computers

MDPI

Article

An Investigation of Functional Size Measurement of Error Correction Overhead in Quantum Software

Youssef Adel [1], Hassan Soubra [2,3], Tuna Hacaloglu [3,*], Pierre Bourque [3], Sylvain Béland [4] and Alain Abran [3]

[1] Department of Computer Science and Engineering, German University in Cairo, New Cairo 11835, Egypt; youssefadel617@gmail.com
[2] Department of Computer Engineering, LyRIDS, ECE Engineering School, OMNES Education, 10 Rue Sextius Michel, 75015 Paris, France; hsoubra@ece.fr
[3] Department of Software and IT Engineering, École de Technologie Supérieure, 1100, Rue Notre-Dame Ouest, Montréal, QC H3C 1K3, Canada; pierre.bourque@etsmtl.ca (P.B.); alain.abran@etsmtl.ca (A.A.)
[4] Software Management Research Center, Doramante Montréal, Montréal, QC H3C 1K3, Canada; sylvain.beland@crgl.com
* Correspondence: tuna.hacaloglu@etsmtl.ca

Abstract

As quantum computing has progressed toward fault-tolerant implementations, quantum error correction (QEC) for quantum hardware and software has become a dominant concern. While conventional QEC measurements, such as qubit overhead, code distance, and error thresholds, quantify physical resource costs, they do not capture the software-level functionality introduced by error detection, correction, and fault-tolerant gate protocols. This study investigated the QEC contribution to the functional size of quantum software and demonstrated that this contribution can be systematically measured using the COSMIC Functional Size Measurement method. A gate-level sizing approach, known as Gates' Occurrences, was used to quantify functional user requirements (FUR) associated with QEC processes, including logical encoding, syndrome extraction, and recovery operations. A prototype measurement tool, implemented in Qiskit (Version 0.7.2), enables the automated calculation of COSMIC Function Points (CFP) from annotated circuit representations. Case studies involving Bell state preparation, Grover's search algorithm, and 15-to-1 magic state distillation revealed that QEC-related functionality can account for 70–85% of the total functional size, even under simplified encoding assumptions. The results underscore the importance of QEC for the functional size of quantum software. These findings offer a reproducible foundation for functional sizing and future fault-tolerant quantum applications.

Keywords: COSMIC Function Points; functional size measurement; quantum computing; quantum error correction; quantum software; Steane code; Shor code

Check for updates

Academic Editors: Paolo Bellavista and George Angelos Papadopoulos

Received: 21 July 2026
Revised: 9 October 2026
Accepted: 11 October 2026
Published: 15 October 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

1. Introduction

Quantum computers promise exponential speed increases in solving certain classes of computational problems, such as integer factorization [1], unstructured search [2], and quantum simulation [3]. However, their practical realization is hindered by the inherent fragility of quantum information. Qubits are highly susceptible to decoherence, gate imperfections, and quantum measurement errors, which can rapidly corrupt quantum states and invalidate computations. To overcome these challenges, quantum error correction (QEC) has emerged as a fundamental element in fault-tolerant quantum computing [4].

A fault-tolerant system ensures that a small number of errors do not propagate to corrupt the entire logical state, even when multiple operations are executed in parallel or in sequence [5]. Achieving fault tolerance imposes stringent constraints on circuit design and code architecture and remains a central challenge in scalable quantum computing. Far from being a peripheral overhead, QEC constitutes a dominant share of the physical and logical resources in most quantum computers. In many large-scale designs, more than 90% of gate operations are dedicated to the error detection, correction, and ancillary procedures necessary for fault-tolerant gate construction [6].

Incorporating robust quantum error correction into quantum computations has significant implications for quantum software. Encoding logical operations, managing syndrome extraction routines, and orchestrating corrective feedback requires precise specifications and tracking at the software level. As a result, a substantial fraction of the functionality present in quantum programs arises not from the functional user requirements (FUR) associated with the algorithm itself but from the error-handling infrastructure.

Although QEC codes such as the Shor code [7], Steane code [8], and surface codes [9] are typically evaluated using hardware-centric metrics, including qubit overhead, code distance, and threshold fidelity, they fail to account for the software-level implications of error correction. From a software engineering perspective, this omission created a measurement blind spot. If quantum error correction (QEC) operations comprise most of the gates in a circuit but are not included in functional size measurements, then the actual size of the software may be undercounted; subsequently, the effort and resources needed to develop and maintain it will notably be underestimated.

These demands highlight the need for systematic approaches to quantify quantum software size, particularly in a way that reflects both the functional behaviour and hardware constraints. In classical software engineering, the COSMIC method [10] offers a standardized method for measuring functional size based on data movement and user-defined functions.

This issue is becoming increasingly important as quantum computing evolves through distinct technological eras. In the current noisy intermediate-scale quantum (NISQ) era, most programs are short and prone to error and rely on mitigation rather than full correction. However, as the field transitions into early and eventually scalable fault-tolerant phases, QEC has become a dominant computational concern [11]. Software measurement methods should, therefore, account for this shift in functionality and operational emphasis.

As quantum programming ecosystems evolve with languages such as Q# [12], Qiskit [13], and Cirq [14], and as higher-level quantum software becomes more modular and abstracted, the use of software measurement methods should become prevalent. In this study, we adopted a software-centric perspective on quantum error correction (QEC). Although it is well understood that QEC codes involve a tight interplay between hardware and software, our treatment deliberately abstracts hardware away to focus exclusively on the software dimension. This abstraction enables the application of software engineering approaches, particularly software size measurements, to QEC. Despite the growing interest in quantum software engineering, no prior work has focused on measuring the functional size of quantum error correction from a software perspective. For the purpose of this study, we also consider QEC and quantum algorithms to be at the circuit level of abstraction.

In the research domain, there is limited attention to monitoring the unit cost of software applications developed within specific application contexts. This contrasts with industrial and business environments, where managers are concerned not only with direct and indirect costs but also with the value and benefits expected from development investments. In software engineering, several measurement methods have been developed to support management-oriented objectives. These methods enable the monitoring of trends in soft-

ware development efforts and costs and their underlying sources, such as the addition of new functionality to satisfy functional requirements and the incorporation of software functionality introduced to address non-functional requirements (NFRs), including QEC.

Accordingly, we aim on the one hand, to measure the size of software functionality devoted to quantum functional requirements and, on the other hand the size of software functionality devoted to QEC. The corresponding ratio can serve as a management-oriented indicator for monitoring development productivity, quality, and cost per unit of delivered functionality.

The objective of this exploratory study is to investigate the applicability of COSMIC functional sizing for quantifying QEC-related functionality, rather than testing a predefined quantitative threshold.

COSMIC, standardized in ISO/IEC 19761, provides a technology-agnostic framework for quantifying the size of software systems based on their functional user requirements from both system functional requirements and system non-functional requirements later assigned to software functions [15]. Previously, COSMIC was used to measure classical software in various domains such as automotive, banking, and telecommunications [16]. Its use in quantum software [17–19] opens new possibilities for new domains such as cryptography, materials science, pharmaceuticals, and finance [20].

In the COSMIC FSM Method, the first phase consists of specifying the measurement purpose and corresponding scope. In previous COSMIC-based quantum software sizing approaches [17–19], researchers primarily focused on measuring quantum functionality based on gate occurrences, gate types, or use cases. However, none explicitly quantified the contribution of Quantum Error Correction (QEC) within quantum circuits. This study differs from previous studies by extending the scope of functional size measurement to QEC and distinguishing between base functionality and QEC-related functionality, thereby enabling the quantification of QEC overhead at the software level.

For this investigation, we adopted the Gates' Occurrences approach [17], which maps quantum gates to the COSMIC functional processes. Using this method, we compute the COSMIC Function Points (CFP) by assigning Entry/Exit movements to gates and Read/Write movements to measurement operations. A prototype measurement tool built into Qiskit automates this process. We applied this approach to three representative circuits: Bell state preparation, Grover's algorithm analyzed under both Steane and Shor error-correcting codes [21], and a 15-to-1 magic state distillation protocol [22]. Each circuit was fully annotated to separate the base functionality and error correction overhead. The key contributions of this study are as follows:

1. Presentation of the first exploration of COSMIC Functional Sizing to quantum error correction.
2. A prototype measurement tool that supports automated COSMIC sizing of quantum circuits, including QEC, using the Gates' Occurrences model.
3. A series of case studies on quantum software with and without QEC.
4. Initial evidence shows that QEC dominates the functional size of quantum software.

The remainder of this paper is organized as follows. Section 2 discusses the quantum noise models and error sources, motivating the need for QEC codes. Section 3 provides a technical overview of QEC techniques, including stabilizer codes and fault-tolerant procedures. Section 4 outlines the COSMIC FSM method and its interpretation of quantum circuits. Section 5 presents the case studies and sizing results. Section 6 discusses the implications of the findings and ties them back to the research question. Section 7 mentions some of the limitations of the current study and proposes directions for future work. Finally, Section 8 concludes the study.

2. Quantum Noise and Error Sources

Quantum computers are inherently sensitive to external disturbances because of the fragile nature of the qubit quantum states. Unlike classical bits, which can stably represent either 0 or 1 with minimal error, quantum bits—or qubits—exist in delicate superpositions of both states [21]. This superposition and entanglement underpin the computational power of quantum systems. However, these properties are easily disrupted by various noise sources that can degrade, corrupt, or collapse the quantum information [23].

Quantum noise can be broadly categorized based on its origin and effect on computation. This categorization helps to motivate the structure of quantum error correction (QEC) schemes.

2.1. Coherent Noise

Coherent noise refers to systematic errors in quantum operations, such as over-rotations, under-rotations, and control inaccuracies during gate execution. These errors arise from deterministic imperfections in hardware or control systems, such as frequency drifts or crosstalk between qubits, and manifest as unintended but unitary transformations [24]. Because they are coherent, these errors accumulate over time and can lead to quadratic increases in the total error with circuit depth [25].

For example, consider a rotation gate that is consistently applied with a small deviation. In short circuits, the impact may be negligible; however, in longer circuits with many repetitions, this small imperfection compounds and causes the overall computation to deviate significantly from the intended outcome. This makes coherent noise particularly problematic in deep quantum algorithms that rely on the precise application of multiple gates [26]. Coherent errors are particularly evident in systems, such as superconducting qubits, in which gate operations are implemented using precisely timed microwave pulses. Minor imperfections in the pulse amplitude, shape, or duration can result in systematic errors that skew the behaviour of quantum gates throughout an entire algorithm [27].

2.2. State Preparation and Measurement (SPAM)

It should be noted that, unless otherwise stated, “measurement” in this paper refers to software sizing. This differs from quantum measurement, which gets the states of qubits (quantum bits) at the end of a computation to extract useful results.

State Preparation and Measurement (SPAM) noise encompasses all errors that occur during the initial preparation of the qubits and their final readout. These processes form a crucial interface between the classical and quantum worlds. Classical controllers prepare the quantum system in a desired initial state, and after computation, they extract classical bits from quantum measurements. However, in practice, neither of these steps is reliable.

During state preparation, a qubit may not properly initialize to the ground state, particularly if the reset mechanism is incomplete or slow, relative to the thermal environment of the system. For example, in superconducting qubits, reset is often achieved through thermal relaxation or active feedback, both of which may leave a non-negligible population in the excited state [28]. This means that a quantum algorithm might begin with a qubit already in the wrong state, which can cascade downstream into incorrect outcomes.

Meanwhile, measurement errors arise from imperfections in the detection mechanism used to read out the qubit states. For instance, in trapped-ion systems, state-dependent fluorescence is used for measurement; if the photon detectors have low quantum efficiency or if laser power fluctuates, the observed signal might not reliably reflect the true qubit state [29].

SPAM noise is particularly problematic because it introduces errors at the boundaries of quantum computations, which either corrupt the initial state or misrepresent the final

result. These are not correctable using conventional quantum error correction, which typically addresses errors that occur during gate operations. Consequently, SPAM noise must be characterized and mitigated independently through calibration routines, error mitigation strategies (such as readout error matrices), or even software-level post-processing to statistically correct for known biases [30].

2.3. Incoherent Noise

Incoherent noise refers to random, non-unitary errors that arise from uncontrolled interactions between a quantum system and its environment. These errors lead to irreversible loss of quantum information and are typically modelled using quantum channels such as amplitude and phase damping [28]. Unlike coherent noise, which results from systematic imperfections, incoherent noise stems from unpredictable processes, such as thermal fluctuations, electromagnetic interference, and spurious coupling in the device substrate.

The two key manifestations of incoherent noise are energy relaxation and dephasing. Energy relaxation, or T_1 decay, occurs when a qubit in the excited state spontaneously transitions to the ground state, typically because of thermal interactions with the environment. Dephasing or T_2 decay affects the relative phase of the superpositions without changing the energy state of the qubit [31]. This arises from random fluctuations in the local fields that alter the qubit energy levels over time. These processes degrade the quantum state and reduce the reliability of gate operations and measurement outcomes. As circuits become deeper or more complex, the probability that a qubit experiences incoherent noise before computation is significantly increased and becomes a central obstacle to scalable quantum computing.

2.4. Gate Imperfection Errors

Gate imperfection errors refer to inaccuracies in how quantum gates, the basic building blocks of quantum algorithms, are implemented in real hardware. Gate errors arise from multiple sources, including control signal imprecision, crosstalk between qubits, thermal fluctuations, and calibration drift. In ion-trap platforms, misalignment of laser beams or frequency drift can similarly degrade gate fidelity. These imperfections accumulate throughout an algorithm, particularly in deeper circuits, and become a major bottleneck in scaling quantum systems [32].

The accuracy of quantum gates is typically quantified using two measurements: gate fidelity and gate error rate. Gate fidelity measures how closely the implemented operation matches the ideal unitary transformation [33]. For instance, a fidelity of 99.9% suggests that the actual gate closely approximates the ideal gate in 999 out of 1000 executions. Conversely, the gate error rate is the probability that the operation deviates from the desired behaviour and introduces a computational error.

2.5. The Effect of Quantum Noise on Computation

As quantum computers scale toward more ambitious tasks, quantum noise remains the most significant bottleneck in achieving reliable and large-scale computation [6]. Quantum gates, which are the fundamental operations used in algorithms, are inherently imperfect. Although individual gate fidelities have reached 99.9% [34], even small error rates accumulate rapidly in deep circuits. For instance, a gate error rate of only $p = 10^{-3}$ becomes catastrophic when gates are repeated billions of times, which is typical for large-scale algorithms, such as Shor's factoring method [6]. Assuming independent gate failures with a constant per-gate error probability, the probability that all gates in a circuit execute correctly is given by. Consequently, the probability that at least one error occurs during the computation is. Figure 1 is generated using this analytical model with, corresponding to a gate fidelity of 99.9%, and assuming independent errors. Figure 1 shows how the

error probability scales with the number of gates. Without error correction, such noise accumulation renders the results of long computations practically unusable.

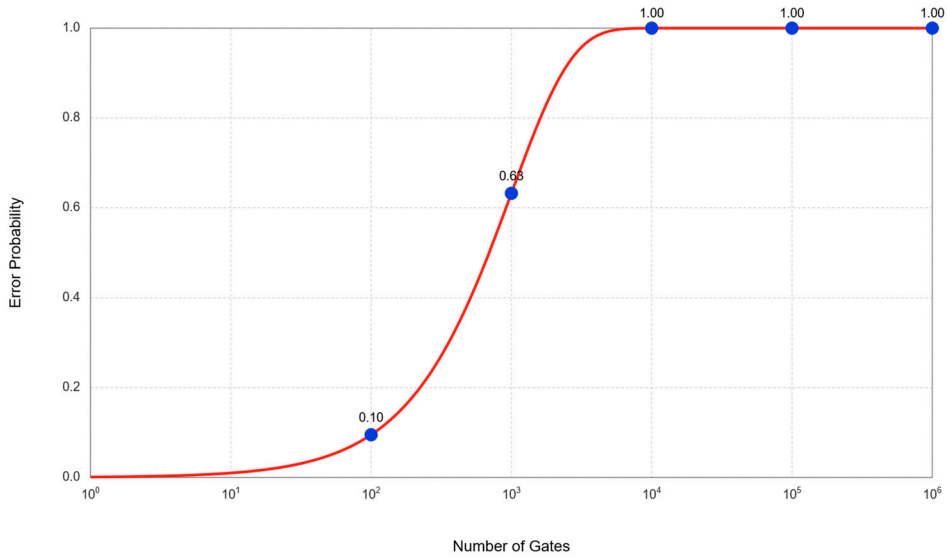

Figure 1. Error probability as a function of gate count, assuming an error rate of 10^{-3}.

The sensitivity of qubits to their environment further complicates this issue. Despite advances in shielding and fabrication, current superconducting qubits have coherence times of approximately 100 microseconds [34]. This is orders of magnitude shorter than the time required to perform complex quantum algorithms. For example, factoring a 1024-bit RSA number with Shor's algorithm would require approximately 7.7×10^{10} gates, which, at 1 nanosecond per gate, would require approximately 77 s [35]. Compared to the 100 microsecond coherence window, this computation would exceed the noise threshold by a factor of a million. Consequently, the quantum information decays long before the algorithm can be completed.

The combination of limited coherence time, frequent gate imperfections, and exponential scaling of quantum algorithm depth underlines why error correction is not just beneficial but also necessary. Without this, even moderately complex quantum computations remain out of reach. The next section explores how quantum error correction addresses these challenges and the resources it requires in terms of qubits, operations, and circuit overhead.

3. Background on Quantum Error Correction

Qubits are susceptible to decoherence, gate errors, and environmental interactions, all of which can corrupt computational outcomes. Therefore, reliable quantum computation critically depends on the development of robust error correction procedures that can detect and correct such errors without destroying the underlying quantum state. This section presents an overview of the foundational techniques in quantum error correction with an emphasis on the transition from classical redundancy-based codes to fully quantum stabilizer codes.

Reliable information processing in classical computing relies heavily on redundancy and error correction. An example is the repetition code, which protects a bit of information by encoding it into multiple copies, such as encoding a logical 0 as 000 and a logical 1 as 111 [36]. If a single bit is corrupted during storage or transmission, then the original value can be recovered using majority voting.

However, quantum systems do not permit such direct replication of information. The no-cloning theorem, a fundamental result of quantum mechanics, prohibits the copying of an arbitrary quantum state and thus precludes any naive adaptation of classical repetition codes to quantum settings [37]. In addition, quantum measurements are inherently destructive; extracting information from a qubit collapses its superposition, potentially destroying the information that one seeks to protect. Furthermore, although classical errors can typically be modelled as discrete events (e.g., bit flips), quantum errors are inherently continuous. A qubit may undergo rotation on the Bloch sphere in any direction. This implies that quantum noise introduces a much richer and more complex error model [21].

To address these challenges, early quantum error correction schemes introduced discretization strategies that reduced arbitrary quantum errors to combinations of bit-flip (X) and phase-flip (Z) errors. This discretization is possible because any small unitary error on a qubit can be expressed as a linear combination of Pauli operations, which enables the detection and correction of continuous errors through the correction of discrete syndromes [36]. However, because quantum states cannot be observed directly without collapse, error detection must be performed indirectly. This is achieved using ancillary qubits (ancillas) that interact with the data qubits in controlled operations designed to extract error information, known as error syndrome, without revealing or disturbing the encoded quantum information [38].

These ideas were concretely realized in early quantum repetition codes designed to handle specific error types. For instance, to protect against bit-flip errors, a logical qubit is encoded across three physical qubits in a symmetric superposition of classical repetition code states. Detection of a single bit flip is performed by checking parity relations between pairs of qubits using ancilla-mediated measurements; once an inconsistency is detected, a corrective operation can be applied. However, this code does not protect against phase flip errors, which can introduce relative sign changes in superpositions that remain undetectable on a computational basis [21].

To address phase-flip errors, a similar three-qubit code can be constructed by switching to a basis in which phase errors manifest as bit flips. This dual code protects against phase noise but not bit flips. The combination of these two strategies yields the Shor code [7], which is the first quantum code capable of correcting arbitrary single-qubit errors. The Shor code encodes a single logical qubit into nine physical qubits by first applying a phase-flip protection scheme to three blocks and then embedding each block in a three-qubit bit-flip repetition code. This concatenation enables the correction of both error types through sequential syndrome extraction and correction, first for bit flips and then for phase flips. Figure 2 shows an example of the encoding circuit for the 9-qubit code. H symbol in the circuit given in figures denotes the Hadamard gate.

Although the Shor code demonstrated the feasibility of full quantum error correction, its structure was designed for a specific type of protection. A more general and scalable framework was introduced through stabilizer formalism. This approach defines a code using a set of operators called stabilizers, which leave the encoded quantum state unchanged. Each stabilizer corresponds to a parity check analogous to those in classical coding theory, and the set of all such checks defines a code space: the collection of quantum states stabilized (i.e., unaffected) by all operators in the set. Errors map codewords to orthogonal subspaces can be identified by measuring the stabilizers, which reveals a syndrome pattern that guides correction [38].

The stabilizer framework generalizes the principles underlying earlier repetition-based constructions. In the three-qubit bit-flip code, for example, the parity between the first and second qubits and between the second and third qubits can be viewed as stabilizers. A bit flip on any of the qubits changes the outcome of these checks and generates a distinct

syndrome. Similarly, in the Shor code, the stabilizers check the parity conditions across both the phase- and bit-flip-protected subblocks. Importantly, the use of ancilla qubits to extract the values of stabilizers ensures that the encoded quantum information is never directly measured. Coherence is therefore preserved, even in the presence of repeated checks. This modularity and generality have made stabilizer codes the foundation for many advanced quantum codes, including surface codes and other topological constructions [9].

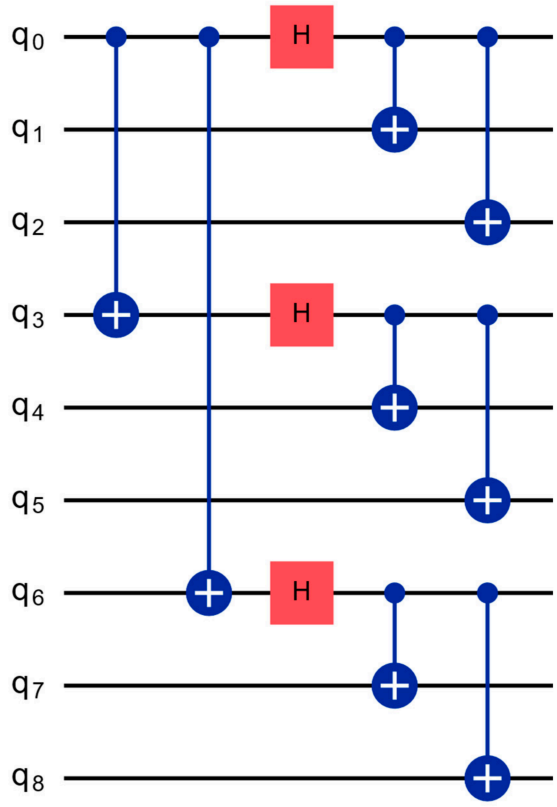

Figure 2. An Example of the encoding circuit for the 9-qubit code.

Logical operations on encoded quantum information must preserve the structure of the code space while manipulating underlying data. These operations, known as logical gates, act on logical rather than physical qubits, and their implementation must be compatible with the error correction scheme to avoid spreading errors. A desirable property of many quantum codes is their ability to implement certain logical gates transversally, which means that the operation acts independently on each physical qubit in the code block. This structure is beneficial because transversal gates naturally limit the propagation of errors across qubits, which is essential for maintaining fault tolerance. For example, in the Steane and Shor codes, logical bit-flip (X) and phase-flip (Z) gates can be implemented transversally by applying the corresponding gate to each physical qubit individually [39]. However, the Eastin-Knill theorem shows that no quantum error-correcting code can implement a universal set of gates using only transversal operations. A way that this manifests for the Clifford+T gate set, for example, is that if the other gates are implemented transversally, the T gate cannot be implemented that way without compromising fault tolerance [40]. Figure 3 shows the syndrome circuit for the Steane code.

To overcome this limitation, fault-tolerant quantum computation employs magic state distillation, a protocol that produces high-fidelity non-Clifford states, typically the T-state, from noisy initial resources. These distilled states can then be injected into a quantum circuit to simulate the action of non-transversal logical gates effectively. The process involves preparing many copies of a noisy magic state and using stabilizer circuits to project

them into a smaller number of purified high-fidelity states [41]. These magic states, once verified and accepted, are used in conjunction with gate teleportation protocols to implement the desired logical operations fault-tolerantly. However, magic-state distillation is resource-intensive and requires large overheads in terms of ancillary qubits and additional circuits, which significantly influences the functional size of the software that controls these operations. As a result, the inclusion of distillation and injection procedures [22] must be accounted for not only in the quantum hardware model but also in the design and measurement of quantum software.

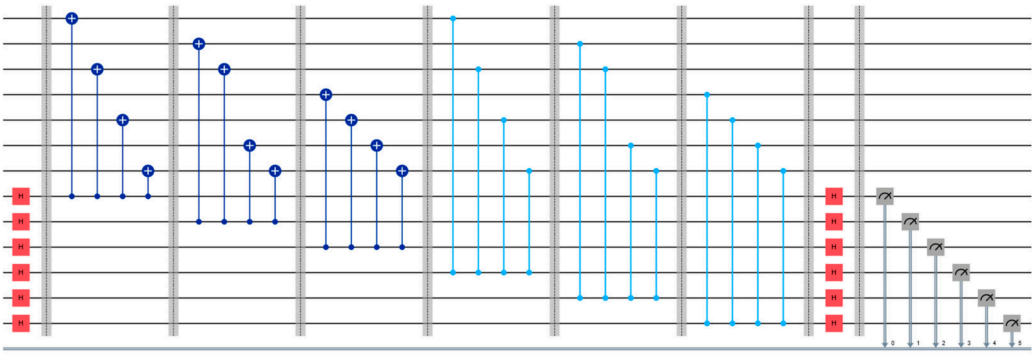

Figure 3. Syndrome measurement circuit for the Steane code.

Surface Codes

Surface codes are a family of topological stabilizer codes that encode quantum information using a two-dimensional arrangement of physical qubits. They extend the stabilizer formalism described above by associating stabilizer operators with the geometry of a two-dimensional surface. The resulting structure allows errors to be detected through measurements of stabilizer operators while the logical information remains encoded across the lattice [42].

A surface-code construction generally distinguishes between data qubits, which store the encoded quantum information, and measurement or syndrome qubits, which are used to extract the values of the stabilizer operators. The stabilizers are typically divided into two classes, corresponding to (X)-type and (Z)-type checks. Measuring these operators produces a syndrome that provides information about the occurrence and location of physical errors without directly measuring the encoded quantum state [9]. Syndrome extraction can be repeated throughout a computation, allowing changes in the syndrome over successive rounds to be used to identify errors and distinguish physical errors from measurement errors [43].

An important parameter of a surface code is its code distance, denoted by (d). The distance is the minimum weight of a nontrivial logical operator, or equivalently the minimum number of physical errors that can combine to produce an undetectable logical error. A distance-(d) code can therefore correct up to errors under the corresponding error model. Increasing the code distance generally increases the number of physical qubits and operations required for the encoded computation while providing greater protection against logical errors [9].

The logical information in a surface code is encoded nonlocally across the physical qubits of the lattice. Logical (X) and (Z) operators can be represented by strings of physical Pauli operations connecting appropriate boundaries of the code. Individual physical errors typically produce detectable changes in the syndrome, whereas a sufficiently large chain of errors can form a logical operator and result in a logical error. The topological structure of the code therefore allows local errors to be detected through their effects on the surrounding stabilizers [42].

Logical operations can be implemented in a surface code using several fault-tolerant techniques. Transversal operations apply a corresponding physical operation independently across the qubits of one or more encoded blocks. For operations supported transversally by the code, this provides a direct method for realizing the corresponding logical operation while limiting the propagation of errors within a code block. Lattice surgery instead performs logical operations by temporarily joining and separating surface-code patches. This is achieved through measurements of joint stabilizers across their boundaries and can be used to implement logical operations such as parity measurements and entangling gates [44]. Defect-based approaches encode logical information using topological defects introduced into the lattice. The defects can be manipulated through code deformation, and braiding one defect around another can implement logical operations such as a controlled-NOT [9].

These approaches provide different ways of realizing operations on the same encoded logical information. In all cases, the underlying logical state remains defined by the code space and its associated stabilizers, while the particular fault-tolerant procedure determines how the logical operation is realized through physical operations on the code.

4. COSMIC Functional Size Measurement and Its Applications in Quantum Software

4.1. Overview of COSMIC

The Common Software Measurement International Consortium (COSMIC) Functional Size Measurement (FSM) method, standardized as ISO/IEC 19761 [10], is a widely accepted and rigorously defined approach to measuring software size based on its functionality as perceived by a software user, whether this user is a human, hardware device, or another software application. Unlike traditional code-centric or complexity-driven sizing techniques, such as lines of code (LOC) and McCabe complexity [45], COSMIC focuses exclusively on quantifying what the software does in terms of functional user requirements (FUR). This user-centered, implementation-agnostic approach makes COSMIC particularly suitable for use across diverse domains and software paradigms.

At the heart of the COSMIC method is the recognition that the software serves its users primarily by processing data. This process typically occurs through interactions at the boundary between the software system and its environment. COSMIC formalizes this view by structuring every measurable functionality into functional processes, each of which comprises elementary data movements that represent these interactions. These movements are the base functional components of functional size and are defined independently of the implementation language, system architecture, or platform. Thus, COSMIC provides a technology-neutral and repeatable method for measuring the size of software functionality even before the implementation stage.

A functional process is the elementary component of a set of functional user requirements. Each process must be triggered by an event, usually initiated by a user or external system, and result in one or more data movements. COSMIC classifies data movements into four distinct and exhaustive types.

1. Entry (E): Movement of a data group from a functional user across the system boundary into the software. This typically corresponds to input operations such as user commands, form submissions, sensor readings, or network requests.
2. Exit (X): Movement of a data group from the software system to a functional user. Examples include the displayed results, status updates, signal transmissions, or file outputs.

3. Read (R): Retrieval of data from persistent storage for use within a functional process. This does not cross the system boundary but represents internal state access (e.g., querying a database or reading a configuration file).
4. Write (W): Transfer of data from a functional process into persistent storage. This captures operations such as logging, state updates, or saving user preferences.

Each of these movements is weighted equally and assigned a value of one COSMIC Function Point (CFP). The total functional size of a software application is determined by summing all CFP across all functional processes derived from the specified user requirements. Importantly, COSMIC does not consider control flow or internal processing complexity. This focus ensures that COSMIC measurements are not dependent on internal implementation decisions.

A key strength of COSMIC is its rigorous definition of the system boundary and its emphasis on the role of functional users. A functional user is any entity (human or system) that exchanges data with software and has a logical stake in the system's operation. Identifying the correct system boundary and users is essential because all measurements are made from this external perspective. This characteristic differentiates COSMIC from structural code measurements and supports its use early in the software lifecycle, including requirements analysis, project estimation, and architectural design.

To clarify the application of the COSMIC Functional Size Measurement in classical software, consider a simple and widely used software function: the user login process. In a typical user login interaction, the user provides authentication credentials (e.g., username and password) that are processed by the system to determine whether access should be granted. From a functional sizing perspective, this interaction comprises a single functional process—authenticate user—which consists of a sequence of elementary data movements, as defined by the COSMIC standard. A Unified Modeling Language (UML) sequence diagram for this use case, shown in Figure 4, illustrates the flow of interactions between the system and its actors (user and database) and the mapping of FUR to COSMIC data movements.

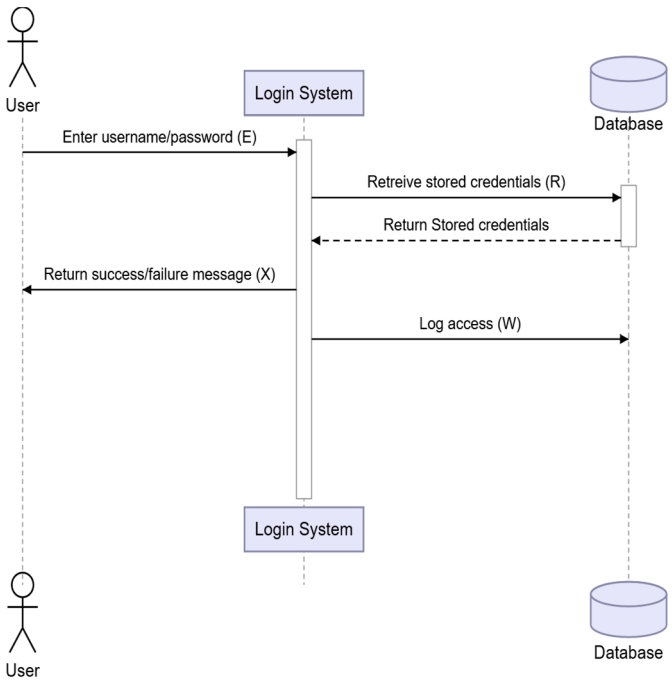

Figure 4. UML Diagram for user login.

The process begins with an Entry (E) movement when the user submits login credentials to the system. The system then performs a Read (R) operation to retrieve the corresponding user record from persistent storage, which is typically a user database. After comparing the submitted credentials and stored values, the system returns a success or failure message to the user via an Exit (X) movement. In this example, where the login activity is tracked for security or auditing purposes, a Write (W) operation is also performed to record the login attempt, timestamp, or session metadata in persistent storage. The resulting COSMIC functional size for this example comprises four data movements.

- Entry (E): Input of username and password.
- Read (R): Retrieval of credentials from the database.
- Exit (X): Return of login outcome.
- Write (W): Logging of the access event.

This yields a total size of four COSMIC Function Points (CFP) for this example of a functional process.

In addition to UML-based models [46], the COSMIC Functional Size Measurement (FSM) method has demonstrated flexibility in measuring a wide range of software artifacts, including source code, user stories, textual requirements, and embedded system specifications [47–50]. This adaptability enables COSMIC to be applied at multiple stages of the software lifecycle and across varying levels of abstraction from early requirement elicitation to implementation [51]. For instance, Soubra and Abran applied the method to Arduino-based embedded platforms to demonstrate how COSMIC can be used to size Internet of Things (IoT) systems. Their demonstration illustrated the feasibility of FSM in resource-constrained environments [50]. COSMIC FSM has also been shown to be effective for effort estimation [48]. Recent studies [52] have demonstrated that COSMIC functional size can be used to estimate web analytics implementation effort. An industrial case study in a FinTech company showed that COSMIC Functional Points measured at the architecture design phase were significantly correlated with actual effort, indicating good estimation accuracy [53].

The broader applicability of COSMIC is further discussed in Dumke and Abran's collection of papers, which details its use across business, telecommunications, and real-time embedded systems [51]. These examples underscore the utility of COSMIC not only in academic analysis but also in industry-grade applications where repeatability, comparability, and reproducibility are essential [54].

With the emergence of quantum software, there has been growing interest in applying COSMIC to quantify the functional size of quantum algorithms and systems. Recent work has shown that COSMIC can still be meaningfully applied by reinterpreting its data movement concepts in a quantum context [17–19]. For example, the initialization of qubit states can be interpreted as Entry movements, whereas quantum measurements (where quantum information collapses to classical outcomes) are seen as Exit movements. Similarly, operations involving memory or classical control registers can be mapped to the Read and Write movements.

Despite these parallels, applying COSMIC to quantum systems introduces new challenges, such as how to model the non-classical interactions inherent in quantum algorithms. Nevertheless, three notable approaches have emerged, each offering a different interpretation of COSMIC principles in the context of quantum computation. These approaches differ in their abstraction levels and the elements identified as COSMIC functional processes.

4.2. Quantum COSMIC Measurement Approaches

4.2.1. Gates' Occurrences Approach

Gates' Occurrences approach [17] represents the most direct and implementation-level application of the COSMIC method to quantum computing. It seeks to translate low-level operations of a quantum circuit into COSMIC functional processes by assigning each gate operation a discrete contribution to the total functional size. In this approach, a quantum circuit is treated as the software system under analysis, and each quantum gate that appears in the circuit is interpreted as an independent functional process, regardless of its type, frequency, or role within the algorithm.

In classical COSMIC, functional processes are defined by their interactions with data: data enters, is read, and then exits or is stored. Translating this logic into the quantum domain, this approach models each gate as performing a unit transformation that "processes" quantum data. Because qubits are altered by the gate, a notional Entry of quantum information is considered to occur at the start of the gate's application and an Exit when the data leaves that gate's scope of operation. This abstraction leads to each gate being mapped to a pair of data movements, one Entry and one Exit, resulting in two COSMIC Function Points (CFP) per gate.

The quantum measurements were also reinterpreted under this scheme using COSMIC data movements. When a qubit is measured, its probabilistic quantum state collapses into a definite classical value. This transformation is modelled as a Write operation, analogous to writing data into classical storage. Subsequently, because the classical result may need to be retrieved or manipulated, a Read operation is also accounted for. Thus, the measurement operations contribute to their own set of functional processes and further increase the total functional size. This interpretation follows the original Gates' Occurrences approach [17] and was adopted in the present study without modification.

The Bell state preparation circuit shown in Figure 5 illustrates this measurement model. This simple yet foundational quantum circuit consists of two quantum gates, a Hadamard (H) gate applied to the first qubit and a controlled-NOT (CX) gate entangling the first and second qubits, followed by measurements on both qubits. According to the Gates' Occurrences approach, each quantum gate contributes one Entry and one Exit, resulting in 4 CFP from the gate operations. Each quantum measurement contributes one Write and one Read, yielding an additional 4 CFP. Therefore, the total functional size of the Bell state circuit is 8 COSMIC Function Points.

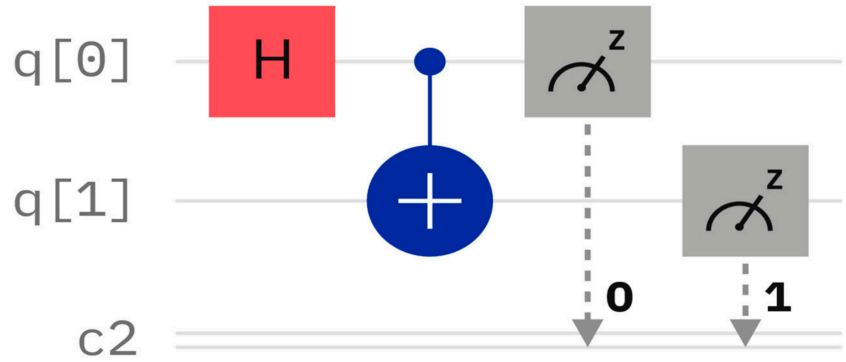

Figure 5. Circuit for preparing the first Bell State.

4.2.2. Gates' Types Approach

The Gates' Types [18] approach introduces a higher level of abstraction that focuses on the types of gates and their logical groupings rather than on their instances. This approach aligns more closely with the intention of COSMIC to measure distinct FUR, rather than their implementation. By shifting from a gate-by-gate count to a process-level perspective,

this approach attempts to group repeated structures and identify unique operations that characterize the behaviour of a quantum algorithm.

4.2.3. Q-COSMIC Approach

While the previous two approaches operate at the circuit level, Q-COSMIC [19] elevates the measurement to the design model layer by integrating COSMIC with quantum-oriented extensions of traditional software modelling languages. Specifically, Q-COSMIC relies on Quantum Unified Modeling Language (Q-UML) [55], a notation designed to express the structural and behavioral aspects of quantum applications at a high level of abstraction.

4.3. Functional Sizing of Error Correction and Fault-Tolerant Computation

Quantum error correction can be regarded as a functionality responsible for detecting and correcting errors within quantum computations that ensures the reliability and correctness of quantum software. From a functional sizing perspective, the operations introduced by error correction can be considered functional requirements related to error correction, which are referred to as error correction overhead. Although they do not directly produce the algorithm's primary computational output (referred to as base functionality), they play a critical role in maintaining the correctness and reliability of the computation [56]. In this study, quantum error correction was examined through the lens of functional size measurement.

The quantum error correction encompasses several well-defined operational phases. It begins with the encoding of quantum information into logical qubits using quantum error-correcting codes, such as the Steane code or the Shor code. These logical qubits are then manipulated through fault-tolerant implementations of the intended logical operations, which may involve transversal gates or more complex techniques, such as magic state injection and distillation. Throughout the execution of the algorithm, syndrome measurements are periodically performed to detect errors without collapsing the logical state. These measurements may be repeated multiple times during the computation to maintain coherence and suppress logical error rates. Although quantum error correction protocols typically define recovery operations based on these measurements, it is common to avoid physical corrections entirely. Instead, the detected errors are tracked classically in a Pauli frame and corrected only logically or at the end of the computation. This method avoids unnecessary quantum operations and reduces the need for potentially noisy correction gates.

Functional size measurement of the QEC is based on the Gates' Occurrences approach [17]. Among the three COSMIC-based approaches proposed for quantum software, this approach provides the most suitable granularity for isolating the overhead introduced by the error correction. Each gate or quantum measurement is treated as an independent functional process with a defined Entry and Exit. This allows for a consistent rule-based accounting mechanism that does not rely on higher-level abstractions or grouping of gate types. By contrast, the Gates' Types approach [18] compresses repeated operations under shared types, potentially undersizing the overhead introduced by repeated syndrome extraction or ancillary gate sequences. Similarly, the Q-COSMIC approach [19], which is aligned with design-level modelling practices, does not offer sufficient granularity to distinguish individual quantum operations or track low-level structural additions to a circuit.

Although alternative COSMIC-based approaches, such as Q-COSMIC [19], may offer higher levels of abstraction and a stronger focus on user-level functionality, the objective of this study was to isolate and quantify the operational contribution of QEC at the circuit level. The Gates' Occurrences approach [17] was therefore selected because it provides a

direct mapping between quantum gate operations and COSMIC data movements, enabling the explicit separation of base functionality and error correction overhead.

The repetitive nature of fault-tolerant computation introduces additional considerations when applying functional sizing. Operations such as syndrome extraction, which may be repeated at each logical time step or between gate layers, cumulatively contribute to the functional size under the Gates' Occurrences approach. Each repetition is treated as an independent functional process, which increases the number of functional processes in proportion to the number of rounds. Although the number of required repetitions is determined by the desired logical error rate and noise characteristics of the physical device, this treatment ensures that each executed quantum operation serves the algorithm's direct functionality and is hence accounted for.

A central component of the entire error correction procedure that was not explicitly addressed in the aforementioned adaptations is the reset operation. Resets are non-unitary quantum operations that reinitialize qubits to a known state, often as a part of ancilla recycling or syndrome extraction. While it is not a unitary gate, the reset alters the state of the system in a meaningful manner and enables subsequent computation. From a functional sizing perspective, reset can be treated analogously to other gate operations. Thus, each reset constitutes a functional process that contributes two CFP, one Entry and one Exit. This treatment ensures the consistent application of the Gates' Occurrences model to both unitary and non-unitary operations within fault-tolerant procedures.

Ancilla qubits play a pivotal role in most fault-tolerant schemes in that they serve as intermediate carriers of error information during syndrome measurement and stabilizer verification. Within the chosen sizing approach, operations preparing, entangling, and measuring ancilla qubits are all interpreted as functional processes. For instance, ancillas are typically initialized in known basis states such as $|0\rangle$ or $|+\rangle$, entangled with data qubits through controlled gates, and subsequently measured to extract error syndromes. Each of these steps contributes to the functional size as an individual gate or the measurement occurrence. As these ancilla-driven processes are repeated throughout the computation to maintain logical coherence, this constitutes a significant portion of the measured functional size, particularly in concatenated or high-distance codes.

For the stated purpose and scope of this study, it is necessary to distinguish between the parts of the circuit that implement the base functionality and those that are part of the error correction overhead. The distinction between base functionality and overhead is not based on the physical realization of the circuit but rather on the measurement principles of COSMIC. Functional size is measured relative to the functional user requirements (FURs), which describe the functionality requested by software users (a human, a hardware device or another software) and define the scope of the measurement. Consequently, operations are classified according to their purpose with respect to the FUR rather than their implementation. Operations that realize the intended logical transformations remain part of the base functionality, regardless of whether they are implemented physically, logically, or fault-tolerantly. Conversely, operations introduced solely to achieve reliable execution on imperfect hardware are classified as error correction overhead. This criterion provides a consistent basis for distinguishing between the two categories across different quantum error correction schemes.

Sizing the functionality introduced by error correction also requires distinguishing between the different abstraction levels in the circuit. A quantum algorithm expressed without error correction represents a higher-level abstraction that specifies idealized operations over logical qubits. In this paper, hardware-level effects, such as noise models, physical gate fidelities, and connectivity constraints, are intentionally abstracted to focus on the functional size at the circuit level. Accordingly, no specific hardware architecture

or error threshold was assumed. In contrast, its fault-tolerant specification and realization decompose these operations into sequences of physical-level subroutines, encoding steps, ancilla-based measurements, and logical gate constructions.

To be more concrete, functional processes that implement the original FUR, such as logical Hadamard or CNOT gates, are considered part of the base functionality even when implemented via transversal or encoded means. Similarly, the encoding procedure that prepares logical basis states reflects the operational domain of the algorithm. By contrast, operations that exist solely to maintain coherence, syndrome measurements, resets, ancillary preparations for stabilizer readouts, and repeated error detection cycles all represent error correction overhead functionality. These operations are not intrinsic to the computational algorithm but are required to support its reliable execution under realistic hardware constraints.

Transversal gates, when used to implement logical operators, are naturally attributed to the original functionality as they serve the intended logical transformations directly. For classification purposes, an operation is assigned to the base functionality if its primary purpose is to realize a functional user requirement or perform the intended logical transformation of the algorithm. An operation is assigned to the error correction overhead if its sole purpose is to detect, diagnose, prevent, or correct errors arising from hardware imperfections. When a fault-tolerant procedure contains both types of functionality, such as magic state distillation, the procedure is decomposed into its constituent functional processes and each process is classified according to its purpose. Operations responsible for preparing or transforming the logical resource required by the algorithm are attributed to the base functionality, whereas operations dedicated to verification, syndrome extraction, error filtering, or rejection of faulty states are attributed to the error correction overhead.

In contrast, operations related exclusively to error tracking and detection are categorized as part of the error correction. More complex schemes, such as magic state distillation, which combines both base and error correction overhead functionality, require additional analysis to separate their contributions, a discussion reserved for the subsequent case study section.

To provide a uniform way for comparing the functional contribution of quantum error correction, we define the QEC Overhead Ratio (QOR) as the proportion of the total measured functional size attributable to error correction:

$$QOR = \frac{CFP_{\text{QEC}}}{CFP_{\text{base}} + CFP_{\text{QEC}}} \times 100\%$$

where CFP_{QEC} denotes the error correction overhead's COSMIC size, and CFP_{base} denotes the quantum algorithm base functionality's COSMIC size. The total measured functional size is therefore $CFP_{\text{total}} = CFP_{\text{base}} + CFP_{\text{QEC}}$. A higher QOR indicates that a greater proportion of the measured functional size is devoted to maintaining computational reliability rather than implementing the algorithm's primary functionality.

For complementary analyses, the relative overhead with respect to base functionality can be expressed as $R_{\text{QEC}} = (CFP_{\text{QEC}}/CFP_{\text{base}}) \times 100\%$. These two indicators should be distinguished when reporting results to ensure consistent interpretation and comparability across studies.

An important limitation of the present measurement scope is the exclusion of classical software functionality required to operate a fault-tolerant quantum computation. In particular, classical post-processing tasks, including syndrome decoding, error tracking, and Pauli frame updates, are not included in the current Gates' Occurrences functional sizing model. Such tasks are essential to interpreting syndrome measurements and maintaining reliable computation. Consequently, the reported CFP values and QEC overhead percentages con-

stitute lower-bound estimates of the functional contribution of QEC within the scope of the present study. They should not be interpreted as complete measurements of the functional size of a fault-tolerant quantum software system. Incorporating the functionality of classical decoding, control, and feedback software could increase the measured functional size of the overall hybrid quantum-classical system. The magnitude of this additional contribution remains to be quantified and constitutes an important direction for future research into hybrid functional size measurement.

5. Case Studies

To evaluate the contribution of quantum error correction (QEC) to the majority of the functional size in quantum circuits, a set of three case studies was conducted. The objective of these is not to compare QEC schemes or performance but rather to demonstrate the application of the COSMIC Functional Size Measurement (FSM) method to quantum error correction procedures.

Three representative quantum circuits were selected for analysis: (1) Bell state preparation, (2) Grover's search algorithm, and (3) a 15-to-1 magic state distillation protocol. Each circuit was implemented in two versions: encoded using the Steane [[7,1,3]] code, and encoded using the Shor [[9,1,3]] code. For the distillation circuit, the encoding is intrinsic to the protocol and uses a 15-qubit code. Across all examples, QEC procedures, including transversal gate construction, syndrome extraction, and fault-tolerant measurement, were included where applicable and fully annotated to distinguish between base functionality and error correction overhead.

The analysis assumes a standard quantum noise model that is consistent with the stabilizer formalism and fault-tolerant computing paradigms [4]. Quantum noise is modelled as stochastic single-qubit Pauli errors and two-qubit correlated Pauli noise during entangling operations. These assumptions are compatible with the stabilizer codes and enable tractable simulations. For simplicity and clarity in measurement, encoding, and error measurement, operations were modelled using ideal gates and no state-of-the-art noise thresholds or hardware limitations were included. This allows the experimental setup to focus purely on the logical and functional aspects of QEC -related operations and how they impact the functional size of the quantum software.

To be more concrete, since the case studies employ a single layer of distance-3 encoding, the Shor and Steane codes yield quadratic suppression through error correction, whereas the 15-to-1 magic state distillation achieves cubic suppression by using the code for error detection. Of course, realistic hardware noise models require much more than a single layer of encoding for large-scale algorithms. For example, Ref. [6] use a level 2 code distance of d = 27 and a level 1 code distance of 17 for factoring a 2048-bit number using Shor's algorithm. Notably, the circuits evaluated in this study were simplified by design. Only a single level of encoding is considered, and QEC codes are not scaled to realistic noise models.

However, this simplification was deliberate. In principle, the distance-3 codes used in this study can be extended through concatenation to achieve stronger error suppression and better align with the requirements of realistic quantum hardware. However, concatenation introduces an exponential increase in circuit size, rapidly scaling our examples from tens of gates to millions or more. Including such large-scale constructions would make the case studies difficult to interpret and detract from the primary objective of this work, which is to apply functional size measurement to quantum error correction. Moreover, even under simplified conditions, error correction introduces a substantial functional overhead. In realistic settings, overhead is significantly magnified. Therefore, these experiments offer a conservative but compelling demonstration of this central research claim.

It is important to note that concatenated distance-3 codes, such as the Shor and Steane codes are not ideal for large-scale implementation due to their demanding connectivity requirements and relatively complex fault-tolerant constructions. In contrast, modern approaches such as surface codes offer more practical advantages, including locality of interactions and planar architectures.

To support this analysis, a prototype measurement tool was developed using Qiskit. The tool implements the Gates' Occurrences approach described in Section 4, parsing quantum circuit descriptions to extract gate operations and classify them into Entry, Exit, Read, and Write movements. The functional size was then computed by counting these operations in accordance with the COSMIC FSM rules. The tool produces structured CFP reports that enable a direct comparison between encoded and unencoded circuits and between base and error correction overhead operations.

To facilitate reproducibility and further exploration, the circuits analyzed in this study, along with all measurement annotations and implementation scripts, were made publicly available online in https://github.com/YoussefAdel617/QuantumErrorCorrectionCosmic (accessed on 10 October 2026). The repository included complete circuit descriptions, relevant documentation, and sample outputs for each case discussed.

The following sections present each case study in detail, including the circuit composition, measurement breakdowns, and analysis of the contributions of error correction overhead versus base functionality.

5.1. Case 1: Bell State

The Bell state circuit was chosen as a foundational test case because of its simplicity and ubiquity in quantum algorithms. It is one of the earliest and most fundamental examples of entanglement and appears in numerous quantum protocols, including but not limited to superdense coding, quantum teleportation, and quantum key distribution [21]. Because it requires only minimal gate operations, it provides a clean and controlled context to observe how fault-tolerant encoding and error correction alter the functional profile of a quantum circuit. Its small scale allows for a detailed inspection of the transformation from an unencoded circuit to its fault-tolerant counterpart and makes it well suited for an initial demonstration of quantum functional sizing.

The unencoded Bell state is generated by applying a Hadamard gate to the first qubit, followed by a CNOT targeting the second qubit. For fault-tolerant analysis, this two-qubit circuit was encoded using two different quantum error correcting codes: Steane 7-qubit code and Shor 9-qubit code. Each logical qubit in the original Bell circuit was replaced by its corresponding encoded representation. The construction of encoded circuits proceeds in multiple steps.

1. Encoding: Logical states are prepared using appropriate initialization and gate sequences defined by each code.
2. Transversal Gates: Logical Hadamard and CNOT operations are implemented transversally and act across the constituent physical qubits of each code block.
3. Syndrome Measurement: Stabilizer generators were measured using ancilla qubits to extract error syndromes after each logical gate.
4. Ancilla Reset and Reuse: After each measurement cycle, the ancilla qubits were reset to allow reuse.

Error correction is assumed to be performed implicitly via syndrome tracking. Physical correction gates are omitted, in line with common practices in which Pauli frame updates are applied in classical software. All circuits were generated and annotated using Qiskit, with gate-level structures explicitly laid out to ensure full visibility of all operations involved.

The functional size of each circuit was measured using Gates' Occurrences approach, as described previously. Every gate is counted as one functional process with a corresponding Entry and Exit, which results in two CFP per gate. The measurement operations were counted as one Write and one Read each. Reset operations are treated analogously to those in unitary gates. The total size was separated into

1. base functionality, which corresponds to logical state preparation and gate execution;
2. error correction overhead, consisting of syndrome measurements, ancilla operations, and resets.

Table 1 summarizes the total number of CFP computed for each category of operations in the encoded Bell state preparation. The results are grouped into base- and error correction overhead functionality and broken down into Entries, Exits, Reads, and Writes. The total size in CFP is computed as the sum of all data movements of the different identified functional processes.

Table 1. Functional size in CFP size for Bell state under Steane and Shor error correction codes.

Code	Functionality	Entry	Exit	Read	Write	CFP
7-qubit Steane	Base	42	42	0	0	84
	Error correction overhead	126	126	18	18	288
	Total	168	168	18	18	372
9-qubit Shor	Base	40	40	0	0	80
	Error correction overhead	144	144	24	24	336
	Total	184	184	24	24	416

5.2. Case 2: Grover's Algorithm

Grover's algorithm was selected as the second benchmark because of its moderate complexity and practical significance. As one of the most widely studied quantum algorithms with proven quantum speedup, Grover's search is an ideal candidate for testing the scalability of functional sizing methods. The version considered in this study targeted a two-qubit search space with the marked or "good" state designated as "11", a standard setup for small-scale demonstrations. In this implementation, the oracle is realized using a controlled-Z (CZ) gate that marks the target state $|11\rangle$ via a phase inversion, while the remaining sequence of Hadamard, Pauli-X, and controlled-NOT operations implements the diffusion operator (inversion about the mean) for the two-qubit search space. Compared to the Bell state, Grover's algorithm involves a larger number of quantum operations, including oracle construction, Hadamard transformations, and inversion about the mean. As a result, it offers a more complex test case but remains analytically tractable. Figure 6 shows the circuit of Grover's algorithm.

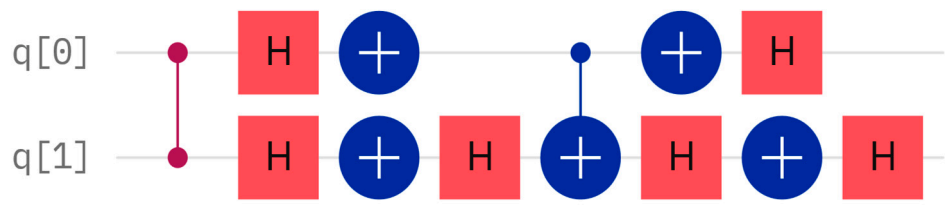

Figure 6. Circuit for Grover algorithm.

The Grover circuits analyzed followed the same methodology as that used in the Bell state case study. Standard two-qubit implementations of Grover's algorithm were manually constructed and encoded using Steane and Shor codes. Fault-tolerant procedures, such as transversal gates and syndrome extraction routines, are incorporated after each

logical operation. Circuit annotations clearly distinguish between base and error correction overhead functionality. Functional sizes in CFP are derived using the Gates' Occurrences approach, with measurement and reset operations treated analogously to the Bell case. Table 2 summarizes the CFP measurements for both codes.

Table 2. Functional size in CFP for Grover's algorithm targeting $|11\rangle$, under Steane and Shor encodings.

Code	Functionality	Entry	Exit	Read	Write	CFP
7-qubit Steane	Base	112	112	0	0	224
	Error correction overhead	588	588	84	84	1344
	Total	700	700	84	84	1568
9-qubit Shor	Base	130	130	0	0	260
	Error correction overhead	672	672	112	112	1568
	Total	802	802	112	112	1828

5.3. Case 3: Magic State Distillation

Magic state distillation is a cornerstone technique in fault-tolerant quantum computing that enables the execution of non-Clifford gates, most notably the T gate, by supplementing an otherwise Clifford-limited instruction set with high-fidelity resource states. Because many quantum error-correcting codes allow only the transversal implementation of Clifford operations, the introduction of non-Clifford gates through direct means is not feasible without compromising the error-correcting properties of the code. Instead, special ancilla states were injected into the computation through gate teleportation. However, such states cannot be directly prepared and are thus distilled from noisy initial versions using specially constructed circuits.

This case study analyzes the 15-to-1 magic-state distillation protocol based on the [[15,1,3]] quantum error correcting code. These states are then encoded into the logical state of the 15-qubit code, on which a transversal T gate is applied. This transversal structure ensures that the logical T operation does not spread errors across qubits in a manner that violates fault-tolerance criteria. The circuit proceeds by measuring the stabilizers of the code, which checks the validity of the encoded state and detects the presence of up to two errors.

In terms of functionality, the purpose of the circuit was to prepare a clean, unencoded T state. Consequently, the functional component includes the preparation of the input states, their encoding into the logical code space, and application of the transversal T gate. The stabilizer measurements and any ancilla-assisted subroutines required to implement them are considered error correction overhead functional components, as they do not contribute directly to the transformation of input data but are instead auxiliary procedures that verify the integrity of the process. Table 3 summarizes the results of the sizing approach when applied to magic state distillation.

Table 3. Functional size in CFP for 15-to-1 magic state distillation.

Functionality	Entry	Exit	Read	Write	CFP
Base	54	54	0	0	108
Error correction overhead	116	116	14	14	260
Total	170	170	14	14	368

5.4. Case 4: Surface Code on a Bell State

Surface codes provide a natural extension of the stabilizer-code constructions considered in this work and have been extensively investigated as a framework for fault-tolerant quantum computation. Their logical operations can be realized through several

different constructions, including transversal operations, lattice surgery, and defect-based approaches. The present case study focuses specifically on a transversal realization of the surface-code Bell-state circuit.

The choice of a transversal construction is primarily a matter of scope rather than a restriction on how surface codes can be implemented. A logical operation supported transversally by the code can be expressed directly as a set of physical operations acting on the corresponding qubits of the encoded blocks. This provides a relatively straightforward way of translating the logical Bell-state circuit into a physical gate-level circuit while retaining a clear correspondence between the logical operations and their underlying physical operations. Other constructions can realize the same logical functionality using different procedures and are therefore not excluded by the approach adopted here.

One consideration with the transversal construction is the connectivity required between the physical qubits of different encoded blocks. A transversal two-qubit operation requires corresponding qubits in the two code blocks to interact. If the underlying hardware imposes a restricted connectivity structure, these qubits may not be directly connected, and additional routing operations, such as SWAPs, may be required. This is a particular consideration for implementations targeting physical architectures with restricted interaction ranges. In contrast, approaches such as lattice surgery are designed to realize logical interactions through operations on neighboring surface-code patches, while defect-based approaches manipulate topological defects within the code.

Accordingly, the circuit used in this case study is defined using an abstract, machine-independent physical gate model. The model specifies the gates used to realize the surface-code construction but does not impose the connectivity or native gate-set constraints of a particular quantum processor. The resulting circuit can therefore be regarded as the gate-level realization of the selected logical construction before hardware-specific compilation. A subsequent compiler could decompose these operations into the native gates of a target device and introduce any routing operations required by its connectivity. A broader scope where connectivity constraints are considered, is left as an important direction for future work.

The case study uses an unrotated surface code with code distance three. Two logical qubits are independently prepared using unitary encoding circuits. The error-correction procedure follows a Steane-style error-correction construction, in which encoded ancilla blocks are prepared and coupled to the encoded data to extract the required error syndromes. The ancilla blocks are subsequently measured and reset. Here, "Steane-style" refers to the method of syndrome extraction using encoded ancilla states and does not imply the use of the seven-qubit Steane code. Table 4 summarizes the results of the sizing approach when applied to the surface code for the bell state.

Table 4. Functional sizes in CFP for Bell state under surface code encoding.

Functionality	Entry	Exit	Read	Write	CFP
Base	66	66	78	0	132
Error correction overhead	354	354	78	78	864
Total	420	420	78	78	996

6. Results and Discussion

The functional size measurement results obtained through the Gates' Occurrences method across four case studies—Bell state preparation using the seven-qubit Steane code, Bell state preparation using the nine-qubit Shor code, Bell state preparation using the surface code, Grover's search algorithm, and 15-to-1 magic state distillation—demonstrate the feasibility of applying a software-centric sizing method to quantum error correction

(QEC). This section presents these results in the context of the core research question: whether QEC can be functionally sized within the context of quantum software and to what extent it contributes to the overall functional size.

Each quantum circuit was allocated to different functionalities: base functionality, meaning those related to FUR, and error correction overhead arising from the QEC schemes. These include error detection routines, such as syndrome measurements, ancillary operations, and reset gates. Table 5 summarizes the functional size in CFP, while Figure 7 illustrates the distribution between base functionality and error correction overhead across all circuits.

Table 5. Comparison of the size in CFP across different circuits and codes. The proportion of error correction overhead functional contributions introduced by quantum error correction is rounded to the nearest integer.

Circuit	Base Functionality CFP	Error Correction Overhead CFP	Total CFP	Error Correction Overhead %
Bell State (7 qubit)	84	288	372	77%
Bell State (9 qubit)	80	336	416	81%
Bell State (surface code)	132	864	996	87%
Grover (7 qubit)	224	1344	1568	86%
Grover (9 qubit)	260	1568	1828	86%
Magic State Distillation	108	260	368	71%

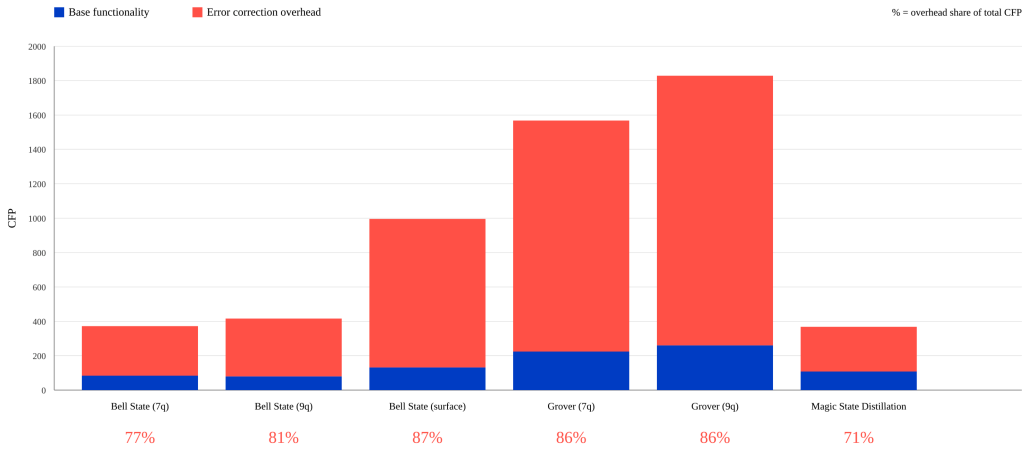

Figure 7. Quantum circuit CFP breakdown: base functionality versus error correction overhead.

Notably, the objective of this study is not to compare the implementation efficiency of alternative fault-tolerant realizations of the same logical functionality. Instead, the reported CFP values quantify the proportion of functionality attributable to error correction within a given implementation. Consequently, the reported overhead percentages should be interpreted as indicators of the relative contribution of QEC-related functionality, rather than measures of implementation optimality or resource efficiency.

Across all examples, the error correction overhead functional contributions consistently dominated the CFP total, ranging from 71% to 87%. This suggests that when error correction is included, the majority of functional elements in the evaluated circuits serve purposes beyond the base functionality. The surface-code case exhibits the largest proportion of error correction overhead among the evaluated Bell-state implementations, with 864 CFP out of a total of 996 CFP attributable to QEC-related functionality.

The case studies confirm that error correction introduces a substantial increase in the functional size of the quantum software. In Grover's algorithm, for example, the inclusion of a single level of fault tolerance using Steane or Shor codes increases the total size in CFP by nearly sevenfold compared to the uncorrected functional baseline. Even in the relatively modest Bell state circuit, the overhead is pronounced, with error correction overhead functional elements accounting for more than 75% of the total CFP in both the encoding schemes. This result holds despite the circuits incorporating only a single level of encoding and assuming simplified noise models.

In the case of magic state distillation, in which the intent is to prepare a high-fidelity non-Clifford resource state, the distinction between the base and error correction overhead functionality is subtle. Logical preparation and transverse application of the T-gate count toward the functional size. However, the decoding and measurement routines necessary to detect and eliminate faulty ancilla states, despite being critical to the procedure, are categorized as error correction overhead functionality because they perform error filtering rather than computation. Even so, they still constitute more than 70% of the total CFP.

These results indicate that Gates' Occurrences method, though limited to low-abstraction, deterministic gate-level models, captures a crucial software-level insight: QEC is currently not a background hardware service but rather a major architectural layer, the size of which scales rapidly with the algorithm size and noise protection requirements. Functional sizing of the error correction processes is demonstrably feasible using this measurement approach. Moreover, QEC routines accounted for a significant portion of the functional size across all evaluated circuits, even under highly simplified assumptions. This supports that QEC is not merely an external hardware concern but an integral component of quantum software size.

The utilized sizing approach intentionally abstracts hardware-specific characteristics. The objective of this study was to quantify the functional contribution of error correction mechanisms at the software level. Although CFP values should not be interpreted as direct measures of hardware cost or performance, they may provide indirect insights into the operational complexity introduced by error correction mechanisms. While the relationship between software size measurement and hardware-related decision-making has received limited attention in the quantum computing literature, evidence from classical software engineering suggests that functional size measurements may provide useful indicators for hardware planning activities [57]. By extension, systematic software sizing approaches in quantum computing may offer valuable insights for hardware feasibility assessments, particularly when considering factors such as logical and physical qubit requirements, gate complexity etc. [58]. In addition, the present study does not establish direct relationships between CFP values and software development effort. Functionalize measurement has historically served as an important basis for effort estimation, project planning, and productivity analysis in classical software engineering. From this perspective, quantifying QEC-related functionality may provide a foundation for future investigations exploring the relationships between size, software development effort, and other project management variables in quantum software engineering. Building on the IEEE and ISO Guide to the Software Engineering Body of Knowledge [59], this study considers Quantum Error Correction (QEC) as a functionality introduced to satisfy the reliability-related non-functional requirements of quantum software.

Increasing the level of fault tolerance generally introduces additional gates, measurements, and ancillary resources, resulting in a corresponding increase in functional size. The circuits used in this study are deliberately restricted to a single level of encoding and simplified error models to maintain clarity. In practical implementations, particularly for larger algorithms, concatenated codes or larger surface-code distances would introduce

substantially more operations and consequently increase the measured functional size. Nevertheless, these findings represent an early exploratory step toward applying software measurements to quantum error correction and a demonstration of sizing approaches adapted from classical software engineering.

7. Limitations and Future Work

Several limitations of this study must be acknowledged. The study focused on small, manually constructed quantum circuits using stabilizer codes and considered only a single level of encoding. The employed error model was idealized by assuming discrete Pauli errors and tensor-decomposable correlations. These simplifications were made deliberately to provide clear examples of functional size measurement and to isolate the contribution of quantum error correction (QEC) without the additional complexity introduced by large-scale, hardware-realistic implementations. Future work should extend the scope of the measurement to more complex requirements, including multi-level encoding, realistic noise models, and hardware connectivity constraints.

More advanced and scalable error-correction schemes should be evaluated under progressively more realistic assumptions. For the surface-code construction considered in this study, this includes larger code distances, alternative logical constructions such as lattice surgery and defect-based approaches, and compilation to hardware architectures with specific connectivity constraints. Other scalable schemes, including concatenated codes and low-density parity-check (LDPC) codes, should likewise be investigated. Integrating such studies with realistic noise models, hardware constraints, and appropriate measurement abstractions will be important for extending functional sizing to larger fault-tolerant quantum systems. Such investigations could examine how functional size evolves with code distance, levels of encoding, syndrome-extraction procedures, logical-gate constructions, and hardware constraints, providing a basis for assessing the applicability of the proposed approach beyond the small-scale exploratory circuits considered in the present study. In this context, automating COSMIC FSM measurement according to the defined measurement rules offers a way to scale the proposed approach to larger fault-tolerant quantum systems. By measuring the system at the COSMIC FSM level, automation can determine its functional size independently of the width and depth of the transpiled physical circuit. This is important because transpilation can alter the number, arrangement, and depth of physical gates required for an implementation, without necessarily changing the functionality represented at the measurement level. Automating the application of the measurement rules would therefore enable consistent and repeatable functional sizing across circuits of different scales and physical implementations, complementing the investigation of increasingly realistic error-correction schemes and hardware architectures.

Another notable limitation is the exclusion of classical post-processing software in functional size measurements. Quantum error correction procedures often rely heavily on classical control systems to interpret measurement outcomes, perform syndrome decoding, make decoding decisions, and trigger corrective operations. Although these classical steps are essential for the execution of fault-tolerant quantum computations, they are not accounted for in the COSMIC measurement presented here. Consequently, the measured functional size and the reported QEC overhead percentages in the case studies discussed in this paper may underrepresent the true size of the overall hybrid quantum–classical software system. Future work should address this gap by exploring hybrid quantum-classical measurement frameworks that can incorporate both quantum and classical functional processes, including syndrome decoding, in a unified model, enabling a more comprehensive assessment of the functional size associated with fault-tolerant quantum computation.

One potential avenue for refinement lies in exploring alternative COSMIC approaches that go beyond basic gate occurrences to include gate types or higher -level functional requirements, as observed in frameworks such as Q-COSMIC. Such approaches may allow for more nuanced measurements of QEC procedures and provide a better alignment with user-level functional requirements. This need arises from an additional limitation of the current study, where the Gates' Occurrences approach treats all gates uniformly from a logical, circuit-level abstraction, without distinguishing or identifying higher-level QEC operations. While suitable for an initial, low-abstraction analysis, this simplification may overlook important differences between gate types and higher-level QEC operations.

Future studies could expand the purpose and scope to include different encoding schemes and corresponding COSMIC measurements on functional size measurements. The scalability of the proposed measurement procedure and instruments (e.g., automated tool for a specific technical environment) for large-scale fault-tolerant quantum circuits containing millions of operations warrants further investigation. In particular, no explicit computational complexity or runtime analysis of the measurement tool has been provided in this work.

Beyond their immediate application in software sizing, functional size measurements play a broader role in software engineering and project management.

Software functional size is a key input for building software development estimation models, effort models, benchmark models, and quality models [56]. Software measurement is a powerful tool for managing software projects that allows the application of engineering principles to software development and providing an objective, quantitative basis for management decisions. For instance, software size gives a measure of the software product itself and can be used to obtain development productivity ratios and build objective estimation models for predicting project effort and duration [57].

In the quantum domain, these benefits are equally vital, particularly as quantum software development becomes more complex and resource-intensive. Accurately estimating the effort required to implement QEC routines, the expected duration of development, or productivity across different hardware platforms could offer critical guidance for both researchers and industry practitioners. Further empirical studies should explore the correlation between functional size and effort, duration, productivity, and quality in the emerging field of quantum software engineering.

8. Conclusions

This study sought to evaluate the functional size contribution of quantum error correction (QEC) within quantum software. Traditionally regarded as a hardware-layer overhead, QEC comprises a complex set of operations that play a central role in the structure and execution of fault-tolerant quantum algorithms. In the examples measured in this exploratory study, the QEC-related functionality constitutes a dominant portion of the total functional size measured by applying the COSMIC Functional Size Measurement (FSM) method.

The results obtained from the case studies sized with COSMIC indicate that QEC-related functionality can represent a substantial proportion of the total functional size of fault-tolerant quantum circuits. However, these findings should be interpreted as preliminary evidence derived from the selected fault-tolerant implementations. Further validation involving larger benchmark circuits, alternative fault-tolerant architectures, and realistic quantum computing environments will be necessary to assess the generalization of these observations.

Employing the Gates' Occurrences approach, the functional size of both the base and error correction overhead functionality was quantified across several representative

circuits: Bell state preparation, Grover's algorithm, and 15-to-1 magic state distillation. In support of these measurements, a prototype tool built in Qiskit was developed to automate the functional sizing of quantum circuits using annotated gate-level descriptions. The results demonstrate that QEC components consistently account for the majority of the total functional size, totaling approximately 70–85% of the measured COSMIC Function Points (CFP). These findings offer quantitative evidence that error correction is currently not peripheral but rather central to quantum software functionality, particularly as systems transition toward fault-tolerant operations.

While limited to small-scale, manually constructed examples, this study illustrates how COSMIC FSM can be used in quantum software workflows and provides early-stage support for the functional sizing of quantum software. Moreover, despite the aforementioned limitations, the findings demonstrate that COSMIC-based sizing is a promising approach to measuring the software size of quantum error correction overhead.

Quantifying the ratio between algorithm-related functionality and QEC-related functionality can provide a management-oriented indicator. This indicator may support future decision-making regarding development effort estimation, productivity analysis, quality and cost management in quantum software engineering.

Ultimately, this work serves as an initial demonstration of how functional size measurements may be brought to bear on error correction, which is one of the most critical components of practical quantum computing. In doing so, it contributes to a broader vision of how software engineering principles help bridge the gap between abstract quantum algorithms and their fault-tolerant realization.

Author Contributions: Y.A.: Conceptualization, Formal Analysis, Methodology, Software, Validation, Visualization, Writing—original draft, Writing—review & editing. H.S.: Conceptualization, Formal Analysis, Methodology, Project administration, Supervision, Validation, Writing—review & editing. T.H.: Conceptualization, Formal Analysis, Methodology, Validation, Writing—review & editing. P.B.: Conceptualization, Formal Analysis, Funding acquisition, Methodology, Project administration, Supervision, Validation, Writing—review & editing. S.B.: Conceptualization, Methodology, Validation, Writing—review & editing. A.A.: Conceptualization, Formal Analysis, Funding acquisition, Methodology, Project administration, Supervision, Validation, Writing—review & editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Mitacs [Grant Numbers IT45410 and IT34807], COSMIC Group, and the ETS Research Chair in Software Product and Project Management.

Data Availability Statement: An online repository of the circuits used is available at: https://github.com/YoussefAdel617/QuantumErrorCorrectionCosmic (accessed on 10 October 2026).

Acknowledgments: During the preparation of this work, the lead author used CHATGPT 5.6 to structure the text and improve its readability. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the content of the published article.

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

Abbreviations

The following abbreviations are used in this manuscript:

COSMIC	Common Software Measurement International Consortium
COSMIC FSM	COSMIC Functional Size Measurement
COSMIC CFP	COSMIC Function Points
FUR	Functional User Requirements
IOT	Internet of Things
LDPC	Low-density parity check
LOC	Lines of code

NISQ	Noisy Intermediate-Scale Quantum
SPAM	State Preparation and Measurement
UML	Unified Modeling Language
QEC	Quantum Error Correction
Q-UML	Quantum Unified Modeling Language

References

1. Shor, P.W. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*; IEEE Computer Society: Washington, DC, USA, 1994; pp. 124–134.
2. Grover, L.K. A Fast Quantum Mechanical Algorithm for Database Search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing: STOC*; Association for Computing Machinery: New York, NY, USA, 1996; pp. 212–219.
3. Daley, A.; Bloch, I.; Kokail, C.; Flannigan, S.; Pearson, N.; Troyer, M.; Zoller, P. Practical quantum advantage in quantum simulation. *Nature* **2022**, *607*, 667–676. [CrossRef] [PubMed]
4. Gottesman, D. An Introduction to Quantum Error Correction and Fault-Tolerant Quantum Computation. In *Proceedings of the Symposium in Applied Mathematics*; AMS: Providence, RI, USA, 2010; Volume 68.
5. Gottesman, D. Theory of Fault-Tolerant Quantum Computation. *Phys. Rev. A Gen. Phys.* **1998**, *57*, 127–137. [CrossRef]
6. Gidney, C.; Ekerå, M. How to Factor 2048 Bit RSA Integers in 8 Hours Using 20 Million Noisy Qubits. *Quantum* **2021**, *5*, 433. [CrossRef]
7. Shor, P.W. Scheme for Reducing Decoherence in Quantum Computer Memory. *Phys. Rev. A Gen. Phys.* **1995**, *52*, 2493–2496. [CrossRef] [PubMed]
8. Steane, A.M. Error Correcting Codes in Quantum Theory. *Phys. Rev. Lett.* **1996**, *77*, 793–797. [CrossRef] [PubMed]
9. Fowler, A.G.; Mariantoni, M.; Martinis, J.M.; Cleland, A.N. Surface codes: Towards practical large-scale quantum computation. *Phys. Rev. A Gen. Phys.* **2012**, *86*, 032324. [CrossRef]
10. *ISO/IEC 19761*; Software Engineering—COSMIC: A Functional Size Measurement Method. International Organization for Standardization: Geneva, Switzerland, 2011.
11. Preskill, J. Quantum Computing in the NISQ Era and Beyond. *Quantum* **2018**, *2*, 79. [CrossRef]
12. Svore, K.; Geller, A.; Troyer, M.; Azariah, J.; Granade, C.; Heim, B.; Kliuchnikov, V.; Mykhailova, M.; Paz, A.; Roetteler, M. Q# enabling scalable quantum computing and development with a high-level dsl. In *Proceedings of the Real World Domain Specific Languages Workshop*; ACM: Rochester, NY, USA, 2018; pp. 1–10.
13. Aleksandrowicz, G.; Alexander, T.; Barkoutsos, P.; Bello, L.; Ben-Haim, Y.; Bucher, D.; Cabrera-Hernández, F.J.; Carballo-Franquis, J.; Chen, A.; Chen, C.-F.; et al. *Qiskit: An Open-Source Framework for Quantum Computing, Version 0.7.2*; Zenodo: Geneva, Switzerland, 2019.
14. Ai, G.Q. Cirq: Quantum Framework by Google 2025. Available online: https://quantumai.google/cirq (accessed on 10 October 2026).
15. Abran, A.; Al-Sarayreh, K.; Lesterhuis, A. (Eds.) Non-Functional Requirements and COSMIC Sizing Practitioner's Guide, COSMIC Version 4.0.2; COSMIC, 2020. Available online: https://cosmic-sizing.org/wp-content/uploads/2020/05/System-NFR-practitioners-v1.0-December-2020.pdf (accessed on 10 October 2026).
16. COSMIC. Sizing Users of the COSMIC Method. Available online: https://cosmic-sizing.org/usage/users-of-the-cosmic-method/ (accessed on 23 July 2025).
17. Khattab, K.; Elsayed, H.; Soubra, H. Functional Size Measurement of Quantum Computers Software. In Proceedings of the 31st International Workshop on Software Measurement and 16th International Conference on Software Process and Product Measurement (IWSM-Mensura 2022), Izmir, Türkiye, 28–30 September 2022; CEUR Workshop Proceedings: CEUR-WS, 2022; Volume 3272. Available online: https://ceur-ws.org/Vol-3272/IWSM-MENSURA22_paper7.pdf (accessed on 10 October 2026).
18. Lesterhuis, A.; Abran, A. Guideline for the Functional Sizing of Quantum Software Requirements (Circuit Strategy); COSMIC, 2025. Available online: https://cosmic-sizing.org/wp-content/uploads/2025/03/Guideline-Quantum-Software-Measurement-Circuit-Strategy.pdf (accessed on 10 October 2026).
19. Valdes-Souto, F.; Perez-Gonzalez, H.G.; Perez-Delgado, C.A. Q-COSMIC: Quantum Software Metrics Based on COSMIC (ISO/IEC19761). *arXiv* **2024**, arXiv:2402.08505.
20. Bova, F.; Goldfarb, A.; Melko, R.G. Commercial applications of quantum computing. *EPJ Quantum Technol.* **2021**, *8*, 2. [CrossRef] [PubMed]
21. Nielsen, M.A.; Chuang, I.L. *Quantum Computation and Quantum Information*; Cambridge University Press: Cambridge, UK, 2010.
22. Litinski, D. A Game of Surface Codes: Large-Scale Quantum Computing with Lattice Surgery. *Quantum* **2019**, *3*, 128. [CrossRef]
23. Suter, D.; Álvarez, G.A. Colloquium: Protecting Quantum Information against Environmental Noise. *Rev. Mod. Phys.* **2016**, *88*, 041001. [CrossRef]

24. Gross, J.A.; Genois, É.; Debroy, D.M.; Zhang, Y.; Mruczkiewicz, W.; Cian, Z.-P.; Jiang, Z. Characterizing coherent errors using matrix-element amplification. *npj Quantum Inf.* **2024**, *10*, 123. [CrossRef]
25. Gutiérrez, M.; Smith, C.; Lulushi, L.; Janardan, S.; Brown, K.R. Errors and Pseudothresholds for Incoherent and Coherent Noise. *Phys. Rev. A* **2016**, *94*, 042338. [CrossRef]
26. Ren, X.; Wan, J.; Liang, Z.; Barbalace, A. Tackling coherent noise in quantum computing via cross-layer compiler optimization. *arXiv* **2024**, arXiv:2410.09664.
27. Figueroa-Romero, P.; Papič, M.; Auer, A.; Vega, I. Estimating the Coherence of Noise in Mid-Scale Quantum Systems. *arXiv* **2024**, arXiv:2409.02110.
28. Resch, S.; Karpuzcu, U.R. Benchmarking Quantum Computers and the Impact of Quantum Noise. *ACM Comput. Surv.* **2021**, *54*, 1–35. [CrossRef]
29. Georgopoulos, K.; Emary, C.; Zuliani, P. Modeling and Simulating the Noisy Behavior of Near-Term Quantum Computers. *Phys. Rev. A* **2021**, *104*, 062432. [CrossRef]
30. Shaib, A.; Naim, M.H.; Fouda, M.E.; Kanj, R.; Kurdahi, F. Efficient noise mitigation technique for quantum computing. *Sci. Rep.* **2023**, *13*, 3912. [CrossRef] [PubMed]
31. Wang, J.; Guo, G.; Shan, Z. SoK: Benchmarking the Performance of a Quantum Computer. *Entropy* **2022**, *24*, 1467. [CrossRef] [PubMed]
32. Blume-Kohout, R. Demonstration of Qubit Operations below a Rigorous Fault Tolerance Threshold with Gate Set Tomography. *Nat. Commun.* **2017**, *8*, 14485. [CrossRef] [PubMed]
33. Sanders, Y.R.; Wallman, J.J.; Sanders, B.C. Bounding Quantum Gate Error Rate Based on Reported Average Fidelity. *New J. Phys.* **2015**, *18*, 012002. [CrossRef]
34. AbuGhanem, M. IBM Quantum Computers: Evolution, Performance, and Future Directions. *J. Supercomput.* **2025**, *81*, 687. [CrossRef]
35. Monz, T.; Nigg, D.; Martinez, E.A.; Brandl, M.F.; Schindler, P.; Rines, R.; Wang, S.X.; Chuang, I.L.; Blatt, R. Realization of a Scalable Shor Algorithm. *Science* **2016**, *351*, 1068–1070. [CrossRef] [PubMed]
36. Chatterjee, A.; Phalak, K.; Ghosh, S. Quantum Error Correction for Dummies. In *Proceedings of the 2023 IEEE International Conference on Quantum Computing and Engineering, QCE*; IEEE: New York, NY, USA, 2023; Volume 1.
37. Wootters, W.K.; Zurek, W.H. A Single Quantum Cannot Be Cloned. *Nature* **1982**, *299*, 802–803. [CrossRef]
38. Gottesman, D. Stabilizer Codes and Quantum Error Correction. Ph.D. Dissertation, California Institute of Technology, Pasadena, CA, USA, 1997.
39. Aliferis, P.; Gottesman, D.; Preskill, J. Quantum accuracy threshold for concatenated distance-3 codes. *Quantum Inf. Comput.* **2006**, *6*, 97–165. [CrossRef]
40. Eastin, B.; Knill, E. Restrictions on Transversal Encoded Quantum Gate Sets. *Phys. Rev. Lett.* **2009**, *102*, 110502. [CrossRef] [PubMed]
41. Bravyi, S.; Kitaev, A. Universal Quantum Computation with Ideal Clifford Gates and Noisy Ancillas. *Phys. Rev. A Gen. Phys.* **2005**, *71*, 022316. [CrossRef]
42. Kitaev, A.Y. Fault-Tolerant Quantum Computation by Anyons. *Ann. Phys.* **2003**, *303*, 2–30. [CrossRef]
43. Fowler, A.G.; Wang, D.S.; Hollenberg, L.C.L. Surface Code Quantum Error Correction Incorporating Accurate Error Propagation. *Quantum Inf. Comput.* **2011**, *11*, 8–19. [CrossRef]
44. Horsman, C.; Fowler, A.G.; Devitt, S.; Van Meter, R. Surface Code Quantum Computing by Lattice Surgery. *New J. Phys.* **2012**, *14*, 123011. [CrossRef]
45. Stavtsev, M.; Bugayenko, Y. Evaluating the Dependency between Cyclomatic Complexity and Response for Class. *arXiv* **2024**, arXiv:2410.06416.
46. Barkallah, S.; Gherbi, A.; Abran, A. Cosmic Functional Size Measurement Using Uml Models. In *Software Engineering, Business Continuity, and Education*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 137–146.
47. Tarhan, A.; Ozkan, B.; Icoz, G.C. A Proposal on Requirements for COSMIC FSM Automation from Source Code. In *26th International Workshop on Software Measurement (IWSM) and the 11th International Conference on Software Process and Product Measurement (Mensura), Berlin, Germany, 5–7 October 2016*; IEEE: New York, NY, USA, 2016; pp. 195–200.
48. Commeyne, C.; Abran, A.; Djouab, R. Effort Estimation with Story Points and Cosmic Function Points-an Industry Case Study. *Softw. Meas. News* **2016**, *21*, 25–36.
49. COSMIC. The COSMIC Functional Size Measurement Method Version 4.0.2: Rice Cooker Case Study, Version 2.0.1; Abran, A., Symons, C., Eds.; COSMIC, August 2018. Available online: https://cosmic-sizing.org/wp-content/uploads/2018/08/Rice-Cooker-2.0.1-GoogleDocs.pdf (accessed on 10 October 2026).

50. Soubra, H.; Abran, A. Functional Size Measurement for the Internet of Things (IoT): An Example Using COSMIC and the Arduino Open-Source Platform. In Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement (IWSM Mensura), Gothenburg, Sweden, 25–27 October 2017; pp. 122–128.
51. Dumke, R.; Abran, A. *COSMIC Function Points: Theory and Advanced Practices*; CRC Press: Boca Raton, FL, USA, 2016.
52. Abdallah, A.; Abran, A.; Qasaimeh, M.; Qasaimeh, M.; Abdallah, B. Measurement of the Functional Size of Web Analytics Implementation: A COSMIC-Based Case Study Using Machine Learning. *Future Internet* **2025**, *17*, 280. [CrossRef]
53. Ersoy, E.; Bagriyanik, S.; Sozer, H. *On the Accuracy of Effort Estimations Based on COSMIC Functional Size Measurement: A Case Study*; ACM: New York, NY, USA, 2024; pp. 528–537.
54. Joint Committee for Guides in Metrology (JCGM). *International Vocabulary of Metrology—Basic and General Concepts and Associated Terms (VIM), Fourth Edition—Committee Draft*; JCGM: Sèvres, France, 2021.
55. Pérez-Castillo, R.; Piattini, M. Design of Classical-Quantum Systems with UML. *Computing* **2022**, *104*, 2375–2403. [CrossRef]
56. *ISO/IEC 25010:2023*; Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—Product Quality Models. International Organization for Standardization: Geneva, Switzerland, 2023.
57. Soubra, H.; Abran, A.; Sehit, M. Functional Size Measurement for Processor Load Estimation in Autosar. In *Software Measurement*; Springer International Publishing: Cham, Switzerland, 2015; pp. 114–129.
58. Hacaloglu, T.; Soubra, H.; Bourque, P.; Abran, A. Quantum Software Size: What Do We Measure, How and Why? A Systematic Literature Review. *IEEE Access* **2026**, *14*, 86893–86914. [CrossRef]
59. Washizaki, H. (Ed.) *Guide to the Software Engineering Body of Knowledge (SWEBOK Guide), Version 4.0*; IEEE Computer Society: Washington, DC, USA, 2024.

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.