The Journal of Systems & Software 195 (2023) 111524

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

How have views on Software Quality differed over time? Research and practice viewpoints[☆]

Ifeanyi G. Ndukwe [a,*], Sherlock A. Licorish [a,1], Amjed Tahir [b], Stephen G. MacDonell [a,c]

[a] Department of Information Science, University of Otago, New Zealand
[b] School of Mathematical and Computational Sciences, Massey University, New Zealand
[c] Software Engineering Research Laboratory, Auckland University of Technology, New Zealand

ARTICLE INFO

Article history:
Received 15 January 2022
Received in revised form 28 July 2022
Accepted 28 September 2022
Available online 3 October 2022

Keywords:
Code snippet quality
Stack Overflow
Software quality

ABSTRACT

Context: Over the years, there has been debate about what constitutes software quality and how it should be measured. This controversy has caused uncertainty across the software engineering community, affecting levels of commitment to the many potential determinants of quality among developers. An up-to-date catalogue of software quality views could provide developers with contemporary guidelines and templates. In fact, it is necessary to learn about views on the quality of code on frequently used online collaboration platforms (e.g., Stack Overflow), given that the quality of code snippets can affect the quality of software products developed. If quality models are unsuitable for aiding developers because they lack relevance, developers will hold relaxed or inappropriate views of software quality, thereby lacking awareness and commitment to such practices.
Objective: We aim to explore differences in interest in quality characteristics across research and practice. We also seek to identify quality characteristics practitioners consider important when judging code snippet quality. First, we examine the literature for quality characteristics used frequently for judging software quality, followed by the quality characteristics commonly used by researchers to study code snippet quality. Finally, we investigate quality characteristics used by practitioners to judge the quality of code snippets.
Methods: We conducted two systematic literature reviews followed by semi-structured interviews of 50 practitioners to address this gap.
Results: The outcomes of the semi-structured interviews revealed that most practitioners judged the quality of code snippets using five quality dimensions: Functionality, Readability, Efficiency, Security and Reliability. However, other dimensions were also considered (i.e., Reusability, Maintainability, Usability, Compatibility and Completeness). This outcome differed from how the researchers judged code snippet quality.
Conclusion: Practitioners today mainly rely on code snippets from online code resources, and specific models or quality characteristics are emphasised based on their need to address distinct concerns (e.g., mobile vs web vs standalone applications, regular vs machine learning applications, or open vs closed source applications). Consequently, software quality models should be adapted for the domain of consideration and not seen as one-size-fits-all. This study will lead to targeted support for various clusters of the software development community.

1. Introduction

Software quality has long been a driving concern in software development, and has been used to evaluate the success of software products (Al-Qutaish, 2010). Poor quality in critical and real-time systems can result in financial loss, permanent disability, mission failure, or death (Jamwal, 2010). ISO8402:1986 defined software quality as "The totality of features and characteristics of a product or service that bear on its ability to satisfy specified or implied needs." ISO9001:2008 further expanded this view as follows: "The quality of something can be defined by comparing a set of inherent characteristics with a set of requirements. High or excellent quality is achieved if those inherent characteristics meet all requirements. If those characteristics do

☆ Editor: Patricia Lago.
* Corresponding author.
E-mail address: glory.ndukwe@otago.ac.nz (I.G. Ndukwe).
URL: http://www.otago.ac.nz/info-science (I.G. Ndukwe).
[1] Funding: Science for Technological Innovation — SfTI (Grant Number: UOOX2001).

not meet all requirements, a low or poor level of quality is achieved." (p. 6)

Several quality models and characteristics have been proposed to measure software quality (Kumar and Gupta, 2017; Miguel et al., 2014; Thapar et al., 2012). However, views on software quality models have shifted over time. For instance, Thapar et al. (2012) categorised software quality models into basic and tailored quality models, while Miguel et al. (2014) extended their work and added a third category to this taxonomy described as open-source quality models. This evidence suggests that software quality models may be tailored to address context-specific areas (e.g., code snippets, Internet of Things (IoT), mobile applications).

These different views of software quality have resulted in variability in the recommended frameworks and models for judging software quality. For instance, Abdallah et al. (2019) proposed a model for measuring the quality of IoT systems, where they singled out Robustness and Scalability as pertinent characteristics. Similarly, Franke et al. (2012) proposed a software quality model for mobile applications. They claimed that their model extracts the essential characteristics relevant for mobile systems and applications. Therefore, there is a need to catalogue software quality models to understand how academics view software quality, synthesise the quality characteristics commonly used to study code quality, and examine those commonly used by actual software practitioners to judge the quality of code snippets.

This latter evidence is particularly necessary because, in recent times, software practitioners often consult help from online collaborative platforms during development (Chatterjee et al., 2020; Xu et al., 2017). Consequently, considering the quality of the code snippets from these platforms is important and can affect the quality of the final software products. If this is not addressed, developers may hold relaxed or misaligned views of software quality, thereby lacking awareness and commitment to suitable practices. It is thus pertinent that studies understand the quality of code snippets and how they are judged. Hence, we conducted two systematic literature reviews. The first study focused on software quality models to catalogue and understand academics' views on software quality. This study is used to establish and understand software quality more widely to provide a baseline of knowledge which is used to inform the second study. The second study focused on the quality characteristics researchers commonly use to study the quality of code snippets in light of the increasing use of code available on online collaborative platforms during software development. In order to validate the relevance of the quality characteristics researchers commonly examine and to establish convergence with practitioners' views, we next performed an in-depth semi-structured interview study focused on practitioners' judgement of code snippet quality.

The main contributions of this paper are as follows. We catalogue software quality models and synthesise how researchers study and view the quality of code snippets. Finally, we investigate practitioners' views of code snippet quality and compare the views on quality attributes between research and practice. Insights gained from this study advance knowledge and provide a suitable reference guide for software practitioners who reuse code from online collaborative portals. Overall, our contributions provide a landscape of insight from systematic literature reviews and practitioners for the extension of research and insights for software development practitioners.

The remaining sections of this paper are organised as follows. We provide the study background and research questions in Section 2. We next detail the methods that are used in Section 3 before presenting and discussing our results in Section 4. We consider threats to the study in Section 5. Finally, Section 6 summarises our outcomes and outlines implications for research and practice.

2. Background and research questions

Software quality models have been around for decades and have demonstrated their worth in improving the quality of software systems (Miguel et al., 2014). The quality models defined by McCall et al. (1977) and Boehm et al. (1978) are two of the most cited for software in general because they outline the primary needs for software practitioners, including quality factors and characteristics, without limiting or focusing on a particular software domain context (e.g., full-fledged or code snippet applications). More recent software quality models have emerged, such as *ISO 9126* and *ISO 25010*, defined by the International Organisation for Standardisation (ISO). The *ISO 9126* quality model is based on *McCall* and *Boehm's* quality models. It identified external and internal quality characteristics of software products and proposed Functionality metrics. Subsequently, the *ISO 25010* standard emerged, updating the ISO 9126 model by redefining the fundamental characteristics and adding Security and Compatibility, increasing quality dimensions from six to eight.

Several studies have examined the software quality models mentioned above (Jamwal, 2010; Rawashdeh and Matalkah, 2006; Shoga et al., 2020; Suman and Rohtak, 2014). For example, Jamwal (2010) evaluated some of these standards and well-known quality models, including *ISO 9126*, *McCall*, and *Boehm*. Their study recommended *ISO 9126* as the most useful software quality model. However, these models did not consider some quality characteristics that are unique to specific domains (e.g., open-source applications). For instance, Bertoa and Vallecillo (2002) pointed out in their position paper that some of the characteristics of *ISO 9126* are too generic for dealing with open-source applications. Hence, they proposed a new set of quality characteristics for evaluating commercial off-the-shelf (COTS) products for Component-based Software Development (CBSD). Their model is a subset of *ISO 9126* that specifies the sub-characteristics that make sense for individual components. Similarly, Rawashdeh and Matalkah (2006) also based their model for evaluating COTS components on *ISO 9126*, notwithstanding a few limitations identified in this model (e.g., *ISO 9126* was held not to show clearly how quality aspects can be measured).

Furthermore, Adewumi et al. (2016) noted that even though the *ISO 25010* quality model can serve as a standard for open-source software (OSS) quality in terms of product quality and quality in use, it did not address the community aspect that is unique to *OSS*. It is plausible that software models may not claim universality because software development has shifted over time, and developers often need to access information dispersed among different documentation sources via the internet (Bacchelli et al., 2012; Chatterjee et al., 2020; Nasehi et al., 2012; Treude and Robillard, 2016; Xu et al., 2017). According to Ponzanelli et al. (2014), question and answer sites such as Stack Overflow provide a platform for knowledge sharing to help developers seek answers to their problems compared to online books and programming language documentation. Consequently, online collaborative platforms have become the primary source of harvesting rich programming-based content with better and faster access to support software development communities (Ahasanuzzaman et al., 2018; Barua et al., 2014; Beyer et al., 2018; Squire, 2015).

Accordingly, studies have examined quality more specifically for smaller code snippets that are often reused from online collaborative platforms (Geremia et al., 2019; Meldrum et al., 2020b; Posnett et al., 2011; Scalabrino et al., 2018; Verdi et al., 2020). For instance, Posnett et al. (2011) proposed a *Readability* metric for judging how easy it is to read a code snippet. Geremia et al. (2019) considered some code quality factors such as *Readability* and *Reusability* as the important quality characteristics for measuring code quality. Verdi et al. (2020) used the common weakness enumeration (CWE) list to scrutinise code snippets for security vulnerabilities. Furthermore, Meldrum et al. (2020b) explored various aspects of code snippet quality, including *Reliability*, *Readability*, *Performance* and *Security*.

A few other studies have also examined practitioners' views of snippet quality (Bai et al., 2019; Buse and Weimer, 2009; Tavakoli et al., 2020). For instance, Buse and Weimer (2009) carried out a study that surveyed the *Readability* rating of 120 participants on 100 code snippets, yielding a novel code *Readability* metric. Similarly, Tavakoli et al. (2020) proposed a predictive model for judging code snippets' quality based on *Completeness*, *Conciseness*, *Correctness* and *Comprehensibility*, as a result of studying how 60 participants with industrial experience and a Computer Science degree judged 3600 code snippets.

While previous work has focused on software quality, we have noticed there is limited research on code snippet quality. Consequently, it is difficult to get an overview of the role of the quality characteristics used in code snippets, especially from the perspective of software practitioners. Hence, to bridge this gap, our work aims to comprehensively understand the quality characteristics used to examine code snippets and compare the differences between the quality characteristics researchers often studied with those the software practitioners consider important when judging code snippet quality. We examine software models focused on software quality and more recent quality characteristics used in research to study the quality of shorter code/snippets by researchers. We then investigate how software practitioners conceptualise/judge the quality of code snippets from online code sources.

Our overarching goal is to compare the quality characteristics that are often studied by researchers with those that software practitioners need and find relevant when assessing code snippet quality. First, we conducted a tertiary study (i.e., a systematic literature review of secondary studies) (Kitchenham et al., 2009) to catalogue software quality models. This study is used to establish and understand software quality more widely to provide a baseline of knowledge which is used to inform the second study. This tertiary study approach provides an overview of research conducted on software quality and recommendations provided for determining software quality. It also provides a summary to understand academics' views on software quality. Second, we then carried out a conventional systematic literature review to analyse primary studies on code snippets. This approach was undertaken to provide a deep comprehension of what has been done on code snippet quality in order to identify gaps in this area of research and synthesise relevant quality dimensions. The second study focused on the quality characteristics researchers commonly use to study the quality of code snippets in light of the increasing use of code available on online collaborative platforms during software development. We only focused on Stack Overflow[2] in this phase because it is the most popular community question and answer portal developers use (Meldrum et al., 2020a). Third, we investigated practitioners' views on code snippets by conducting semi-structured interviews to gain rich and detailed knowledge of how practitioners judge the quality of code snippets. Insights from this latter study are used to validate the relevance of the quality characteristics researchers commonly examine and to establish convergence or otherwise with practitioners' views.

[2] https://stackoverflow.com/

2.1. First phase research questions — software quality

The main research question for this study is — ***RQ1*** *What are the models that are commonly referenced when considering software quality?* This research question aims to examine all the software quality models by reviewing secondary studies to identify the models that are referenced in each study. The intended outcome of this study is to catalogue all the software quality models in order to learn about the quality characteristics considered. The outcome of this study also lays the foundation for the second phase of our study. This study contributes to evidence-based software engineering by systematically identifying and thoroughly synthesising new evidence based on the data extracted from selected research publications in software quality. Sub-questions are outlined as follows to guide this phase of the study:

RQ1a*. What are the software quality models commonly referenced by review papers?*

RQ1b*. What are the quality characteristics observed to be commonly attributed to software quality models?*

RQ1c*. How similar are the characteristics in software quality models?*

2.2. Second phase research questions — code snippet quality

The main research question for the second phase of this study is — ***RQ2*** *What are the research interests, programming languages and quality characteristics used to investigate code snippet quality?* This question aims to investigate code snippet quality and synthesise the knowledge gained from reviewing papers focused on the quality of these particular artefacts. The outcome of this study is to have a descriptive summary of the various dimension and quality attributes researchers primarily study concerning code snippet quality and the programming languages used mainly by researchers to evaluate the quality of code snippets on Stack Overflow. This study contributes to the knowledge on code snippet quality dimensions that are the focus of academic research on code snippet repositories. The sub-research questions are as follows:

RQ2a*. What are the areas of research interest on code snippet quality?*

RQ2b*. What are the programming languages popularly used to evaluate code snippet quality?*

RQ2c*. What are the quality characteristics used to investigate the quality of code snippets?*

RQ2d*. What are the relationships between the quality characteristics used to determine the quality of code snippets?*

2.3. Third phase research question — practitioners' views of code snippet quality

The research question for this study is — ***RQ3*** *How have views on Software Quality differed between research and practice?* This research question aims to investigate how practitioners perceive the quality of code snippets and to understand the quality characteristics that practitioners use to judge the quality of code snippets. We also aspire to know if there are other attributes we have not yet considered as an academic community when evaluating code snippet quality, to validate the relevance of the quality characteristics researchers commonly examine, and to establish convergence with practitioners' views. Hence, we interviewed participants who often use Stack Overflow when performing programming tasks to determine the software quality characteristics

Table 1
First phase database searches.

Search terms	Databases	Subtotals
("Software Quality" OR "Quality Model" OR "Product Quality" OR "Software Quality Framework") AND ("Systematic Review" OR "Systematic Mapping" OR "Comparative Study" OR "Review")	ACM Digital Library	68
	IEEE Xplore Digital Library	27
	Scopus	18
	Springer	2
	Google Scholar	93
Total Studies		**208**

they identify as important measures when judging code snippet quality. Overall, the outcome of this study helps to establish how views on code snippet quality have differed between research and practice. This study contributes to the overall quality of code snippets and to creating awareness in the software development community around using code snippets from open source repositories such as Stack Overflow. The sub-research questions are as follows:

RQ3a. *How do software practitioners judge the quality of code snippets?*

RQ3b. *What are the differences and similarities in quality dimensions between code snippet quality and software quality?*

3. Research methods

This section addresses the relationship between the three research phases. The main research question in the first study focuses on software quality, which we considered broad, and the output from this study provides the breadth necessary to provide baseline knowledge for the second phase of this study. The second main research question focuses on code snippet quality in light of the increasing use of code available on online collaborative platforms during software development. Here the quality characteristics from the first phase are used as background knowledge to extract quality characteristics studied mainly by researchers on code snippet quality in the second. Lastly, the quality characteristics from phase two are used as baseline categories in the third phase of the research to tease out the quality characteristics software practitioners consider important when judging the quality of code snippets. However, we also tease out categories not considered by researchers, as derived from phase two. Furthermore, we compare outcomes from the second and third phases to assess convergences and differences in views on the quality of code snippets between researchers and practitioners to achieve the overall goal of this research.

3.1. First phase methods — software quality

In this phase, we undertook a tertiary literature review following the guidelines proposed by Kitchenham et al. (2010). In this case, the goal of the review is to assess systematic literature reviews (also referred to as secondary studies) to address research questions (**1a**, **1b** and **1c**). The steps followed in the systematic literature review are documented below.

3.1.1. Search process

We performed a broad automatic search of five specific databases for secondary studies that have titles including one or more of the search terms: *quality model*, *software quality*, *product quality*, *software quality framework*, *systematic review*, *systematic mapping*, *comparative study*, *review*. We searched ACM Digital Library, IEEE Xplore Digital Library, Scopus, Springer and Google Scholar, in line with the recommended databases for software engineering reviews (Kitchenham et al., 2010). All searches were based on titles. The searches took place in May 2021, covering all years of publications. Table 1 summarises the generic search strings and returned results from the different databases (refer to the replication package in Section 3.4 for the detailed search strings we used to search each database and the number of papers returned from each search string accordingly).

3.1.2. Study selection

The first author integrated the results for the different searches and undertook an initial screening of the 208 secondary studies found, focused on excluding irrelevant or duplicated studies. Consequently, the following papers were excluded, 48 duplicate titles, 119 irrelevant titles, and 10 irrelevant abstracts. The second author performed an informal *reliability check* on 40 (25%) randomly selected excluded papers at this stage (Liao and Hitchcock, 2018). The inter-rater agreement level between the two authors returned a value of 0.93, indicating the two authors reached excellent an agreement (Cohen, 1960). Consequently, after discussion and shared context, the two authors resolved the initial disagreement on two excluded papers. Thereafter, the first author subjected the remaining 31 papers to the inclusion and exclusion criteria, where 9 papers remained (see Section 3.1.3).

Furthermore, we performed a manual search via backwards snowballing of the included articles (Molléri et al., 2016). Contextual reading of the papers provided references to related and similar studies. Articles collected during this stage were also aggregated into the candidates list and selected according to the criteria described above. The process was performed once, as no relevant references were available after the first iteration. Consequently, we found 2 additional papers. Finally, a total of 11 papers were selected (refer to the replication package in Section 3.4 for details of the selected papers) and used for the data extraction process described in Section 3.1.4. The second author also checked any papers included and excluded at this stage.

3.1.3. Inclusion and exclusion criteria

We define our inclusion criteria based on the known models and their requirements or specifications for judging the quality of research, as follows:

1. The paper is a systematic literature review, mapping study, narrative review (unstructured review), or comparative study, from peer-reviewed journals or conference proceedings.
2. The paper performed a comparative analysis of software quality models and their quality characteristics. This criterion provides "fast" evidence and reduces the timeframes in retrieving frequencies, similarities, and differences between various software quality models and their characteristics.

Secondary studies on the following topics were excluded:

1. The paper focused on aspects of software quality outside our scope (e.g., cost, control, impacts or prediction).
2. The paper considered software quality models only but did not identify their various quality characteristics or generalise the characteristics of software quality models.
3. The paper is written in a language other than English.

Table 2
Second phase database searches.

Search terms	Databases	Subtotals
("Quality" OR "Code Quality" OR "Snippet Quality" OR "Coding Quality" OR "Code Snippets" OR "Code Fragments") AND ("Stack Overflow" OR "Stackoverflow")	ACM Digital Library	66
	IEEE Xplore Digital Library	67
	Scopus	116
	Springer	50
	ScienceDirect	13
	Wiley InterScience	2
Total Studies		**314**

3.1.4. Data extraction and synthesis

Using a manual process, the first author undertook the data extraction from the secondary studies used in this research phase. The following data were extracted, software quality models, the year the quality model was proposed, and the quality characteristics that were identified and used to evaluate or compare the software quality models.

We held weekly meetings where the research team discussed various aspects of the research work. Part of these meetings discussed the extracted data at this stage and resolved any discrepancies discovered regarding a few papers. For example, paper (p4) reported that Dromey was proposed in 1995, while paper (p62) reported a different year, 1992 (refer to the replication package in Section 3.4 for the secondary studies that were extracted at this phase). We decided to refer to the primary sources to get the correct proposed year for conflicting model years. At the end of this process, we extracted 60 software quality models and 73 unique characteristics that were used to compare quality models against each other in the selected papers. The result of this extraction process is discussed further in Section 4.1.

3.2. Second phase methods — researchers' views on code snippet quality

In this phase, we conducted a systematic literature review to explore the research space on code snippet quality. We describe our review process to answer the three research questions (**2a**, **2b**, and **2c**), enabling for deeper understanding of research views on code snippet quality.

3.2.1. Search process

The first author performed broad searches on six databases starting from 2009 for primary studies on Stack Overflow. The year was chosen as the start date of our search because Stack Overflow emerged in late 2008. Hence, this study reviewed 12 years of papers on Stack Overflow code quality. All the searches took place in June 2021 and were based on title and abstract. The following terms were used to perform the searches: *stack overflow*, *code quality*, *snippet quality*, *code fragments*, and *code snippets*. The electronic data sources searched were ACM Digital Library, IEEE Xplore Digital Library, Scopus, Springer ScienceDirect, and Wiley InterScience. Google Scholar was excluded from this search after it returned about 15,700 search results, of which most of the papers were irrelevant to our study (Lorigo et al., 2008; Meldrum et al., 2020b). Table 2 summarises the search strings and the papers retrieved from the different databases (refer to the replication package in Section 3.4 for the detailed search strings we used to search each database and the number of papers returned from each search string accordingly).

3.2.2. Study selection

The first author integrated the results for the different searches and undertook an initial screening of the 314 primary studies found based on titles and abstracts. This screening was based on excluding studies that were irrelevant or duplicates. Consequently, the following papers were excluded, 146 duplicate titles and 115 irrelevant titles or abstracts. The second author performed a reliability check on 42 (25%) randomly selected papers to ascertain the trustworthiness of the excluded papers (Liao and Hitchcock, 2018). The inter-rater agreement level between the two authors was 0.89 (Cohen, 1960). This value is considered as an excellent agreement between the authors. However, we discussed a few differences and reached a consensus on all the papers; after discussing and sharing context. After the first author subjected the remaining 53 papers to the inclusion and exclusion criteria, 23 papers remained (see Section 3.2.3).

Furthermore, since most of the selected papers for review were recently published, we performed a manual search via backwards snowballing of the included articles (Molléri et al., 2016). A contextual reading of the papers provided references to related and similar studies. Articles collected during this stage were also aggregated into the candidates list and selected according to the criteria described above. The process was performed twice, as a few relevant references were available after the first iteration. Consequently, we found 4 additional papers. Finally, 27 papers were selected for the data extraction process described in Section 3.2.3. The second author also checked any papers included and excluded at this stage (refer to the replication package in Section 3.4 for details on the selected papers).

3.2.3. Inclusion and exclusion criteria

The selection process relied on a thoroughly specified benchmark to select relevant primary studies (Ali and Petersen, 2014; Petersen and Ali, 2011). Hence, we include:

1. Papers that we believe contribute to the body of knowledge on Stack Overflow code quality, such as providing metrics to measure code quality, including articles that performed code snippets evaluation.
2. Papers that are from peer-reviewed journals and conference proceedings.

Primary studies on the following topics were excluded:

1. Stack Overflow papers that only apply data mining, natural language processing, machine learning or deep-learning techniques, such as employing opinion mining to determine users' discussion about software qualities on Stack Overflow.
2. Papers that investigated the quality of Stack Overflow posts (questions, answers, or tags) generally without considering the quality of code snippets or performing code evaluation in the questions or answers.
3. Papers that used Stack Overflow data to conduct qualitative or quantitative analysis without performing code analysis, for example, a qualitative study on Stack Overflow users to determine the characteristics of a Stack Overflow post.
4. Papers are written in a language other than English.

3.2.4. Data extraction and synthesis

The first author undertook the data extraction. However, the second author cross checked 6 (25%) randomly selected papers to ensure inter-researcher consistency, and the agreement level was

92% (Liao and Hitchcock, 2018). There was only one disagreement among the two authors on one of the quality attributes, which was later discussed and resolved.

The qualitative synthesis approach was used to synthesise extracted data (Kitchenham and Charters, 2007). The line of argument synthesis was specifically used because we were interested in using predefined themes to infer meanings from sections of interest in the selected studies on Stack Overflow code snippet quality. First, the individual studies were analysed using the directed content analysis method, and predefined themes of interest provided the context for our analysis based on our research questions (Hsieh and Shannon, 2005). Our units of analysis were sentences or paragraphs that ostensively bore relevance to the thematic categories (Hannay and Jørgensen, 2008). The predefined themes of interest are listed as follows:

1. Code Violations — bugs found in code snippets resulting in unexpected behaviours or stopping the code from running.
2. Code Vulnerabilities — insecurities or weaknesses found in code snippets.
3. Code Reuse — reusing of code snippets from other projects hosted on publicly available repositories such as GitHub and Android apps.

Other data that were extracted are quality attributes/characteristics and programming languages the paper used to perform their code snippet quality evaluation.

3.3. Third phase methods — practitioners' views of code snippet quality

Towards addressing the research questions (**3a** and **3b**), we outline the methods used in this phase to understand how practitioners judge the quality of code snippets. We discuss how the definition of each software quality characteristic has differed from the practitioners' views and compare the practitioners' and researchers' views on quality characteristics.

3.3.1. Interviews

Data was collected through semi-structured interviews, which were conducted between July–August 2021. The interviews comprised 6 demographic questions (see Table 3) and 8 main interview questions. However, given the objectives of this study, we only report data from four interview questions (refer to the replication package in Section 3.4 for the details on the interview questions, only the highlighted questions were used for this study). In terms of the basis for the interview questions that were used in this study, we provide reflections as follows:

IQn1. *How do you judge code quality in the answers posted on the portals you use?* We asked this question in order to extract general knowledge about how the software practitioners that participated in this study perceived the quality of code snippets when they are searching for solutions.

IQn2. *When using the portal, if you have more than one solution provided for your problem, why would you pick one over the other and what is the motivation for this decision?* We asked participants this follow-up question in order to probe further and tease out knowledge about how participants evaluated the quality of code snippets when multiple solutions were available for a single problem. This question helped us to learn more about practitioners' basis for discriminating (in terms of quality) among many solutions online.

IQn3. *The standard quality models cover key areas of software quality, including Functionality, Efficiency, Maintainability, Usability, Reliability, Portability, Compatibility and Security. At the same time, others have tended to focus on code snippet quality along the dimensions of Security, Readability, Reusability and Completeness. Which ones in the list do you believe are particularly relevant to the quality of code snippets?* This question aims to filter down which of the quality characteristics software participants that participated in this study considered important when judging code snippet quality. Additionally, this question came as the third interview question because we wanted to reduce bias in directing the participants on how they would judge the quality of code snippets, as asked in the first interview question. Of note also is that this question reflected the outcomes of our two systematic reviews.

IQn4. *Are there other aspects of quality not discussed that you find important when assessing the quality of code snippets?* This question aims to extract other relevant quality characteristics that the software practitioners that participated in this study considered important when judging the quality of code snippets but were not listed in the previous question. Here we probed additional code snippet quality characteristics that are considered necessary by practitioners in order to assess the convergence of research and practice.

3.3.2. Participants

The study was publicised, and its purpose explained, having received human and behavioural ethics approval from the university in which the study was conducted (D21/218). Convenience sampling was adopted because it provided the authors with easy and regular access to interview software practitioners with years of industry experience in software development, and those that regularly use open source online repositories like Stack Overflow. Additionally, the convenience sampling technique allowed us to carry out the study in a time-efficient and cost-effective manner. Of note also is that all of the practitioners interviewed had industry experience and regularly developed software.

We invited participants to participate in the study by emailing staff and student members belonging to the computer and information science division of one of the Universities in New Zealand. We also invited staff and students who write code from other divisions to participate in the study. We then extended the invitation to the same institution's ICT unit and expanded to a few other tech companies in New Zealand. The invitation began with a short explanation of the research being conducted, the length of the interview, the address of the interview venue, a unique code, and a link to an online Google sheet with available time slots and meeting points. Interested participants would enter their unique code against the available time slot that suits them and then select a meeting point option (e.g., remotely or in-person).

Fifty-two (52) candidates initially indicated their interest in participating in this study. However, 2 potential respondents dropped out during the interview when they did not meet the selection criterion. The selection criterion was simply determined by answering "No" to the first interview question (Do you refer to free online sources or CQA portals for help or contribute?). Hence, Fifty (50) software practitioners participated in this study. The sample size is deemed adequate for the chosen (convenience) sampling method (Marshall, 1996). We reached saturation where no new information or viewpoints were gained from new subjects after analysing the 39th interview (Runeson and Höst, 2009).

Table 3
Participants' demographics (PG: post graduate diploma).

Demographics		Number	Percent
Gender	Male	41	82%
	Female	9	18%
Education	PhD degree	20	40%
	MSc degree	14	28%
	BSc degree	13	26%
	Diploma	3	6%
Occupation	Software Developers	24	48%
	Computing Academics	14	28%
	Database Administrators	7	14%
	PG Computing Students	5	10%
Experience (Years)	1 to 5	18	41%
	6 to 10	6	13%
	11 to 15	5	11%
	16 to 20	5	11%
	21 to 25	6	13%
	26 or more	5	11%
Organisation Size (Employees)	0 to 999	14	31%
	1000 to 9999	28	62%
	10000 or more	3	7%
Programming Language Competence	Others	54	27%
	Python	36	18%
	C	26	13%
	Java	25	13%
	C++	21	11%
	R	13	6%
	Javascript	9	5%
	Pascal	7	4%
	C#	7	4%

3.3.3. *Demographic records*

The participants' demographic information was recorded as part of the interview process, including gender, education, occupation, years of development experience, programming language competency, and company size. These details were used to complement those from the interviews in supporting our analysis and interpretation of the outcomes (see Table 3 for detailed participants' demographic information).

3.3.4. *Tools and measures*

The conversations during the interview were audio-recorded using Zoom[3] and transcribed verbatim using Otter.ai.[4] A co-author then verified transcripts by reading through each transcript while listening to the audio play from the online transcription tool to correct spelling errors. In order to ensure that interviewees were not cited wrongly, it was agreed that the transcribed interviews were sent back to 25% of them (Lincoln and Guba, 1985). The respondents' IDs, responses and questions were treated as the units of analysis to identify transcripts. For example, GN1AQn4 is a combination of participant ID (e.g., GN1, GN2, …, GN50), the highlighted comment section (e.g., A, B, C, …, Z), and the interview question (e.g., Qn1, Qn2, …, Qn8).

3.3.5. *Data extraction and synthesis*

We applied a deductive approach based on the eleven key characteristics identified from our prior research in phases 1 and 2, commonly used to measure software and code snippet quality (Potter and Levine-Donnerstein, 1999). Specifically, the directed approach to content analysis was used to validate and extend the predetermined categories and provide predictions about the categories of interest and their relationships (Hsieh and Shannon, 2005). This approach also helped determine the key concepts and initial coding categories used in this study (Hickey and Kipping, 1996; Potter and Levine-Donnerstein, 1999). Furthermore, we read the transcripts for familiarisation. Then we separated them into subcategories using predetermined categorised codes. Data that could not be categorised within the predetermined coding scheme were analysed and given a new category to further extend and enrich our catalogue of the quality dimensions of code snippets. Lastly, Spearman's rho correlation test was conducted to determine whether any relationship exists between the various features in the demographics data (refer to the replication package in Section 3.4 for details). Outcomes from the three phases of analyses are provided in the next section.

[3] https://zoom.us/
[4] https://otter.ai/

3.4. *Datasets*

The research protocol followed, and the final/intermediate data (e.g., search query results, included papers, excluded papers, extracted final data, and detailed network diagrams) of this study are provided in a unified replication package available online (see https://doi.org/10.5281/zenodo.6909618).

4. Results and discussion

We document and discuss our results in this section, considering the first outcomes from our review of software quality more generally in *Section* 4.1. We next present the review results and discuss these outcomes for code snippet quality in *Section* 4.2. Finally, we present results and discuss the outcomes of our semi-structured interviews in *Section* 4.3, targeting and teasing out practitioners' views of the relevant quality dimensions of code snippets.

4.1. *First phase — what are the models that are commonly referenced when considering software quality?*

RQ1a. *What are the software quality models commonly referenced by review papers?* This question aimed to catalogue all the software quality models that have been proposed to evaluate software quality. We used a bar plot to represent the frequency of

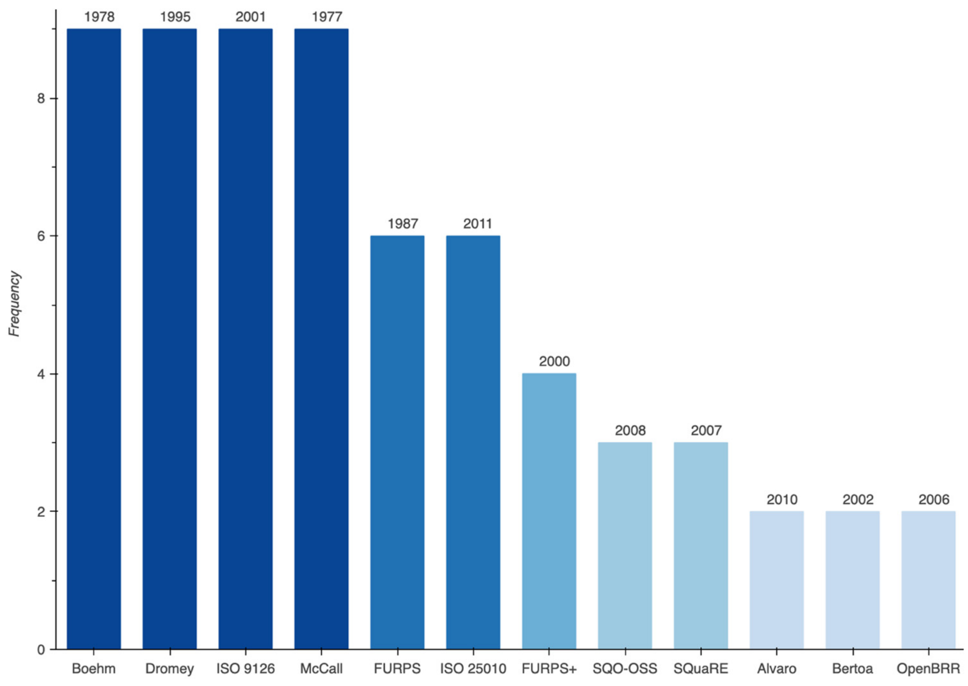

Fig. 1. Frequencies of software quality models with the year each model was proposed.

references made to software quality models. As shown in Fig. 1, models with darker blue colours represent higher frequencies, while lighter blue represents the lower frequency models. The year individual models were proposed are found on the top of each bar. Of the 60 software quality models mentioned in the review papers selected for this synthesis, *Boehm*, *Dromey*, *ISO 9126*, *ISO 25010*, *FURPS* and *McCall* were the top six most commonly referenced models. *FURPS+*, *SQO-OSS*, and *SQuaRE* were in the middle range, while *Alvaro*, *Bertoa*, *OpenBRR* were in the bottom range (refer to the replication package in Section 3.4 for a detailed figure on all the models, including the rarely referenced models).

McCall's model is one of the earliest software quality models (McCall et al., 1977). This model divides quality into 11 characteristics: *Correctness*, *Reliability*, *Efficiency*, *Integrity*, *Usability*, *Maintainability*, *Testability*, *Flexibility*, *Portability*, *Reusability*, and *Interoperability*. Each one of these 11 characteristics is related to a set of criteria. A quality criterion can be related to more than one quality characteristic and vice versa. Each criterion can be associated with one or more quality measure that captures some aspect of the criterion. *Boehm's* model is another seminal software quality model that was proposed in 1978. This model mainly focuses on software quality from the developer's viewpoint and splits quality into seven quality characteristics (also known as intermediate constructs): *Portability*, *Reliability*, *Efficiency*, *Human Engineering*, *Testability*, *Understandability*, and *Modifiability*. These intermediate constructs are further divided into primitive constructs, which could be measured using metrics (Boehm et al., 1978). The *Dromey* model is based on the perspective of product quality. This model is theoretical and centred around four pillars of product quality, with quality characteristics defined under each (Dromey, 1995).

Among the top five models, *Boehm*, *Dromey*, and *McCall* are commonly combined or adapted to give rise to newer models (Miguel et al., 2014; Narasimhan and Hendradjaya, 2007; Sharma and Baliyan, 2011). For example, knowledge from the above models mixed with international efforts led to the international standard *ISO 9126* for software quality. The *ISO 9126* comprises a basic set of 6 independent quality characteristics: Functionality, Reliability, Usability, Efficiency, Maintainability, and Portability. Additionally, *ISO 9126* is a recognised model in the software quality community, discussed and referenced by many academics and researchers in software quality (Fitrisia and Hendradjaya, 2014; Hendradjaya et al., 2014). However, some early and popular software quality models have also been criticised. For instance, Hendradjaya et al. (2014) asserted that the characteristics and sub characteristics of the ISO 9126 model are too generic.

The significant recognition of *ISO 25010* is also notable, given its recent emergence. This recognition indicated in the result is in line with Nistala et al. (2019) and Oriol et al. (2014) acknowledging the emergence of the *ISO 25010* as a model starting to be widely used in the measurement of software quality after the existence of its revision, *SQuaRE*. Similarly, *FURPS+* was extended from *FURPS* by IBM, considering only the user requirements and disregarding developer considerations (Al-Badareen et al., 2011). This evidence further suggests that newer standards and context-specific models have attracted increasing attention from researchers. Compared to other top models, there were generally low occurrences of open-source models from the secondary studies, *SQO-OSS* was at the top (3 occurrences), followed by *OpenBRR* (2 occurrences) and others (1 occurrence). This trend is not surprising, considering that most open-source models emerged in the mid-2000s. However, it is also an important indication of the emergence of context-specific models or models designed to fit specific needs (Kumar and Gupta, 2017; Miguel et al., 2014; Thapar et al., 2012).

RQ1b. *What are the quality characteristics observed to be commonly attributed to software quality models?* This question aimed to get more insight into the quality characteristics commonly used to compare or evaluate software quality models. Fig. 2 shows the 20 top characteristics that commonly appeared in the synthesis of software quality models in the review papers. Overall, *Reliability*, *Maintainability*, *Efficiency*, *Usability*, *Functionality* and *Portability* emerged as the top six characteristics. *Testability*, *Reusability*, *Understandability*, *Security*, *Interoperability*, *Integrity*, and *Correctness* were in the middle tier. *Flexibility*, *Maturity*, *Safety*, *Performance*, *Operability*, *Supportability*, and *Modifiability* were towards the bottom of the chart.

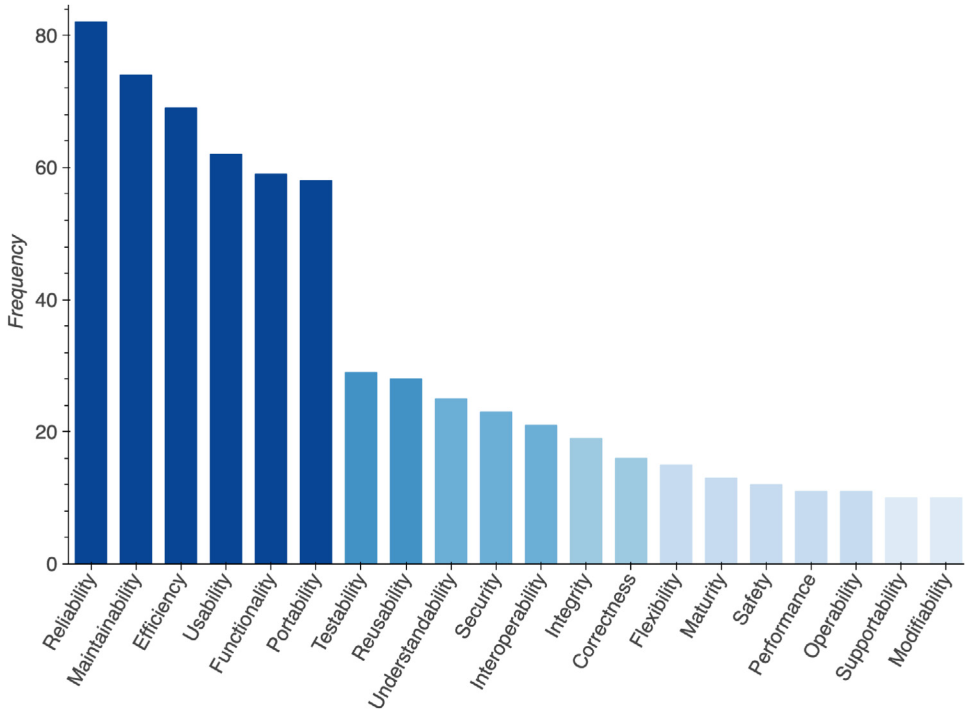

Fig. 2. Frequencies of top twenty software quality characteristics.

We should acknowledge that those model disparities between the sub-characteristics and characteristics may have impacted the interpretation of the middle-range and bottom-range characteristics, as observed by Singh and Kannojia (2013). For instance, in the *ISO 25010* model, *Testability* is a sub-characteristic of *Maintainability*. Also, *Understandability* is a sub-attribute of *Usability*, *Integrity* of *Security* and *Correctness* of *Functionality*. Another example is the *McCall* model, where *Testability*, *Interoperability*, *Integrity*, and *Correctness* appeared as main attributes with similar definitions. Additionally, some of the quality characteristics are common among models, e.g., *Reliability*. In contrast, others are particular to some models, e.g., *Maintainability*, which is present in *ISO 9126* but absent from the *FURPS* model, introducing *Supportability* and presenting *Maintainability* as a sub-characteristic under *Supportability*. We also noticed some quality characteristics were used interchangeably to refer to the same concept (e.g., Usability and *Operability* in *ISO 9126*, *ISO 25010*, and *Bertoa*, or *Modifiability* in *ISO 25010* and *Changeability* in *Boehm*).

The top six characteristics in this study correspond to the six characteristics of the *ISO 9216* model. This result perhaps indicates the relevance of *ISO 9216* in evaluating a wide range of software applications (Jamwal, 2010; Shoga et al., 2020; Suman and Rohtak, 2014). However, Singh and Kannojia (2013) acknowledged that *Security* is an important quality characteristic used in determining the quality of modern-day software, especially within the information era where *Security* is a top priority. Our result in Fig. 2 supports this claim as well. Even though Security is newly added to *ISO 25010*, it still emerges among the top ten quality characteristics. This outcome also indicates that a model can draw attention due to a specific quality characteristic that addresses a concern.

RQ1c. *How similar are the characteristics in software quality models?* Understanding the relationship between quality characteristics is important to sustaining a sufficient software quality-driven development process and related outcomes (Azuma, 1996). In this section, we explore the relationship between the quality characteristics of the software quality models. Fig. 3 shows a network graph of all the models referenced more than once by the review papers. This network graph represents the various models and characteristics, while edges link a model to various characteristics.[5]

The nodes are grouped into various sizes and colours. The dark blue nodes are the smallest, and they represent the models, while the yellow, light green and green bolder nodes represent the quality characteristics belonging to the various models. The sizes and colours of the characteristic nodes are proportional to the degree of similarity. The light green nodes are the smallest and represent the quality characteristics with the lowest degree of similarity, followed by the green nodes. Lastly, the yellow nodes represent the highest degree of similarity characteristics.

We slightly adjusted all node degrees to maintain clarity in the network graph representation. For instance, we adjusted the model nodes to the same degree (5°) to automatically assign them the same size and colour. On the other hand, we slightly increased the degrees of the characteristics nodes so that the nodes with small degrees were still visible. However, we only report the actual degrees of the characteristics nodes in this study. Hence, *Reliability* has the highest degree of similarity (11°), followed by *Usability* (10°), then *Maintainability* (9°), *Efficiency*, *Functionality* and *Portability* (7°), *Testability* (3°), *Integrity*, *Interoperability*, *Performance*, *Reusability*, *Security*, *Supportability* (2°), and others (1°).

Furthermore, *Reliability*, *Usability*, *Maintainability*, *Efficiency*, *Functionality*, and *Portability* emerged as the most notable characteristics nodes in the yellow category. *Testability*, *Integrity*, *Interoperability*, *Performance*, *Reusability*, *Security*, and *Supportability* were in the middle light-green category. Lastly, the others in the green category surfaced as the minor characteristics nodes.

Among the six notable characteristics already identified in this study, existing studies pointed out that earlier models (*McCall* and *Boehm*) did not consider *Functionality* as an attribute of software quality compared to their successors (*Dromey*, *ISO 9126*, and

[5] Detailed network graph of all the reference models and characteristics.

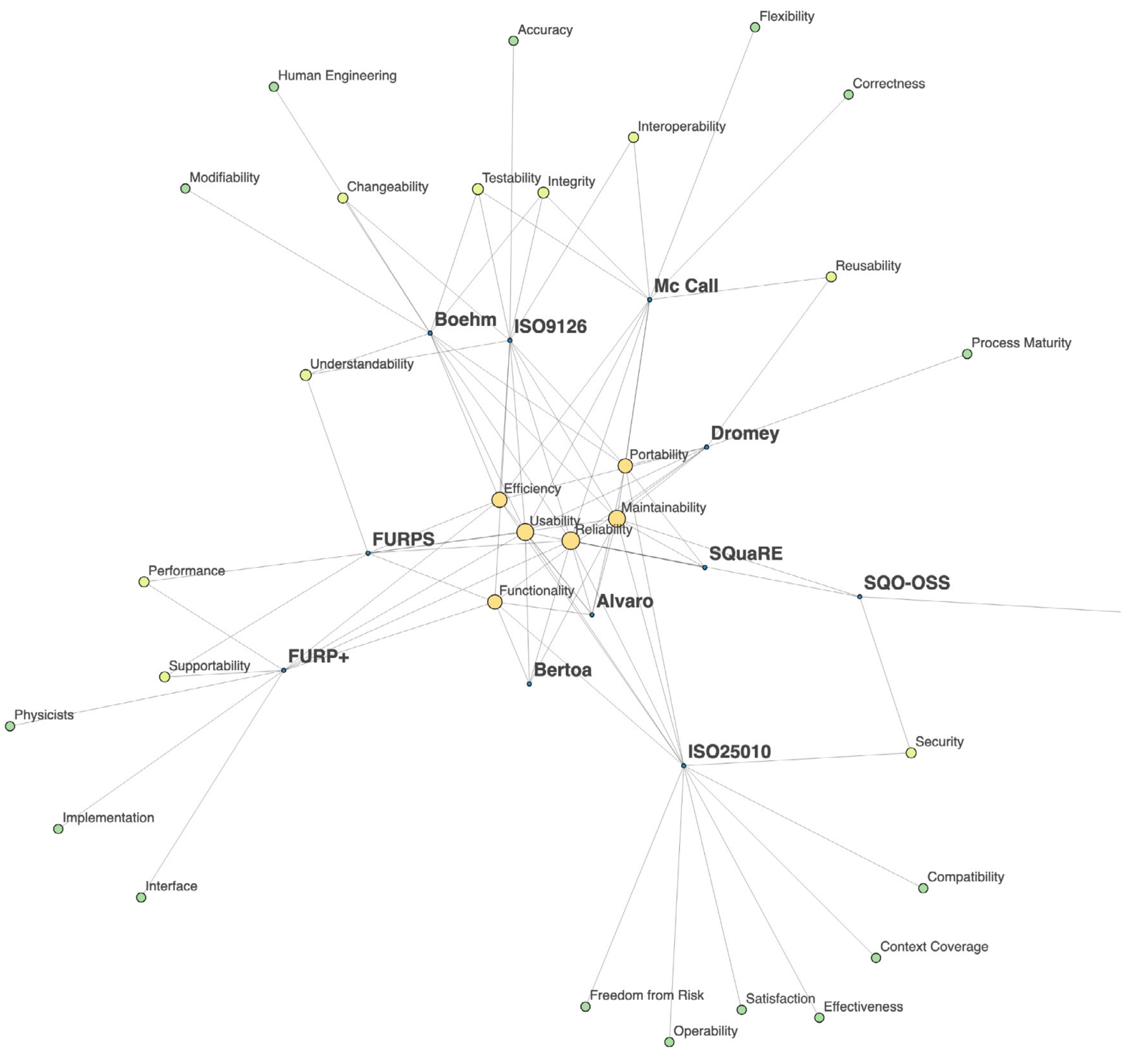

Fig. 3. Network diagram of software quality models and characteristics.

ISO 25010) (Jamwal, 2010; Rawashdeh and Matalkah, 2006). In this study, five out of the six notable characteristics (excluding *Functionality*) corroborate (Singh and Kannojia, 2013) as valuable attributes associated with software quality. The six most notable yellow nodes that correspond to *ISO 9126* six characteristics indicate that a clear majority of the software quality models commonly recommend these features to measure software quality (Saini et al., 2020; Shoga et al., 2020).

Furthermore, Fig. 4 shows the trends of the six notable characteristics over the last 40 years. We notice that there are differences in the quality characteristics over the years. For example, between 1990 to 2000, *Maintainability* and *Reliability* topped, and between 2000 to 2010, *Functionality* topped the list, even though the earlier models did not initially include this dimension. While *Usability* was on top of *Functionality* and *Efficiency* between 1990 to 2000, between 2000 to 2010, it was replaced by *Functionality* and *Efficiency*.

Some studies have focused on specific quality characteristics or recommended that quality characteristics be introduced for specific domains (e.g., IoT, machine learning) (Kim, 2016; Pons and Ozkaya, 2019; Vogelsang and Borg, 2019). For example, a study conducted by Pons and Ozkaya (2019) suggested that existing software quality models may not be sufficient to address the systems that contain machine learning components. They recommended that newer models with specific quality characteristics are needed to address such systems. Fig. 4 provides evidence that further supports this claim. Although *Portability* has consistently risen throughout the years, all the other quality characteristics experienced a spike between 2000 to 2010 and a drop between 2010 to 2020. This spike in the quality characteristics indicates the interest and relevance of the newer models within that era (e.g., *ISO 9126* and *ISO 25010*). However, it is also important to note that the drop on the right may indicate a shift from models with defined quality characteristics generalised to suit all software quality needs to context-specific models with specific quality characteristics to address specific software quality needs (e.g., code snippet quality).

4.2. Second phase — what are the research interests, programming languages and quality characteristics used to investigate code snippet quality?

RQ2a*. What are the areas of research interest on code snippet quality?* We extracted themes from the papers focused on the code quality of Stack Overflow. Fig. 5 shows the three topics typically discussed in the reviewed papers. Here it is shown that papers that analysed Stack Overflow *Code Violations* were the most frequent (13 papers), followed by *Code Vulnerabilities* (11 papers), then *Code Reuse* (7 papers) (refer to the replication package in Section 3.4 for a detailed breakdown of these various themes of interest).

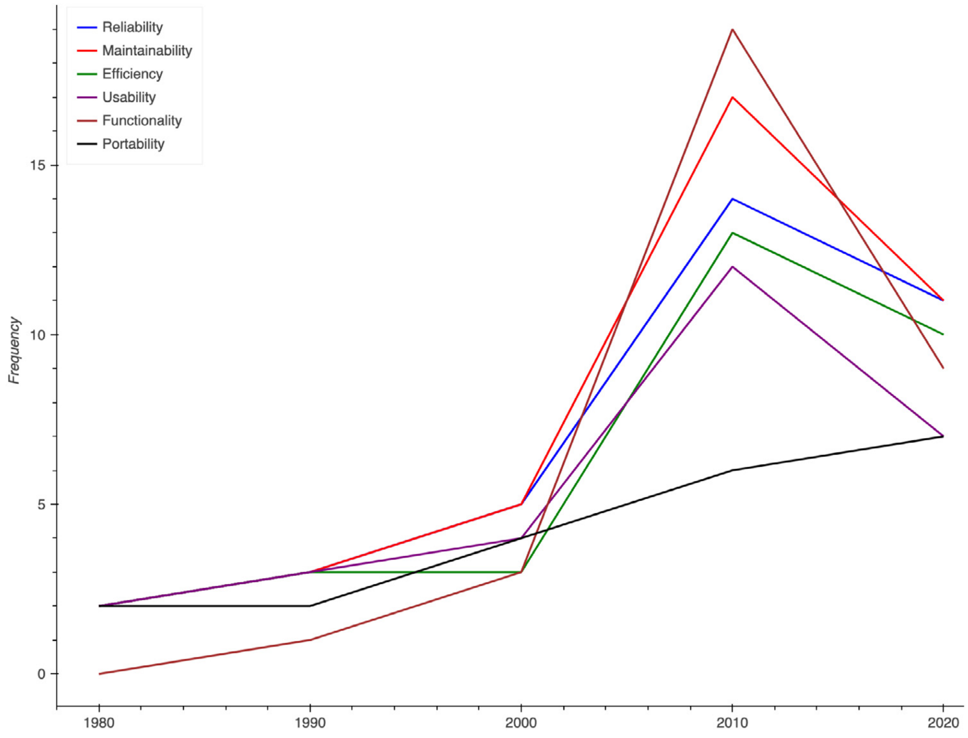

Fig. 4. Trends show how the six most notable characteristics have varied over time.

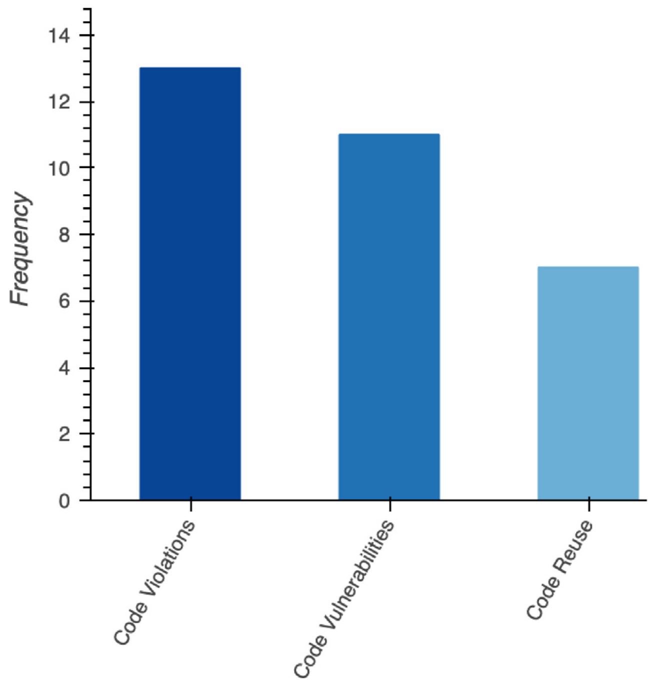

Fig. 5. Topics of interest on Stack Overflow code snippet quality.

Code violations. These are bugs found in code snippets that could produce unexpected behaviours or stop the code from running (Campos et al., 2019; Tavakoli et al., 2016; Zhang et al., 2018). Code snippets that do not follow naming conventions, generate warnings, are incomplete and need refactoring due to structural issues, all fall under Code Violations (Subramanian and Holmes, 2013; Treude and Robillard, 2017). They also include code snippets that are outdated mostly due to the use of outdated libraries, deprecated, and obsolete code snippets (Bafatakis et al., 2019; Campos et al., 2019; Ragkhitwetsagul et al., 2019; Tavakoli et al., 2020).

Zhang et al. (2018) classified API usage violations in Stack Overflow into three categories: missing control constructs, missing/incorrect order of API calls, and incorrect guard conditions. The missing control constructs happen in the Java programming language when code constructs such as `exception handling`, `checks`, and `finally` are missing, where appropriate, leading to unexpected behaviours and crashes due to erroneous values or no cleanups. Missing/incorrect order of API calls, as the name implies, is when a crucial API call is either missed or called in an incorrect order. This anomaly may lead to unexpected behaviours as well. Finally, incorrect guard conditions involve invoking APIs under improper guard conditions; for example, not emptying a map before calling the first key could cause a runtime exception.

In related prior work, Bafatakis et al. (2019) categorised code violations found in Stack Overflow code snippets into errors (return outside functions, no value for parameter), warnings (bad indentation, unused imports or variables), and convention (invalid name, training whitespace). Furthermore, Tavakoli et al. (2020) perceived code violation as deficient codes that may not be easy to understand because they are buggy, incomplete or concise without informative parts to solve a problem.

Code vulnerabilities. These refer to the insecurities or weaknesses found in Stack Overflow code snippets (mainly in the answer codes). They can result from the following; code injection (Chen et al., 2019b; Rahman et al., 2019a; Zhang et al., 2021), lack of education in addressing security issues (Acar et al., 2016), complex API and extensive code snippets (Meldrum et al., 2020a; Meng et al., 2018; Ye et al., 2018), and prioritisation of other requirements over security (Acar et al., 2016; Meng et al., 2018; Zhang et al., 2021).

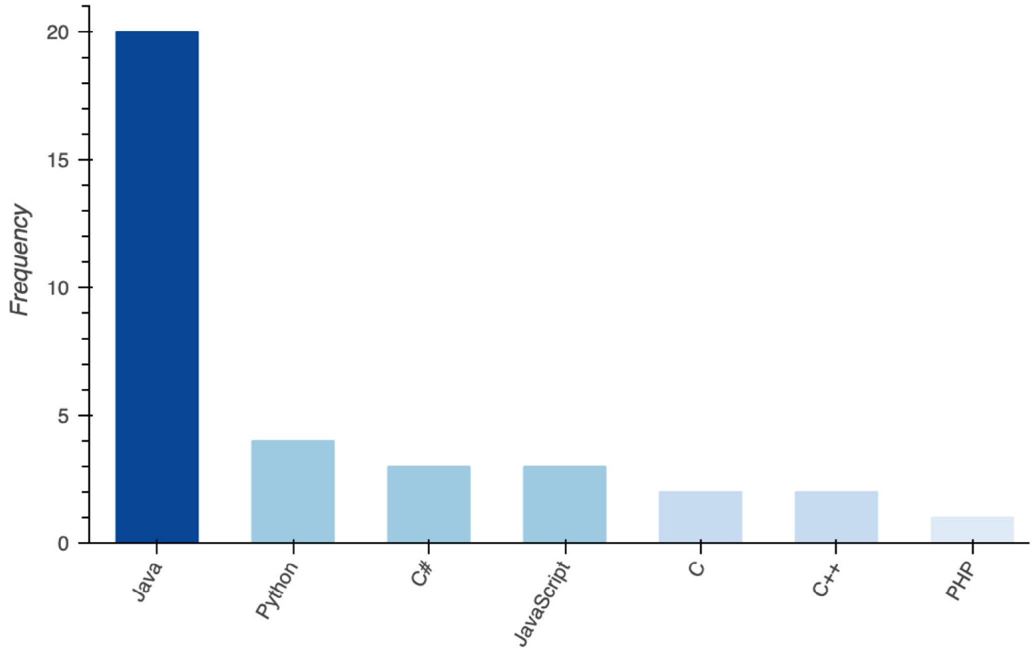

Fig. 6. Programming languages used to investigate Stack Overflow code snippet quality.

Rahman et al. (2019a) found that 7.1% of Stack Overflow code fragments examined contained insecure coding practices such as code injection or insecure cypher. In addition, Zhang et al. (2021) found that security issues in Stack Overflow are growing, albeit only 2% of the C/C++ code snippet answers on Stack Overflow with insecure code injection are detected. Several insecure and vulnerable code infiltrations found on Stack Overflow code snippets could pose serious security threats to real-world software systems such as Android apps (Acar et al., 2016; Chen et al., 2019b; Ye et al., 2018). Acar et al. (2016) identified that not addressing the security implications of using insecure code snippets creates the possibility for developers to copy and paste insecure functional solutions that avoid existing security measures without realising their actions.

Code reuse. This topic refers to the reuse of code snippets in other projects such as GitHub and Android apps. Other studies have looked at code reuse regarding the legality and licensing violations (Adaji and Vassileva, 2016; Arwan et al., 2015; Ellmann and Schnecke, 2018; Nishinaka et al., 2020; Rahman et al., 2019b; Zhang et al., 2018). However, our work considered determining quality code snippets worth reusing (Geremia et al., 2019; Reid et al., 2020; Yang et al., 2016, 2017) and how the reuse of code snippets affects project quality (Abdalkareem et al., 2017; Ahmad and Cinnéide, 2019; Digkas et al., 2019).

Geremia et al. (2019) analysed some code snippet characteristics such as lines of code, identifiers length, and the number of loops. They found that they were relevant in predicting code snippets of high quality worthy of reusing. Likewise, Yang et al. (2016) found that the compilable rate of statically-typed languages such as Java and C were low compared to dynamically-typed languages like Python and JavaScript. Consequently, statically-typed languages are harder to automatically reuse in projects.

Some studies believe that reusing code snippets from Stack Overflow reduces the quality of projects. For instance, Abdalkareem et al. (2017) found that the percentage of bug-fixing comments in each project file examined in their study increased after adding the Stack Overflow code snippet. Similarly, Ahmad and Cinnéide (2019) observed that recipient classes of a significant number of open-source community projects hosted on GitHub that reused code snippets from Stack Overflow deteriorated in cohesion. In contrast, a few studies believe that reusing code snippets rather improves the quality of software projects. For instance, Digkas et al. (2019) found that the effort required to fix code inefficiencies was significantly reduced in open-source projects such as those on Github that reused code snippets from Stack Overflow. They claim that reusing code from knowledge-sharing communities boosts productivity and increases project quality. Such a view may be deemed contentious, however, especially when considering that Stack Overflow code may possess quality concerns (Meldrum et al., 2020a).

RQ2b. *What are the programming languages popularly used to evaluate code snippet quality?* Fig. 6 shows the programming languages that were used to evaluate Stack Overflow code snippet quality in the various papers. Here it is shown that Java was studied the most (20 papers), followed by Python (4 papers), C# and JavaScript (3 papers), C and C++ (2 papers each), and PHP (1 paper). It is not surprising to see that Java topped the list of programming languages used to investigate the code quality of Stack Overflow.

Java[6] is one of the most popular and researched general-purpose programming languages on Stack Overflow (Ahmad and Cinnéide, 2019; Meldrum et al., 2020a; Treude et al., 2011; Ye et al., 2018). This popularity is partly due to the Java programming language for Android application development (Acar et al., 2016; Chen et al., 2019b; Fischer et al., 2017). Android is an open-source and customisable operating system for mobile devices that dominate the smartphone market by 82.8% (Hou et al., 2016a). According to Hou et al. (2016b), mobile devices have ever-expanding capabilities that have surpassed desktop and other media. Millions of Android apps have been installed on billions of users' mobile devices, attracting several software developers into the space of Android development. However, it has been asserted that most Android mobile apps have ineffective security implementations, largely due to untrained, preoccupied, or overwhelmed developers (Ye et al., 2018).

6 https://stackoverflow.com/tags

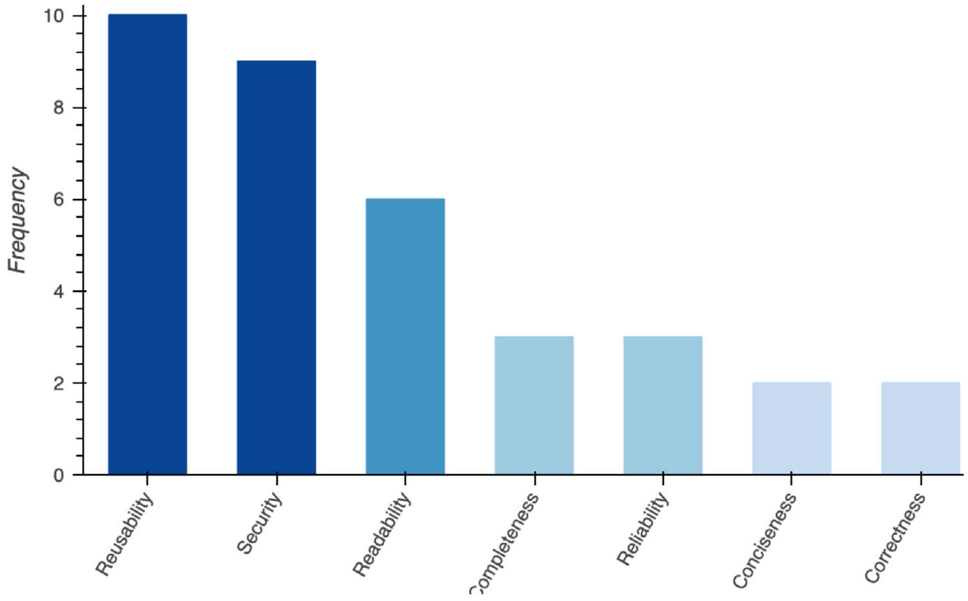

Fig. 7. Frequency of characteristics used by researchers' to measure Stack Overflow code snippet quality.

RQ2c. *What are the quality characteristics used to investigate the quality of code snippets?* Fig. 7 shows the 7 quality characteristics that were used by more than one paper to study the quality of code snippets. *Reusability*, *Security*, and *Readability* emerged as the top three. These were followed by *Completeness* and *Reliability* in the middle tier, and lastly, *Conciseness* and *Correctness* as the bottom two. Other quality characteristics such as *Complexity*, *Comprehensibility*, *Performance*, *Size*, and *Understandability* were also used by not more than one paper.

Bi et al. (2021) employed architecture tactics to address quality attributes of code snippets following the ISO 25010 standard. However, our study shows that the above-listed attributes are those that researchers consider important when judging Stack Overflow code quality. Consequently, several studies have investigated *Reusability* (Ahmad and Cinnéide, 2019; AlOmar et al., 2020; Verdi et al., 2020), *Security* (Meng et al., 2018; Zhang et al., 2021), *Readability* (Campos et al., 2019; Meldrum et al., 2020a), Completeness (Tavakoli et al., 2020; Subramanian and Holmes, 2013), and *Reliability* (Zhang et al., 2018; Abdalkareem et al., 2017). For example, Ahmad and Cinnéide (2019) investigated reusability of code snippet quality by studying how the quality of the program evolves over the time of the project. Meng et al. (2018) pointed out that metadata from Stack Overflow data, like accepted answers, responders' reputation scores, and high vote counts, can further mislead developers to take insecure advice from wrongly promoted posts with vulnerable code because they are upvoted and highly viewed. Hence, they stressed the need to research the security of code snippets to reduce the impact of producing low-quality products with security holes. In comparison, Campos et al. (2019) identified some *Readability* violations in code snippets, including indentation, variable related issues, coding best practices, and irregular-whitespace violations. Tavakoli et al. (2016) identified *Reliability* as a significant metric in judging code snippet quality. They investigated how APIs are misused in code snippets, including missing control constructs and missing or incorrect order of API calls. Furthermore, Tavakoli et al. (2020) investigated the need for code snippet quality improvement in *Completeness* and other characteristics. The evidence here extends the conclusion drawn from the first phase of our study, where we can see that software quality shifts given the need for newer development contexts and context-specific areas.

RQ2d. *What are the relationships between the quality characteristics used to determine the quality of code snippets?* This question aimed to understand the relationships between primary studies on code snippet quality and the various software quality characteristics used to investigate code snippet quality. The network diagram shown in Fig. 8 represents the various characteristics used to measure the quality of Stack Overflow code snippets from the papers reviewed in this study. The nodes are the review papers and quality characteristics, and the edges are the link between a paper and the various code quality characteristics. The nodes are grouped into various sizes and colours. The dark blue nodes are the smallest, and they represent the papers on code quality, while the yellow, light green and green bolder nodes represent the code snippet quality characteristics studied by the various papers. The sizes and colours of the quality characteristics nodes are proportional to the degree of similarity. The light green nodes are the smallest and represent the quality characteristics with the lowest degree of similarity, followed by the green nodes, and lastly, the yellow nodes represent the quality characteristics with the highest degree of similarity.

We adjusted all node degrees slightly to maintain clarity in the representation of the network graph. For instance, we adjusted the nodes that represent the various papers to the same degree (5°) to automatically assign them the same size and colour. On the other hand, we slightly increased the degrees of the characteristics nodes so that the nodes with small degrees were still visible. However, we only reported the actual degrees of the characteristics nodes in this study.

As indicated in Fig. 8, *Security* has the highest degree of similarity (11°), followed by *Reusability* (10°), then *Readability* (6°), *Reliability* and *Completeness* (3°), *Conciseness*, and *Correctness* (2°), *Complexity*, *Performance*, *Size*, and *Understandability* (1°). Hence, we can see that *Security*, *Reusability*, and *Readability* emerged as the most notable characteristics used by researchers to evaluate the quality of Stack Overflow code snippets (refer to the replication package in Section 3.4 for a detailed diagram with all the characteristics).

Ahmad et al. (2018), performed a study to identify software quality requirements discussion topics contributed by *iOS* developers who use Stack Overflow. Findings revealed that *iOS* developers focus mostly on *Usability*, *Reliability*, and *Functionality*, and are comparatively less focused on *Efficiency* and *Portability*,

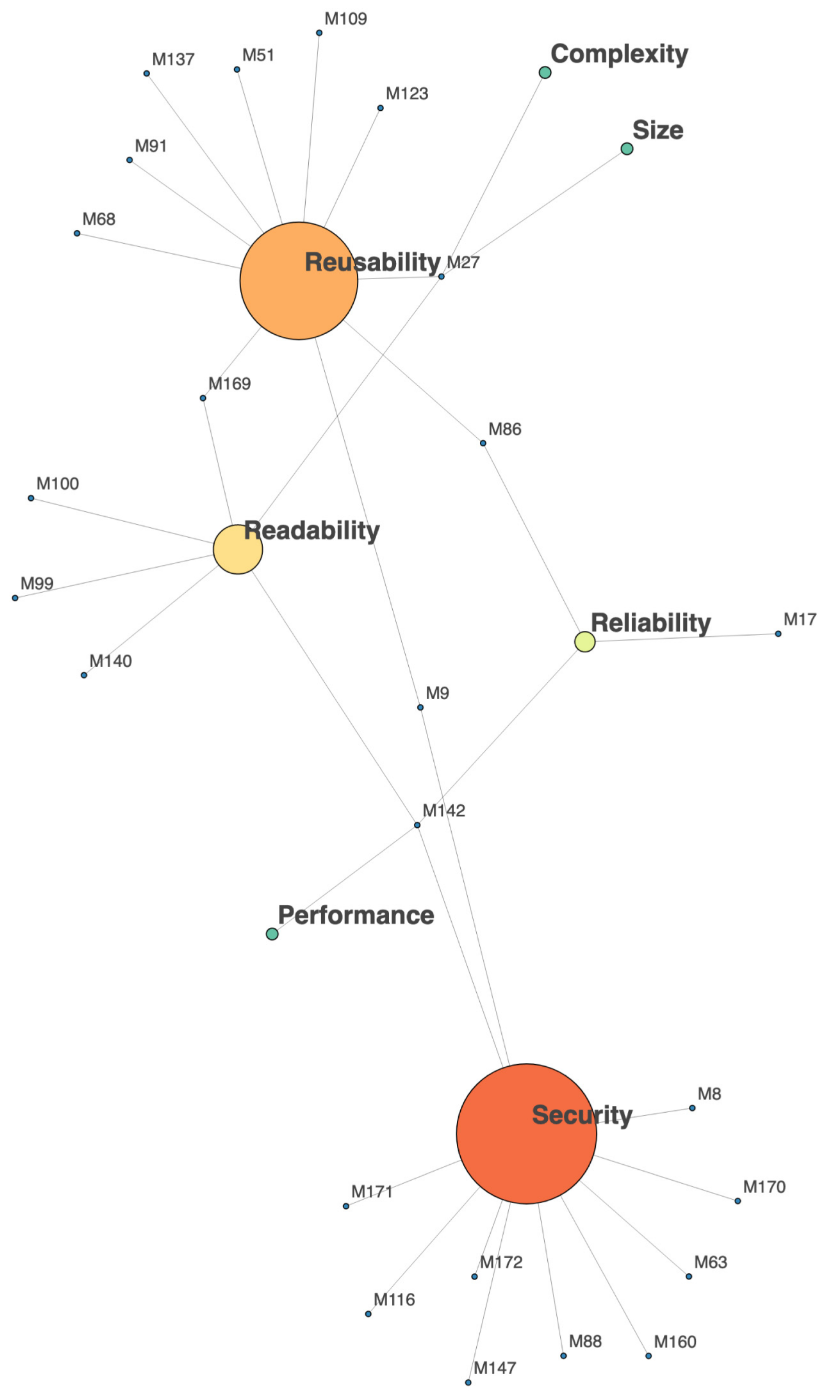

Fig. 8. The network diagram shows that academics focused more on the *Security*, *Reusability* and *Readability* attributes of code quality; (***Mxy** nodes indicate the review papers*).

while they almost neglect *Maintainability*. Similarly, Uddin et al. (2020) developed a mining tool that mined *API* usage scenarios from Stack Overflow. The result showed that *Usability*, *Performance* and *Security* are some of the most reviewed aspects of Stack Overflow. Our study shows that academics generally focus more on *Security*, *Readability*, and *Reusability* and less on the *Reliability* of code snippets. We did not find evidence that *Functionality*, *Usability* and *Portability* have been studied, perhaps partly because most of the quality characteristics are difficult to measure, which is in line with the study performed by Berander et al. (2005). Additionally, it is challenging to measure the *Functionality*, *Usability* or *Maintainability* of code snippets, especially when they are mostly fragments of code that are not complete enough to solve major programming problems, as would be typical in full-fledged applications (Tavakoli et al., 2016).

We can also see that there is no one model suitable for everything. Different types of developers at different stages of the software development process or using different platforms may need to emphasise specific code quality characteristics over others (Ahmad et al., 2018; Uddin et al., 2020). For instance, *Web* developers may consider different code quality attributes compared to *IoT* developers. This emphasis may extend to novice developers vs experienced developers, or individual developers vs developers working for an organisation. Hence, it is clear that the focus on software quality attributes may vary depending on the specific domain. That said, evidence presented thus far relates

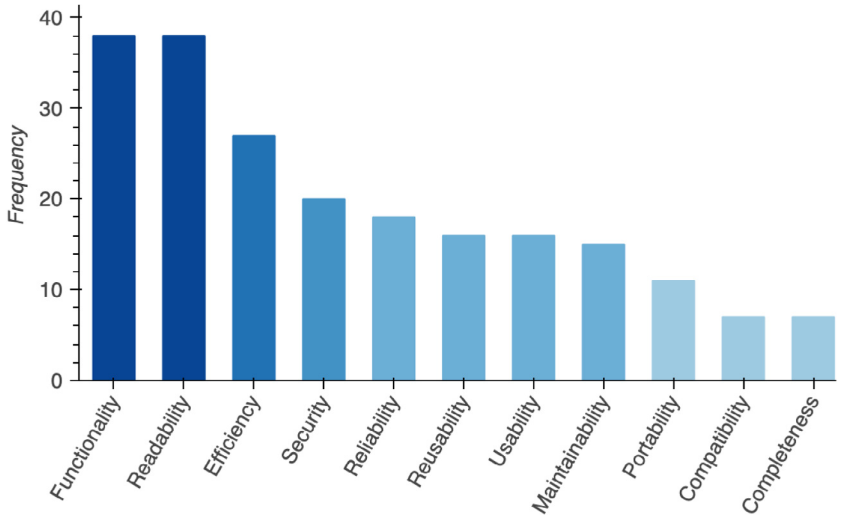

Fig. 9. Frequency of features users consider important for judging the quality of CQA code snippets.

largely to studies that were considered by researchers. How does the practitioner community assess this issue? We answer this question next.

4.3. Third phase — how have views on software quality differed between research and practice?

***RQ3a**. How do software practitioners judge the quality of code snippets?* This question aimed to get the actual practitioners' viewpoint because they are the ones who use the code snippets in practice. Fig. 9 shows the frequencies of the relevant software quality characteristics the software practitioners who participated in this study consider important when judging CQA code snippets (refer to the replication package in Section 3.4 for a detailed breakdown of participants' views on code snippet quality). These quality characteristics include *Functionality* and *Readability* (37 practitioners), *Efficiency* (27 practitioners), *Security* (20 practitioners), *Reliability* (18 practitioners), *Reusability* (16 practitioners), *Maintainability* and *Usability* (15 practitioners), *Portability* (11 practitioners), *Compatibility* (7 practitioners), and *Completeness* (6 practitioners).

The Spearman's rho correlation test we conducted using an alpha value of 0.05 showed no relationship between the top four quality characteristics participants considered important when judging code quality and the programming languages they popularly used for software development. For example, we observed a minimal negative correlation when these quality characteristics were compared with the participants who used Java programming language, ($rs(50) = -0.07$, $p > 0.05$). Similarly, minimal negative/positive correlations were also observed in the other popular programming languages participants used, as follows, Python: ($rs(50) = 0.07$, $p > 0.05$), C++: ($rs(50) = 0.04$, $p > 0.05$), C: ($rs(50) = -0.11$, $p > 0.05$), JavaScript: ($rs(50) = 0.08$, $p > 0.05$), (refer to the replication package in Section 3.4 for details).

RQ3b. What are the differences and similarities in quality dimensions between code snippet quality and software quality? This question aimed to discuss the meaning of quality dimensions of code snippets compared to those of software quality generally. At the same time, we compare the differences and similarities between the practitioners' and researchers' views on each code quality dimension. The following subsections have synthesised the various quality dimensions.

4.3.1. Functionality

In this study, thirty-three practitioners perceive the *Functionality* of code snippets as working code snippets that can solve their particular coding problems. For instance, one participant expressed: "…that the code snippet solves the problem […] the particular part of the code can achieve the expected outcome." (Practitioner GN10CQn4). Another participant further expressed: "…code snippets to produce the right solution." (Practitioner GN12CQn5).

The *ISO 25010* model defines *Functionality* as the degree to which a product or system functions meet stated and implied needs when used under specified conditions. The *ISO 25010* model is a revision of the *ISO 9126* model with minor changes and maintains the same definitions and structure. However, it offers eight characteristics: the same six characteristics of *ISO 9126*, plus *Security* and *Interoperability* (also referred to as *Compatibility*), which were eliminated from the Functionality subcharacteristics, defined as follows:

- *Functional Completeness* — the degree to which the set of functions covers all the specified tasks and user objectives.
- *Functional Correctness* — the degree to which a product (or code snippet) provides the correct results with the needed degree of precision.
- *Functional Appropriateness* — the degree to which the functions facilitate the accomplishment of a specified task.

The software practitioners in this study explicitly appeal to *Functionality*, with a primary focus on *Functional Correctness* in code snippets performing the tasks intended by users. Fitzgerald and Stol (2014), used the term "service" in software architecture to refer to *Functionality*, or a set of software functionalities to be useable by different consumers for different purposes. Similarly, comments from 6 other participants suggested that useable code snippets should have a separation of concerns, where each function concentrates on performing a specific task, "…should be separated and deliver that part a client needs […] one part only deals with memory management, another user interface, while another networking. (Practitioner GN40BQn4)." Therefore, software practitioners that participated in this study believed that *Functional Appropriateness* also applied to code snippets. However, *Functional Completeness* did not surface from any of the software practitioners. The reason could partly be because code snippets are naturally incomplete, and it is difficult for a code snippet to cover a major specified task or user objective. Also, when applied to code snippets, *Functionality* is relative because a code snippet may solve one user's problem for a particular case but may fail to solve another user's problem for another particular case. Hence, it becomes difficult to objectively measure the *Functionality* of code snippets.

4.3.2. Readability

In this study, fifteen participants recounted that *Readability* is about understanding the code snippets. They mainly emphasised that the code snippets' elegance helps the user quickly understand what the snippet is doing and enables an easy explanation of code logic when relating snippets to other team members. For instance, one of the participants expressed: " Can I understand what the snippet is doing by just reading the code? [...] avoid solutions that use convoluted libraries" (Practitioner GN48AQn4). Furthermore, nine other practitioners expressed that short codes are desired to enhance the *Readability* quality of code snippets. However, they also pointed out that compact code snippets may decrease *Readability*. "I might want the code snippet to be quite compact, which may not be quite readable or vice versa." (Practitioner GN12EQn5). Another practitioner gave an illustration using Python list comprehension and method chaining: "...list comprehension may be shorter and quite readable compared to loops [...] method chaining loses *Readability* after a number of method calls." (Practitioner GN25DQn4).

While ten practitioners expressed this viewpoint that comments are an essential feature that a good code snippet should possess, they also argued that overly commented snippets might reduce *Readability*. For instance, one practitioner advised that code snippets should be reasonably commented, as expressed: " ...comments in code snippets should be sensible and not over commented." (Practitioner GN23Qn7). Lastly, three practitioners associated *Readability* with *Maintainability*, as one of the practitioners expressed: "...*Readability* could also be expressed as a form of *Maintainability* or vice versa." (Practitioner GN12Qn7).

Readability is a quality attribute that influences how easily a given piece of code can be read and understood by humans (Alawad et al., 2019; Posnett et al., 2011). Even though *Readability* has never surfaced as any of the characteristics or sub-characteristics in the software quality models, it is a critical factor for code snippet quality that is highly recommended to make program code readable during software development (Elshoff and Marcotty, 1982; Meldrum et al., 2020a). According to Buse and Weimer (2008), everyone who has written code has an intuitive understanding of *Readability*, which includes code indentation, identifier naming, comments, and documentation.

In addition to the line of thought of the practitioners in this study, some studies have also associated *Maintainability* with *Readability* (Aggarwal et al., 2002; Lu et al., 2016). According to Aggarwal et al. (2002), source code *Readability* and *Documentation Readability* are critical and related to the *Maintainability* of a project. Similarly, Lu et al. (2016) believe that *Maintainability* and *Readability* are related and will primarily affect the quality of a code. Apart from associating *Maintainability* with *Readability*, some studies on code snippet quality used *Understandability* (Tavakoli et al., 2020) and *Comprehensibility* (Tavakoli et al., 2016) interchangeably to represent *Readability*.

Readability is an important quality characteristic that is widely researched in the context of code snippet quality, as it is needed to improve the quality of online code snippets (Bafatakis et al., 2019; Duijn et al., 2015; Meldrum et al., 2020a). Meldrum et al. (2020a) measured *Readability* by checking if code snippets meet coding conventions and whether they are properly explained in comments. They believed that quality code snippets should follow standard readability conventions according to the programming language to ensure the code can be easily understood and maintained in the future. Therefore, to ensure quality code snippets and avoid confusing software developers and practitioners, it is important to have concise and understandable code snippets that are readable.

4.3.3. Efficiency

In this study, eleven practitioners perceive good quality code snippets as fast executing code snippets. For instance, one participant narrated, "I tend to look at code snippets that would run fast because I do mostly real-time computation, which requires high-speed computation, and computation speed is important". (Practitioner GN7DQn4). Another practitioner went further to express how he/she would investigate code snippets' *Efficiency*: "...I read the comments to know what is said about the snippet [...] performs better, more optimised or execution time is faster compared to the others." (Practitioner GN2Qn5).

In addition to execution time, ten practitioners also referred to compact code as more efficient code. For instance, one participant explained: "...it may be a case that being compact is better than being verbose or code snippets have unnecessarily extra variables that may hamper performance." (Practitioner GN12DQn5). Another participant further expressed: "...more compact codes are also more efficient codes depending on the particular programming language." (Practitioner GN10DQn4).

Considering the thoughts expressed by several practitioners in this study, *Efficiency* was seen as a time behaviour or the degree to which the processing time and throughput rates of a code snippet will perform its function.*ISO 9126* defined *Efficiency* as the ability of the software product to provide appropriate performance relative to the resources used under stated conditions. *Efficiency* is also interpreted in the *ISO 9126* as the capacity of a software product to use the appropriate amounts and types of resources to perform their functionalities while adhering to efficiency standards and conventions. Additionally, we only found one study investigating the *Efficiency* of code snippets (e.g., Meldrum et al., 2020a). They referred to *Performance* under *Efficiency* or used the terms interchangeably. Their study also recommended that the performance or efficiency of code snippets should be considered in proposed solutions. Therefore, quality code snippets should be served in a way that saves processing time or reduces the number of processing steps.

4.3.4. Security

In this study, four practitioners noted that they try to avoid code snippets that may introduce security holes that can affect the integrity of their outcomes. For example, one participant said: "I am mindful of copying and pasting code snippets I do not fully understand because that could be a risk [...] introducing some security issues or vulnerabilities to what I am building." (Practitioner GN42DQn4). Eleven other practitioners expressed caution in copying code snippets to ensure no injection vulnerabilities are embedded in them. "...copying and pasting a large block of code snippets can introduce some security threats." (Practitioner GN21DQn4). Another participant expressed: "...never copy a regular expression directly without understanding what it does and testing it is safe to use." (Practitioner GN16CQn4).

ISO 25010, a standard for software product quality, introduced security as one of the main software product quality characteristics. The *Security* characteristic is defined as "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 authorisation." It comprises five sub characteristics: *Confidentiality*, *Integrity*, *Non-repudiation*, *Accountability*, and *Authenticity* (Xu et al., 2013).

Software practitioners' view of *Security* in relation to code snippets relates to the integrity of code snippets. As mentioned above, *Integrity* is one of the sub-characteristics under the *Security* quality characteristics of *ISO 25010*, and is defined as the degree to which an application prevents unauthorised access to or modification of computer programs or data. Researchers who have studied code snippets have also investigated code snippets along the line of *Integrity* (Rahman et al., 2019a; Ye et al., 2018; Zhang et al., 2021). For instance, Rahman et al. (2019a) investigated insecure code blocks posted on Stack Overflow. Findings show that 7.1% of the Stack Overflow code fragments examined contained at least one of the six insecure coding practices: code injection, cross-site scripting, insecure cypher, insecure communications, race conditions, and insecure data serialisation. Licorish and Nishatharan (2021) also duplicated these findings more recently. Therefore, software practitioners must be aware of possible security risks in code snippets and carefully use them for software development to avoid introducing security holes during the software development process.

4.3.5. *Reliability*

In this study, 10 participants associated *Reliability* with code snippets "…working well for all the cases …" (User GN21CQn4) and properly implemented error-free solutions, "…error handling should be well defined and graceful to prevent final products from crashing" (User GN31CQn4). Several software *Reliability* models have emerged as people attempt to quantify software *Reliability* and understand the characteristics of how and why software fails (Quyoum et al., 2010; Sahu and Srivastava, 2019). For instance, Sahu and Srivastava (2019) defined Attacks (e.g., worms, viruses, threats) and Failures (e.g., errors, faults and defects) as the two effects of unreliable software. Similarly, researchers who attempt to qualify the *Reliability* of code snippets have also associated it with error-free snippets without defects or violations (e.g., Zhang et al. 2018 and Meldrum et al., 2020a). However, these views of *Reliability* are far from the definition of *Reliability* in the literature (Gokhale and Trivedi, 1999; Smidts et al., 1998). For instance, the *ISO 25010* model defines *Reliability* as the degree of performance for a system under specified conditions such as time and cost. It is composed of the following sub-characteristics: *Maturity*, *Availability*, *Fault tolerance*, and *Recoverability*.

According to Sahu and Srivastava (2019), quantifying *Reliability* through detecting errors, faults and bugs as early as possible during the software development process is important to building a reliable system. Some participants believe, "…Stack Overflow should provide a way to measure the *Reliability* of code snippets in posts …" (Practitioner GN7Qn8). This idea of detecting the *Reliability* of code snippets situated on online platforms could create enough awareness for the software practitioners who rely on code snippets during the software development stage. Therefore, if modern-day software developers depend heavily on online code snippets, they must be reliable to build quality software systems.

4.3.6. *Reusability*

In this study, ten practitioners believed that quality code snippets regarding *Reusability* are code snippets that can be reused or easily remodelled to fit their purpose. For instance, one of the participants recounted, "…it is nice to have a self-contained function that I can reuse in my project, rather than something I will copy and paste all over the place" (Practitioner GN18EQn4). Basili and Rombach (1988) expanded the definition of *Reusability* to include "use of everything associated with a software project, including knowledge." The notion of *Reusability* has been around for a very long time, since humans began to solve problems (Poulin, 1994). The solution to a solved problem can be applied to other similar new problems, and sometimes only a part of that solution is adapted to fit the new problem. Proven solutions, used repeatedly to solve the same problem, become accepted, generalised and standardised (Prieto-Diaz, 1993).

Reusability is a sub-characteristic under Maintainability in the *ISO 25010* model. They defined *Reusability* as the degree to which an asset can be used in more than one system or in building other assets. The practitioners who participated in this study also held this same view of *Reusability* regarding code snippets. Similarly, some researchers have studied the *Reusability* of code snippets along this same line (e.g., Digkas et al. (2019) studied the effect of snippet code reuse on the quality of open-source projects on Github, and Yang et al. (2017) investigated how programmers used code snippets in their real projects). However, other researchers have also viewed *Reusability* as compilable code snippets. For instance, Yang et al. (2016) rationalised quality code snippets and *Reusability* as compilable and runnable snippets. Reid et al. (2020) shared similar views; their study showed that about 70% of the code snippets were not reusable because they were not compilable. The *Reusability* of code snippets should increase the opportunity for error discovery and enhance the quality of software globally.

4.3.7. *Maintainability*

In this study, seven practitioners focused on the future evolution of a code snippet when searching for one. For instance, one practitioner recounted: "…using a working deprecated function is not maintainable" (Practitioner GN46BQn4). Six other practitioners have associated *Maintainability* with other attributes such as *Usability*, *Reusability*, *Reliability*, and *Completeness*, as follows: "…*Reusability* is tied to *Maintainability* […] slightly changing code snippets for a new purpose" (Usability GN47CQn4); " […] how easy it is to adapt […] *Maintainability* is also closely related to *Reliability*" (User GN11BQn4).

Maintainability is commonly perceived as a quality attribute defined as the effort needed to make specified modifications to a component implementation (Broy et al., 2006). *ISO 25010* defined *Maintainability* as the degree of effectiveness and efficiency with which a product or system can be modified to improve, correct, or adapt to changes in environment and requirements. It consists of the following sub-characteristics: *Modularity*, *Reusability*, *Analysability*, *Modifiability*, and *Testability*. Practitioners in this study discussed *Maintainability* in light of *Modifiability* — the degree to which a product or system can be effectively and efficiently modified with a code snippet without introducing defects or degrading existing product quality. However, we did not find any study on code snippet quality that investigated the *Maintainability* of code snippets, even though it can also be argued that *Readability* also falls under *Maintainability* or vice versa.

4.3.8. *Usability*

Five participants referred to *Usability* as an important attribute for judging code snippets. The usability perspective is one of the most frequently desired characteristics of software quality models (Xenos, 2001). Different definitions of *Usability* have been used, making it a potentially confusing concept for software developers (Bevan and Azuma, 1997). According to Seffah and Metzker (2004), the standardisation organisations, software industry, and researchers, have not consistently defined the *Usability* software quality characteristic. In three different standards, the term *Usability* has been defined differently:

- "The capability of the software product to be understood, learned, used, and attractive to the user, when used under specified conditions in *ISO 9126*."
- "The extent to which specified users can use a product to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use" (Stewart, 1998).
- "The ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component" (Schneidewind, 1997).

Considering all the possible definitions, it is challenging to specify the measurable *Usability* characteristics and their interpretations from different perspectives. For example, one participant expressed, "…I want the snippet to be easy to use […] if I have to replace some of the variables with mine, I will want it to be obvious to me the ones I will need to replace" (Practitioner GN19DQn4). The Usability of code snippets is essential for software practitioners because it facilitates good performance and high productivity. Practitioners believe that a code snippet that features good *Usability* will allow the user to perform the expected tasks more efficiently.

However, when we compare the definitions of *Usability* above, the practitioners' view of *Usability* is closer to that of Stewart (1998). This difference in the*ISO 9126* definition of *Usability* may have resulted from the overlapping meanings and interpretations of *Usability* (Abran et al., 2003). Additionally, Abran et al. (2003) pointed out some weaknesses such as ambiguity and unclear architecture in the *Usability* definition of *ISO 9126* that are challenging for software practitioners to grasp clearly. This ambiguity could have also contributed to why researchers on code snippet quality have not investigated Usability. Hence, a more comprehensive and integrated model must be developed to reach a greater consensus.

4.3.9. Portability

In this study, twelve practitioners consider some *Portability* features when collecting code snippets. For example, one participant recounted, "…to know what particular code snippet version will work in the environment I am using." (Practitioner GN2BQn4). Another participant further expressed: "…having a seamless solution across platforms. (Practitioner GN6CQn4)." According to Lenhard and Wirtz (2013), *Portability* in software engineering can be defined as moving software among different runtime platforms without rewriting it partly or wholly. The *ISO 25010* also defined *Portability* as 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. It is composed of *Adaptability*, *Installability* and *Replaceability*. Practitioners in this study perceived the *Portability* of code snippets in light of *Adaptability* — the degree to which a code snippet can effectively and efficiently be adapted for different or evolving hardware, software or other operational or usage environments. *Portability* is critical because of the variety of common runtime platforms and the requirement for developing genuinely agile and flexible systems that do not lock in their users (Mooney, 1995). However, we did not find any study on code snippet quality investigating this area. It will be interesting to see how researchers view the Portability of code snippets in future work.

4.3.10. Compatibility

In this study, five participants perceived *Compatibility* as the ability of code snippets to agree with their current programming environment or operating system. For instance, one practitioner noted, "…to fit with current work" (Practitioner GN43CQn4). Another participant further expressed: "…code is compatible with what I already have running. (Practitioner GN11GQn4)." ISO 25010 defines *Compatibility* as 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. This characteristic is composed of the following two sub-characteristics: *Co-existence* and *Interoperability*. The software practitioners' view of the *Compatibility* of code snippets deviates from this definition and aligns more with *Portability*. These conflicting views show that the differences may be subtle; however, they could impact researchers' interest and possibly have contributed to why no studies were found on *Compatibility* in the literature on code snippet quality.

4.3.11. Completeness

In this study, five practitioners considered *Completeness* when collecting code snippets from CQA portals as relevant to code snippet quality. One of the practitioners recounted, "…handle the various use cases and production-ready" (Practitioner GN31 BQn4). Another said, "We look for relatively complete snippets to focus attention on other tasks like testing, approvals, integration for fast product delivery" (Practitioner GN11FQn4). *Completeness* is defined as "the state or condition of having all the necessary or appropriate parts" (Hadad et al., 2015). Additionally, requirements *Completeness* is defined as "a quality demanded to the set of software requirements and to each requirement itself, to ensure that there is no information left aside" (Packard, 2018). *Functional completeness* — the degree to which the set of functions covers all the specified tasks and user objectives. It is a sub-characteristics of *Functionality* in the *ISO 25010* model. The above definitions of *Completeness* may suit the practitioners' view of *Completeness* to a certain degree. However, researchers of code snippet quality have argued that it is challenging to define and measure the *Completeness* of code snippets because code snippets are incomplete by nature. For instance, Tavakoli et al. (2020) showed that more than half of the code snippets they investigated were incomplete. Similarly, Subramanian and Holmes (2013) showed that 82% of code snippets investigated were incomplete. Therefore, collaborative community platforms like Stack Overflow may consider checking the syntax of submitted code snippets to ensure they have enough information to the extent they can at least compile.

4.3.12. Others

The other characteristics participants mentioned and considered important when judging code snippet quality include *Simplicity*, *Community Acceptance*, *Code currency*, *Testability*, *Feeling*, and *Scalability* (refer to the replication package in Section 3.4 for a detailed breakdown of the other software quality attributes participants considered when judging code snippets.) Practitioners measure the simplicity of code snippets in terms of the code complexity (Munson and Khoshgoftaar, 1992; Tashtoush et al., 2014); such as how short the code snippets are in terms of lines of code. They also referred to simple code snippets as clear, clean, elegant, and straightforward, that aid easy understanding and integration into their projects. Participants usually associate complex code snippets with unnecessary libraries that introduce more dependencies.

The community acceptance of a code snippet is established by the users' reputation and the authority of the code author (Chen et al., 2019a; Nasehi et al., 2012). Practitioners' are more confident in the answers posted by high reputation users. They believe that these posts are always well informed and worth reading, copying or modifying to suit their use cases. It is important to note that even though participants refer to code snippets that have been accepted and upvoted with high scores as high quality, this may not necessarily be the case. Some participants argued that the upvoted answer could be slightly different from what they are searching for because their use case may vary slightly. Hence, they sometimes drill down the answer thread for alternative solutions and read the discussions around the answers, especially to find out people's detailed views.

In this information era where things move quickly (Porter and Heppelmann, 2014), the currency of code snippets is a significant factor for many practitioners. Code snippets that may have been popular in the past may become outdated today. Practitioners want current answers rather than answers that may have applied to older software. For instance, the library version of a code snippet solution used may influence the choice of the user selection of an answer. Therefore, CQA portals should tag obsolete code snippets or those that do not adhere to current coding standards.

Furthermore, some participants acknowledged that Testability is an important quality they want to validate when collecting code snippets from CQA portals. They want to isolate the copied snippets and run automated testing for potential faults before integrating them into their current project. A few practitioners used the term "trial and error" to refer to Testability. They view Testability as a gamble, try the code snippet to see if it fits; otherwise, try another code snippet until they find one that fits their use case best. This quality is somewhat related to "Feeling". However, in the case of Feeling, practitioners have a gut feeling or belief that the snippet is a good solution. Lastly, the few participants that mentioned Scalability as a quality dimension indicated that they consider this dimension when judging code snippets for the flexibility of running the code snippet on a single instance or cluster. They are more concerned about the architecture or how the code snippets are written or modified to perform multiprocessing whenever the need arises.

4.4. Summary discussion of the research phases

In this study, we carried out different research phases to examine researchers' and practitioners' views on code snippet quality. We systematically reviewed secondary studies on software quality in the first phase, establishing a baseline knowledge and understanding of software quality. This phase also revealed evidence that suggested researchers' increasing interest in the newer model standards (e.g., ISO 9126 and ISO 25010). We also observed emerging interest in some context-specific models designed to fit specific needs (e.g., code snippets). Further analysis carried out on the quality characteristics of the software quality models established that the six most commonly referenced quality characteristics showed shifting emphases over time. We are also aware that over the years, software practitioners have concentrated on particular quality characteristics in different eras, which may also account for the shifting emphases. However, we also acknowledge that this calls for investigation in research. The second phase of this study reviewed studies on code snippet quality. The outcome of this review showed that the top six quality characteristics mostly studied by the researchers on code snippet quality differed from the top six studied on software quality characteristics, suggesting that software quality is context-specific (e.g., full-fledged applications vs code snippets or machine learning context vs regular programming context vs Android programming context). To conclude, the insights we derived from the latter study validated the relevance of the quality characteristics researchers commonly examined and established convergence with practitioners' views. We observed that most code snippet quality characteristics the software practitioners considered important were mostly derived from the software quality characteristics. However, the software practitioners' definitions of most of the quality characteristics differed from their original software quality definition. We also observed there were differences in the code snippet quality characteristics often studied by researchers compared to those considered important by the software practitioners when judging code snippet quality. In summary, while the researchers mainly researched *Reusability*, *Security*, and *Readability*, we observed differing views from the software practitioners, who considered *Functionality*, *Readability*, and *Efficiency* the three most important quality characteristics for judging code snippet quality.

5. Threats to validity

We discuss several threats to the validity of our studies according to the guidelines proposed by Kitchenham and Charters (2007) and how these threats were particularly mitigated in our study.

Construct validity. refers to the relationship between theory and practice. An individual author compiled the papers used in this study. Therefore, there might have been cases where the paper's relevance may have been misjudged. However, to mitigate this threat, a second author validated the relevance of the selected papers, thus reducing this threat Petersen et al. (2008).

Furthermore, the quality of search engines can affect the completeness of relevant papers selected for this study. Since our selection is based on a set of search terms, we possibly missed some other studies where authors used other terms to represent a quality characteristic of software quality and code snippet quality, as pointed out by Wohlin et al. (2013). That said, to mitigate the likelihood of this threat materialising, searches were conducted on the respective databases using broad search strings (e.g., *Systematic Review*, *Comparative Study*, and *Systematic Mapping*) in the first phase and (e.g., *Stack Overflow* and *StackOverflow*) in the second phase.

Additionally, we may have missed some relevant papers from other sources in line with the recommended databases used in this study. Hence, we complemented the search with backward snowball sampling of all studies after reading the full text of each paper (Jalali and Wohlin, 2012). Consequently, the addition of two and four new papers in this study's first and second phases resulting from the backward snowballing process suggests that we were likely to have surveyed an exhaustive list of studies rather than missed pertinent ones.

Researcher bias during the data extraction and classification phase is also a threat. Kitchenham et al. (2009) and Brereton et al. (2007) found it useful to have one researcher extract the data and another review the extraction. Pilot evaluations were carried out to ensure that the first two authors shared the same understanding, increasing the likelihood of a reliable systematic review and quality evaluation. However, since this step involves human judgement, the threat cannot be completely eliminated.

Furthermore, the deductive approach we used in the third phase to collect the interview data on how the participants' viewed or judged code snippets may present challenges to the results we reported (Potter and Levine-Donnerstein, 1999). Using theory has some inherent limitations, in that researchers approach the data with an informed but, nonetheless, strong bias. Hence, we may be more likely to find supportive rather than nonsupportive evidence of our predetermined codes used to interview the participants on how they would judge code quality (Hsieh and Shannon, 2005; Potter and Levine-Donnerstein, 1999). Second, in answering the probe questions, some participants may receive cues to answer in a certain way or agree with the questions to please researchers (Hsieh and Shannon, 2005). However, during the interview section, we tried to suppress this bias by asking the participant to mention other qualities they perceived as important to the judgement of code quality that we failed to mention (see Section 4.3.12 for the several attributes participants consider important when judging code snippets). There is also a potential threat that the transcripts may be misunderstood or incompletely reported as we based our analysis and report solely on the interview data. This process may affect the totality of the interpretation. As a control action to demonstrate internal consistency and credibility, we performed member checking, where the participants validated that their transcripts were correctly and not incompletely reported or misunderstood.

Internal validity. It was noted that some studies use different terminologies to represent the same quality characteristics. In fact, we observed that some studies used *Operability* as one of the quality characteristics of the ISO 9126, while others used *Usability* interchangeably. Indeed, this can lead to a degree of ambiguity in the results. One way to mitigate this challenge will be to redefine the conflicting quality characteristics (e.g., renaming *Operability* to *Usability* or vice versa). However, we did not want to tamper with the outcome of our results. Hence, we presented the results as were reported from our selected studies (e.g., studies that used *Operability* were not combined with studies that used *Usability*.)

External validity. Stack Overflow and other Q&A communities are widely used by practitioners and software professionals worldwide (Meldrum et al., 2017). However, in this study, we only interviewed participants within New Zealand. Additionally, the sampling strategy used in this study led us to practitioners that regularly used knowledge from online portals during software development. However, some respondents had an academic background, notwithstanding their linkages to the industry and many years of experience in major software development companies. While this detail was self-reported, such respondents may not use Stack Overflow and other Q&A communities as frequent and exhaustive as active developers in the software development industry, albeit, we screened respondents to make sure they were active users on these platform. Consequently, these factors may be a potential threat to the external validity of the results and findings of this study (Kitchenham et al., 2011; Marshall, 1996). Accordingly, we hope to perform future work to enhance the external validity of our research.

Conclusion validity. We believe our interview results provide a comprehensive framework for practitioners who rely on online code snippets and consider their quality an important feature or want to improve the quality of coding practice as their business strategy. However, we could still argue that the findings from the interview with the 50 participants may not be generalised to the wider views on the quality of Stack Overflow code snippets because participants were not from all domains (e.g., mobile, web, and standalone application). Hence, the outcomes of software participants focused on mobile applications may be different in terms of their judgement of code quality compared to those focused on standalone or web applications. In future research, we hope to investigate software practitioners who use code snippets when developing for specific domains (e.g., mobile vs web, standalone application) and compare quality characteristics to see how they may differ.

Finally, our work considered Stack Overflow because it is the most widely used by many studies related to code snippet quality. However, we admit that Stack Overflow does not have exclusivity of code snippets. Therefore, the conclusion of this study may be biased by limiting the study to only one case. Accordingly, more studies from diverse communities are needed to increase the generalisability of this study.

6. Summary and implications

This study explored researchers' and practitioners' views on software quality and the quality of code snippets. First, we systematically reviewed secondary studies on software quality to summarise researchers' views on software quality. Review papers on software quality were identified, and eleven papers were selected and used for this study. The outcome of this phase showed academics' views on software quality models. We noticed differences in the six quality characteristics over the last 40 years. We believe that these differences may have resulted from the top priority concerns software practitioners have faced within particular periods. However, further research is needed to investigate why there are shifting differences in quality characteristics of software quality models over time. We also noticed that more recent models like the *ISO 9126* and *ISO 25010* gained popularity, relevance and importance 2000 to 2010, even though they started to decline from 2010 to 2020. We speculate that this indicates probable decentralisation in the quality characteristics and shifting from software quality models that have defined quality characteristics to suit all the quality needs to context-specific frameworks that highlight specific quality characteristics to address specific quality needs (e.g., code snippets).However, this also calls for further investigations in future research. Second, a systematic literature review was carried out on primary studies on code quality to understand researchers' perceptions of code snippet quality. Twenty-seven papers were selected and used in this phase. The result showed that the quality characteristics investigated by the researchers on code snippets deviated from the earlier models. For example, the researchers' top-six quality characteristics on code snippet quality are different from the top-six quality characteristics for software quality. This outcome further reveals that perspectives on software quality are context-specific (e.g., full-fledged applications vs code snippets or machine learning context vs regular programming context vs Android programming context). In the third study, the outcomes of rich and detailed knowledge on practitioners' views on how they judge the quality of code snippets were provided. We see there are differences between the quality characteristics often studied by researchers compared with those the software practitioners considered important when judging code snippet quality. For example, the practitioners' perceive Functionality and Efficiency as important quality characteristics for judging code snippets in practice. On the other hand, we did not find any study researching these quality characteristics for code snippets. It is also important to note that the term *Readability* is considered an important quality characteristic specific to the context of code snippets, both in research and practice. However, this term has never emerged as a quality characteristic in any software quality model.

It is surmised that software quality models are not one-size-fits-all and may not be referenced heavily in recent times because of the shift to context-specific models due to changes in the modus operandi of software systems (e.g., moving more towards automation, heavy use of tools, and the emergence of collaboration environments). Consequently, we also hope to explore furthermore code snippets analysis on the various attributes that practitioners consider important when judging code snippet quality, including Functionality, Readability, Efficiency, Security and Reliability.

Based on the comparison of the existing software quality models, outcomes show there are no universally suitable software quality models. As a result, the findings of this study have several implications for research and practice. In terms of research, our work complements earlier studies, and the knowledge provided may serve as a catalogue for the sifting views and evaluation of software quality. From practitioners' perspective, we outline implications as follows:

First phase implications. The outcome of this phase implies that model views have shifted over time based on the practitioners' and users' needs (Colakoglu et al., 2021; Kumar and Gupta, 2017). *ISO 9126* came in the internet era when there was a need for Security and data protection due to the increasing number of security attacks. Consequently, newer models like *ISO 25010* emerged to enhance systems and software security and protection (Peters and Aggrey, 2020). Nevertheless, some of the newer models may not be suitable for measuring software product quality in the future (Miguel et al., 2014; Thapar et al., 2012); hence models will need to be consistently updated.

Second phase implications. The emphasis should shift from developing comprehensive models that include all possible software characteristics to developing models that only include the essential quality characteristics for the domain (Wagner et al., 2015). Hence, building context-specific models should be the focus of model developers. In our context, this study focused on code snippet quality and found that focusing on the essential quality characteristics is important for practitioners, and may also reduce the burden of code snippet evaluation via existing quality assessment models, which are often laborious and time-consuming to implement (Hauge et al., 2009; Stol and Ali Babar, 2010).

Third phase implications. Code snippets provided by highly reputable contributors or highly upvoted counts are generally considered high quality. Practitioners must be cautious not to consider only code snippet posts by highly reputable contributors or those with highly upvoted counts, but to apply their judgements, following some of the code snippet quality characteristics presented in this study. In addition, practitioners should also focus on their understanding of code snippet quality, how poor snippets may affect the entire quality of software products, and how best to use the snippets.

Additionally, what is important for the research community may differ from what is important for practitioners. Software developers have different lenses on quality, hence the need for specific models relevant to a given purpose, instead of a single model that may be considered generalised. Therefore, no one model is suitable for everything; this calls for customised and flexible models. In particular, researchers are challenged to learn about practitioners' views in tailoring suitable solutions that are purposeful and of utility. In the context of code snippet quality, code quality is mostly viewed out of the lens of suitable Functionality, Readability, Efficiency, Security and Reliability. Concrete approaches for ensuring that these dimensions of quality are preserved would be noteworthy for the practitioner community, in ensuring high-quality software.

CRediT authorship contribution statement

Ifeanyi G. Ndukwe: Conceptualization, Structuring, Literature review, Data curation, Methodology, Results, Discussion, Writing – original draft. **Sherlock A. Licorish:** Conceptualization, Structuring, Data curation, Data validation, Literature review, Methodology, Results, Discussion, Writing – review & editing. **Amjed Tahir:** Literature review, Discussion, Writing – review & editing. **Stephen G. MacDonell:** Literature review, Methodology, Discussion, Writing – review & editing.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Ifeanyi Glory Ndukwe reports financial support was provided by The Science for Technological Innovation National Science Challenge (SfTI).

Data availability

The authors do not have permission to share data.

References

Abdalkareem, R., Shihab, E., Rilling, J., 2017. On code reuse from stackoverflow: An exploratory study on android apps. Inf. Softw. Technol. 88, 148–158. http://dx.doi.org/10.1016/j.infsof.2017.04.005.

Abdallah, M., Jaber, T., Alabwaini, N., Abd Alnabi, A., 2019. A proposed quality model for the internet of things systems. In: 2019 IEEE Jordan International Joint Conference on Electrical Engineering and Information Technology. JEEIT, IEEE, pp. 23–27. http://dx.doi.org/10.1109/JEEIT.2019.8717516.

Abran, A., Khelifi, A., Suryn, W., Seffah, A., 2003. Usability meanings and interpretations in ISO standards. Softw. Qual. J. 11 (4), 325–338. http://dx.doi.org/10.1023/A:1025869312943.

Acar, Y., Backes, M., Fahl, S., Kim, D., Mazurek, M.L., Stransky, C., 2016. You get where you're looking for: The impact of information sources on code security. In: 2016 IEEE Symposium on Security and Privacy. SP, IEEE, pp. 289–305. http://dx.doi.org/10.1109/SP.2016.25.

Adaji, I., Vassileva, J., 2016. Towards understanding user participation in stack overflow using profile data. In: International Conference on Social Informatics. Springer, pp. 3–13. http://dx.doi.org/10.1007/978-3-319-47874-6_1.

Adewumi, A., Misra, S., Omoregbe, N., Crawford, B., Soto, R., 2016. A systematic literature review of open source software quality assessment models. SpringerPlus 5 (1), 1–13. http://dx.doi.org/10.1186/s40064-016-3612-4.

Aggarwal, K.K., Singh, Y., Chhabra, J.K., 2002. An integrated measure of software maintainability. In: Annual Reliability and Maintainability Symposium. 2002 Proceedings (Cat. No. 02CH37318). IEEE, pp. 235–241. http://dx.doi.org/10.1109/RAMS.2002.981648.

Ahasanuzzaman, M., Asaduzzaman, M., Roy, C.K., Schneider, K.A., 2018. Classifying stack overflow posts on API issues. In: 2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering. SANER, IEEE, pp. 244–254. http://dx.doi.org/10.1109/SANER.2018.8330213.

Ahmad, M., Cinnéide, M.O., 2019. Impact of stack overflow code snippets on software cohesion: a preliminary study. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories. MSR, IEEE, pp. 250–254. http://dx.doi.org/10.1109/MSR.2019.00050.

Ahmad, A., Li, K., Feng, C., Sun, T., 2018. An empirical study on how iOS developers report quality aspects on stack overflow. Int. J. Mach. Learn. Comput. 8 (5), 501–506. http://dx.doi.org/10.18178/ijmlc.2018.8.5.736.

Al-Badareen, A.B., Selamat, M.H., Jabar, M.A., Din, J., Turaev, S., 2011. Software quality models: A comparative study. In: International Conference on Software Engineering and Computer Systems. Springer, pp. 46–55. http://dx.doi.org/10.1007/978-3-642-22170-5_4.

Al-Qutaish, R.E., 2010. Quality models in software engineering literature: an analytical and comparative study. J. Am. Sci. 6 (3), 166–175.

Alawad, D., Panta, M., Zibran, M., Islam, M.R., 2019. An empirical study of the relationships between code readability and software complexity. http://dx.doi.org/10.48550/arXiv.1909.01760, arXiv preprint arXiv:1909.01760.

Ali, N.B., Petersen, K., 2014. Evaluating strategies for study selection in systematic literature studies. In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 1–4. http://dx.doi.org/10.1145/2652524.2652557.

AlOmar, E.A., Barinas, D., Liu, J., Mkaouer, M.W., Ouni, A., Newman, C., 2020. An exploratory study on how software reuse is discussed in stack overflow. In: International Conference on Software and Software Reuse. Springer, pp. 292–303. http://dx.doi.org/10.1007/978-3-030-64694-3_18.

Arwan, A., Rochimah, S., Akbar, R.J., 2015. Source code retrieval on stackoverflow using lda. In: 2015 3rd International Conference on Information and Communication Technology (ICoICT). IEEE, pp. 295–299. http://dx.doi.org/10.1109/ICoICT.2015.7231439.

Azuma, M., 1996. Software products evaluation system: quality models, metrics and processes—International standards and Japanese practice. Inf. Softw. Technol. 38 (3), 145–154. http://dx.doi.org/10.1016/0950-5849(95)01069-6.

Bacchelli, A., Ponzanelli, L., Lanza, M., 2012. Harnessing stack overflow for the IDE. In: 2012 Third International Workshop on Recommendation Systems for Software Engineering. RSSE, IEEE, pp. 26–30. http://dx.doi.org/10.1109/RSSE.2012.6233404.

Bafatakis, N., Boecker, N., Boon, W., Salazar, M.C., Krinke, J., Oznacar, G., White, R., 2019. Python coding style compliance on stack overflow. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories. MSR, IEEE, pp. 210–214. http://dx.doi.org/10.1109/MSR.2019.00042.

Bai, W., Akgul, O., Mazurek, M.L., 2019. A qualitative investigation of insecure code propagation from online forums. In: 2019 IEEE Cybersecurity Development (SecDev). IEEE, pp. 34–48. http://dx.doi.org/10.1109/SecDev.2019.00016.

Barua, A., Thomas, S.W., Hassan, A.E., 2014. What are developers talking about? an analysis of topics and trends in stack overflow. Empir. Softw. Eng. 19 (3), 619–654. http://dx.doi.org/10.1007/s10664-012-9231-y.

Basili, V.R., Rombach, H.D., 1988. Towards a Comprehensive Framework for Reuse: A Reuse-Enabling Software Evolution Environment. University of Maryland Maryland, USA, 19910001295.

Berander, P., Damm, L.-O., Eriksson, J., Gorschek, T., Henningsson, K., Jönsson, P., Kågström, S., Milicic, D., Mårtensson, F., Rönkkö, K., et al., 2005. Software quality attributes and trade-offs. Blekinge Inst. Technol. 97 (98), 99.

Bertoa, F., Vallecillo, A., 2002. Quality attributes for COTS components. I+D Comput. 1 (2), 128–143.

Bevan, N., Azuma, M., 1997. Quality in use: Incorporating human factors into the software engineering lifecycle. In: Proceedings of IEEE International Symposium on Software Engineering Standards. IEEE, pp. 169–179. http://dx.doi.org/10.1109/SESS.1997.595963.

Beyer, S., Macho, C., Di Penta, M., Pinzger, M., 2018. Automatically classifying posts into question categories on stack overflow. In: 2018 IEEE/ACM 26th International Conference on Program Comprehension. ICPC, IEEE, pp. 211–21110. http://dx.doi.org/10.1145/3196321.3196333.

Bi, T., Liang, P., Tang, A., Xia, X., 2021. Mining architecture tactics and quality attributes knowledge in stack overflow. J. Syst. Softw. 111005. http://dx.doi.org/10.1016/j.jss.2021.111005.

Boehm, B.W., Brown, J.R., Lipow, M., 1978. Quantitative evaluation of software quality. In: Proceedings of the 2nd International Conference on Software Engineering. pp. 592–605.

Brereton, P., Kitchenham, B.A., Budgen, D., Turner, M., Khalil, M., 2007. Lessons from applying the systematic literature review process within the software engineering domain. J. Syst. Softw. 80 (4), 571–583. http://dx.doi.org/10.1016/j.jss.2006.07.009.

Broy, M., Deissenboeck, F., Pizka, M., 2006. Demystifying maintainability. In: Proceedings of the 2006 International Workshop on Software Quality. ACM, pp. 21–26. http://dx.doi.org/10.1145/1137702.1137708.

Buse, R.P., Weimer, W.R., 2008. A metric for software readability. In: Proceedings of the 2008 International Symposium on Software Testing and Analysis. ACM, pp. 121–130. http://dx.doi.org/10.1145/1390630.1390647.

Buse, R.P., Weimer, W.R., 2009. Learning a metric for code readability. IEEE Trans. Softw. Eng. 36 (4), 546–558. http://dx.doi.org/10.1109/TSE.2009.70.

Campos, U.F., Smethurst, G., Moraes, J.P., Bonifácio, R., Pinto, G., 2019. Mining rule violations in javascript code snippets. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories. MSR, IEEE, pp. 195–199. http://dx.doi.org/10.1109/MSR.2019.00039.

Chatterjee, P., Kong, M., Pollock, L., 2020. Finding help with programming errors: An exploratory study of novice software engineers' focus in stack overflow posts. J. Syst. Softw. 159, 110454. http://dx.doi.org/10.1016/j.jss.2019.110454.

Chen, M., Fischer, F., Meng, N., Wang, X., Grossklags, J., 2019a. How reliable is the crowdsourced knowledge of security implementation? In: 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, pp. 536–547. http://dx.doi.org/10.1109/ICSE.2019.00065.

Chen, L., Hou, S., Ye, Y., Bourlai, T., Xu, S., Zhao, L., 2019b. iTrustSO: an intelligent system for automatic detection of insecure code snippets in stack overflow. In: 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining. ASONAM, IEEE, pp. 1097–1104. http://dx.doi.org/10.1145/3341161.3343524.

Cohen, J., 1960. A coefficient of agreement for nominal scales. Educ. Psychol. Measur. 20 (1), 37–46.

Colakoglu, F.N., Yazici, A., Mishra, A., 2021. Software product quality metrics: A systematic mapping study. IEEE Access 9, 44647–44670. http://dx.doi.org/10.1109/ACCESS.2021.3054730.

Digkas, G., Nikolaidis, N., Ampatzoglou, A., Chatzigeorgiou, A., 2019. Reusing code from stackoverflow: the effect on technical debt. In: 2019 45th Euromicro Conference on Software Engineering and Advanced Applications. SEAA, IEEE, pp. 87–91. http://dx.doi.org/10.1109/SEAA.2019.00022.

Dromey, R.G., 1995. A model for software product quality. IEEE Trans. Softw. Eng. 21 (2), 146–162. http://dx.doi.org/10.1109/32.345830.

Duijn, M., Kucera, A., Bacchelli, A., 2015. Quality questions need quality code: Classifying code fragments on stack overflow. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. IEEE, pp. 410–413. http://dx.doi.org/10.1109/MSR.2015.51.

Ellmann, M., Schnecke, M., 2018. Two perspectives on software documentation quality in stack overflow. In: Proceedings of the 4th ACM SIGSOFT International Workshop on NLP for Software Engineering. IEEE, pp. 6–9. http://dx.doi.org/10.1145/3283812.3283816.

Elshoff, J.L., Marcotty, M., 1982. Improving computer program readability to aid modification. Commun. ACM 25 (8), 512–521. http://dx.doi.org/10.1145/358589.358596.

Fischer, F., Böttinger, K., Xiao, H., Stransky, C., Acar, Y., Backes, M., Fahl, S., 2017. Stack overflow considered harmful? the impact of copy&paste on android application security. In: 2017 IEEE Symposium on Security and Privacy. SP, IEEE, pp. 121–136. http://dx.doi.org/10.1109/SP.2017.31.

Fitrisia, Y., Hendradjaya, B., 2014. Implementation of ISO 9126-1 quality model for asset inventory information system by utilizing object oriented metrics. In: 2014 International Conference on Electrical Engineering and Computer Science. ICEECS, IEEE, pp. 229–234. http://dx.doi.org/10.1109/ICEECS.2014.7045252.

Fitzgerald, B., Stol, K.-J., 2014. Continuous software engineering and beyond: trends and challenges. In: Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering. ACM, pp. 1–9. http://dx.doi.org/10.1145/2593812.2593813.

Franke, D., Kowalewski, S., Weise, C., 2012. A mobile software quality model. In: 2012 12th International Conference on Quality Software. IEEE, pp. 154–157. http://dx.doi.org/10.1109/QSIC.2012.49.

Geremia, S., Bavota, G., Oliveto, R., Lanza, M., Di Penta, M., 2019. Characterizing leveraged stack overflow posts. In: 2019 19th International Working Conference on Source Code Analysis and Manipulation. SCAM, IEEE, pp. 141–151. http://dx.doi.org/10.1109/SCAM.2019.00025.

Gokhale, S.S., Trivedi, K.S., 1999. A time/structure based software reliability model. Ann. Softw. Eng. 8 (1), 85–121. http://dx.doi.org/10.1023/A:1018923329647.

Hadad, G.D., Litvak, C.S., Doorn, J.H., Ridao, M., 2015. Dealing with completeness in requirements engineering. In: Encyclopedia of Information Science and Technology, third ed. IGI Global, pp. 2854–2863. http://dx.doi.org/10.4018/978-1-4666-5888-2.ch279.

Hannay, J., Jørgensen, M., 2008. The role of deliberate artificial design elements in software engineering experiments. IEEE Trans. Softw. Eng. 34 (2), 242–259. http://dx.doi.org/10.1109/TSE.2008.13.

Hauge, O., Osterlie, T., Sorensen, C.-F., Gerea, M., 2009. An empirical study on selection of open source software-preliminary results. In: 2009 ICSE Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development. IEEE, pp. 42–47. http://dx.doi.org/10.1109/FLOSS.2009.5071359.

Hendradjaya, B., et al., 2014. The application model of learning management system quality in asynchronous blended learning system. In: 2014 International Conference on Electrical Engineering and Computer Science. ICEECS, IEEE, pp. 223–228. http://dx.doi.org/10.1109/ICEECS.2014.7045251.

Hickey, G., Kipping, C., 1996. A multi-stage approach to the coding of data from open-ended questions. Nurse Res. 4 (1), 81–91. http://dx.doi.org/10.7748/nr.4.1.81.s9.

Hou, S., Saas, A., Chen, L., Ye, Y., 2016a. Deep4maldroid: A deep learning framework for android malware detection based on linux kernel system call graphs. In: 2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops. WIW, IEEE, pp. 104–111. http://dx.doi.org/10.1109/WIW.2016.040.

Hou, S., Saas, A., Ye, Y., Chen, L., 2016b. Droiddelver: An android malware detection system using deep belief network based on api call blocks. In: International Conference on Web-Age Information Management. Springer, pp. 54–66. http://dx.doi.org/10.1007/978-3-319-47121-1_5.

Hsieh, H.-F., Shannon, S.E., 2005. Three approaches to qualitative content analysis. Qual. Health Res. 15 (9), 1277–1288. http://dx.doi.org/10.1177/1049732305276687.

Jalali, S., Wohlin, C., 2012. Systematic literature studies: database searches vs. backward snowballing. In: Proceedings of the 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. IEEE, pp. 29–38. http://dx.doi.org/10.1145/2372251.2372257.

Jamwal, D., 2010. Analysis of software quality models for organizations. Int. J. Latest Trends Comput. 1 (2), 19–23, 2045-5364.

Kim, M., 2016. A quality model for evaluating IoT applications. Int. J. Comput. Electr. Eng. 8 (1), 66. http://dx.doi.org/10.17706/ijcee.2016.8.1.66-76.

Kitchenham, B., Brereton, O.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.

Kitchenham, B.A., Budgen, D., Brereton, O.P., 2011. Using mapping studies as the basis for further research–a participant-observer case study. Inf. Softw. Technol. 53 (6), 638–651. http://dx.doi.org/10.1016/j.infsof.2010.12.011.

Kitchenham, B., Charters, S., 2007. Guidelines for performing systematic literature reviews in software engineering. Technical Report, Software Engineering Group, School of Computer Science and Mathematics, Keele University & Department of Computer Science, University of Durham.

Kitchenham, B., Pretorius, R., Budgen, D., Brereton, O.P., Turner, M., Niazi, M., Linkman, S., 2010. Systematic literature reviews in software engineering–a tertiary study. Inf. Softw. Technol. 52 (8), 792–805. http://dx.doi.org/10.1016/j.infsof.2008.09.009.

Kumar, A., Gupta, D., 2017. Paradigm shift from conventional software quality models to web based quality models. Int. J. Hybrid Intell. Syst. 14 (3), 167–179. http://dx.doi.org/10.3233/HIS-180249.

Lenhard, J., Wirtz, G., 2013. Measuring the portability of executable service-oriented processes. In: 2013 17th IEEE International Enterprise Distributed Object Computing Conference. IEEE, pp. 117–126. http://dx.doi.org/10.1109/EDOC.2013.21.

Liao, H., Hitchcock, J., 2018. Reported credibility techniques in higher education evaluation studies that use qualitative methods: A research synthesis. Eval. Program Plan. 68, 157–165. http://dx.doi.org/10.1016/j.evalprogplan.2018.03.005.

Licorish, S.A., Nishatharan, T., 2021. Contextual profiling of stack overflow java code security vulnerabilities initial insights from a pilot study. In: 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C). pp. 1060–1068. http://dx.doi.org/10.1109/QRS-C55045.2021.00160.

Lincoln, Y.S., Guba, E.G., 1985. Naturalistic Inquiry. sage.

Lorigo, L., Haridasan, M., Brynjarsdóttir, H., Xia, L., Joachims, T., Gay, G., Granka, L., Pellacini, F., Pan, B., 2008. Eye tracking and online search: Lessons learned and challenges ahead. J. Am. Soc. Inf. Sci. Technol. 59 (7), 1041–1052. http://dx.doi.org/10.1002/asi.20794.

Lu, Y., Mao, X., Li, Z., Zhang, Y., Wang, T., Yin, G., 2016. Does the role matter? an investigation of the code quality of casual contributors in github. In: 2016 23rd Asia-Pacific Software Engineering Conference. APSEC, IEEE, pp. 49–56. http://dx.doi.org/10.1109/APSEC.2016.018.

Marshall, M.N., 1996. Sampling for qualitative research. Family Pract. 13 (6), 522–526. http://dx.doi.org/10.1093/fampra/13.6.522.

McCall, J.A., Richards, P.K., Walters, G.F., 1977. Factors in software quality. volume i. concepts and definitions of software quality. Technical Report, Defense Technical Information Center, ADA049014.

Meldrum, S., Licorish, S.A., Owen, C.A., Savarimuthu, B.T.R., 2020a. Understanding stack overflow code quality: A recommendation of caution. Sci. Comput. Progr. 199, 102516. http://dx.doi.org/10.1016/j.scico.2020.102516.

Meldrum, S., Licorish, S.A., Savarimuthu, B.T.R., 2017. Crowdsourced knowledge on stack overflow: A systematic mapping study. In: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering. ACM, pp. 180–185. http://dx.doi.org/10.1145/3084226.3084267.

Meldrum, S., Licorish, S.A., Savarimuthu, B.T.R., 2020b. Exploring research interest in stack overflow–a systematic mapping study and quality evaluation. http://dx.doi.org/10.48550/arXiv.2010.12282, arXiv preprint arXiv:2010.12282.

Meng, N., Nagy, S., Yao, D., Zhuang, W., Argoty, G.A., 2018. Secure coding practices in java: Challenges and vulnerabilities. In: Proceedings of the 40th International Conference on Software Engineering. ACM, pp. 372–383. http://dx.doi.org/10.1145/3180155.3180201.

Miguel, J.P., Mauricio, D., Rodríguez, G., 2014. A review of software quality models for the evaluation of software products. http://dx.doi.org/10.5121/ijsea.2014.5603, arXiv preprint arXiv:1412.2977.

Molléri, J.S., Petersen, K., Mendes, E., 2016. Survey guidelines in software engineering: An annotated review. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 1–6.

Mooney, J.D., 1995. Portability and reusability: common issues and differences. In: Proceedings of the 1995 ACM 23rd Annual Conference on Computer Science. pp. 150–156.

Munson, J.C., Khoshgoftaar, T.M., 1992. Measuring dynamic program complexity. In: ESEM 16: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. Vol. 9, (58), ACM, pp. 1–6. http://dx.doi.org/10.1145/2961111.2962619.

Narasimhan, V.L., Hendradjaya, B., 2007. Some theoretical considerations for a suite of metrics for the integration of software components. Inform. Sci. 177 (3), 844–864. http://dx.doi.org/10.1016/j.ins.2006.07.010.

Nasehi, S.M., Sillito, J., Maurer, F., Burns, C., 2012. What makes a good code example?: A study of programming q&a in StackOverflow. In: 2012 28th IEEE International Conference on Software Maintenance. ICSM, IEEE, pp. 25–34. http://dx.doi.org/10.1109/ICSM.2012.6405249.

Nishinaka, R., Ubayashi, N., Kamei, Y., Sato, R., 2020. How fast and effectively can code change history enrich stack overflow? In: 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS). IEEE, pp. 467–478. http://dx.doi.org/10.1109/QRS51102.2020.00066.

Nistala, P., Nori, K.V., Reddy, R., 2019. Software quality models: A systematic mapping study. In: 2019 IEEE/ACM International Conference on Software and System Processes. ICSSP, IEEE, pp. 125–134. http://dx.doi.org/10.1109/ICSSP.2019.00025.

Oriol, M., Marco, J., Franch, X., 2014. Quality models for web services: A systematic mapping. Inf. Softw. Technol. 56 (10), 1167–1182. http://dx.doi.org/10.1016/j.infsof.2014.03.012.

Packard, V.L., 2018. Encyclopedia of information science and technology. Idea Group Reference 1, 1-50140-794-x.

Peters, E., Aggrey, G., 2020. An ISO 25010 based quality model for ERP systems. Adv. Sci., Technol. Eng. Syst. 5 (2), 578–583. http://dx.doi.org/10.25046/aj050272.

Petersen, K., Ali, N.B., 2011. Identifying strategies for study selection in systematic reviews and maps. In: 2011 International Symposium on Empirical Software Engineering and Measurement. IEEE, pp. 351–354. http://dx.doi.org/10.1109/ESEM.2011.46.

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) 12. pp. 1–10. http://dx.doi.org/10.14236/ewic/EASE2008.8.

Pons, L., Ozkaya, I., 2019. Priority quality attributes for engineering AI-enabled systems. http://dx.doi.org/10.48550/arXiv.1911.02912, arXiv.

Ponzanelli, L., Bavota, G., Di Penta, M., Oliveto, R., Lanza, M., 2014. Mining stackoverflow to turn the ide into a self-confident programming prompter. In: Proceedings of the 11th Working Conference on Mining Software Repositories. ACM, pp. 102–111. http://dx.doi.org/10.1145/2597073.2597077.

Porter, M.E., Heppelmann, J.E., 2014. How smart, connected products are transforming competition. Harv. Bus. Rev. 92 (11), 64–88.

Posnett, D., Hindle, A., Devanbu, P., 2011. A simpler model of software readability. In: Proceedings of the 8th Working Conference on Mining Software Repositories. ACM, pp. 73–82. http://dx.doi.org/10.1145/1985441.1985454.

Potter, W.J., Levine-Donnerstein, D., 1999. Rethinking validity and reliability in content analysis. J. Appl. Commun. Res. 27 (3), 256–284. http://dx.doi.org/10.1080/00909889909365539.

Poulin, J.S., 1994. Measuring software reusability. In: Proceedings of 1994 3rd International Conference on Software Reuse. IEEE, pp. 126–138. http://dx.doi.org/10.1109/ICSR.1994.365803.

Prieto-Diaz, R., 1993. Status report: Software reusability. IEEE Softw. 10 (3), 61–66. http://dx.doi.org/10.1109/52.210605.

Quyoum, A., Dar, M.-U.-D., Quadri, S., 2010. Improving software reliability using software engineering approach- a review. Int. J. Comput. Appl. 10 (5), 41–47.

Ragkhitwetsagul, C., Krinke, J., Paixao, M., Bianco, G., Oliveto, R., 2019. Toxic code snippets on stack overflow. IEEE Trans. Softw. Eng. http://dx.doi.org/10.1109/TSE.2019.2900307.

Rahman, A., Farhana, E., Imtiaz, N., 2019a. Snakes in paradise?: Insecure python-related coding practices in stack overflow. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories. MSR, IEEE, pp. 200–204. http://dx.doi.org/10.1109/MSR.2019.00040.

Rahman, M., Rigby, P., Palani, D., Nguyen, T., 2019b. Cleaning stackoverflow for machine translation. In: 2019 IEEE/ACM 16th International Conference on Mining Software Repositories. MSR, IEEE, pp. 79–83. http://dx.doi.org/10.1109/MSR.2019.00021.

Rawashdeh, A., Matalkah, B., 2006. A new software quality model for evaluating COTS components. J. Comput. Sci. 2 (4), 373–381.

Reid, B., Treude, C., Wagner, M., 2020. Optimising the fit of stack overflow code snippets into existing code. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion. ACM, pp. 1945–1953. http://dx.doi.org/10.1145/3377929.3398087.

Runeson, P., Höst, M., 2009. Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. 14 (2), 131–164. http://dx.doi.org/10.1007/s10664-008-9102-8.

Sahu, K., Srivastava, R., 2019. Revisiting software reliability. Data Manag., Anal. Innov. 221–235. http://dx.doi.org/10.1007/978-981-13-1402-5_17.

Saini, G., Panwar, D., Kumar, S., Singh, V., 2020. A systematic literature review and comparative study of different software quality models. J. Discrete Math. Sci. Cryptogr. 23 (2), 585–593. http://dx.doi.org/10.1080/09720529.2020.1747188.

Scalabrino, S., Linares-Vásquez, M., Oliveto, R., Poshyvanyk, D., 2018. A comprehensive model for code readability. J. Softw.: Evol. Process 30 (6), http://dx.doi.org/10.1002/smr.1958.

Schneidewind, N., 1997. IEEE standard for a software quality metrics methodology revision and reaffirmation. In: Software Engineering Standards, International Symposium on. IEEE Computer Society, p. 278. http://dx.doi.org/10.1109/SESS.1997.596008.

Seffah, A., Metzker, E., 2004. The obstacles and myths of usability and software engineering. Commun. ACM 47 (12), 71–76. http://dx.doi.org/10.1145/1035134.1035136.

Sharma, V., Baliyan, P., 2011. Maintainability analysis of component based systems. Int. J. Softw. Eng. Appl. 5 (3), 107–118.

Shoga, M.Y., Chen, C., Boehm, B., 2020. Recent trends in software quality interrelationships: A systematic mapping study. In: 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C). IEEE, pp. 264–271. http://dx.doi.org/10.1109/QRS-C51114.2020.00052.

Singh, B., Kannojia, S.P., 2013. A review on software quality models. In: 2013 International Conference on Communication Systems and Network Technologies. IEEE, pp. 801–806. http://dx.doi.org/10.1109/CSNT.2013.171.

Smidts, C., Stutzke, M., Stoddard, R.W., 1998. Software reliability modeling: an approach to early reliability prediction. IEEE Trans. Reliab. 47 (3), 268–278. http://dx.doi.org/10.1109/24.740500.

Squire, M., 2015. "Should we move to stack overflow?" measuring the utility of social media for developer support. In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. Vol. 2, IEEE, pp. 219–228. http://dx.doi.org/10.1109/ICSE.2015.150.

Stewart, T., 1998. Ergonomic requirements for office work with visual display terminals (VDTs): Part 11: Guidance on usability. International Organization for Standardization ISO 9241, 89–122.

Stol, K.-J., Ali Babar, M., 2010. Challenges in using open source software in product development: a review of the literature. In: Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development. ACM, pp. 17–22. http://dx.doi.org/10.1145/1833272.1833276.

Subramanian, S., Holmes, R., 2013. Making sense of online code snippets. In: 2013 10th Working Conference on Mining Software Repositories. MSR, IEEE, pp. 85–88. http://dx.doi.org/10.1109/MSR.2013.6624012.

Suman, M.W., Rohtak, M., 2014. A comparative study of software quality models. Int. J. Comput. Sci. Inf. Technol. 5 (4), 5634–5638.

Tashtoush, Y., Al-Maolegi, M., Arkok, B., 2014. The correlation among software complexity metrics with case study. http://dx.doi.org/10.48550/arXiv.1408.4523, arXiv.

Tavakoli, M., Heydarnoori, A., Ghafari, M., 2016. Improving the quality of code snippets in stack overflow. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing. ACM, pp. 1492–1497. http://dx.doi.org/10.1145/2851613.2851789.

Tavakoli, M., Izadi, M., Heydarnoori, A., 2020. Improving quality of a post's set of answers in stack overflow. In: 2020 46th Euromicro Conference on Software Engineering and Advanced Applications. SEAA, IEEE, pp. 504–512. http://dx.doi.org/10.1109/SEAA51224.2020.00084.

Thapar, S.S., Singh, P., Rani, S., 2012. Challenges to development of standard software quality model. Int. J. Comput. Appl. 49 (10).

Treude, C., Barzilay, O., Storey, M.-A., 2011. How do programmers ask and answer questions on the web?(nier track). In: Proceedings of the 33rd International Conference on Software Engineering. ACM, pp. 804–807. http://dx.doi.org/10.1145/1985793.1985907.

Treude, C., Robillard, M.P., 2016. Augmenting API documentation with insights from stack overflow. In: 2016 IEEE/ACM 38th International Conference on Software Engineering. ICSE, IEEE, pp. 392–403. http://dx.doi.org/10.1145/2884781.2884800.

Treude, C., Robillard, M.P., 2017. Understanding stack overflow code fragments. In: 2017 IEEE International Conference on Software Maintenance and Evolution. ICSME, IEEE, pp. 509–513. http://dx.doi.org/10.1109/ICSME.2017.24.

Uddin, G., Khomh, F., Roy, C.K., 2020. Mining API usage scenarios from stack overflow. Inf. Softw. Technol. 122, 106277. http://dx.doi.org/10.1016/j.infsof.2020.106277.

Verdi, M., Sami, A., Akhondali, J., Khomh, F., Uddin, G., Motlagh, A.K., 2020. An empirical study of c++ vulnerabilities in crowd-sourced code examples. IEEE Trans. Softw. Eng. http://dx.doi.org/10.1109/TSE.2020.3023664.

Vogelsang, A., Borg, M., 2019. Requirements engineering for machine learning: Perspectives from data scientists. In: 2019 IEEE 27th International Requirements Engineering Conference Workshops. REW, IEEE, pp. 245–251. http://dx.doi.org/10.1109/REW.2019.00050.

Wagner, S., Goeb, A., Heinemann, L., Kläs, M., Lampasona, C., Lochmann, K., Mayr, A., Plösch, R., Seidl, A., Streit, J., et al., 2015. Operationalised product quality models and assessment: The quamoco approach. Inf. Softw. Technol. 62, 101–123. http://dx.doi.org/10.1016/j.infsof.2015.02.009.

Wohlin, C., Runeson, P., Neto, P.A.d.M.S., Engström, E., do Carmo Machado, I., De Almeida, E.S., 2013. On the reliability of mapping studies in software engineering. J. Syst. Softw. 86 (10), 2594–2610. http://dx.doi.org/10.1016/j.jss.2013.04.076.

Xenos, M., 2001. Usability perspective in software quality. In: Usability Engineering Workshop, the 8th Panhellenic Conference on Informatics with International Participation, Southern Cyprus.

Xu, H., Heijmans, J., Visser, J., 2013. A practical model for rating software security. In: 2013 IEEE Seventh International Conference on Software Security and Reliability Companion. IEEE, pp. 231–232. http://dx.doi.org/10.1109/SERE-C.2013.11.

Xu, B., Xing, Z., Xia, X., Lo, D., 2017. Answerbot: Automated generation of answer summary to developers' technical questions. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering. ASE, IEEE, pp. 706–716. http://dx.doi.org/10.1109/ASE.2017.8115681.

Yang, D., Hussain, A., Lopes, C.V., 2016. From query to usable code: an analysis of stack overflow code snippets. In: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories. MSR, IEEE, pp. 391–401.

Yang, D., Martins, P., Saini, V., Lopes, C., 2017. Stack overflow in github: any snippets there? In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR). IEEE, pp. 280–290. http://dx.doi.org/10.1109/MSR.2017.13.

Ye, Y., Hou, S., Chen, L., Li, X., Zhao, L., Xu, S., Wang, J., Xiong, Q., 2018. ICSD: An automatic system for insecure code snippet detection in stack overflow over heterogeneous information network. In: Proceedings of the 34th Annual Computer Security Applications Conference. ACM, pp. 542–552. http://dx.doi.org/10.1145/3274694.3274742.

Zhang, T., Upadhyaya, G., Reinhardt, A., Rajan, H., Kim, M., 2018. Are code examples on an online q&a forum reliable?: a study of API misuse on stack overflow. In: 2018 IEEE/ACM 40th International Conference on Software Engineering. ICSE, IEEE, pp. 886–896. http://dx.doi.org/10.1145/3180155.3180260.

Zhang, H., Wang, S., Li, H., Chen, T.-H.P., Hassan, A.E., 2021. A study of C/C++ code weaknesses on stack overflow. IEEE Trans. Softw. Eng. http://dx.doi.org/10.1109/TSE.2021.3058985.

Ifeanyi G. Ndukwe is an Assistant Research Fellow in the Department of Information Science at the University of Otago, New Zealand. He was awarded his Ph.D. by the University of Otago in 2021. His research portfolio covers deep learning, machine learning and natural language processing techniques, software code quality, software analytics, and static analysis tools.

Sherlock A. Licorish is a Senior Lecturer in the Department of Information Science at University of Otago, New Zealand. He was awarded his Ph.D. by Auckland University of Technology (AUT), and joined University of Otago in 2014. His research portfolio covers agile methodologies, practices and processes, teams and human factors, human computer interaction, research methods and techniques, software code quality, static analysis tools, machine learning applications and software analytics. Sherlock occupies several service roles across the university, nationally and internationally.

Amjed Tahir is a Senior Lecturer in Software Engineering at Massey University, New Zealand. He obtained a Ph.D. degree from the University of Otago, NZ, in 2016. Amjed research is mainly in empirical software engineering, with special interest in software maintenance and testing problems. He has served in the organising and program committee of several software engineering conferences, including ICSE, ICSME and MSR. Amjed is an Executive member of Software Innovation New Zealand.

Stephen MacDonell is Professor of Software Engineering at Auckland University of Technology and Professor in Information Science at the University of Otago, both in New Zealand. Stephen was awarded BCom(Hons) and MCom degrees from the University of Otago and a Ph.D. from the University of Cambridge. His research has been published in IEEE Transactions on Software Engineering, ACM Transactions on Software Engineering and Methodology, ACM Computing Surveys, Empirical Software Engineering, Information & Management, the Journal of Systems and Software, Information and Software Technology, and the Project Management Journal, and his research findings have been presented at more than 100 international conferences. He is a Fellow of IT Professionals NZ, Senior Member of the IEEE and the IEEE Computer Society, and Member of the ACM, and he serves on the Editorial Board of Information and Software Technology. Stephen is also Deputy Director and Theme Leader for Data Science & Digital Technologies in New Zealand's National Science Challenge Science for Technological Innovation, Technical Advisor to the Office of the Federation of Māori Authorities Pou Whakatāmore Hangarau - Chief Advisor Innovation & Research, and Deputy Chair of Software Innovation New Zealand (SI^NZ).