, M., Ralphson, M., Ratovsky, R., and Sarid, U. OpenAPI Specification. Tech. Rep. v3.1.0, OpenAPI Initiative, 2021.

[36] Murphy, L., Alliyu, T., Macvean, A., Kery, M. B., and Myers, B. A. Preliminary Analysis of REST API Style Guidelines. In *Workshop on Evaluation and Usability of Programming Languages and Tools* (2017).

[37] Murugan, S. Top 10 Python REST API Frameworks in 2024. https://www.browserstack.com/guide/top-python-rest-api-frameworks, 2024.

[38] Network Working Group. Uniform Resource Locators (URL). RFC 1738, Internet Engineering Task Force (IETF), 1994.

[39] Network Working Group. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986, Internet Engineering Task Force (IETF), 2005.

[40] Neumann, A., Laranjeiro, N., and Bernardino, J. An Analysis of Public REST Web Service APIs. *Transactions on Services Computing 14*, 4 (2021).

[41] Nguyen, H. V., Tolsdorf, J., and Lo Iacono, L. On the Security Expressiveness of REST-Based API Definition Languages. In *Trust, Privacy and Security in Digital Business* (2017).

[42] Palma, F., Gonzalez-Huerta, J., Founi, M., Moha, N., Tremblay, G., and Guéhéneuc, Y.-G. Semantic Analysis of RESTful APIs for the Detection of Linguistic Patterns and Antipatterns. *International Journal of Cooperative Information Systems (IJCIS) 26*, 02 (2017).

[43] Parnas, D. L. Software Aging. In *International Conference on Software Engineering (ICSE)* (1994).

[44] Patton, M. Q. *Qualitative Research and Evaluation Methods*. Sage, 2015.

[45] Peldszus, S., Grosser, K., Konersmann, M., Brunotte, W., Ahrens, M., Schneider, K., and Jürjens, J. Too Many Issues: Automatically Prioritizing Analyzer Findings by Tracing Security Importance. *Transactions on Software Engineering and Methodology (TOSEM) 35*, 3 (2026).

[46] Peldszus, S., Rutenkolk, J., Heide, M., Sollmann, J., Klatt, B., Köhne, F., and Berger, T. Appendix: Interview guide and transcript of questions. https://doi.org/10.5281/zenodo.19390280, 2026.

[47] Petrillo, F., Merle, P., Moha, N., and Guéhéneuc, Y.-G. Are REST APIs for Cloud Computing Well-Designed? An Exploratory Study. In *International Conference on Service-Oriented Computing* (2016).

[48] Piccioni, M., Furia, C. A., and Meyer, B. An Empirical Study of API Usability. In *International Symposium on Empirical Software Engineering and Measurement (ESEM)* (2013).

[49] Preston-Werner, T. Semantic versioning. Tech. Rep. 2.0.0, 2023.

[50] Quintana, C., Krajcik, J., and Soloway, E. Scaffolding Design Guidelines for Learner-Centered Software Environments. In *Annual Meeting of the American Educational Research Association* (2002).

[51] RapidAPI. Rapid APIs State of APIs. https://stateofapis.com/, 2022.

[52] Rauf, I., Troubitsyna, E., and Porres, I. A Systematic Mapping Study of API Usability Evaluation Methods. *Computer Science Review 33* (2019), 49–68.

[53] Robles, N., Potes, N., Garcés, K., Izquierdo, J. L. C., and Cabot, J. Exploratory Analysis of the Structural Evolution of public REST APIs. In *Congresso Ibero-Americano em Engenharia de Software* (2023).

[54] Rodríguez, C., Baez, M., Daniel, F., Casati, F., Trabucco, J. C., Canali, L., and Percannella, G. REST APIs: A Large-Scale Analysis of Compliance with Principles and Best Practices. In *International Conference on Web Engineering* (2016).

[55] Santos, W. Which API Types and Architectural Styles are Most Used? *Programmable Web* (2017). Accessible via the Internet Archive: https://web.archive.org/web/20220712162418/https://www.programmableweb.com/news/which-api-types-and-architectural-styles-are-most-used/research/2017/11/26.

[56] Sinha, A. Client-Server Computing. *Communications of the ACM 35*, 7 (1992).

[57] SmartBear Software. Swagger – API Development for Everyone. https://swagger.io, 2024.

[58] SonarSource SA. SonarQube Coding Rules. https://next.sonarqube.com/sonarqube/coding_rules, 2025.

[59] Storey, M. D., Ernst, N. A., Williams, C., and Kalliamvakou, E. The Who, What, How of Software Engineering Research: A Socio-Technical Framework. *Empirical Software Engineering (EMSE) 25*, 5 (2020).

[60] Sundberg, E., Ekmark, T., and Ayele, W. Y. Validating API Design Requirements for Interoperability: A Static Analysis Approach Using OpenAPI. *arXiv abs/2511.17836* (2025).

[61] VERBI GmbH. MAXQDA. https://www.maxqda.com/, 2024.

[62] Verborgh, R., and Dumontier, M. A Web API Ecosystem through Feature-Based Reuse. *Internet Computing 22*, 3 (2018).

[63] Wilde, E. Surfing the API Web: Web Concepts. In *The Web Conference* (2018).

[64] Wilde, E. API Guidelines in the Wild. https://dret.github.io/guidelines/, 2024.
[65] Williams, M., and Moser, T. The Art of Coding and Thematic Exploration in Qualitative Research. *International Management Review 15* (2019).
[66] Yamamoto, R., Ohashi, K., Fukuyori, M., Kimura, K., Sekiguchi, A., Umekawa, R., Uehara, T., and Aoyama, M. A Quality Model and Its Quantitative Evaluation Method for Web APIs. In *Asia-Pacific Software Engineering Conference (APSEC)* (2018).
[67] Zalando SE Opensource. API Guildelines. https://opensource.zalando.com/restful-api-guidelines, 2021.
[68] Zhang, T., Hartmann, B., Kim, M., and Glassman, E. L. Enabling Data-Driven API Design with Community Usage Data: A Need-Finding Study. In *Conference on Human Factors in Computing Systems* (2020).