OASQuali: Automated Quality Analysis of OpenAPI Specifications

Decrop, Alix; Vandeloise, Mikel; Heymans, Patrick; Perrouin, Gilles

Published in:
Proceedings of the 26th International Conference on Web Engineering, ICWE 2026

Publication date:
2026

Document Version
Peer reviewed version

OASQuali: Automated Quality Analysis of OpenAPI Specifications

Alix Decrop, Mikel Vandeloise, Patrick Heymans, and Gilles Perrouin

NADI/PReCISE, Faculty of Computer Science, University of Namur, Namur, Belgium
{alix.decrop,mikel.vandeloise,patrick.heymans,gilles.perrouin}@unamur.be

Abstract. The rapid adoption of modern web architectures (i.e., REST APIs) relies on clear and accurate documentation. The OpenAPI Specification (OAS) standard is widely adopted for this purpose, as it is machine-readable and provides a range of API-related fields. However, maintaining high-quality documentation is complex, as the frequent omission of important information significantly hinders the effectiveness of testing tools and API understanding. To address this problem, we introduce OASQuali, a tool that quantifies OAS quality across five dimensions: format, version, metadata, servers, and descriptions/examples. We conduct a large-scale empirical evaluation with 2,529 public specifications. Our results reveal a mean documentation quality of 67.11%. While aggregate quality is independent of API size, we identify a systemic "semantic gap" in large-scale implementations: As APIs enlarge, they become more up-to-date with OAS versions (53.10% → 88.00%). However, the quality of their descriptions and examples drastically decreases (42.18% → 24.00%). Notably, the absence of parameter examples in 86.95% of specifications represents a major bottleneck for operational clarity. OASQuali provides a rigorous framework for revealing these deficiencies, paving the way for more thoroughly documented web services.

Keywords: OpenAPI Specification · REST APIs · Automated Analysis · Quality Metrics

1 Introduction

Many modern web architectures rely on REpresentational State Transfer (REST) APIs [9] that communicate via the HTTP protocol [8]. While these interfaces leverage standardized methods and headers, the diversity in implementation patterns, ranging from parameter handling to authentication mechanisms, makes simple protocol knowledge insufficient for integration [11]. Consequently, developers depend on documentation, predominantly using the widely adopted OpenAPI Specification (OAS) standard, to define machine-readable contracts in JSON or YAML formats [19].

However, maintaining high-quality OAS documentation is a complex and error-prone task. Prior research indicates that more than half of published specifications contain inconsistencies relative to their actual implementation [11].

Furthermore, the frequent omission of optional fields, such as human-readable descriptions and response examples, significantly degrades API usability [12]. Ambiguous endpoints, such as `GET /status` lacking descriptive metadata or structural schemas, hinder integration and prevent the effective use of automated testing and mock generation tools [5,10]. Such ambiguities also affect API users, who may not understand operations and/or parameter usage.

Existing research has primarily focused on structural metrics for maintainability [4] or the automated detection of REST design rule violations [3]. Yet, the automated assessment of API documentation usability (e.g., metadata presence, rich descriptions and examples, up-to-date OAS version, etc.) remains largely unaddressed [15]. This paper bridges this gap by investigating the quality thresholds of public REST API documentation through a multidimensional analytical lens.

Overall, this paper provides the following contributions:

- **OASQuali:** A deterministic tool to automate OAS quality assessment. Unlike standard validators, it fully resolves modular specifications (handling reference dependencies) to ensure metrics are calculated on the complete API specifications.
- **Quality Framework:** A formalization of OAS quality as a multidimensional vector that covers format (Structural Integrity), OAS version (Evolutionary Maturity), metadata (Administrative Governance), servers (Operational Readiness), and descriptions/examples (Semantic Communicability).
- **Empirical Evaluation:** A large-scale study of 2,529 public specifications originating from the *APIs.guru* directory [2], revealing that while average quality is tolerable, significant deficiencies persist regardless of API size.
- **Replication Package:** A publicly available repository containing our implementation of OASQuali and the curated dataset to ensure study reproducibility [7].

The remainder of this paper is structured as follows: Section 2 provides the background and related work; Section 3 details the tool methodology and architecture; Section 4 presents our empirical results; Section 5 discusses the implications of our findings; Section 6 addresses the potential threats to validity; and Section 7 concludes the paper with future research directions.

2 Background and Related Work

REST Architecture and HTTP. Modern web services predominantly implement the REpresentational State Transfer (REST) architectural style [9]. REST defines a set of constraints, including statelessness, a uniform interface, and client-server decoupling, to ensure scalability and evolvability. These services leverage the HyperText Transfer Protocol (HTTP) [8] for communication, utilizing standard methods (`GET`, `POST`, `PUT`, `PATCH`, `DELETE`) to perform CRUD (Create, Read, Update, Delete) operations on resources identified by URIs.

OpenAPI Specification (OAS). The OpenAPI Specification [19] (formerly known as Swagger) is the industry-standard for documenting REST APIs. An OAS document provides a machine-readable contract (in JSON or YAML) covering API metadata, available endpoints, parameter constraints, response schemas, and other API-related fields. Designed for automation, such specifications are also frequently rendered via tools such as *Swagger Editor* [17] to serve as a primary documentation for human developers.

Related Work. The automated analysis of REST API documentation has evolved along four primary research axes. While significant progress has been made in structural and architectural validation, the assessment of semantic quality remains underexplored.

- **Documentation Reliability vs. Implementation:** Hosono et al. [11] conducted a validation of consistency between documentation and implementation, revealing that nearly half of the specifications diverge from actual API behaviour. Their work focuses on correctness (fidelity to code) rather than intrinsic quality (content usability). Similarly, studies on API evolution show that breaking changes are often undocumented, emphasizing the accuracy of the specification yet overlooking its communicability. Additionally, Sohan et al. [18] identified usage examples as a critical component for the effectiveness of REST API documentation.
- **Structural Metrics and Maintainability:** A significant body of work focuses on complexity metrics derived from static analysis. Bogner et al. [4] adapted classic software quality metrics (e.g., Weighted Methods per Class) to service interfaces to assess maintainability. Serbout et al. [14,15] introduced *APIstic*, a framework computing over 800 structural metrics (e.g., schema depth, parameter counts) to analyse API ecosystems. While these approaches provide important insight into structural complexity, they are descriptive rather than normative regarding documentation richness: a technically complex API can be valid even if it lacks human-readable descriptions.
- **Design Compliance and Rule Enforcement:** Building on Massé's foundational REST rules [12] (e.g., using noun-based URIs or avoiding file extensions), tools like *RESTRuler* [3] automate the detection of syntactic violations (e.g., URI styles or misuse of HTTP methods). Complementarily, Singjai et al. [16] focus on ensuring conformance between API descriptions and higher-level architectural design decisions. These tools ensure syntactic and architectural compliance but do not evaluate the semantic utility of the content (e.g., the meaningfulness of a parameter example).
- **API Discovery and Recommendation:** Approaches like *API-Miner* by Moon et al. [13] leverage NLP (Sentence-BERT) and graph-based learning to improve API discovery. While they utilize descriptions to compute semantic similarity, they do not treat documentation richness as a standardized quality metric for developers.

Despite these contributions, existing tools focus mainly on architectural compliance or structural complexity. The automated assessment of human-related usability and semantically valid descriptions/examples through OAS remains largely unexplored. OASQuali addresses this gap by providing a deterministic framework to quantify documentation richness.

Terminology. In the remainder of this paper, an *API* refers to a REST (or RESTful) service, and a *specification* denotes its documentation in the OAS format [19].

3 Methodology

The design of OASQuali is motivated by the need for a deterministic framework to assess the structural and semantic quality of OpenAPI Specifications. To bridge the gap between raw interface definitions and quantifiable metrics, we propose a multi-stage analysis pipeline that systematically decomposes the specification's layers.

Specification, Acquisition, and Normalization. To ensure programmatic flexibility, OASQuali implements two acquisition modalities:

- **Local Mode:** Tailored for static analysis, it accepts path descriptors for JSON or YAML files, performing automatic normalization and metadata extraction from file headers.
- **In-Memory Mode:** Designed for integration into Continuous Integration (CI) pipelines, it processes pre-parsed objects.

Referential Resolution and Dereferencing. A critical challenge in OAS analysis is the modularity offered through the `$ref` keyword (i.e., a reference to an element definition for reusability purposes). Performing a naive static analysis on the root document would yield incomplete metrics and underestimate documentation coverage. OASQuali incorporates a preprocessing layer that traverses the specification to resolve all JSON pointers. This process "flattens" the API's object graph into a canonical, fully expanded representation, ensuring that subsequent quality evaluations are validly executed on OAS files.

Structural Validation and Compliance. Once flattened, the specification undergoes a two-tier validation protocol acting as a fail-fast gatekeeper:

1. **Syntactic Integrity:** A raw syntax verification ensures that the document adheres to JSON/YAML serialization standards.
2. **Schema Conformance:** Formal validation against official OpenAPI schema (v2.0, v3.0.x, v3.1.x). The engine logs specific `OpenAPIValidationError` instances to provide high-granularity feedback.

Multi-dimensional Quality Assessment. We formalize the quality of an OpenAPI Specification $\mathcal{S}$ as a multidimensional vector:

$$\mathcal{Q}(\mathcal{S}) = \langle \mathcal{I}, \mathcal{E}, \mathcal{G}, \mathcal{O}, \mathcal{C} \rangle$$

where each component represents an orthogonal dimension of usability:

- **Structural Integrity ($\mathcal{I}$):** Syntactical validation of JSON and adherence to the OAS metamodel.
  Motivation: While Massé defines rules for URI syntax [12], our metric extends this to the syntactic validity of the entire machine-readable specification, ensuring interoperability with tooling [19].
- **Evolutionary Maturity ($\mathcal{E}$):** Enforcement of Semantic Versioning (SemVer) patterns for OAS via regular expression matching.
  Motivation: Fielding emphasizes evolvability as a key constraint [9]. We operationalize this by verifying standard versioning patterns, a vital practice for client stability in distributed systems.
- **Administrative Governance ($\mathcal{G}$):** Verification of API metadata availability (i.e., API title, licence, contact).
  Motivation: Massé highlights the need for clear metadata to facilitate consumption [12]. Notably, we quantify this by enforcing the presence of legal (licence) and social (contact) metadata, which are often neglected in practice.
- **Operational Readiness ($\mathcal{O}$):** Evaluation of server presence, reachability, and security (HTTPS enforcement and active probes).
  Motivation: Unlike theoretical design guides [12], this dimension empirically verifies the static reachability of the implementation, addressing the reliability gap identified by Hosono et al. [11].
- **Semantic Communicability ($\mathcal{C}$):** Calculation of coverage metrics for description and example fields in OAS files.
  Motivation: While Massé advocates for "self-descriptive messages" [12], he does not provide metrics for documentation richness.

We associate with each dimension a set of *evaluations*. These evaluations correspond to specific checks performed by OASQuali and are listed in Table 1. For example, the evaluation `oas-version` checks a specification's adherence to the last version of the OAS standard, while the evaluation `parameter-examples` measures example coverage relative to the number of API parameters.

Synthesis and Scoring Mechanism. The final phase synthesizes outcomes into a structured report. For each dimension d, a success ratio S_d is calculated as:

$$S_d = \frac{\sum \text{passing evaluations of } d}{\sum \text{total evaluations of } d}$$

The global Composite Quality Index (CQI) is then derived from the weighted sum of these ratios:

$$CQI = \sum_{d \in \{\mathcal{I}, \mathcal{E}, \mathcal{G}, \mathcal{O}, \mathcal{C}\}} S_d \times w_d$$

where S_d is the success ratio of the dimension d and w_d is the weight assigned to the dimension d ($\sum w_d = 1$). By default, each dimension is assigned an equal weight (since we have five dimensions: $w_d = 0.2$), facilitating objective benchmarking across heterogeneous datasets.

4 Evaluation

The empirical assessment of OASQuali quantifies documentation quality across a diverse corpus of public OpenAPI Specifications and identifies prevalent structural and semantic deficiencies.

4.1 Research Questions

This study addresses three fundamental research questions (RQs):

- **RQ.1 (Formalization):** How can documentation quality be modelled as a set of quantifiable evaluations, derived from the OpenAPI Specification standard?
- **RQ.2 (State of Practice):** What is the quality distribution among public OpenAPI Specifications, and does API size correlate with documentation maturity?
- **RQ.3 (Critical Pitfalls):** Which specific quality evaluations exhibit the highest failure rates, and what are the implications for API usability?

4.2 Dataset

The evaluation corpus comprises 2,529 OAS files retrieved from the *APIs.guru* directory [2]. This dataset ensures external validity through significant heterogeneity, spanning diverse application domains (e.g., finance, geolocation, aviation, etc.) and varying API sizes.

4.3 Experimental Setup

We evaluated OASQuali on a machine with a 2.4GHz processor and 16GB of RAM. The protocol involved:

1. Local acquisition of specifications and metadata via the APIs.guru REST API.
2. Execution of the OASQuali analysis pipeline across the entire dataset.

As OASQuali is deterministic, results are reproducible and independent of stochastic variance. To ensure the practical utility of our tool, we evaluated its performance on standard consumer-grade hardware. The pipeline processes most specifications in under two seconds, demonstrating its viability for real-time integration into Continuous Integration (CI) environments without introducing significant latency.

4.4 RQ.1 - Formalization: Quality Evaluations

To define and quantify our quality evaluations, we operationalize the multi-dimensional quality model introduced in Section 3. While the quality vector $\mathcal{Q}(\mathcal{S})$ provides the theoretical framework, this step maps the dimensions to 17 deterministic evaluations.

Structural Integrity ($\mathcal{I}$), Evolutionary Maturity ($\mathcal{E}$), and Administrative Governance ($\mathcal{G}$) dimensions are assessed through binary presence and validity checks. The Operational Readiness ($\mathcal{O}$) dimension extends such checks by also assessing server reachability through HTTP requests and response analysis. In contrast, the Semantic Communicability ($\mathcal{C}$) dimension employs more sophisticated heuristics. For `description` fields, the engine executes a sequential validation pipeline comprising three stages. First, it ensures existence and normalization by verifying that the field contains non-whitespace characters, which are then stripped and lowercased for consistent matching. Second, to distinguish meaningful documentation from placeholders (e.g., "TBD", "fix me"), a verbosity threshold is applied where the word count W_d must exceed a configurable limit (default $W_{min} = 3$). The default count filters out trivial/non-informative outputs, which were frequently observed during preliminary experiments and did not convey meaningful content. Finally, for route operations, a keyword context check verifies the description against a dictionary of functional terms (e.g., "retrieve", "list" for `GET` operations) to ensure that the text accurately describes the performed action. For dimensions measuring coverage (`descriptions` and `examples`), a score S is calculated:

$$S = \frac{V}{F} \times 100$$

where V is the count of valid instances and F is the total number of related functions (routes, responses, or parameters). An evaluation is successful if $S \geq 80\%$. This threshold was chosen as a conservative heuristic to ensure significant narrative depth/coverage of the expected content. Table 1 details the mapping between quality dimensions and concrete evaluations.

RQ.1 Summary. We established 17 evaluations to quantify OpenAPI Specification quality, based on the five dimensions defined in the quality vector $\mathcal{Q}(\mathcal{S})$. This realization transitions from binary validation to a nuanced assessment of an API's maturity for both humans and machines.

4.5 RQ.2 - State of Practice: Quality Assessment

To assess documentation maturity, we executed OASQuali across the full dataset of 2,529 public OpenAPI Specifications. For each specification, we operationalize the Composite Quality Index (CQI) introduced in Section 3, assigning a default equal weight $w_d = 0.20$ to each of the five dimensions to ensure a balanced assessment.

Table 1. Set of 17 evaluations formalized from the quality vector $\mathcal{Q}(\mathcal{S})$, to assess the quality of an OpenAPI Specification.

Dimension	Evaluation	Description
$\mathcal{I}$	`validate-json`	Validate JSON serialization standards.
	`validate-oas`	Validate adherence to the OAS metamodel.
$\mathcal{E}$	`oas-version`	Verify version $\geq$ 3.0.0 for modern feature support.
$\mathcal{G}$	`api-title`	Verify presence of the API title field.
	`api-description`	Verify presence of the API-level description.
	`api-contact`	Verify presence of contact metadata.
	`api-version`	Verify presence of a Semantic Version identifier.
	`api-licence`	Verify presence of legal licensing terms.
	`api-terms`	Verify presence of terms of service.
$\mathcal{O}$	`server-url`	Verify presence of server endpoint definitions.
	`server-validity`	Verify validity of defined endpoint URLs.
	`scheme`	Verify enforcement of secure protocols (HTTPS).
$\mathcal{C}$	`route-descriptions`	Validate route description verbosity and keywords.
	`response-descriptions`	Validate response description verbosity.
	`parameter-descriptions`	Validate parameter description verbosity.
	`response-examples`	Verify presence of valid response examples.
	`parameter-examples`	Verify presence of valid parameter examples.

The analysis reveals a mean documentation quality of 67.11%, with individual scores ranging from 28.00% to 96.67%. To investigate the impact of API size (in terms of routes/operations) on documentation standards, the APIs of the dataset were categorized into five percentile-based bins: *micro* (bottom 5%), *small*, *medium*, *large*, and *very large* (top 5%). Small, medium, and large sizes are divided evenly into three bins of the remaining 90%. Due to significant outliers in the *very large* category, specifically three APIs containing 2,958, 11,422, and 22,361 routes respectively, a logarithmic scale was applied to the x-axis for distribution analysis.

As illustrated in Figure 1, quality scores are distributed heterogeneously across all size categories. The results indicate that aggregate documentation quality is independent of API size, within a stable range between $\approx$ 70% for micro APIs and $\approx$ 65% for very large APIs. This suggests that API scale is not directly correlated to overall API documentation quality when considering all five quality dimensions together.

However, a granular dimensional analysis reveals divergent trends as API scale increases:

- **Evolutionary Maturity ($\mathcal{E}$):** OAS versioning quality increases significantly with size (53.10% $\rightarrow$ 88.00%), suggesting that larger APIs are more frequently updated to modern standards.
- **Semantic Communicability ($\mathcal{C}$):** Conversely, semantic quality sharply declines (42.18% $\rightarrow$ 24.00%), highlighting a systematic lack of descriptions and examples in complex APIs. Moreover, the semantic communicability dimension has the lowest score among all dimensions, suggesting an important gap in documented descriptions/examples in REST API documentation.

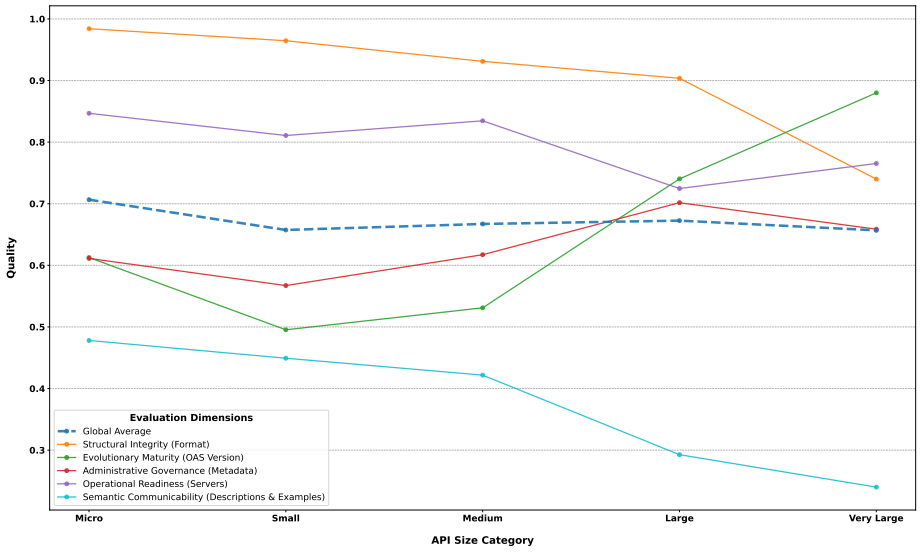

Fig. 1. Distribution of documentation quality across percentile-based API size categories.

– **Structural Integrity ($\mathcal{I}$):** Format-related compliance steadily decreases (98.41% → 74.00%), likely due to the increased maintenance burden and higher probability of syntax errors in large-scale API specifications.

RQ.2 Summary. Public OpenAPI Specifications exhibit a moderate average quality of 67.11%. While aggregate quality is independent of API size, our results suggest that larger APIs tend to prioritize evolutionary standards over semantic clarity. Furthermore, the decline in format compliance among large-scale API specifications suggests that maintenance complexity remains a significant barrier to structural integrity.

4.6 RQ.3 - Critical Pitfalls: Highest Failure Rates

To identify the primary contributors to documentation deficiency, we analyze the failure rates across the 17 deterministic evaluations. Figure 2 illustrates the frequency of failing evaluations by identifier (introduced in Table 1), pinpointing specific evaluations for which compliance is most frequently violated.

The evaluation `evaluate-parameter-examples` (related to parameter examples) indicates that the most frequent failures occur in 86.95% of the analyzed dataset. This maps to a lack of actionable data samples for API parameters. Similarly, the evaluation of route descriptions (`evaluate-route-descriptions`) constitutes the third most frequent failure, with 63.90% of specifications failing to meet the verbosity and keyword thresholds for route-level documentation. These

findings align with the study by Sohan et al. [18], which identified usage examples as a critical component for the effectiveness of REST API documentation.

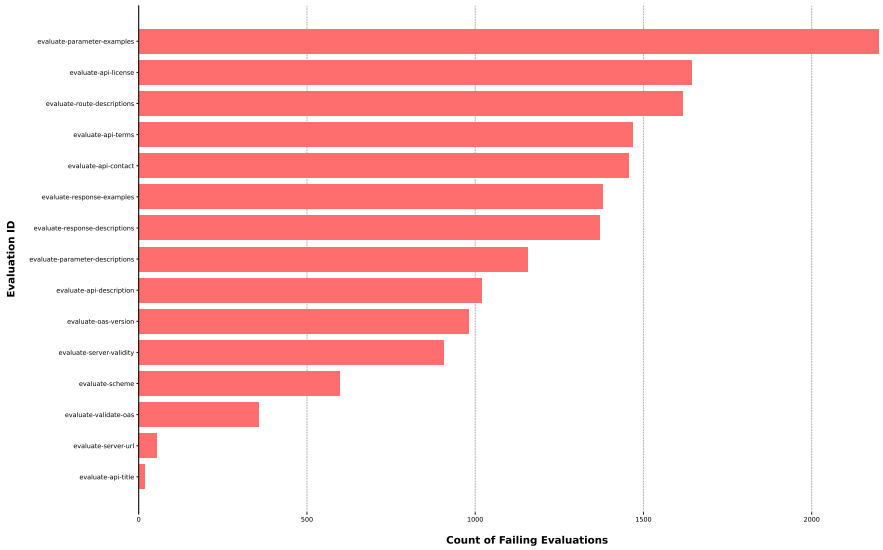

Fig. 2. Frequency of failing evaluations by identifier ($N = 2,529$).

The distribution of failures across the five quality dimensions is summarized in Figure 3. By aggregating the results, we observe that the Semantic Communicability ($\mathcal{C}$) dimension is the primary source of poor documentation quality, accounting for 37.1% of the aggregate failures. This is followed by Evolutionary Maturity ($\mathcal{E}$) (23.6%) and Administrative Governance ($\mathcal{G}$) (22.5%).

While Structural Integrity ($\mathcal{I}$) is the lowest contributor to overall failures (4.3%), it remains a factor in larger specifications. These results demonstrate a significant disparity between machine-readable structural validity and the presence of descriptive and operational metadata.

RQ.3 Summary. Documentation deficiencies are predominantly related to the Semantic Communicability ($\mathcal{C}$) dimension (37.1% of failures). The most frequent individual failures are the absence of parameter examples (86.95%) and insufficient route descriptions (63.90%). These findings indicate that while structural compliance is largely achieved, specifications systematically lack the semantic depth required for operational clarity.

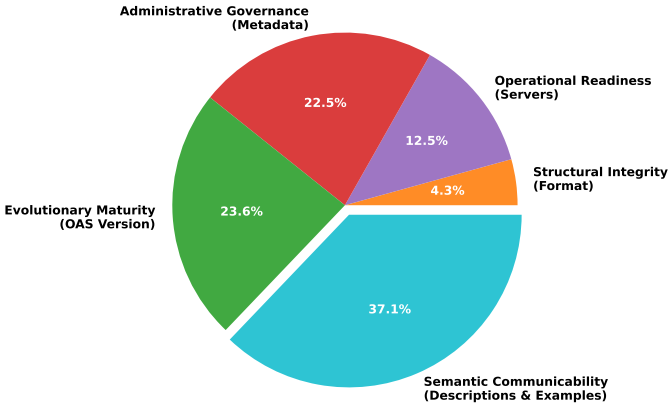

Fig. 3. Breakdown of quality failures categorized by evaluation dimensions.

5 Discussion

The evaluation of 2,529 public OpenAPI Specifications reveals a paradoxical landscape of documentation quality. While aggregate quality metrics appear relatively stable across API size categories, granular dimensional analysis uncovers significant structural shifts that impact the practical utility of these machine-readable specifications.

Evolutionary Standards vs. Semantic Depth. Our analysis demonstrates a distinct prioritization of technical modernization over communicative richness in large-scale APIs. Compliance with Evolutionary Maturity ($\mathcal{E}$) significantly improves with scale, increasing from 53.10% in micro APIs to 88.00% in the very large category. This trend suggests that larger organizations ensure better alignment with modern OAS versions. However, this technical maturity does not imply Semantic Communicability ($\mathcal{C}$), which drops from 42.18% to 24.00% as API size increases. This "semantic gap" indicates that narrative documentation, specifically descriptions and examples, does not proportionally scale with functional complexity. In large ecosystems, documentation appears to be treated as a syntactic requirement rather than a medium for human comprehension or automated consumption.

Maintenance Complexity and Structural Fragility. The observed decline in Structural Integrity ($\mathcal{I}$) among large-scale specifications, dropping from 98.41% to 74.00%, highlights the significant maintenance burden associated with complex specifications. Large APIs are more susceptible to format and schema violations, likely due to the challenges of manual maintenance or the limitations of automated generation tools when dealing with massive object graphs. Given that structural compliance is the primary gatekeeper for most OAS tooling, this decline in integrity for large APIs represents a critical instability in the web engineering

pipeline. Adding this to the observation that 86.95% of all analyzed specifications lack parameter examples, the reliability of downstream automated processes such as testing and mock generation is heavily compromised.

Bottleneck of Operational Clarity. The pervasive failure of semantic evaluations, accounting for 37.1% of all documented quality deficiencies, identifies a significant bottleneck for API usability. The systemic absence of parameter examples (86.95%) and route-level descriptions (63.90%) confirms that most public specifications lack the operational clarity necessary for efficient integration. As noted by Sohan et al. [18], the absence of concrete usage examples directly impacts developer effectiveness. In an engineering context increasingly focused on automated integration, neglecting the semantic layer remains a primary obstacle to achieving high-quality web services.

Tool Limitations. OASQuali focuses on static analysis of OAS files, which may exhibit some limitations. Indeed, the tool does not assess specification and implementation consistency (e.g., verify if all implemented routes are documented, if undocumented endpoints exist, if schemas and HTTP status codes match the actual implementation, etc.). This aspect requires code-level/dynamic analysis, which is not considered in the scope of this work. Moreover, as the tool is mainly static, a successful server availability evaluation does not necessarily mean that the implemented service behaves exactly as documented at runtime. Thus, the tool is better suited as a documentation quality checker than a full API correctness validator.

6 Threats to Validity

The validity of our findings is subject to several threats, which we categorized and mitigated as follows.

Construct Validity. Construct validity concerns the extent to which our 17 evaluations accurately represent the multidimensional quality vector $\mathcal{Q}(\mathcal{S})$. **(1)** To mitigate the risk of arbitrary metric definition, our evaluations are grounded in established architectural principles and an inductive analysis of the OAS standard. **(2)** The application of word-count thresholds and keyword matching in the semantic dimension ($\mathcal{C}$) reduces the risk of treating placeholder text as valid documentation.

Internal Validity. Internal validity refers to the correctness of our analysis pipeline and the accuracy of the resulting metrics. **(1)** A primary threat is the presence of bugs in our tool's implementation. We mitigated this through extensive unit testing and manual cross-validation of results. **(2)** Naive static analysis often fails to resolve complex `$ref` dependencies (cf. Section 3), leading to skewed coverage metrics. To solve this issue, we utilized an OAS library capable of dereferencing specifications by replacing references with their actual data.

External Validity. External validity relates to the generalizability of our findings. **(1)** While the APIs.guru directory provides high heterogeneity, it may not

fully represent private enterprise APIs or niche domains. We mitigated this by analyzing over 2,529 specifications, ensuring a statistically significant sample size across various scales and industries. **(2)** Our study is restricted to the OpenAPI Specification. While OAS is the de facto industry standard, our findings may not extend to other formats like RAML or Blueprint. **(3)** API specifications are dynamic entities. Since our dataset reflects a static snapshot from February 2026, subsequent updates may have resolved some identified quality deficiencies.

7 Conclusion and Future Work

We introduced OASQuali, a deterministic tool that assesses OpenAPI Specification (OAS) quality, which aims at documentation usability rather than structure. By formalizing quality through a 5-dimensional vector $\mathcal{Q}(\mathcal{S}) = \langle \mathcal{I}, \mathcal{E}, \mathcal{G}, \mathcal{O}, \mathcal{C} \rangle$ and defining a set of 17 quality evaluations related to these dimensions, we conducted a large-scale empirical study on 2,529 public specifications extracted from the APIs.guru directory. Our analysis revealed a mean global quality of 67.11%. Crucially, while aggregate quality remains independent of API size, we identified a significant "semantic gap" in large-scale implementations: as APIs expand, evolutionary compliance increases (53.10% $\rightarrow$ 88.00%), yet semantic communicability drops (42.18% $\rightarrow$ 24.00%). The systemic absence of parameter examples (86.95%) and route-level descriptions (63.90%) remains the primary bottleneck for both API user experience and automated integration. Overall, the multi-faceted approach of our tool supports REST API governance processes [1]. Future work will follow three research axes:

1. **LLM Integration:** We aim to leverage Large Language Models (LLMs) to automatically generate missing semantic metadata (i.e., descriptions and examples) identified by OASQuali. We plan to extend our prior work [6] in this direction to bridge the observed semantic gap. Moreover, domain-specific ontologies could be included with the help of LLMs, so that semantic relevance may be checked in a more realistic way.
2. **Quality Over Time:** We plan to track the evolution of quality metrics over time, assessing how documentation matures across successive API versions.
3. **Cross-Format Generalization:** We intend to extend our tool to support alternative Interface Description Languages (IDLs) such as RAML and API Blueprint, enabling a comparative study of quality across documentation standards for REST APIs.

Use of Generative AI. Generative AI was used for limited rephrasing purposes in this paper. The authors have carefully reviewed the final content of the paper and take full responsibility for it.

Disclosure of Interests. The authors have no competing interests related to the content of this article.

Acknowledgments. Gilles Perrouin is an FNRS Research associate.

References

1. Ahmad, M., Geewax, J.J., Macvean, A., Karger, D., Ma, K.L.: API Governance at Scale. In: Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Practice. p. 430–440. ICSE-SEIP '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3639477.3639713
2. APIs.guru: APIs.guru - Wikipedia for Web APIs (2026), https://apis.guru
3. Bogner, J., Kotstein, S., Abajirov, D., Ernst, T., Merkel, M.: RESTRuler: Towards Automatically Identifying Violations of RESTful Design Rules in Web APIs. In: 2024 IEEE 21st International Conference on Software Architecture (ICSA). pp. 123–134 (2024). https://doi.org/10.1109/ICSA59870.2024.00020
4. Bogner, J., Wagner, S., Zimmermann, A.: Collecting service-based maintainability metrics from restful api descriptions: static analysis and threshold derivation. In: European Conference on Software Architecture. pp. 215–227. Springer (2020). https://doi.org/10.1007/978-3-030-59155-7_16
5. Corradini, D., Montolli, Z., Pasqua, M., Ceccato, M.: DeepREST: Automated Test Case Generation for REST APIs Exploiting Deep Reinforcement Learning. In: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering. p. 1383–1394. ASE '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3691620.3695511
6. Decrop, A., Devroey, X., Papadakis, M., Schobbens, P.Y., Perrouin, G.: You can rest now: Automated rest api documentation and testing via llm-assisted request mutations (2025), https://arxiv.org/abs/2402.05102
7. Decrop, A., Vandeloise, M.: OAS Quality Tool (2026), https://github.com/alixdecr/oas-quality-tool
8. Fielding, R., Nottingham, M., Reschke, J.: RFC 9110: HTTP Semantics (2022), https://www.rfc-editor.org/rfc/rfc9110.html
9. Fielding, R.T.: Architectural styles and the design of network-based software architectures. University of California, Irvine, Irvine, USA (2000)
10. Hatfield-Dodds, Z., Dygalo, D.: Deriving semantics-aware fuzzers from web API schemas. In: Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings. p. 345–346. ICSE '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3510454.3528637
11. Hosono, M., Washizaki, H., Fukazawa, Y., Honda, K.: An empirical study on the reliability of the web api document. In: 2018 25th Asia-Pacific Software Engineering Conference (APSEC). pp. 715–716 (2018). https://doi.org/10.1109/APSEC.2018.00103
12. Massé, M.: REST API design rulebook: designing consistent RESTful web service interfaces. " O'Reilly Media, Inc." (2011)
13. Moon, S.Y., Kerr, G., Silavong, F., Moran, S.: Api-miner: an api-to-api specification recommendation engine. In: Proceedings of the 1st IEEE/ACM Workshop on Software Engineering Challenges in Financial Firms. p. 9–16. FinanSE '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3643665.3648049
14. Serbout, S., Lauro, F.D., Pautasso, C.: Web apis structures and data models analysis. In: 2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C). pp. 84–91 (2022). https://doi.org/10.1109/ICSA-C54293.2022.00059

15. Serbout, S., Pautasso, C.: Apistic: A large collection of openapi metrics. In: Proceedings of the 21st International Conference on Mining Software Repositories. p. 265–277. MSR '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3643991.3644932
16. Singjai, A., Zdun, U.: Api description-based conformance assessment of architectural design decision. In: 2022 IEEE International Conference on Service-Oriented System Engineering (SOSE). pp. 59–68 (2022). https://doi.org/10.1109/SOSE55356.2022.00013
17. SmartBear Software: Swagger Editor (2026), https://editor.swagger.io
18. Sohan, S.M., Maurer, F., Anslow, C., Robillard, M.P.: A study of the effectiveness of usage examples in rest api documentation. In: 2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). pp. 53–61 (2017). https://doi.org/10.1109/VLHCC.2017.8103450
19. The Linux Foundation: OpenAPI Initiative (2026), https://www.openapis.org