



A Systematic Mapping of Quality Models for AI Systems, Software and Components

Mohamed Abdullahi Ali ^{1,*}, Ng Keng Yap ^{1,2}, Abdul Azim Abd Ghani ¹, Hazura Zulzalil ¹,
Novia Indriaty Admodisastro ¹ and Amin Arab Najafabadi ¹

¹ Department of Software Engineering and Information Systems, Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang 43400, Selangor, Malaysia

² Institute for Mathematical Research, Universiti Putra Malaysia, Serdang 43400, Selangor, Malaysia

* Correspondence: mrmoo2030@gmail.com

Abstract: Recently, there has been a significant increase in the number of Artificial Intelligence (AI) systems, software, and components. As a result, it is crucial to evaluate their quality. Quality models for AI have in fact been proposed, but there is a lack of Systematic Mapping Studies (SMS) for quality models in AI systems, software, and components. The goal of this paper is to understand, classify, and critically evaluate existing quality models for AI systems, software, and components. This study conducts an SMS to investigate quality models proposed by various authors in the past. The study only found quality models for AI systems and software. So far, the SMS has revealed no work on AI software component quality models. Finally, the limitations of the quality models and the implications for future research and development efforts are discussed.

Keywords: quality model; artificial intelligence software and systems; data driven software components; software components



Citation: Ali, M.A.; Yap, N.K.; Ghani, A.A.A.; Zulzalil, H.; Admodisastro, N.I.; Najafabadi, A.A. A Systematic Mapping of Quality Models for AI Systems, Software and Components. *Appl. Sci.* **2022**, *12*, 8700. <https://doi.org/10.3390/app12178700>

Academic Editor: Paolino Di Felice

Received: 15 June 2022

Accepted: 6 August 2022

Published: 30 August 2022

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2022 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

The concept of Artificial Intelligence (AI) is not new [1] and originated in the 1950s. AI can be defined as the effort to automate intellectual tasks normally performed by humans. In addition, AI is a general field and has several branches, but the most popular fields are Machine Learning (ML) and Deep Learning (DL). ML is trained rather than explicitly programmed. Recently, ML has rapidly become the most popular and prominent field in AI. This paradigm shift is driven by the availability of faster and more resilient hardware, as well as being fuelled by the amount of data that exists today.

Software engineers are normally focused on practical engineering concerns. Currently, software engineering research and communities have fallen under the spell of AI [2]. AI is defined as making machines intelligent, and Software Engineering (SE) is the process of defining, designing, and deploying for complex and challenging systems. As ML algorithms in various fields are in great demand, the development of machine learning software systems (MLS) is rapidly increasing [3]. In addition, Shafiq et al. [4] highlighted that the software development industry is rapidly adopting ML to transition modern-day software systems towards highly intelligent and self-learning systems. The quality of MLS is different from that of conventional software systems in the sense that it depends on the amount and distribution of training data in a model which learns and inputs data during operation [3]. This is a major challenge for the quality assurance of MLS development for enterprises.

However, there are existing secondary studies on the quality of AI software [5], as well as surveys on the quality of ML software [6], the literature does not appear to currently contain any systematic mapping study (SMS) on software quality for AI-based software, systems, and components. This has motivated to conduct a broader SMS to provide an up-to-date, holistic, and comprehensive view of the state-of-the-art of software quality for

AI software, systems, and components. This has motivated us to conduct a broader SMS to provide an up-to-date, and comprehensive view of the existing software quality models for AI software, systems, and components. This has the aim of analysing existing quality models, and gaps concerning quality for AI software, systems, and components.

The remainder of the paper is organized as follows: Section 2 introduces background of the study. Section 3 provides related works. Section 4 discusses the review method. Section 5 presents analysis and results. Section 6 provides limitations and future works. Section 7 presents threats of validity. Finally, Section 8 offers a conclusion.

2. Background of the Study

This section briefly presents the context in which the review is placed. To achieve this, synergies between SE and AI and definitions of AI systems, software, and components are discussed. Next, several types of software quality models are presented, namely, traditional software, and software components. Finally, this section discusses related works on AI and ML Software Components.

2.1. On the Synergies between Software Engineering (SE) and Artificial Intelligence (AI)

The meanings of AI and SE have been closely related since their beginning. AI methods have been used to support SE tasks (AI4SE), and SE methods have been used to develop AI software (SE4AI). Perkusich et al. [7] recently defined AI4SE as a portfolio of SE techniques that “explore data (from digital artefacts or domain experts) for knowledge discovery, reasoning, learning, planning, natural language processing, perception, or supporting decision-making”.

The development of AI4SE has been driven by the rapid increase in size and complexity of software systems and, in consequence, of SE tasks. Wherever software engineers reached their cognitive limits, “automatable” methods were the subject of research [7]. While searching for solutions, the SE community observed that a number of SE tasks can be formulated as data analysis (learning) tasks and thus can be supported, for example, with ML algorithms.

The SE4AI applications were limited to simply implementing AI algorithms as standalone programs [8]. As the size and complexity of AI-based software systems grew, as did their practical and commercial applications, more advanced SE methods became necessary. The breakthrough occurred when AI components were integrated into well-established software systems, such as expert systems or driving control. It quickly became clear that, due to the unique nature of AI (e.g., reliance on learning data), traditional SE methods were no longer appropriate (e.g., leading to technical debt [9]). This necessitated the revision of traditional SE paradigms and methods, as well as the development of new ones.

2.2. Definitions on AI Systems, Software, and Components

According to Pons and Ozkaya [10], “AI systems and in particular those with ML elements are systems that learn from data for decision-making, hence are not designed to comply with conventional requirements specifications”. Sculley et al. [9] also stated that AI systems involve the interpretation of external data, learning, and performing intelligent tasks.

Based on the European Commission definition, AI-based systems can be purely one of two types. The first is software-based, acting in the virtual world (e.g., voice assistants, image analysis software, search engines, speech and face recognition systems). Alternatively, AI can be embedded in hardware devices (e.g., advanced robots, autonomous cars, drones, or Internet of Things applications) [11].

There are numerous definitions of AI software in the literature. AI software uses techniques from the field of AI [12]. Practitioners and academics do not use a standard term to name systems that incorporate AI/ML/DL capabilities. Software that applies AI techniques can be referred to as “AI software” [13] or “AI-based software” [14]. AI software, specifically ML implementations, differ from traditional software in that their logic is not

explicitly programmed, but rather automatically created by learning from data [15]. Foidl and Felderer [16] defined an ML-based software system “as any kind of system that applied algorithms to data and used ML models for making intelligent decisions automatically based on discovered relationships, patterns and knowledge from data”. In [17], the authors defined ML software as being developed by machine learning (that is, it is trained software). Similarly, Braiek and Khomh [18], argued that an ML program is a software implementation of a statistical learning algorithm that requires substantial expertise in mathematics (e.g., linear algebra, statistics) to understand its internal functions. ML software may train using ML models, and this is called ML software or Deep Learning (DL) models, and such software can be referred to as DL software. DL-based software (in short, DL software) integrates DL models trained using a large data corpus with DL programs written based on DL frameworks such as TensorFlow and Keras [19].

Lastly, AI software components will be defined. Ahmad et al. [20] presented a Componentized Neural Network (CompoNet). The CompoNet facilitates the development of neural network-based software systems by designing and implementing neural networks as software components. These components can be reused effortlessly across different applications with minimal programming time. Finally, these components can be utilized using standard application programming interfaces (API).

Moreover, Siebert et al. [21] defined AI software components, specifically ML software components as data-driven software components. A data-driven software component is a piece of software that solves a given task (e.g., image segmentation, sentiment analysis, classification, etc.), using methods from data science, such as ML, data mining, natural language processing, signal processing, statistics, etc. The functionality of data-driven software components (or at least part of it) is not entirely defined by the programmer in the classical way (by programming it directly) but is derived (i.e., learned) from data.

2.3. Traditional Software Quality Models

Traditionally, software systems are constructed deductively, by writing down the rules that govern system behaviours as program code [22]. Similarly, Kuwajima and Ishikawa [23] stated that the traditional software is based on logical design and rules.

This section discusses some previous studies of traditional software quality models such as McCall [24], Bohem [25], Grady [26], ISO 25010 [27], and [28–33]. Firstly, McCall [24] proposed a software quality model and categorized it from three perspectives: product operation, product revision, and product transition. The model contains several quality attributes: correctness, reliability, efficiency, integrity, usability, maintainability, flexibility, testability, portability, reusability, and interoperability.

Bohem [25] improved McCall’s quality model by proposing a quality model with quantitative evaluation. The model in that study contains the following quality attributes: portability, reliability, efficiency, usability, testability, understanding, and flexibility.

Grady [26] conducted a study on a quality model for conventional software. The study contains five quality characteristics, namely: functionality, usability, reliability, performance, and supportability. The main limitation of the study is that it is only limited to the quality characteristics of traditional software evaluation.

ISO 25010 [27] is a leading quality model for assessing software products. The study contains eight quality characteristics for software products, namely, functional suitability, compatibility, performance efficiency, security, maintainability, portability, reliability, and usability.

Furthermore, ISO 9126 is an international standard for assessing the quality of software products [28]. This standard contains one international standard, namely, ISO 9126-1 [28], and three technical reports, which are ISO TR 9126-2 [29], ISO TR 9126-3 [30], and ISO TR 9126-4 [31]. ISO 9126-1 contains external and internal quality characteristics and quality-in-use. External and internal quality characteristics are functionality, reliability, usability, efficiency, maintainability, and portability. Quality-in-use comprises four sub-characteristics, which are effectiveness, productivity, safety, and satisfaction. ISO TR 9126-2

contains external metrics for software products, while ISO TR 9126-3 contains internal metrics for software products. ISO TR 9126-4 contains quality-in-use metrics.

Next, ISO/IEC 25000 is a series of standards used for the evaluation of software products and is also known as SQuaRE (System and Software Quality Requirements and Evaluation) [32]. ISO/IEC 25000 is an evolution of the ISO 9126 standard, which was used for software product evaluation. This standard contains four parts, namely, ISO/IEC 2501n, ISO/IEC 2502n, ISO/IEC 2503n, and lastly ISO/IEC 2504n.

In addition, Dromey [33] presented a quality model for software products and processes. The model was organized into three models, namely, the implementation model, requirement model, and design model. The implementation model contains several quality characteristics, namely, functionality, reliability, efficiency, usability, process maturity, maintainability, portability, and reusability.

Traditional software quality models such as McCall [24], Bohem [25], Grady [26], ISO 25010 [27], and [28–33] were based on software system-level quality characteristics, but not at the component level [34]. For instance, fault tolerance quality characteristics typically evaluate at the system level, rather than the software component level. However, some quality characteristics are appropriate to measure software components, others are not [34]. The next section discusses in detail existing quality models of software components.

2.4. Software Component Quality Models

Firstly, there will be some definitions of software components from the literature. A software component quality model aims to evaluate the quality of the component level [35]. According to Szyperski et al. [36], a software component is a unit of composition with contractually specified interfaces and explicit context dependencies only.

Despite the definition, software component quality models should be tailored to use only the characteristics that apply to components [36]. Therefore, the quality of software components is measured via interfaces due to the unavailability of information on the component to the third parties [37]. This kind of measurement will increase the understandability of component reusability. Moreover, external metrics are based on whatsoever information is available regarding the components such as interface and component documentation [38].

This section will discuss several existing studies of software component quality models, such as [39–54]. Bertoa and Vallecillo [39] identified commercial off-the-shelf (COTS) component quality characteristics based on ISO 9126, namely, functionality, reliability, usability, efficiency, maintainability, and portability.

Rawashdeh and Matakah [40] proposed a quality model specifically for COTS component evaluation and selection. The quality models contain the following quality characteristics: functionality, reliability, usability, efficiency, maintainability, and manageability.

Upadhyay et al. [41] carried out a study of the software component quality model (SCQM), which aims to evaluate the quality of software components. The SCQM contains eight main quality characteristics, which are functionality, reliability, usability, efficiency, maintainability, portability, reusability, and traceability, and each of these have sub-characteristics.

In [42], the authors proposed a component quality model for COTS components based on ISO 9126. COTS are commercially available software components. The authors identified quality characteristics, sub-characteristics, and measures for COTS. The study presented nine quality characteristics, which are maintainability, testability, functionality, efficiency, reliability, usability, security, portability, and reusability.

Ismail et al. [43] proposed a set of quality characteristics and sub-characteristics for reusability of software components. The study identified three main quality characteristics, which are understandability, adaptability, and portability. Understandability has two sub-characteristics, documentation level and observability. Adaptability has one sub-characteristic, customizability. Portability also has one sub-characteristic, external dependency. To evaluate the proposed software components, a survey was conducted at

Universiti Malaysia Terengganu (UMT). The respondents included 18 software component experts, 8 officers for application development, and 10 computer science lecturers.

In [44], the authors investigated the quality characteristics of software component selection based on ISO 2010:2011. The study contains five quality characteristics: effectiveness, efficiency, satisfaction, safety, and usability.

Ali and Yap [45] proposed a software component quality model specifically for the design level. The authors found eleven quality characteristics, which are compositionality, reusability, coupling, configurability, encapsulation, complexity, usability, slim, testability, cohesion, and interface documentation.

Simao and Belchior [46] presented a set of quality characteristics and sub-characteristics for software components based on the ISO 9126 standard. The study contains six main quality characteristics, namely, functionality, reliability, usability, efficiency, maintainability, and portability.

Alvaro et al. [47] designed a software component quality model based on ISO 9126. The study contains seven quality characteristics, which are functionality, reliability, usability, efficiency, maintainability, portability, and marketability, and each of them has sub-characteristics.

Another study [48] presented a quality model aimed at evaluating COTS components, which was adapted from ISO 9126. The proposed model comprises quality characteristics and sub-characteristics. The model contains four quality characteristics, which are functionality, reusability, maintainability, and conformance.

In [49], the authors introduced a quality model for component-based software which is derived from ISO 9126. The proposed model added several sub-characteristics to the ISO 9126 which are reusability, complexity, scalability, traceability, and flexibility. The study comprises six main characteristics, namely, functionality, reliability, usability, efficiency, maintainability, and portability.

Choi et al. [50] developed a quality model for software component evaluation based on ISO 9126. The proposed model contains eight main characteristics: functionality, reusability, portability, maintainability, usability, reliability, efficiency, and modularity.

Bertoa et al. [51] identified quality characteristics for usability of software components, which were derived from the ISO 9126 standard. These characteristics were understandability, learnability, and operability. The study demonstrated that the observable elements of software components are documentation and some of the component's functional elements.

Mahmood et al. [52] conducted a survey on quality assurance and assessments for component-based systems. The study revealed that quality assurance and assessments in component-based systems are different from conventional software systems due to their hidden internal structures. The study also emphasized that delivering quality systems depends on the quality of components used. The authors highlighted several characteristics that contribute to the overall quality of a system, namely, performance, reliability, maintainability, and testability. The study also revealed a lack of automated tools in the literature for measuring quality characteristics.

Thapar et al. [53] developed a reusability-based quality framework for software components. The aim of this study was to standardize the quality evaluation process, which helps stakeholders select software components that meet their requirements and application areas, where the stakeholders are users, managers, and developers. The proposed framework contains the main quality characteristics, with each having sub-characteristics. There were six main characteristics, namely, functionality, reliability, usability, efficiency, maintainability, and portability.

Tahir et al. [54] introduced a quality assessment model for software components. Their proposed model contains eight primary characteristics, namely, design quality, plugability, reusability, functionality, reliability, maintainability, documentation quality, and efficiency. There were also thirty-three sub-characteristics in the model.

The existing software component quality models, such as [39–54], are not applicable to ML software because ML software is all about data [9]. Similarly, the study [55] highlighted

that the quality assurance of ML software differs from conventional software because ML software deals with data and also has characteristics quite different from software systems. Furthermore, the advances in AI technologies and data-driven ML techniques and in how to build high-quality AI software and applications has become a very hot subject [56]. For this reason, the special features of AI software bring new challenges and issues for validating software or system quality. Moreover, the specific quality characteristics of ML software are missing from the existing software component quality models. The existing studies on AI and ML software components are discussed in the next section.

2.5. AI and ML Software Components

Several studies have componentized AI neural networks and ML models by exploiting reusability of component-based approaches. This has led to the rapid development of ML models, without requiring software engineers to have great knowledge of ML models and algorithms and also reducing maintenance complexity.

Ahmad et al. [20] componentized three neural network models, namely, the Multi-Layer Perceptron, the Learning Vector Quantization, and the Adaptive Resonance Theory family of networks. The authors adapted a component-based software engineering approach to neural network models to eliminate the tightly coupled solutions that end up in monolithic and non-reusable programming efforts. Furthermore, CompoNet (Componentized-neural-Networks) facilitates the development of neural network-based software systems by designing and implementing neural networks as software components. These components can be reused effortlessly across different applications, and thus, the shipping of trained models from the simulation to production environments is possible with minimal programming effort.

In another area, building energy prediction plays an important role in steering design towards the required sustainability regulations. Several issues have been encountered in building energy prediction such as the time-consuming nature of detailed Building Energy Modelling (BEM), meta-models within the design process, and the lack of energy modelling experts in the early design stages. To overcome these issues, Singaravel et al. [57] proposed component-based ML Modelling (MLM) for design-stage energy prediction. The component-based ML model was evaluated using weather data from Amsterdam, Brussels, and Paris. The results indicated that developing MLMs with diverse datasets and appropriate input parameters could result in models that generalize well under different design situations, provided that the new data match the distribution of the training data. Further research on the generalization of component-based MLM for building design with additional data and input parameters was performed to identify the full potential of such an approach.

Singaravel et al. [58] proposed a Component-Based ML (CBML) approach, specifically a deep learning architecture for Building Performance Simulations (BPS). BPS is a technology which helps to develop buildings that adhere to energy-efficiency requirements in high-performance building design and operation. The proposed CBML approach solved several existing issues in BPS, such as designers lacking fundamental knowledge of physical phenomena in BPS, making it challenging for them to understand simulation results and take appropriate design decisions. In addition, CBML helps to mitigate the time-consuming nature of simulating multiple design options using BPS. Moreover, the CBML approach plays an important role in the reusability of ML models for design situations that are different from the training situation. The study concluded that the proposed CBML approach supports the design process of buildings, especially in early phases. Components are plugged together as needed in the process, and prediction is performed immediately without additional modelling or significant computation time.

3. Related Works

This section lists down previous studies on AI-software and system quality. Their scope, objectives, and findings are compared with this SMS.

Gezici and Tarhan [5] performed a systematic literature review (SLR) of 29 papers on AI software quality. They discovered quality characteristics and their assurance, challenges, solutions, and existing quality models for AI-based software quality from 1988 to 2020. This present study aims to explore the quality of AI software, systems, and components together. The SMS is much broader compared to the SLR of [5], which only covered AI software quality, while this study aims to cover different quality levels of AI software, systems, and components.

Lwakatare et al. [16] reviewed 72 papers on the development and maintenance of machine learning-based software systems, as well as difficulties and solutions. This research does not focus on product quality, and the software examined is ML-based.

Braiek and Khomh [20] reviewed the current testing practices for ML-based systems. They found 37 linked main studies between 2007 and 2019 in the Engineering Village, Google Scholar, and Microsoft Academic Search databases. To ensure that ML programs were reliable, the authors looked for and found obstacles when testing such software systems, offered solutions advocated by research in the literature, and made recommendations based on their findings. This research intends to assist practitioners improve the quality of machine learning algorithms by teaching them how to test them. Unlike the earlier study, this paper focuses on different levels of AI software, systems, and components.

Riccio et al. [59] conducted a systematic mapping study in the context of functional testing for machine learning systems (MLS), analyzing 70 papers. While this paper is interested in AI-based software, systems, and components, ref. [59] were more interested in MLS testing, difficulties, and solutions. While their research questions are focused on testing, the ones in this study are focused on AI software, systems, and component quality. As a result, the contributions and consequences of [59] differ significantly from the current study.

Next, Zhang et al. [60] conducted a comprehensive survey study to examine ML software testing trends, difficulties, and solutions. They gathered 138 articles from 2007 to 2019 that focused on testing properties such as accuracy, robustness, efficiency, privacy, and fairness. They established and demonstrated existing machine learning testing methods and workflows, and evaluated quality attributes such as accuracy, robustness, and others as specific to testing of ML software systems, similar to the present study. The context and domain of [60], on the other hand, differ from this study, which aims to learn more about product quality than just software testing.

Consequently, the shortcomings mentioned below have motivated the researchers to conduct a broader and more functional systematic mapping study:

- From an analysis of the related literature, there is still a large gap in terms of addressing the quality of AI software, systems, and components.
- None of the former studies have explored this gap in the literature.
- Although previous research has increased our understanding of specific cases and contexts of AI-based software, it has not provided a clearer picture of AI software, systems, and component quality or of related challenges in experimental, academic, and industrial contexts.

This paper analyses previous studies that propose quality models, together with their respective evaluation methods, quality characteristics and limitations. In my knowledge, none of them provides similar comprehensive SMS in the context of software quality for AI software, systems, and components. Finally, Table 1 highlights how this study differs from the previous ones.

Table 1. Comparison of Related Works.

References.	Quality Perspective	Research Method
[5]	AI software	SLR
[16]	ML-based software systems	SLR
[20]	ML systems	Survey
[59]	ML systems	SMS
[60]	ML systems	Survey
Our study	AI software, systems, and components	SMS

4. Review Method

This study employs a methodology proposed by [61–63]. It was decided to combine the automated search and the left snowballing search in order to differentiate the maximum number of primary studies since this main study was sufficient. During the study process, a protocol was drawn up, consisting of five main tasks: (1) the Research Questions (RQs) were defined; (2) the search strategy was outlined; (3) the study criteria and the process were defined; (4) the quality evaluations of primary studies were assessed; and (5) the data extraction and classification scheme were established.

4.1. Research Questions

In order to assure the relevance of the RQs, the following criteria have been specially structured [64]:

- Population: Software Quality, Software Engineering, and Machine Learning;
- Intervention: AI system, software, and components;
- Outcomes: classification of AI quality models into AI systems, software, and components. Table 2 lists the three RQs aimed at establishing the search strategy for data extraction.

Table 2. Research Questions (RQs).

RQ	Research Questions	Objective and Motivation
RQ1	What are the quality models for AI Software?	To investigate the existing quality models for AI software.
RQ2	What are the quality models of AI Systems?	To investigate the existing quality models for AI systems.
RQ3	What are the quality models for AI Software Components?	To investigate quality models for AI software components.

4.2. Search Strategy

For the construction of the search string [65], the following steps were taken: the main phrases were expanded to include synonyms and alternative spelling structures, the keywords were investigated in the relevant studies, the Boolean OR was used to construct synonyms and other spelling structures, and the Boolean AND was used to merge the key phrases.

There are two major categories of digital libraries [66]: Indexed engines and publishers' sites. Indexed engines include ISI Web of Science (WoS), Scopus, Google Scholar, CiteSeer, and Ei Compendex and Inspec (EI). Publishers' sites include IEEE Xplore, ACM, Science Direct, SpringerLink, and Wiley Inter-Science (WIS). Table 3 presents the four digital library engines of the study.

Table 3. Electronic databases used in automated search.

Electronic Databases	Uniform Resource Locator (URL)
IEEEExplore	https://ieeexplore.ieee.org
SCOPUS	https://www.scopus.com
Google Scholar	https://scholar.google.com
ACM Digital Library	http://dl.acm.org.com

4.3. Study Selection

For studies recovered from digital library engines, the main study responsive to the RQs in this study was selected in several processes, including criteria for selection and exclusion. These criteria were used in conjunction with the matching justification used in Table 4.

Table 4. Inclusion and Exclusion Criteria.

Category	Inclusion	Exclusion
Publication type	Journal articles and conference proceedings from 2000–2021	Theses and book chapters (only in the automated search).
Publication topic	Topic must be related to Computer science and Information Technology	Topics are not related to Computer Science and Information Technology.
Inter	Primary Studies.	Secondary studies.
Intervention	Studies that propose solutions for AI and ML systems, software and components quality.	Studies that do not propose solutions for AI and ML systems, software, and components quality.

Figure 1 shows the multi-stage approach for selecting the studies. This selection procedure aims to separate the relevant research which is consistent with the SMS objective from that which is not. Four databases were selected, namely, Scopus, IEEE, ACM, and Google Scholar. After this, primary research studies were extracted from the selected digital libraries using the constructed search string.

Furthermore, all searches were subject to inclusion and exclusion criteria as in Table 3. For instance, duplicate studies were removed from the results. In addition, the titles and abstracts of potential studies were screened carefully according to the following method:

- Publication type: the publishing type exclusion was not identified because it was carried out in every digital library on an automated search.
- Publication topic: a review of the publishing topic was performed to verify that it fulfilled the inclusion and exclusion criteria justification. A SCImago Journal Rank (SJR) [67] was used to identify each area and category of journals. Inspection or further screening of the journal website took place if a journal title in SCImago was not available.
- Study type: there have been no selections from any sort of secondary study (for example SLRs, SMSs, or review articles).
- Intervention: studies determined to have been excluded that did not fit in with the aims of the SMS. During this process, a majority of the studies excluded were identified.

Studies that complied with the inclusion criteria were considered relevant and were accepted for full-text screening. At the end of this stage, a full-text screening of 44 studies was carried out.

Finally, 10 studies were selected following the guidance criteria, using the forward and backward snowball techniques [68]. At first, the documents were accumulated via snowballing back and forth (i.e., through the citations and references). For Step 2, 10 studies were chosen after comparison with the titles of the 10 studies (from Step 1), and no studies were excluded (from reading the abstracts and conclusions). No studies were acquired just by snowballing and reading the whole text of the publications using the generic, specific, and qualitative criteria for evaluation screening.

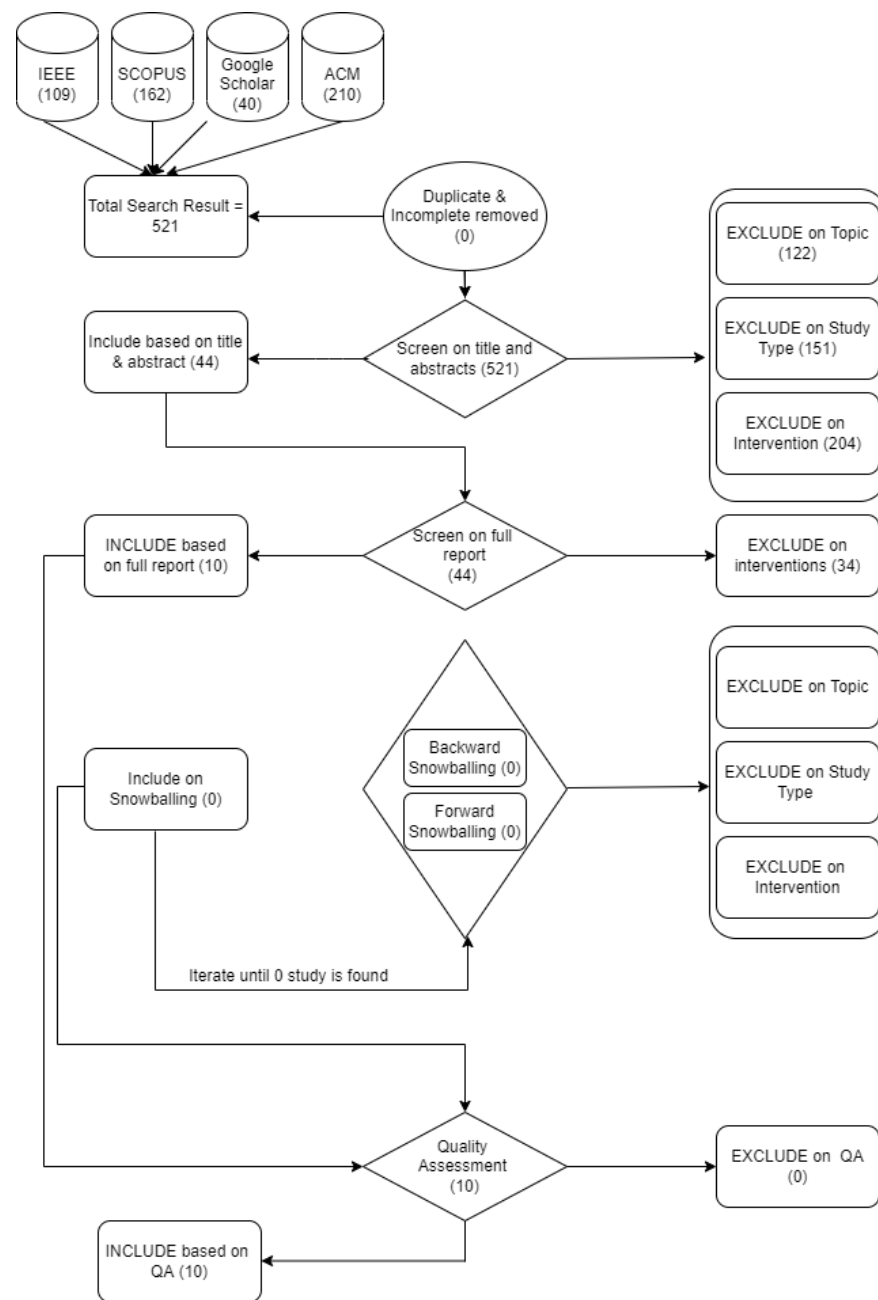


Figure 1. Study Selection Stages.

4.4. Data Extraction and Classification Scheme

Ten Primary Studies were chosen, which were called Selected Primary Studies (SPSs). Information was gathered from the SPSs based on the classification schema. The purpose of this stage of the investigation was to map and categorize the papers so that the RQs could be addressed. The SPSs were divided into three RQ groups as follows.

4.4.1. Quality Models for AI Software's (Related to RQ1)

This section discusses the existing quality models for AI software such as those reported in [3,69–72].

4.4.2. Quality Models of AI Systems (Related to RQ2)

This section discusses existing quality models for AI systems, such as [10,21,23,73,74]. In summary, each of the AI system quality models focuses on the quality characteristics of specific AI systems.

4.4.3. AI Software Component Quality Models (Related to RQ3)

This section investigates quality models that play crucial roles in AI Software components. The already-conducted SMS reveals a lack of quality models for AI software components.

5. Analysis and Results

This section presents the results derived from the review. First, an overview of the demographic data from Excel is presented. Then, the answers to the RQs are reported.

5.1. Overview of Selected Studies

The Primary Selection (PS) process resulted in the identification of 10 relevant studies. Figure 2 shows the distribution of selected studies by year. In 2019, the most papers were published (four), followed by 2020, when three were published.

Figure 3 shows the number of selected studies of AI quality models by their level. AI Quality models at the system level and software level were the most frequently published (five papers each). There were no papers on quality models at the AI component level. Lastly, Figure 4 shows the frequent quality characteristics in selected studies of AI quality models for software and systems. Privacy and Security are the most widely discussed quality characteristics in the selected studies.

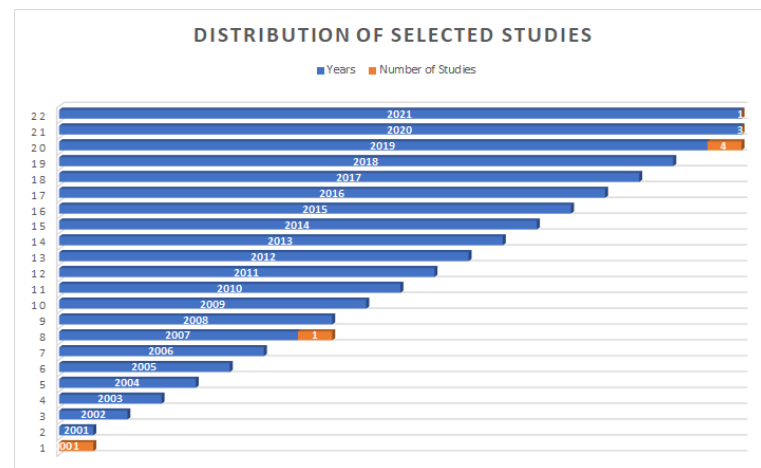


Figure 2. Distribution of Selected Studies (Year).

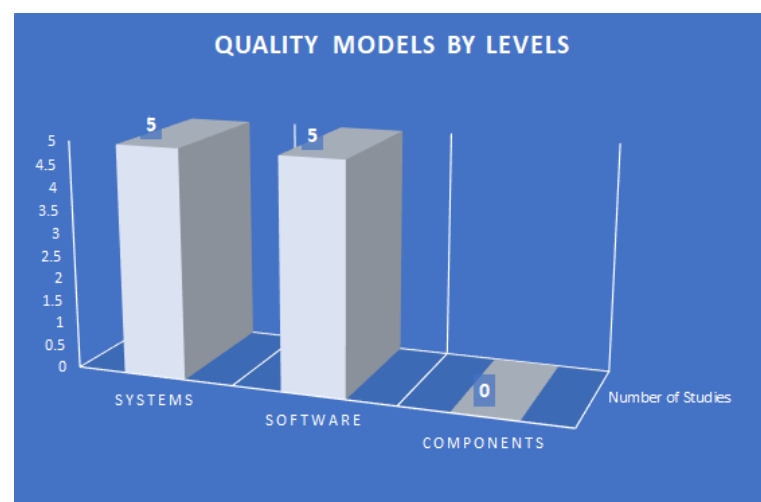


Figure 3. Quality Models by Levels.

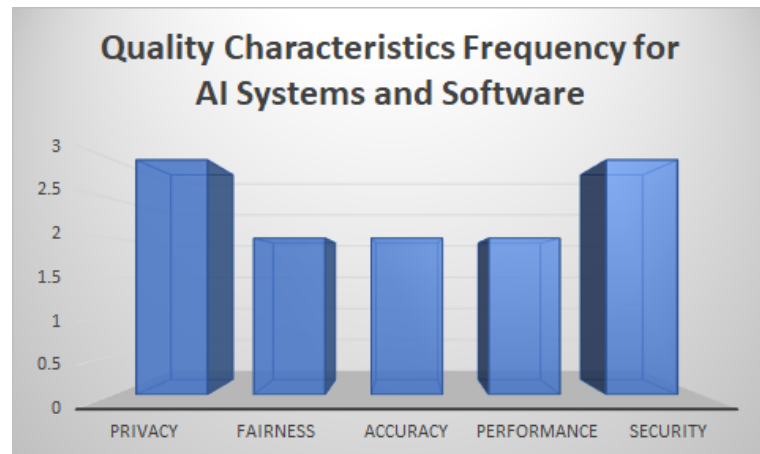


Figure 4. Quality Characteristics Frequency for AI Systems and Software.

5.2. Data Synthesis

After quality assessment, a total of 10 studies were selected in the SMS. In this section, we provide the answers to our research questions.

5.2.1. RQ1: What Are the Quality Models for AI Software?

The following quality models for AI software were identified in the study. Nakamichi et al. [3] proposed a set of quality characteristics for ML software by extending ISO 25010 software products, Horkoff [69] discussed the quality characteristics of non-functional requirements (NFR) of ML algorithms, Lenarduzzi et al. [70] proposed quality characteristics of ML software, ref. [71] proposed quality characteristics for Artificial neural-network-based applications, and, lastly, Nakajima [72] conducted a study on quality evaluation of Deep Neural Networks (DNN) software.

The above listed quality models for AI software consider the quality characteristics for the AI software level. Table 5 and Table 6 shows quality models and characteristics for AI software, respectively.

Based on the SMS carried out in this study, few high-quality models for AI software have been proposed. However, only [3] carried out empirical evaluation with limited experts, and the rest of the studies were left out of the empirical evaluation. Furthermore, when compared to traditional systems, research on quality models or standards in AI software receives little attention. The AI software quality models are insufficient in terms of the quality characteristics that they propose. They only proposed a few quality characteristics, leaving out many others. In the literature, there is still a need for comprehensive AI software quality models or standards with empirical evaluations to evaluate the acceptability and applicability of the proposed quality characteristics.

Table 5. Existing Quality Models for AI Software.

Study	Quality Level	Evaluation	Weakness
[3]	AI software	Empirical evaluation	The study only consider the quality of ML software at the requirement engineering phase. The empirical evaluation of the proposed quality characteristics was performed by few experts.
[69]	AI software	No evaluation	This study highlighted the need for empirical methods such as surveys or interviews to derive knowledge from ML experts or researchers by asking what are the important qualities for ML software.
[70]	AI software	No evaluation	The study is more focused on quality issues for AI software. It only presents few quality characteristics for AI software such as compatibility, incorruptibility, and completeness.

Table 5. *Cont.*

Study	Quality Level	Evaluation	Weakness
[71]	Artificial neural network (ANN) applications	No evaluation	They presented quality characteristics for ANN applications. Based on this study, they are not comprehensive and require an update as the study was published two decades ago. They have not conducted any validation of their quality characteristics.
[72]	AI software	No evaluation	This research proposed quality assurance framework using third-part certifications and assessed the how activities are conducted in the development process but did not refer to assuring quality directly.

Table 6. Quality Characteristics for AI Software.

Study	Quality Level	Quality Characteristics
[3]	AI Software	Sufficiency of Accuracy of Trained Model, Appropriateness of Model Training Process, Appropriateness of Quality Maintenance Means for Training Data, Understandability, Easiness of Resource Update, Easiness of Software Update, Appropriateness of Security and Privacy Assurance Means, and Appropriateness of Resource Utilization.
[69]	AI Software	Accuracy, performance, fairness, transparency, security, privacy, testability, and reliability.
[70]	AI Software	Compatibility, incorruptibility, and completeness.
[72]	AI Software	Prediction performance quality includes learning model selection, generalization, coverage of trained model, and invariants. Training mechanism quality includes traceability, inspection of training process, and static analysis of code. Lifecycle support quality includes outliers, sample bias, learning model reuse, and monitoring.

5.2.2. RQ2: What Are the Quality Models of AI Systems?

The SMS identified some existing quality models for AI systems. For instance, Pons and Ozkaya [10] studied quality characteristics for AI-enabled systems; Kuwajima and Ishikawa [23] provided holistic insights on the quality of AI systems by incorporating the characteristics of ML and AI ethics into traditional software quality concepts (ISO 25000); and Vinayagasundaram and Srivatsa [73] aimed to measure the quality of Software Architecture in AI Systems.

These quality models only focused on quality characteristics at the AI system level. AI systems can be software-based (e.g., voice assistants, image analysis software, search engines, speech and face recognition systems), or AI systems can be embedded in hardware devices (e.g., advanced robots, autonomous cars, drones, or Internet of Things applications) [11]. Table 7 and Table 8 shows quality models and characteristics for AI systems, respectively.

A few quality models have been proposed for AI systems so far, but only [74] conducted an empirical evaluation, and the rest were left out of this study. Moreover, research on quality models or standards in AI systems has attracted less attention compared to traditional systems. The AI system quality models are not comprehensive in terms of the quality characteristics they propose. They only proposed a few quality characteristics, and many others were left out. The need for comprehensive AI system quality models or standards is still present in the literature.

Table 7. Existing Quality Models for AI Systems.

Study	Quality Level	Evaluation	Weakness
[10]	AI systems	No evaluation	This study proposed a few quality characteristics for AI systems in the public sector. It did not conduct evaluation of the acceptability of the proposed quality characteristics. This study is a position paper and lacks comprehensiveness in terms of the quality characteristics it covered.
[21]	ML systems	Case study	The study proposed quality characteristics for ML systems. The proposed quality model was based on four views: Data view, System view, Infrastructure view, and Environment view. Three case studies have been conducted for the evaluation for the proposed quality characteristics.
[23]	AI systems	No evaluation	Only provided holistic insights incorporating the ML nature and AI ethics to SQuaRe from traditional software engineering. Only focused on external quality characteristics for SQuaRe, and evaluation was not performed either empirically or practically.
[73]	AI systems	No evaluation	Only proposed limited quality characteristics for AI system architecture. Empirical evaluation for the proposed quality characteristics from experts is also missing.
[74]	AI systems	Empirical evaluation	This study proposed evAIa (evaluate AI approaches) for AI system quality assurance (QA). EvAIa leads to transparency about the current state of QA. EvAIa received fast acceptance, for example, in a centralized AI competence centre which introduced evAIa as a standard for its project QA. The study conducted evaluation of the acceptability of the proposed quality characteristics.

Table 8. Quality Characteristics for AI Systems.

Study	Quality Level	Quality Characteristics
[10]	AI Systems	Security, privacy, data-centricity, sustainability, and explain-ability.
[23]	AI Systems	Risk on human autonomy, collaboratability, fairness, privacy, accountability, maintainability, modifiability, operability, modularity, integrity, and uncertainty
[73]	AI Systems	Testability, usability, readability, and self-descriptiveness.
[74]	AI Systems	Adequacy, Bias, Completeness, Compliance, Execution environment, Worst cases (compute power, memory size), Third party reliability, Completeness, Process-chain, Regulations/compliance, Training and testing chain, Model transparency, Model adequateness, Model robustness, Model completeness, Configuration, Execution environment, Monitoring, Validation, Training and testing chain, Third party, and “Model fitting”

5.2.3. RQ3: What Are the Quality Models for AI Software Components?

Currently, there are several software development paradigms [75]. In addition, the specificity of each software paradigm requires the development of its own software quality model. For instance, quality models for traditional software, object-oriented software, service-oriented software, and component-based paradigms have been discussed so far. Recently, few quality models for the AI and ML software paradigm have been proposed. The AI software component paradigm, specifically, the ML software component paradigm, differs from the AI and ML software paradigm because such components are black boxes (i.e., their internal structure is hidden). The quality of such components can only be

measured through external interfaces. Quality assessments of individual ML software components will have an impact on the overall system and software.

The SMS found no quality models for AI software components. Gharib et al. [76] supported the need of AI software component quality models, specifically ML software components. To the best of the authors' knowledge, there are no available quality assurance models for AI software components, specifically ML software components. The main purpose is that the quality evaluation of such software components will impact on overall software quality and functionality [77]. Moreover, the failure of such software components will lead to system failures [78].

Future research can focus on the development of an AI software component, particularly, ML software component quality models to bridge the gap in existing literature. Such quality models will assess the quality of ML software components.

6. Limitations and Future Works

This paper has aimed to investigate and assess existing quality models for AI software, systems, and components. Several quality models for AI software were found [3,69–72], as well as for systems [10,21,23,73,74]. These quality models only focus on quality characteristics at the software and system levels. Moreover, each of these studies proposed a few quality characteristics and still lack a comprehensive quality model. The existing quality models for AI software and systems mostly were not evaluated empirically and practically.

So far, no work on AI software component quality models has been proposed. Gharib et al. [76] highlighted the need of a quality model for AI software components, specifically ML software components. The main goal is that the quality evaluation of such a software component will have an impact on the overall quality and functionality of the software [77].

On the other hand, most of the papers in the SMS have not followed a consistent method of proposing quality models for AI systems and software. The traditional quality models such as McCall, Boehm, and ISO 25010 were built through hierarchical structures and based on the 'ilities' procedure. However, quality models for AI systems and software do not follow a hierarchical structure or the 'ilities' procedure during the development of quality models.

Future directions in research should be as follows. Firstly, existing quality models for AI systems and software should be improved with a comprehensive quality characteristic by following the procedure of 'ilities' based on traditional quality models. Then, models should be evaluated empirically to prove their effectiveness and applicability. Secondly, this SMS shows a lack of quality models for AI software components, specifically ML software components. Quality models for ML software components increase the quality of overall ML software and components. Thus, the need for comprehensive AI software component quality models is still present in the literature.

Lastly, for traditional software and systems, there are various quality metrics used to measure quality characteristics. Often, these metrics cannot be used directly to measure AI software and systems, and it is challenging to employ or transfer them to measure the quality of AI-based components [78]. Due to the unique characteristics of AI software systems, there is a need for new quality models and metrics specific to AI-based software [3,5].

7. Threats to Validity

According to Petersen et al., SMS and SLR studies, like empirical studies, require validity considerations [79].

7.1. Internal Validity

Section 4 describes the systematic approach that was used for article selection. Search engines, search terms, and inclusion and exclusion criteria are carefully defined and reported to ensure that this review is repeatable. Limitations in search terms and search engines, as well as bias in applying exclusion and inclusion criteria, are problematic issues in the selection process.

7.2. External Validity

The external validity of this SMS study is concerned with how far the findings can be generalized. As described in Section 4, the article selection approach used defined search terms that resulted in all primary studies being written in English; studies written in other languages were excluded.

7.3. Construct Validity

In this study, the threats to this type of validity were the suitability of RQs and the categorization scheme used for data extraction. To limit construct threats in this study, research questions are designed to cover our goal as well as various aspects of secondary studies in the area of software quality for AI. A classification scheme is used to answer questions.

8. Conclusions

AI has become a crucial component in modern system and software development. Moreover, current software engineering has fallen under the influence of AI. Their quality assessment differs from traditional software, systems, and components. Some existing quality models for the AI context have been proposed. Quality models assess different levels: for example, they may assess system-level quality characteristics (this includes hardware and software properties), software-level characteristics (only limited to software characteristics), or component-level characteristics.

As far as the researchers are aware, no systematic mapping for AI systems, software, and components have been investigated so far. Because of this, the need to investigate, categorize, and analyze the state-of-the-art quality models for AI systems, software, and components have become urgent. Therefore, in this paper, an SMS to discover the existing quality models for such systems, software, and components was carried out. The main contribution is to discover and analyze quality models. Moreover, this study also synthesized quality perspectives, evaluation methods, limitations, and future directions of proposed quality models for AI systems, software, and components. Finally, the study only found quality models on AI systems and the software level. Based on this SMS, no quality models on the level of AI software components were found. Therefore, there is a need to develop such quality models in the future, specifically ML software component quality models.

Furthermore, a comprehensive quality model for AI systems and software is still absent from the literature. The SMS only found quality models containing quality characteristics for AI systems and software, and this evaluation is subjective and qualitative. Research on quantitative evaluation methods, models, or metrics are still absent for AI software, systems, and components. Without metrics, assessments and evaluations are considered to be subjective and unamenable to quantitative comparison [80]. Metrics enable software quality to be quantifiable and countable.

Author Contributions: Conceptualization, M.A.A.; methodology, M.A.A.; validation, N.K.Y. and M.A.A.; formal analysis, M.A.A.; investigation, M.A.A. and N.K.Y.; resources, M.A.A.; data curation, M.A.A. and N.K.Y.; writing—original draft preparation, M.A.A.; writing—review and editing, N.K.Y., A.A.A.G., H.Z., N.I.A. and A.A.N.; visualization, M.A.A., N.K.Y., N.I.A. and A.A.N.; supervision, N.K.Y., A.A.A.G., H.Z. and N.I.A.; project administration, N.K.Y.; funding acquisition, N.K.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Universiti Putra Malaysia (UPM).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: For researchers who meet the criteria for accessing the data, data are available from the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

- Chollet, F. *Deep Learning mit Python und Keras: Das Praxis Handbuch vom Entwickler der Keras-Bibliothek*; MITP-Verlags GmbH & Co., KG.: Shelter Island, NY, USA, 2018.
- Harman, M. The role of artificial intelligence in software engineering. In Proceedings of the 2012 First International Workshop on Realizing AI Synergies in Software Engineering (RAISE), Zurich, Switzerland, 5 June 2012; pp. 1–6.
- Nakamichi, K.; Ohashi, K.; Namba, I.; Yamamoto, R.; Aoyama, M.; Joeckel, L.; Heidrich, J. Requirements-driven method to determine quality characteristics and measurements for machine learning software and its evaluation. In Proceedings of the 2020 IEEE 28th International Requirements Engineering Conference (RE), Zurich, Switzerland, 31 August–4 September 2020.
- Shafiq, S.; Mashkoo, A.; Mayr-Dorn, C.; Egyed, A. Machine Learning for Software Engineering: A Systematic Mapping. *arXiv* **2020**, arXiv:2005.13299.
- Gezici, B.; Tarhan, A.K. Systematic literature review on software quality for AI-based software. *Empir. Softw. Eng.* **2022**, *27*, 1–65. [\[CrossRef\]](#)
- Masuda, S.; Ono, K.; Yasue, T.; Hosokawa, N. A survey of software quality for machine learning applications. In Proceedings of the 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Vasteras, Sweden, 9–13 April 2018; pp. 279–284.
- Perkusich, M.; e Silva, L.C.; Costa, A.; Ramos, F.; Saraiva, R.; Freire, A.; Perkusich, A. Intelligent software engineering in the context of agile software development: A systematic literature review. *Inf. Softw. Technol.* **2020**, *119*, 106241. [\[CrossRef\]](#)
- Martínez-Fernández, S.; Bogner, J.; Franch, X.; Oriol, M.; Siebert, J.; Trendowicz, A.; Wagner, S. Software Engineering for AI-Based Systems: A Survey. *arXiv* **2021**, arXiv:2105.01984.
- Sculley, D.; Holt, G.; Golovin, D.; Davydov, E.; Phillips, T.; Ebner, D.; Dennison, D. Hidden technical debt in machine learning systems. In *Advances in Neural Information Processing Systems*; Curran Associates, Inc.: New York, NY, USA, 2015.
- Pons, L.; Ozkaya, I. Priority Quality Attributes for Engineering AI-enabled Systems. *arXiv* **2019**, arXiv:1911.02912.
- European Commission. 2021. Available online: <https://digital-strategy.ec.europa.eu/en/library/definition-artificial-intelligence-main-capabilities-and-scientific-disciplines> (accessed on 11 July 2021).
- Rushby, J. *Quality Measures and Assurance for AI (Artificial Intelligence) Software*; Technical Report; NASA Langley Technical Report Server: Hampton, VA, USA, 1988.
- Nemecek, S.; Bemley, J. A Model for Estimating the Cost of AI Software Development: What to do if there are no Lines of Code? In Proceedings of the 1993 Proceedings IEEE International Conference on Developing and Managing Intelligent System Projects, Washington, DC, USA, 29–31 March 1993; pp. 2–9.
- Druffel, L.; Little, R. Software engineering for AI based software products. *Data Knowl. Eng.* **1990**, *5*, 93–103. [\[CrossRef\]](#)
- Lwakatare, L.E.; Raj, A.; Crnkovic, I.; Bosch, J.; Olsson, H.H. Large-Scale Machine Learning Systems in Real-World Industrial Settings A Review of Challenges and Solutions. *Inf. Softw. Technol.* **2020**, *127*, 106368. [\[CrossRef\]](#)
- Foidl, H.; Felderer, M. Risk-based data validation in machine learning-based software systems. In Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation, ACM: New York, NY, USA, 2019; pp. 13–18.
- Sato, N.; Kuruma, H.; Ogawa, H. Unsupposable Test-data Generation for Machine-learned Software. *arXiv* **2020**, arXiv:2005.10442.
- Braiek, H.B.; Khomh, F. On testing machine learning programs. *J. Syst. Softw.* **2020**, *164*, 110542. [\[CrossRef\]](#)
- Chen, Z.; Cao, Y.; Liu, Y.; Wang, H.; Xie, T.; Liu, X. Understanding Challenges in Deploying Deep Learning Based Software: An Empirical Study. *arXiv* **2020**, arXiv:2005.00760.
- Ahmad, U.; Gavrilov, A.; Lee, S.; Lee, Y.K. CompoNet: Programmatically embedding neural networks into AI applications as software components. In Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007), Patras, Greece, 29–31 October 2007; Volume 1, pp. 194–201.
- Siebert, J.; Joeckel, L.; Heidrich, J.; Trendowicz, A.; Nakamichi, K.; Ohashi, K.; Aoyama, M. Construction of a quality model for machine learning systems. *Softw. Qual. J.* **2021**, *30*, 307–335. [\[CrossRef\]](#)
- Khomh, F.; Adams, B.; Cheng, J.; Fokaefs, M.; Antoniol, G. Software engineering for machine-learning applications: The road ahead. *IEEE Softw.* **2018**, *35*, 81–84. [\[CrossRef\]](#)
- Kuwajima, H.; Ishikawa, F. Adapting SQuARE for quality assessment of artificial intelligence systems. In Proceedings of the 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), Berlin, Germany, 27–30 October 2019; pp. 13–18.
- McCall, J.A.; Richards, P.K.; Walters, G.F. *Factors in Software Quality, Volumes I, II, and III*; US Rome Air Development Center Reports; US Department of Commerce: Washington, DC, USA, 1977.
- Boehm, B.W.; Brown, J.R.; Kaspar, H.; Lipow, M.; MacLeod, G. Merritt.: *Characteristics of Software Quality*; Elsevier: Amsterdam, The Netherlands, 1978.
- Grady, R.B. *Practical Software Metrics for Project Management and Process Improvement*; Prentice-Hall, Inc.: Englewood Cliffs, NJ, USA, 1992.
- ISO/IEC 25010:2011. 2011. Available online: <https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en> (accessed on 8 September 2019).
- ISO/IEC 9126-1; Software Engineering-Product Quality-Part 1: Quality Model. International Organization for Standardization: Geneva, Switzerland, 2001.

29. ISO/IEC TR 9126-2; Software Engineering-Product Quality-Part 2: External Metrics. International Organization for Standardization: Geneva, Switzerland, 2003.
30. ISO/IEC TR 9126-3; Software Engineering-Product Quality-Part 3: Internal Metrics. International Organization for Standardization: Geneva, Switzerland, 2003.
31. ISO/IEC TR 9126-4; Software Engineering-Product Quality-Part 4: Quality in Use Metrics. International Organization for Standardization: Geneva, Switzerland, 2004.
32. The ISO/IEC 25000 Series of Standards. 2014. Available online: <https://iso25000.com/index.php/en/iso-25000-standards?limit=4&limitstart=0> (accessed on 20 August 2020).
33. Dromey, R.G. Cornering the chimera [software quality]. *IEEE Softw.* **1996**, *13*, 33–43. [\[CrossRef\]](#)
34. Jaffar, R.N.; Hussain, A.A.A.M.; Chiad, W. A new model for study of quality attributes to components based development approach. *Period. Eng. Nat. Sci.* **2019**, *7*, 1177–1185. [\[CrossRef\]](#)
35. Miguel, J.P.; Mauricio, D.; Rodríguez, G. A review of software quality models for the evaluation of software products. *arXiv* **2014**, arXiv:1412.2977.
36. Szyperski, C.; Gruntz, D.; Murer, S. *Component Software: Beyond Object-Oriented Programming*; Pearson Education: London, UK, 2002.
37. Basha, N.M.J.; Moiz, S.A. Component based software development: A state of art. In Proceedings of the IEEE-International Conference on Advances in Engineering, Science and Management (ICAESM-2012), Nagapattinam, India, 30–31 March 2012; pp. 599–604.
38. Chahal, K.K.; Singh, H. A metrics based approach to evaluate design of software components. In Proceedings of the 2008 IEEE International Conference on Global Software Engineering, Bangalore, India, 17–20 August 2008; pp. 269–272.
39. Bertoa, M.F.; Vallecillo, A. Quality attributes for COTS components. *I+ D Comput.* **2002**, *1*, 128–143.
40. Rawashdeh, A.; Matalkah, B. A new software quality model for evaluating COTS components. *J. Comput. Sci.* **2006**, *2*, 373–381. [\[CrossRef\]](#)
41. Upadhyay, N.; Despande, B.M.; Agrawal, V.P. Towards a software component quality model. In Proceedings of the International Conference on Computer Science and Information Technology, Penang, Malaysia, 22–24 February 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 398–412.
42. Land, R.; Alvaro, A.; Crnkovic, I. Towards efficient software component evaluation: An examination of component selection and certification. In Proceedings of the 2008 34th Euromicro Conference Software Engineering and Advanced Applications, Parma, Italy, 3–5 September 2008; pp. 274–281.
43. Ismail, S.; Kadir, W.M.W.; Noor, N.M.M.; Mohd, F. Determining Characteristics of the Software Components Reusability for Component Based Software Development. *J. Telecommun. Electron. Comput. Eng. (JTEC)* **2017**, *9*, 213–216.
44. Nazir, S.; Anwar, S.; Khan, S.A.; Shahzad, S.; Ali, M.; Amin, R.; Cosmas, J. Software component selection based on quality criteria using the analytic network process. In *Abstract and Applied Analysis*; Hindawi Publishing Corporation: London, UK, 2014.
45. Ali, M.A.; Yap, N.K. Software Component Quality Model. *Int. J. Eng. Adv. Technol.* **2019**, *9*, 1758–1762. [\[CrossRef\]](#)
46. Simão, R.P.; Belchior, A.D. Quality characteristics for software components: Hierarchy and quality guides. In *Component-Based Software Quality*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 184–206.
47. Alvaro, A.; De Almeida, E.S.; Meira, S.L. A software component quality model: A preliminary evaluation. In Proceedings of the 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO'06), Cavtat, Croatia, 29 August–1 September 2006; pp. 28–37.
48. Kim, S.D.; Park, J.H. C-QM: A Practical Quality Model for Evaluating COTS Components. In Proceedings of the 21st IASTED International Conference on applied informatics, Innsbruck, Austria, 10–13 February 2003; pp. 991–996.
49. Sharma, A.; Kumar, R.; Grover, P.S. Estimation of quality for software components: An empirical approach. *ACM SIGSOFT Softw. Eng. Notes* **2008**, *33*, 1–10. [\[CrossRef\]](#)
50. Choi, Y.; Lee, S.; Song, H.; Park, J.; Kim, S. Practical S/W component quality evaluation model. In Proceedings of the 2008 10th International Conference on Advanced Communication Technology, Gangwon, Korea, 17–20 February 2008; Volume 1, pp. 259–264.
51. Bertoa, M.F.; Troya, J.M.; Vallecillo, A. Measuring the usability of software components. *J. Syst. Softw.* **2006**, *79*, 427–439. [\[CrossRef\]](#)
52. Mahmood, S.; Lai, R.; Kim, Y.S.; Kim, J.H.; Park, S.C.; Oh, H.S. A survey of component based system quality assurance and assessment. *Inf. Softw. Technol.* **2005**, *47*, 693–707. [\[CrossRef\]](#)
53. Thapar, S.S.; Singh, P.; Rani, S. Reusability-based quality framework for software components. *ACM SIGSOFT Softw. Eng. Notes* **2014**, *39*, 1–5. [\[CrossRef\]](#)
54. Tahir, H.; Khanum, A.; Tahir, R. SCAM—Software Component Assessment Model. *Int. J. Comput. Sci. Inf. Secur.* **2011**, *9*, 229.
55. Ma, L.; Juefei-Xu, F.; Xue, M.; Hu, Q.; Chen, S.; Li, B.; See, S. Secure Deep Learning Engineering: A Software Quality Assurance Perspective. *arXiv* **2018**, arXiv:1810.04538.
56. Tao, C.; Gao, J.; Wang, T. Testing and Quality Validation for AI Software—Perspectives, Issues, and Practices. *IEEE Access* **2019**, *7*, 120164–120175. [\[CrossRef\]](#)
57. Singaravel, S.; Geyer, P.; Suykens, J. Component-based machine learning modelling approach for design stage building energy prediction: Weather conditions and size. In Proceedings of the 15th IBPSA Conference, San Francisco, CA, USA, 7–9 August 2017; pp. 2617–2626.

58. Singaravel, S.; Suykens, J.; Geyer, P. Deep-learning neural-network architectures and methods: Using component-based models in building-design energy prediction. *Adv. Eng. Inform.* **2018**, *38*, 81–90. [CrossRef]
59. Riccio, V.; Jahangirova, G.; Stocco, A.; Humbatova, N.; Weiss, M.; Tonella, P. Testing machine learning based systems: A systematic mapping. *Empir. Softw. Eng.* **2020**, *25*, 5193–5254. [CrossRef]
60. Zhang, J.M.; Harman, M.; Ma, L.; Liu, Y. Machine learning testing: Survey, landscapes and horizons. *IEEE Trans. Softw. Eng.* **2020**, *48*, 1–36. [CrossRef]
61. Keele, S. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*; Technical Report, ver. 2.3 EBSE Technical Report; EBSE: London, UK, 2007.
62. Petersen, K.; Feldt, R.; Mujtaba, S.; Mattsson, M. Systematic mapping studies in software engineering. In Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE), Bari, Italy, 26–27 June 2008; pp. 68–77. Available online: <http://dl.acm.org/citation.cfm?id=2227115.2227123> (accessed on 25 August 2020).
63. Ali, A.Q.; Sultan, A.B.M.; Abd Ghani, A.A.; Zulzalil, H. A Systematic Mapping Study on the Customization Solutions of Software as a Service Applications. *IEEE Access* **2019**, *7*, 88196–88217. [CrossRef]
64. Kitchenham, B.A.; Mendes, E.; Travassos, G.H. Cross versus within-company cost estimation studies: A systematic review. *IEEE Trans. Softw. Eng.* **2007**, *33*, 316–329. [CrossRef]
65. Kitchenham, B.; Mendes, E.; Travassos, G.H. A systematic review of cross-vs. within-company cost estimation studies. In Proceedings of the 10th International Conference on Evaluation and Assessment in Software Engineering (EASE), Swindon, UK, 10–11 April 2006; pp. 1–10.
66. Chen, L.; Babar, M.A.; Zhang, H. Towards an evidence-based understanding of electronic data sources. In Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering (EASE), Keele University, UK, 12–13 April 2010. Available online: <http://dl.acm.org/citation.cfm?id=2227057.2227074> (accessed on 15 October 2019).
67. SCImago. *SCImago Journal and Country Rank*; Scimago Lab: Madrid, Spain, 2019. Available online: <https://www.scimagojr.com/> (accessed on 25 August 2020).
68. Wohlin, C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, London, UK, 13–14 May 2014; pp. 1–10.
69. Horkoff, J. Non-functional requirements for machine learning: Challenges and new directions. In Proceedings of the 2019 IEEE 27th International Requirements Engineering Conference (RE), Jeju, Korea, 23–27 September 2019; pp. 386–391.
70. Lenarduzzi, V.; Lomio, F.; Moreschini, S.; Taibi, D.; Tamburri, D.A. Software Quality for AI: Where we are now? In Proceedings of the International Conference on Software Quality (SWQD 2020), Vienna, Austria, 14–17 January 2020.
71. Leung, W.K.; Simpson, R. Neural metrics—software metrics in artificial neural networks. In Proceedings of the KES'2000. Fourth International Conference on Knowledge-Based Intelligent Engineering Systems and Allied Technologies. Proceedings (Cat. No. 00TH8516), Brighton, UK, 30 August–1 September 2000; Volume 1, pp. 209–212.
72. Nakajima, S. Quality Evaluation Assurance Levels for Deep Neural Networks Software. In Proceedings of the 2019 International Conference on Technologies and Applications of Artificial Intelligence (TAAI), Kaohsiung, Taiwan, 21–23 November 2019; pp. 1–6.
73. Vinayagasundaram, B.; Srivatsa, S.K. Software quality in artificial intelligence system. *Inf. Technol. J.* **2007**, *6*, 835–842. [CrossRef]
74. Poth, A.; Meyer, B.; Schlicht, P.; Riel, A. Quality Assurance for Machine Learning—An approach to function and system safeguarding. In Proceedings of the 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS), Macau, China, 11–14 December 2020; pp. 22–29.
75. Marir, T.; Mokhati, F.; Bouchlaghem-Seridi, H.; Acid, Y.; Bouzid, M. QM4MAS: A quality model for multi-agent systems. *Int. J. Comput. Appl. Technol.* **2016**, *54*, 297–310. [CrossRef]
76. Gharib, M.; Lollini, P.; Botta, M.; Amparore, E.; Donatelli, S.; Bondavalli, A. On the Safety of Automotive Systems Incorporating Machine Learning based Components: A Position Paper. In Proceedings of the 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W), Luxembourg, 25–28 June 2018; pp. 271–274.
77. Kahtan, H.; Bakar, N.A.; Nordin, R. Reviewing the challenges of security features in component based software development models. In Proceedings of the 2012 IEEE Symposium on E-Learning, E-Management and E-Services, Kuala Lumpur, Malaysia, 21–24 October 2012; pp. 1–6.
78. Ishikawa, F. Concepts in Quality Assessment for Machine Learning—From Test Data to Arguments. In Proceedings of the International Conference on Conceptual Modeling, Xi'an, China, 22–25 October 2018; Springer: Cham, Switzerland, 2018; pp. 536–544.
79. Petersen, K.; Vakkalanka, S.; Kuzniarz, L. Guidelines for conducting systematic mapping studies in software engineering: An update. *Inf. Softw. Technol.* **2015**, *64*, 1–18. [CrossRef]
80. Bukhari, Z.; Yahaya, J.; Deraman, A. Metric-based Measurement and Selection for Software Product Quality Assessment: Qualitative Expert Interviews. *Int. J. Adv. Comput. Sci. Appl.* **2019**, *10*, 223–231. [CrossRef]