AN API MANAGEMENT SOFTWARE QUALITY METAMODEL BASED ON SQUARE AND GQM

Data de submissão: 03/03/2025 *Data de aceite: 05/03/2025*

Eder dos Santos
Instituto de Tecnología Aplicada, Universidad Nacional de la Patagonia Austral, Argentina
CIT Santa Cruz, CONICET, Argentina

Sandra Casas
Instituto de Tecnología Aplicada, Universidad Nacional de la Patagonia Austral, Argentina
CIT Santa Cruz, CONICET, Argentina

ABSTRACT: With the widespread adoption of API in modern software ecosystems, the need for robust API management practices has become increasingly apparent. However, challenges persist in establishing a comprehensive framework for software product quality. To address this, this work introduces API-MQM, a tailored metamodel that leverages the ISO/IEC 25010 standard as a reference framework and employs the Goal-Question-Metric (GQM) paradigm. The design method adhered to Model-Driven Architecture (MDA) principles and emphasized the identification of API management capabilities as core quality requirements. To validate the proposal, both theoretical and empirical methods were conducted. The findings underscore the novelty and consistency of the proposed approach and outline directions for future research in the field.

KEYWORDS: API Management, Quality metamodel, Quality model, Software Engineering, Software Quality

1 I INTRODUCTION

In recent years, the distribution models for information systems have shifted towards Everything as a Service (XaaS) [1] paradigms, where organizations provide their digital assets to customers as services [2]. These services are commonly supported by APIs that adhere to REST (Representational State Transfer) principles [3]. As a result, APIs have experienced significant global proliferation. According to the Postman annual survey [4], 1.29 billion API requests were generated in 2023 by more than 25 million users worldwide.

The emergence of the API Economy scenario has placed additional pressure on developing, deploying, and maintaining information systems. Since APIs have become a critical component [5], organizations need to proactively address the risks of failure by enhancing their API management capabilities [6] [7]. This involves the effective management of their APIs through specialized software products, commonly referred to as API Management Platforms.

API Management Platforms offer fundamental functionalities to create, analyze, and manage APIs in a secure and scalable ecosystem, serving as the core of digital integration strategies. These platforms are equipped with features such as API access control, comprehensive documentation, and monitoring and usage analytics. As a result, they streamline API management processes [7] [8] [9].

API management activities present numerous challenges within both internal [6] [10] and decentralized [11] software ecosystems. To address these challenges, many software quality models define a set of quality attributes or characteristics, establishing the groundwork for evaluating the quality of software elements. This evaluation is pivotal in providing actionable solutions that can be readily embraced by industry professionals to enhance the quality of software ecosystems. From a metamodel-driven standpoint, models are developed based on metamodels.

To our knowledge, despite the proliferation of software quality metamodels in current literature, a notable gap exists in the exploration of API management software quality metamodels. This void motivated the development of this work. We propose an API management-specific software quality metamodel designed to address quality evaluation from conceptual, operational, and quantitative perspectives. Grounded in the ISO 25000 standards series (particularly the ISO/IEC 25010:2023, ISO/IEC 25023:2016, and ISO/IEC 25040:2011 standards [12] [13] [14]), our metamodel provides a structured framework for defining quality characteristics. To facilitate the specification of measures, we also employ the Goal-Question-Metric (GQM) [15] approach. Additionally, the proposed metamodel encompasses API management capabilities, thereby organizing API management activities within a cohesive framework.

This paper is structured as follows. In section 2, we discuss the frameworks we adopted and provide a brief overview of the metamodel design along with its associated challenges. Section 3 then introduces a three-step methodology for constructing a general software quality metamodel, based on the principles of Model Driven Architecture and concept factoring. In section 4, we present the designed metamodel and provide a comprehensive breakdown of the findings. Section 5 details both the theoretical and empirical validation of the metamodel. Finally, in section 6, we offer a succinct discussion that analyzes the proposed metamodel from different perspectives and provides insights into future research.

2 I RELATED WORK

2.1 The ISO 25000 standards series

As APIs are used across an increasingly wide range of application areas, the development or selection of high-quality API management software products becomes paramount. Therefore, thorough specification, measurement, and evaluation are pivotal to ensure adequate quality.

The SQuaRE series is structured into several divisions. Within the scope of this paper, the relevant divisions are outlined as follows:

- The Quality Model Division (2501n) provides models for system and software product quality, quality in use, and data quality. The general software product quality model [12] encompasses nine primary quality characteristics, namely: functional suitability, performance efficiency, compatibility, interaction capability, reliability, security, maintainability, flexibility, and safety.
- The Quality Measurement Division (2502n) includes a system and software product quality measurement reference model, definitions of quality measures, and practical guidance for their implementation. The framework for quality measurement is illustrated in [13].
- The Quality Evaluation Division (2504n) offers requirements, recommendations, and guidelines for system and software product evaluation. An overview of the evaluation process is presented in [14].

2.2 The GQM approach

Quality definition, measurement, and evaluation must be accurate and meaningful to provide valuable insights according to the needs of users and organizations. In this context, the Goal Question Metric (GQM) paradigm [15] is widely used for defining metrics. GQM enables the identification of relevant data collection criteria and the establishment of interpretive mechanisms, creating a goal-oriented framework for software measurement.

The GQM approach operates at three hierarchical levels: conceptual (goals), operational (questions), and quantitative (metrics). Goals define the focus of the research endeavor and the reason for its study. Questions characterize specific aspects of the measurement object with respect to a selected quality concern. Metrics comprise a collection of measurements that can be used to address the formulated questions, drawn from both subjective and objective collected data.

2.3 API management capabilities

Based on recent literature, API Management quality requirements are delineated and distilled primarily as practices and capabilities [7] [8] [16]. Practices are defined as any activities explicitly aimed at improving, fostering, and overseeing API usage, while capabilities are defined as the capacity to accomplish specific goals related to API Management by executing two or more interconnected practices.

API management capabilities are categorized across diverse schemes. For instance, the classification proposed by [7] includes the following capability groupings: i. Developer Enablement for APIs (API Discovery, Developer and App Onboarding, Collaboration and Community, Developer Enablement Administration); ii. Secure, Reliable and Flexible Communications (Authentication and Authorization, Threat Detection, Data Privacy, Traffic Management, Interface Translation, Service Orchestration and routing); iii. API Life cycle Management (API Publication, Version Management, Change Notification, Issue Management); iv. API Auditing, Logging and Analytics (Activity Logging, User Auditing, Business Value Reporting, Contract Management, Advanced Analytics, Service-Level Monitoring).

Recent research has focused on characterizing quality aspects related to API management capabilities and activities as quality requirements, using the ISO/IEC 25010 standard series as a reference model. From the practitioners' perspective, functional suitability, security, reliability, and performance efficiency are critical factors that align closely with user needs and the intended functionality of API management platforms [17]. It was also identified in the scientific current research [18].

2.4 Software quality metamodels

To establish a comprehensive quality model, several authors recommend overcoming ambiguity and completeness issues by defining a formal quality metamodel. As suggested by [19], a quality metamodel can be described as a flexible and user-friendly collection of constructs and rules designed to facilitate the construction of quality models on a formal basis. The overarching goal is to foster a shared understanding for effective evaluation and management of software quality across the entire lifecycle of a software product [20].

Metamodels serve as representation schemes at a conceptual level, often depicted through diagrams that encompass entities and their relationships. Software quality metamodels commonly feature entities pertinent to the definition, measurement and evaluation of quality such as 'quality attribute', 'quality characteristic', 'measure', and 'metric', among others.

From a metamodel-driven perspective, the initial step involves developing a metamodel to establish the concept, scope, and vocabulary, providing a static holistic view of a domain. Subsequently, a quality model is formulated, delineating a set of quality characteristics as instances of the elaborated metamodel [21]. Furthermore, numerous researchers have introduced metamodels grounded in existing quality models [20].

From another perspective, software quality models and metamodels can be classified based on their purpose as either basic or tailored [20]. Basic models, such as [12], have a hierarchical structure adaptable to various types of software products and focus on evaluation and improvement. Tailored models, on the other hand, are specific to particular domains or applications, where the importance of features may vary compared to a general model. According to a systematic review conducted by [20], tailored models are often derived from basic models, involving the addition or modification of sub-factors to meet the specific needs of specialized domains or applications.

Specifically in the field of API management software quality, the systematic mapping study by [18] highlights that more than half of the research does not explicitly utilize formal models. However, it also underscores that common meta-model elements, such as features and metrics, are widely employed across these studies. These findings emphasize the need for the proposed metamodel in this paper, which aims to establish a common framework to enable more formal and cohesive research in the field.

3 I METHODS

Drawing upon principles from model-driven architecture, we have developed a structured four-step approach aimed at creating a unified metamodel named API-MQM (API Management Quality Metamodel). In this section, we detail each step of this approach, elucidating its complexities and nuances.

Step 1 – Design Questions and Preliminary Analysis

The initial phase involved conducting a preliminary analysis by reviewing a representative collection of existing software quality metamodels. To achieve this objective, we updated the dataset provided by [20] to include API-related metamodels. This approach followed the systematic search protocol outlined in guidelines such as [22, 23, 24]. Subsequently, we classified the updated data based on relevant criteria.

This exploration aimed to address a series of key design questions (DQs), with the primary objective of providing guidance for either adopting an existing metamodel or developing a new one. The DQs are outlined as follows:

DQ 1: Which software quality metamodels address APIs in general and API management in particular?

This DQ aims to identify existing metamodels that specifically address the quality attributes of APIs and API management. Understanding which metamodels are used helps in assessing the current landscape and determining whether these metamodels sufficiently cover the particular aspects of API quality. This can highlight gaps in the existing literature and establish a baseline for developing a more comprehensive metamodel tailored to both APIs and API management.

DQ 2: What are the most common elements in software quality metamodels?

By identifying the most common elements across various software quality metamodels, this DQ seeks to uncover the fundamental components and attributes that are widely recognized as critical for evaluating software quality. This information is crucial for ensuring that the proposed metamodel includes relevant and widely accepted elements, thus enhancing its applicability and effectiveness.

DQ 3: Which software quality models are used as a reference for each metamodel?

This DQ explores the foundational quality models that influence the development of various metamodels. Understanding which models are referenced helps to identify the theoretical underpinnings and best practices that shape these metamodels. It provides insight into how well-established quality principles are integrated into the metamodels and whether they align with current standards and practices in software quality characterization and measurement.

DQ 4: What frameworks are employed as references for evaluation in the metamodels analyzed in the study?

This DQ focuses on the frameworks used for evaluating software quality evaluation. By examining the frameworks referenced for evaluation, the study aims to understand the criteria and methodologies applied to assess software quality. This helps in evaluating the robustness of existing frameworks and identifying potential improvements or alternative approaches that could be adopted in the proposed metamodel.

Step 2 – PIM Construction and Concept Factoring

To address this step, the guidelines outlined by [19] where followed to ensure clarity, coherence, and consistency in capturing key concepts, while avoiding ambiguity and resolving potential contradictions. The approach to constructing the general Platform-Independent Model (PIM) involved several key steps. Firstly, we identified the most common elements drawn from the data gathered in DQ2 findings. Subsequently, we analyzed the most frequently referenced model in the literature (from DQ3 data) and the prevailing evaluation method (from DQ4 data), conducting an in-depth analysis of their elements and their relationship with the elements found in DQ2. Finally, to achieve a tailored metamodel, we integrated API management capabilities as a factored concept.

Step 3 – Building the Metamodel

The metamodel was constructed using the Unified Modeling Language (UML), since it defines a standardized framework for designing elements and specifying relationships between them.

Step 4 – Metamodel validation

As highlighted by [24] [25], metamodels can be validated through both theoretical and empirical methods. In the quality metamodel proposed in this work, the focus is on API management capabilities, where quality characteristics are evaluated. These characteristics are assessed using specific metrics, which are designed to address questions aligned with the overall goals. This process is further facilitated by tools that automate the measurement and evaluation procedures. To validate this proposal, three validation methods were conducted.

From the theoretical perspective, it is noteworthy that many design challenges (DC) are usually faced while creating metamodels [26]. These challenges often represent a series of significant design and scope limitations. As this paper presents a design proposal, a construct validation was conducted in light of these challenges, aiming at validating the designed metamodel and identifying weaknesses and research opportunities.

To substantiate the feasibility and effectiveness of the proposed metamodel, two empirical validation methods were conducted, namely analysis [24] and example [24]. First, a comprehensive literature mapping and comparative analysis of existing metamodels was performed, juxtaposing them with the developed metamodel in this study. This analysis focused particularly on examining the metamodel elements and their application in existing literature. Finally, to assess the practical applicability of the proposed approach, an instance of the metamodel is presented.

4 I RESULTS

We identified and classified a total of 29 software quality metamodels. Of these, 28 studies were retrieved from [20]. An additional study [21] was discovered during the execution of the proposed search protocol across major global databases, including IEEEXplore, SpringerLink, ACM Digital Library, and ScienceDirect. These metamodels were systematically classified to provide insights into the DQs as outlined below.

4.1 Design Questions

DQ 1: Which metamodels address API in general and API management in particular?

Findings indicate that while one metamodel addresses APIs in a general context, none specifically focus on API management. Additionally, we observed that six metamodels include considerations for API-related technologies, specifically webservices and microservices. Based on these findings, we proceeded to address the remaining design questions. Findings are outlined as follows.

DQ 2: What are the most common elements in software quality metamodels?

14 categories of metamodel elements were identified. Four of them encompass aggregated entities, following the UML aggregation concept. Table 1 showcases the five most prevalent metamodel elements. The "Metamodel Element" column lists the different elements within the metamodel. Alternative terms or synonyms for each metamodel element are provided for flexibility and clarity. The "Aggregated Elements" column indicates related elements or subcategories grouped under each metamodel element. Finally, the "Frequency" column indicates the prevalence of each element, denoted by the number of metamodels referencing it.

Metamodel Element	Most common synonims	Aggregate elements	Frequency
Quality Attribute	Characteristic / Feature / Factor	Subcharacteristic	28
Metric	Measure	Base Metric / Derived Metric	24
Entity	Component / Artifact	- - -	19
Evaluation	Assessment	Formula / Rule	15
Measurement	Measurement Method / Approach	Result / Value	14

Table 1. Common elements of software quality metamodels.

DQ 3: Which software quality models are taken as reference for the metamodel?

To address this question, we processed the available data, recognizing that it was not normalized. The absence of normalization stemmed from the presence of numerous metamodels that referenced more than one quality model. Table 2 presents the five most frequently cited models.

Within the examined studies, it's significant to note that both ISO/IEC 9126 and its successor, the ISO/IEC 25000 series, were both referenced in three studies. This indicates that a total of 21 studies (72.41%) utilized both series as reference models.

Reference Model	Metamodels	% of total
ISO/IEC 9126	16	55.17%
Boehm	9	31.03%
McCall	8	27.59%
ISO/IEC 25010	8	27.59%
Dromey	6	20.69%

Table 2. Quality models in metamodels.

DQ 4: What methods/techniques are used as reference for evaluation in the metamodel?

Table 3 reveals that nearly half of the studies did not utilize any formal evaluation framework. However, among the works that explicitly incorporated evaluation methods, 80% (12 out of 15) opted for the GQM approach [15].

Evaluation Method	Metamodels	% of total
Not specified	14	48.28%
GQM	11	37.93%
FCM	2	6.90%
SQUID	1	3.45%
GQM and FCM	1	3.45%

Table 3. Reference evaluation methods.

4.2 Building the Metamodel

To build the metamodel, we integrated the factored concepts derived from DQ2 common elements, following the standards outlined in ISO/IEC SQuaRE, and drew upon GQM definitions, which serve as prevalent quality models and evaluation methodologies. To tailor the metamodel to the specific context of API management, we introduced API management capabilities as a factored concept. Furthermore, we extended the initial framework to encompass considerations for the development and adoption of tools as a factored concept, as such tools are designed to retrieve measured data.

To provide clarity and transparency regarding the origins and definitions of the entities comprising the API-MQM metamodel, Table 4 showcases API-MQM elements represented as Entities, along with their corresponding sources in the literature.

API/MQM Entity	Source
API Management Capability	[7] [8] [16]
Characteristic	[12]
Goal	[15]
Question	[15]
Metric	[13] [15]
Measurement Method	[13]
Tool	[14]

Table 4. API/MQM Entities.

The selected concepts have been incorporated into the designed metamodel, as depicted in Figure 1. This metamodel organizes the concepts into three distinct packages that are interconnected, as outlined below. Within the Package Domain, we encapsulated concepts related to API management capabilities and the conceptual perspective of [15] (Goals). It is noteworthy that API management capabilities serve as the foundation for instantiating quality requirements within the metamodel. Package System contains concepts related to software product quality and the GQM operational and quantitative levels. It is noteworthy that Metric is a factored concept from both [13] (Measure) and [15] (Metric). The Package System encompasses concepts pertaining to software product quality and operates at both the operational and quantitative levels of [15]. Notably, the concept of Metric is derived from both [13] (referred to as Measure) and [15] (referred to as Metric). Package Development refers to the amalgamation of measurement methods and the utilization of both adopted and custom-developed tools for measurement purposes.

5 I VALIDATION

5.1 Theoretical Validation

Next, a synthesis of the DC were addressed in the approach of this work.

DC1 – Terminology inconsistencies: To address this challenge, we established a theoretical foundation, drawing upon established works such as This challenge was addressed by adopting a theoretical ground [7] [12] [15] [20] to avoid terminology inconsistencies. We also adopted the most common elements used in a set of 29 software quality metamodels.

DC2 – Partial definition: In pursuit of our study's objective to offer a thorough analysis of quality definition and measurement, the proposed metamodel deliberately omitted guidelines for evaluation and decision criteria. Other elements were also omitted due to addressing challenges DC3 and DC6.

DC3 – Lack of focus: To address this challenge, our approach focused on limiting the scope of stakeholders and other levels of abstraction, especially concerning aggregated entities such as base metrics and derived metrics, among others. While our coverage may be limited in this regard, we ensured the implementation of the most common elements. However, it's important to note that this issue was further investigated and elaborated upon in the empirical validation phase of our study.

DC4 – Disregard for process quality: Although the designed metamodel covers the development and adoption of tools to measure quality, it is important to clarify that this paper does not delve into the specifics of adopting or developing products in a manner that it assures a desired quality level.

DC5 – Lack of integration with current practices: This work did not delve into providing solutions for this particular challenge, as it falls outside the scope of our study.

DC6 – Lack of simplification and validation: To address this concern, all conceptual redundancies were meticulously eliminated. Then, an analysis of design challenges was performed to serve as a theoretical validation, complemented by a comprehensive literature mapping for empirical validation. Additionally, we offered recommendations for future validation methods to further strengthen our findings.

DC7 – Interdependencies and measure interpretations not clear: Despite resolving interdependencies through the design methodology outlined in this paper, the proposed metamodel lacks to provide guidance on interpreting measurement results.

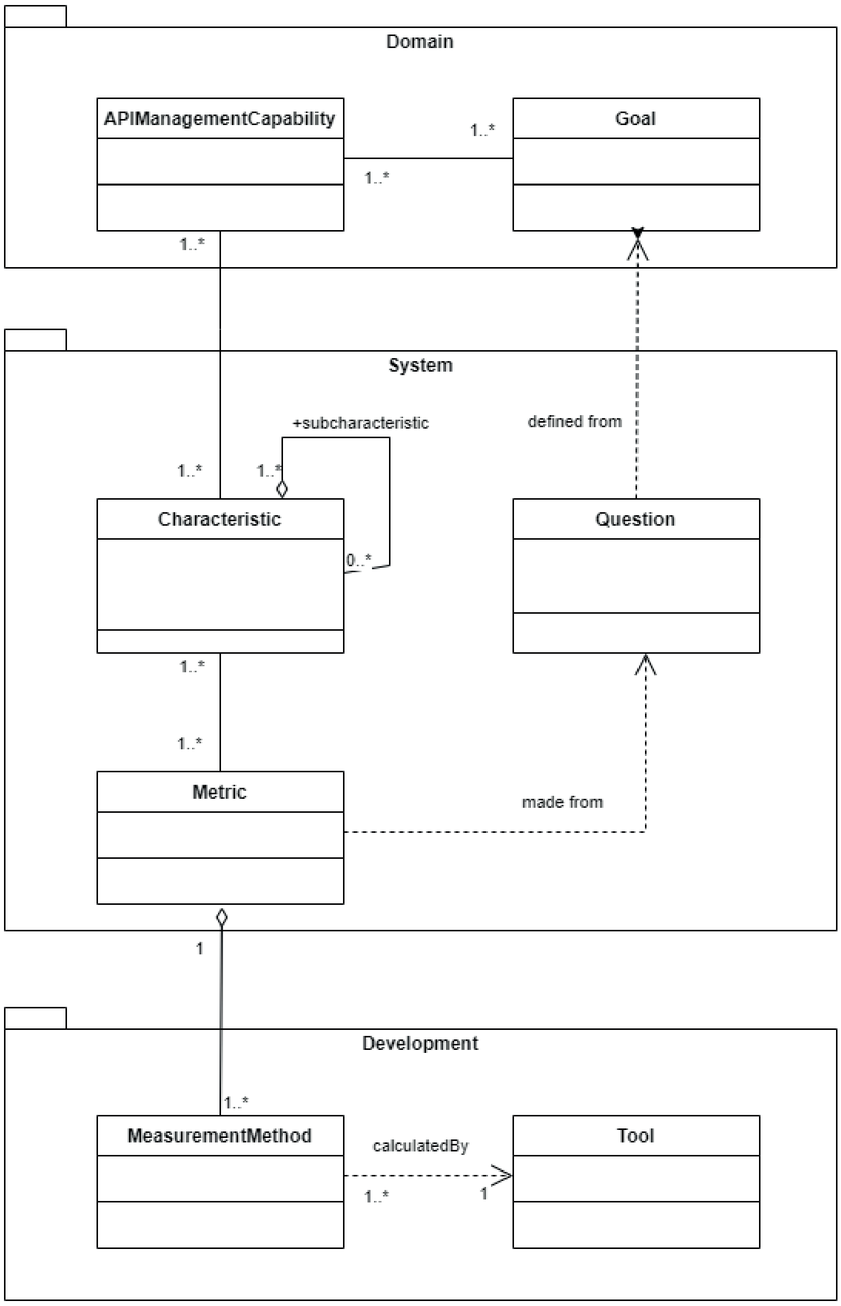


Figure 1: API-MQM Quality Metamodel for API Management UML representation

DC8 – Lack of tool support: The developed meta-model includes entities that refer to implemented or custom-made tools for automating the measurement process within API Management platforms.

DC9 – Lack of guidelines for improvements: To address this challenge, a series of future work was provided in this paper.

5.2 Empirical validation

A comparative analysis is summarized in Tables 5 and 6.

Upon initial examination, Table 5 presents a comprehensive overview of the entities defined in API-MQM, along with their corresponding equivalents within existing software quality metamodels. It also includes the frequency of occurrence of each element and its alternative names. Noteworthy is the alignment between the elements proposed in API-MQM and those commonly identified in software quality metamodels.

API-MQM Entity	Equivalent Common Element	Frequency
API Management Capability	Quality Requirement	3 (10.34%)
Characteristic	Quality attribute	28 (96.55%)
Goal	Quality goal	8 (27.59%)
Question	Quality aspect	12 (41.38%)
Metric	Metric	24 (82.76%)
Measurement Method	Measurement	14 (48.28%)
Tool	Instrument	5 (13.79%)

Table 5. Frequency of API-MQM entities within software quality metamodels.

On the other hand, table 6 highlights common metamodel elements absent from the API-MQM metamodel. Specifically, categories such as “Entity”, “Unit”, “View”, “Scale”, and “Quality Model” were omitted due to challenge C6, particularly stemming from the need for simplification. Additionally, elements related to “Evaluation” and “Data Analysis” were excluded, as the primary focus of this work lies in the introduction of a metamodel aimed at guiding the definition and measurement of quality elements.

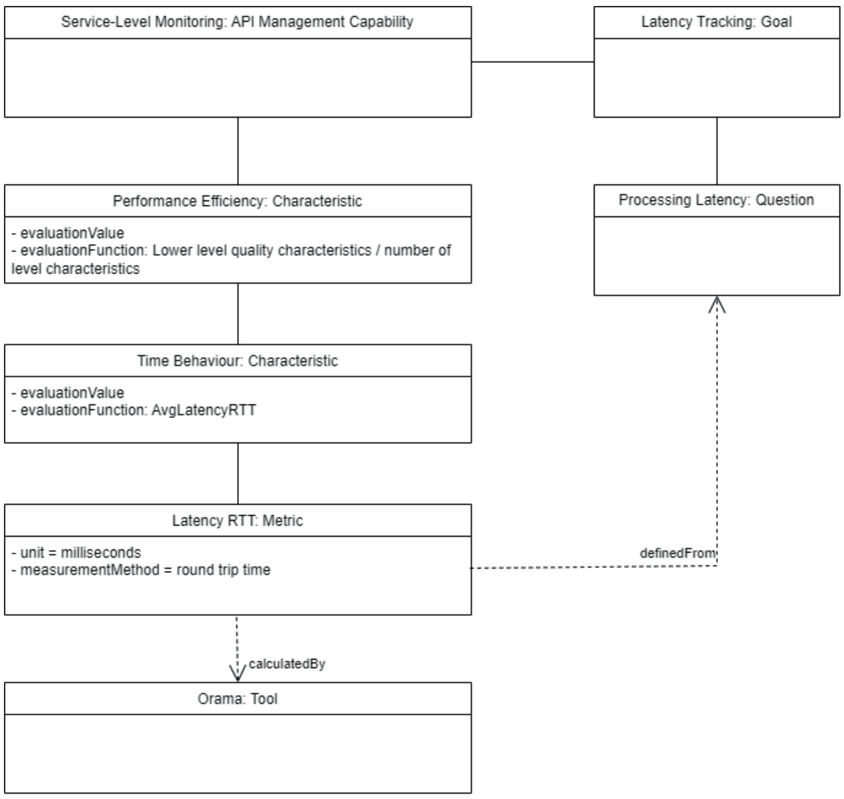
Element	Most common synonyms
Entity	Component, Artifact
Evaluation	Assessment model
Quality model	- - -
Unit	Measurement unit
Data analysis	Analysis model, Decision criteria
View	Viewpoint, Stakeholder
Scale	Measurement scale

Table 6. Elements not addressed within API-MQM.

Additionally, an instance of the proposed metamodel is introduced as shown in Figure 2. This instance can be summarized as follows: Within the Domain package, the Service-Level Monitoring capability [7] is selected as the target API management feature, with Latency Tracking defined as the conceptual perspective (Goal) [15] This goal is further described as "tracking network performance of the API management software to ensure efficient data transmission". In the System package, Time Behavior is chosen as a sub-characteristic of Performance Efficiency, in accordance with [12]. From an operational perspective, the objective is to define the method for measuring processing latency, addressing the question: "How is processing latency measured?"

From the obtained question, a metric was defined: Latency RTT, which stands for Latency Round Trip Time. Round Trip Time (RTT) is the duration it takes for a data packet to travel from the source to the destination and back again to the source. This metric provides a precise measurement of the network latency experienced during API requests, reflecting the efficiency of data transmission in the API management software. This metric allows for the definition of the evaluation function for the "Time Behavior" quality characteristic. The evaluation function is derived by calculating the average latency from a given set of *n* concurrent API requests, providing a comprehensive measure of the system's performance efficiency. Therefore, the evaluation function *F* is defined by Eq (1). Finally, the Orama Framework, which is instantiated as a Tool object in this work, is introduced as" a support tool for evaluating Function-as-a-Service-oriented environments" by [27].

$$F = \sum_{i=1}^{n} \frac{LR_i}{n} \quad (1)$$



Service-Level Monitoring: API Management Capability
Latency Tracking: Goal
Performance Efficiency: Characteristic
- evaluationValue
- evaluationFunction: Lower level quality characteristics / number of level characteristics
Processing Latency: Question
Time Behaviour: Characteristic
- evaluationValue
- evaluationFunction: AvgLatencyRTT
Latency RTT: Metric
- unit = milliseconds
- measurementMethod = round trip time
definedFrom
calculatedBy
Orama: Tool

Figure 2: Service-Level Latency Monitoring.

6 | DISCUSSION

The systematic integration of quality considerations into the API Management domain is mandatory to set the focus on prevention and to foster continuous improvement. By providing practitioners with quality models that help to assure acceptable degrees of quality, it is expected to enhance the likelihood of acceptance for a particular API Management Platform. To achieve it, designing a tailored API Management software quality metamodel and its associated models represents a crucial endeavor in sustaining continuous quality management and assurance.

This paper presented two main contributions. Firstly, it offered an in-depth examination of the key challenges associated with designing quality metamodels in Software Engineering, along with an updated survey of the current terminology, relevant standards, and evaluation methods employed in the field. Subsequently, API-MQM was introduced as a bespoke software quality metamodel tailored for API Management platforms. Our decision to undertake this endeavor was motivated by the absence of existing metamodels within the domain, necessitating the development of a customized solution to address the identified gaps. To accomplish this, we embraced the prevailing de facto frameworks, namely ISO/

IEC 25000 and GQM, given their significant influence and widespread adoption within the software engineering community for designing software quality metamodels.

Software quality models traditionally prescribe a static set of characteristics and their interrelationships so that they should rather be selected dynamically based on stakeholder needs. Thus, software quality models may not always align with the evolving needs of stakeholders. To address this limitation, metamodels emerge as a flexible solution to help building quality models in a formal basis.

Also important, the use of a metamodel allows to discuss the continuing quality characteristics in the future. This flexibility ensures the relevance and applicability of the quality framework across different contexts and evolving standards. As an example, we suggest API-MQM is suitable for different versions of [12], namely 2011 and 2023 versions. Also, API-MQM support API Management capabilities that can vary in their classification as software and stakeholders needs evolve constantly.

Finally, subjecting API-MQM to both theoretical and empirical assessments allow to suggest that the proposed metamodel met the primary domain requirements. It also incorporated various common metamodel elements identified in existing literature and effectively addressed several well-documented design challenges. These concluding remarks affirm the robustness and relevance of the developed metamodel within the context of API management. model should encompass all elements referenced in the metamodel across different levels of abstraction. This approach would provide further concrete evidence of the metamodel's utility in real-world scenarios and enhance its practical value in software engineering contexts.

7 | FUTURE WORK

While this study has identified certain limitations, two main future research directions are proposed.

Primarily, the development of an empirically validated quality model and its comprehensive evaluation stands as a crucial next step to in-depth validate the applicability, feasibility, and effectiveness of the metamodel. Such a

Finally, this metamodel can be extended and improved by addressing various design challenges such as: i. adding entities and guidelines for evaluation and decision criteria; ii. incorporating aggregated entities to enhance semantics; iii. specifying desired quality levels for measured elements; iv. Providing guidance on developing and adopting software products to ensure a desired quality level; and v. assessing and recommending integration strategies for incorporating this quality metamodel and its derived quality models into existing quality management practices within organizations. We suggest this roadmap will serve as a guiding framework to explore unaddressed challenges and refine the proposed approach. By identifying key areas warranting further investigation, we aim to fortify the theoretical and empirical foundations of our proposal and contribute to advancing the field of API management quality.

COMPETING INTERESTS

The authors have declared that no competing interests exist.

FUNDING

The first author is supported by CONICET through an internal PhD scholarship, which underscores his commitment to and expertise in their research. This support validates the research group's plan and the author's qualifications, further ensuring the successful advancement of the proposed project.

AUTHORS' CONTRIBUTION

The authors confirm contribution to the paper as follows. EDS: Conceptualization, Investigation, Writing-Original draft preparation, Validation, Writing-Reviewing and Editing; SC: Methodology, Writing-Reviewing and Editing; Supervision; Validation. All authors reviewed the results and approved the final version of the manuscript.

REFERENCES

[1] Y. Duan, G. Fu, N. Zhou, X. Sun, N. C. Narendra, and B. Hu, "Everything as a service (XaaS) on the cloud: Origins, current and future trends", in 2015 IEEE 8th International Conference on Cloud Computing, 2015, pp. 621–628.

[2] C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, "Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications". Vienna: Springer Vienna, 2014.

[3] R. T. Fielding, "Architectural styles and the design of network-based software architectures", 2000.

[4] 2023 state of the API report, p. 58, 2023. [Online]. Available: https://voyager.postman.com/pdf/2023-state-of-the-api-report-postman.pdf

[5] J. Bloch, "How to design a good api and why it matters", in Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications, ser. OOPSLA '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 506–507.

[6] S. Andreo and J. Bosch, "Api management challenges in ecosystems," in Software Business, S. Hyrynsalmi, M. Suoranta, A. Nguyen-Duc, P. Tyrvainen, and P. Abrahamsson, Eds. Cham: Springer International Publishing, 2019, pp. 86–93.

[7] B. De, API management: an architect's guide to developing and managing APIs for your organization, first edition ed., ser. For professionals by professionals. Berkeley, CA: Apress, 2017.

[8] S. Preibisch, Api development: a practical guide for business implementation success. New York, NY: Springer Science+Business Media, 2018.

[9] A. Gamez-Diaz, P. Fernandez, and A. Ruiz-Cortes, “Governify for apis: SLA-driven ecosystem for api governance,” in Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ser. ESEC/FSE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 1120–1123.

[10] K.-B. Schultis, C. Elsner, and D. Lohmann, “Architecture challenges for internal software ecosystems: a large-scale industry case study”, in Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, ser. FSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 542–552.

[11] E. Wilde and M. Amundsen, “The challenge of API management: Api strategies for decentralized api landscapes”, in Companion Proceedings of The 2019 World Wide Web Conference, ser. WWW ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1327–1328.

[12] “ISO/IEC 25010:2023 - systems and software engineering - systems and software quality requirements and evaluation (SQuaRE) - product quality model”, 2023.

[13] “ISO/IEC 25023:2016 - systems and software engineering - systems and software quality – requirements and evaluation (SQuaRE) - measurement of system and software product quality”, 2016.

[14] “ISO/IEC 25040:2011 - systems and software engineering - systems and software quality requirements and evaluation (SQuaRE) - evaluation process”, 2011.

[15] V. R. Basili, G. Caldiera, and H. D. Rombach, “The goal question metric approach”, in Encyclopedia of software engineering. New York, NY, USA: Wiley-Interscience, 1994.

[16] M. Mathijssen, M. Overeem, and S. Jansen, “Identification of practices and capabilities in API management: A systematic literature review”, 2020. [Online]. Available: http://arxiv.org/abs/2006.10481

[17] E. dos Santos and S. Casas, “API management and SQuaRE: A comprehensive overview from the practitioners’ standpoint”, in Computer Science – CACIC 2023. Cham: Springer Nature Switzerland, 2024, pp. 137–150.

[18] E. dos Santos, and S. Casas, “Unveiling quality in API management: A systematic mapping study”, in 2024 L Latin American Computer Conference (CLEI), 2024.

[19] A. Khammal, Y. Boukouchi, M. A. Hanine, and A. Marzak, “General meta model of software quality,” International Journal of Computer Science and Information Technologies, vol. 7, no. 4, 2016.

[20] N. Yılmaz and A. K. Tarhan, “Meta-models for software quality and its evaluation: A systematic literature review”, in Joint Proceedings of the 30th International Workshop on Software Measurement and the 15th International Conference on Software Process and Product Measurement, ser. CEUR Workshop Proceedings, vol. 2725. Mexico City: CEUR-WS.org, 2020.

[21] R. Yamamoto, K. Ohashi, M. Fukuyori, K. Kimura, A. Sekiguchi, R. Umekawa, T. Uehara, and M. Aoyama, “A quality model and its quantitative evaluation method for web APIs”, in 2018 25th Asia-Pacific Software Engineering Conference (APSEC), 2018, pp. 598–607.

[22] K. Petersen, S. Vakkalanka, and L. Kuzniarz, “Guidelines for conducting systematic mapping studies in software engineering: An update”, Information and Software Technology, vol. 64, pp. 1–18, Aug. 2015.

[23] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, “Systematic literature reviews in software engineering – A systematic literature review”, Information and Software Technology, vol. 51, no. 1, pp. 7–15, Jan. 2009.

[24] M. Shaw, “Writing good software engineering research papers”, in 25th International Conference on Software Engineering, 2003. Proceedings. Portland, OR, USA: IEEE, 2003, pp. 726–736.

[25] P. Ralph, “Toward methodological guidelines for process theories and taxonomies in software engineering”, IEEE Transactions on Software Engineering, 2018, 45(7), pp. 712-735.

[26] C. Cachero, C. Calero, and G. Poels, “Metamodeling the quality of the web development process' intermediate artifacts”, in Web Engineering, L. Baresi, P. Fraternali, and G.-J. Houben, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 74–89.

[27] L. R. d. Carvalho and A. Araujo, “Insights into the Performance of Function-as-a-Service Oriented Environments Using the Orama Framework”, SN Computer Science, vol. 4, no. 3, p. 305, 2023.