Electronic Communications of the EASST
Volume 85 Year 2025

deRSE25 - Selected Contributions of the 5th Conference for Research Software Engineering in Germany

Edited by: René Caspart, Florian Goth, Oliver Karras, Jan Linxweiler, Florian Thiery, Joachim Wuttke

Quality Characteristics for Software in HPC Environments: A Systematic Literature Review

Camilla Lüttgens and Marius Politze

DOI: 10.14279/eceasst.v85.2690

Quality Characteristics for Software in HPC Environments: A Systematic Literature Review

Camilla Lüttgens[1] **and Marius Politze**[2]

[1] luettgens@itc.rwth-aachen.de
[2] politze@itc.rwth-aachen.de
IT Center
RWTH Aachen University, Germany

Abstract: Research outputs in general require certain qualities to facilitate reuse as described by the FAIR Principles. For research software specifically, software engineering methods can help realize these goals. However, the desired qualities may differ between commercial and research software or even software in high-performance computing (HPC) environments. This work aims to analyze the current research on software quality in HPC and, in particular, identify important quality characteristics. Therefore, we conducted a systematic literature review on 29 relevant papers and compared our results to the established ISO/IEC 25010 SQuaRE (Systems and software Quality Requirements and Evaluation) quality model. In our analysis, the most frequently considered quality characteristics are performance, correctness, portability and maintainability, as well as trade-offs between performance and other characteristics. Compared to SQuaRE, our findings do not include compatibility, security and safety aspects. The insights from this study provide a starting point for research software engineers in quality considerations for their applications. Additionally, the comparison to SQuaRE could indicate gaps that should receive more attention to improve the reusability of HPC applications.

Keywords: software quality, high-performance computing, research software engineering, quality characteristics, ISO/IEC 25010, SQuaRE

1 Introduction

The FAIR Principles [WDA+16] describe qualities for research outputs in general that facilitate their reuse. For research software specifically, reusability means that a software is "both usable (can be executed) and reusable (can be understood, modified, built upon, or incorporated into other software)" as defined in the FAIR4RS Principles [BCK+22]. Similarly, Collberg and Proebsting [CP16] emphasize the importance of sharing software for repeatability to allow evaluation and for benefaction to enable building upon existing results. Software engineering methods can help realize these goals but need to be adapted to the needs and conditions of research software.

For scientists across many domains, it is often necessary to make use of a high-performance computing (HPC) system's computing and memory capabilities. In this context, reuse of existing software can even be a necessity because it is infeasible to re-implement complex applications from scratch. Moreover, executions are expensive with respect to the required resources making it even more critical to ensure that the software runs without errors and produces the expected results. In this study, we therefore do not look at research software in general but specifically at software in HPC environments. HPC software has unique challenges that impact how software quality is approached including the focus on performance optimization, system heterogeneity, execution on clusters that are not under the control of the developer or researcher, and increased complexity due to parallelization [BLC+16, BCC+08, SB12]. For instance, automated tests run in a Continuous Integration (CI) pipeline are more difficult since hardware-specific implementations require actual cluster environments rather than isolated setups like Docker containers. Additionally, developing reliable tests for parallel code introduces further complications.

This study aims to deepen our understanding of how quality is defined for software in HPC environments. To achieve this, we conducted a Systematic Literature Review (SLR) of existing research on the topic. In particular, we analyze the quality characteristics that are discussed. We employ the established ISO/IEC 25010 SQuaRE (Systems and software Quality Requirements and Evaluation) software quality model [Int23] for a comparison with our findings to identify commonalities and potential gaps. As such, our insights provide a starting point for quality evaluation of HPC research software and contribute towards improving its reusability.

2 Methodology

The purpose of this study is to investigate the existing research on quality characteristics for software in HPC environments. It can be structured with the following research questions:

RQ1 What is the current state of research on quality characteristics of software in HPC environments?

RQ2 Which quality characteristics of software in HPC environments are considered in the literature?

RQ3 How do the quality characteristics from RQ2 compare to existing software quality models?

To address these research questions, we examine the state of research for RQ1 by analyzing the publication years, types of contribution, domains and connections between publications. We focus specifically on the quality characteristics and compare them to existing quality models that will be introduced in Subsection 3.1. Therefore, we have conducted the SLR oriented towards the guidelines by Kitchenham and Charters [KC07], which have been specifically adapted for software engineering research. The steps of our process have been developed and discussed collaboratively by both authors and will be described in the following sections.

2.1 Data Sources and Search Strategy

We are interested in literature about both software quality and HPC. Thus, we have constructed our search query to connect both topics. For the *HPC* part, we included various notations of *high-performance computing*. For the *software quality* part, we also included *code quality* because software in HPC is often referred to as “code”. Additionally, we considered *software* *sustainability*. Although there is no standard and comprehensive definition of software sustainability, it can, from a technical perspective, be viewed as an overarching concern that includes multiple quality characteristics [VKB+21]. Lastly, we included the higher-level field of *software engineering*, but only when *quality* or *sustainability* are also explicitly mentioned. The resulting search string is constructed as follows:

$$\left(\begin{array}{c}\text{"software quality"}\\ \textbf{or}\\ \text{"code quality"}\\ \textbf{or}\\ \text{"software sustainability"}\\ \textbf{or}\\ (\text{"software engineering" } \textbf{and} \text{ ("quality" } \textbf{or} \text{ "sustainability"))}\end{array}\right) \textbf{ and } \left(\begin{array}{c}\text{"hpc"}\\ \textbf{or}\\ \text{"high performance comput*"}\\ \textbf{or}\\ \text{"high-performance comput*"}\end{array}\right)$$

As relevant literature can come from different domains and disciplines - HPC, software engineering or the application domain of the software - we have used three different indices: Scopus[1], BASE (Bielefeld Academic Search Engine)[2], and dblp computer science bibliography[3]. The query has been adapted to the syntax and capability of each search engine.

2.2 Selection Criteria

To find relevant publications, we manually filtered the results based on the inclusion and exclusion criteria outlined in Table 1. This step was conducted solely by the first author. During the selection process, we were able to exclude many publications by reviewing their titles, abstracts and keywords because they either fit into one of the exclusion criteria or did not meet one of the first two inclusion criteria. Applying especially the third inclusion criterion, however, required a more thorough reading of the full articles. To do so, we initially examined each paper's introduction and conclusion to gain an understanding of its context. This also helped us identify which sections were most relevant to quality characteristics; those sections were read in detail, while other parts were skimmed to ensure no important information was overlooked.

[1] https://www.scopus.com/
[2] https://www.base-search.net/
[3] https://dblp.org/

Inclusion criteria	Exclusion criteria
1. Quality refers to an application software (excluding, for example, HPC infrastructure software, result quality and quality of service) 2. The software is intended to be executed in an HPC environment 3. The characteristics of software quality are defined and analyzed	1. Written in another language than English 2. Duplicates 3. Pre-prints or earlier / shorter versions of a publication that is also included 4. Not in continuous full-text format (e.g. tutorials, posters, presentations, extended abstracts) 5. Primarily about teaching the topics

Table 1: Inclusion and exclusion criteria.

2.3 Data Extraction

The extraction step was also carried out solely by the first author using the same reading process as described in Subsection 2.2 for filtering. The extracted data was recorded in a CSV file and the corresponding text was marked in the PDF of each publication. In the following, we describe the data that was collected for each included publication.

To generally characterize the state of research for RQ1, we collected the publication year. We also aimed to understand how the publications are interconnected and therefore extracted all of their references without filtering. Whenever possible, we utilized the DOI (Digital Object Identifier) as a unique identifier for each resource. The references were partially obtained using the CrossRef API[4] but were checked and completed manually.

To group the publications based on their contributions, we use the following categories:

- The proposal of a **tool or process** that helps to measure or improve software quality or certain aspects of it.
- The presentation of a **software** including a description of how software quality is approached in its development.
- An **analysis** of software quality, for example, the current state or the impact of certain factors.

It is important to note that these categories are not mutually exclusive. For instance, the proposal of a tool or process often includes an analysis quantifying its effect on software quality or references to specific software use cases. Similarly, the presentation of a software often includes tools and processes employed to achieve quality goals. In such instances, we assigned the publication to the category that best reflects its primary focus.

Furthermore, we collected whether the publication refers to software within a specific domain or methodology, helping us evaluate the generalizability of our results. For domains, we used the

[4] https://api.crossref.org/

scientific disciplines of the DFG subject area structure [Deu24]: *Humanities and Social Sciences*, *Life Sciences*, *Natural Sciences*, and *Engineering Sciences*. Publications that address multiple domains or are not specific to an application domain were classified as *General*.

The most important information for our study is the software quality characteristics, which is the basis for addressing RQ2 and RQ3. We extracted the characteristics that were stated as being important for software quality. Moreover, we noted if trade-offs are mentioned between two or more quality characteristics.

2.4 Data Synthesis and Mapping

To compare the extracted quality characteristics with existing software quality models, we mapped them to the quality characteristics defined by the International Organization for Standardization [Int23]. Therefore, we revisited the marked text from the publications to understand how the characteristics were defined or in what context they were used, such as the concrete goals they were associated with. An initial suggestion for the mapping was prepared by the first author and included explanations where necessary. This draft was then discussed and refined with the second author and several colleagues. A summary of the characteristics and their mapping can be found in Table 7 and is detailed in Subsection 4.3.

3 Related Work

In the following, we will provide a brief overview on the software quality models that will be used for comparison and studies related to our research.

3.1 Software Quality Models

Software quality models formalize the different characteristics that are important to a software's quality and show the relationships and hierarchies between them. For the comparison to our results, we will mainly use the SQuaRE (Systems and software Quality Requirements and Evaluation) / ISO 25010 quality model [Int23]. An overview of its characteristics and corresponding sub-characteristics can be found in Figure 1. Aljarallah and Lock [AL19] find that it is not only recent, widely accepted and the most complete, but also covers the characteristics important for software sustainability. Their results suggest that the characteristics for software quality and software sustainability are similar, but the focus and therefore the importance of the characteristics is different.

While SQuaRE addresses software quality in general, Koteska et al. [KMP18] propose a quality model specifically tailored to scientific applications. In addition to the characteristics shown in Figure 2, their model includes metrics for quantitative measurement. All of the characteristics are also present in SQuaRE: Although *changeability* and *portability* are not included directly, it is noted that "[m]odifiability is a combination of changeability and stability" and *flexibility* was called *portability* in the previous version [Int23]. In contrast, the quality model for scientific software contains significantly fewer characteristics. Therefore, we primarily utilize SQuaRE for comparison with the quality characteristics from our results while also examining how our findings align with those identified by Koteska et al.

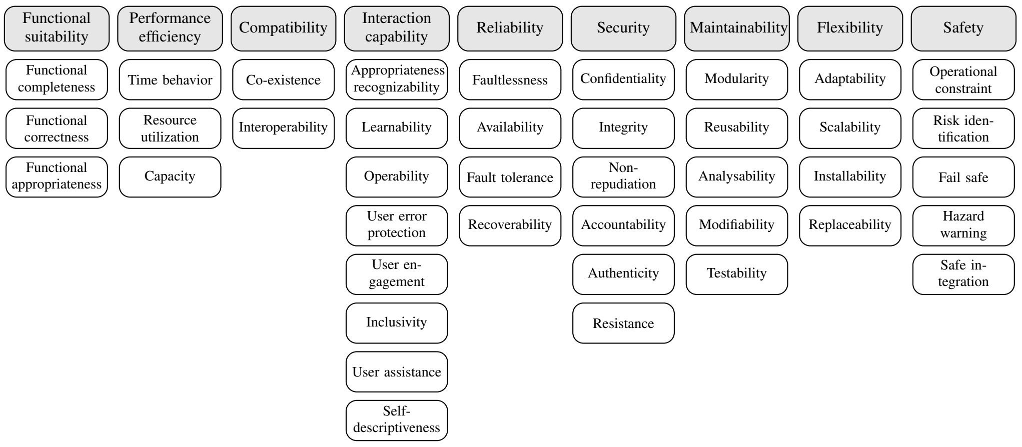


Figure 1: Software Product Quality Model in SQuaRE (Systems and software Quality Requirements and Evaluation) / ISO 25010 [Int23] with the characteristics in gray boxes on the top row and their respective sub-characteristics below.

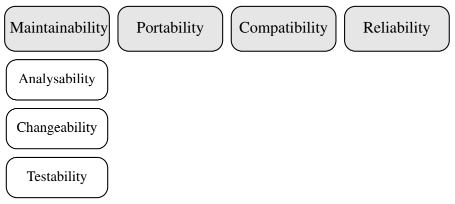


Figure 2: Software quality model for scientific applications [KMP18] with the characteristics in gray boxes on the top row and their sub-characteristics below if they exist.

3.2 Similar Studies

There are several studies on software engineering in HPC. Carver et al. [CKSP07] present lessons learned from case studies of HPC software for computational science and engineering. In there, the most highly ranked project goals are correctness, performance, portability and maintainability. Moreover, they find that performance often competes with the other goals and that portability has high importance across all of their case studies. Additionally, a lesson learned is that verification and validation is difficult in this environment.

Similar challenges are identified by Basili et al. [BCC$^+$08] in their observations of software engineers on how code is developed in the domain of HPC simulation software. While their research is not primarily about quality, it is related to two of the challenges they identify: Balancing performance and development effort - including portability and maintenance - and validating HPC simulation software, which they find to be qualitatively different from commercial software. They conclude that software engineering practices have to be tailored to meet the specific needs of the HPC community.

In [GSND20], Grannan et al. examine a set of successful HPC scientific simulation applications regarding their productivity- and sustainability-related practices. They find that developers increasingly emphasize software engineering practices and factors like longevity, robustness and productivity of software developers. This study is also mainly about software engineering practices which is connected to quality characteristics because they can be improved with certain practices.

[PMV+16] introduces the “Scalability-Efficiency/Maintainability-Portability Trade-Off” in HPC simulation software and present an SLR on the topic. They find evidence for the phenomenon in the literature but identify the need for an empirical foundation, metrics and methods. All attributes involved in the trade-off correspond to quality characteristics in SQuaRE, as shown in Subsection 3.1.

Arvanitou et al. [AACC21] conduct a study on software engineering practices in research software. Their research is not specific to HPC, but they note that “because scientific software often demands a large number of calculations over vast amounts of data, these applications make heavy use of High-Performance Computing” [AACC21]. While the study is again mainly focused on practices, they also collect data on quality attributes to catalog the impact of practices on quality attributes. They find that the most studied quality characteristics are performance, productivity, maintainability, portability and scalability. They conclude that, therefore, performance must always be taken into account and trade-offs should be analyzed.

In summary, there are multiple studies on topics of software engineering in HPC with results that relate to quality characteristics and thus allow for comparison. The main novelty of our study lies in its direct focus on quality characteristics themselves and the comparison with software quality models.

4 Results

The search following the process outlined in Subsection 2.1 yielded a total of 282 results on August 8th, 2024, after excluding duplicates. After filtering according to the selection criteria detailed in Subsection 2.2, 29 relevant publications remained. We also included a step to consolidate data from multiple publications if they were by the same first author, on the same topic, and had identical data extracted. All selected works are published as journal articles or conference proceedings. The data extracted from these publications can be found in [Lum25b] and the source code for the figures based on this information in [Lum25a].

4.1 General Results

Figure 3 shows the distribution of publications over the years. The earliest included paper was published in 2004. From 2011, publications appear more regularly. It is important to note that the count for 2024 includes only publications released up until August.

When analyzing references of the publications to find out how they are connected, we find that there are only three references to another of the publications. While multiple references are cited across more than one publication, as summarized in Table 5, none appear more than three times. Eight publications do not have any common references with others.

The categories proposed in Subsection 2.3 are represented fairly evenly as it can be seen in Figure 4. A slight majority of the results focus on tools or processes for measuring or improving software quality but there are also 33% and 27% respectively about the analysis of software quality and software quality in specific applications. An overview of the publications in each category can be found in Table 2, Table 3 and Table 4.

Figure 5 shows the application domains specified in the publications. By far the majority is classified as "general" meaning that there is either no domain mentioned or multiple domains. All disciplines are represented except for *Humanities and Social Sciences*. With half of the publications being domain specific, the *Software* category has noticeably more than the rest of the results. However, this is expected, given that these publications are about a concrete software project which naturally often is specific to an application domain. In addition, we find that several publications specify their focus on HPC *simulation* software. Apart from that, no other methodologies are mentioned. The detailed numbers can be found in Table 6.

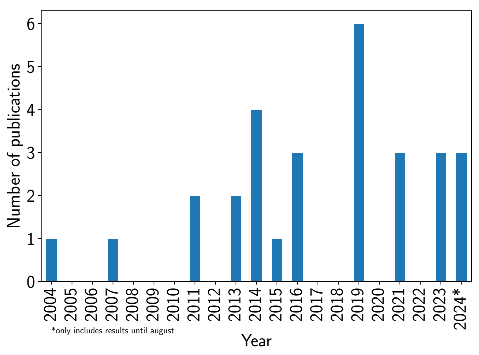


Figure 3: Frequency of publications per year.

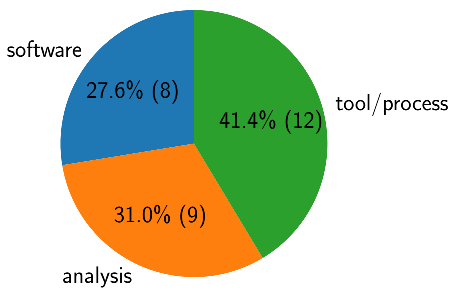


Figure 4: Categorization of the publications.

Author	Summary
Al-Johany et al. [ASEA24]	Static analysis tool for detecting and correcting MPI defects
Anzt et al. [ACC+19]	Framework for automatic performance evaluation
Bigot et al. [BLC+16]	Quality procedures in the development workflow
Cohen et al. [CCC+14]	Component-based framework for platform-agnostic usage
Damevski and Dahlgren [Dam11, DD11]	Design by contract with reduced overhead
Feld et al. [FGH+21]	Automated build and testing infrastructure
Hallissy et al. [HLS+16]	Quality assurance for computation-based engineering
Hussain et al. [HCN21]	Automated detection of code quality rule violation
Panas et al. [PQV07]	Code quality feedback tool
Sáez et al. [SSY+24]	Procedures to improve the quality of existing codes
Sgambati and Anderson [SA23]	HPC container strategy for software quality assurance
Tronge et al. [TCG+24]	Container-based continuous integration scalability testing tool

Table 2: Summary of the included publications in the tool/process category.

Author	Summary
D'Amore et al. [DMBC14]	Insertion of PETSc in *NEMO*
Dembinski et al. [DNRU19]	Design and development of *CORSIKA 8*
Feichtinger et al. [FDK$^+$11]	Design and development of *WaLBerla*
Godoy et al. [GHW$^+$23]	Incorporating modern software engineering practices in *QMCPACK*
Löffler et al. [LBAS14]	Design and development of *Cactus*
Melone and Jones [MJ23]	Continuous integration system for the *HTR solver*
Mirams et al. [MAB$^+$13]	Design and development of *Chaste*
Pflüger and Pfander [PP16]	Trade-off between computational efficiency and other quality attributes in *SG++*

Table 3: Summary of the included publications in the software category.

Author	Summary
Arenaz and Martorell [AM19]	Experimental evaluation of Parallelware static code analysis tools
Baker [Bak21]	Overview of challenges and employed strategies in HPC for climate science
Bernabé et al. [BGI$^+$19]	Portability study of an ADTCA implementation using OpenCL
Källén et al. [KHH14]	Evaluation of the effect of refactoring on code quality
Kempf and Bastian [KB19]	Comparison of development models regarding quality criteria
Koteska and Mishev [KM13]	Analysis of scientific software development to identify suitable practices
Milewicz and Rodeghero [MR19]	Position paper about the importance of usability in scientific HPC software
Post and Kendall [PK04]	Lessons learned on quality engineering in the ASCI program
Wagner et al. [WPM15]	Structured overview of challenges in simulation software engineering

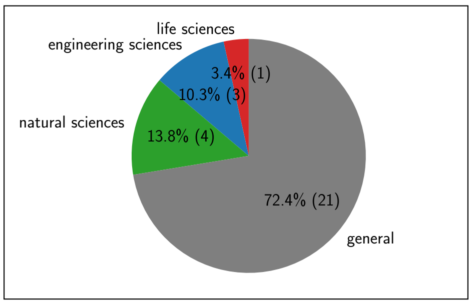
Table 4: Summary of the included publications in the analysis category.

Description	Value
Direct references between the selected publications	3
References that are used in 2 of the selected publications	25
References that are used in 3 of the selected publications	4
Publications that have no common references	8

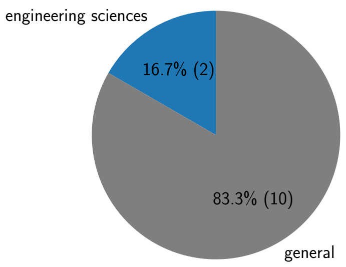
Table 5: Numbers extracted from the references of the publications.

Category	Simulation-specific results
Tool/process	2/12 (16.67%)
Analysis	4/9 (44.44%)
Software	6/8 (75.0%)
Overall	12/29 (41.38%)

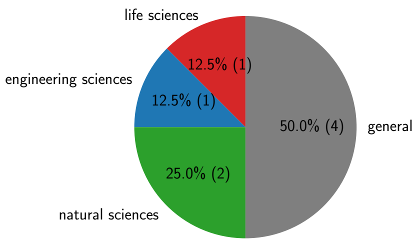
Table 6: Specification of the *simulation* methodology overall and per category.



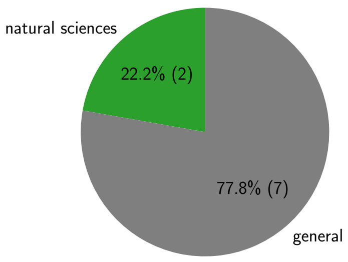
(a) Overall



(b) Tool/process



(c) Software



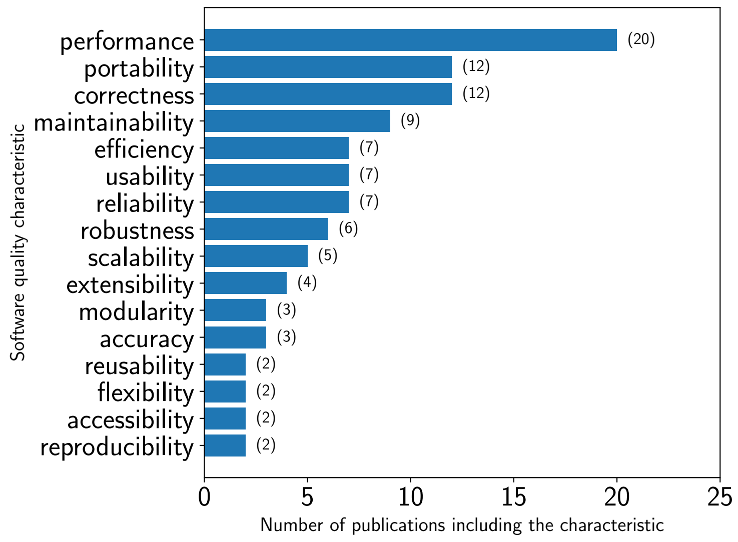
(d) Analysis

Figure 5: Distribution of application domains for the overall results and per category.

4.2 Quality Characteristics

In our initial analysis of the quality characteristics that were mentioned in the publications, we mostly left the characteristics as they were extracted. The only modifications were made to simplify certain terms, including that *(computational) performance* was shortened to *performance*, *computational efficiency* to *efficiency* and *code and performance portability* to *portability*. In one instance, we have done the opposite and made characteristics more specific due to varying meanings of the same term. *Consistency* in [Bak21] refers to outputs leading to the same scientific conclusion even though they are not bit-identical so we specified the term to *result consistency*. In [HLS$^+$16], on the other hand, *consistency* is defined as there being no regression in quality, which is why we have renamed it to *quality consistency*.

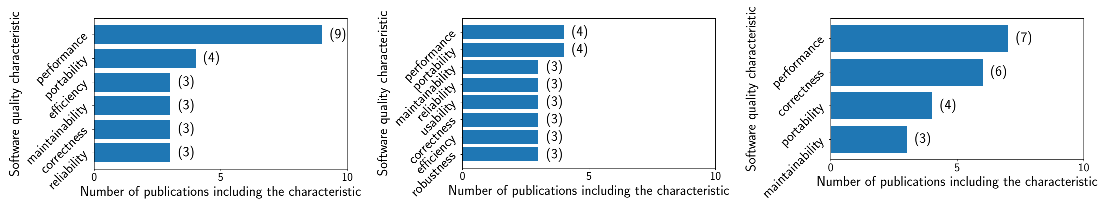
An overview of all the characteristics and their frequencies can be found in Figure 6. Figure 7 shows the same per category with characteristics mentioned fewer than three times removed for clarity. Overall, as well as within each category, *performance* is the most important characteristic or one of the most important characteristics. This is especially the case in the *tool/process* category. In contrast, the results for *software* are more evenly distributed. The four most frequently mentioned characteristics are *performance*, *correctness*, *portability* and *maintainability*. They also appear at least three times in every category.



Quality characteristics included once:

- comprehensibility
- traceability
- simplicity
- transparency
- functionality
- sustainability
- quality consistency
- stability
- expandability
- understandability
- adaptivity
- development effort
- energy efficiency
- result consistency

Figure 6: Frequency of each quality characteristic that appears more than once on the left and a list of characteristics appearing once on the right.



(a) Tool/process (b) Software (c) Analysis

Figure 7: Frequency of appearance of each quality characteristic per category. Only characteristics with at least three appearances are shown.

4.3 Mapping to the ISO/IEC 25010 SQuaRE Software Quality Model

For the mapping of the characteristics from Subsection 4.2, there are different scenarios. The most straightforward case occurs when a characteristics already appears in the SQuaRE model as either a characteristic or sub-characterstic using the same term, a synonym, or a very similar term. Examples for direct matches are *maintainability* as a characteristic and *modularity* as a sub-characteristic. Regarding synonyms, we have mapped *adaptivity* to *adaptability*. Additionally, *portability* was mapped to *flexibility* and *usability* to *interaction capability* because the terms were used in the previous SQuaRE version ISO/IEC 25010:2011 [Int23]. For *portability*, however, it should be noted that in the context of HPC it refers the portability to another platform and not to other requirements or contexts of use, which *flexibility* includes as well according to the SQuaRE definition [Int23]. Similar terms included mapping *performance* to *time behaviour* and *extensibility* to *modifiability*. *Accuracy* was mapped to *correctness* because "[p]recision is one of the attributes of correctness" [Int23].

If terms are used more broadly and include aspects of multiple closely related SQuaRE characteristics, we have looked into the usage in the source material and mapped them to the most prevalent characteristic. *Accessibility* appears twice in our results and has been split into *inclusivity* and *user assistance* compared to the previous SQuaRE version ISO/IEC 25010:2011 [Int23]. In [CCC$^+$14], the term is used in the context of creating user interfaces that abstract away part of the complexity of HPC codes to make them more accessible to users with different backgrounds making it fit best to *inclusivity*. In contrast, [KB19] emphasizes assisting users and developers with good error handling, documentation and training resources and thus refer more to *user assistance*. *Development effort* is described as "how difficult [it] is to write and maintain the code" [BGI$^+$19]. While the complexity of writing the code is relevant when comparing different approaches, it does not matter that much for the quality anymore once the software exists. Therefore, we have mapped it to *maintainability*.

Next, some characteristics fit under an existing characteristic in SQuaRE but address aspects not yet represented within the sub-characteristics. These include *comprehensibility*, *simplicity* and *transparency* under *maintainability*, *quality consistency* under *reliability*, *understandability* under *interaction capability*, and *energy efficiency* under *performance efficiency*. We have again looked at the context of how the terms were used in the sources to map it to the best fitting characteristic. As *comprehensibility* is used in the context of new developers in [WPM15], we used *maintainability* as the main characteristic. Although *understandability* is a very similar term, it fit better under *interaction capability* because it is discussed in the context of scientists extending the software for their use cases in [FDK$^+$11]. Additionally, we classify *scalability* as an added sub-characteristic of *performance efficiency*, despite it also appearing as a sub-characteristic of *flexibility* in SQuaRE. There, it is described as the "capability of a product to handle growing or shrinking workloads or to adapt its capacity to handle variability" [Int23]. In the context of HPC, however, it refers to the ability to improve performance by utilizing additional resources. Thus, we deemed it more appropriate as a sub-characteristic of *performance efficiency*.

Lastly, two characteristics could not be mapped directly to any SQuaRE characteristic: *reproducibility* and *traceability*. While they do not fit exactly into existing categories, they are nonetheless connected to them. For a software to compute reproducible results, it has to function reliably and have functional requirements that support reproducibility. Traceability contributes to maintainability, particularly in terms of analysability, as well as flexibility and compatibility.

An important point to highlight is the mapping of *sustainability* to *maintainability*. As mentioned earlier, we view sustainability as an overarching goal that encompasses multiple characteristics. However, in the publication from which we extracted this term, it is defined as "aspects of sustainability of the software development process" [KB19], including elements such as testing and documentation. Therefore, we find the mapping to be appropriate within this particular context.

A summary of the mapping with explanations can be found in Table 7. Figure 8 provides an overview of the characteristics and their frequencies grouped by the main characteristics of SQuaRE. Given that certain characteristics have a lot more sub-characteristics than others, we also visualized the frequency of each main characteristics, counting instances where the characteristic or at least one of its sub-characteristics is considered, in Figure 9. Additionally, it is noteworthy that three characteristics are not represented in our results at all: *compatibility*, *security* and *safety*.

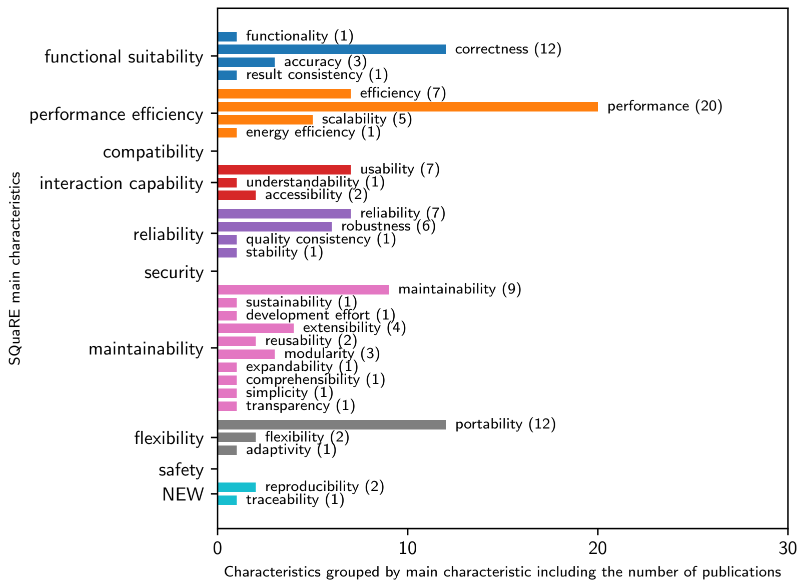


Figure 8: Frequency of appearance of each quality characteristic mapped to the SQuaRE main characteristics.

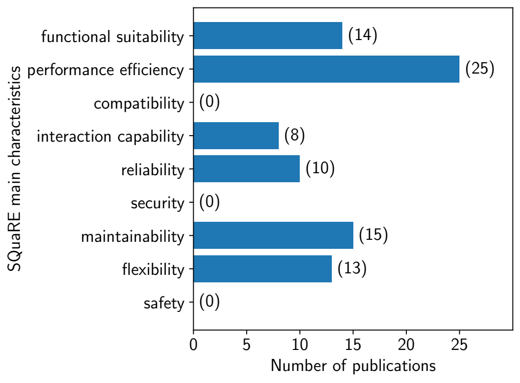


Figure 9: Frequency of appearance of a SQuaRE main characteristic or at least one of its sub-characteristics.

Characteristic	SQuaRE characteristic	SQuaRE sub-characteristic	Explanation
performance	performance efficiency	time behaviour	
portability	flexibility	-	Instead of *flexibility*, *portability* was used in the previous SQuaRE version ISO/IEC 25010:2011 [Int23].
correctness	functional suitability	functional correctness	
maintainability	maintainability	-	
efficiency	performance efficiency	-	
usability	interaction capability	-	Instead of *interaction capability*, *usability* was used in the previous SQuaRE version ISO/IEC 25010:2011 and is “a prerequisite for usability” [Int23].
reliability	reliability	-	
robustness	reliability	-	Robustness is associated with a software working reliably even on adverse conditions like unexpected inputs or faults.
scalability	performance efficiency	NEW	In SQuaRE, *Scalability* is a sub-characteristic of *flexibility*. In the context of HPC, however, it is used as the ability to improve performance by utilizing additional resources which is why we decided to add it as a sub-characteristic of *performance efficiency*.
extensibility	maintainability	modifiability	
modularity	maintainability	modularity	
accuracy	functional suitability	correctness	In SQuaRE, it is noted that “Precision is one of the attributes of correctness” [Int23].
reusability	maintainability	reusability	
flexibility	flexibility	-	
accessibility	interaction capability	inclusivity	*Accessibility* has been split into *inclusivity* and *user assistance* compared to the previous SQuaRE version ISO/IEC 25010:2011. In [CCC+14], the term is used in the context of creating user interfaces that abstract away part of the complexity of HPC codes to make them more accessible to users with different backgrounds.
accessibility	interaction capability	user assistance	In contrast to the row above, [KB19] emphasizes assisting users and developers with good error handling, documentation and training resources.
reproducibility	NEW	-	*Reproducibility* is not represented in SQuaRE and does not fit exactly into any of its characteristics.
comprehensibility	maintainability	NEW	In [WPM15], *comprehensibility* is used in the context of new developers.
traceability	NEW	-	While multiple quality characteristics are related to *traceability* (for example *analysability*), we could not exactly map it to one of the characteristics in SQuaRE.
simplicity	maintainability	NEW	*Simplicity* is a quality that makes comprehending, testing, modifying, extending and overall maintenance of a software easier.
transparency	maintainability	NEW	Similar to *simplicity*, *transparency* is also a quality that improves *maintainability*, especially related to *comprehensibility* and *analysability*.
functionality	functional suitability	-	
sustainability	maintainability	-	In general, we consider *sustainability* to be an overarching goal that includes multiple characteristics. As the term is used for “aspects of sustainability of the software development process” [KB19] in the source, including, for example, testing and documentation, we find that it fits best into the *maintainability* characteristic.
quality consistency	reliability	NEW	In [HLS+16], the term is desribed at there being no regression in the other quality characteristics.
stability	reliability	faultlessness	A software is considered stable if it performs without fault.
expandability	maintainability	modifiability	
understandability	interaction capability	NEW	Although it is similar to *comprehensibility*, *understandability* is discussed in the context of scientists extending the software for their use cases in [FDK+11].
adaptivity	flexibility	adaptability	
development effort	maintainability	-	*Development effort* is described as “how difficult [it] is to write and maintain the code” [BGI+19]. While the complexity of writing the code is relevant when comparing different approaches, it does not matter that much for the quality anymore once the software exists.
energy efficiency	performance efficiency	NEW	
results consistency	functional suitability	functional correctness	*Consistency* of outputs in [Bak21] means they lead to the same scientific conclusion even though they are not bit-identical.

Table 7: Mapping of the quality characteristics from the publications to those in SQuaRE [Int23]. (sub)-characteristics that are not reflected in SQuaRE are denoted as “NEW”. A “-” in the sub-characteristic column means that it is mapped to the main characteristic directly.

4.4 Trade-offs between Quality Characteristics

Eleven of the publications describe a trade-off between different quality characteristics. Nearly all of them involve *performance efficiency* characteristics in relation to another characteristic, often described as the performance cost of a measure aimed at improving other qualities. The other characteristics include aspects of *maintainability* (eight times), *portability* (four times), *usability* (two times) and *correctness* (one time). Other trade-offs that appear one time each include those between *energy consumption* and *performance*, as well as between *correctness* and *usability*. Notably, six of these instances occur within the *analysis* category, indicating that trade-offs are predominantly discussed in that context.

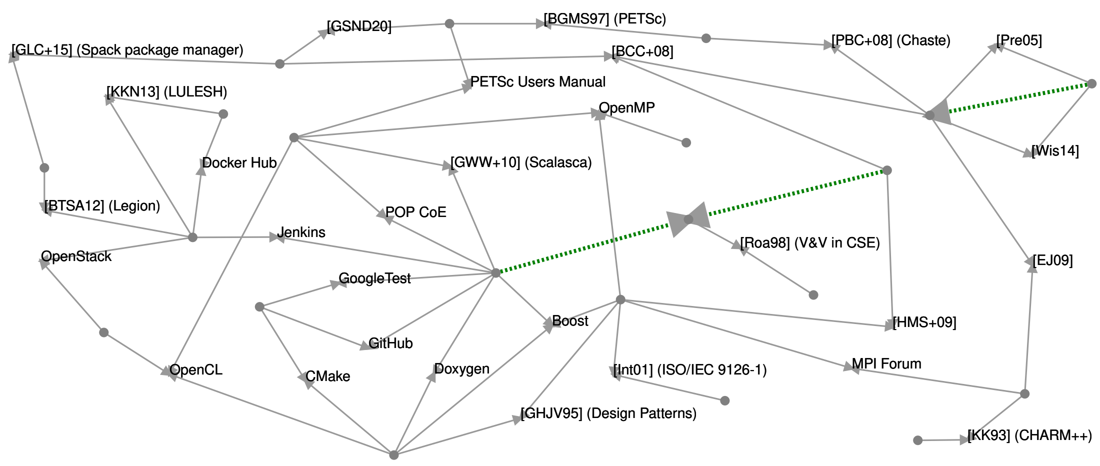
For example, Kempf and Bastian [KB19] illustrate trade-offs in software quality characteristics across different development models. They note that while writing HPC code from scratch yields optimal performance, hardcoding problem and hardware specifics negatively affects other qualities. To enhance especially extensibility, frameworks can be used instead. They define interfaces and encapsulate code. However, these interfaces may introduce assumptions that limit performance. Additionally, while frameworks can improve portability when combined with generative programming, this may restrict extensibility and impact sustainability [KB19]. Similarly, Källén et al. [KHH14] found that refactoring code improved maintainability but reduced performance mainly due to the introduction of dynamic polymorphism. They suggest a compromise by employing static polymorphism in performance-critical parts.

5 Discussion

This section is organized by the research questions introduced in Section 2. We point out the results that we believe give insight on answering the questions, give our interpretations and - if applicable - compare them to the results of other studies. Lastly, we reflect on the strengths and weaknesses of the study.

5.1 State of Research (RQ1)

Research on quality characteristics for HPC software can be described as relatively new, with the first included study dating back to 2004. This timeframe coincides with the period where Arvanitou et al. [AACC21] observed a significant increase in research focused on software engineering practices for scientific software. In contrast, our results indicate that this substantial increase in research specifically related to quality characteristics in HPC occurred only around 2011/2013, nearly ten years later. However, given that we only have 29 datapoints spanning 20 years, any conclusions drawn regarding trends should be interpreted cautiously. Nevertheless, the results clearly demonstrate that quality characteristics are a current topic of interest in HPC. Following the period covered in Arvanitou et al.'s study through 2019, we identified nine additional publications, further indicating that this area of research remains active.



Reference	Author	Title
[BGMS97]	Balay et al.	Efficient Management of Parallelism in Object-Oriented Numerical Software Libraries
[BCC+08]	Basili et al.	Understanding the High-Performance-Computing Community: A Software Engineer's Perspective
[BTSA12]	Bauer et al.	Legion: Expressing locality and independence with logical regions
[EJ09]	Easterbrook and Johns	Engineering the Software for Understanding Climate Change
[GLC+15]	Gamblin et al.	The Spack package manager: bringing order to HPC software chaos
[GHJV95]	Gamma et al.	Design Patterns: Elements of Reusable Object-Oriented Software
[GWW+10]	Geimer et al.	The Scalasca performance toolset architecture
[GSND20]	Grannan et al.	Understanding the landscape of scientific software used on high-performance computing platforms
[HMS+09]	Hannay et al.	How do scientists develop and use scientific software?
[Int01]	International Organization for Standardization	Software engineering — Product quality
[KK93]	Kale and Krishnan	CHARM++: a portable concurrent object oriented system based on C++
[KKN13]	Karlin et al.	LULESH 2.0 Updates and Changes
[PBC+08]	Pitt-Francis et al.	Chaste: using agile programming techniques to develop computational biology software
[Pre05]	President's Information Technology Advisory Committee	Computational Science: Ensuring America's Competitiveness
[Roa98]	Roache	Verification and Validation in Computational Science and Engineering
[Wis14]	Wissenschaftsrat	Bedeutung und Weiterentwicklung von Simulation in der Wissenschaft

Figure 10: All common references between the included papers (dots). Thick green dashed arrows indicate direct citations of included papers. References have a descriptive name if they refer to a website. An interactive version can be found in https://slr-hpc-software-quality-794fea.pages.git.nrw/ and [Lum25a].

The majority of the papers in our study can be categorized as tools or processes for software quality specifically developed for HPC, highlighting the need to adapt those designed for general software. In this category, the quality characteristic *performance* is particularly prominent, indicating that the focus on performance is a key distinguishing feature of HPC software that makes it necessary to adapt tools and processes or create new ones. The other categories, however, also account for nearly one-third of the papers each, suggesting that this topic is addressed in both research (the analysis category) and practical software projects (the software category). In these cases, the distribution of quality characteristics is more balanced.

The publications with at least one common reference are visualized as a network in Figure 10. They form one connected cluster without significant references connecting many of our results, as all references are cited by at most three publications. This observation reinforces our assumption that a comprehensive overview connecting all of this research is indeed lacking. It is important to note, however, that the connections only work "backwards" and do not include other publications that cite the papers included in our review.

When looking at the works cited by multiple papers from our study, we find two of the studies mentioned in our related work: [BCC+08] and [GSND20]. Additionally, it is noteworthy that many of the connections are software tools such as GitHub, Doxygen and OpenMP. Many of these references are websites and not scientific papers. As they are not always included in the reference list, it is possible that even more publications refer to these tools within their texts. The varying ways in which the software PETSc is cited - through its documentation, website, a paper describing it ([BGMS97]), and possibly more - illustrate the challenges associated with citing software. Lastly, we also see that an earlier version of the SQuaRE quality model ([Int01]) has been cited twice.

5.2 Quality Characteristics (RQ2)

Unsurprisingly, *performance* and other characteristics related to efficiency are important in HPC. They are described as a "key factor" [TCG+24], "the holy grail" [WPM15] or "the defining measure" [KB19] for HPC applications. Consequently, several of the included publications present approaches to integrate performance evaluation into Continuous Integration processes [ACC+19, MJ23, TCG+24]. Additionally, techniques such as Design by Contract must be adapted to reduce overhead at runtime [Dam11, DD11]. However, discussions around trade-offs indicate that while performance and efficiency are an important goal, other aspects are also significant. Pflüger and Pfander [PP16], for example, call finding the "cut-off between computational efficiency and usability, flexibility, extensibility" a critical design decision.

Another characteristic that we found to be particularly important in HPC is *portability*. While portability generally refers to software flexibility [Int23], it here specifically pertains to software functioning across heterogeneous platforms. The detailed study on the portability of an algorithm in [BGI+19] illustrates different factors that play a role when evaluating portability - performance, energy consumption and code complexity - and the trade-offs between them.

Correctness is also a main concern according to our results. Therefore, automated testing is also desirable for HPC applications; however, setting up these tests can be challenging, especially when different cluster setups should be tested to ensure portability as well [BLC+16, FGH+21]. Another issue when checking for correctness arises, for example, if results are not bit-identical, as described by Baker [Bak21] in the context of earth system models. Additionally, defects can be introduced through the use of MPI, necessitating separate correctness checks [ASEA24].

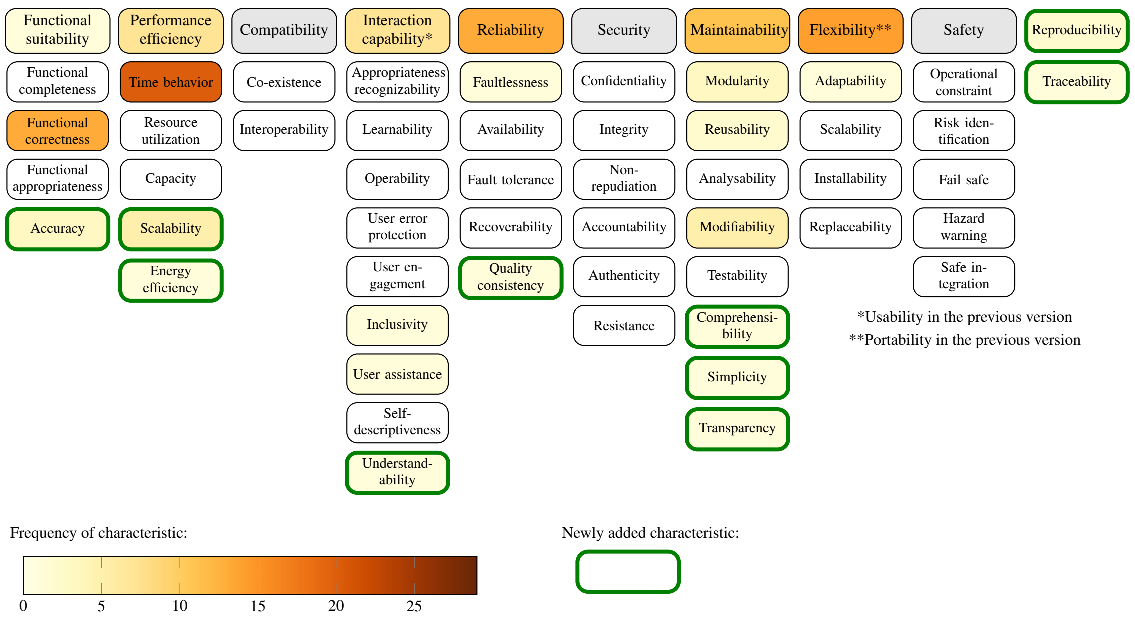
The fourth characteristic we found mentioned frequently is *maintainability*. Considerations include, for example, feedback tools that are unobtrusive and thus still allow for making trade-offs for performance [PQV07], as well as design decisions that make frameworks modular and extensible [DNRU19, FDK+11, LBAS14].

When comparing our findings with existing literature, we also observe the "Scalability/Efficiency-Maintainability/Portability Trade-off" identified by [PMV+16]. This is evident both in the importance assigned to each of these characteristics and in discussions surrounding their trade-offs. Our four most frequently mentioned quality characteristics are the same as the most highly ranked project goals in [CKSP07], but in a different order, as performance is ranked second after correctness in their results. They find that performance "is important only to the extent that the software can be used by its customers" [CKSP07]. Additionally, there are similarities to the quality attributes found by Arvanitou et al. [AACC21]. We especially align with their conclusion that performance always needs to be considered and that measures aimed at improving other characteristics should account for their impact on performance. According to our results, *correctness* (ranked 6th in their study) and *portability* (ranked 4th) are emphasized more in HPC which can be explained by the complexity from parallelization and system heterogeneity. *Productivity* (ranked 2nd in their study), which refers to development efficiency, was not included in our results. A reason may be that this attribute is not named explicitly within the context of software quality characteristics.

5.3 Comparison to Software Quality Models (RQ3)

Figure 11 visualizes the characteristics identified in our results within the SQuaRE quality model. Six out of the nine main characteristics are represented in our findings. Drawing conclusions for the sub-characteristics, however, is more difficult as publications often only mention the main characteristic. Moreover, specific characteristics such as *installability* may not be explicitly listed. Notably, especially *maintainability* comprises several distinct sub-characteristics that are often considered explicitly in the literature.

During the data extraction and mapping process, we observed that it was often difficult to decide whether a characteristic referred to *interaction capability* (usability) or *maintainability*. For instance, in [KB19], *accessibility* is used to describe typical usability aspects like error handling while also referring to "how easy it is for a developer to get the code, run it and adapt it to its needs". Similarly, *understandability* in [FDK+11] is about the "easy integration of new simulation scenarios and numerical methods". This ambiguity can be attributed to a common characteristic of research software, where the developers are often the researchers themselves and thus also the users of their own software. When evaluating the quality of a research software, it must be taken into account that the two qualities cannot always be separated.



Functional suitability	1
Functional correctness	13
Accuracy	3
Performance efficiency	7
Time behavior	20
Scalability	5
Energy efficiency	1
Interaction capability	7
Inclusivity	1
User assistance	1
Understandability	1

Reliability	13
Faultlessness	1
Quality consistency	1
Maintainability	11
Modularity	3
Reusability	2
Modifiability	5
Comprehensibility	1
Simplicity	1
Transparency	1

Flexibility	14
Adaptability	1
Reproducibility	2
Traceability	1

Figure 11: SQuaRE quality model with the characteristics from the results highlighted depending on the frequency. New characteristics from the results are added in green.

Another important insight is presented by the characteristics in SQuaRE that were not represented in our results: *compatibility*, *security* and *safety*. Possible explanations include that the characteristics are implicitly considered, that they are in general not relevant to HPC software, or that they represent aspects currently overlooked but deserving of attention. Regarding *compatibility*, we argue that it is not irrelevant as *interoperability* is one of its sub-characteristics, which facilitates interaction with other software and is part of the FAIR and FAIR4RS criteria [BCK+22]. Examples like the xSDK (Extreme-scale Scientific Software Development Kit), where *interoperability* is one of the main objectives [BDG+17], show that it is also not generally overlooked. It can be argued that aspects of *security* and *safety* are often less critical for HPC applications. However, they can become more important when working with sensitive data, when making software available for reuse in different contexts, or when integrating code into more interactive platforms.

Lastly, there are characteristics in our findings that are not represented in SQuaRE. While their frequency of occurrence is relatively low, we believe that especially *reproducibility* should not be ignored in considerations of HPC software quality, as it is highly relevant in research contexts. Although reproducibility is more about the results generated by a software than the software itself, it plays a role in multiple other criteria. A software can have mechanisms and design decisions that enable or improve reproducibility. An example is adding the option for a seed in a pseudo-random number generator algorithm [BLC$^+$16].

In Subsection 3.1 we also introduced a quality model specific to the quality of scientific software [KMP18]. When comparing the main characteristics of this model to our results, we note that all three are included. However, we find that with *performance* and *correctness*, two of the most frequent characteristics from our study, are missing. They are indirectly addressed by, for instance, requiring *reliability* to make sure the software works as intended in long execution times. However, if the model is used for HPC software, we suggest making use of the model's flexibility and adding them as separate characteristics to have them included explicitly.

Additionally, *usability* was not included because "in scientific applications, the end-users are usually the scientists themselves" [KMP18]. As discussed earlier, we also found it challenging to differentiate between *usability* and *maintainability* and agree with the benefits of merging them into one characteristic. We recommend, however, to then incorporate usability-related sub-characteristics such as *understandability* and *accessibility*. Especially in HPC, there are established and popular tools with many users beyond the developers. Moreover, even when users need to write code themselves, they may only need to interact with certain components rather than all core implementations.

5.4 Strengths and Weaknesses

A particular strength of the study is that almost a third of the analyzed publications focus on a specific software in HPC and how quality is approached in these real-world applications. This means the results do not solely represent an academic or theoretical perspective but also offer valuable insights into the state of practice and quality characteristics relevant to actual HPC applications. Furthermore, our analysis indicates that the majority of the results are not specific to a single application domain. This enhances the external validity of our conclusions, suggesting that they are generally applicable to HPC software. The notable exception is a high representation of studies specific to simulation software.

To enhance the internal validity of the study and reduce bias in the design, we discussed the design of the SLR and took several measures to minimize the risk of missing relevant studies: We incorporated synonyms in the search strings, used multiple databases and search engines covering multiple domains, and analyzed common references in the selected publications. Especially the mapping of the extracted quality characteristics to the SQuaRE model has been discussed with various colleagues to incorporate diverse perspectives. However, the study selection and data extraction steps were conducted solely by the first author, which may introduce individual bias. This limitation could affect the reproducibility and objectivity of our findings.

Lastly, it must be noted that the study by design only considers publications that explicitly address software quality. Consequently, research on individual quality characteristics is excluded if it does not state that it pertains to software quality. There is, for example, extensive research on the performance of applications that does not mention the term *quality*. Additionally, it is possible that there are aspects of HPC software quality that are considered obvious or standard, rarely stated explicitly, and thus excluded from our analysis. We consciously decided not to include these implicit perspectives in the scope of our study because it could make an objective and unbiased data extraction infeasible.

6 Conclusion and Outlook

Overall, we find that *performance*, *portability*, *correctness*, and *maintainability* are the most frequently considered characteristics for the quality of HPC applications. This covers four out of the nine main quality characteristics in SQuaRE. Additionally, aspects of *reliability* and *interaction capability* (*usability*) are considered in HPC. Importantly, trade-offs are often discussed, especially regarding the performance costs that come with improving other qualities. In our analysis, we could not find inclusions of *compatibility*, *security*, and *safety* - the remaining characteristics from SQuaRE. The results confirm that focusing on HPC software specifically is justified due to the importance of different quality characteristics. However, we also observed overlap with general software quality attributes and parallels in the employed practices. Existing tools and processes to enhance software quality can - and should - be used in HPC as well, but may require adaptation to address the specific characteristics of HPC software.

We hope our findings serve as a baseline regarding key quality aspects worth considering during the development or maintenance of HPC applications, but also in evaluating possible risks when deciding whether a software should be used. We recognize that decisions on how to weigh different factors, which specific quality goals to set and which measures to use depend on individual projects and can not be generalized. Therefore, our goal is to develop a template for such a documentation structured by the quality characteristics from our results that facilitates assessment and comparison of software by making these decisions clear and linking related resources.

Future research could extend our study and extract data on the metrics used to evaluate the characteristics to find common metrics that could be used as recommendations or estimates and find gaps where suitable metrics are missing. In addition, further studies on quality characteristics in HPC using alternative methodologies could complement our findings by focusing more on implicit quality aspects and practical experiences. A concrete approach could be to examine the repositories and documentation of the software mentioned in the publications of this study, provided that this information is available.

Bibliography

[AACC21] E.-M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, J. C. Carver. Software engineering practices for scientific software development: A systematic mapping study. *Journal of Systems and Software* 172:110848, Feb. 2021.
doi:10.1016/j.jss.2020.110848

[ACC+19] H. Anzt, Y.-C. Chen, T. Cojean, J. Dongarra, G. Flegar, P. Nayak, E. S. Quintana-Ortí, Y. M. Tsai, W. Wang. Towards Continuous Benchmarking: An Automated Performance Evaluation Framework for High Performance Software. In *Proceedings of the Platform for Advanced Scientific Computing Conference*. PASC '19. ACM, June 2019.
doi:10.1145/3324989.3325719

[AL19] S. Aljarallah, R. Lock. A Comparison of Software Quality Characteristics and Software Sustainability Characteristics. In *Proceedings of the 2019 3rd International Symposium on Computer Science and Intelligent Control*. ACM, Sept. 2019.
doi:10.1145/3386164.3389078

[AM19] M. Arenaz, X. Martorell. Parallelware Tools: An Experimental Evaluation on POWER Systems. In Weiland et al. (eds.), *High Performance Computing*. Pp. 352–360. Springer International Publishing, 2019.
doi:10.1007/978-3-030-34356-9_27

[ASEA24] N. A. Al-Johany, S. A. Sharaf, F. E. Eassa, R. A. Alnanih. Static Analysis Techniques for Fixing Software Defects in MPI-Based Parallel Programs. *Computers, Materials & Continua* 79(2):3139–3173, 2024.
doi:10.32604/cmc.2024.047392

[Bak21] A. H. Baker. On Preserving Scientific Integrity for Climate Model Data in the HPC Era. *Computing in Science & Engineering* 23(6):16–24, Nov. 2021.
doi:10.1109/mcse.2021.3119509

[BCC+08] V. R. Basili, J. C. Carver, D. Cruzes, L. M. Hochstein, J. K. Hollingsworth, F. Shull, M. V. Zelkowitz. Understanding the High-Performance-Computing Community: A Software Engineer's Perspective. *IEEE Software* 25(4):29–36, July 2008.
doi:10.1109/ms.2008.103

[BCK+22] M. Barker, N. P. Chue Hong, D. S. Katz, A.-L. Lamprecht, C. Martinez-Ortiz, F. Psomopoulos, J. Harrow, L. J. Castro, M. Gruenpeter, P. A. Martinez, T. Honeyman. Introducing the FAIR Principles for research software. *Scientific Data* 9(1), Oct. 2022.
doi:10.1038/s41597-022-01710-x

[BDG+17] R. Bartlett, I. Demeshko, T. Gamblin, G. Hammond, M. A. Heroux, J. Johnson, A. Klinvex, X. Li, L. C. McInnes, J. D. Moulton, D. Osei-Kuffuor, J. Sarich,

B. Smith, J. Willenbring, U. M. Yang. xSDK Foundations: Toward an Extreme-scale Scientific Software Development Kit. *Supercomputing Frontiers and Innovations* 4(1):69–82, Mar. 2017.
doi:10.14529/jsfi170104

[BGI+19] S. Bernabé, C. García, F. D. Igual, G. Botella, M. Prieto-Matias, A. Plaza. Portability Study of an OpenCL Algorithm for Automatic Target Detection in Hyperspectral Images. *IEEE Transactions on Geoscience and Remote Sensing* 57(11):9499–9511, Nov. 2019.
doi:10.1109/tgrs.2019.2927077

[BGMS97] S. Balay, W. D. Gropp, L. C. McInnes, B. F. Smith. *Efficient Management of Parallelism in Object-Oriented Numerical Software Libraries*. Pp. 163–202. Birkhäuser Boston, 1997.
doi:10.1007/978-1-4612-1986-6_8

[BLC+16] J. Bigot, G. Latu, T. Cartier-Michaud, V. Grandgirard, C. Passeron, F. Rozar. An approach to increase reliability of HPC simulation, application to the Gysela5D code. *ESAIM: Proceedings and Surveys* 53:248–270, Mar. 2016.
doi:10.1051/proc/201653015

[BTSA12] M. Bauer, S. Treichler, E. Slaughter, A. Aiken. Legion: Expressing locality and independence with logical regions. In *2012 International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Nov. 2012.
doi:10.1109/sc.2012.71

[CCC+14] J. Cohen, C. Cantwell, N. Chue Hong, D. Moxey, M. Illingworth, A. Turner, J. Darlington, S. Sherwin. Simplifying the Development, Use and Sustainability of HPC Software. *Journal of Open Research Software* 2(1), 2014.
doi:10.5334/jors.az

[CKSP07] J. C. Carver, R. P. Kendall, S. E. Squires, D. E. Post. Software Development Environments for Scientific and Engineering Software: A Series of Case Studies. In *29th International Conference on Software Engineering (ICSE'07)*. IEEE, May 2007.
doi:10.1109/icse.2007.77

[CP16] C. Collberg, T. A. Proebsting. Repeatability in computer systems research. *Communications of the ACM* 59(3):62–69, Feb. 2016.
doi:10.1145/2812803

[Dam11] K. Damevski. Offline enforcement of contracts for high-performance computing. *Concurrency and Computation: Practice and Experience* 23(13):1465–1473, Sept. 2011.
doi:10.1002/cpe.1702

[DD11] K. Damevski, T. Dahlgren. Parallel Object Contracts for High Performance Computing. In *2011 IEEE International Symposium on Parallel and Distributed Processing*

Workshops and Phd Forum. IEEE, May 2011.
doi:10.1109/ipdps.2011.263

[Deu24] Deutsche Forschungsgemeinschaft. DFG Classification of Scientific Disciplines, Research Areas, Review Boards and Subject Areas (2024-2028). Apr. 2024.
https://www.dfg.de/resource/blob/331950/85717c3edb9ea8bd453d5110849865d3/fachsystematik-2024-2028-en-data.pdf

[DMBC14] L. D'Amore, A. Murli, V. Boccia, L. Carracciuolo. Insertion of PETSc in the NEMO stack software driving NEMO towards exascale computing. In *2014 International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, July 2014.
doi:10.1109/hpcsim.2014.6903761

[DNRU19] H. Dembinski, L. Nellen, M. Reininghaus, R. Ulrich. Technical Foundations of CORSIKA 8: New Concepts for Scientific Computing. In *36th International Cosmic Ray Conference (ICRC 2019)*. Proceedings of Science 236. July 2019.
doi:10.5445/IR/1000099298

[EJ09] S. M. Easterbrook, T. C. Johns. Engineering the Software for Understanding Climate Change. *Computing in Science & Engineering* 11(6):65–74, Nov. 2009.
doi:10.1109/mcse.2009.193

[FDK+11] C. Feichtinger, S. Donath, H. Köstler, J. Götz, U. Rüde. WaLBerla: HPC software design for computational engineering simulations. *Journal of Computational Science* 2(2):105–112, May 2011.
doi:10.1016/j.jocs.2011.01.004

[FGH+21] C. Feld, M. Geimer, M.-A. Hermanns, P. Saviankou, A. Visser, B. Mohr. Detecting Disaster Before It Strikes: On the Challenges of Automated Building and Testing in HPC Environments. In Mix et al. (eds.), *Tools for High Performance Computing 2018 / 2019*. Pp. 3–26. Springer International Publishing, 2021.
doi:10.1007/978-3-030-66057-4_1

[GHJV95] E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

[GHW+23] W. F. Godoy, S. E. Hahn, M. M. Walsh, P. W. Fackler, J. T. Krogel, P. W. Doak, P. R. C. Kent, A. A. Correa, Y. Luo, M. Dewing. Software engineering to sustain a high-performance computing scientific application: QMCPACK. In *USRSE23 Conference Proceedings*. 2023.
doi:10.5281/ZENODO.10420938

[GLC+15] T. Gamblin, M. LeGendre, M. R. Collette, G. L. Lee, A. Moody, B. R. de Supinski, S. Futral. The Spack package manager: bringing order to HPC software chaos. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. SC '15, pp. 1–12. ACM, Nov. 2015.
doi:10.1145/2807591.2807623

[GSND20] A. Grannan, K. Sood, B. Norris, A. Dubey. Understanding the landscape of scientific software used on high-performance computing platforms. *The International Journal of High Performance Computing Applications* 34(4):465–477, Jan. 2020.
doi:10.1177/1094342019899451

[GWW+10] M. Geimer, F. Wolf, B. J. N. Wylie, E. Ábrahám, D. Becker, B. Mohr. The Scalasca performance toolset architecture. *Concurrency and Computation: Practice and Experience* 22(6):702–719, Mar. 2010.
doi:10.1002/cpe.1556

[HCN21] S. Hussain, K. Chicoine, B. Norris. Empirical Investigation of Code Quality Rule Violations in HPC Applications. In *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering*. EASE '21, pp. 402–411. ACM, June 2021.
doi:10.1145/3463274.3463787

[HLS+16] B. P. Hallissy, J. P. Laiosa, T. C. Shafer, D. H. Hine, J. R. Forsythe, J. Abras, N. S. Hariharan, C. Dahl. HPCMP CREATE-AV Quality Assurance: Lessons Learned by Validating and Supporting Computation-Based Engineering Software. *Computing in Science & Engineering* 18(1):52–62, Jan. 2016.
doi:10.1109/mcse.2015.136

[HMS+09] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, G. Wilson. How do scientists develop and use scientific software? In *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*. IEEE, May 2009.
doi:10.1109/secse.2009.5069155

[Int01] International Organization for Standardization. Software engineering — Product quality. 2001.
https://www.iso.org/standard/22749.html

[Int23] International Organization for Standardization. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Product quality model. 2023.
https://www.iso.org/standard/78176.html

[KB19] D. Kempf, P. Bastian. An HPC Perspective on Generative Programming. In *2019 IEEE/ACM 14th International Workshop on Software Engineering for Science (SE4Science)*. IEEE, May 2019.
doi:10.1109/se4science.2019.00008

[KC07] B. Kitchenham, S. M. Charters. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical report, Evidence-Based Software Engineering (EBSE), 2007.
https://www.researchgate.net/publication/302924724

[KHH14] M. Källén, S. Holmgren, E. T. Hvannberg. Impact of Code Refactoring Using Object-Oriented Methodology on a Scientific Computing Application. In *2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation*. IEEE, Sept. 2014.
doi:10.1109/scam.2014.21

[KK93] L. V. Kale, S. Krishnan. CHARM++: a portable concurrent object oriented system based on C++. In *Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*. OOPSLA '93. ACM, Oct. 1993.
doi:10.1145/165854.165874

[KKN13] I. Karlin, J. Keasler, R. Neely. LULESH 2.0 Updates and Changes. Technical report LLNL-TR-641973, Lawrence Livermore National Laboratory, Aug. 2013.
doi:10.2172/1090032

[KM13] B. Koteska, A. Mishev. Software Engineering Practices and Principles to Increase Quality of Scientific Applications. In Markovski and Gusev (eds.), *ICT Innovations 2012*. Pp. 245–254. Springer Berlin Heidelberg, 2013.
doi:10.1007/978-3-642-37169-1_24

[KMP18] B. Koteska, A. Mishev, L. Pejov. Quantitative Measurement of Scientific Software Quality: Definition of a Novel Quality Model. *International Journal of Software Engineering and Knowledge Engineering* 28(03):407–425, Mar. 2018.
doi:10.1142/s0218194018500146

[LBAS14] F. Löffler, S. R. Brandt, G. Allen, E. Schnetter. Cactus: Issues for Sustainable Simulation Software. *Journal of Open Research Software* 2(1):e12, July 2014.
doi:10.5334/jors.au

[Lum25a] C. Lummerzheim. Code for "Quality Characteristics for Software in HPC Environments: A Systematic Literature Review". 2025.
doi:10.5281/ZENODO.16986219

[Lum25b] C. Lummerzheim. Data for "Quality Characteristics for Software in HPC Environments: A Systematic Literature Review". 2025.
doi:10.5281/ZENODO.14898873

[MAB$^+$13] G. R. Mirams, C. J. Arthurs, M. O. Bernabeu, R. Bordas, J. Cooper, A. Corrias, Y. Davit, S.-J. Dunn, A. G. Fletcher, D. G. Harvey, M. E. Marsh, J. M. Osborne, P. Pathmanathan, J. Pitt-Francis, J. Southern, N. Zemzemi, D. J. Gavaghan. Chaste: An Open Source C++ Library for Computational Physiology and Biology. *PLoS Computational Biology* 9(3):e1002970, Mar. 2013.
doi:10.1371/journal.pcbi.1002970

[MJ23] C. Melone, S. Jones. Verifying Functionality and Performance of HPC Applications with Continuous Integration. In *Practice and Experience in Advanced Research Computing*. PEARC '23. ACM, July 2023.
doi:10.1145/3569951.3597557

[MR19] R. Milewicz, P. Rodeghero. Position Paper: Towards Usability as a First-Class Quality of HPC Scientific Software. In *2019 IEEE/ACM 14th International Workshop on Software Engineering for Science (SE4Science)*. IEEE, May 2019.
doi:10.1109/se4science.2019.00012

[PBC+08] J. Pitt-Francis, M. O. Bernabeu, J. Cooper, A. Garny, L. Momtahan, J. Osborne, P. Pathmanathan, B. Rodriguez, J. P. Whiteley, D. J. Gavaghan. Chaste: using agile programming techniques to develop computational biology software. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 366(1878):3111–3136, June 2008.
doi:10.1098/rsta.2008.0096

[PK04] D. E. Post, R. P. Kendall. Software Project Management and Quality Engineering Practices for Complex, Coupled Multiphysics, Massively Parallel Computational Simulations: Lessons Learned From ASCI. *The International Journal of High Performance Computing Applications* 18(4):399–416, Nov. 2004.
doi:10.1177/1094342004048534

[PMV+16] D. Pflüger, M. Mehl, J. Valentin, F. Lindner, D. Pfander, S. Wagner, D. Graziotin, Y. Wang. The Scalability-Efficiency/Maintainability-Portability Trade-Off in Simulation Software Engineering: Examples and a Preliminary Systematic Literature Review. In *2016 Fourth International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering (SE-HPCCSE)*. IEEE, Nov. 2016.
doi:10.1109/se-hpccse.2016.008

[PP16] D. Pflüger, D. Pfander. Computational Efficiency vs. Maintainability and Portability. Experiences with the Sparse Grid Code SG++. In *2016 Fourth International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering (SE-HPCCSE)*. IEEE, Nov. 2016.
doi:10.1109/se-hpccse.2016.007

[PQV07] T. Panas, D. Quinlan, R. Vuduc. Tool Support for Inspecting the Code Quality of HPC Applications. In *Third International Workshop on Software Engineering for High Performance Computing Applications (SE-HPC '07)*. IEEE, May 2007.
doi:10.1109/se-hpc.2007.8

[Pre05] President's Information Technology Advisory Committee. *Computational Science: Ensuring America's Competitiveness*. National Coordination Office for Information Technology Research & Development, 2005.

[Roa98] P. J. Roache. *Verification and Validation in Computational Science and Engineering*. Hermosa Albuquerque, NM, 1998.

[SA23] M. Sgambati, M. Anderson. Software Quality Assurance for High Performance Computing Containers. In *Practice and Experience in Advanced Research Comput-*

ing. PEARC '23. ACM, July 2023.
doi:10.1145/3569951.3593596

[SB12] M. Schmidberger, B. Brügge. Need of Software Engineering Methods for High Performance Computing Applications. In *2012 11th International Symposium on Parallel and Distributed Computing*. IEEE, June 2012.
doi:10.1109/ispdc.2012.14

[SSY+24] X. Sáez, A. Soba, J. V. Ylla Català, G. Saxena, M. Garcia-Gasulla, C. Morales, D. V. Dorca, M. Komm, A. Podolnik, J. Romazanov, E. Sánchez, J. L. Velasco, M. J. Mantsinen. The Advanced Computing Hub at BSC: improving fusion codes following modern software engineering standards. *Plasma Physics and Controlled Fusion* 66(7):075014, May 2024.
doi:10.1088/1361-6587/ad4589

[TCG+24] J. Tronge, J. Chen, P. Grubel, T. Randles, R. Davis, Q. Wofford, S. Anaya, Q. Guan. An HPC-Container Based Continuous Integration Tool for Detecting Scaling and Performance Issues in HPC Applications. *IEEE Transactions on Services Computing* 17(1):156–168, Jan. 2024.
doi:10.1109/tsc.2023.3337662

[VKB+21] C. C. Venters, S. A. Kocak, S. Betz, I. Brooks, R. Capilla, R. Chitchyan, L. Duboc, R. Heldal, A. Moreira, S. Oyedeji, B. Penzenstadler, J. Porras, N. Seyff. Software Sustainability: Beyond the Tower of Babel. In *2021 IEEE/ACM International Workshop on Body of Knowledge for Software Sustainability (BoKSS)*. IEEE, June 2021.
doi:10.1109/bokss52540.2021.00009

[WDA+16] M. D. Wilkinson, M. Dumontier, I. J. Aalbersberg, G. Appleton, M. Axton, A. Baak, N. Blomberg, J.-W. Boiten, L. B. da Silva Santos, P. E. Bourne, J. Bouwman, A. J. Brookes, T. Clark, M. Crosas, I. Dillo, O. Dumon, S. Edmunds, C. T. Evelo, R. Finkers, A. Gonzalez-Beltran, A. J. G. Gray, P. Groth, C. Goble, J. S. Grethe, J. Heringa, P. A. C. 't Hoen, R. Hooft, T. Kuhn, R. Kok, J. Kok, S. J. Lusher, M. E. Martone, A. Mons, A. L. Packer, B. Persson, P. Rocca-Serra, M. Roos, R. van Schaik, S.-A. Sansone, E. Schultes, T. Sengstag, T. Slater, G. Strawn, M. A. Swertz, M. Thompson, J. van der Lei, E. van Mulligen, J. Velterop, A. Waagmeester, P. Wittenburg, K. Wolstencroft, J. Zhao, B. Mons. The FAIR Guiding Principles for scientific data management and stewardship. *Scientific Data* 3(1), Mar. 2016.
doi:10.1038/sdata.2016.18

[Wis14] Wissenschaftsrat. Bedeutung und Weiterentwicklung von Simulation in der Wissenschaft. Position paper, 2014.

[WPM15] S. Wagner, D. Pflüger, M. Mehl. Simulation software engineering: experiences and challenges. In *Proceedings of the 3rd International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering*. SE-HPCCSE '15. ACM, Nov. 2015.
doi:10.1145/2830168.2830171