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 RESTFul 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 1,192,664 API operations with histories distributed across 407,028 commits belonging to 149,704 distinct APIs. The APIs were selected out of a set of 271,111 APIs mined from GitHub, which are described in 780,078 OpenAPI description artifacts.

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. We found that only 5.2% of the *explicit-deprecated* operations and 8.0% of the *deprecated-in-description* operations end with a removal. We conclude that developers tend to avoid the operations removal after their deprecation, and even when they remove them, the tendency is do it in the first two years after the deprecation. While there is a low negative correlation between the presence of deprecated operations and the APIs age, we found also a weak negative correlation between the total amount of the defined operations and the ratio between the number of explicit-deprecated and the total amount of operations. This finding indicates the presence of more explicit-deprecated operations in smaller APIs, rather than in bigger ones.

1 Introduction

Web APIs evolve in different ways (e.g. introduce/alter/refactor/remove endpoints) and for a multitude of reasons [6, 7, 16] (e.g., to adapt them to emerging client or provider needs). The extension of an API by adding new features is usually a safe operation, which does not affect existing clients. Conversely, when API maintainers want to remove or alter existing functionalities [4, 17, 18], and consequently introduce breaking changes, they should guarantee the stability of their offerings [9] for example announcing those modifications in order to let

clients be aware of possible abnormal behaviours of their applications, in case they will not update them [3, 8].

The goal of this study is to determine whether and to which extent API maintainers make use of deprecation [10, 15] to announce future potentially breaking changes. To do so, we analyze Web APIs described using OpenAPI [12], because of its growing industry adoption [5] 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?

Q4: How does the amount of deprecated operations depend on the API size?

Q5: Do developers tend to remove or to keep deprecated operations? How much time is needed for their eventual removal?

To answer these questions we collected a large number of distinct OpenAPI models (149,704) and all of their change history from GitHub to analyse the changes that occur on the level of the API operations over time. Our goal is to assess how often developers introduce breaking changes in practice and how often they use deprecation to announce them before actually starting to remove operations. This study is possible because from version 3.0 of OpenAPI a deprecation annotation was introduced so that API maintainers can inform client developers about future changes that might cause their clients' malfunction. However, in earlier OpenAPI versions this annotation could only be represented using natural language text in the operations descriptions.

Overall, we found a high stability of operations over time and a small amount of them effectively removed after their deprecation. The presence of deprecated operations shows a low correlation with the API age and we detect a weak correlation between the amount of deprecated operations and the API size, measured as the total number of operations. Our measurements reveal a tendency not to remove the operations after their deprecation. Their eventual removal, when it happens, takes around 2 years (median).

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 selected case studies. In Section 5 we discuss the results. Section 6 summarizes related work. We conclude our study and indicate possible future work in Section 7.

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 [12] specifications, 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 save associated metadata (commits timestamp, API title, versions and others),

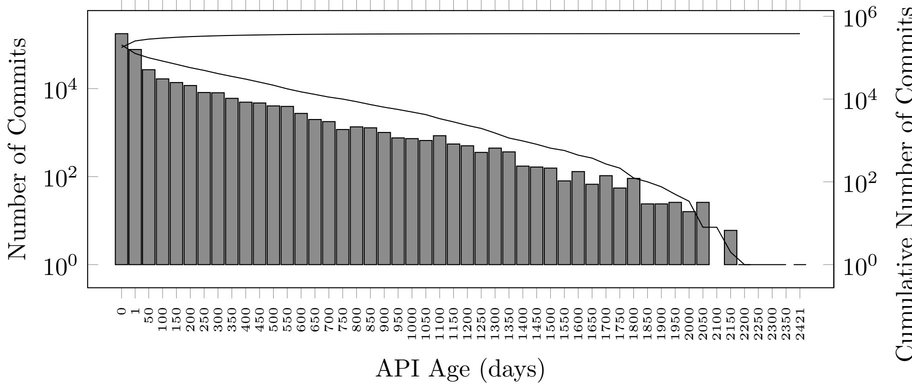


Fig. 1. Distribution of the API age metric across all commits

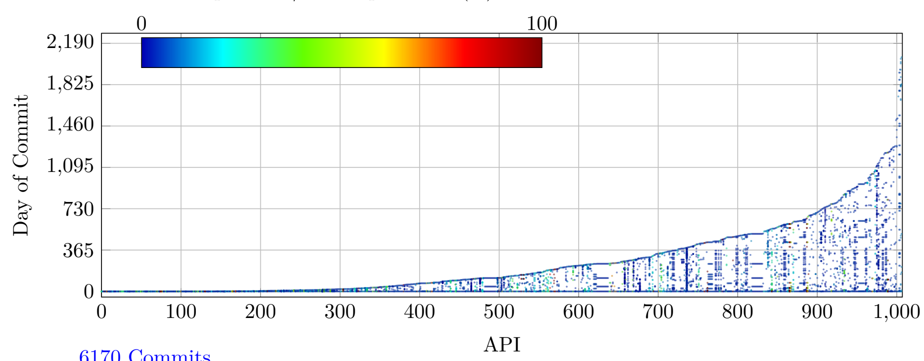
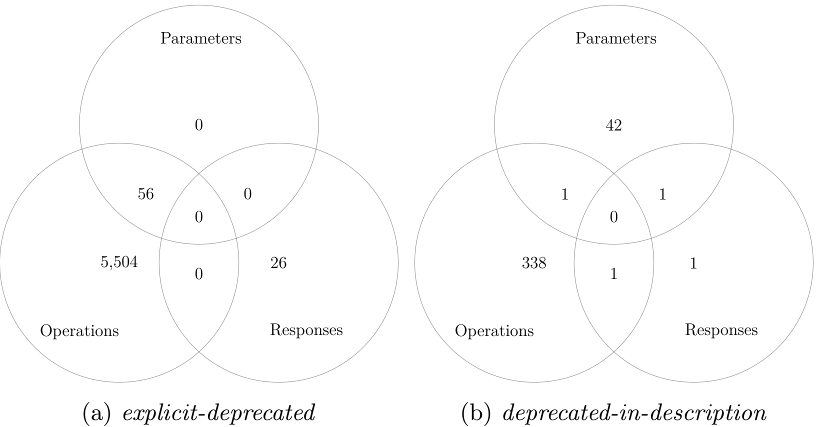


Fig. 2. Overview of APIs, sorted by relative age, with at least 1 deprecated operation and a minimum history of 2 commits

and validates their compliance with Swagger and OpenAPI standards using the *Prance* [13] and *open-api-spec-validator* [11] 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 APIs with no operations defined in their histories, which are a total of 12,645 APIs. The resulting dataset is composed by 137,059 unique and valid APIs, distributed across 333,936 commits. In Fig. 1, we count the number of commits, for every API, and we show how they distribute themselves across the APIs lifespan. 109,247 APIs (79.7% of the total APIs) present only one commit (API Age=0) as show in the first bar of Fig. 1.

The resulting dataset contains 1,491 APIs which have more than one commit and at least one commit including deprecated components. Fig. 2 presents an overview of the dataset. Each dot corresponds to one commit. Its color indicates the relative amount of deprecated operations. The commits are positioned on the Y axis according to their timestamp relative to the first commit of the API



(a) *explicit-deprecated* (b) *deprecated-in-description*

Fig. 3. Classification of APIs depending on how and where deprecated components (operations, parameters, responses) have been detected

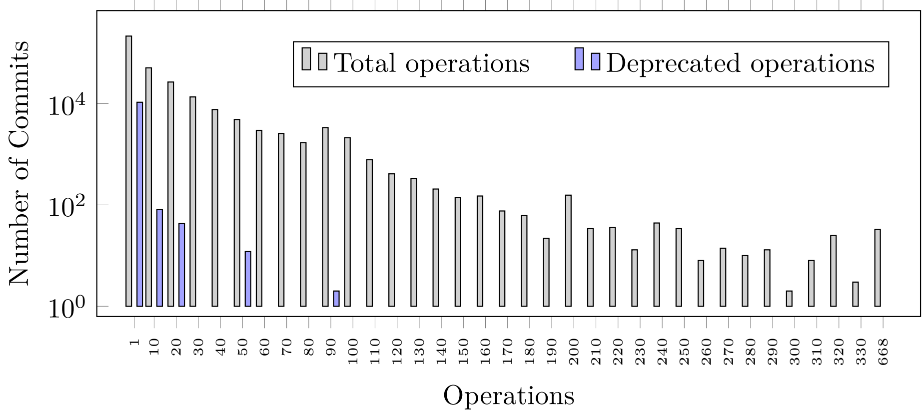


Fig. 4. Distribution of Total and Deprecated Operations over all API Commits

history. All commits of the same API description are aligned vertically. The APIs on the X axis are sorted according to their age, with the oldest ones found on the right side. We can see that 740 APIs have a commit history of less than one year, while a very small number reaches more than 5 years of age.

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 analyzing components - defined as the operations, parameters, responses

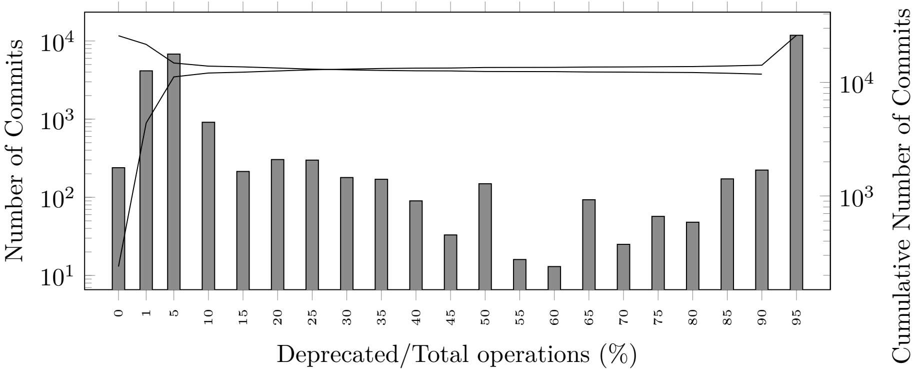


Fig. 5. Distribution of φ_c^{dep} over all API Commits

and schema - descriptions fields. The latter heuristic is implemented searching a list of keywords, formed by a list of the words which start with the prefix *deprecat-*, in components description fields. Using the previous classification we detected 5,586 APIs which contain *explicit-deprecated* components (Fig. 3.a) and 384 APIs which have *deprecation-in-description* components (Fig. 3.b). Fig. 4 shows how the API size (in terms of the Number of Operations metric) is distributed across all commits. The histograms also show a comparison, for each API size bin, between the total number of operations and the deprecated ones; as we can see the dataset contains relatively large artifacts (with hundreds of operations). The number of deprecated operations found within them however does not grow as much. We also measured that 95.8% of the 137,059 APIs considered don't have any deprecated operations in their histories. To assess the relative amount of deprecated operations we define the indicator:

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

$$O_c \coloneqq \{\, op \mid op \text{ is an operation detected in the commit c} \,\}$$
$$O_c^{dep} \coloneqq \{\, dop \mid dop \text{ is a deprecated}^1 \text{ operation detected in the commit c} \,\}$$

In Fig. 5 we show a histogram and cumulative distribution function of the φ_c^{dep} indicator with a logarithmic Y axis scale. Since this was plotted over the commits with at least one deprecated operation, the 0 value bin counts how many commits have $0 \leq \varphi_c^{dep} < 1$. Fig. 6 shows how the same 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.

[1] From here onwards, we focus on operations that are *explicit-deprecated* or *deprecated-in-description*, without further distinguishing how they have been detected.

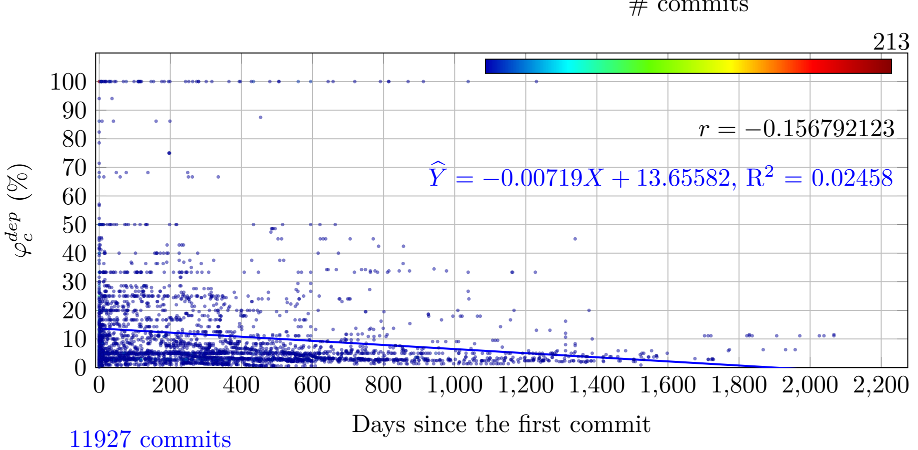


Fig. 6. Deprecated operations ratio φ_c^{dep} vs. API relative age: Does the presence of deprecated operations increase over time?

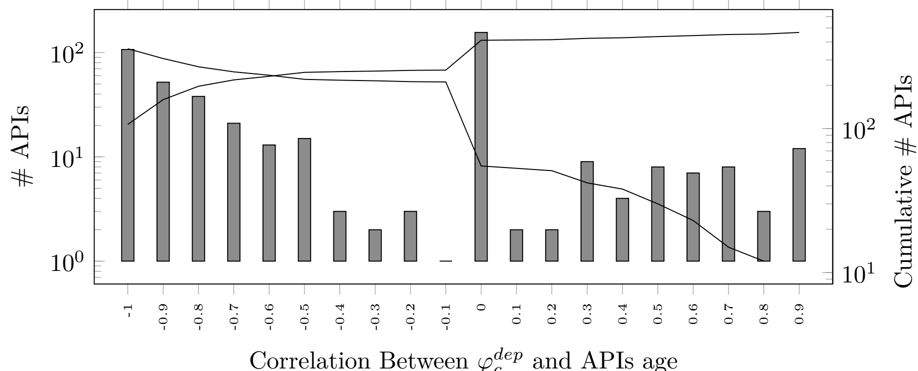


Fig. 7. Distribution of the Correlation r_i^{age} over 453 API histories

More in detail, we computed the same correlation r_i^{age} separately across each API history i. In Fig. 7 we present the histogram showing the distribution of the $< \varphi_c^{dep}$, age$>$ correlation over 466 APIs for which it was possible to compute it. We can observe that 33.7% of the 466 APIs analyzed have $-0.1 \leq r_i^{age} \leq 0.1$ while 53.9% of the APIs have a negative correlation $-1 \leq r_i^{age} \leq -0.2$.

In Fig. 8 we can observe the relationship between φ_c^{dep} and the API size across all commits. The dots color shows how many commits we detect with the same φ_c^{dep} and the same total number of operations $|O_c|$. The discretization effect on the left side is due to the indicator being computed over small APIs with less than 100 operations. Overall, the proportion of deprecated operations tends to decrease when the number of operations increase. The overall correlation r^{ops} value shows a weak negative global correlation between the two analyzed metrics.

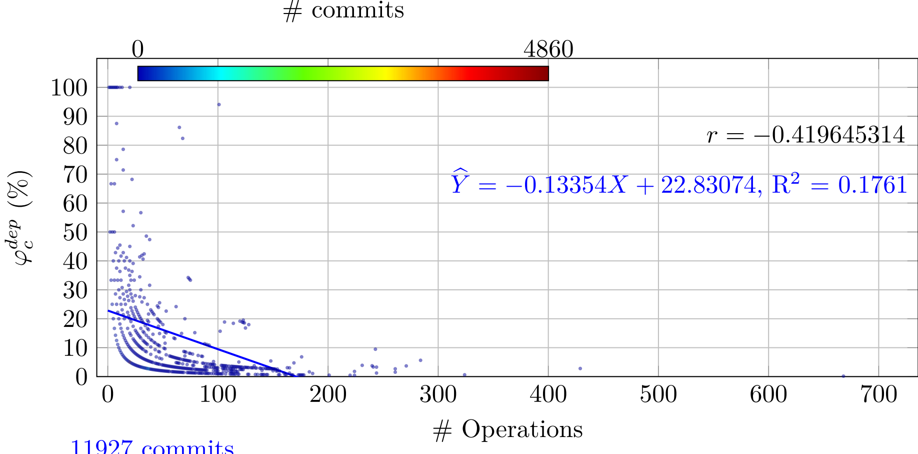


Fig. 8. Deprecated operations ratio φ_c^{dep} vs. API size: Does the presence of deprecated operations grow within larger APIs?

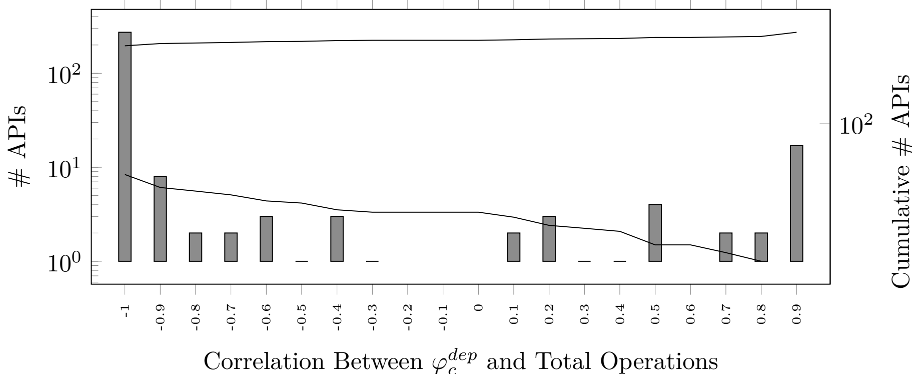


Fig. 9. Distribution of the Correlation r_i^{ops} over 257 API histories

As before, we also calculated the correlation separately r_i^{ops} for each API history i. Fig. 9 shows the distribution of the correlations r_i^{ops} over all APIs. The vast majority (86.5%) of the 325 APIs analyzed have $-1 \leq r_i^{ops} \leq -0.8$.

3.2 Operation state model

Based on tracking the changes ocurring to all API operations for each commit, we inferred the state model shown in Fig. 10. Once created (c), an operation 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 rep-

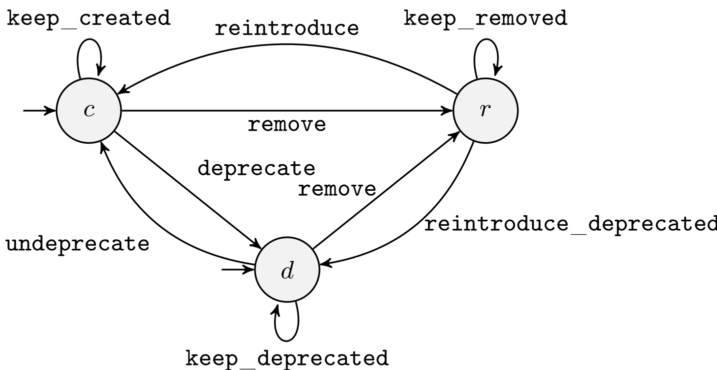


Fig. 10. Operations State Diagram

Table 1. Summary of Operations State Model (Number of Operations with Initial States, Final States and Transitions)

	initial	459,593	1,887	N/A
final	transition	created	deprecated	removed
391,292	created	3,787,534	758	74,120
2,136	deprecated	113	15,050	490
68,052	removed	14,515	80	624,567

resented by the `undeprecate` transition which occurs when an operation is not marked anymore as deprecated. To simplify the analysis and reduce its cost, in this section we focus at the operation level neglecting the parameters, responses and schema levels. Every state has its own self-loop transition which represents operations that remain in the same state between two consecutive commits. In Table 1 we count the operations in each of their initial and final states as well all the transitions between pairs of states. We measured also that 0.8% of APIs include *reintroduce* and *reintroduce_deprecated* transitions in their histories and 10.9% of the 70,457 operation removals are later reintroduced.

We observed that 94.8% of the *explicit-deprecated* operations and 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 land in the same state, we counted a total of 559,673 operation state transitions across all commits. Table 2 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 3.

Table 2. Statistics on the Time Between State Transitions

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

Table 3. Operations Following State Transition Sequences

Transition Sequence	Number of Operations
created → removed	64,740
created → removed → created	4,968
created → removed → (created → removed)*	2,837
created → (removed → created)*	1,555
created → deprecated	636
created → deprecated → removed	60
created → deprecated → created	34

3.3 Deprecated Operation Stability

After the deprecation, an operation can be removed, kept or brought back to a non-deprecated state. In Fig. 11 we show the distribution of the age of removed and maintained operations after their deprecation, using bins of 100 days. In Table 4 we reported the previous distributions, with the addition of operations brought back to non-deprecated state, setting the bins size to one year. In addition, we computed the likelihood for a deprecated operation to be removed (p_r), kept (p_k) or restored in a previous non-deprecated state (p_u).

4 Case studies

Out of the API collection we have selected four examples of APIs in which we can observe the evolution of their size over a period up to two years. They were selected as representatives of different classes of evolution behaviors, including cases in which both deprecation and operation removal occur. In Fig. 12 (left) we show a timeline of the total number of operations (black) and the number of deprecated operations (red) measured at every commit. The operation state model inferred from each API history is shown on the right column. The numbers on the edges indicate how many operations underwent the corresponding state transitions. The first case study shows how the Kubermatic API has grown from 14 until more than 100 operations over 1.7 years. The growth is not monotonic, as 77 operations have been permanently removed at various points in time. In the second case study the NASA UTM API (known as “NUSS UAS Operator

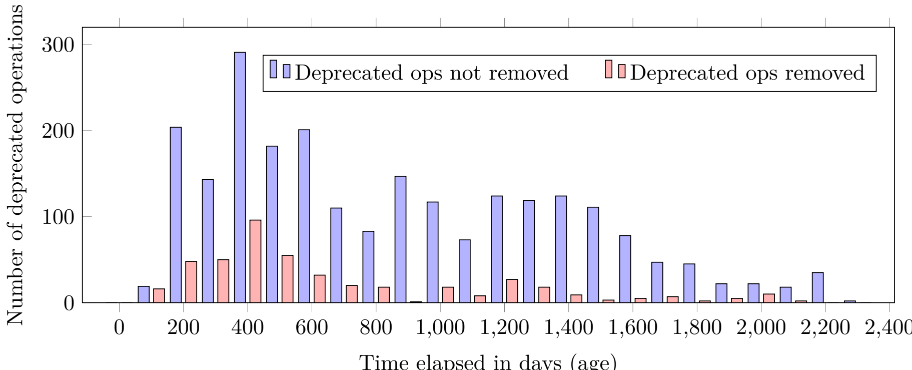


Fig. 11. Age reached by removed and not removed deprecated operations

Table 4. Stability of Deprecated Operations

Time (years)	Ops removed (p_r)	Ops not removed (p_k)	Ops un-deprecated (p_u)
$[\ 0, 1\ )$	96 (0.22)	319 (0.75)	13 (0.03)
$[\ 1, 2\ )$	204 (0.21)	759 (0.77)	20 (0.02)
$[\ 2, 3\ )$	53 (0.11)	406 (0.86)	11 (0.02)
$[\ 3, 4\ )$	60 (0.13)	410 (0.85)	10 (0.02)
$[\ 4, 5\ )$	18 (0.06)	299 (0.91)	10 (0.03)
$[\ 5, 6\ )$	19 (0.17)	87 (0.76)	8 (0.07)
$[\ 6, 7\ )$	0 (0.00)	40 (1.00)	0 (0.00)

API") undergoes a significant redesign after 485 days, when 1/3 of its operations are removed, and another third is deprecated and a few days later also removed. A similar behavior can be observed in the third case study, where two of the removed operations get reintroduced, one of which returns to the original non-deprecated state. The last case study shows the Mysterium Network API (known as "Tequila API") which also presents a larger proportion of directly removed operations. Also towards the end of its history 5 operations are deprecated and a similar number gets introduced shortly afterwards, probably suggesting that a replacement can be found.

5 Discussion

Q1: How operations evolve over time? How stable are they? 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. In this study, we detect 2,114 operations (0.2%) which remain in a *deprecated* state, as *explicit-deprecated* or *deprecated-in-description*, 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*.

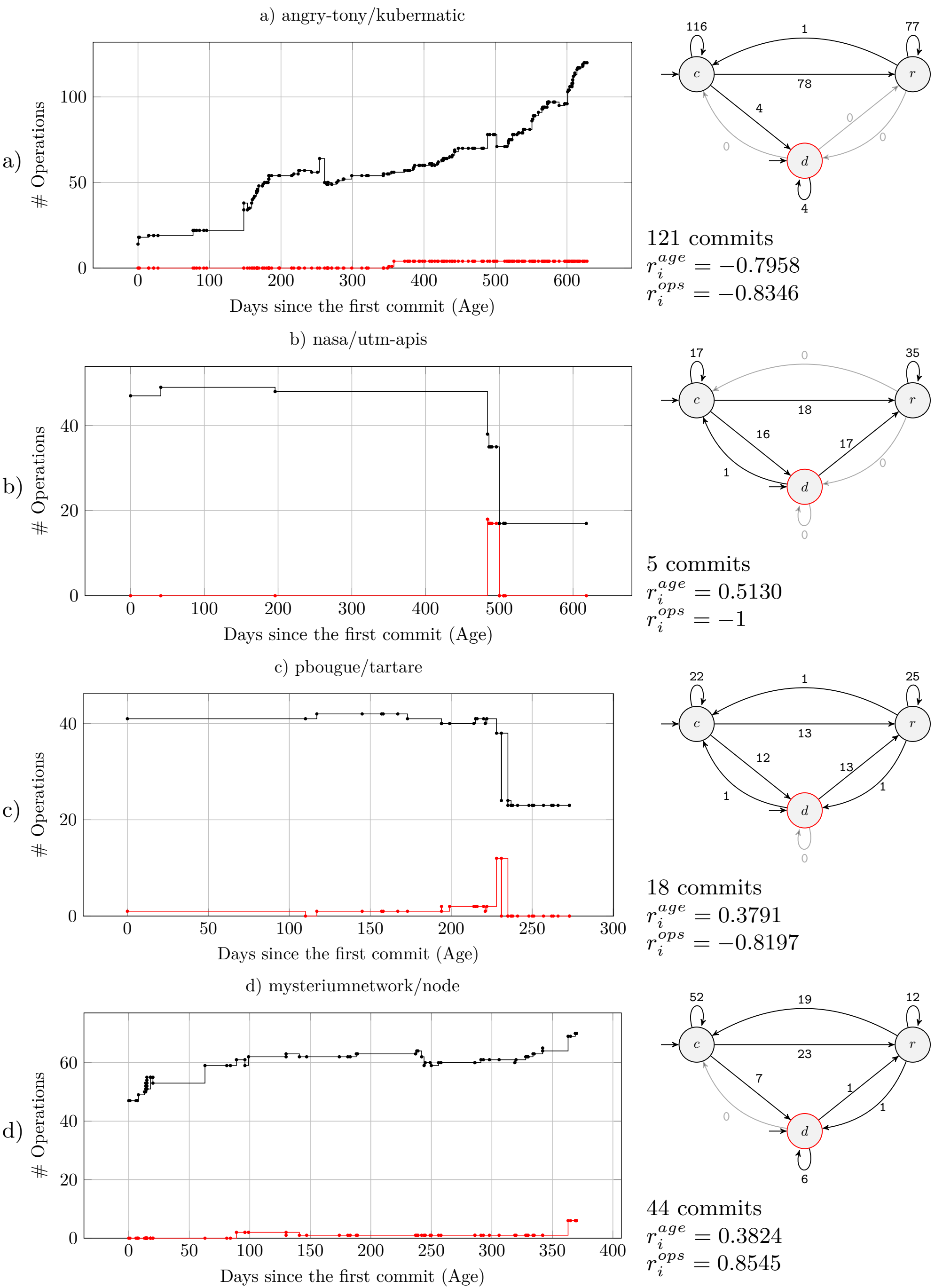


Fig. 12. API Evolution with Deprecated Operations – Case Studies

We also observe that 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, only 14.8% of the initial *create* transitions end with a final removal of the involved operations passing through the transitions sequence *create* $\rightarrow$ *remove*, i.e. with this sequence the developers potentially introduce breaking changes, due to the absence of the intermediate *deprecate* transition.

Q2: How often an operation is declared deprecated before its removal? 5.2% of the *explicit-deprecated* operations and 8.0% of the *deprecated-in-description* operations end with a removal.

Q3: Does the amount of deprecated operations always increase over the API commit histories? According to our measurements the number of deprecated operations, overall, have a small negative correlation with the age of the corresponding API description (Fig. 6). When analyzing individual API histories, we found some with a positive correlation between the two variables (Fig. 7).

Q4: How does the amount of deprecated operations depend on the API size? A weak negative correlation is found between φ^{dep} and the total number of operations, suggesting that, overall, proportionally less deprecated operations are found in larger APIs. This is confirmed by the more detailed analysis on the individual API histories (Fig. 9).

Q5: The developers tend to remove or keep the deprecated operations? How much time is needed for the removal? Analyzing the probabilities reported in Table 4 and the distribution of deprecated operations shown in Fig. 11 we can infer that developers tend to not remove operations after their deprecation but, in case they do it, the tendency is to remove them in the first 2 years.

6 Related Work

6.1 API Deprecation

Deprecation notifications in Web APIs has been studied by Yasmin et al. in [19]. In this work we are performing a broader-deeper analysis that goes towards the same direction of [19] on 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 dataset is formed by Swagger and OAS they mined from APIs.guru GitHub repository. The authors filtered the dataset removing duplicate APIs, erroneous OAS and unstable versions, resulting in 2,224 OAS that belongs to histories of

Table 5. Comparison of dataset sizes and main findings between Yasmin et al. [19] and this study

	Yasmin et al.	This study
Number of APIs	1,368	137,059
Number of OAS artifacts	2,224	360,882
APIs with deprecated components	16.0%	4.2%
APIs *always-follow*	12.0%	0.1%
APIs *always-not-follow*	85.0%	95.8%
APIs *mixed*	3.0%	4.1%

1,368 APIs. Then, they built a multi-version corpus comparing each OAS with the correspondent one in the previous version. The application of RADA on the filtered dataset shows that 11.55% of the versioned OAS contains at least one deprecated API element, forming a subset of the dataset composed by 219 deprecation-related 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. This approach, as explained by the authors, might suffer from some false positive and false negative cases. Later, the authors sampled 50 random API versions, which contain 3,444 API operations, and manually label the components deprecation. At the API version level, the comparison between the manual labeling and the RADA results shows that the RADA accuracy is 100% while at the API operation level RADA achieves a precision of 94% and a recall of 100%. 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 behaviour of i) and ii). In Table 5 we compare Yasmin et al. dataset and findings with our study results. 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 deprecated-removed protocol, in other words those APIs doesn't declare deprecation of components before removing them, and only 45% of the studied APIs suggest alternatives consistently for all deprecation-related operations.

Sawant et al. conducted a study [14] set up in two phases: an exploratory investigation, followed by the evaluation of the desirability and feasibility of enhancements they propose to the deprecation mechanism. In the first part of their study, the authors investigate about the reasons that bring API producers to deprecate features, eventually remove them from their APIs, and their expectations about the consumers reactions. They use an interpretive descrip-

tive technique to conduct and analyze interviews with 17 developers who work on APIs and they challenge their findings conducting a survey with 170 Java professionals. They propose three enhancements to the deprecation mechanism: two of them are related to the deprecation mechanism while the third address an issue at language level (Java). Their findings show how the API producers are wary about removing deprecated features and our study confirm their finding as shown in Fig. 11 and Table. 4. The authors found also that APIs consumers tend to promptly react to deprecation only if the reasons behind the deprecation is serious. This behaviour is probably due to the costs needed by the reactions. The authors proposal is denoted as **RSW**, which is the acronym of the three enhancements proposed: *i*) Removal dates should be marked; *ii*) Severity should be marked and *iii*) Warning mechanisms should be generic. The authors compared the deprecation mechanism implementation in 23 popular and new languages concluding they are not consistent in implementing it and none of them implement **RSW** fully. Unfortunately in OpenAPI there is no explicit support for such detailed deprecation metadata.

Brito et al. measure the usage of deprecation messages in Java and investigate about the need of a tool to recommend such messages [1]. The authors performed a large-scale analysis over 661 real world Java systems discovering that *i*) 64% of the API elements analyzed are deprecated with replacement messages: *ii*) they didn't find a concrete effort to improve deprecation messages over time and *iii*) they found statistically significantly differences between systems whom adopt a deprecation mechanism with replacement messages and ones which didn't use it. Unlike our approach that consists on tracking deprecation in each committed version of the APIs, Brito et al. analyzed only the first and the last release of each system considering the detected differences representative of a general overview of the system's evolution process.

6.2 Web APIs Evolution

Li et al. perform an empirical study [7] on Web API migration analyzing five popular APIs. The authors clustered the detected API changes into 16 change patterns and they identified 6 new problems in migrating Web API clients. Furthermore, they identified two unique characteristics of Web API evolution processes. They observe that there are more changes at the Java level (high level libraries provided by the APIs maintainers) rather than at the HTTP level and Web API evolution affects more operations than local API changes. Sohan et al. analyzed the evolution of multiple Web APIs [16] collected from proprietary and open-source environments. They used the differences between literature and industry practices to identify unresolved research problems. The authors identified six new change patterns to identify changes in Web API versions and provide also a list of recommendations for developers of evolving Web APIs such as the use of semantic versioning, differentiate the releases for bug fixes and new features, auto generate API documentation cross-linked with change logs and provide live API explorers. They found a lack of standard approaches to deal with Web API versioning, documentation and change communication. In our

study we found a similar problem regarding the deprecation process: while the new OpenAPI standard provide a precise way to deprecate operations, some developers prefer to annotate components as deprecated with natural language using the OAS description fields. In this paper we extended the findings of the authors of [2] on Web API evolution with more fine-grained metrics, focusing on detecting the presence of potentially breaking changes and the usage of the deprecation concept, on a much larger collection of API descriptions.

7 Conclusion

In this empirical study we found that most API descriptions in our collection do not have any deprecated operation in their commit histories. Still, within 5,805 APIs we detected a total of 21,754 operations deprecated explicitly or using natural language descriptions. We found that only 5.2% of the *explicit-deprecated* and 8.0% of the *deprecated-in-description* operations were eventually removed after their deprecation while 14.1% of the *created* operations were removed without first annotating them with any kind of deprecation. In Fig. 11 we can observe the distributions of the age of operations removed and kept after their deprecation. We can conclude, observing the probabilities measured in Table 4, that developers tend not to remove operations after their deprecation but, in case they do it, the tendency is to remove them within the first 2 years. Furthermore, we detect a low negative correlation between the number of deprecated operations and the age of their API and a weak negative correlation between the total amount of the operations and the ratio between the number of explicit-deprecated and the total amount of operations. The latter finding indicates that explicit-deprecated operations are more frequent in smaller APIs, rather than in bigger ones.

While it is interesting to study the operations evolution and how the API maintainers tend to use (or not) the deprecation concept, in order to handle the introduction of the breaking changes, we want to investigate possible extensions of this study to answer the following research questions:

– How many deprecated operations are found in APIs offered in a production environment?

– How many deprecated, and removed, operations are replaced with an another endpoint?

– How many API descriptions report replacement information when they introduce a deprecation?

Acknowledgements

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

References

[1] Brito, G., Hora, A., Valente, M.T., Robbes, R.: Do developers deprecate apis with replacement messages? a large-scale analysis on java systems. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), vol. 1, pp. 360–369 (2016), https://doi.org/10.1109/SANER.2016.99

[2] 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, Springer, Biarritz, France (May 2021)
[3] Dig, D., Johnson, R.: How do apis evolve? a story of refactoring. Journal of software maintenance and evolution: Research and Practice **18**(2), 83–107 (2006)
[4] 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), pp. 420–424 (2014)
[5] Karlsson, S., Čaušević, A., Sundmark, D.: Quickrest: Property-based test generation of openapi-described restful apis. In: 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), pp. 131–141 (2020), https://doi.org/10.1109/ICST46399.2020.00023
[6] Lauret, A.: The Design of Web APIs. Manning (2019)
[7] Li, J., Xiong, Y., Liu, X., Zhang, L.: How does web service api evolution affect clients? In: 2013 IEEE 20th International Conference on Web Services, pp. 300–307 (2013), https://doi.org/10.1109/ICWS.2013.48
[8] Li, J., Xiong, Y., Liu, X., Zhang, L.: How does web service API evolution affect clients? In: Proc. 20th International Conference on Web Services (ICWS) (2013)
[9] Lübke, D., Zimmermann, O., Pautasso, C., Zdun, U., Stocker, M.: Interface evolution patterns — balancing compatibility and flexibility across microservices lifecycles. In: Proc. 24th European Conference on Pattern Languages of Programs (EuroPLoP 2019), ACM (2019)
[10] Murer, S., Bonati, B., Furrer, F.: Managed Evolution - A Strategy for Very Large Information Systems. Springer (2010), ISBN 3-642-01632-4
[11] open-api-spec-validator: https://github.com/p1c2u/openapi-spec-validator (2022), accessed: 2022-05-11
[12] OpenAPI Initiative: https://www.openapis.org/ (2022), accessed: 2022-05-11
[13] Prance: https://pypi.org/project/prance/ (2022), accessed: 2022-05-11
[14] Sawant, A.A., Aniche, M., van Deursen, A., Bacchelli, A.: Understanding developers' needs on deprecation as a language feature. In: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), pp. 561–571 (2018), https://doi.org/10.1145/3180155.3180170
[15] Sawant, A.A., Huang, G., Vilen, G., Stojkovski, S., Bacchelli, A.: Why are features deprecated? an investigation into the motivation behind deprecation. In: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 13–24 (2018), https://doi.org/10.1109/ICSME.2018.00011
[16] Sohan, S., Anslow, C., Maurer, F.: A case study of web api evolution. In: 2015 IEEE World Congress on Services, pp. 245–252 (2015), https://doi.org/10.1109/SERVICES.2015.43
[17] Varga, E.: Creating Maintainable APIs. Springer (2016)
[18] Wang, S., Keivanloo, I., Zoua, Y.: How do developers react to RESTful API evolution? In: Proc. International Conference on Service-Oriented Computing, p. 245–259, Springer (2014)
[19] Yasmin, J., Tian, Y., Yang, J.: A first look at the deprecation of restful apis: An empirical study. In: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 151–161 (2020), https://doi.org/10.1109/ICSME46990.2020.00024