

Review

Not peer-reviewed version

---

# Comprehensive Review of Metrics and Measurements of Quantum Systems

---

[Hassan Soubra](#)\*, [Hatem Elsayed](#)\*, [Yousef Elbrolosy](#), Youssef Adel, Zeyad Attia

Posted Date: 8 April 2025

doi: 10.20944/preprints202504.0503.v1

Keywords: quantum metrics; quantum computing; benchmarks; quantum software; functional size measurement; COSMIC ISO 19761








Preprints.org is a free multidisciplinary platform providing preprint service that is dedicated to making early versions of research outputs permanently available and citable. Preprints posted at Preprints.org appear in Web of Science, Crossref, Google Scholar, Scilit, Europe PMC.

Copyright: This open access article is published under a Creative Commons CC BY 4.0 license, which permit the free download, distribution, and reuse, provided that the author and preprint are cited in any reuse.

Disclaimer/Publisher's Note: The statements, opinions, and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions, or products referred to in the content.

Review

# Comprehensive Review of Metrics and Measurements<sup>1</sup> of Quantum Systems

Hassan Soubra<sup>1,†,\*</sup>, Hatem Elsayed<sup>2,†,\*</sup>, Yousef Elbrolosy<sup>3,†</sup>, Youssef Adel<sup>4,†</sup>  
and Zeyad Attia<sup>5,†</sup>

<sup>1</sup> hsoubra@ece.fr

<sup>2</sup> hatem.elsayed@tum.de

<sup>3</sup> yousefelbrolosy8@gmail.com

<sup>4</sup> yousefadel617@gmail.com

<sup>5</sup> zeyad.attia102@gmail.com

\* Correspondence: hsoubra@ece.fr (H.S.); hatem.elsayed@tum.de (H.E.)

† Current address: Department of Computer Engineering, Ecole Centrale d'Electronique-ECE Lyon, 24 rue Salomon Reinach, 69007 Lyon, France

‡ These authors contributed equally to this work.

**Abstract:** Quantum computing promises to offer significant computational advantages over classical computing, leveraging principles such as superposition and entanglement. This necessitates effective metrics and measurement techniques for evaluating quantum systems, aiding in their development and performance optimization. However, due to fundamental differences in computing paradigms, and current immaturity of quantum software abstractions, classical software and hardware metrics may not directly apply to quantum computing, where the distinction between software and hardware can still be somewhat indiscernible compared to classical computing. This paper provides a comprehensive review of existing quantum software and hardware metrics in the scientific literature, highlighting key challenges in the field. Additionally, it investigates the application of Functional Size Measurement (FSM) based on COSMIC ISO 19761 to measure quantum software. Three FSM approaches are analyzed by applying them to Shor's and Grover's algorithms, with measurement results compared to assess their effectiveness. A comparative analysis highlights the strengths and limitations of each approach, emphasizing the need for further refinement. The insights from this study contribute to the advancement of quantum metrics, especially software metrics and measurement, paving the way for the development of a unified and standardized approach to quantum software measurement and assessment.

**Keywords:** quantum metrics; quantum computing; benchmarks; quantum software; functional size measurement; COSMIC ISO 19761

## 1. Introduction

Quantum computing promises unprecedented computational advantages and speedups over classical computing, leveraging principles such as superposition and entanglement [1]. These capabilities enable quantum computers to solve certain problems exponentially faster than their classical counterparts. As quantum hardware advances at a rapid pace, it becomes increasingly essential to develop robust, efficient, and reliable quantum software [2]. To achieve this, we need effective tools and methods to evaluate quantum software, which is the role of metrics. Thus, understanding how to assess quantum software is crucial for the future of quantum computing.

**Computer hardware metrics** are essential for evaluating system performance, diagnosing issues, and optimizing resource utilization in classical computing environments ranging from personal computers to large-scale data centers. These metrics include CPU usage, memory consumption, disk I/O, network throughput, temperature, power consumption, and GPU load, all of which help ensure system reliability, security, and efficiency [3]. Among these, execution time—the duration a system

takes to complete a specific task—is a crucial performance indicator that directly impacts efficiency in software execution, real-time processing, and benchmarking. Execution time is influenced by several factors, including CPU clock speed, core count, cache size, RAM speed, disk type, and instruction set architecture, making it a key metric in high-performance computing and energy-efficient system design [4]. By utilizing profiling tools like Intel VTune, NVIDIA Nsight, or benchmarking software, developers and system administrators can analyze execution time to improve system responsiveness, reduce power consumption, and enhance user experience in applications ranging from gaming to artificial intelligence [5].

Transitioning from classical hardware metrics to quantum systems, Quantum hardware is evaluated today using a range of performance metrics that determine its computational capability and practical limitations. For example, Quantum Volume (QV), introduced by IBM, measures a quantum system's overall capability by considering factors such as gate fidelity, qubit connectivity, and circuit depth, providing a standardized benchmark for comparing different quantum processors [6]. Circuit Layer Operations Per Second (CLOPS) assesses the speed at which a quantum processor can execute quantum circuits, reflecting its real-time computational efficiency [7]. Another crucial metric, qubit coherence time, represents the duration a qubit retains its quantum state before decoherence, which directly impacts the feasibility of running deep quantum circuits [8]. Gate fidelity quantifies the accuracy of quantum gate operations by comparing their actual performance against ideal theoretical models, ensuring the reliability of quantum computations [9]. Additionally, error rates measure noise levels that introduce inaccuracies in quantum calculations, affecting the stability and precision of quantum algorithms [10]. Collectively, these metrics shape our understanding of quantum processor capabilities and their suitability for various quantum algorithms.

**Software metrics** provide a mathematical framework for mapping the entities of a software system to numerical values [11]. These metrics allow for the assessment of classical software systems by extracting relevant entities and calculating corresponding metric values. A software metrics tool implements these metric definitions, facilitating automated evaluations.

The term "software metrics" encompasses a wide range of activities related to measurement in software engineering. These activities include generating numbers that characterize properties of software code, developing models that predict software resource requirements and quality, and quantifying aspects of quality control and assurance, such as recording and monitoring defects during development and testing. While software metrics offer many benefits, their most significant contribution lies in providing information to support managerial decision-making during the software development phase[12].

However, applying classical software metrics to quantum systems presents unique challenges due to the distinct nature of quantum computing. It is unclear how well classical methods can be adapted to quantum contexts, or whether entirely new metrics should be developed. The core challenge stems from quantum phenomena like superposition, entanglement, and quantum interference, which introduce complexities absent in classical computing. These fundamental properties introduce complexities that diverge from classical computing paradigms [9,13], necessitating novel approaches to software measurement. Furthermore, the rapidly evolving landscape of quantum hardware, coupled with resource constraints and scalability issues, further complicates the task of assessing quantum software performance [14].

**Measurement in software engineering** has traditionally served as a vital tool for evaluating performance, complexity, and maintainability, enabling developers to ensure the quality and efficiency of their systems [15–17]. These metrics allow for the quantitative assessment of aspects such as code size, maintainability, and system performance, helping developers to predict software behavior and optimize system functionality. However, these classical software metrics often fall short when applied to quantum software due to the fundamental differences in architecture, computation, and execution models [18,19]. As a result, there is an emerging need to establish a dedicated framework for quantum software measurement, which can cater to these distinctive features.

Current research in this field remains in its infancy, with scholars like Sicilia et al. (2022) [18] emphasizing the lack of comprehensive approaches for measuring quantum software artifacts and processes. Without well-defined metrics, developers and researchers are left without the tools to assess the effectiveness of quantum programs, hindering progress in this rapidly evolving domain. Furthermore, Zhao [19] argues for the development of size and structure metrics specifically designed for quantum software, which can enable more precise and structured evaluations across different levels of software abstraction.

Functional Size Measurement (FSM) is a standardized approach used in software engineering to quantify the functional requirements of a system based on the services it provides to its users. FSM evaluates software by assessing its functionality from the user's perspective, focusing on what the software does rather than how it is implemented. The functional size is typically expressed in function points, a unit that enables developers and project managers to estimate project scope, cost, and size in a technology-independent manner [20]. FSM has been formalized in various international standards, including COSMIC ISO 19761, which is recognized for its applicability across a wide range of software types and domains [21]. By providing a consistent measure, FSM allows for accurate benchmarking, project estimation, and performance evaluation, helping to improve software development practices and outcomes.

To date, only three significant studies [22–24] have explored the use of functional size measurement techniques tailored to quantum software. All three studies were based on the COSMIC method – ISO 19761 [21], a well-established functional size measurement approach for classical software, and have sought to adapt this classical measurement approach to the unique characteristics of quantum systems.

This paper aims to explore the multifaceted challenges in quantum software measurement as documented in the scientific literature. Throughout this exploration, we investigate current methodologies, metrics, and tools for quantum software measurement, discussing their implications for software assessment. Furthermore, we address the challenges that arise due to the different nature of quantum computers, highlighting opportunities for innovation and enhancement. In addition, we apply three COSMIC-based Functional Size Measurement approaches to different quantum algorithms: Shor's algorithm [25] for factoring, Grover's algorithm [26] for search. By comparing the results across these diverse quantum algorithms, we aim to assess the applicability and limitations of each measurement approach.

The paper is structured as follows: Sections 2 and 3 provide overviews of classical hardware and software metrics, respectively. Section 4 examines quantum metrics discussed in the scientific literature. Section 5 explores the challenges in developing quantum software metrics. Section 6 focuses on COSMIC-based measurement approaches for quantum software. Finally, Section 7 presents conclusions and future directions.

## 2. Overview of Classical Hardware Metrics

Classical hardware (HW) metrics serve as fundamental indicators for assessing computing system performance, resource efficiency, and reliability. These metrics enable system designers, engineers, and administrators to monitor and optimize both hardware and software interactions, ensuring optimal functionality across various computing environments [3].

### 2.1. Commonly Used Hardware Performance Metrics

- **CPU Utilization:** Measures the percentage of processing power used at a given time, indicating workload efficiency and potential bottlenecks [27].
- **Memory Usage:** Tracks the amount of RAM occupied by running processes, affecting system responsiveness and multitasking capabilities [4].
- **Disk I/O (Input/Output):** Evaluates the read and write speeds of storage devices, influencing application load times and data retrieval performance [28].



- **Network Throughput:** Determines data transfer rates over a network, crucial for cloud computing, real-time streaming, and distributed systems [29].
- **Temperature and Power Consumption:** Helps assess system cooling efficiency and energy usage, essential for sustainable computing and high-performance workloads [30].
- **GPU Load:** Measures graphical processing unit (GPU) utilization, significant for gaming, artificial intelligence (AI), and high-performance computing [31].

Each of these metrics is measured using specialized tools and methodologies tailored to different computing needs.

## 2.2. Measurement Techniques and Tools

Accurate measurement of hardware metrics relies on profiling tools, benchmarking software, and real-time monitoring utilities:

- **Profiling Tools:** These include Intel VTune for CPU performance analysis, NVIDIA Nsight for GPU workload profiling, and Perf for Linux-based performance monitoring [5].
- **Benchmarking Software:** Applications like SPEC CPU, Geekbench, and PassMark provide standardized performance tests for comparative hardware evaluation [32].
- **Real-time Monitoring Utilities:** Built-in system tools such as Windows Task Manager, Linux top and htop, and macOS Activity Monitor help track system resource usage dynamically [33].

## 3. Overview of Classical Software Metrics

In software engineering, for classical computers, metrics and measurements play a crucial role in evaluating the quality [14], complexity [34], and maintainability [35] of software systems. By quantifying various attributes of software artifacts, metrics provide valuable insights into the development process and facilitate informed decision-making. This introduction aims to elucidate the fundamental concepts of major classical software metrics and measurement techniques, laying the groundwork for understanding their significance in software development practices.

At its core, software metrics encompass quantitative measures used to assess different aspects of software products, processes, and resources. These metrics serve as objective indicators of software quality and performance, enabling stakeholders to monitor progress, identify potential risks, and prioritize improvement efforts. By applying systematic measurement techniques [36], software engineers can gain valuable insights into the characteristics and behavior of software systems, ultimately leading to more effective software development and management practices.

The literature review on software metrics encompasses various dimensions, including product metrics, process metrics, and resource metrics. Product metrics focus on evaluating the properties and characteristics of software artifacts, such as code complexity, size, and cohesion [19]. Process metrics, on the other hand, assess the efficiency and effectiveness of software development processes [22,24], providing insights into factors such as productivity, cycle time, and defect density. Resource metrics quantify the resources expended during software development, including time, effort, and cost.

Among the multitude of software metrics available, several key metrics have emerged as foundational components of software measurement. These metrics address critical aspects of software quality and performance, serving as essential tools for software engineers and project managers alike. Some of the major classical software metrics are lines of code (LOC), cyclomatic complexity, coupling and cohesion measures, and defect density [19].

### 3.1. Functional size measurement

Functional size measurement methods focus on quantifying the functional requirements of software systems, providing a standardized way to measure the size of software functionalities [37]. Unlike traditional lines of code-based metrics, functional size measurement methods assess software size based on the functionality delivered to users rather than the implementation details. This approach enables a more accurate and objective assessment of software size and effort estimation [38].

A second-generation functional size measurement method is the COSMIC (Common Software Measurement International Consortium) method- ISO 19761, which focuses on measuring the functional size of software from a user's perspective. COSMIC measures functional size regarding functional user transactions, representing discrete interactions between the user and the software system. By quantifying these transactions and considering factors such as data movements and processing logic, COSMIC provides a robust measure of software size independent of implementation technologies [38].

Other functional size measurement methods, such as NESMA (Netherlands Software Metrics Association) Function Point Analysis and Mark II, offer alternative approaches to quantifying software size based on functional requirements and user interactions [39].

By understanding the basic concepts of these major classical software metrics and measurement techniques, software professionals can enhance their ability to assess and improve software quality, productivity, and maintainability. Through the systematic application of software metrics, organizations can optimize their software development processes, mitigate risks, and deliver higher-quality software products to meet the evolving needs of stakeholders and users.

## 4. Literature Review On Quantum Metrics and Measurements <sup>2</sup>

Over the past decade, researchers have proposed several approaches for the measurement of both quantum hardware and software. This section presents the key Quantum metrics and measures found in the literature.

### 4.1. Quantum Hardware Metrics

- Quantum Volume (QV) [6,40] is a measure that focuses on low-level, hardware-related aspects of quantum devices. QV is used to benchmark noisy intermediate-scale quantum devices (NISQ) by taking into account all relevant hardware parameters: performance and design parameters. It also includes the software behind circuit optimizations and is architecture-independent, that is, it can be used on any system that can run quantum circuits. It quantifies the largest square quantum circuit—where the number of qubits in the circuit equals the number of gate layers—that a device can successfully implement with high fidelity. To determine a device's quantum volume, randomized square circuits of increasing size are generated, where each layer consists of a random permutation of qubit pairs followed by the application of random two-qubit gates. These circuits are compiled according to the hardware's constraints and executed repeatedly to evaluate the heavy output probability, which is the likelihood of measuring outcomes that occur more frequently than the median output in the ideal distribution. A circuit size is considered successfully implemented if this probability exceeds  $\frac{2}{3}$  on average. The quantum volume is then defined as  $2^n$  where  $n$  is the largest square circuit that meets this criterion. It is intended to be a practical way of measuring progress towards improved gate error rates in near-term quantum computing.
- Another recent hardware benchmark is the Circuit Layer Operations per Second (CLOPS) [7], which is more concerned with the performance of a Quantum Processing Unit (QPU). Wack et al. [7] identified three key attributes for quantum computing performance: quality, scale, and speed. Quality is the size of a quantum circuit that can be faithfully executed and is measured by the mentioned QV metric. Scale is measured by the number of qubits in the system. And for the Speed attribute, measuring CLOPS is introduced. The CLOPS measure is a derived one and is calculated as the ratio of the total number of QV layers executed to the total execution time. In general, the CLOPS benchmark is a speed benchmark that is designed to allow the system to leverage all quantum resources on a device to run a collection of circuits as fast as possible.
- Qpack Scores, presented by Donkers et al., 2022 [41], is another benchmark that aims to be application-oriented and cross-platform for quantum computers and simulators. It provides a quantitative insight into how well Noisy Intermediate Scale Quantum (NISQ) systems can perform. In particular, systems that make use of scalable Quantum Approximate Optimization Algorithm (QAOA) and Variational Quantum Eigensolver (VQE) applications. QPack scores

comprise different sub-scores. The need for sub-scores arose from the fact that some quantum computers may be fast but inaccurate, while other systems are accurate but have longer execution times, which led to the realization that quantum systems cannot be benchmarked solely on a single factor. Qpack combines the following 4 sub-scores to form the overall sub-score. Runtime, Accuracy, Scalability, and Capacity. Runtime calculates the average execution time of quantum circuits on quantum computers. The Accuracy sub-score measures how close the actual outcome of an execution compares to an ideal quantum computer simulator. Scalability provides insight into how well the quantum computer can handle an increase in circuit sizes. Finally, the Capacity score indicates how usable the qubits in a quantum computer are compared to the actual number of qubits provided. These Benchmarks are run on different quantum computers using the cross-platform library LibKet which allows for the collection of quantum execution data to be used as a performance indicator.

- Other research, [42] attempts to quantify environmentally induced decoherence in quantum systems, which is another important factor affecting the amount of error in quantum measurements. In particular, Fedichkin et al. [42] tackled the problem from a physics and a mathematical point of view, exploiting approaches based on the asymptotic relaxation time scales and fidelity measures of decoherence, ultimately formulating an approach to measure the deviation of the actual density matrix, from the ideal one describing the system without considering environmental interactions.
- In "A Functional Architecture for Scalable Quantum Computing" Sete et al. 2016, [43] presented a functional architecture for superconducting quantum computing systems as well as introduced the total quantum factor (TQF) as a performance metric for a quantum processor. To calculate TQF, the average coherence time of all qubits on the chip is divided by the longest gate time in the qubits universal set and is then multiplied by the number of qubits. TQF gives rough estimates of the size of the quantum circuit that can be run before the processor's performance decoheres.

#### 4.2. Quantum Software Metrics

- Jianjun Zhao, in "Some Size and Structure Metrics for Quantum Software" [19], proposed some extensions of classical software metrics into quantum software metrics. Zhao first considered some Basic Size metrics. These metrics include Code Size, Design Size, and Specification Size. An example of Code Size metrics include Lines of Code (LOC), which is the most commonly used metric of source code program length. Secondly, for Design Size metrics, Zhao proposed the usage of quantum architectural description language (qADL) as an extension to classical architectural description language (ADL) to formally specify the architectures of a quantum software system. For the Specification Size metric, Zhao referred to Quantum Unified Modeling Language (Q-UML) as an extension to the general purpose, well-known Unified Modeling Language (UML). Basic *Structure* metrics were also considered. Examples include McCabe's Complexity Metric, as well as Henry and Kafura's information flow Metric.
- Finžgar et al., [44] introduced QUARK. The QUARK framework aims to facilitate the development of application-level benchmarks. Written in Python, it aims to be modular and extensible. The architecture of the framework comprises 5 components. Benchmark Manager, which orchestrates the execution of the benchmark. Application, which defines the workload, validation, and evaluation function. Mapping translates the application data and problem specification into a mathematical formulation that is suitable for a solver. The solver is responsible for finding feasible and high-quality solutions to the proposed problem. And Device, which is the quantum device the problem would be run on. After executing the benchmarks, QUARK collects the generated data and executes the validation and evaluation functions. In general, it tries to ensure the reproducibility and verifiability of solutions by automating and standardizing benchmarking systems' critical parts.
- In their endeavor to enhance the comprehensibility and accessibility of quantum circuits, J. A. Cruz-Lemus et al. [45] introduce a set of metrics that are categorized into distinct classes. The initial classification concerns circuit size, delineating circuit width as the number of qubits and

circuit depth as the maximal count of operations on a qubit. Subsequently, the concept of circuit density is introduced, quantified by the number of gates applied at a specific step to each qubit, with metrics encompassing maximum density and average density. Further classifications pertain to the number of gates within the circuit, commencing with single-qubit gates. Within this domain, metrics are proposed to enumerate the quantity of Pauli X, Y, and Z gates individually, alongside their collective count, the quantity of Hadamard gates, the proportion of qubits initially subject to said gates, the count of other gates, and the aggregate count of single-qubit gates. Moving forward, the authors introduce metrics to calculate the average, maximal quantity, and ratio of CNOT and TOFFOLI gates applied to qubits, in addition to their collective counts, as of multi-qubit gates. Aggregate metrics are composed of the total count of gates, the total count of controlled gates, and the ratio of single gates to total gates. Furthermore, oracles and controlled oracles are addressed, wherein their counts, averages, and maximal depths are identified, along with the ratio of qubits influenced by them. Lastly, measurements are addressed, highlighting the quantity and proportion of qubits measured, alongside the percentage of ancilla qubits utilized. The authors emphasize the need for validating this set of metrics to ascertain their comprehensiveness. Notably, the authors refrain from providing explicit elucidation on how these metrics may enhance the understandability of quantum circuits.

- Martiel et al.[46] introduces the Atos Q-score, a benchmark designed to evaluate the performance of quantum processing units (QPUs) in solving combinatorial optimization problems. The Q-score is application-centric, hardware-agnostic, and scalable, providing a measure of a processor's effectiveness in solving the MaxCut problem using a quantum approximate optimization algorithm (QAOA). The objective of the MaxCut problem is to identify a subset  $S$  of the vertex set of a graph in a way that maximizes the count of edges connecting  $S$  with its complement. The authors assess the Q-score, which involves running the QAOA for a MaxCut instance and evaluating its performance in both noisy and noise-free systems. They later mention that the Q-score can be used to assess the effectiveness of the software stack. Additionally, the authors discuss the scalability of the Q-score and its potential applications for tracking QPU performance as the number of qubits and problem sizes increases. Furthermore, they introduce an open-source Python package, for computing the Q-score, offering a useful resource for researchers and developers to measure the Q-score of any QPUs integrated with the library.
- Ghoniem et al. [47] aimed at measuring the performance of a quantum processor based on the number of quantum gates required for various quantum algorithms. Experimental evaluations were conducted on nine IBM back-end Quantum Computers, employing four different algorithms: a 5-Qubit entangled circuit algorithm, a 5-Qubit implementation of Grover's quantum search algorithm, a 5-Qubit implementation of Simon's quantum algorithm, and the 7-Qubit implementation of Steane's error correction quantum algorithm. Additionally, quantitative data on the number of quantum gates after the transpilation process using the default and maximum optimization settings were collected. This facilitated the analysis of how the availability and compatibility of different gate sets impact the total number of quantum gates needed for executing the various quantum algorithms. The default optimization level led to fewer gates in all algorithms except Simon's. Four of the nine backends exhibited a decrease in the number of gates when using the maximum level compared to the default optimization level.
- QUANTIFY [48] is an open-source framework for the quantitative analysis of quantum circuits based on Google's own Cirq framework. QUANTIFY includes as part of its key features, analysis and optimization methods, semi-automatic quantum circuit rewriting, and others. The architecture of QUANTIFY is designed in a classical workflow manner consisting of four-step types: (1) circuit synthesis, (2) gate level transpilation (e.g. translation from one gate set into another), (3) circuit optimization, and (4) analysis and verification. It also provides several metrics to verify the circuit under study, such as the number of Clifford+T (T-count), Hadamard, and CNOT gates, as well as the depth of the circuit and the Clifford+T gates (T-depth).



- In the rapidly evolving field of quantum software, it is paramount to ensure the presence of quality assurance measures, especially if quantum computers are to be used on large-scale real-world applications in the following years. This is what Díaz Muñoz A. et al. [49] set out to do by accommodating a series of classical quality metrics to the quantum realm, in addition to introducing novel measurements geared specifically towards hybrid (quantum/classical) systems maintainability. The paper expands upon the work done by J. A. Cruz-Lemus et al. [45] to develop a set of quantum software-specific metrics in 2 main areas. First, they introduce numerous metrics for gate counts that were not mentioned in the original work, such as counts for S, T, RX, RY, RZ, CP, and Fredkin gates among others. Second, they explore the Number of initialization and restart operations via the following metrics: Number and Percentage of qubits with any reset (to 0 state) or initialize (to arbitrary state) operations.
- SupermarQ [50] attempts to methodically adapt benchmarking methodology from classical computing to the quantum one. While the work is primarily focused on quantum benchmark design, the authors do come up with 6 metrics termed feature vectors to evaluate quantum computer performance on said benchmarks. Program communication, which ranges from zero for sparse connections to nearly one for dense connections, aims to measure the amount of communication required between qubits. Meanwhile, Critical-Depth, representing the circuit's minimal time, focuses on two-qubit interactions, larger numbers indicating greater serialization. The significance of entanglement in a given circuit, known as the Entanglement Ratio, is also captured by measuring the proportion of 2 qubit interactions. Parallelism quantifies the amount of parallel operations that a certain algorithm maintains without being majorly affected by correlated noise events (cross-talk). Applications with high parallelism pack many operations into a limited circuit depth; as a result, their parallelism characteristic is near to 1. Qubit activity is captured throughout execution by Liveness, which shows the frequency of active vs idle qubits. Lastly, Measurement emphasizes reset operations and mid-circuit measurement, relating their frequency to the overall depth of the circuit.

## 5. Challenges Facing the Development of Quantum Software Metrics

This section delves into the key challenges facing the development of quantum software metrics, exploring the unique obstacles posed by quantum systems and some of the limitations of the existing approaches outlined above.

To begin our discussion, we first consider hardware-oriented metrics, their merits, their limitations and thus the need for software-oriented metrics as well. Hardware-oriented metrics like QV [6,40] and CLOPS [7] account for critical factors such as circuit depth, error rates, and execution speed, which are essential for understanding the capabilities of quantum devices. For example, Quantum Volume (QV) evaluates the largest random square circuit a device can reliably execute, incorporating both the number of qubits and the effective error rate. This provides a holistic measure of a device's ability to handle complex computations while accounting for noise and errors. Similarly, CLOPS measures the speed of quantum circuit execution, reflecting how efficiently a Quantum Processing Unit (QPU) can process quantum operations. This is particularly useful for applications requiring rapid iterations, such as variational quantum algorithms. QPack Scores [41] goes a step further by combining runtime, accuracy, scalability, and capacity into a single benchmark, offering a more nuanced evaluation of a quantum system's performance across different dimensions.

These metrics also draw analogies to classical computing benchmarks, such as FLOPS (Floating Point Operations Per Second) and LINPACK, which measure the performance of classical processors and supercomputers. By providing standardized ways to compare quantum devices, QV and CLOPS help bridge the gap between classical and quantum computing, offering familiar frameworks for performance evaluation. Additionally, metrics like QV explicitly consider the interplay between circuit depth and error rates, which are critical for assessing the feasibility of running quantum algorithms on noisy intermediate-scale quantum (NISQ) devices. For instance, QV incorporates the effective

error rate and the achievable circuit depth, providing a measure of how well a device can maintain coherence and accuracy as circuit complexity increases. Equation (1) [6,40] defines how the quantum volume metric quantifies the space-time volume occupied by a model circuit with random two-qubit gates that can be reliably executed on a given device; where  $n$  is the number of qubits required to run a given algorithm,  $n'$  the number of active qubits,  $\epsilon_{\text{eff}}$  the effective error rate which specifies how well a device can implement arbitrary pairwise interactions between qubits, and the achievable circuit depth  $d$  is given to be  $\simeq \frac{1}{n\epsilon_{\text{eff}}}$ .

$$V_Q = \max_{n' \leq n} \min \left[ n', \left( \frac{1}{n' \epsilon_{\text{eff}}(n')} \right)^2 \right] \quad (1)$$

One major challenge facing these approaches is the probabilistic nature of quantum systems. Quantum computers produce probabilistic results due to the inherent uncertainty in quantum mechanics. This poses a problem for metrics like QV and QPack Scores, which rely on comparing outcomes to ideal simulations. The lack of repeatability in quantum results means that benchmarks may yield inconsistent results across multiple runs, complicating the evaluation process. Additionally, hardware-oriented metrics are often tailored to specific device architectures and gate sets, making them less applicable to other platforms. For example, QV assumes a specific circuit structure (random square circuits) that may not reflect the requirements of real-world applications, while CLOPS focuses on execution speed but does not account for the quality of results, which can vary significantly depending on the algorithm and hardware noise.

More importantly, however is the lack of software considerations of these approaches. Hardware-oriented metrics like QV, CLOPS, and QPack Scores fail to address the unique challenges of quantum software development. For instance, Quantum Volume (QV) measures the largest random square circuit a device can reliably execute but does not provide any insights into the efficiency or quality of the algorithms running on the device. A high QV score does not guarantee that a quantum algorithm will perform well or produce accurate results, as it does not account for the specific requirements of the algorithm or the effectiveness of error mitigation techniques.

Similarly, CLOPS measures the speed of quantum circuit execution but does not evaluate the quality of the results or the efficiency of the underlying algorithms. A device with high CLOPS might execute circuits quickly but produce inaccurate results due to high error rates or poor error mitigation. This lack of consideration for software quality limits the usefulness of CLOPS in assessing the overall performance of quantum systems.

QPack Scores attempt to address some of these limitations by combining runtime, accuracy, scalability, and capacity into a single benchmark. However, they still focus primarily on hardware performance and do not provide a comprehensive evaluation of quantum software. For example, QPack Scores do not account for the usability of quantum programming frameworks, the maintainability of quantum code, or the effectiveness of error mitigation techniques. These aspects are critical for the development of practical quantum applications but are overlooked by hardware-oriented metrics.

To fully realize the potential of quantum computing, it is essential to develop software metrics that address the unique challenges of quantum software development. These metrics should evaluate the efficiency, robustness, and maintainability of quantum algorithms, as well as the usability of quantum programming frameworks and the effectiveness of error mitigation techniques.

Since classical software metrics have been extensively researched and refined over decades, it is perhaps worth considering whether such measures could be extended to the quantum case. Indeed, it is possible to have already existing software metrics and simply modify them to account for some quantum attributes, such as the number of qubits, reads, and writes from quantum registers in addition to classical registers and so on.

Such an approach has been considered, evidenced by Jianjun Zhao [19], who suggested many notable extensions of classical software metrics, as discussed in the literature review above. The LOC measure for the quantum code shows how he attempted a coherent extension. Zhao divided the result into several aspects, accounting for the different aspects of a quantum computer. Those aspects are the

number of LOC, the number of LOC related to quantum gate operations, the number of LOC related to quantum measurements, the total number of qubits used, and the total number of unique quantum gates used.

Nevertheless, it must be acknowledged that programming on a quantum computer, albeit having some similarities to classical computing, is significantly different. One of the primary challenges in extending classical metrics to quantum software lies in the unique architecture and resource constraints of quantum systems. Quantum algorithms are executed on quantum circuits composed of qubits and quantum gates, with performance heavily influenced by factors such as qubit coherence times, gate fidelity, and error rates. Metrics like lines of code, which are designed to evaluate classical code structure, fail to capture the intricacies of quantum circuits, such as gate depth, entanglement patterns, or the trade-offs between circuit width and depth.

The approach of counting specific gates in quantum circuits, as seen in the works of J. A. Cruz-Lemus et al. [45], QUANTIFY [48], and Díaz Muñoz A. et al. [49], represents a significant step toward developing quantum-specific software metrics. By focusing on gate counts—such as the number of single-qubit gates (e.g., Pauli X, Y, Z, Hadamard), multi-qubit gates (e.g., CNOT, TOFFOLI), and specialized gates (e.g., Clifford+T, S, T, RX, RY, RZ)—these metrics aim to provide a quantitative basis for evaluating the complexity, efficiency, and resource usage of quantum circuits. However, while this approach offers a more quantum-specific perspective compared to classical metrics, it is not without its limitations and challenges.

Counting gates, while useful, does not fully capture the complexity or performance of a quantum circuit. Quantum circuits are influenced by factors such as qubit connectivity, gate fidelity, error rates, and the depth of the circuit (i.e., the number of sequential operations). A circuit with a low gate count but poor qubit connectivity might require extensive SWAP operations to execute, increasing its effective depth and runtime. Similarly, a circuit with a high number of low-fidelity gates might perform worse than a circuit with fewer but higher-fidelity gates. Gate-counting metrics alone fail to account for these nuances, potentially leading to misleading assessments of circuit efficiency or quality.

Furthermore, Quantum hardware platforms often support different native gate sets, which are the fundamental operations that a particular quantum device can execute natively and efficiently. For example, IBM's quantum processors typically use a gate set consisting of single-qubit gates (e.g., X, Y, Z, S, T, Hadamard) and the CNOT gate [51], while trapped-ion systems like those from IonQ may support native gates such as Mølmer-Sørensen gates for multi-qubit interactions [52]. Similarly, photonic quantum computers may rely on continuous-variable operations rather than discrete gate sets [53]. This diversity in hardware architectures means that a circuit optimized for one device may require significant transpilation—translation into a different gate set—to run on another device. As a result, gate counts can vary dramatically depending on the underlying hardware, making metrics like T-count or CNOT counts inconsistent and difficult to compare across platforms. For instance, a quantum circuit with a low T-count on one device might transpile into a circuit with a high number of native gates on another device, rendering the original metric meaningless in the new context.

This lack of universality resulting from the abundance of quantum hardware architectures is perhaps the most apparent challenge facing the development of quantum software metrics. In fact, the problem extends even further with various architectures being better at certain algorithms than others. In a study conducted by Linke et al. [54] they compare the performance of an Ion-Trap quantum computer with a superconducting quantum computer on some small quantum circuits. It was discovered that the ion-trap computer's increased connectivity allowed it to have a higher relative success rate on the Toffoli circuit (which had more two-qubit gates) than the Margolis circuit where both computers performed similarly. The authors concluded that the performance of an algorithm is intrinsically linked to the underlying hardware architecture hence any set of metrics aimed at measuring software performance would need to account for this.

Subsequently, it could be said that unlike classical computing, where hardware and software metrics are often treated as separate domains, quantum computing remains deeply intertwined with

its hardware due to the field's infancy and the unique challenges posed by quantum mechanics. The direct link between hardware and software in quantum systems means that any meaningful software metrics must account for the underlying hardware architecture, including qubit connectivity, gate fidelity, and error rates. This interdependence complicates the development of universal quantum software metrics, as performance can vary significantly across different devices and algorithms. As a result, quantum software metrics must be designed to bridge the gap between hardware capabilities and software requirements, ensuring they remain relevant across diverse platforms and applications.

One promising approach to addressing these challenges is the adaptation of functional size measurement methods, such as the COSMIC ISO 19761 standard[21], which is widely used in classical software engineering to measure the functional size of software systems. The COSMIC method focuses on quantifying the functionality delivered to users, making it potentially adaptable to quantum software by accounting for quantum-specific operations such as qubit manipulations, gate applications, and measurements. By extending this method to include quantum functionalities, it may be possible to develop a standardized framework for measuring the size and functionality of quantum algorithms in a hardware-agnostic manner. The following section showcases three different approaches developed on the COSMIC method for quantum software, highlighting its potential as a foundation for quantum software metrics.

## 6. Quantum Software Measurement Approaches Based on COSMIC ISO 19761

### 6.1. Overview of COSMIC ISO 19761

The COSMIC (Common Software Measurement International Consortium) ISO 19761 standard [21] is a widely adopted functional size measurement (FSM) method used for classical software. It was developed to provide a consistent and universal approach to measuring the functional size of software based on its functional user requirements (FURs). COSMIC is primarily focused on quantifying the data movements between the software and its users or external systems, making it a flexible and scalable model applicable to various types of software applications, including business applications, real-time systems, and now emerging domains like quantum software.

#### 6.1.1. Key Principles of COSMIC ISO 19761

The COSMIC method measures software size by analyzing functional processes, which are triggered by user inputs and lead to responses. A functional process consists of sub-processes known as data movements. These data movements are categorized into four main types:

1. **Entry (E):** Represents data entering the system from an external user or another system.
2. **Exit (X):** Involves data exiting the system, typically in response to an entry or internal process.
3. **Read (R):** Refers to the system accessing or retrieving data from a persistent storage.
4. **Write (W):** Indicates the storage or writing of data into persistent storage.

Each data movement reflects how the system interacts with its environment and is fundamental to defining the functional size of the software. The functional size is expressed in COSMIC Function Points (CFP), a unit that reflects the amount of functionality provided by the software based on its data movements. By abstracting the functional size, COSMIC enables comparison across different software projects and systems, independent of programming language or technology.

#### 6.1.2. Benefits of COSMIC ISO 19761

1. **Technology-Independent:** COSMIC is designed to be neutral. It can be applied to a wide variety of software domains, including traditional business systems, real-time systems, and more complex areas such as telecommunications, control systems, and embedded software. This adaptability holds promise for quantum systems, where standardized measurement approaches are still emerging.
2. **Scalability:** COSMIC is adaptable to both small and large systems, and it has been applied in diverse industries such as banking [55], telecommunications [56], and aerospace.



3. **Accuracy and Precision:** By focusing on functional user requirements and the actual interactions between the system and its users, COSMIC offers precise and repeatable measurements. This level of accuracy is crucial for project estimation, cost analysis, and performance benchmarking.

### 6.1.3. COSMIC in the Context of Quantum Software

Recent studies have adapted COSMIC for use in quantum systems by reinterpreting the traditional data movements (Entry, Exit, Read, Write) in the context of quantum processes [22–24]. For example, in quantum algorithms, the entry could correspond to initializing quantum states, while the exit could represent the collapse of quantum states into classical bits during measurement. As quantum computing progresses, frameworks like COSMIC offer a foundational approach to measuring software size in this new domain, enabling comparisons between classical and quantum systems. However, challenges remain in fully adapting COSMIC to account for the intricacies of quantum operations and algorithms, such as entanglement and non-local data movement.

### 6.2. *The three Quantum COSMIC Measurement Approaches*

To date, only three significant studies [22–24] have investigated the adaptation of functional size measurement specifically for quantum software, based on the COSMIC method—ISO 19761. We present a comparison of these three approaches in terms of the inputs used for measurement, functional processes, data movement types, and their respective units. All three approaches define the size of the circuit as the number of COSMIC Functional Points (CFP) of the identified functional processes. However, each of the three measurement approaches offers a slightly different technique as to what is considered in the circuit as a functional process.

#### 6.2.1. Approach 1 [22]: Gates' Occurrences

This approach treats each gate as a functional process, counting multiple occurrences of the same gate type as separate processes. It measures the COSMIC functional size in CFP by identifying data movements in Qiskit corresponding to operations by quantum gates and qubit measurements.

1. Identify each circuit as a system.
2. Identify each gate as a functional process and count gates.
3. Assign one Entry and Exit data movement per gate.
4. Identify measurement commands as Data Writes.
5. Identify reads from classical bits as Data Reads.
6. Assign 1 CFP for each Entry, Exit, Read, and Write.
7. Aggregate CFPs for each functional process to calculate its size.
8. Aggregate all CFPs to determine the system's functional size.

In Step 1, analyze each circuit separately, then aggregate CFPs across circuits. Steps 2-3 count gates and assign 2 CFPs per gate for Entry and Exit movements. Step 4 assigns 1 CFP per measurement (treated as a Write), and Step 5 adds 1 CFP per data Read (reading the result of the measurement). Finally, Step 8 aggregates all CFPs to calculate the software's total functional size.

#### 6.2.2. Approach 2 [23]: Gates' Types

This approach is differentiated from the first one in that it identifies types rather than occurrences. It accounts for each type of gate as a functional process, with functional process input and functional process output. This approach clearly distinguishes between ancillary qubits and input vectors, with the latter being groups of qubits defined by the size of the problem instance. Additionally, tensored operators acting on these input vectors are operators performing the same function, and those contributing to a shared task are grouped into their respective functional processes.

Directives for identifying the functional processes:

1. *An operator acts on qubits, and common action leads to an operator type. For instance, if a circuit contains an X gate acting on qubit  $q_0$  and also contains an X gate acting on qubit  $q_1$ , identify one X gate functional process type. Similarly, for a CX gate with qubit  $q_0$  controlling  $q_1$  and a CX gate with qubit  $q_1$  controlling  $q_0$ .*
2. *Identify an Input functional process that conveys the input state(s).*
3. *A vector (string of bits) and a separate value are considered different types.*
4. *Identify one functional process for all operators with a common action in the circuit.*
5. *Identify one functional process for n-tensored operator variants, identifying another functional process for each different n.*
6. *Identify one functional process type for each operator type defined by a human user (these operators are often indicated by  $U_f$  in the literature).*

It is important to now note several remarks for this approach, that would help better understand how to properly apply it.

For inputs, a vector (a string of bits) and a separate value are considered different types and each is considered a functional process (a different layer), with 1 Entry and 1 Write for each, resulting in 2 CFP per entry unless COSMIC measurement begins after state preparation of the qubits is finished. For example, if one input is a string of bits and another is a separate value, a separate Entry and Write are counted for each, totaling 4 CFP. Each gate, even if repeated, is a Functional Process considered as a layer with 1 Entry, x Reads, and x Writes, based on the number of wires entering and exiting; CNOT and swap gates involve 1 Read and 1 Write, as they likely read from one wire and operate on another. Measurement, if present, is a layer involving an Entry, Read, and Write on the classical register. For each Entry, Exit, Read, or Write, 1 CFP is counted for each.

### 6.2.3. Approach 3 [24]: Q-COSMIC

This approach is more abstract than the previous two approaches. It extends COSMIC to Q-COSMIC by adhering to the principles of Quantum-Unified Modelling Language (Q-UML), which is an attempt to generalize UML for quantum software projects to abstract implementation details. The authors argue that since COSMIC is a measurement technique applied to software design documents—most notably UML diagrams—it is natural to showcase Q-COSMIC through Q-UML diagrams.

The three phases of a Q-COSMIC analysis:

#### Strategy

1. *Determine the purpose of the COSMIC measurement.*
2. *Define the functional users and identify each as a functional process.*
3. *Specify the scope of the Functional User Requirements (FURs) to be measured.*

*After that, a context diagram is generated.*

#### Mapping

1. *Map the FURs onto a COSMIC Generic Software Model.*
2. *Identify the data movements needed for specific software functionality.*

#### Measurement

1. *Count and aggregate the data movements for each use case, with each movement representing functionality that contributes to the size of the software project/system.*

In the following subsections we will introduce the Quantum Algorithm classes to which we will compare and contrast the three COSMIC approaches we discussed above.

### 6.3. Quantum Algorithm Classes

We focused on two key classes of quantum algorithms as defined by Nielsen & Chuang (2010) [9]: Quantum Fourier Transform (QFT) and Quantum Search, based on their prominence in quantum computing. While these are significant, they represent relatively simple quantum algorithms, however, they were primarily chosen because they are widely known and provide foundational insights into quantum computing.

The following algorithms represent each class:

- Grover's Algorithm for the Quantum Search class, due to its efficiency in searching unstructured databases. Figure 1 shows the circuit implementing Grover's algorithm.
- Shor's Algorithm for the QFT class, since it is one of the most famous applications. Shor's algorithm uses the QFT to find patterns that help in factoring large numbers, solving a problem that classical computers find hard. This makes it a potential threat to modern encryption methods, which rely on the difficulty of factoring large numbers to secure data. Figure 2 is a simplified circuit implementing Shor's algorithm to find the prime factors of 21.

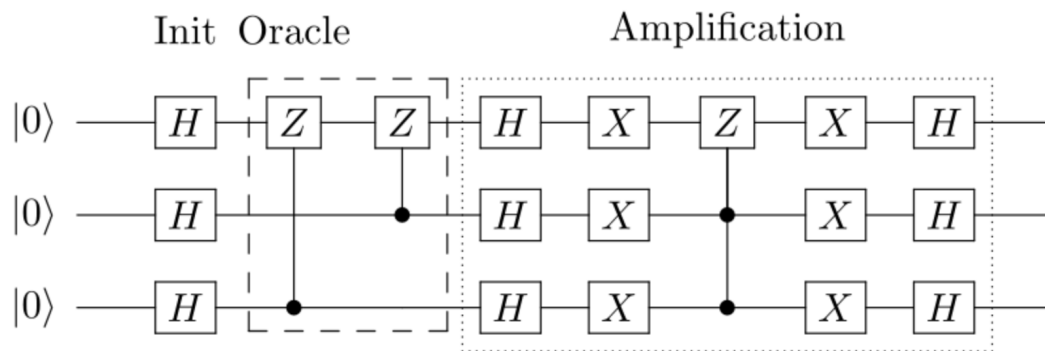


Figure 1. Grover's Algorithm. [57]

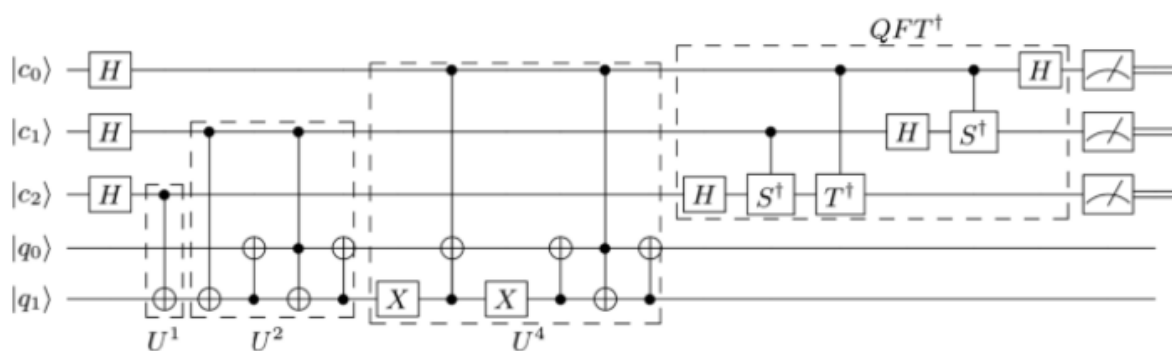


Figure 2. Shor's Algorithm to factor the number 21. [58]

### 6.4. Measurement Comparison

In this section, we investigate the measurement results of applying the different approaches when measuring the size of quantum software implementing Shor's algorithm and Grover's algorithm. Table 1 compares and summarizes the results of three approaches.

**Table 1.** A Summary of the Findings of the 3 approaches (Approach 3 according to [24]).

Approach	Data Movement Type	Grover's	Shor's
1	Entry	18	20
	Exit	18	20
	Write	3	3
	Read	3	3
	Total	42 CFPs	46 CFPs
2	Entry	5	9
	Write	5	9
	Read	4	7
	Total	14 CFPs	25 CFPs
3 UC1	Entry	NA	2
	Exit	NA	2
	QEntry	NA	1
	QExit	NA	1
	Total	NA	6 CFPs
3 UC2	Total	NA	4 CFPs

#### Approach 1 [22]: Gates' Occurrences

We applied this approach to the quantum circuit in Figure 1, which implements Grover's algorithm. The circuit consists of:

- 18 quantum gates, where 1 functional process is identified for each gate, with 1 entry and 1 exit for each process.
- 3 measurement operations, where we identified 1 write operation and 1 read operation for each (the measurement operation is omitted from the circuit for brevity).

We applied this approach to the quantum circuit in Figure 2, which implements Shor's algorithm to factor the number 21. The circuit consists of:

- 20 quantum gates, where 1 functional process is identified for each gate, with 1 entry and 1 exit for each process.
- 3 measurement operations, where we identified 1 write and 1 read for each.

#### Approach 2 [23]

First, we applied this approach to Grover's algorithm to the circuit in Figure 1.

- Input Functional Processes
  - Identify a functional process for the input vector. Count 1 Entry and 1 Write
- Operation Functional Processes
  - Identify 3 functional processes for the Hadamard, Controlled Z, and X gates acting on the input vector. Count 3 Entry, 3 Read, and 3 Write.
- Measurement Functional Processes
  - Identify a functional process for measuring the input vector. Count 1 Entry, 1 Read, and 1 Write.

Next, we apply the same approach to Shor's algorithm in Figure 2 as follows:

- Input Functional Processes
  - Identify a functional process for the input vector consisting of qubits (c0, c1, c2). Count 1 Entry and 1 Write.
  - Identify a functional process for the input vector consisting of qubits (q0, q1). Count 1 Entry and 1 Write.
- Operation Functional Processes



- Identify 3 functional processes for the Hadamard,  $S^+$ , and  $T^+$  gates acting on the input vector consisting of qubits (c0, c1, c2). Count 3 Entry, 3 Read, and 3 Write.
- Identify 2 functional processes for the CNOT and X gates acting on the input vector consisting of qubits (q0, q1). Count 2 Entry, 2 Read, and 2 Write.
- Measurement Functional Processes
  - Identify a functional process for measuring the input vector consisting of qubits (c0, c1, c2). Count 1 Entry, 1 Read, and 1 Write.
  - Identify a functional process for measuring the input vector of qubits (q0, q1). Count 1 Entry, 1 Read, and 1 Write.

### Approach 3 [24]: Q-COSMIC

The Q-COSMIC analysis of the factoring software using Shor's algorithm involves two use cases: UC1 (quantum) and UC2 (classical). UC1 has six total data movements: two Exits (2X), two Entries (2E), one Quantum Exit (1QX), and one Quantum Entry (1QE), resulting in 6 QCFP (Q-COSMIC Functional Points). UC2, a classical system, has 4 CFPv5. The total functional size is 10 points, with 60 % (6 CFP) from the classical layer and 40 % (4 QCFP) from the quantum layer.

### 6.5. Discussion

The three approaches to adapting the COSMIC method for quantum software—Approach 1 (Gates' Occurrences), Approach 2 (Gates' Types), and Approach 3 (Q-COSMIC)—offer distinct perspectives on measuring the functional size of quantum circuits. Each approach has its strengths and limitations, and their application to quantum algorithms like Grover's and Shor's reveals important insights into the challenges of quantifying quantum software size.

Approach 1: Gates' Occurrences treats each gate occurrence as a separate functional process, assigning Cosmic Functional Points (CFPs) based on data movements such as Entry, Exit, Read, and Write. For example, in Grover's algorithm, 18 gates result in 18 functional processes, each contributing 2 CFPs (Entry and Exit), while measurements add additional CFPs for Reads and Writes. This method is straightforward and granular, capturing every operation in the circuit. Its strength lies in its granularity, providing a detailed breakdown of every gate and measurement operation, which is useful for fine-grained analysis. However, this approach struggles with scalability, as the CFP count can become excessively high for large circuits with many gates. Additionally, it does not account for gate types or hardware-specific optimizations, potentially leading to inflated metrics for circuits with repeated gates.

Approach 2: Gates' Types groups gates by type rather than occurrences, treating each gate type as a functional process. For example, all Hadamard gates in a circuit are counted as a single functional process. This reduces redundancy and focuses on the diversity of operations rather than their frequency. The strength of this approach is its efficiency, as it reduces the CFP count by grouping similar gates, making it more scalable for larger circuits. It also emphasizes the functional intent of the circuit, aligning more closely with the purpose of the operations. However, it may lose detail by overlooking the impact of repeated operations, which could be significant in some algorithms. Additionally, identifying and grouping gate types requires a deeper understanding of the circuit's structure, which may not always be straightforward.

Approach 3: Q-COSMIC is the most abstract of the three, extending COSMIC to quantum software through Quantum-Unified Modelling Language (Q-UML). It focuses on functional processes at a higher level, such as input preparation, quantum operations, and measurements, rather than individual gates. For example, in Shor's algorithm, it distinguishes between quantum and classical layers, assigning CFPs based on data movements between these layers. The strength of this approach lies in its abstraction, providing a high-level view of the software's functionality, making it suitable for design and architectural analysis. It is also hardware-agnostic, as it focuses on functional processes rather than specific gates, making it adaptable across platforms. However, it may lack granularity,

as it does not capture the intricacies of gate-level operations, which are critical for performance optimization. Additionally, its reliance on Q-UML and higher-level abstractions may make it less accessible for developers focused on implementation details.

From the comparison, several key insights emerge. First, there is a trade-off between granularity and scalability. Approach 1 offers high granularity but struggles with scalability for large circuits, while Approach 2 strikes a balance by grouping gate types, reducing redundancy while maintaining functional relevance. Approach 3 sacrifices granularity for scalability and abstraction, making it more suitable for high-level design. Second, hardware dependence varies across the approaches. Approach 1 and Approach 2 are more sensitive to hardware-specific gate sets and optimizations, as they rely on gate-level details. In contrast, Approach 3, being more abstract, is less affected by hardware variations, making it more universal but less precise for size evaluation. Third, the approaches differ in their focus on functionality versus operational details. Approach 1 and Approach 2 focus on operational details (gates and measurements), which are critical for performance tuning, while Approach 3 emphasizes functional processes, aligning more closely with software engineering principles and design considerations.

Finally, the applicability to quantum algorithms varies across the approaches. For algorithms like Grover's and Shor's, Approach 1 and Approach 2 provide detailed insights into gate-level size, which is useful for optimization. Approach 3, while less detailed, offers a broader perspective on the algorithm's functional requirements, which is valuable for system design and integration. Together, these approaches demonstrate the need for a multi-layered framework that combines granular gate-level metrics with higher-level functional analysis, ensuring comprehensive evaluation of quantum software across different stages of development and deployment. This layered approach would address the unique challenges of quantum software while leveraging the strengths of each method.

## 7. Conclusions

This paper explored the challenges of developing quantum software metrics and evaluated three distinct adaptations of the COSMIC ISO 19761 method for measuring quantum software functional size. Given the inherent complexities of quantum computing—including hardware dependencies, probabilistic execution, and unique programming paradigms—establishing reliable software metrics is crucial for assessing quantum algorithm efficiency and scalability. Our analysis highlighted the shortcomings of hardware-oriented metrics, such as Quantum Volume (QV) and CLOPS, and examined how COSMIC-based approaches can offer a structured way to measure quantum software size.

### 7.1. Main Contributions

This work makes several key contributions to the field of quantum metrics, especially software metrics:

**Identifying the Gaps in Hardware-Oriented Metrics:** We outlined the limitations of existing hardware-driven metrics (e.g., QV, CLOPS, QPack Scores) and demonstrated their inadequacy in capturing quantum software characteristics such as maintainability, algorithmic efficiency, and functional size.

**Extending Classical Software Metrics and measurements to Quantum Computing:** We examined the feasibility of adapting classical software engineering metrics, particularly the COSMIC ISO 19761 method, to quantum software. We discussed the challenges in extending traditional metrics and highlighted the necessity of integrating quantum-specific attributes such as qubit usage, gate types, and measurement operations.

#### **Comparing Three COSMIC-Based Approaches for Quantum Software Measurement:**

**Approach 1: Gates' Occurrences** – Provides a fine-grained analysis by treating each gate as a separate functional process but suffers from scalability issues.

**Approach 2: Gates' Types** – Groups gates by type, reducing redundancy while maintaining relevance, offering a balance between detail and efficiency.

Approach 3: Q-COSMIC – Abstracts functional processes using Q-UML, making it hardware-agnostic and suitable for high-level software analysis but less precise for gate-level optimization.

**Demonstrating Applicability to Real Quantum Algorithms:** We applied these approaches to well-known quantum algorithms (Grover's and Shor's) to illustrate their effectiveness, revealing trade-offs between granularity, scalability, and hardware dependence.

These contributions lay the foundation for a structured, scalable, and functionally relevant measurement framework for quantum software, bridging the gap between classical and quantum software engineering.

## 7.2. Limitations and Future Work

While this study provides a significant step toward quantum software metrics, several challenges remain:

**Hardware Dependence and Standardization Challenges:** Gate-level metrics are highly sensitive to the underlying quantum hardware. As quantum computing platforms evolve, it will be necessary to develop metrics that remain meaningful across different architectures. This could involve establishing a standardized quantum software benchmarking framework.

**Integration with Quantum Software Development Tools:** To ensure practical adoption, quantum software metrics should be integrated into existing quantum programming environments (e.g., Qiskit, Cirq, PennyLane). Future work could focus on developing automated tools that compute COSMIC-based quantum metrics directly from quantum circuit representations.

**Expanding Functional Size Metrics for Hybrid Quantum-Classical Systems:** Many quantum algorithms rely on classical pre- and post-processing. Extending COSMIC-based approaches to hybrid quantum-classical workflows will be essential for measuring the full computational size of quantum applications.

**Empirical Validation through Real-World Case Studies:** The proposed measurement approaches should be tested on a broader range of quantum applications beyond Grover's and Shor's algorithms, including optimization problems, quantum machine learning, and error-corrected quantum systems.

The field remains in its early stages. There is a clear need for continued research to develop more precise and comprehensive measurement techniques that can support the evolving landscape of quantum software engineering. A standardized, scalable, and hardware-aware measurement framework would not only enhance the evaluation of quantum software performance but also facilitate benchmarking, optimization, and interoperability across different quantum platforms. As quantum computing moves closer to practical applications, establishing such a framework will be essential for ensuring the reliability, efficiency, and scalability of quantum software solutions in the future.

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

**Funding:** This research received no external funding

**Data Availability Statement:** The data used to support the findings of the study are available from the corresponding author upon reasonable request.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## Notes

<sup>1</sup> Unless stated otherwise, *Measurements* in this paper does not refer to the measurement of a quantum particle, which collapses its quantum state, but rather refers to concepts related to measuring the size and attributes of a quantum system.

- 2 Most of the metrics and measurement in the literature review in section 4 are specific to analyzing system properties such as size, performance, scalability and related characteristics. However, there exists, one might argue, a crucial test [59] that could be considered a metric as well. This test, which is device independent, certifies whether the system is indeed quantum. For instance, if a company claims to offer a Quantum Random Number Generator (QRNG) device, this test would verify whether the device truly harnesses the intrinsic uncertainties of quantum mechanical systems to generate randomness, rather than relying on classical processes.

## References

1. Horowitz, M.; Grumblin, E., Eds. *Quantum Computing: Progress and Prospects*; National Academies Press: Washington, DC, 2019.
2. Martonosi, M.; Roetteler, M. Next Steps in Quantum Computing: Computer Science's Role. *arXiv Preprint* **2019**, arXiv:1903.10541.
3. Hennessy, J. L.; Patterson, D. A. *Computer Architecture: A Quantitative Approach*, 6th ed.; Morgan Kaufmann: Cambridge, MA, 2017.
4. Stallings, W. *Computer Organization and Architecture*, 11th ed.; Pearson: Boston, MA, 2020.
5. Srinivasan, S.; Engel, G. *The Performance Analysis of Computer Systems: Techniques and Tools for Efficient Performance Engineering*; Springer: Cham, Switzerland, 2018.
6. Cross, A. W.; Bishop, L. S.; Sheldon, S.; Nation, P. D.; Gambetta, J. M. Validating Quantum Computers Using Randomized Model Circuits. *Phys. Rev. A* **2019**, *100*, 032328. <https://doi.org/10.1103/PhysRevA.100.032328>.
7. Wack, A.; Paik, H.; Javadi-Abhari, A.; Jurcevic, P.; Faro, I.; Gambetta, J. M.; Johnson, B. R. Quality, Speed, and Scale: Three Key Attributes to Measure the Performance of Near-Term Quantum Computers. *arXiv* **2021**.
8. Ballance, C. J.; Harty, T. P.; Linke, N. M.; Sepiol, M. A.; Lucas, D. M. High-Fidelity Quantum Logic Gates Using Trapped-Ion Hyperfine Qubits. *Phys. Rev. Lett.* **2016**, *117*, 060504. <https://doi.org/10.1103/PhysRevLett.117.060504>.
9. Nielsen, M. A.; Chuang, I. L. *Quantum Computation and Quantum Information*, 10th Anniversary ed.; Cambridge University Press: Cambridge, UK, 2010.
10. Preskill, J. Quantum Computing in the NISQ Era and Beyond. *Quantum* **2018**, *2*, 79. <https://doi.org/10.22331/q-2018-08-06-79>.
11. Lincke, R.; Lundberg, J.; Löwe, W. Comparing Software Metrics Tools. In *Proceedings of the 2008 International Symposium on Software Testing and Analysis*; ACM: New York, NY, USA, 2008; pp. 131–142.
12. Fenton, N. E.; Neil, M. Software Metrics: Successes, Failures, and New Directions. *J. Syst. Softw.* **1999**, *47* (2–3), 149–157. [https://doi.org/10.1016/S0164-1212\(99\)00014-1](https://doi.org/10.1016/S0164-1212(99)00014-1).
13. Savchuk, M. M.; Fesenko, A. V. Quantum Computing: Survey and Analysis. *Cybern. Syst. Anal.* **2019**, *55*, 10–21.
14. Mohanty, S. N. Models and Measurements for Quality Assessment of Software. *ACM Comput. Surv.* **1979**, *11* (3), 251–275. <https://doi.org/10.1145/356789.356792>.
15. Sommerville, I. *Software Engineering*, 9th ed.; Addison-Wesley: Boston, MA, USA, 2011.
16. Fenton, N.; Pfleeger, S. L. *Software Metrics: A Rigorous and Practical Approach*; CRC Press: Boca Raton, FL, USA, 2014.
17. Pressman, R. S. *Software Engineering: A Practitioner's Approach*, 8th ed.; McGraw-Hill: New York, NY, USA, 2014.
18. Sicilia, M. A.; Mora-Cantalops, M.; Sánchez-Alonso, S.; García-Barriocanal, E. Quantum Software Measurement. In *Quantum Software Engineering*; Serrano, M. A., Pérez-Castillo, R., Piattini, M., Eds.; Springer: Cham, Switzerland, 2022. [https://doi.org/10.1007/978-3-031-05324-5\\_10](https://doi.org/10.1007/978-3-031-05324-5_10).
19. Zhao, J. Some Size and Structure Metrics for Quantum Software. In *Proceedings of the 2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE)*, Madrid, Spain, 2021; pp. 22–27. <https://doi.org/10.1109/Q-SE52541.2021.00012>.
20. Abran, A.; Robillard, P. N. Function Points Analysis: An Empirical Study of Its Measurement Processes. *IEEE Trans. Softw. Eng.* **1996**, *22* (12), 895–909.
21. COSMIC. Functional Size Measurement – Method Overview. Available online: <https://cosmic-sizing.org> (accessed on 2024).
22. Khatlab, K.; Elsayed, H.; Soubra, H. Functional Size Measurement of Quantum Computers Software. Presented at IWSM-Mensura, 2022.



23. Lesterhuis, A. *COSMIC Measurement Manual for ISO 19761, Measurement of Quantum Software Circuit Strategy; A Circuit-Based Measurement Strategy*, 2024.
24. Valdes-Souto, F.; Perez-Gonzalez, H. G.; Perez-Delgado, C. A. Q-COSMIC: Quantum Software Metrics Based on COSMIC (ISO/IEC 19761). *arXiv Prepr.* **2024**, arXiv:2402.08505.
25. Shor, P. W. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Rev.* **1999**, 41 (2), 303–332.
26. Grover, L. K. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*; ACM: New York, NY, USA, 1996; pp. 212–219.
27. Patterson, D. A.; Hennessy, J. L. *Computer Organization and Design RISC-V Edition: The Hardware/Software Interface*; Morgan Kaufmann: Cambridge, MA, 2021. ISBN: 9780128245583
28. Tanenbaum, A. S.; Bos, H. *Modern Operating Systems, Global Edition*; Pearson Education: United Kingdom, 2023. ISBN:9781292727899, 1292727896
29. Dumitras, T.; Narasimhan, P. Why Do Upgrades Fail and What Can We Do About It?: Toward Dependable, Online Upgrades in Enterprise System Software. In *Lecture Notes in Computer Science*; Springer: 2009; Vol. 5927, pp. 97–112. [https://doi.org/10.1007/978-3-642-10445-9\\_18](https://doi.org/10.1007/978-3-642-10445-9_18).
30. Rusu, A.; Lysaght, P. Thermal Management in High-Performance Computing: Techniques and Trends. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* **2021**, 40 (11), 2154–2167.
31. Kirk, D. B.; Hwu, W. W. *Programming Massively Parallel Processors: A Hands-on Approach*, 3rd ed.; Morgan Kaufmann: Cambridge, MA, 2016.
32. Henning, J. L. SPEC CPU2000: Measuring CPU Performance in the New Millennium. *IEEE Comput.* **2000**, 33 (7), 28–35.
33. Silberschatz, A.; Galvin, P. B.; Gagne, G. *Operating System Concepts*, 10th ed.; Wiley: Hoboken, NJ, 2018.
34. Sinha, B. R.; Dey, P. P.; Amin, M.; Badkoobehi, H. Software Complexity Measurement Using Multiple Criteria. *J. Comput. Sci. Coll.* **2013**, 28 (4), 155–162.
35. Heitlager, I.; Kuipers, T.; Visser, J. A Practical Model for Measuring Maintainability. In *Proceedings of the 6th International Conference on the Quality of Information and Communications Technology (QUATIC 2007)*, Lisbon, Portugal, 12–14 Sept 2007; IEEE, 2007; pp 30–39.
36. Srinivasan, K. P.; Devi, T. A Comprehensive Review and Analysis on Object-Oriented Software Metrics in Software Measurement. *Int. J. Comput. Sci. Eng.* **2014**, 6 (7), 247.
37. Soubra, H.; Chaaban, K. Functional Size Measurement of Electronic Control Units Software Designed Following the AUTOSAR Standard: A Measurement Guideline Based on the COSMIC ISO 19761 Standard. In *Proceedings of the 2012 Joint Conference of the 22nd International Workshop on Software Measurement and the 7th International Conference on Software Process and Product Measurement*, Assisi, Italy, 17–19 Oct 2012; IEEE, 2012; pp 45–54.
38. Abran, A.; Symons, C.; Ebert, C.; Voegelzang, F.; Soubra, H. Measurement of Software Size: Contributions of COSMIC to Estimation Improvements. In *Proceedings of The International Training Symposium*, Marriott Bristol, United Kingdom, 2016; pp 259–267.
39. NESMA (Netherlands Software Metrics Association). *Nesma on Sizing - NESMA Function Point Analysis (FPA) Whitepaper*; 2018.
40. Bishop, L. S.; Bravyi, S.; Cross, A.; Gambetta, J. M.; Smolin, J. Quantum Volume. *Quantum Volume. Technical Report* **2017**.
41. Donkers, H.; Mesman, K.; Al-Ars, Z.; Möller, M. QPack Scores: Quantitative Performance Metrics for Application-Oriented Quantum Computer Benchmarking. *arXiv Preprint arXiv:2205.12142* **2022**.
42. Fedichkin, L.; Fedorov, A.; Privman, V. Measures of Decoherence. *Quantum Inf. Comput.* **2003**, 5105, SPIE.
43. Sete, E. A.; Zeng, W. J.; Rigetti, C. T. A Functional Architecture for Scalable Quantum Computing. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*; IEEE, 2016; pp 1–6.
44. Finžgar, J. R.; Ross, P.; Hölscher, L.; Klepsch, J.; Luckow, A. Quark: A Framework for Quantum Computing Application Benchmarking. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*; IEEE, 2022; pp 226–237.
45. Cruz-Lemus, J. A.; Marcelo, L. A.; Piatini, M. Towards a Set of Metrics for Quantum Circuits Understandability. In *International Conference on the Quality of Information and Communications Technology*; Springer International Publishing: Cham, 2021.
46. Martiel, S.; Ayral, T.; Allouche, C. Benchmarking Quantum Coprocessors in an Application-Centric, Hardware-Agnostic, and Scalable Way. *IEEE Trans. Quantum Eng.* **2021**, 2, 1–11.

47. Ghoniem, O.; Elsayed, H.; Soubra, H. Quantum Gate Count Analysis. In *2023 Eleventh International Conference on Intelligent Computing and Information Systems (ICICIS)*; IEEE, 2023.
48. Oumarou, O.; Paler, A.; Basmadjian, R. QUANTIFY: A Framework for Resource Analysis and Design Verification of Quantum Circuits. In *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*; IEEE, 2020.
49. Muñoz, A. D.; Rodríguez Monje, M.; Piattini Velthuis, M. Towards a Set of Metrics for Hybrid (Quantum/Classical) Systems Maintainability. *J. Univ. Comput. Sci.* **2024**, *30* (1), 25–48.
50. Tomesh, T.; Gokhale, P.; Omole, V.; Ravi, G. S.; Smith, K. N.; Viszlai, J.; Wu, X.-C.; Hardavellas, N.; Martonosi, M. R.; Chong, F. T. Supermarq: A Scalable Quantum Benchmark Suite. In *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*; IEEE, 2022; pp 587–603.
51. Shukla, A., Sisodia, M., Pathak, A. (2020). "Complete characterization of the directly implementable quantum gates used in the IBM quantum processors." *Physics Letters A*, 384(18), 126387.
52. Chen, J. S., Nielsen, E., Ebert, M., Inlek, V., Wright, K., Chaplin, V., ... Gamble, J. (2024). "Benchmarking a trapped-ion quantum computer with 30 qubits". *Quantum*, 8, 1516.
53. Kalajdzievski, T., Arrazola, J. M. (2019). "Exact gate decompositions for photonic quantum computing". *Physical Review A*, 99(2), 022341.
54. Linke, Norbert M., Dmitri Maslov, Martin Roetteler, Shantanu Debnath, Caroline Figgatt, Kevin A. Landsman, Kenneth Wright, and Christopher Monroe. "Experimental comparison of two quantum computing architectures." *Proceedings of the National Academy of Sciences* 114, no. 13 (2017): 3305-3310.
55. Souto, F. V., Pedraza-Coello, R., Olguín-Barrón, F. C. (2020). "COSMIC Sizing of RPA Software: A Case Study from a Proof of Concept Implementation in a Banking Organization". In *IWSM-Mensura*.
56. Bağrıyanık, S., Karahoca, A., Ersoy, E. (2015). "Selection of a functional sizing methodology: A telecommunications company case study". *Global Journal on Technology*, 7(7), 98-108.
57. Qiskit. Grover's Algorithm. Available online: <https://github.com/qiskit-community/qiskit-textbook/blob/main/content/ch-algorithms/grover.ipynb> (accessed on 2024).
58. Skosana, U.; Tame, M. Demonstration of Shor's Factoring Algorithm for  $n = 21$  on IBM Quantum Processors. *Sci. Rep.* **2021**, *11* (1). <https://doi.org/10.1038/s41598-021-95973-w>.
59. Pironio, S.; Acín, A.; Massar, S.; de La Giroday, A. B.; Matsukevich, D. N.; Maunz, P.; Monroe, C. Random Numbers Certified by Bell's Theorem. *Nature* **2010**, *464* (7291), 1021–1024. <https://doi.org/10.1038/nature09008>.

**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.