A Mapping Study on JavaScript Quality Attributes and Metrics

Ioannis Zozas[1], Stamatia Bibi[1], Apostolos Ampatzoglou[2], Elvira-Maria Arvanitou[3], Pantelis Angelidis[1], Markos Tsipouras[1]

[1] Dept. of Electrical & Computer Engineering, University of Western Macedonia, Kozani, Greece

[2] Dept. of Applied Informatics, University of Macedonia, Thessaloniki, Greece

[3] Dept. of Information and Electronic Engineering, International Hellenic University, Thessaloniki, Greece

izozas@uowm.gr, sbibi@uowm.gr, apostolos.ampatzoglou@gmail.com, earvanitoy@gmail.com, paggelidis@uowm.gr, mtsipouras@uowm.gr

Abstract

Context: Although JavaScript dominates modern software development, research on its quality attributes remains scarce, despite the fundamental differences that distinguish it from other languages. This motivates dedicated research related to JavaScript quality attributes and metrics.

Objective: This paper aims to identify (a) the quality attributes of JavaScript language that are mainly studied and (b) the quality metrics that are used to quantify them. Additionally, the paper provides information on the tools that can be used to measure quality metrics.

Method: To achieve these goals, we have conducted a mapping study on 7 journals and 8 conferences of high quality. A total of 142 primary studies, published between 2002 and February 2025, have been selected and analyzed, to identify and classify software metrics to high-level quality attributes, as described in ISO/IEC 25010:2011.

Results: Maintainability, Security, Reliability, and Usability quality attributes are the most studied ones. Furthermore, 78 generic and 48 JavaScript-specific metrics were identified. A wide dispersion of metrics has been identified for assessing each quality attribute, based on different development tasks. Moreover, a variety of tools and benchmarks were identified.

Conclusion: A clear research trend in JavaScript quality assessment related to issues that involve software reuse, code testing, and dynamic code analysis has been identified. Yet differences among primary studies in quality assessment and quantification, along with tool adoption indicate the need for further exploration of these recurring topics.

Keywords: *JavaScript, Software Quality, Quality Attributes, Quality Metrics, Mapping Study*

1. Introduction

Tom de Marco, in one of the most prolific quotes of software engineering ("*We cannot control something that we cannot measure"*) [16] has underlined that metrics can be the foundation of software management and control [24]. The term software quality is a nebulous term, that refers to the desired characteristics of a software product from the viewpoint of different stakeholders, such as the developers, the end-users, the client, and the product owner per se [15]. To measure properly software quality, a plethora of metrics have been proposed over the past years based on the unique characteristics of the software product under consideration emphasizing the implementation language, the scope of the software, and the development phase. The numerous software quality metrics that can be found in the literature [49] have been reviewed and mapped, from secondary studies into different quality attributes providing useful classification schemas and synthesized knowledge that can assist future research.

The metric classification schemas proposed by recent secondary studies present limitations that arise from the fact that they do not incorporate the recent trend toward multi-paradigm programming. Currently, most software is developed with scripting languages. These languages combine different programming styles and can be used for various purposes. The most popular language of this type is JavaScript, which offers the opportunity to develop a variety of general-purpose applications, without strict programming rules, with the support of open-source

frameworks and libraries that can be reused, and with no constraints on the programming paradigm adopted [18]. Despite the popularity of the language, there is no classification schema on the metrics that can be used to assess the quality of JavaScript applications. The major limitations of existing secondary studies are:

- they emphasize metrics that describe typical programming paradigms, such as object-oriented programming [35] and procedural programming [49], leaving unexplored metrics that are appropriate for measuring quality aspects of multi-paradigm languages, such as JavaScript (JS). JavaScript programming language shares syntax commonalities with other languages, therefore several metrics recorded in other literature reviews [34] [44] [46] [49] and mapping studies are applicable [42], but there is a plethora of metrics stemming from the unique features offered by the language, that capture quality aspects of the applications developed in JavaScript language that is not yet summarized in any study. These involve distinctive programming features [39], with most notable the lack of typed variables and classes, the prototypal inheritance chain, the handling of functions, and the fact that JavaScript applications use the web browser as a host environment.
- they focus mostly on metrics used for measuring stand-alone, usually desktop applications [10]. On the other hand, modern application development with JavaScript concerns different types of applications from web and mobile applications, simple scripts embedded in devices, and programming frameworks, to reusable libraries requiring the adoption of robust language-specific dynamics [38] and static analysis techniques for extensive testing [5]. All these features have created the need to further explore and map the metrics that are used for assessing the quality of JavaScript implementations [20].

Despite the abundance of secondary studies on software quality metrics, most have focused on statically typed, object-oriented languages. JavaScript, as a dynamically typed, multi-paradigm language used extensively in both client- and server-side environments, poses unique challenges in software quality assessment. In this study, we conduct a mapping stidy across high-quality venues to identify the quality attributes most frequently studied in the context of JavaScript, the metrics used to quantify them, and the tools and workflows through which they are applied. By focusing exclusively on JavaScript, we expose language-specific metric categories—such as those related to dynamic typing, asynchronous execution, and dependency management—that are poorly addressed in general-purpose reviews. Our results provide a synthesized, JS-focused mapping of quality metrics, uncovering both prevailing trends and significant gaps in tool support and empirical validation. To address these limitations, this paper presents a mapping study on JavaScript quality attributes and metrics as appointed by the current state of research. The study aims to identify and classify software metrics to high-level quality attributes described in the ISO model [22]. Thus, the contribution of this study is summarized as follows:

- **We highlight the most studied software quality attributes in the context of JavaScript application development and present the metrics that can be used to quantify them.** This overview contributes a comprehensive list of the most important high-level quality attributes that have been identified concerning JavaScript development and a list of metrics for quantifying them. The quality attributes considered are the ones appointed by the ISO model [22]. On this basis, researchers and practitioners can focus on managing quality attributes that are considered crucial in JS applications and pay attention to understudied aspects of quality that may lead to application failures. Also, the interrelation between different high-level quality attributes and the corresponding metrics that can be used for their quantification is discussed. For example, popular metrics, such as the number of bugs can be used for the quantification of almost all ISO quality attributes and therefore should be more carefully considered.
- **We classify software metrics in different software tasks.** The goal of the study here is to identify metrics that can be used for different development language-specific tasks, as appointed by the authors of primary studies (i.e., browser- compatibility, dependencies update, dynamic testing, etc.).

- **We provide a list of tools that can be used for automatically calculating the metrics identified**. A list of tools and the metrics that they calculate is provided. Also, for popular, open-source tools, we provide a short description and discuss how they can be used to assess the quality of JS applications.
- **We present a list of applications/systems that are used for assessing the quality of JS applications.** A list of the popular applications and their sources is presented in an attempt to guide future research on selecting the relevant applications and sources of information that can be used for validating new types of metrics and assessing the quality of JavaScript applications.

To achieve these goals, we performed a mapping study accumulating knowledge from the results discussed in primary studies, published in high-quality venues in the domain of Software Engineering. In total, we considered as relevant 142 primary studies presented in Appendix B. Additionally; to be able to capture as many JS-specific metrics as possible, we did not focus only on studies that introduce or evaluate quality attributes but allowed the inclusion of studies that used metrics for any software engineering purpose. Our results show that quality attributes such as *Maintainability* and *Security* are often studied in the context of application development with popular JS-specific metrics including *XSS bugs* and *callbacks*. The development workflows that are mostly studied and assessed with the help of metrics are the Implementation and Deployment workflows and the most popular tool is the V8 JavaScript engine. The applications used for validation and benchmarking can be popular Websites or may be retrieved from GitHub and NPM.

2. Related Work and Background Information

2.1 Secondary Studies on Software Quality

Several secondary studies have been published in the last years, reviewing and mapping software quality metrics and attributes. Arvanitou et. al. [7] presented a mapping study on the state-of-research of Product Quality attributes. In total, 154 papers have been identified as primary studies to provide insight into the selection of the appropriate quality attributes and metrics based on the application domain. Also, in this study, metric validity issues and tool availability for metric calculation have been discussed. Furthermore, Nuez-Varela et. al. [35] published a mapping study on source code metrics and discussed the state of metrics and trends. A total of 226 studies has been reviewed to identify over 300 source code metrics. Moreover, Goel et. al. [19] presented a broad survey to identify object-oriented metrics to quantify internal quality attributes. Furthermore, they provided researchers with an overview of the current state of metrics as well as insight into object-oriented metric proposals. Finally, Alkharabsheh et. al. [2] conducted a literature review to present results on design smell detection, the scope of smells, detection approaches, tools, applied techniques, validation evidence, evaluation resources, programming language support, and the relation between detected smells and software quality attributes. A total of 395 articles has been reviewed to organize knowledge on design smell detection and pinpoint future trends.

In the case of specific software development phases, Arvanitou et. al. [7] associate the most frequent quality attributes to each development phase, linking the maintenance phase to maintainability, design and implementation phases to maintainability and testability, requirements explicitly to traceability, completeness, and consistency, and last but not least, architecture and project management phases to functionality. Elberzhager et al. [14] proposed the combination of static and dynamic quality assurance techniques for all (pre-code and post-code) development phases. On the other hand, Kupiainen et al. [31] concluded that some metrics are more prominent in certain phases.

Concerning static and dynamic analysis, Tahir et al. [47] conducted a mapping study on dynamic metrics and software quality to identify metrics for future research. They reviewed 60 identified primary studies out of 8 journals and 9 conferences, pinpointing the importance of complexity, cohesion, and coupling over quality – the latter two are limited and supported by older versions of JavaScript. Similarly, Elberzhager et al. [14] conducted a mapping study on dynamic and static quality assurance techniques, based on four digital libraries (Inspec, Compendex,

IEEE, and ACM). A total of 51 primary studies has been reviewed to result in combining both static and dynamic analysis as a more effective technique in source code inspection.

To a further extent, *concerning specific quality attributes,* Garousi et. al. [17] published a literature review study on 120 industrial and 46 academic test smell sources and provided guidelines for smell prevention, detection, and correction to improve maintainability. Furthermore, the authors identified the largest catalog of test smells, along with a summary of guidelines, techniques, and the tools available to deal with those smells. In the same direction, Radjenović et. al. [42] performed a systematic literature review including 106 papers, to identify software metrics and assess their applicability in software fault prediction. They concluded that object-oriented metrics are used twice as often as traditional source code metrics and process metrics, in fault recognition.

Concerning code language attributes and the development process, Oliveira et. al. [36] conducted an empirical study on productivity metrics, aiming to identify how researchers measure productivity, which metrics are appropriate for quantification, and how are classified based on commit activity. To a lesser extent, Saraiva et al. [46] present a systematic mapping study to identify the object and aspect-oriented code maintainability metrics. A total of 67 aspect-oriented metrics and 575 object-oriented metrics were identified and classified by the software attribute measured. The search strategy identified papers until June 2011 and was conducted on four digital libraries (IEEE, ACM, Compendex, and ScienceDirect), resulting in the selection of 138 primary studies.

Concerning metrics, Riaz et al. [44] conducted a systematic review of metrics concerning software maintainability, by reviewing 14 primary studies extracted from 4 digital libraries. To a further extent, Jabangwe et al. [24] conducted a mapping study on reliability, maintainability, effectiveness, and functionality quality attributes rather than solely maintainability. They reviewed 99 primary studies on the same digital libraries to conclude that maintainability is the most frequently studied attribute, while the CK metric suite is the most common. On the other hand, Kitchenham [26] conducted a preliminary mapping study to explore software metrics. By using the Scopus, IEEE, ACM, and Elsevier digital libraries, he reviewed 87 primary studies, he concluded that empirical rather than theoretical validation was the most popular type of metric evaluation. Varela et. al. [49] presented a systematic mapping study on source code metrics indicating growth but also dispersion in the field of source code metrics research. The authors focus on aspect, object, and feature-oriented programming, pinpointing a lack in this field of research requirements on metrics, tools, and programming languages.

Other mapping studies also exist that are *context-specific*, i.e., they explore quality concerning a particular domain or development paradigm. Mahdavi-Hezavehi et. al. [33] conducted a systematic literature review on quality attributes in self-adaptive systems, while Kupiainen et al. [31] focused on quality metrics in industrial agile development on the process level. Oriol et al. [37] published a mapping study on quality models for web services and Vargas et al. [48] explored quality attributes of Serious Games, Additionally, Abdellatief et al. [6] published a mapping study on component-based software engineering.

A summary of the related work, literature reviews, and mapping studies on software quality and/or metrics are presented in Table 1. By comparing our research effort to the related work, several differences arise in both approach methods and context as presented in Table 1. The current study:

- **Is context-specific as it synthesizes information on quality attributes and metrics that refer to JavaScript application development**. Even though there are studies in the literature that are context-specific see Table 1, none of them focus on the quality of JavaScript applications.
- **Covers the whole range of quality assessment processes** in the context of JS from the mapping of metrics to quality attributes, the mapping of metrics to the software development phase, and the use of tools and benchmarks that can be used to evaluate and assess quality.

- **Offers a set of quality metrics to assess JS-specific tasks and language-related mechanisms** that involve the multi-paradigm nature of the language, the web browser dependencies, the dynamic nature of the applications, and the fact that JS applications present stricter efficiency and security requirements.

Table 1 - Review summary of software quality and/or metrics studies

Study	Software Quality	Software Metrics	Context-specific	Tool reporting	Benchmark reporting	Field mapping	JS related
Alkharabsheh et. al. [2]	x	x	x	x			
Abdellatief et. al. [6]	x	x	x				
Arvanitou et. al. [7]	x	x		x		x	
Elberzhager et. al. [14]	x					x	
Garousi et. al. [17]	x	x		x			
Goel e.t al. [19]	x	x	x				
Jabangwe et. al. [24]	x	x	x				
Kitchenham et. al. [26]	x	x					
Li et. al. [31]	x	x	x			x	
Misra et. al. [33]	x			x			
Oliveira et. al. [35]	x	x		x	x		
Oriol et. al. [36]	x	x	x		x		
Park et. al. [37]	x		x				
Ramesh et. al. [42]	x	x	x				
Richards et. al. [44]	x	x	x				
Tahir et. al. [46]	x	x	x				
Vargas et. al. [47]	x	x	x	x			
Varela et. al. [48]	x						
Zhang et. al. [49]	x	x		x	x		
This study	x	x	x	x	x	x	x

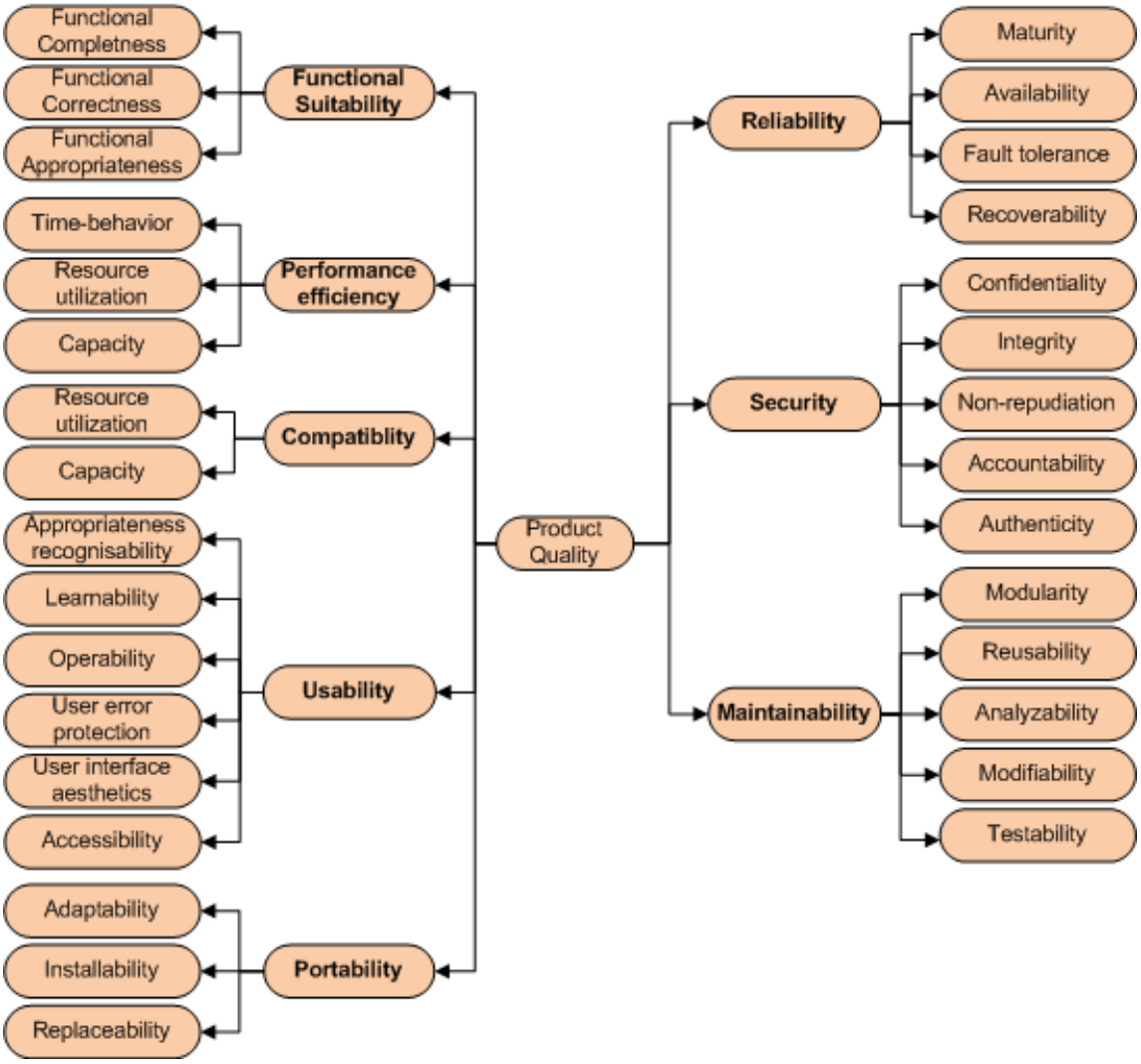
2.2 ISO/IEC 25010:2011 Quality Model

For our research, we have selected to use the ISO/IEC 25010:2011 quality model as a classification schema to map quality metrics to quality attributes, since it is the most recent quality model, built based on an international consensus. The ISO/IEC series of International Standards, entitled "*Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation (SQuaRE)*", covers software quality requirements specification and system/software quality evaluation. Its purpose is to assist those developing and acquiring systems and software products with the specification and evaluation of quality requirements. The model differentiates between Product Quality attributes and Quality in Use attributes. Product Quality attributes relate to static properties of software and dynamic properties of the computer system, while Quality in Use attributes relate to the degree to which a product or system can be used by specific users to meet their needs to achieve specific goals. We will study only product quality attributes since we want to determine the metrics that can be used to assess the level to which JavaScript applications have reached the requirements set by the development team and the customer [5] rather than on quality-in-use metrics. The latter focuses on the user perspective and how the user subjectively assesses the quality of the application [15]. Therefore, we believe that assessing quality in use attributes would require a mapping of different types of studies published in topic-specific venues (i.e., usability, human-computer interaction thematic venues) within the discipline of Human-Computer Interaction and not in core Software Engineering venues. Product quality is divided into 8 high-level characteristics as depicted in Figure 1, decomposed into secondary characteristics.

Based on the ISO/IEC model definitions, *Functional suitability* represents the degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions. *Performance efficiency* represents the performance relative to the number of resources used under stated conditions. *Usability* is the degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use. Similarly, *Reliability* is the degree to which a system, product, or component performs specified functions under specified conditions for a specified period. *Security* is the degree to which a product or system protects information and data so that persons or other products or systems

have the degree of data access appropriate to their types and levels of authorization. *Maintainability* represents the degree of effectiveness and efficiency with which a product or system can be modified to improve it, correct it, or adapt it to changes in environment, and requirements. *Portability* is the degree of effectiveness and efficiency with which a system, product, or component can be transferred from one hardware, software, or, other operational or usage environment to another. Finally, *Compatibility* is the degree to which a product, system, or component can exchange information with other products, systems, or components, and/or perform its required functions while sharing the same hardware or software environment.

Figure 1 - ISO/IEC Product Quality high-level and secondary-level attributes



3. Study Design

This section presents the protocol of the literature review that was conducted according to the guidelines of Petersen et al. [41]. The protocol constitutes a plan that describes the research questions and the steps for conducting the study. The reporting of this secondary study is based on the SEGRESS guidelines [30]. The SEGRESS checklist for this study is presented in Table 2.

Table 2 – SEGRESS guidelines check-list

SEGRESS item	**Discussion**
Title	The paper is entitled A Literature Review Study
Structured abstract	Followed based on journal guidelines
Opening	The first paragraph of the introduction
Rationale	The third paragraph of the introduction
Objectives	Section 3.1
Eligibility criteria	Section 3.2
Information sources	Section 3.2.1
Search strategy	Section 3.2.2
Selection process	Section 3.2.1
Data collection process	Section 3.2.2
Data items	Section 3.4
Study risk of bias assessment	Section 6
Effect measures	Not applicable
Analysis and synthesis methods	Synthesis not applicable, just classification
Reporting bias assessment	Not applicable
Certainty assessment	Not applicable
Study selection	Figure 2
Study characteristics	Section 4
Results of individual studies	Section 4
Results of analyses and synthesis	Section 4
Reporting biases	Section 6
Discussion	Section 5
Registration and protocol	The protocol is presented in Section 3

3.1 Objectives and Research Questions

The goal of this study, stated using the Goal-Question-Metrics (GQM) format [8], is to: ***analyze*** the existing literature on JavaScript for the ***purpose*** of characterization ***with respect to***: (a) the popularity of quality attributes and metrics in the research community, (b) the differences across different workflows and development tasks, (c) the level of empirical validation, and (d) the provided tool support ***from the point of view*** of researchers and practitioners ***in the context of*** JavaScript software quality assessment. Based on this goal, we have set the following research questions:

RQ1. ***Which quality attributes, as defined by the ISO/IEC 25010:2011 model, are studied?*** *RQ1* identifies the Product Quality attributes that are studied within the context of JavaScript application development. Also, it highlights the quality attributes that are jointly studied and provides a distribution of the number of metrics identified in the quality attributes of the ISO model.

RQ2. ***Which metrics can be used for assessing a specific quality attribute?*** *RQ2* examines the assessment of each quality attribute by metrics. It highlights the metrics that have been proposed as indicators of quality attributes, as well as the metrics that are considered JavaScript-related (i.e., metrics that are tightly connected to the structure of JavaScript language). In addition, this RQ explores *the quality metrics with respect to the implementation task they intend to monitor*. To this extent, we will provide a mapping between the quality metrics identified and the software engineering tasks in the context in which they are used and validated.

This mapping is performed solely based on the keywords the characterization of the study and the quality metrics as attributed by the authors of the primary studies.

RQ3 ***Which tools are used for automatically calculating software metrics?*** *RQ3* provides a summary of the tools that can offer the means for automating the calculation of metrics for JavaScript code. In this scope, it investigates the most common tools used to calculate the metrics identified in *RQ2*. Moreover, it examines each tool to guide metric calculation tool assembly.

RQ4. ***(a) What type of validation is used for the identified metrics?*** RQ4a provides insights on whether the metrics that appear in JavaScript-oriented studies are validated and in the case that they are whether empirical or theoretical validation is preferred.

(b)Which data/systems are used to validate metrics for JavaScript systems? *RQ4b* investigates the types of data /systems used by the primary studies to validate the proposed methods and metrics.

3.2 Search Process

The search strategy of the current study was defined based on the goal and the research questions. We have not selected to apply the search process on the complete digital libraries' contents, rather than on a limited number of software engineering venues. Kitchenham et. al. [27][28] proposed targeted searches at carefully selected venues to: (a) exclude low-quality papers from the final dataset, and (b) avoid low-quality grey literature. This approach has been widely applied in many systematic secondary studies in the field of software engineering [29]. For this reason, the applied search process targets collecting high-quality papers, published at premium software engineering venues.

3.2.1. Selection of Publication Venues

The search strategy of the current study was defined based on the goal and the research questions as described in Section 3.1. An initial attempt at a broad automated search produced an overwhelming volume of low-quality or irrelevant studies, making it difficult to identify the truly relevant work. It turned out that a large body of publications references JavaScript only as a means of building industrial systems and evaluates the quality of the resulting services rather than providing metrics or assessing product quality itself. To address this, and after a very thorough piloting, we have identified that relevant papers are coming from SE-specific venues and not general-purpose ones. Thus, we adopted the narrow-scope approach and focused on academically validated metrics rather than exhaustively cataloging all possible references. This strategy prioritizes credibility, ensuring that the selected studies are methodologically sound and scientifically reliable—an essential consideration when the goal is to understand, apply, or evaluate metrics in practice. We performed an automated search on selected digital library portals and specific publication venues. Targeted searches at carefully selected venues are acknowledged by Kitchenham et. al. [27], [28], as a good practice in software engineering secondary research, as a means to retrieve top-quality primary studies. Our choice to focus on top-quality venues is motivated by two reasons:

(a) The **broad research area** of this mapping study: according to Wohlin et al. [7], the broad research area, may set the retrieval of all possible primary studies as an unrealistic goal. In these cases, it is preferable to select a representative set of primary studies instead of the whole population [41]. This is also verified by the recent guidelines of Ampatzoglou et. al. [4] where the venue selection processes described in [7], [11], and [27] are mentioned as a good practice for isolating top-quality venues in the cases where a very broad topic is investigated.

(b) The **Data Validity** threat mitigation: The quality of the primary studies greatly affects the quality of the secondary study, especially in the case of literature reviews. In [4], it is mentioned that the selection of top-quality venues is the top mitigation action for the threat to validity: "Quality of Primary Studies" categorized under "Data Validity".

Therefore, the venues were chosen following the research of Karanatsiou et al. [25], which is the most recent publication of bibliometric papers, scholars, and institutions in software engineering. We adopted the narrow-scope approach and focused on academically validated metrics rather than exhaustively cataloging all possible references. This strategy prioritizes credibility, ensuring that the selected studies are methodologically sound and scientifically reliable—an essential consideration when the goal is to understand, apply, or evaluate metrics in practice. Four criteria were taken into consideration for filtering the search area of venues: (cr1) the classification of the venue should fall within the "Computer Software" topic and the evaluation of the venue should be at least at level "B" according to the Australian Research Council; (cr2) the venues should be strictly relevant to the software engineering field; (cr3) the average number of citations per published article per month should be at least 1; and (cr4) the venues should be of general-scope, i.e. we selected journals whose topic is not limited to phases or activities—with the only exception being conferences that are within the special interest of this study (e.g., CSMR/WCRE, ICSME, SANER, JSME).

3.2.2. Search Strategy and Article Filtering

To extract candidate primary studies from digital libraries, a keyword-based search has been developed. The main idea for constructing the search string was to use only the name of the targeted programming language, to assure high recall, since the precision will be guaranteed in the manual application of the inclusion/exclusion criteria. During the initial search phase of our research, we used the string ("JavaScript OR JS") in the abstract or the full title. We observed that many results were not primarily concerned with JavaScript itself but rather mentioned it incidentally (e.g., as the implementation language for a prototype or demonstration tool). To ensure that the retrieved studies had JavaScript as their main focus, we required its presence in the title or abstract. This approach reduced noise and helped us capture research that explicitly addresses JavaScript quality, while avoiding papers where JavaScript played only a peripheral role. We then refined the search to ("Quality" OR "Metrics") AND ("JavaScript" OR "JS"), but this resulted in too few studies to support the research. To balance precision and recall, the search string was broadened to include only two terms. The final search string that was applied in the title and abstract is:

$$(JavaScript\ OR\ JS).$$

From the automated search, we obtained 418 papers. Next, the candidate primary studies have been filtered for analysis, by applying a two-step manual filtering phase that involves the following inclusion criteria (IC):

1. The articles should perform actual research involving JavaScript applications, not just referring to the term JavaScript in the context of modern scripting languages. For example, consider a paper that focuses on Python, but mentions JS as an alternative scripting language.
2. The articles should perform research related directly or indirectly to software application quality assessment (i.e., mentioning at least one QA from the ISO/IEC model).

In this process, the process of manual filtering included a full paper examination. Moreover, we have used the following exclusion criteria [29]: **EC.1.** The primary study is an editorial, position paper, keynote, opinion, tutorial, poster, or panel; **EC.2.** The study is not written in English—not applicable based on the selected venues. An

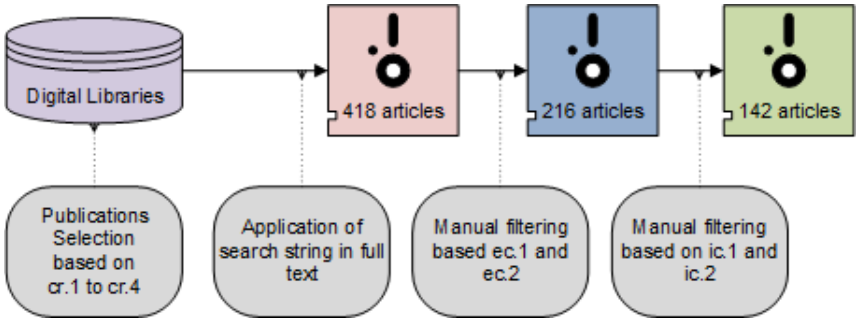
overview of the selection process is presented in Figure 2. A summary of the inclusion and exclusion criteria is presented in *Table 3*.

Table 3 – Inclusion and exclusion criteria

Criteria	Definition	Rationale
IC.1	Include research involving JavaScript applications	JavaScript as a scripting language is often included a supplementary to other languages. The article scope focuses solely on actual JavaScript applications.
IC.2	Include research related directly or indirectly to software application quality assessment	While quality is a broad and fuzzy term, research related to at least one quality attribute deriving from the ISO/IEC model is included as to further research each quality dimension.
EC.1	Exclude editorial, position paper, keynote, opinion, tutorial, poster, or panels	The current study targets for inclusion formal, peer-reviewed academic works, such as research articles, peer-reviewed conference papers, systematic reviews, or journals, that highlight higher quality research.
EC.2	Exclude non-English venues	Venues written in other languages than English are excluded since translation to English may incorporate bias.

Our team members have handled every article selection phase to resolve possible conflicts. In this process, full documentation of the papers produced by the search process, as well as the number of papers that were finally selected for our research. The results of this process are presented in Section 4. After applying the inclusion and exclusion criteria 142 relevant papers were retained in the dataset of primary studies.

Figure 2 - Overview of the search process



3.3 Keywording of Abstracts / Classification Phase

Petersen et al. [23] proposed a method for scheme classification for primary studies to answer each research question, by keywording the paper abstracts. Since we would not be able to extract all the required information for the classification schema from the abstract, we have applied the keywording technique to the full text of the manuscripts. The full text of each study was reviewed to identify the information as designed in our research.

3.4 Data Collection

During the data collection phase, we recorded the scores for a set of variables that describe each primary study. To this end, we have selected several variables that will allow us to efficiently answer the set research questions. Data collection was handled by the first two authors and possible conflicts were resolved by the other researchers. For every study, we extracted and assigned values to the following variables:

v.1. *Title* – Records the title of the paper.

v.2. *Author* – Records the list of authors of the paper.

v.3. *Publication venue* – Records the name of the publication venue.

v.4. *DOI* – Records the Digital Object Identifier (DOI) number of the paper.

v.5. *Type* – Records the type of the venue, either conference, journal, or other.

v.6. *Year* – Records the publication year of the paper.

v.8. *Keywords* – Records paper keywords provided by the author or provided by the publisher if the author does not include any.

v.9. *Publisher* – Records the publisher's name of the primary study.

v.10. *Quality attribute* – Records which quality attribute(s) are researched in the primary study. The number of quality attributes varies from at least one to many. The names of the QAs have been retrieved and recorded, based on the classification offered by ISO [23].

v.11. *Quality Metric* – Records a list of the names of quality metrics investigated in the study. The recorded metrics are marked either as *generic* (**v.11a**) or *JavaScript-related* (**v.11b**).

v.12. *Tool Availability for Quality Metrics* – Records whether a metric can be calculated by a specific tool automatically. Furthermore, the variable records tools that the primary study authors have created for the study.

v.13. *Source of Data for Validation* – Records the source of the software projects from which data were retrieved for the primary studies to evaluate the efficiency of the proposed methods (i.e., popular web pages, GitHub projects, other commercial projects)

v.14. *Application/System* – Records the application or system used to validate the proposed metrics or methods.

v.15. *Field/Task* – Records the field or task of the software development process for which the metrics are used (i.e., dependencies updating, testing, etc.).

3.5 Data Analysis

The variables v.1 to v.9 as described above, are used for documentation purposes. Table 4 presents the mapping between the variables and RQs, as well as the analysis method for each research question.

Table 4 - Mapping of Variables to RQs

Research question	Variables	Analysis Method
RQ1	v.10, v.11	Descriptives for v.10, and v.11
RQ2	v.10, v.11a, v.11b, v.15	Crosstabs between v.10, v.11a, v.11b, v.15
RQ3	v.10, v.11a, v.11b, v.12	Descriptives for v.12, Crosstabs between v.10, v.11a, v.11b, v.12
RQ4	v.10, v.11a, v.11b, v.13, v.14	Descriptives for v.13, v.14, Crosstabs between v.10, v.11a, v.11b, v.13, v.14

4. Results

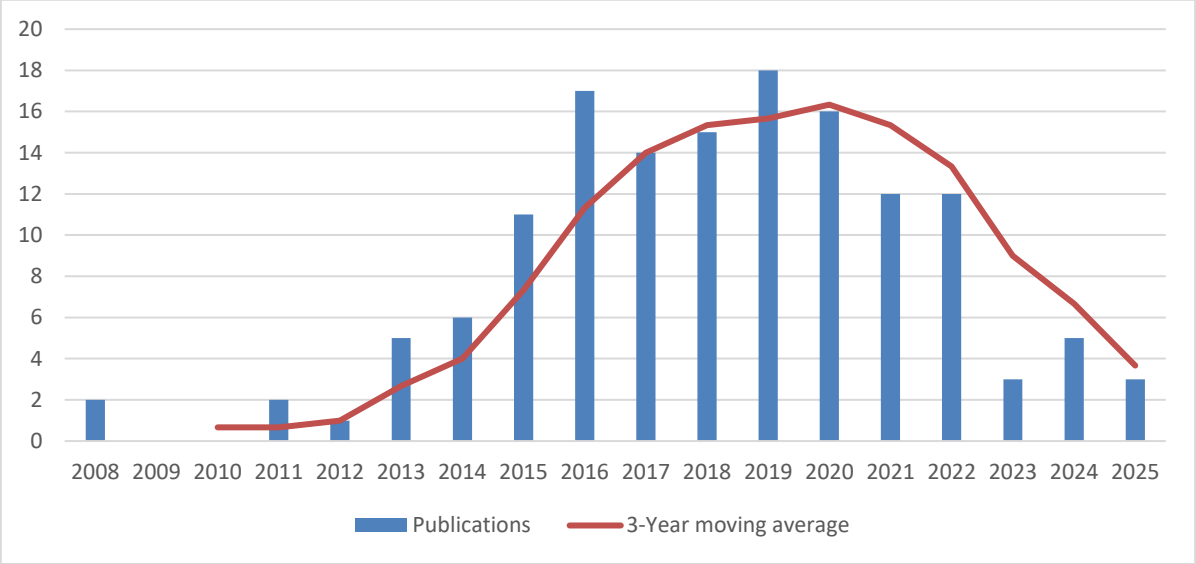
This section presents the results of this mapping study categorized by Research Question (RQ). To begin with, we present in Table 5 the selected studies per publication venue. For each publication venue, the table includes: (a) the papers returned as candidate primary studies based on the selection process, (b) the papers qualified after venue selection and filter appliance, and (c) the final number of primary studies that consist of our data set.

Table 5 - Study selection per publication venue

Name	Papers returned	Papers automatically filtered by title/abstract	Papers included
ESEC and the ACM SIGSOFT ISFSE	71	43	28
IEEE Transactions on Software Engineering	30	12	12
ACM Transactions on Software Engineering and Methodology	13	5	5
International Conference on Software Engineering	72	45	33
Empirical Software Engineering	32	15	5
Journal of Systems and Software	19	10	10
Software: Practice and Experience	42	12	6
Automated Software Engineering Conference	41	27	5
IEEE International Conference on Software Maintenance	16	8	5
International Symposium on Emp. Software Engineering and Measurement	14	5	5
International Conference on Software Process	0	0	0
Information and Software Technology	18	4	4
Software Analysis, Evolution, and Reengineering	32	21	19
IEEE Software	14	6	2
Software: Evolution and Process	4	3	3
Total	**418**	**216**	**142**

Fig. 3 presents the yearly evolution along with the 3-year moving average of the number of publications published in the field. We can observe that research related to JavaScript application quality is relatively recent starting from 2008, while actual research activity is observed after 2013 (especially from 2015) when the number of studies has increased. Thus, in the last few years, researchers have attempted to monitor the quality of JavaScript application development.

Figure 3 – Publication trend



4.1. Quality Attributes Relevant to JavaScript Applications (RQ_1)

For answering RQ_1, we first searched the full text of each manuscript for the existence of a substring related to the quality attributes of the ISO model (e.g., maintain*→ maintainability). Then we listed all quality attributes, based on the percentage of articles in which they appear in our dataset—see Figure 4. Based on the results, 63% of the primary studies mention four quality attributes: Maintainability, Security, Reliability, or Usability. The rest of the

quality attributes, i.e., Functional Suitability, Portability, Performance Efficiency, and Compatibility present shares of appearance below 7%.

Figure 4 – Frequency of Quality Attributes

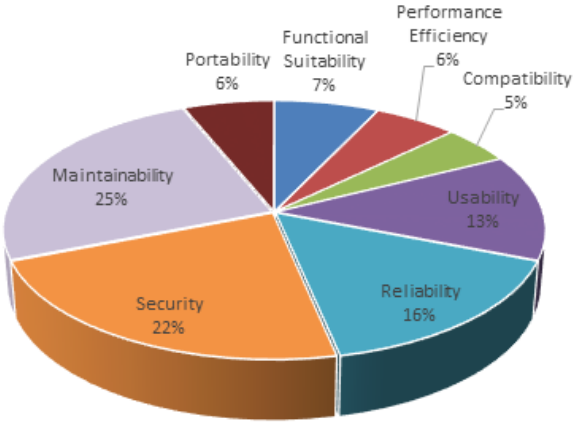


Figure 5 depicts the number of studies in which more than two quality attributes appear. Figure 4 represents, as row and column, the eight quality attributes of the ISO model. The value in each cell corresponds to the number of studies that mention both quality attributes. For example, in position (1,3) we observe that the number of studies that mention both *Functional Suitability* and *Compatibility* is 2. *Maintainability* and *Reliability are* the most common attributes that are studied together, followed by *Reliability* and *Security*. These quality attributes are discussed in most of the primary studies.

Figure 5 – First-order Quality Attributes Hierarchy Association

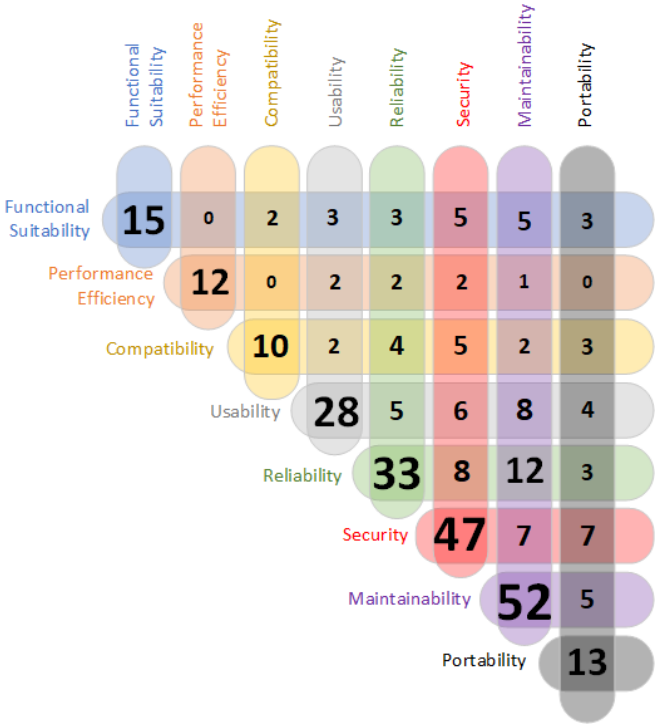
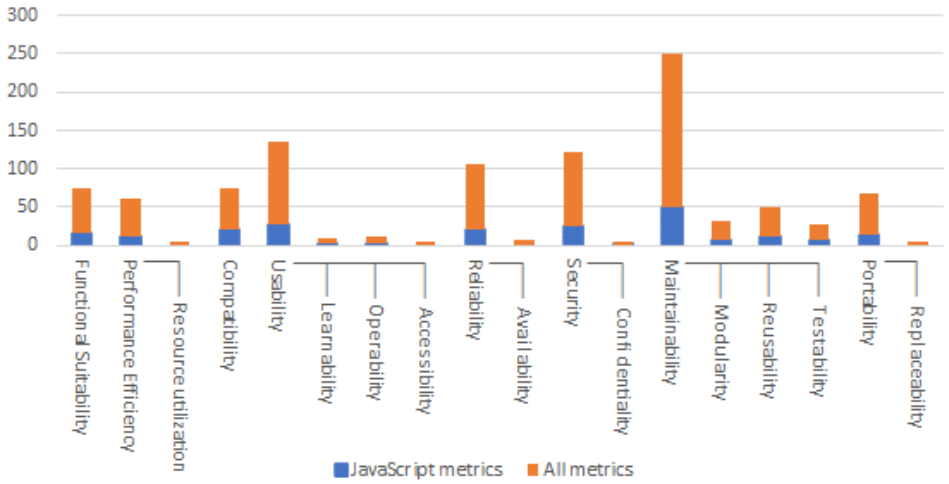


Figure 6 presents with the orange bars the frequencies (number of unique occurrences) of the different metrics proposed to assess the values of the 1st and 2nd level quality attributes of the ISO model. Furthermore, with the blue color, we denote the frequencies of the metrics across quality attributes that are JavaScript-specific (i.e., these metrics were proposed taking into consideration the unique attributes of the JavaScript programming language). The quality attributes are presented in a hierarchical order, where the low-level attributes are presented under the

group of each first-level attribute. The second-level attributes are depicted from left to right after the HL attribute they belong to. Attributes that are not associated with any metric are not included in Figure 5.

Figure 6 – Frequencies of the metrics retrieved for each quality attribute



From Figure 5 we can observe that four 2nd level attributes (*Resource utilization*, *Accessibility*, *Availability,* and *Replaceability*) are not associated with any JavaScript-exclusive metrics. Moreover, the JavaScript-related metrics are mostly proposed for the Maintainability, the Security, the Reliability, and the Usability quality attributes. The most noticeable low-level quality attributes with the largest number of metrics for both JS and non-JS are the *Modularity*, *Reusability,* and *Testability* quality attributes, which correspond to the large number of studies that focus on Maintainability. An important finding is also that for the *Operability* attribute, the number of JavaScript-related metrics exceeds the number of the non-JavaScript metrics, while concerning the *Testability* attribute, both metric groups are equal in size.

4.2. Quality Metrics for Measuring Quality in JS Applications (RQ_2)

In this section, we present the results of the mapping metrics to quality attributes. We note that the metrics listed under Table 4 are either direct measurements of the quality attributes (e.g., TIME for Performance) or proxies (e.g., LOC for Performance). By examining the full dataset as obtained from the analysis of primary studies, we recorded for each study at least one quality metric, that was either used in experiments/case studies or directly suggested by the studies as a new metric. After collecting all the metrics, the first author of the study grouped metrics referring to the same subject but presented different names. All the metrics recorded are grouped into generic and JavaScript-related metrics and presented in Appendices C and D.

Table 6 presents the occurrences of the most frequently used metrics per quality attribute, where generic metrics are separated with a horizontal line from JavaScript-related metrics. We should note that *BUGS*, *VULN,* and *CODESMELLS* have been characterized as JavaScript-related metrics, since most primary studies focus on JavaScript-explicit cases of bugs, vulnerabilities, or code smells in contrast to generic cases of these. Analytically the metrics that have been characterized as *BUGS*, *VULN,* and *CODESMELLS* can be found in the Appendices. The most frequently appearing metrics are *OBJECTS* followed by *LOC*, *COVERAGE*, *FUNCTIONS,* and *VARIABLES*. The *OBJECTS* metric in JavaScript is quite ambiguous since while the language supports objects, the definition and usage of objects are not standardized like other object-oriented languages. The *LOC* metric was also expected because it is considered a metric that measures the size of a program and is closely related to the effort required to develop and maintain source code. Regarding JavaScript-related metrics, the most frequently appearing metrics are *BUGS*, *DEPEND, VULN, MUTATIONS,* and *CALLBACKS*. Elaborating more on the most frequently appearing JavaScript-related metrics we can say that:

- The *BUGS* category includes metrics referring to JavaScript-specific problems, with the most notable *type of comparison checks*, *call site operations*, *atomicity* and *API violations*, *object redefining* including *inconsistent type warnings*, *missing user interface components,* and *violations in shared sources*.

Table 6 - Metrics with the highest number of occurrences (Occ.) per quality attribute

Usability

Metric	Occ.	Explanation
LOC	14	Lines of code and derivatives
CLASSES	8	Number of classes
CC	6	Cyclomatic complexity
FUNCTIONS	5	Number of functions
RELEASES	5	Number of releases
FILES	4	Number of files
STMENTS	4	Number of statements
COVERAGE	4	Code coverage
DEVELOPERS	4	Number of developers
FPS	3	Frames per second
CONDITIONS	3	Number of conditions
AGE	3	Release interval days
CODESMELL	26	Number of code smells
BUGS	21	Number of bugs
DEPEND	10	Number of dependencies
OBJECTS	8	Number of objects
POPUP	5	Page pop-ups
HOMEPAGE	5	Home page evaluation
MEDIA.ADV	4	Media advertisements
BUGS.DENS	3	Bugs density
BUGS.UI	3	Bugs in UI
MEDIA	3	Media invocation
UICOMPMISS	3	Missing UI components

Security

Metric	Occ.	Explanation
LOC	12	Lines of code
COVERAGE	6	Code coverage
STRINGS	4	String objects
RELEASES	4	Number of releases
AGE	4	Release interval days
FSACCESS	4	File system access
MEDIA	4	Media invocation
VULN	23	Number of Vulnerabilities
OBJECTS	18	Number of objects
BUGS	15	Number of bugs
DEPEND	14	Number of dependencies
XSS	11	Cross-site scripting
CALLBACKS	5	Call-back accepting functions
EVAL	5	Eval statements
DOM	4	DOM manipulation
OBFS	4	Obfuscations

Reliability

Metric	Occ.	Explanation
LOC	10	Lines of code and derivatives
VARIABLES	7	Number of variables
COVERAGE	6	Code coverage
FUNCTIONS	5	Number of functions
CLASSES	4	Number of classes
ISSUESO	4	Open issues
RETURN	4	Return statements
EXCEPTION	4	Undefined exceptions
BUGS	17	Number of bugs
OBJECTS	14	Number of objects
DEPEND	10	Number of dependencies
CALLBACKS	5	Call-back accepting functions
VULN	5	Number of vulnerabilities
DYNAMIC	5	Dynamic calls
CODESMELL	4	Code smells

Maintainability

Metric	Occ.	Explanation
LOC	19	Lines of code and derivatives
FUNCTIONS	13	Number of functions
LOOPS	11	Number of loops
VARIABLES	11	Number of variables
COVERAGE	9	Code coverage
CC	8	Cyclomatic complexity
CLASSES	8	Number of classes
CLONES	7	Code clones
RELEASES	6	Number of releases
FILES	5	Number of files
DEADCODE	5	Dead code
ARRAY	5	Array objects and operations
SWITCH	5	Switch statements
DEVELOPERS	5	Number of developers
AGE	4	Release interval days
EXCEPTION	4	Undefined exceptions
COMMITS	4	Number of commits
CODESMELL	30	Number of code smells
BUGS	20	Number of bugs
OBJECTS	16	Number of objects
MUTATIONS	15	Number of code mutations
DEPEND	14	Number of dependencies
CALLBACKS	12	Call-back accepting functions
EVAL	6	Eval statements
DOM	6	DOM manipulation
DYNAMIC	5	Dynamic calls
VULN	4	Number of vulnerabilities
THIS	4	This keyword statements

Performance Efficiency

Metric	Occ.	Explanation
VARIABLES	4	Number of variables
FUNCTIONS	4	Number of functions
LOC	4	Lines of Code
TIME	4	Execution time
ARRAY	4	Array objects and operations
SIZE	3	Average release size
BUGS	8	Number of bugs
OBJECTS	6	Number of objects
DEPEND	4	Number of dependencies

Functional Suitability

Metric	Occ.	Explanation
LOC	5	Lines of code
COVERAGE	3	Code coverage
CRYPTFUNC	3	Cryptographic functionality
BUGS	9	Number of bugs
DYNAMIC	2	Dynamic calls
ENCURICMP	2	EncodeURIComponent use
BUGDFLOW	2	Dataflow multi-references
DCENTRALY	2	Package degree centrality

Portability

Metric	Occ.	Explanation
LOC	4	Lines of code and derivatives
COVERAGE	4	Code coverage
RELEASES	4	Number of releases
FSACCESS	1	File system access
UTILFUNC	1	Number of utility functions
OBJECTS	4	Number of Objects
VULN	4	Number of Vulnerabilities

Compatibility

Metric	Occ.	Explanation
LOOPS	7	Number of loops
DEPEND	5	Number of dependencies
VULN	4	Number of vulnerabilities
CROSSLANG	3	Cross-language invocations
CODEDEPR	3	Code deprecation
BOM	3	Browser object manipulation
DOM	2	DOM manipulation
WEBVIEW	2	WebView object manipulation
API.USAGE	2	API usage
API.INCFG	2	API incorrect configuration

- The *VULN* category includes JavaScript-related vulnerabilities with the most common *insecure transport channels, prototype pollution* or *protection overrides*, *disclosure of HTTP headers* information, *credential leaks*, and *HTTP strict transport security failures*.
- The *DEPEND* category presents metrics related to source code dependencies. These involve the *number of direct or indirect dependencies*, *the size, and depth of dependent packages*, *unused or outdated packages*, as well as the *degree of centrality of trivial packages*.
- The *MUTATION* category includes metrics related to the changes in the code made during testing. Metrics grouped in this category involve *ARI* or *SRI* (arbitrary or systematic renaming of a single identifier), *ILS* or *DLS* (insertion or deletion of a line during testing), *SDL* or *SIL* (small deletion or insertion within a line), and other *changes of property* or *reassigned types*.
- The *CALLBACKS* category includes metrics related to JavaScript callback-accepting function combinations to achieve certain tasks. These include *synchronous, asynchronous,* and *nested callbacks*.
- The *CODESMELLS* category includes general code smell categories (e.g., *long methods*, *God classes* or *long parameter list*), as well as code smells that apply to JS or JS-like languages (e.g., *top functions, global*).

Table 7 presents a list of metrics investigated by primary studies based on the scope and the goal of the primary study, i.e., the task that is assessed with the help of metrics. This variable is extracted based on the keywords that the authors stated in the original study. Interesting findings can be extracted from Table 7 for further discussion. On the most frequently appearing categories, we can say that:

- *Testing* is the most studied phase presenting a wide diversity in the fields of study. While testing was expected to be the most popular phase as it is related to maintenance (which is the most dominant field of study for all developing processes), another popular phase is *Environment*, because JavaScript development is closely related to the environment of the application (whether this includes client-side web browsers or server-side NPM).

- The most common problem in analyzing JavaScript is the dynamic nature of the language. Thus, *dynamic* and *static* analysis are popular fields of study. In either case, JavaScript-specific metrics that relate to the dynamic nature of the language (like *CALLBACKS* and *XHR*) are present in both categories. These metrics are also included in most phases and fields, supporting the dynamic nature and popularity of JavaScript in web application development.
- *Maintenance* is the most dominant field of study for all workflows and is related to most of both generic and JavaScript-related metrics. However, fields like *DOM* and *Web browsers* present high frequencies. DOM manipulation and web browser interactions are essential in JavaScript development as developing web applications is the most common use of the language. These also include a variety of JavaScript-specific metrics that relate to the dynamic nature of the language (e.g., *NEW, EVAL, MUTATIONS*).
- *DEPEND* is a popular metric included in most phases and fields, indicating the strong expanded JavaScript ecosystem. Library and package reuse are strong research fields among the venues, that are included mostly in testing, deployment, and environment phases, but not in requirements or implementation.
- Metrics associated with specific frameworks (e.g., jQuery, Lodash, or Node). Framework-related metrics such as *LODASH* and *JQUERYEV*, are mostly included in *testing* phases and involve tasks related to debugging.

Table 7 - Metrics categorization based on the scope of the primary study

Scope	Studies	Metrics
Dependencies	[S24], [S27], [S29], [S30], [S34], [S38], [S41], [S66], [S69], [S71], [S72], [S74], [S76], [S84], [S90], [S95], [S97], [S101], [S107], [S117], [S125]	ARRAY, BLOCKS, CC, CLASSES, COMMENTS, COVERAGE, CROSSLANG, DEADCODE, DIT, ENERGY, EXCEPTION, FILES, FORKS, FSACCESS, FUNCTIONS, LOC, LOOPS, MEDIA, METHODS, NETWORK, NOA, PARM, RETURN, STARS, STMENTS, STRINGS, TIME, VARIABLES
		ANONYM, ARROW, BINARY, BUGS, DEPEND, DOM, DYNAMIC, EVAL, GLOBAL, JSDOC, LOADURL, LOC, MUTATIONS, NEW, NODEF, OBFS, OBJECTS, STRICT, VARIABLES, VULN, WITH, XSS
Asynchronous Programming	[S16], [S127]	CLASSES, CONDITIONS, COVERAGE, FUNCTIONS, LOC, LOOPS, SWITCH, TRYCATCH
		BUGS, MUTATIONS
Testing	[S3], [S5], [S15], [S16], [S17], [S18], [S25], [S35], [S44], [S50], [S80], [S85], [S86], [S88], [S91], [S94], [S98], [S103], [S113], [S120], [S123], [S126], [S138]	AGE, ARRAY, BLOCKS, BUGS, CASE, CC, CLASSES, CLONES, CODESMELL, COMMENTS, COMMITS, CONDITIONS, COVERAGE, DEADCODE, DEVELOPERS, DIRS, ENTROPY, EXCEPTION, FANIN, FANOUT, FILES, FORKS, FUNCTIONS, INSTABILITY, ISSUESC, ISSUESO, LOC, MEMORY, MI, MODULES, NOA, OBJECTS, PARM, PULLS, REGEX, RELEASES, RETURN, STMENTS, STRINGS, SWITCH, TIME, VARIABLES
		ANGFUNC, ANONYM, ARROW, BUGS, CALLBACKS, CODESMELL, DEPEND, DOM, DYNAMIC, EVAL, INTEGERS, JQUERYEV, JSON, LODASH, MUTATIONS, NATIVE, NEW, NODEF, OBFS, OBJECTS, PROMISES, STRICT, THIS, VULN, WITH, XSS
Static analysis	[S10], [S14], [S19], [S28], [S43], [S53], [S55], [S66], [S78], [S89], [S111], [S112], [S122], [S128], [S129], [S133]	BLOCKS, CC, CLASSES, CLONES, COMMITS, CONDITIONS, COVERAGE, DEADCODE, DIRS, ENTROPY, FILES, FPS, FUNCTIONS, ISSUESC, ISSUESO, LOC, ME, PARM, REGEX, RETURN, STMENTS, STRINGS, SWITCH, TD, TRYCATCH, VARIABLES
		ANONYM, ARROW, BUGS, CALLBACKS, COOKIES, DEPEND, DOM, FNRANK, GLOBAL, LOADURL, MUTATIONS, NATIVE, OBFS, OBJECTS, THIS, VULN, XHR, XSS
Dynamic Analysis	[S2], [S11], [S28], [S36], [S40], [S54], [S75], [S79], [S81], [S83], [S96], [S104], [S109], [S115], [S142]	AGE, BADGES, CC, CHARS, CLONES, CLOUD, CNGCOST, COMMITS, CONDITIONS, CORESIZE, COVERAGE, CROSSLANG, DEVELOPERS, DIRS, DOWNLOADS, ENTROPY, FILES, FOD, FORKS, FPS, FSACCESS, GITIGNORE, HEFF, HOMEPAGE, HPARM, ISSUESC, ISSUESO, LICENCE, LINTERS, LOC, MEDIA, MEMORY, MI, NETWORK, PULLS, README, RELEASES, RETURN, RTT, STARS, TIME, WATCHERS
		BUGS, CALLBACKS, DEPEND, EVAL, LOADURL, NEW, OBFS, OBJECTS, VULN, WITH, XHR

Scope	Studies	Metrics
Maintenance	[S1], [S6], [S8], [S12], [S22], [S25], [S31], [S33], [S39], [S42], [S45], [S52], [S56], [S63], [S65], [S67], [S70], [S73], [S77], [S92], [S93], [S100], [S102], [S110], [S116], [S119], [S131], [S132], [S134], [S135], [S136], [S141]	AGE, ARRAY, ASSIGN, BINARY, CC, CLASSES, COGC, COMMITS, CONDITIONS, COVERAGE, CPU, CROSSLANG, DEADCODE, DEVELOPERS, DIT, ENERGY, EXCEPTION, FILES, FSACCESS, FUNCTIONS, IDENTIFIERS, ISSUESO, LICENCE, LITERALS, LOC, LOOPS, MEDIA, MEMORY, METHODS, PARM, PURE, REGEX, RELEASES, RETURN, STMENTS, STRINGS, SWITCH, TIME, VARIABLES
		BUGS, CALLBACKS, CODESMELL, COOKIES, DEPEND, DOM, DYNAMIC, EVAL, JSON, LOADURL, MUTATIONS, NEW, NODEF, OBJECTS, PROMISES, STRICT, THIS, VULN, XHR, XSS
Performance/ Energy Efficiency	[S13], [S20], [S23], [S37], [S46], [S61], [S62], [S82], [S118], [S121]	AGE, CLONES, COMMITS, CONDITIONS, COVERAGE, DEADCODE, DEVELOPERS, ENERGY, LOC, LOOPS, ME, RELEASES, STARS, STRINGS, SWITCH, TRYCATCH, VARIABLES, WATCHERS
		CODESMELL, DEPEND, DOM, DYNAMIC, EVAL, GLOBAL, INTEGERS, MUTATIONS, OBJECT, PROMISES, XHR
Web browser	[S21], [S47], [S49], [S60], [S64], [S90], [S99], PS114], [S124]	AGE, BLOCKS, COMMENTS, CONDITIONS, CONSOLE, COVERAGE, CROSSLANG, DEADCODE, FSACCESS, FUNCTIONS, LOC, LOOPS, NETWORK, RELEASES, RETURN, STRINGS, TIME, VARIABLES
		BUGS, CALLBACKS, CODESMELL, DEPEND, DOM, DYNAMIC, EVAL, GLOBAL, JSDOC, MUTATIONS, NEW, OBFS, OBJECTS, THIS, XHR, XSS

4.3. Tool Support (RQ_3)

In this section, we present the tools found in the literature for calculating JS metrics. A total of 119 tools have been identified across 142 papers. Table 8 presents the most common tools included in the primary studies, as well as the list of the metrics calculated by these tools. A full list of the identified tools and their frequencies among the investigated venues is presented in Appendix E. This diversity of tools can be explained by the fact that most of them are custom-made and cover a small fraction of the metrics that serve the goal of the primary study. In many cases, the source code of the tool is accessible via a source code repository (in most cases the GitHub platform). It is worth mentioning that enough primary studies do not name the tools used.

Table 8 - Common metrics tools

Tool	Metrics	Studies
V8 JS engine[1]	CC, CLONES, CNGCOST, CORESIZE, COVERAGE, DIRS, FILES, FOD, FUNCTIONS, HEFF, HPARM, LOC, MI, RETURN, STMENTS	[S4], [S16], [S25], [S30], [S72], [S82]
	BUGS, DEPEND, DOM, INTEGERS, MUTATIONS, VULN	
SunSpider[2]	CASE, CLONES, COMMITS, CONDITIONS, COVERAGE, FPS, FUNCTIONS, LINTERS, LOC, MEDIA, README, RETURN, RTT, STMENTS, SWITCH	[S7], [S36], [S72], [S84], [S105], [S121]
	CODESMELL, DEPEND, DOM, MUTATIONS, OBFS, OBJECTS, VULN, XHR	
Esprima[3]	CLASSES, COMMENTS, CONDITIONS, CONSOLE, COVERAGE, CPU, DIRS, ENTROPY, FUNCTIONS, ISSUESC, LOC, LOOPS, RELEASES, SWITCH, TRYCATCH	[S61], [S64], [S68], [S91], [S131]
	ARROW, BUGS, DOM, GLOBAL, JSDOC, MUTATIONS, NEW, OBJECTS, STRICT, WITH, XSS	

[1] https://v8.dev/
[2] http://www2.webkit.org
[3] https://esprima.org/

Tool	Metrics	Studies
TAJS[4]	ARRAY, CLONES, COMMITS, DEADCODE, DIRS, ENTROPY, FILES, FUNCTIONS, ISSUESC, ISSUESO, LOC, PARM, RETURN, STMENTS, STRINGS, TIME, VARIABLES	[S12], [S40], [S72], [S80], [S120]
	BUGS, CALLBACKS, DEPEND, DOM, DYNAMIC, MUTATIONS, OBFS, OBJECTS, XHR	
SonarQube[5]	ARRAY, ASSIGN, BINARY, BUGS, CC, COGC, CONDITIONS, COVERAGE, EXCEPTION, FSACCESS, FUNCTIONS, IDENTIFIERS, LITERALS, LOC, PARM, REGEX, RELEASES, RETURN, STMENTS, SWITCH, VARIABLES	[S1], [S38], [S112], [S127]
	BUGS, DOM, LOADURL, XSS	
ESLint[6]	ARRAY, ASSIGN, BINARY, CC, CONDITIONS, COVERAGE, EXCEPTION, FSACCESS, FUNCTIONS, IDENTIFIERS, LITERALS, LOC, PARM, RELEASES, RETURN, STMENTS, SWITCH, VARIABLES	[S21], [S23], [S96], [S112]
	BUGS, CALLBACKS, DEPEND, DYNAMIC, EVAL, OBJECTS, STRICT, VULN, XSS	
Octane[7]	CASE, CLONES, COVERAGE, ENERGY, FPS, LOC, SWITCH	[S36], [S88], [S105], [S106]
	CODESMELL, MUTATIONS, OBJECTS	
JSNice[8]	CLASSES, DIT, FILES, FUNCTIONS, LOC, METHODS, NOA	[S41], [S98], [S119]
	OBJECTS, VULN	
JSClassFinder[9]	ARRAY, COGC, FILES, LOC, LOOPS, PARM	[S1], [S33], [S74]
	BUGS, CALLBACKS, EVAL, NODEF, OBJECTS, XHR	
Snyk[10]	BLOCKS, CLASSES, DEADCODE, FUNCTIONS, LOC, RETURN, VARIABLES	[S8], [S22], [S92]
	ANONYM, ARROW, BUGS, DEPEND, OBJECTS, VULN, THIS	
WALA[11]	COMMITS, CPU, DIRS, ENTROPY, EXCEPTION, FILES, ISSUESC, ISSUESO, LOC, LOOPS	[S110], [S120], [S131]
	BUGS, XSS	
JSAI[12]	ARRAY, CLONES, CPU, DEADCODE, FUNCTIONS, LOC, RETURN, STMENTS, STRINGS	[S40], [S72], [S131]
	DEPEND, DOM, MUTATIONS, OBFS, XSS	

Most tools presented in Table 8 are characterized by the authors as open source or free to use, with minor exceptions including *Understand* and *TraceAnalyzer*. Most of the non-free to use, however, include a commercial license. Overall, tool support can be considered as sufficient since many tools are presented in the primary studies covering different aspects of quality. The most frequently used tools are:

[4] https://www.brics.dk/TAJS/
[5] https://www.sonarsource.com
[6] https://eslint.org/
[7] https://github.com/laravel/octane
[8] http://jsnice.org/
[9] https://github.com/aserg-ufmg/JSClassFinder
[10] https://snyk.io
[11] https://github.com/wala/WALA
[12] https://github.com/nystrom/jsai

- *V8 JavaScript engine* is developed by Google and it provides extensive debugging and semantic analysis capabilities.
- *SunSpider* is a JavaScript benchmark that tests the performance of the core JavaScript language source code.
- *Esprima* is a high-performance, standard-compliant ECMAScript parser, used to perform lexical or syntactic analysis.
- *TaJS* is a dataflow analysis for JavaScript that infers type information and call graphs. It supports ECMAScript 5 and its standard library, the HTML DOM, and the browser API.
- *SonarQube* is an open-source platform to manage the source code quality trained to handle over 20 programming languages. It supports both static and dynamic source code analysis to identify metrics and characteristics.

Concerning the metric support by the identified tools, Table 8 presents a full crosstab list between each metric and the supported tools. Each tool can be associated with the estimation of multiple metrics. Based on these results, the metrics with the widest support are *LOC*, *FUNCTIONS,* and *COVERAGE*. Concerning the metric support by each tool identified in the venues, a full list has been included in Appendices F and G.

In total, 49 studies indicated that a tool was specifically developed for the purpose of the particular research. Furthermore, 99 studies used a single tool, while 35 used multiple tools to achieve their goals. The case of developing a new tool for metric extraction instead of utilizing existing tools is a strong indication that the current tools might not meet current metrics research requirements, leading to a lack of confidence by the authors. Núñez-Varela et. al. [30] also pinpointed this outcome, but also the results inconsistencies across different tools. The most important factors that lead to the decision to develop a new tool are the dependency on the set of metrics and languages most common tools accept, differences in metric definition, and differences in metric computation.

Table 9 – Tool comparison

Tool & repository	**Metric support**		**Performance**	**Stars**	**Forks**	**CI/CD**	**Platform compatibility**	**License**
	Generic	**JS**						
V8 JS [13]	15	6	Low	24.4k	4.2k	Yes	Npm	BSD-3-Clause
SunSpider[14]	15	8	Low	7	3	Yes	All OS	BSD/LGPL
Esprima[15]	15	11	Low	7.1k	774	No	Npm	BSD-2-Clause
TAJS[16]	17	9	Low	196	39	No	Java	Apache 2.0
SonarQube[17]	21	4	High	9.9k	2.1k	Yes	Cloud/All OS	LGPL / Mixed
ESLint[18]	18	9	Low	26.2k	4.8k	Yes	Npm	Apache 2.0
Octane[19]	7	3	High	3.9k	319	Yes	Cloud	Mixed
JSNice[20]	7	2	High	289	28	No	Cloud	Open source

[13] https://github.com/v8/v8
[14] https://github.com/WebKit/JetStream
[15] https://github.com/jquery/esprima
[16] https://github.com/cs-au-dk/TAJS
[17] https://github.com/SonarSource/sonarqube
[18] https://github.com/eslint/eslint
[19] https://github.com/laravel/octane
[20] https://github.com/brettlangdon/jsnice

Tool & repository	Metric support		Performance	Stars	Forks	CI/CD	Platform compatibility	License
	Generic	JS						
JSClass-Finder[21]	6	6	Low	65	2	No	Composer	MIT
Snyk[22]	7	7	High	5.2k	644	Yes	Cloud	Mixed
WALA[23]	10	2	Low	814	237	No	Java	Eclipse PL 2.0
JSAI[24]	9	5	Low	13	6	No	Npm	Open source

Table 9 presents a comparison of the most popular tools. It should be emphasized that the reported metric support (either generic or JavaScript-oriented) reflects only the metrics identified in this study. Regarding **tool performance**, cloud-based solutions generally achieve higher computational performance due to greater processing capacity compared to local environments. For tools offering **mixed-license options**, the paid plans typically provide enhanced computational speed and capacity. When applied to large-scale source code projects, the performance of **non-cloud tools** depends heavily on the local infrastructure available. Based on **popularity metrics** from the GitHub platform (as of September 2025), the *V8 engine*, *ESLint*, and *SonarQube* emerge as the most widely used tools, each exceeding one thousand stars and forks. It should be noted that certain tools, such as *SunSpider* and *JSAI*, no longer maintain active GitHub repositories. Concerning **continuous integration and deployment (CI/CD) support**, approximately half of the tools offer automated integration, while the remainder do not. In terms of **platform compatibility**, most tools rely on the server-side *NPM* platform, and many are cloud-based. Only a few provide **standalone implementations** supporting all major operating systems. With respect to **licensing**, all tools are open-source and free to use; however, *SonarQube*, *Octane*, and *Snyk* offer additional features via premium membership plans. Finally, in terms of **usability and support**, all tools are well-documented, maintain dedicated websites, and provide either community-driven or direct support channels. Documentation, wikis, and forums are available across all tools to assist users in troubleshooting and adoption.

4.4. Validation of Metrics (RQ4)

In this section we present the results regarding the type of validation used by the primary studies to evaluate the quality attributes (RQ4a) and regarding the systems and applications that are used for the quality assessment of the proposed methods (RQ4b).

Concerning RQ4a, we initially examined the type of validation adopted by each primary study. First it is important to notice that all 142 primary studies include a validation process related to the quality attributes monitored. Table 10 presents a summary of our findings. We observe that in all the cases the validation is empirical, aiming at investigating the efficiency with which a specific metric quantifies the corresponding factor while the theoretical validation of the proposed metrics was not preferred. In the case of empirical validation, most of the primary studies utilize data from Open-Source applications in contrast to the use of data from Industrial settings. This outcome is expected due to the "openness" of JavaScript applications that allow for the plethora of freely available, reusable applications that currently have become a trend. In a few studies, surveys that capture the expert's opinion, are also preferred but usually as a complementary validation method with quantitative data.

[21] https://github.com/aserg-ufmg/JSClassFinder

[22] https://github.com/snyk

[23] https://github.com/wala/WALA

[24] https://github.com/nystrom/jsai

Table 10 - Types of metric validation

Type of validation	#	Primary studies
Validation based on data from Industrial applications	3	[S5], [S6], [S13]
Validation based on data from Open-Source applications	139	[S1 – S4], [S7 – S12], [S14 – S142]
Validation based on expert opinion	12	[S1 – S2], [S5 – S7], [S27], [S38], [S40], [S47], [S93], [S112], [S117]

Regarding RQ4b, the data/ projects used for validation purposes present great diversity with respect to the sources of input. Table 11 presents the details of the sources of data that are used by the primary studies for the evaluation of their findings (RQ4b). The type includes scripts as published in the GitHub platform or other various repositories, npm packages, scripts derived from web pages or web applications, source code from mobile applications, scripts derived from books, tutorials, benchmarks, or custom-made applications. Known JavaScript applications, GitHub projects, webpages, scripts, and node JS packages are the main sources of validation data.

Table 11 - Types of data used

Type	#	Primary studies	Quality Attributes
GitHub projects	53	[S3-S4], [S8], [S14-S15], [S17], [S19-S20], [S28], [S31], [S39-S41], [S43], [S45], [S49], [S51], [S60], [S63], [S66], [S68], [S70-S72], [S74], [S76-S78], [S83], [S85], [S88], [S93], [S96], [S100], [S103], [S106], [S109-S110],[S113-S116],[S123],[S125-S127],[S130],[S134-S137],[S139-S140]	Security Reliability Usability Functional Suitability
npm packages	37	[S2], [S4], [S5], [S6], [S11], [S13], [S17], [S18], [S22], [S29], [S33], [S48], [S53], [S54], [S56], [S57], [S58], [S62], [S64], [S67], [S75], [S80], [S81], [S84], [S92], [S94], [S95], [S97], [S102], [S107], [S112], [S114], [S120], [S123], [S124], [S132], [S138]	Security Reliability Portability Compatibility
Web pages & web applications	25	[S7], [S9], [S12], [S16], [S26], [S27], [S32], [S33], [S36], [S42], [S44], [S47], [S55], [S69], [S86], [S90], [S108], [S111], [S117], [S118], [S121], [S122], [S123], [S131], [S137]	Security, Reliability Usability Functional Suitability
Projects from various repositories[25]	21	[S1], [S21], [S25], [S33], [S37], [S46], [S50], [S52], [S59], [S73], [S78], [S82], [S87], [S91], [S95], [S99], [S101], [S105], [S117], [S119], [S129]	Usability, Reliability, Performance Efficiency
Mobile applications	5	[S35], [S38], [S65], [S104], [S133]	Security
Books, tutorials, benchmarks, custom	3	[S24], [S61], [S98]	Reliability

GitHub and *npm* packages are used frequently to validate JavaScript quality-related research. This is a reasonable finding since the platform repositories offer access to open-source projects thus leading to an easier and more automated process of data retrieval. Harvesting data over these platforms is preferred as a less time-consuming method by most of the studies. We should note that other repositories are also used (e.g., SourceForge, GitLab, OW2) but to a much lesser extent than GitHub. These repositories are less popular than GitHub which is the dominant repository in popularity, size, and users today. Web pages and web applications are also used as a source of data with the help of special crawling tools (e.g., SunSpider) that are used to automate the data collection process. An interesting finding is that no commercial projects appear. This can be explained since open access to commercial source code is limited due to copyright restrictions.

A small number of primary studies evaluate their findings based *on Interviews, Surveys, Scripts from books/tutorials, and custom-made applications*. Interviews and surveys provide valuable empirical data, but they are time-consuming methods that require high expertise and arise validity issues based on the sample. On the other hand, ready-to-use scripts from either books, tutorials, benchmarks, or custom-made scripts may raise similar validity issues because they may not provide generic statements on the analysis. In addition to the types of data sources

[25] SourceForge, GitLab, OW2, or sources not included

used, presented in Table 11, several JS applications are frequently used to validate the findings of primary studies. In total 757 different applications were used in the studies, out of which 595 have a unique presence, while 162 appear in two or more studies (without considering the version of the application). A summary of the most frequently appearing applications/ systems is presented in Table 12. The full list of applications is presented in Appendix H, while a full list including all benchmarks with unique appearance in the studies, is included in the supplementary repository[26].

Table 12 – Applications/ systems used for validation

Application	Occurrences	Primary studies	Source
jQuery	16	[S18][S20][S29][S30][S69][S71][S72][S76][S75][S78] [S79][S88][S100][S102][S106][S107]	https://jquery.com/
React	9	[S29][S67][S69][S70][S72][S102] [S106][S107][S137]	https://react.dev/
Express.js	7	[S69][S70][S67][S72][S98][S106][S107]	https://expressjs.com/
Angular.js	7	[S78][S69][S99][S94][S102] [S106][S107]	https://angular.io/
Lodash	6	[S30][S59][S67][S69][S72][S100]	https://lodash.com/
PDF.js	6	[S9][S29][S72][S93][S97][S106][S107]	https://mozilla.github.io/pdf.js/
Moment.js	6	[S67][S69][S70][S72][S100][S102]	https://momentjs.com/
Backbone	6	[S29][S75][S94][S102][S106][S107]	https://backbonejs.org/
Vue.js	5	[S18][S69][S70][S99][S100]	https://vuejs.org/
Raytracer	5	[S15][S27][S48][S93][S97]	https://github.com/ercang/raytracer-js
Ember.js	5	[S76][S79][S78][S94][S102]	https://emberjs.com/

There is a clear tendency in the studies to measure more than one system for validation purposes (~ 9 systems per publication). Overall, the use of systems is common practice in metrics validation, while the research community seems to accept particular systems as well designed for measurement. Due to the size of the JavaScript ecosystem, new metrics and tools can be proposed for these paradigms if more products become available to the research community. Last but not least, it is necessary for new studies, metrics, and tools for common benchmarks in the JavaScript ecosystem (e.g., node-oriented studies [S19] [S41] [S123]).

5. Discussion

5.1 Interpretation of Results

The ***Quality Attribute*** that is mostly associated with JavaScript development is *Maintainability* which is in alignment with past research [17][42], that targets the prevention, detection, and correction of code faults. Additionally, *Security* is the second most studied attribute. Both *Maintainability* and *Security* are considered factors of most importance to the software lifecycle [5][S17]. In the case of JavaScript, code reuse and library inclusion are common practices that introduce third-party code, and arise in the aftermath of possible maintenance, security, and testing issues [14][47]. Regarding second-level quality attributes, we can see that most attributes are understudied, a finding that is not supported by past research [13][24]. On the second level quality attributes, *Reusability, Modularity,* and *Testability* are the most popular, including the largest number of JavaScript-explicit metrics. This is

[26] https://github.com/zozas/js-quality-metrics

expected under the scope of the language evolution towards web orientation and development trends towards code modular extension and reusability [20].

Regarding the ***software metrics*** that are used for the ***quantification/assessment*** of quality attributes, we observe that they are derived from various software artifacts, either for measuring the efficiency of static or dynamic analysis, the incorporated dependencies and reusable code, frameworks, and operation environments. This is expected as JavaScript in contrast to other languages, is capable of dynamic scripting, encourages the use of an expanded ecosystem through code reuse and dependencies, and functions either as a client or server-side language causing unpredictable behavior during execution. Under this prism, new metrics are identified to assess the quality of the software, as more effective compared to classic metrics [14][18][38][47]. An example is the wide use of libraries, dependencies, and reusable code, which incorporate complexity, vulnerabilities, and code smells, as well as possible maintenance or security issues. Furthermore, another example is the environment of operation, where through a web browser, security and reliability issues may arise due to code mutations, DOM manipulation, framework limitations, and cross-language affections. As a result, a wide dispersion of metrics is identified, and all quality attributes can be quantified by two or more metrics, mostly by weighted factor sum functions based on a regression model [27], which are the most popular methods even if these are often criticized and disputed [29]. In addition, quantification based on a single metric is rare and is regarded as a non-trivial task [7]. In general, the most frequent metrics influence the most frequent quality attributes, while the majority of the researched metrics include a type of validation, most commonly an empirical [26][39] rather a theoretical one [28]. Finally, we should mention that popular metrics suites (e.g., CK metrics) tend to have less significance than past research [24][46], while *LOC* being the oldest metric, is the most popular but never used without the contribution of other metrics [10].

On the metric utilization in relation to the ***scope*** of the article, most studies target *Testing* and *Maintenance* fields. In expansion to the above, Dynamic and Static analysis methods are often researched as means to prevent security and bug issues (that are typical on JS applications whose behavior is defined at run time), and to ensure compliance with design specifications. Also, the *Environment* in which JS applications operate, i.e., client or server side, web browsers, mobile phones, and frameworks are often investigated, a finding that is expected because the language is used for a variety of purposes that include mobile, web, desktop applications and it can be used complementary with other languages. Concerning the ***tools*** used to calculate the metrics, we observe a wide dispersion of tools that have been used by the researchers while there is no dominant tool for multipurpose quality assessment. One quarter of the researchers develop themselves a special purpose tool, a fact that indicates the lack of confidence in existing tools, probably coming because of the differences in metric definition and computation. This alone highlights the need for further research. Additionally, JavaScript related metrics tend to have limited tool support, with Google V8 engine being the most popular tool due to its debugging capabilities. Regarding the ***validation method*** used in the evaluation of each primary study we conclude that empirical validation is exclusively performed in all cases, a fact that rises questions regarding the validity of the metrics, whose conformance to the principles of measurement theory was not tested. The systems/ applications used for evaluation purposes are mostly JavaScript frameworks (i.e., jQuery, Lodash, etc.) [10][18] and server-side NPM technologies [37][48] along with node-oriented studies [S19][S41][S123]. On the other hand, there are many studies that employ data extracted from GitHub while other repositories (e.g., *SourceForge*, *OW2*) present a decline in usage. Scrapping commercial web pages is a common practice but declining over the last years. Furthermore, qualitative methods including interviews and surveys are present but to a lesser degree.

5.2 Implications for Researchers and Practitioners

In this section, we discuss the main implications of this study for researchers and practitioners, identify future directions, and provide recommendations. To assist both researchers and practitioners,

Figure 7 presents a mind-map for the quality assessment landscape. The map includes a subset of tools and metrics recorded by this study that is selected based on their popularity in the primary studies.

Concerning ***practitioners***, to exploit desired quality attributes, we provide corresponding metrics as drivers of influence for each attribute. While quantification of the latter has not been detected on our data set of primary studies, the metric selection process can be guided by the results of our study. Finally, to automate the process of collecting and estimating desired metrics, we present all tools provided by the primary studies, including the estimation support of each tool to calculate the metrics. Interpreting the mind map of

Figure 7 some useful advices for ***practitioners*** are to:

- *Select the metrics and tools that will be employed, based on the type of application under development and the scope of quality assessment.* Since JS programming language is used for a variety of purposes it is important to differentiate the quality assessment techniques applied based on the needs of the deployment environment. For web development place emphasis in DOM related and browser compatibility quality aspects, for web applications place emphasis in reuse, when it comes to mobile applications energy efficiency and performance is important. On the other hand, for server-side applications security and privacy are important.
- *Place emphasis on the static and dynamic analysis of the behavior of JS applications and apply extended Testing.* It is observed that the dynamic nature of the language may cause unexpected behavior. To mitigate this threat, practitioners should control the values of certain related metrics (i.e. XSS, vulnerabilities) and make sure they do not present sudden increases. In this direction several mutation metrics can be used towards identifying the ending time of testing process.
- *Use existing tools to automate the process of quality assessment.* Quality assessment of the various software artifacts with the use of tools can help practitioners quickly identify high-risk software components and even with the help of Linders (i.e., ESLint) proceed with targeted corrective actions.

We also encourage ***researchers*** to contribute to the following open research topics as identified by the current SLR study:

- *Quantification methods and models of quality attributes*, based on quality metrics. While many researchers associate metrics with quality attributes, a lack of a synthesized quantification of quality attributes is present, as to suggest as a future direction the development of quantification methods and predictive models.
- *Quality of 3rd part dependencies*. Despite the reusability aspect of Maintainability being often studied, most efforts focus on simple metrics (number of dependencies, loadURLs, etc.). More synthesized metrics related to monitoring the quality of dependencies (i.e., dependencies causing security vulnerabilities), and their updates (i.e., metrics appointing the need of updates) are needed as reuse is the current trend in JS development [19].
- *Development of JS-specific benchmarks for monitoring the quality of applications*. More work is required to develop a ground truth tool that can be widely used for quantifying JavaScript quality attributes. This shortcoming arises mostly from the different methodologies that are used to capture JS quality the various metrics, rulesets, and score mechanisms to quantify it. Currently, we recorded several tools, none of which recognized as a state-of-the-art solution for JS applications, that are either general purpose tools (e.g., SonarQube) or focus on specific aspects of quality (e.g., Snyk) a fact that contributes to the lack of confidence that is observed on existing tools. In this direction, providing a benchmark that can calculate a variety of both general purpose and JS- oriented quality metrics for the different scopes of implementation, as presented in Table 6, would help towards increasing the confidence of the community in using existing tools to monitor the quality of JS applications instead of developing new ones.
- *Theoretical Validation of Metrics.* The theoretical validation of new metrics is necessary for testing the efficiency and correctness of the metric. In this direction it is suggested apart from the empirical validation to place emphasis on the theoretical validation aiming at mathematically proving that the proposed metric holds basic properties of software measurement (e.g., non-negativity, normalization, etc.) [9].
- *Systematic efforts for JS-oriented metric research*. From the findings of this study, we observe that research in the field is mainly focused on efforts to improve a process related to JS – application development (i.e., static analysis, application execution, DOM program representation) and not a quality attribute or a quality metric per se (quality attributes and metrics are used/mentioned as side effects most of the times). The targeted,

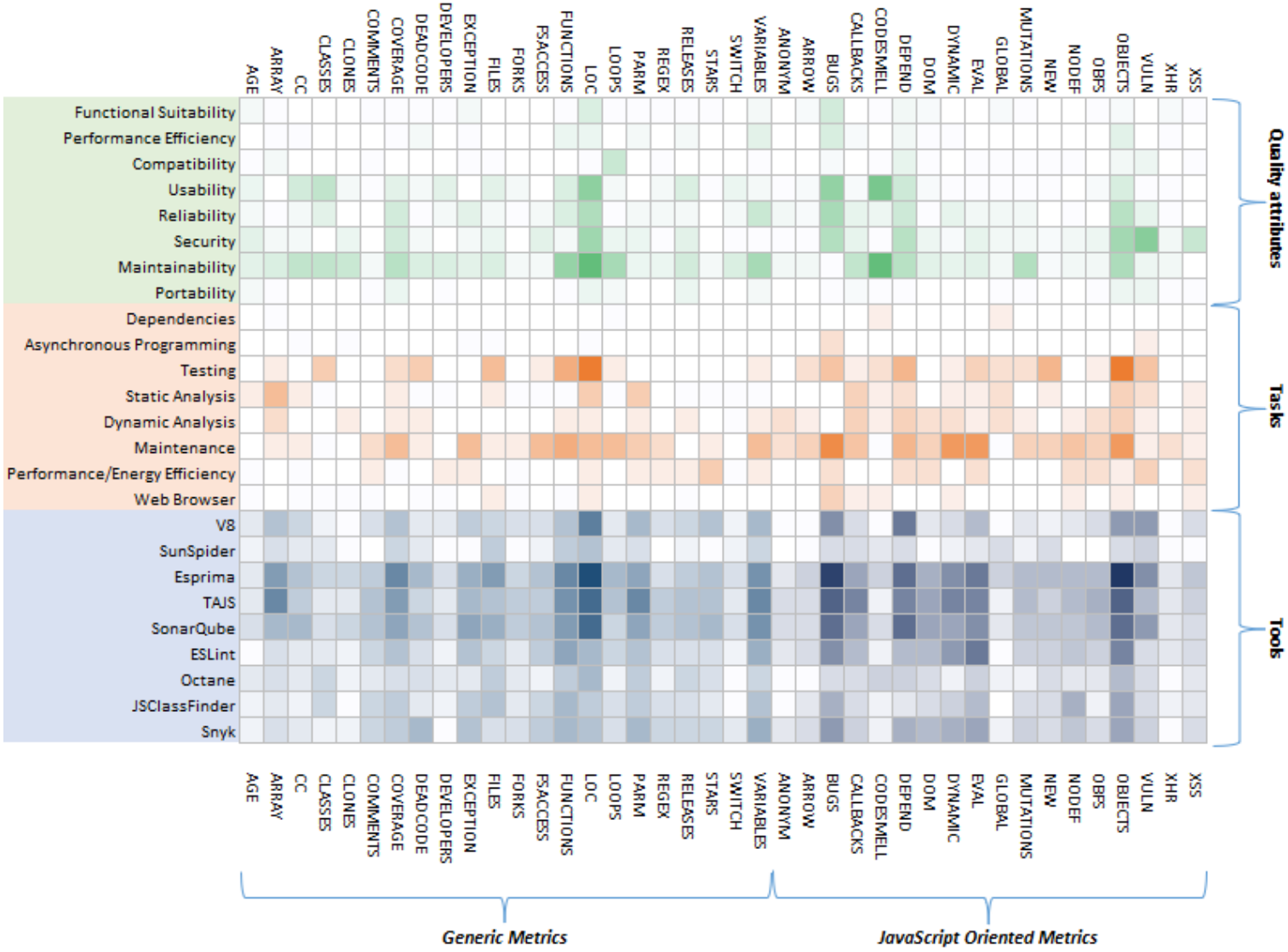
thorough investigation of *particular quality attributes and the associated metrics* is necessary for formulating a research corpus contributing towards the sustainability of the applications and the elimination of the unexpected evolution of these applications.

In addition, a full taxonomy is provided in Appendix I, while Table 9 presents a comparative performance evaluation of the most popular tools identified in the reviewed venues. The taxonomy table serves as a crosstab guide for both researchers and practitioners, linking metrics, quality attributes, tasks, and supported tools, and should be interpreted in conjunction with Figure 7. The latter highlights the most frequently cited quality attributes, tasks, and tools across the literature.

To illustrate the practical applicability of Figure 7, for diverse purposes, stakeholders and objectives we present the following example scenarios for both researchers and practitioners:

- Scenario 1 – Quality-focused analysis (i.e Security analysis): A corporation concerned with the Security quality attribute of its source code may focus on metrics such as LOC, BUGS, DEPEND, OBJECTS, VULN, and XSS. Tools like Esprima, TAJS, V8, and SonarQube support all these metrics individually and can be selected accordingly.
- Scenario 2 – Task- focused analysis (i.e supporting the testing process of JS): A developer aiming to test source code and assess Reliability may focus on metrics including CLASSES, COVERAGE, LOC, VARIABLES, BUGS, DEPENDENCIES, and OBJECTS. Tools such as Esprima, Octane, and JSClassFinder provide partial or full support for these metrics.
- Scenario 3 – Tools- focused analysis (i.e. Using TAJS for assesing Performance Efficiency): A practitioner already using specific tools for metric measurement, for example TAJS, may explore Performance Efficiency in tasks such as web browser applications by focusing on metrics supported by the tool, including LOC, BUGS, CALLBACKS, NODEFF, and OBJECTS.
- Scenario 4 – Metrics-focused analysis (i.e. Research exploration with existing metrics datasets): A researcher with sufficient datasets on specific metrics, for instance including OBFS metric, may investigate its impact on quality attributes such as Usability, Security, and Maintainability across different tasks—most notably Testing and Dynamic Analysis. Furthermore, the researcher can compare metric measurements across various supporting tools, including TAJS, SonarQube, and Esprima.

Figure 7 – JS Quality Attributes, Tasks and Tools heatmap



6. Threats to Validity

In this section, we present the threats to validity that concern our study, based on the guidelines proposed by Ampatzoglou et. al. [4].

In terms of *study-selection validity*, restricting retrieval to primary studies from selected publication venues rather than conducting broad searches across digital libraries may have omitted relevant work published elsewhere. This risk is intentional and stems from two factors: (a) the large volume of irrelevant publications typically retrieved through broad searches, and (b) concerns regarding the overall quality of such publications. Our goal was to collect only high-quality studies, and top-tier venues often provide richer indexing and editorial filtering that can identify rigorous contributions not always evident from authors' metadata. Therefore, we employed a narrow, venue-focused search strategy for JavaScript quality metrics—a focused topic within software engineering whose most credible research is concentrated in a small set of high-quality outlets (e.g., *Empirical Software Engineering*, *Journal of Systems and Software*, ICSE, ESEM; see Appendix A). We acknowledge this may reduce recall but increase the precision and trustworthiness of included studies. [27][28][29].

On the *data* validity, and more specifically on the search string construction, the current small number of keywords used to build search string might lead to missing primary studies in which the authors have not used common terms related to our focus of study. Furthermore, some primary studies do not include keywords at all. In the latter case, the publisher of each study provides keywords as means to categorize each study. In variable *v.8.* we have collected the full list of keywords, either provided by the authors or the publishers. Under this prism, we believe that it is highly unlikely for either the authors or the publishers not to have used the corresponding terms in the full text of each study. Additionally, at least two authors performed data collection and analysis to examine the results of each other, as a means to reduce the possibility of data collection inaccuracies, or the risk to *exclude relevant articles*. Lastly, from our searching space we have excluded *grey literature* since the goal of the study was imposing the use of only a limited number of journals and conferences that would guarantee the quality of the obtained papers.

In addition, concerning the *study selection bias*, we believe that publication bias does not exist, since the communities that publish in the selected venues cover the whole spectrum of software engineering research. Also, concerning data synthesis, frequency analysis and cross-tabulation are objective methods less prone to researcher bias.

Finally, concerning *repeatability* of this study, we believe that the current study can be easily replicated since the study protocol is extensively described in Section 3. Moreover, the data collection and analysis process as described in the same section involves limited subjective judgement to ensure automation of the process.

7. Conclusions

In this mapping study, we performed automatic searches in 7 journals and 8 conferences of high quality for the selection of relevant studies for JavaScript application quality assessment. A total of 142 studies were selected and analyzed in order to answer our research questions. This study provides evidence that Maintainability, Security and Reliability quality attributes are mostly studied. There are also found in literature a lot of JavaScript- specific related metrics that should be taken into account, while it is expected that new metrics will be proposed contributing to the assessment of the dynamic nature of the language. Additionally, the fact that the applications depend greatly on 3rd party libraries highlights the need to assess the quality of these dependencies. Our findings reveal a wide dispersion of tools and metrics employed, therefore we believe that practitioners and researchers in the field are in need of ground- truth unified benchmarks and tools to monitor the quality of JavaScript applications.

References

1. Alves, V., Niu, N., Alves, C., Valenca, G., 2010. Requirements engineering for software product lines: a systematic literature review. Inf. Softw. Technol. Elsevier 52 (8), 806–820.
2. Alkharabsheh, K., Crespo, Y., Manso, E., Taboada, J. 2018. Software Design Smell Detection: a systematic mapping study. Software Quality Journal (2018): 1-80.
3. Amanatidis, T., Chatzigeorgiou, A., 2016. Studying the evolution of PHP web applications. In Information and Software Technology, 72, 48-67.
4. Ampatzoglou, A., Bibi, S., Avgeriou, P., Verbeek, M., & Chatzigeorgiou, A. (2019). Identifying, categorizing and mitigating threats to validity in software engineering secondary studies. Information and Software Technology, 106, 201-230.
5. Andreasen, E., Gong, L., Møller, A., Pradel, M., Selakovic, M., Sen, K., Staicu, C. 2017. A Survey of Dynamic Analysis and Test Generation for JavaScript. ACM Comput. Surv. 50, 5, Article 66 (November 2017), 36 pages.
6. Abdellatief, M., Sultan, A., Ghani, A., Jabar, M., 2013. A mapping study to investigate component-based software system metrics. J. Syst. Softw. 86 (3), 587–603.
7. Arvanitou, E., Ampatzoglou, A., Chatzigeorgiou, A., Galster, M., Avgeriou, P. 2017. A Mapping Study on Design-Time Quality Attributes and Metrics. Journal of Systems and Software. 127. 52-77.
8. Basili, V.R., Caldiera, G., Rombach, H.D., 1994. Goal question metric paradigm. In: Encyclopedia of Software Engineering. John Wiley & Sons, pp. 528–532.
9. Briand, L., Daly, J., Wüst, J., 1999. A Unified Framework for Coupling Measurement in Object-Oriented Systems, Transactions on Software Engineering, IEEE Computer Society, 25 (1), pp. 91-121, January.
10. Boekesteign, J. 2012. JavaScript code quality analysis, Master thesis, TU Delft, Faculty of Electrical Engineering, Mathematics and Computer Science, Department of Software and Computer Technology, Netherlands
11. Cai, K., Card, D., 2008. An An analysis of research topics in software engineering –2006. J. Syst. Softw. 81 (6), 1051–1058 Elsevier.
12. Chidamber, S., Kemerer, C., 1994. A metrics suite for object-oriented design. Trans. Softw. Eng. IEEE Comput. Soc. 20 (6), 476–493
13. Eckhardt, J., Vogelsang, A., Fernandez, D.M., 2017. Are "non-functional" requirements really non-functional? An investigation of non-functional requirements in practice. In: International Conference on Software Engineering (ICSE 2016), IEEE Computer Society, pp. 832–842.
14. Elberzhager, F., Münch, J., Tran, N., 2012. A systematic mapping study on the combination of static and dynamic quality assurance techniques. Inf. Softw. Technol. 54 (1), 1–15 Elsevier
15. Estdale, J., Georgiadou, E., 2018. Applying the ISO/IEC 25010 Quality Models to Software Product, in: Larrucea, X., Santamaria, I., O'Connor, R.V., Messnarz, R. (Eds.), Systems, Software and Services Process Improvement. Springer International Publishing, Cham, pp. 492–503.
16. Fenton, N., Bieman, J. 2014. Software Metrics: A Rigorous and Practical Approach, Third Edition (3rd. ed.). CRC Press, Inc., USA. page 11
17. Garousi, V., Küçük, B., 2018, Smells in software test code: A survey of knowledge in industry and academia, Journal of Systems and Software, Volume 138, pages 52-81.
18. Gizas, A., Christodoulou, S., Papatheodorou, T. 2012. Comparative evaluation of javascript frameworks. In Proceedings of the 21st International Conference on World Wide Web (WWW '12 Companion). Association for Computing Machinery, New York, NY, USA, 513–514.
19. Goel, B., Bhatia, P. 2013. Analysis of reusability of object-oriented systems using object-oriented metrics. SIGSOFT Softw. Eng. Notes 38, 4 (July 2013), 1–5.
20. Hafiz, M., Hasan, S., King, Z., Wirfs-Brock, A. 2016. Growing a language: An empirical study on how (and why) developers use some recently-introduced and/or recently-evolving JavaScript features, Journal of Systems and Software, Volume 121, pages 191-208.
21. Halstead, M.H., 1977. Elements of software science. Elsevier Science Inc. USA, New York.
22. ISO/IEC IS 9126-1. 2001. Software Engineering - Product Quality – Part 1: Quality Model. International Organization for Standarization, Geneva, Switzerland.
23. ISO: ISO/IEC 25010:2011, Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.
24. Jabangwe, R., Börstler, J., Šmite, D., Wohlin, C., 2004. Empirical evidence on the link between object-oriented measures and external quality attributes: a systematic literature review. Empirical Softw. Eng. 20 (3), 640–693 Springer.
25. Karanatsiou, D., Li, Y., Arvanitou, E., Misirlis, N., Wong, W. 2019. A bibliometric assessment of software engineering scholars and institutions (2010–2017), Journal of Systems and Software, Volume 147, pp 246-261
26. Kitchenham, A., 2010. What's up with metrics? A preliminary mapping study. J. Syst. Softw. 83 (1), 37–51 Elsevier.

27. Kitchenham, B., Brereton, P., Turner, M., Niazi, M., Linkman, S., Pretorius, R., Bud- gen, D., 2009. The impact of limited search procedures for systematic literature reviews: a participant-observer case study. In: 3rd International Symposium on Empirical Software Engineering and Measurement. IEEE Computer Society, Lake Buena Vista, pp. 336–345. FLorida, 15 –16 October.
28. Kitchenham, B., Brereton, P., Turner, M., Niazi, M., Linkman, S., Pretorius, R., Bud- gen, D., 2010. Refining the systematic literature review process –two participant-observer case studies. Empirical Softw. Eng. 15 (6), 618–653 Springer.
29. Kitchenham, B., Brereton, P., Budgen, D., Turner, M., Bailey, J., Linkman, S., 2009. Systematic literature reviews in software engineering: a systematic literature review. Inf. Softw. Technol. 51 (1), 7–15 Elsevier.
30. Kitchenham, B., Madeyski, L., Budgen, D., 2023. SEGRESS: Software Engineering Guidelines for REporting Secondary Studies, in IEEE Transactions on Software Engineering, vol. 49, no. 3, pp. 1273-1298, 1 March 2023.
31. Kupiainen, E., Mäntylä, M.V., Itkonen, J., 2015. Using metrics in Agile and Lean Soft- ware Development – A systematic literature review of industrial studies. Inf. Softw. Technol. Elsevier 62, pp. 143-163.
32. Li, W., Henry, S., 1993. Object-oriented metrics that predict maintainability. J. Syst. Softw. 23 (2), 111–122 Elsevier.
33. Mahdavi-Hezavehi, S., Durelli, V., Weyns, D., Avgeriou, P., 2017. A systematic literature review onmethods that handle multiple quality attributes in architecture-based self-adaptive systems, Information and Software Technology, 90 (2017): 1-26.
34. Misra, S., Cafer, F. 2012. Estimating Quality of JavaScript. International Arab Journal of Information Technology. 9. 535-543.
35. Nuez-Varela, A., Prez-Gonzalez, H., Martnez-Perez, F., Soubervielle-Montalvo, C. 2017. Source code metrics. J. Syst. Softw. 128, C (June 2017), 164–197.
36. Oliveira, E., Fernandes, E., Steinmacher, I., Cristo, M., Conte, T., Garcia, A. 2020. Code and commit metrics of developer productivity: a study on team leaders' perceptions. Empir Software Eng.
37. Oriol, M., Marco, J., Franch, X., 2014. Quality models for web services: a systematic mapping. Inf. Softw. Technol. 56 (10), 1167–1182 Elsevier.
38. Park, C, Lee, H, Ryu, S. 2018. Static analysis of JavaScript libraries in a scalable and precise way using loop sensitivity. Softw Pract Exper. 48: 911– 944.
39. Passier, H., Stuurman, S., Pootjes, H. 2014. Beautiful JavaScript: how to guide students to create good and elegant code. In Proceedings of the Computer Science Education Research Conference (CSERC '14). Association for Computing Machinery, New York, NY, USA, 65–76.
40. Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M., 2008. Systematic mapping studies in software engineering. In: 12th International Conference on Evaluation and Assessment in Software Engineering (EASE'08). Bari, Italy, British Computer So- ciety Swinton, pp. 68–77. 26 –27 June.
41. Petersen, K., Vakkalanka, S., Kuzniarz, L., 2015. Guidelines for conducting systematic mapping studies in software engineering: An update, Information and Software Technology, Volume 64, Pages 1-18
42. Radjenović, D., Heričko, M., Torkar, R., Živkovič, A. 2013. Software fault prediction metrics. Inf. Softw. Technol. 55, 8 (August 2013), 1397–1418.
43. Ramesh, V., Glass, R., Vessey, I. 2004. Research in computer science: an empirical study. J. Syst. Softw. 70, 1–2 (February, 2004), 165–176.
44. Riaz, M., Mendes, E., Tempero, E., 2009. A systematic review on software maintainability prediction and metrics. In: 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM'09). IEEE Computer Society, Florida, USA, pp. 367–377. 15-16 October.
45. Richards, G., Hammer, C., Burg, B., Vitek, J., 2011. The Eval That Men Do. In: Mezini M. (eds) ECOOP 2011 – Object-Oriented Programming. ECOOP 2011. Lecture Notes in Computer Science, vol 6813. Springer, Berlin, Heidelberg
46. Saraiva, J., Barreiros, E., Almeida, A., Lima, F., Alencar, A., Lima, G., Soares, S., Castor, F., 2012. Aspect-oriented software maintenance metrics: a systematic mapping study. In: 16th International Conference on Evaluation & Assessment in Software Engineering (EASE 2012). IET, pp. 253–262.
47. Tahir, A., MacDonell, S.G., 2012. A systematic mapping study on dynamic metrics and software quality. In: 28th IEEE International Conference on Software Maintenance (ICSM). IEEE Computer Society, Riva del Garda, Trento, Italy, pp. 326–335. 23-28 September.
48. Vargas, J.A., García-Mundo, L., Genero, M., Piattini, M., 2014. A systematic mapping study on serious game quality. In: 18th International Conference on Evaluation and Assessment in Software Engineering (EASE '14), London, UK, pp. 13–14. Article 15, ACM May.
49. Varela-Nuñez, A., Pérez-Gonzalez, H., Martínez-Perez, F., Soubervielle-Montalvo, C. 2017. Source code metrics: A systematic mapping study Journal of Systems and Software. (128). 164-197.
50. Zhang, H., Babar, M., Tell, P., 2011. Identifying relevant studies in software engineering. Inf. Softw. Technol. 53 (6), 625–637 Elsevier.

51. Zozas, I., Bibi, S., Ampatzoglou, A., Sarigiannidis, G.P., 2019. Estimating the Maintenance Effort of JavaScript Applications, SEAA 2019: 212-219.