ISSN 2007-9737

Towards a Standardized Evaluation of APIs Non-Functional Requirements Focused on Completeness and Soundness Qualities

Joanna Alvarado-Uribe[1], Ari Y. Barrera-Animas[2], Miguel Gonzalez-Mendoza[1], Ariel Lucien Garcia-Gamboa[1], Neil Hernandez-Gress[1]

[1] Tecnologico de Monterrey,
School of Engineering and Sciences, Monterrey,
Mexico

[2] Universidad Panamericana,
Facultad de Ingeniería, Mexico City,
Mexico

{joanna.alvarado, mgonza, ariel.garcia, ngress}@tec.mx, aribarrera@up.edu.mx

Abstract. Nowadays, well-designed user documentation plays a relevant role in the development and delivery of high-quality software products. It minimizes software maintenance costs either for developers and managers by following quality standards during software life-cycle process. In this research work it is proposed an assessment approach consisting of a set of metrics, a methodology, and a guideline focused on supporting the testing of non-functional requirements of Application Program Interfaces (APIs). An emphasis is made on the evaluation of soundness and completeness qualities of the official documentation of APIs to support its evaluation of documentation's usability. Furthermore, a straightforward criterion is proposed to rate the soundness and completeness qualities in an easily readable way.

Keywords. Automation testing, non-functional requirements, quality, completeness.

1 Introduction

International standards for software documentation revision stand out that well-designed documentation not only assists users and helps to reduce training and support costs, but also enhances the reputation of the final product.

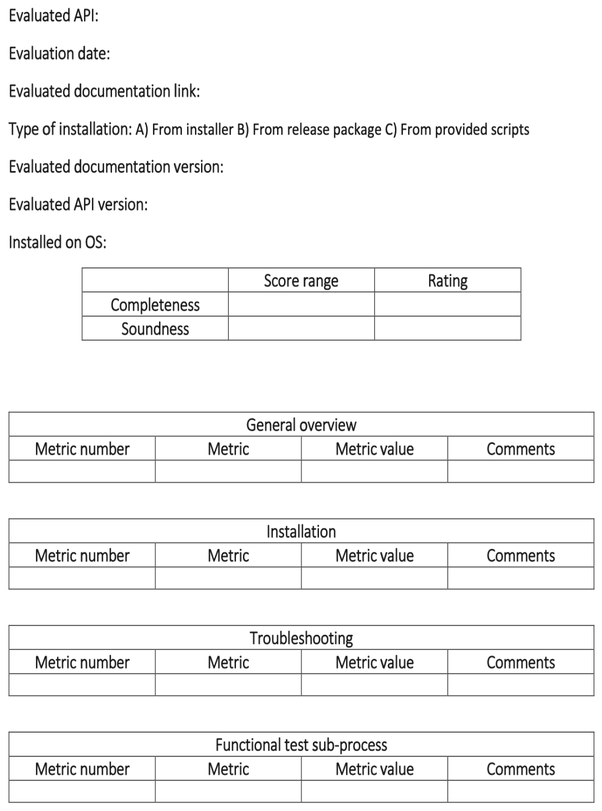
Through processes, such as verification, validation testing, and expert review of content during the development of a software product, developers and managers obtain valuable feedback about the accuracy and usability of their work [20, 27]. Throughout the State-of-the-Art, it has been stated that non-functional requirements are a key aspect to the success of a project [1, 11, 10, 28].

For example, the non-functional requirements related to usability issues [10, 28, 23, 22, 26], such as usability and system testing of documentation [20].

Notwithstanding, it is also known that the requirements engineering community has not established a definition for non-functional requirements as well as a methodology or process that allows eliciting, documenting, and validating them [6, 5, 11, 12, 9, 25] in an easy and transversal multidisciplinary manner.

Therefore, this article aims to address one of the gaps previously mentioned: a methodology that supports the assessment of a non-functional requirement through a set of metrics and a qualification criterion.

This approach directly benefits development enthusiasts, software development companies, the engineering community and researchers in general who seek to maintain adequate documentation based on internationally accepted



Evaluated API:

Evaluation date:

Evaluated documentation link:

Type of installation: A) From installer B) From release package C) From provided scripts

Evaluated documentation version:

Evaluated API version:

Installed on OS:

	Score range	Rating
Completeness		
Soundness		

General overview			
Metric number	Metric	Metric value	Comments

Installation			
Metric number	Metric	Metric value	Comments

Troubleshooting			
Metric number	Metric	Metric value	Comments

Functional test sub-process			
Metric number	Metric	Metric value	Comments

Fig. 1. General structure of the proposed evaluation guideline

guidelines and standards without having to prepare different documents depending on a particular evaluation agency.

Specifically, the proposed metrics and methodology allow any software development, including those services that comprehend any integration of Artificial Intelligence algorithms, to have a suitable, fast, and multidisciplinary way to accelerate the incorporation of improvements through the life cycle of a product and guarantee its quality.

The main contributions of this article are four-fold:

– A set of metrics to evaluate soundness and completeness qualities of documentation related to the installation of an API.

– A methodology focused on assessing non-functional requirements incorporating the proposed metrics.

– A guideline for each evaluation of the soundness quality and the completeness quality that comprehends the proposed evaluation metrics.

– A criterion for rating the documentation soundness and completeness based on the proposed evaluation metrics.

The rest of the article is organized as follows. Section 2 provides the State-of-the-Art related to software quality, emphasising concepts and definitions associated with functional and non-functional requirements, and evaluation standards focused on testing software as well as documentation.

Later, Section 3 presents the proposed evaluation metrics and guideline to support the testing of documentation according to the non-functional requirements: soundness and completeness. Finally, Section 4 provides the conclusions of this research.

2 State of the Art

Nowadays, software products offer increasingly robust functionality in both within applications and across multiple platforms. In order to offer this, software designs include underlying programming mechanics, rules, calculations, and algorithms that are only discernible through several testings. In consequence, user documentation endures as an essential component of software products [27].

As previously stated, there is a need for a standardized evaluation metric and/or methodology that can be used transversally in the different application fields of the documentation of an API. Consequently, this section presents the foundations and standards that support this research work.

2.1 Software Quality

The ISO (International Organization for Standardization [17]) 9000:2000 standard defines quality as *"degree to which a set of inherent characteristics fulfills requirements", where a requirement is defined as a "need or expectation that is stated, generally implied or obligatory"* [9].

On the other hand, software quality is defined by the Institute of Electrical and Electronics Engineers (IEEE [14]) as the *"degree to which a system, component, or process meets specified requirements" and the "degree to which a system, component, or process meets customer or user needs or expectations"*.

Thereupon, aspects of software quality were included in the definition; being defined as *"Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software"* [8].

Finally, a commonly used extended definition of the software quality assurance is provided by IEEE as *"a systematic, planned set of actions necessary to provide adequate confidence that the software development process or the maintenance process of a software system product conforms to establish functional technical requirements as well as with the managerial requirements of keeping the schedule and operating within the budgetary confines"* [8].

According to the definitions given above, the requirements that must be met are generally classified into two types: functional and non-functional. This research work focuses on non-functional requirements.

2.1.1 Non-Functional Requirements

The term non-functional requirements were first used in 1985 by Roman, G-C. [25] to refer to the constraints in the software's design complexity.

That is, the non-functional requirements restrict the types of solutions might be considered. Moreover, the term is also referred to as quality attributes, goals, extra-functional requirements, and non-behavioral requirements [9, 6, 5, 11, 12, 28, 22, 4, 1].

Table 1. Metric Category: General Overview

#	Metric
1	Is versioned.
2	Includes the release date.
3	Has a table of contents.
4	Establishes the versions of the software it applies to.
5	Specifies the target user.
6	Presents the highlighted or listed changes from previous versions.
7	Uses simple vocabulary familiar to the users.
8	Specifies terms and/or acronyms included in a glossary.
9	Includes links to additional information when needed.
10	Presents the tasks in the order in which they are performed.
11	Provides a general description of the APIs.
12	Indicates the skills required by the installer user.
13	Properly describes the list of supported platforms.

However, a formal specification of the requirements that can be classified into the non-functional category is difficult to obtain. In the literature, the non-functional requirements are those requirements that are not considered or categorized as functional requirements. For example, the response to failure and human factors are requirements whose solution can not be known until performing empirical evaluations [25, 6, 5].

Notwithstanding, several works have been performed in the aim to characterize and classify the non-functional requirements [9, 6, 5, 11, 12, 28, 22, 4, 3, 26, 1]. The first taxonomy in the literature that proposed a classification is the one presented in [25] which includes the following constraints: Interface, performance, operating, life-cycle, economic, and political.

Later on, a costumer-oriented and technically-oriented classification for non-functional requirements was published by the Rome Air Development Center (RADC) [6]. The quality attributes that the classification embraces are efficiency, integrity, reliability, survivability, usability, correctness, maintainability, verifiability, expandability, flexibility, interoperability, portability, and reusability.

As previously mentioned, despite the fact that several research works propose a taxonomy or a classification scheme, there is no formal definition or a complete list of non-functional requirements or either a universal classification scheme that fits different application domains [9, 25, 6, 5, 11, 28, 4, 3, 26, 1].

Table 2. Metric Category: Installation

#	Metric
1	Steps are ordered.
2	Summary of steps is provided.
3	Steps are easy to follow.
4	Steps/commands are complete.
5	Tracking of progress is shown.
6	Steps for software dependencies are provided.
7	Supported versions of software dependencies are specified.
8	Required ports are indicated.
9	Installation on different platforms is correctly executed.
10	Software prerequisites are specified.
11	Security issues that may apply (firewall, irreversible actions, etc.) are indicated.
12	Hardware prerequisites are specified.
13	Exception messages are displayed (i.e. when not enough resources are found).
14	Required resources are defined.
15	Requirement of administrative privileges is specified.
16	A log file for all messages is integrated (or these messages are displayed within the installation process).
17	Logs location is provided.
18	Confirmation messages on critical operations are specified.
19	Installation process is complemented with examples and figures.
20	If the installation process fails: Error feedback is provided.
21	If the installation process fails: Instructions to recover from error are provided.
22	If the installation process fails: The recovery process is indicated and it allows completing the installation.
23	If the installation process is not achieved, it could be because the installation is not possible at all.
24	If the installation process is not achieved, it could be because the procedure is too complex.
25	If the installation process is not achieved, it could be because the procedure takes more than one working day.
26	A large number of installation steps are presented.

2.2 Evaluation Standards

The International Organization for Standardization (ISO [17]), the International Electrotechnical Commission (IEC [13]), the Institute of Electrical and Electronics Engineers (IEEE [14]) Standards documents, IEEE Societies, and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA [15]) Standards Board are committed to the development of international standards through a consensus development process that is approved by the American National Standards Institute (ANSI [2]) [20, 27].

The following standards define some minimum requirements to be considered when designing, developing, and reviewing the software's documentation.

However, the standards only provide informational guidelines and examples of its use in some organisations and do not provide a consensus checklist of qualities.

– Systems and Software Engineering - Requirements for Designers and Developers of User Documentation (ISO/IEC/IEEE 26514:2010).

  The standard provides the requirements for the design and development of software user documentation and defines the documentation process from the documentation developers' point of view. The minimum requirements for the structure, information content, format of user documentation, and informative guidance for user documentation style are provided [27].

– Systems and software engineering - Requirements for testers and reviewers of information for users (ISO/IEC/IEEE 26513:2017).

  The standard is oriented to supporting the need to provide consistent, complete, accurate, and usable documentation to software users. This test involves the use of documentation alongside the corresponding test software to verify that the documentation is consistent with the software [20].

– Software and Systems Engineering - Software Testing - Part 3: Test Documentation (ISO/IEC/IEEE 29119-3:2013).

  This part belongs to the standard ISO/IEC/IEEE 29119 which aims to define an internationally-agreed set of standards for software testing. The standard identifies as required contents that need to be included in test policies the following: the overview information, document specific information, introduction, and test policy statements [21].

The presented standards identifies and provide some guidelines to check and verify that documentation of software is technical accurate and consistent. That is, these standards helps to determine if the documentation is technically accurate (verify), but does not determine whether the documentation is usable (validate) [20].

Thus, these standards are centred to help technicians/developers in the life-cycle of a software product. To verify if a software's documentation is usable, then the usability testing of the documentation is performed. The usability testing of the documentation helps to determine if the information embraced in the documentation meets the users' needs; such as if it is understandable and if they can apply it [20, 19].

Furthermore, common metrics used for usability testing from Common Industry Format for Usability Test Reports [16] are efficiency, effectiveness, and satisfaction [20]. From State-of-the-Art it is noticeable that it is a need to enhance and increase the metrics and guidelines that are used to verify not only the documentation that are used by technicians during the life-cycle of the product development, but also the documentation that are at disposal of end-users of a product.

The pursuit of metrics that help to ensure to complete non-functional requirements in documentation test is important to attain an international standard. This presents an opportunity to enhance the documentation verification by not only check for its consistency, but also for their usability for both experienced users (technicians, developers, etc) and common users (general public).

3 Proposed Evaluation Metrics

As pointed out in the State-of-the-Art, the process of documentation evaluation involves several qualities (aspects) that developers or specialists must take into account to ensure the overall quality of the documentation.

This article provides metrics focused on the evaluation of end-users' documentation by supporting the testing of non-functional requirements of Application Program Interfaces (APIs). Specifically, this proposal allows evaluating the soundness and completeness qualities of the documentation available on the official websites of APIs.

The proposed metrics aim to help the development of an accurate, usability, and completeness users' documentation of APIs within the life-cycle of product development.

Table 3. Metric Category: Troubleshooting

#	Metric name
1	Troubleshooting section is provided.
2	Common possible errors are listed with instructions to solve them.

Table 4. Metric Category: Functional Test Sub-Process

#	Metric name
1	Instructions to launch/start the API are provided.
2	Arguments that can be provided at the start-up command are detailed.
3	Sanity check procedure is clearly described.
4	The system works as expected.

It is noteworthy to mention that, as similar to the ISO/IEC/IEEE standards previously described, the proposed evaluation metrics can highlight problems with the evaluated API; however, resolving these problems is out of the scope of this research work.

The proposed metrics aim to maintain an accurate and useful content comprehended in the APIs' documentation. Furthermore, to facilitate the documentation evaluation, a guideline that comprehends the metrics is proposed.

This guideline aims not only to help developers and/or specialists to ensure an accurate and complete evaluation of the documentation of APIs during the life-cycle of product development but to ease the tasks during the usability evaluation.

To achieve this goal, several previously presented aspects are concentrated in a guideline in a structured form that facilitates both the evaluation of documentation during the non-functional evaluation and the management and control of updating tasks. The general structure of the proposed guideline is shown in Figure 1.

To seek clearness and fluency, the proposed metrics are concentrated in four categories according to their evaluation purpose. The four established categories are: General overview, Installation, Troubleshooting, and Functional test sub-process.

Table 5. Metrics Involved in the Completeness and Soundness Quality Evaluation

Metric category	**Metric # for**	
	Completeness	Soundness
General overview.	1-4, 6-10, 13	1-8, 10-13
Installation.	2, 6, 7, 10-12, 14, 17, 19	2, 6-8, 10-12, 14, 17, 19
Troubleshooting.	1, 2	1, 2
Functional test sub-process.	1, 3	1-3

Table 6. Rating Scale of the Completeness and Soundness Quality

Rating	**Score range**	
	Completeness	Soundness
Very Poor/Bad.	0 - 3	0 - 4
Poor.	4 - 7	5 - 8
Fair.	8 - 11	9 - 13
Average.	12 - 15	14 - 17
Good.	16 - 19	18 - 22
Very Good.	20 - 22	23 - 26
Excellent.	23	27

The categories and their respective metrics are detailed in Table 1, Table 2, Table 3, and Table 4.

The structure of the evaluated aspects in each category is concentrated in four columns to ease the evaluation process.

– Column 1: Metric number. A numeric value that identifies the metric in its corresponding category.

– Column 2: Metric. The metric that is evaluated.

– Column 3: Metric value. A value that represents the measurement of the metric. Only one of the following values can be selected at a time: Yes, No, Partially, or Not Available (NA).

– Column 4: Comments. A section where any relevant comment about the metric that is being evaluated can be placed.

Once the metrics that help in the evaluation of non-functional requirements are defined and a guideline is provided, metrics for evaluating the completeness and soundness qualities of documentation must be selected.

In consequence, the evaluation metrics taken into account need to be mapped regarding their purpose of evaluating these two qualities. The selection of metrics that integrates the completeness and soundness qualities measurements were based on the standards presented in Section 2.

Despite that all metrics concentrated in the guideline are relevant for the overall documentation evaluation, not all of them are required to evaluate the two previously introduced qualities. Therefore, 23 metrics were considered to evaluate the completeness quality and 27 metrics were contemplated to measure the soundness quality. The metrics to evaluate the completeness and soundness qualities are detailed in Table 5.

Afterward, a criterion must be contemplated to give a rating to the tested documentation. This rating will serve as an easily readable base-line for users and developers to notice the quality of the documentation that is reviewing. The main goal of this rating is to simplify the process of reviewing documentation of software, specifically APIs documentation.

Hence, a straightforward criterion that adds only the positives values of the metrics in each quality is applied. That is, the rating score of the qualities will be established by adding only the “Yes” values of each metric that embrace that quality.

Metrics values of “No”, “Partially”, and “NA” are not taken into account since these values represent that the corresponding information evaluated for that metric is incomplete or is missing. Consequently, the maximum value of the criterion for the completeness quality is 23 and for the soundness quality is 27.

The addition of these qualities by themselves does not help to get an insight into the documentation’ quality. Therefore, a seven rating scale is chosen to provide an abstraction and clearance about the quality of the documentation tested.

The rating scale scores are: 1) “Very Poor/Bad”, 2) “Poor”, 3) “Fair”, 4) “Average”, 5) “Good”, 6) “Very Good”, and 7) “Excellent”.

This representation is friendly-user and easy to understand by both developers and general users of the documentation. Moreover, the seven rating scale clearly defines the status of the documentation qualities.

Following the straightforward approach, the maximum value of each quality is divided into this seven rating scale to obtain the ranges that will determine each score on the scale. The ranges of the ratings of the completeness and soundness qualities are concentrated in Table 6.

As noticed from Table 6 the “Excellent” rating is the only one that has no range of scores, being achievable only if it has all metrics covered with the “Yes” value. By scoring the maximum rate in this form, the quality evaluation of the documentation ensures that there is no lack or deficiency in any aspect of the content.

On the opposite, the minimum rating can not be expressed through the zero value for both qualities. The zero value itself could represent that the documentation does not have any single metric with the “Yes” value or that there is no documentation at all.

Thus, to concentrate both representations with no distinctions for the purposes of evaluation, the zero-score is integrated into the range of the “Very Poor/Bad” rating.

4 Conclusion

From the State-of-the-Art, it can be noticed that there is a need for the engineering community, to reach a consensus and define a methodology that can evaluate the non-functional requirements of a project in an easy, multidisciplinary, and standardized manner.

This will allow a fast and reliable way to review and integrate improvements in the documentation during the development of software products. In this regard, the proposed approach aims to obtain an accurate and complete non-functional evaluation of completeness and soundness qualities of any API.

The proposed metrics, the methodology, and the guideline allow delivering high-quality documentation of APIs to developers within the minimum time required.

This is achieved by taking care of relevant metrics that are of interest to developers during the development of documentation in the software’s life-cycle process. Therefore, the proposed metrics concentrated in each guideline make it possible to ensure a comprehensive review of the soundness and completeness qualities of the documentation since these metrics are based on the standards introduced in the State-of-the-Art.

Moreover, another advantage of following the approach proposed in this work is that after evaluating the API’s documentation with the proposed metrics concentrated on the proposed guideline, several relevant qualities and aspects related to the usability of the documentation will also be covered before performing a process of usability evaluation by itself.

Consequently, the APIs’ documentation will require less effort in the validation and correction tasks during the usability evaluation. Thus, developers and end-users can rely on to have an accurate, useful, and friendly API’s documentation. Consequently, the proposed approach can be applied to other types of domains that need to evaluate their software documentation.

Finally, the approach of this proposal supports following one of the recommendations established in the standard ISO/IEC 26514:2008 [27]; that is, to obtain high-quality software documentation, its development, verification, and evaluation should be an integral part of the software life-cycle process and should not be regarded after the software’s implementation.

Further work could focus on developing a form of pilot evaluation to provide a complete and reliable non-functional evaluation model. For this, the incorporation of more evaluation metrics related to other qualities of the documentation would have to be addressed, thus achieving a complete evaluation of properties related to the non-functional evaluation, such as usability.

Moreover, standards and frameworks related to risk management and governance of developments incorporating Artificial Intelligence, such as [18, 24], will be included to strengthen the proposed metrics and evaluation methodology. Furthermore, to test the proposed approach, the cloud-based technological platform called FIWARE [7] will be considered.

The test case will focused on testing the installation manuals of the FIWARE Generic Enablers (GEs) [7], whose task is also relevant to the FIWARE initiative itself [7], by using the set of metrics, the qualification criterion, the assessment methodology, and the guideline proposed in this research.

Acknowledgments

The authors would like to thank the FI-NEXT project, FIWARE Foundation, and Tecnologico de Monterrey for the support to carry out this research; the FI-NEXT project is co-funded by the EU's Horizon2020 programme under agreement number 732851 and the CONACYT's agreement number 274451. We really appreciate the participation, collaboration, and review of the people involved in this research work.

References

1. **Aiello, M., Fiorini, L., Georgievski, I. (2021).** Software engineering smart energy systems. Handbook of Smart Energy Systems, pp. 1–29. DOI: 10.1007/978-3-030-72322-4_21-1.
2. **ANSI (2023).** American national standards institute. https://www.ansi.org/.
3. **Bhowmik, T., Quoc-Do, A. (2019).** Refinement and resolution of just-in-time requirements in open source software and a closer look into non-functional requirements. Journal of Industrial Information Integration, Vol. 14, pp. 24–33. DOI: 10.1016/j.jii.2018.03.001.
4. **Casamayor, A., Godoy, D., Campo, M. (2010).** Identification of non-functional requirements in textual specifications: A semi-supervised learning approach. Information and Software Technology, Vol. 52, No. 4, pp. 436–445. DOI: 10.1016/j.infsof.2009.10.010.
5. **Chung, L., do Prado-Leite, J. C. S. (2009).** On non-functional requirements in software engineering. Conceptual Modeling: Foundations and Applications, pp. 363–379. DOI: 10.1007/978-3-642-02463-4_19.
6. **Chung, L., Nixon, B. A., Yu, E., Mylopoulos, J. (2012).** Non-functional requirements in software engineering. Springer Science+Bussiness Media.
7. **FIWARE Foundation (2023).** FIWARE. https://www.fiware.org/foundation.
8. **Galin, D. (2004).** Software quality assurance: From theory to implementation. Pearson Education Limited.
9. **Glinz, M. (2005).** Rethinking the notion of non-functional requirements. Proceedings of the Third World Congress for Software Quality, Vol. 2, pp. 55–64.
10. **Glinz, M. (2007).** On non-functional requirements. 15th IEEE International Requirements Engineering Conference (RE 2007), pp. 21–26. DOI: 10.1109/RE.2007.45.
11. **Gómez-Sotelo, K. I., Baron, C., Esteban, P., Gutiérrez-Estrada, C. Y., Laredo-Velázquez, L. J. (2018).** How to find non-functional requirements in system developments. IFAC-PapersOnLine, Vol. 51, No. 11, pp. 1573–1578. DOI: 10.1016/j.ifacol.2018.08.272.
12. **Grimshaw, D. J., Draper, G. W. (2001).** Non-functional requirements analysis: Deficiencies in structured methods. Information and Software Technology, Vol. 43, No. 11, pp. 629–634. DOI: 10.1016/S0950-5849(01)00171-9.
13. **IEC (2023).** IEC: International Electrotechnical Commission.
14. **IEEE Advancing Technology for Humanity (2023).** The professional home for the engineering and technology community worldwide.
15. **IEEE SA Standards Association (2023).** Standards IEEE.

16. **International Organization for Standardization (2023).** Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Common Industry Format (CIF) for usability test reports.

17. **ISO International Organization for Standardization (2023).** ISO: Global standards for trusted goods and services.

18. **ISO/IEC 23894:2023 (2023).** Information technology — Artificial intelligence — Guidance on risk management. International Organization for Standardization.

19. **ISO/IEC JTC1/SC7 Working Group 26 (WG26) (2023).** Software Testing. International Organization for Standardization.

20. **ISO/IEC/IEEE 26513:2017 (2017).** Systems and software engineering – Requirements for testers and reviewers of information for users. International Organization for Standardization, pp. 1–126.

21. **ISO/IEC/IEEE 29119-3:2013 (2013).** Software and systems engineering – Software testing –Part 3: Test documentation. International Organization for Standardization, pp. 1–138.

22. **Kopczyńska, S., Nawrocki, J., Ochodek, M. (2018).** An empirical study on catalog of non-functional requirement templates: Usefulness and maintenance issues. Information and Software Technology, Vol. 103, pp. 75–91. DOI: 10.1016/j.infsof.2018.06.009.

23. **Metsa, J., Katara, M., Mikkonen, T. (2007).** Testing non-functional requirements with aspects: An industrial case study. Seventh International Conference on Quality Software (QSIC 2007), pp. 5–14. DOI: 10.1109/QSIC. 2007.4385475.

24. **National Institute of Standards and Technology (2023).** NIST AI RMF playbook. Information Technology Laboratory.

25. **Roman (1985).** A taxonomy of current issues in requirements engineering. Computer, Vol. 18, No. 4, pp. 14–23. DOI: 10.1109/MC.1985.1662861.

26. **Thakurta, R. (2013).** A value-based approach to prioritise non-functional requirements during software project development. International Journal of Business Information Systems, Vol. 12, No. 4, pp. 363–382. DOI: 10.1504/ IJBIS.2013.053213.

27. **The Institute of Electrical and Electronics Engineers, Inc (2011).** IEEE standard for adoption of ISO/IEC 26514:2008 systems and software engineering–requirements for designers and developers of user documentation. DOI: 10.1109/IEEESTD.2011.5712775.

28. **Zou, J., Xu, L., Yang, M., Zhang, X., Yang, D. (2017).** Towards comprehending the non-functional requirements through developers' eyes: An exploration of stack overflow using topic analysis. Information and Software Technology, Vol. 84, pp. 19–32. DOI: 10.1016/j.infsof.2016.12.003.

Article received on 09/06/2023; accepted on 21/09/2023. Corresponding author is Miguel Gonzalez-Mendoza.