




To deprecate or to simply drop operations? An empirical study on the evolution of a large OpenAPI collection

Fabio Di Lauro , Souhaila Serbout , Cesare Pautasso 
fabio.di.lauro@usi.ch,souhaila.serbout@usi.ch,c.pautasso@ieee.org

Software Institute, USI, Lugano, Switzerland

Abstract

OpenAPI is a language-agnostic standard used to describe Web APIs which supports the explicit deprecation of interface features. To assess how APIs evolve over time and observe how their developers handle the introduction of breaking changes, we performed an empirical study on a dataset composed of more than one million API operations described using OpenAPI and Swagger format. Our results focus on detecting breaking changes engendered by operations removal and whether and to which extent deprecation is used to warn clients and developers about dependencies they should no longer rely on. Out of the 41,627 APIs considered, we found only 263 (0.6%) in which some operations are deprecated before being removed, while the developers of 10,242 (24.6%) of them directly remove operations without first informing their clients about the potentially breaking change. Furthermore, we found that only 5.2% of the *explicit-deprecated* operations and 8.0% of the *deprecated-in-description* operations end with a removal, suggesting a tendency to deprecate operations without removing them. Overall, we observed a low negative correlation between the relative amount of deprecated operations and the age of the corresponding APIs.

1 Introduction

Web APIs evolve in different ways (e.g. introduce/alter/refactor/remove endpoints) and for a multitude of reasons [4, 5, 10]. The extension of an API by adding new features is usually a safe operation, which does not affect existing clients. Conversely, when API maintainers need to remove or alter existing functionalities [2, 11], and consequently introduce breaking changes, they should guarantee the stability of their offerings [6] for example announcing those modifications well in advance in order to make clients aware of possible abnormal behaviours of their applications, in case they will not update them [5].

The goal of this study is to determine whether and to which extent Web API maintainers make use of deprecation [7] to announce future potentially breaking changes. One may expect that such practice is well established, given the wide and growing adoption of HTTP-based APIs across the industry.

To assess whether this is indeed the case, we analyze a large collection of Web APIs [1] described using OpenAPI [8], because of its growing industry adoption [3, 9] and its support for explicit deprecation metadata.

In particular, we aim to answer the following research questions:

Q1: How do API operations evolve over time? How stable are they?

Q2: How often an operation is declared deprecated before its removal?

Q3: Does the amount of deprecated operations always increase over the API commit histories?

Overall, we found high stability of API operations over time and that the number of deprecated operations shows a positive correlation with the API age only for a subset of the collected API histories. After mining the operation state model from all observed API changes, we measured that the majority of removed operations had not been deprecated before their removal. This unexpected result requires further study to determine whether it is due to the relative novelty of the deprecation metadata or to a lack of explicit API evolution guidelines and tools to enforce them.

The rest of this paper is structured as follows. Section 2 presents an overview of the dataset used in this study. Section 3 shows our results and we discuss them in Section 4. Section 5 summarizes related work. We conclude our study and indicate possible future work in Section 6.

2 Dataset Overview

We mined GitHub from December 1st, 2020 to December 31th, 2021 looking for YAML and JSON files, which comply with the OpenAPI [8] standard specification, in order to retrieve API descriptions artifacts. The mining activity produced a total of 271,111 APIs with their histories contained in a total of 780,078 commits. We built a tool, hereinafter called *crawler*, that mines those artifacts and saves associated metadata (commits timestamp, API title, versions, and others), and validates their compliance with Swagger and OpenAPI standards using the *Prance* and *open-api-spec-validator* tools, and finally parse them and extract relevant information for this study. After the validation process, we obtained a dataset of 166,763 valid APIs from which we removed 17,059 APIs with duplicate histories. Subsequently, we removed 12,645 APIs with no operations defined in their histories. This data cleanup resulted in 137,059 APIs with at least one operation and 41,627 unique APIs with valid descriptions, at least one operation, and more than one commit in their history.

3 Results

3.1 Deprecation Detection

In this study we distinguish two types of deprecation: i) *explicit-deprecation* introduced at operations, parameters and schema levels through the dedicated *deprecated* field, defined from OpenAPI 3.0; ii) *deprecation-in-description*, detected

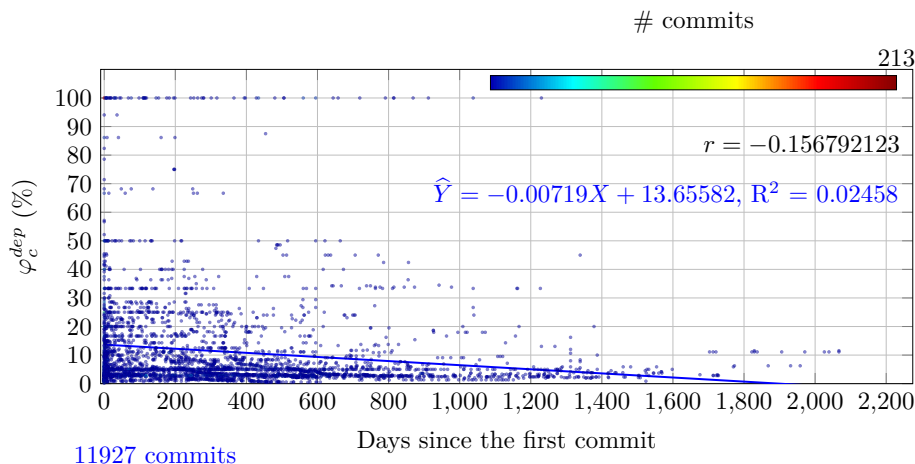


Fig. 1. Deprecated operations ratio φ_c^{dep} vs. API relative age: Does the presence of deprecated operations increase over time?

analyzing descriptions fields written in natural language. The latter heuristic is implemented by matching a list of terms, formed by words which start with the prefix *deprecate-*, against the text of description fields. This is similar to the detection heuristic of the earlier study by Yasmin et al. [12].

We detected 5,586 APIs which contain *explicit-deprecated* components and 384 APIs which have *deprecation-in-description* components. Out of these, only 165 APIs make use of both techniques to annotate deprecated components.

3.2 Operation Stability over Time

To assess the relative amount of deprecated operations we define the indicator:

$$\varphi_c^{dep} = \frac{|O_c^{dep}|}{|O_c|} \quad \text{where: } O_c^{dep} \subseteq O_c \quad (1)$$

$$O_c := \{ op \mid op \text{ is an operation detected in the commit } c \}$$

$$O_c^{dep} := \{ dop \mid dop \text{ is a deprecated operation detected in the commit } c \}$$

Fig. 1 shows how the indicator φ_c^{dep} changes depending on the commit age (relative to the first commit timestamp of the API history). The dots color shows how many commits we found with the same φ_c^{dep} at the same age. Considering all commits of all APIs together, we can observe a very small negative correlation r^{age} between the two variables. More in detail, we computed the same correlation r_i^{age} separately across each API history i . In Fig. 2 we present the histogram showing the distribution of the $\langle \varphi_c^{dep}, \text{age} \rangle$ correlation over 466 APIs for which it was possible to compute it. We can observe that 157 (33.7%) of the 466 APIs

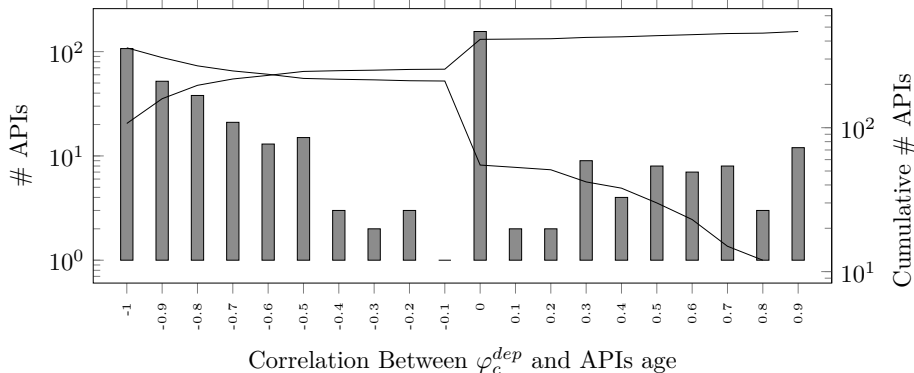


Fig. 2. Distribution of the Correlation r_i^{age} over 453 API histories

analyzed have $-0.1 \leq r_i^{age} \leq 0.1$ while 251 (53.9%) of the APIs have a negative correlation $-1 \leq r_i^{age} \leq -0.2$.

3.3 Operation state model

Based on tracking the changes occurring to all API operations for each commit, we inferred the state model shown in Fig. 3. Once created (c), an operation¹ can change its state to deprecated (d) or removed (r). Sometimes the APIs maintainers can choose to reintroduce a removed operation bringing it back to a c (*reintroduce* transition) or d state (*reintroduce_deprecated* transition). We define the **deprecate** transition when a commit introduces an *explicit-deprecation* or a *deprecation-in-description* for an operation. The opposite state change is represented by the **undeprecate** transition which occurs when an operation is not marked anymore as deprecated. Every state has its own self-loop transition which represents operations that remain in the same state between two consecutive commits.

In Fig. 3 we count how many operations were found in each of their initial and final states as well all the transitions between pairs of states. We measured also that 1,188 (2.9%) APIs include *reintroduce* and *reintroduce_deprecated* transitions in their histories and 7,663 (10.9%) of the 70,457 operation removals are later reintroduced.

In Fig. 4 we can observe how widely adopted are different API evolution practices are. For 9,779 APIs, operations are only added, thus ensuring backwards compatibility. In 5,106 APIs, operations are only deprecated, thus avoiding breaking changes. In 603 APIs, operations are only removed, thus potentially breaking clients depending on them. A different set of 9,122 APIs adopts both operation addition and removal, without performing any intermediate deprecation. Overall, deprecated operations are found in 5,805 APIs.

¹ To simplify the analysis and reduce its cost, in this section we focus at the operation level neglecting the parameters, responses and schema levels

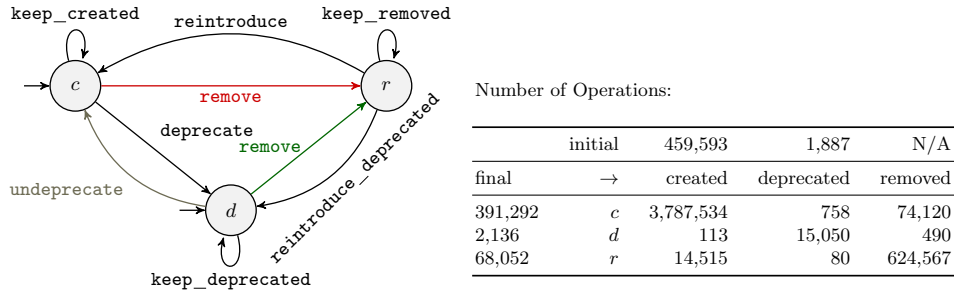


Fig. 3. Operations State Model

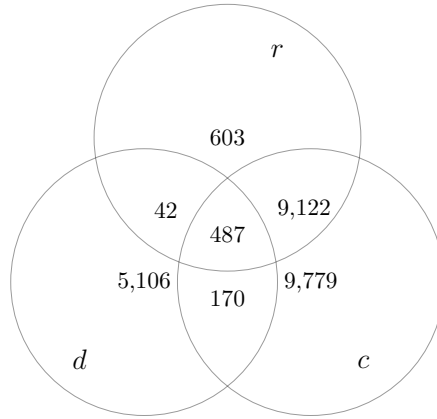


Fig. 4. Number of APIs which present at least one deprecated (*d*), removed (*r*), and/or created (*c*) operation in one commit of their history

We observed that 7,669 (94.8%) of the *explicit-deprecated* operations and 844 (92.0%) of the *deprecated-in-description* operations remain in the deprecated state (*d*). This means that for most operations, they are not removed after being deprecated and persist in further commits, until the last one. Excluding the transitions which start and terminate in the same state, we counted a total of 559,673 operation state transitions across all commits. Table 1 presents some statistics on their duration. On average, operations get reintroduced much faster than what it takes to remove them. Also, the longest transition from deprecated to removed took 43.2 months. In some cases, few operations did repeatedly get removed but also reintroduced (up to 50 times), as shown with the sub-sequences marked with * in Table 2.

Overall, considering all of the 41,627 APIs analyzed in this study, we detected that only 263 (0.6%) of these APIs include the removal of previously deprecated operations for at least one commit while many more, 10,242 (24.6%) of them directly remove operations (Fig. 5).

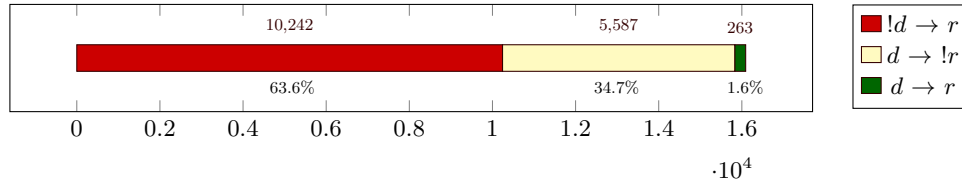


Fig. 5. Number of APIs classified based on the presence in at least one commit of their history, for the same operation undergoing different state transitions: only deprecate ($!d \rightarrow r$), only remove ($d \rightarrow !r$), or deprecated followed by remove ($d \rightarrow r$).

Table 1. Statistics on the Time Between State Transitions

transition	minimum	average	median	maximum
created \rightarrow removed	0	14.6 wks	9.5 d	67.5 mths
removed \rightarrow created	0	2 mins	46.4 hrs	37.4 mths
created \rightarrow deprecated	2 mins	41.9 wks	29.5 wks	50.1 mths
deprecated \rightarrow created	2 mins	34.1 d	52.5 hrs	13.5 mths
deprecate \rightarrow removed	0	66 d	9.8 d	43.2 mths
removed \rightarrow deprecated	64 secs	15.3 d	51.4 hrs	20.6 wks

4 Discussion

Q1: How operations evolve over time? How stable are they? 384,715 (83.4%) of the initial transitions end in a *created* final state passing through only one *create* transition. This result denotes a high stability of the analyzed operations.

We also detect 2,114 (0.2%) operations which remain in a *deprecated* state until the end of their history but only 466 operations follow the *deprecate-remove* path, i.e. they conclude their lifecycle with a *deprecate* transition followed by a *remove*. We also observe that 1,887 (0.4%) of the initial transitions lead directly to the deprecated state, thus indicating that collection includes few artifact histories that lack the initial created state. Furthermore, we can observe from Table 2 that 67,577 (14.7%) operations out of the 459,593 operations that were initially *created* end with a final removal of the involved operations passing through the transitions sequences *create* \rightarrow *remove* and *created* \rightarrow *removed* \rightarrow (*created* \rightarrow *removed*)*, i.e. with these sequences the developers could potentially introduce breaking changes, due to the absence of the intermediate *deprecate* transition.

Q2: How often an operation is deprecated before its removal?

Most operations are removed without being previously deprecated. Out of the 67,577 removed operations, only 466 had been previously deprecated. Furthermore, 419 (5.2%) of the *explicit-deprecated* operations and 73 (8.0%) of the *deprecated-in-description* operations end with a removal.

Table 2. Operations and APIs Following State Transition Sequences

Transition Sequence	# Operations	# APIs
created → removed	64,740	9,837
created → removed → created	4,968	1,234
created → removed → (created → removed)*	2,837	584
created → (removed → created)*	1,555	255
created → deprecated	636	262
created → deprecated → removed	60	25
created → deprecated → created	34	7

Q3: Does the amount of deprecated operations always increase over the API commit histories? According to our measurements the number of deprecated operations, overall, has a small negative correlation with the age of the corresponding API description (Fig. 1). When analyzing individual API histories, we found 59 APIs with a positive correlation between the two variables (Fig. 2).

5 Related Work

Deprecation of Web APIs has been studied by Yasmin et al. in [12]. In this work we are performing a broader-deeper analysis of a recently collected dataset of larger APIs with longer change histories. Yasmin et al. collected 3,536 OAS belonging to 1,595 unique RESTful APIs and they analyzed RESTful API deprecation on this dataset, proposing a framework called RADA (RESTful API Deprecation Analyzer). The authors filtered the dataset removing duplicate APIs, erroneous OAS and unstable versions, resulting in 2,224 OAS that belongs to histories of 1,368 APIs. In this work, we adopted the same heuristics used by RADA and applied them to a much larger API collection. It consists on determining which OAS components are deprecated by the providers based on the optional boolean `deprecated` field and a list of keywords to be searched in components description fields in order to identify potential components deprecation. Yasmin et al. cluster the considered APIs within three categories: i) *always-follow* for APIs which always deprecate before removing elements. ii) *always-not-follow* for APIs which introduce breaking changes without any deprecation information in previous versions; iii) *mixed* which contains APIs that show an hybrid behavior of i) and ii). While Yasmin et al. consider deprecation at operation, request parameters and responses level, in our study we focus only at operation level. The study performed by Yasmin et al. reveals that the majority of the considered RESTful APIs do not follow the *deprecate-remove* protocol. Our study confirm this conclusion, as stated in sub-section 3.3.

6 Conclusion

Do developers deprecate or simply remove operations as they evolve their APIs? In this empirical study we found that developers follow backwards compatible practices as they tend to grow their APIs by adding new operations. When removing operations, however, they do not often annotate them as deprecated, thus potentially breaking clients without previously warning them about the operation about to be removed. Another finding is that the operation state model reconstructed from observing API changes is a fully connected graph: there exists at least one operation in some API in which any possible state transition (between the created, deprecated and removed states) occurs. Further work is needed to investigate and gain a better understanding of the reasons for these observations, for example, by further classifying APIs based on whether they are still being developed or they have already been deployed in production.

Acknowledgements

This work is funded by the SNSF, with the API-ACE project nr. 184692.

References

- [1] Di Lauro, F., Serbout, S., Pautasso, C.: Towards large-scale empirical assessment of web apis evolution. In: 21st International Conference on Web Engineering (ICWE2021), Springer, Biarritz, France (May 2021)
- [2] Hora, A., Etien, A., Anquetil, N., Ducasse, S., Valente, M.T.: APIEvolutionMiner: Keeping api evolution under control. In: Proc. IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE) (2014)
- [3] Karlsson, S., Čaušević, A., Sundmark, D.: Quickrest: Property-based test generation of openapi-described restful apis. In: IEEE 13th International Conference on Software Testing, Validation and Verification (ICST) (2020)
- [4] Lauret, A.: The Design of Web APIs. Manning (2019)
- [5] Li, J., Xiong, Y., Liu, X., Zhang, L.: How does web service api evolution affect clients? In: IEEE 20th International Conference on Web Services (2013)
- [6] Lübke, D., Zimmermann, O., Pautasso, C., Zdun, U., Stocker, M.: Interface evolution patterns — balancing compatibility and flexibility across microservices life-cycles. In: Proc. 24th European Conference on Pattern Languages of Programs (EuroPLOP 2019), ACM (2019)
- [7] Murer, S., Bonati, B., Furrer, F.: Managed Evolution - A Strategy for Very Large Information Systems. Springer (2010)
- [8] OpenAPI Initiative: <https://www.openapis.org/> (2022), accessed: 2022-05-11
- [9] Serbout, S., Pautasso, C., Zdun, U.: How composable is the web? an empirical study on openapi data model compatibility. In: Proc. IEEE World Congress on Services (ICWS Symposium on Services for Machine Learning), IEEE, Barcelona, Spain (July 2022)
- [10] Sohan, S., Anslow, C., Maurer, F.: A case study of web api evolution. In: IEEE World Congress on Services (2015)
- [11] Varga, E.: Creating Maintainable APIs. Springer (2016)
- [12] Yasmin, J., Tian, Y., Yang, J.: A first look at the deprecation of restful apis: An empirical study. In: IEEE International Conference on Software Maintenance and Evolution (ICSME) (2020)