

Functional Size Measurement Of Quantum Computers Software

Khaled Khattab¹, Hatem Elsayed² and Hassan Soubra¹

¹German University in Cairo, New Cairo, Egypt

²Technical University of Munich, Munich, Germany

Abstract

Software measurement is an effective technique for project management. It helps engineers to apply engineering concepts to software development, providing a quantitative and objective foundation for process and technology decisions. Many measurement procedures based on international standards have been proposed to obtain the functional size of software. Some of the proposed procedures are automated to minimize measurement variance caused by individual interpretations. However, all of the proposed procedures are focused on 'classical computer' Software, and none addressed Quantum Computer Software. Based on the COSMIC-ISO 19761, and with functional requirements implemented in Qiskit, this paper presents a functional size measurement (FSM) procedure for Quantum Computer Software. The mapping of essential concepts in both Qiskit and COSMIC, as well as the establishment of mapping rules for obtaining the information held in the Qiskit programs that is necessary for measurement, are the foundations of the FSM approach proposed in this paper. Consequently, this procedure provides the foundation for automating the measurement of Quantum Computer Software expressed in Qiskit.

Keywords

Quantum Computers, Quantum Metrics, FSM, COSMIC ISO 19761, Measurement, Measurement Automation, Qiskit

1. Introduction

Quantum computing hardware is rapidly evolving. When only two linked high-quality qubits (quantum bits) were the state of the art only a few years ago, quantum computers of 5, 8, 9, 11, 16, 19, 20, and even 53 programmable and interactable qubits are now available today. In a few years, quantum software should be widely used in research and the industry for cybersecurity, notably cryptography, in the context of the internet of things (IoT) [1].

Software measurement is an effective technique for project management. It helps engineers to apply engineering concepts to software development, providing a quantitative and objective foundation for process and technology decisions. Software size, for example, is a measure of the software product itself that can be used to calculate software development productivity ratios and create objective estimating models for project effort and duration prediction. In practice, the size of software, as measured in function points, is strongly connected to project work

IWSM/MENSURA 22, September 28–30, 2022, Izmir, Turkey


✉ khaled.khattab@student.guc.edu.eg (K. Khattab); hatem.elsayed@tum.de (H. Elsayed);

hassan.soubra@guc.edu.eg (H. Soubra)

🆔 0000-0002-8413-8319 (H. Elsayed); 0000-0003-0056-6015 (H. Soubra)



© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

effort. In addition, Software functional size can be used for several purposes, namely: to obtain system-related technical indicators early in the design phase (e.g. processor load [2] and energy needs [3] [4]), which in turn can impact performance and hence cost.

The COSMIC method is a recognized ISO standard: ISO 19761 Software Engineering - COSMIC – A functional size measuring method. On the COSMIC website, the newest edition of the COSMIC handbook, version 5.0, is accessible (www.cosmic-sizing.org). While this version incorporates certain improvements, the COSMIC method’s core concepts have remained intact since it was initially published in 1999. COSMIC based FSM procedures and automation tools for particular environments have also been proposed e.g. [5] and [6].

This paper is organized as follows: Section 2 presents related work on quantum computing metrics and measurements, Section 3 presents a quantum computing overview. Sections 4 and 5 provide overviews of the COSMIC method and the Qiskit open-source software development kit, then discuss how to apply the COSMIC method to Qiskit, including a Qiskit example. Section 6 introduces the FSM procedure designed for software specified using Qiskit and its rules. Section 7 presents an illustration of the proposed approach applied to an example. Section 8 presents the steps followed in automating the proposed FSM procedure and the design of the automation tool. Conclusions follow in section 9.

2. Related Work on Quantum Computing Metrics and Measurement

Quantum metrics are scarce because Quantum computers are complex and relatively new compared to classical computers. A number of studies were found in the literature, in this section, the three most relevant ones are presented.

Volumetric benchmarks are a class of rectangular quantum circuits in which the dimensions d and w are uncoupled to explore time/space performance trade-offs. Each VB establishes a relationship between circuit shapes – (w,d) pairs – and test suites C . (w, d) . A test suite is a collection of test circuits with a similar structure. A single circuit C , a particular list of circuits $C_1 \dots C_n$ that must all be run, or a vast collection of alternative circuits equipped with a distribution $P_r(c)$ can all be part of the test suite C for a given circuit form $(C)[7]$.

$$\omega = \hat{\omega} + b_g + n_g$$

In [8], the quantum noise metric is presented. Quantum noise may emerge from several different sources. In any quantum system, there is never a single source of noise. It’s difficult to figure out what the sources are and what their respective contributions are. Unwanted interaction with the environment (both separate events and the inevitable decay of quantum states), unwanted interaction between qubits, and imprecise control operations are all possible causes. Each of them introduces errors with distinct features, resulting in a variety of models.

In [9], some software metrics were introduced from classical computers and how to map them to the quantum computing realm. The author tackled different types of software metrics. Firstly, code size metrics are mapped: Lines-of-Code (LOC) and Halstead’s Software Science metrics to quantum computers. Secondly, Design Size metrics: Architectural Design Size metrics and Detailed Design Size metrics are mapped to quantum computers. Finally, some structure metrics such as McCabe’s Complexity Metric and Henry and Kafura’s Information Flow metric are also

mapped to quantum computers.

To our best knowledge, the literature review shows no published works proposing Quantum Computer Software Functional Size Measurement.

3. Quantum Computing Overview

3.1. Quantum bit

A classical bit is a binary information unit used in traditional computing. It can have one of two values: either 0 or 1. On the other hand, a quantum bit (or a qubit) differs from a conventional bit in that any qubit with state $|\psi\rangle$ can be represented by a linear combination (superposition) of two bases in the quantum state space: $|0\rangle = [1 \ 0]^T$ and $|1\rangle = [0 \ 1]^T$, such that:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$$

where $\alpha, \beta \in \mathbb{C}$, and the probability of either state being measured is a sure event.

$$||\psi||^2 = |\alpha|^2 + |\beta|^2 = 1$$

This can also be illustrated as a point in space by the Bloch sphere representation, where states $|0\rangle$ and $|1\rangle$ are at the poles of the sphere[10] as shown in Figure 1 where

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right) |0\rangle + e^{i\phi} \sin\left(\frac{\theta}{2}\right) |1\rangle$$

.

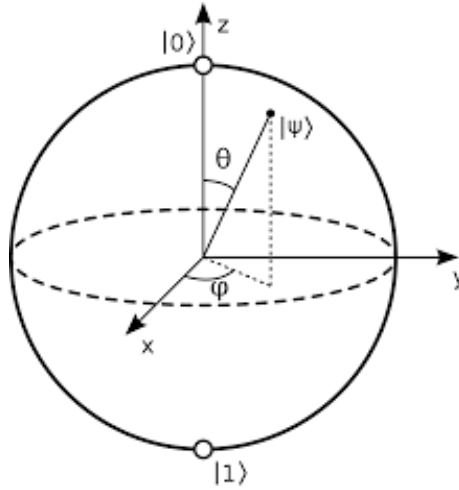


Figure 1: Bloch sphere via [8]

3.2. Quantum circuit

Quantum circuits, also known as quantum logic circuits, are the most widely used general-purpose quantum computing models, representing circuits that operate on qubits as an abstract notion. A quantum circuit is a set of quantum gates that are linked together. And since any multiple qubit operation can be described by a unitary transformation matrix U (reversible and maintain the norm of the operands); this transformation U performed by the circuit determines the structure, types and number of gates, and connectivity scheme of the quantum circuit. Since any unitary matrix U is always invertible, the number of input and output qubits of a quantum circuit is identical, and the quantum operators are reversible [11]. When a qubit is measured, the measurement alters the state of the qubit, collapsing the superposition state $|\psi\rangle$ of $|0\rangle$ and $|1\rangle$ to a state without superposition (either $|0\rangle$ or $|1\rangle$). However, once a qubit has been measured, the measurement cannot be reversed. Therefore, the measurement is non-unitary (not reversible and have probabilistic implementations)[10].

3.3. Quantum gates

A quantum gate can alter the state of a qubit in a similar way to that of a logic gate in a digital circuit changing the state of a classical bit. A quantum gate can have just one input and one output (single quantum state transition) or numerous inputs and multiple outputs (multiple quantum state transition). Because the quantum operators must be reversible, no information can be lost in quantum computing operations and the number of inputs and outputs must be equal. Furthermore, an arbitrary classical circuit can be replaced by a quantum circuit that is reversible by using the Toffoli gate, which can be used to implement NAND gates. Other important quantum gates include: the Hadamard Gate (H) which can be used to prepare a qubit in superposition, the Pauli-(X,Y,Z) gates (the Pauli-X (X) gate is the quantum counterpart of the classical NOT gate), and the Controlled U gate which applies a unitary operation on the second (target) qubit if the first (control) qubit is set to 1 (an example is the CNOT (CX) gate). A Controlled U gate could also be used to acquire entangled qubits. Quantum entanglement is an important element in quantum computing that is used in crucial effects such as fast quantum error-correction, quantum algorithms, and quantum teleportation [10].

3.4. Quantum programming

The process of developing and building executable quantum computer programs to achieve a certain computing outcome is known as quantum programming. A quantum program is made up of code blocks with both classical and quantum components. A quantum program on a quantum computer employs a quantum register of qubits to conduct quantum operations and a classical register of classic bits to record qubit state observations and conditionally apply quantum operators. As a result, a typical quantum program has two types of instructions. One type of instruction is known as classical instructions, which work with the state of classical bits and apply conditional expressions. Quantum instructions, for example, act on the state of qubits and measure their values.

4. COSMIC Overview

COSMIC is an ISO-approved method for measuring FURs. It specifies how to divide the system into layers and how to distinguish between the various data transfers in a system using what are known as Functional Process boundaries. It also specifies the standards for determining the measurement's granularity. A COSMIC Functional Point (CFP) is the measurement unit, and each data transfer is 1 CFP in size.

According to COSMIC, there are four types of data movements:

- Entry (E): A data movement that moves a data group from a functional user across the boundary into the functional process where it is required.
- Exit (X): A data movement that moves a data group from a functional process across the boundary to the functional user that requires it.
- Read (R): A data movement that moves a data group from persistent storage into the functional process that requires it.
- Write (W): A data movement that moves a data group from inside a functional process to persistent storage.

Persistent storage is defined as storage that allows a process to recover data that has been changed by another functional process or another instance of the same process after a functional process has terminated.

COSMIC rules cannot be used to directly measure the size of FURs; two additional steps must be completed first:

- Measurement Strategy Phase, in which the scope and the purpose of the measurement is defined. This is done by applying the COSMIC Software Context Model.
- Mapping phase, where the measurement rules of COSMIC are mapped and defined for the domain being measured.

After the measurement phase, the final software size is obtained by adding the sizes of all the functional processes within the measurement scope.

5. Qiskit Overview

Qiskit (qiskit.org) is an open-source software development kit (SDK) for programming quantum computers at the circuit, pulse, and algorithm levels. It contains tools for writing and modifying quantum programs, as well as for running these programs on local computer simulators or IBM Quantum Experience Prototype quantum devices. It uses the circuit model for universal quantum computation and can be used with any quantum hardware that follows this paradigm.

IBM Research developed Qiskit to facilitate the creation of software for IBM Quantum Experience, a cloud service for quantum computing. External supporters, usually from academic institutions, also contribute.

The Python programming language is used in the main version of Qiskit. Swift and JavaScript versions were briefly explored, however development of these versions was discontinued.

Instead, there is MicroQiskit, a simplified reimplementation of fundamental functionalities that can be easily transferred to other platforms.

There are several Jupyter notebooks with demonstrations of quantum computers in action. Source code for scientific research papers using Qiskit as well as a series of activities to help individuals understand the fundamentals of quantum programming are two examples. An open-source textbook based on Qiskit is offered to complement college courses on quantum algorithms or quantum computation

5.1. Qiskit example

```

1  from qiskit import *
2  circuit = QuantumCircuit(3,3)
3  circuit.x(0)
4  circuit.barrier()
5  circuit.h(1)
6  circuit.cx(1,2)
7  circuit.cx(0,1)
8  circuit.h(0)
9  circuit.barrier()
10 circuit.measure([0,1],[0,1])
11 circuit.draw(output='mpl')
12 circuit.barrier()
13 circuit.cx(1,2)
14 circuit.cz(0,2)
15 circuit.draw(output='mpl')
16 circuit.measure(2,2)
17 simulator = Aer.get_backend('qasm_simulator')
18 result = execute(circuit, backend = simulator, shots = 1000 ).result()
19 counts = result.get_counts()
20 from qiskit.tools.visualization import plot_histogram
21 plot_histogram(counts)
22
23
24

```

Figure 2: Qiskit code

Figure 2 shows a simple qiskit code containing a circuit and some quantum gates. The code implements a quantum teleportation algorithm. First, a quantum register with 3 qubits and a classical register with 3 classical bits were created and then combined to form a quantum circuit. Second, quantum gates such as the H gate, the X gate, and the CX gate were added. Finally, the circuit of the code, as shown in Fig.4, runs on a local simulator called qasm simulator and stores the values of the qubits in the classical bits.

This circuit could also run on a real quantum computer and the results could be presented in different ways:

- Matrix : the result = $\begin{bmatrix} A & B \end{bmatrix}^T$ where A represent the probability of being 0 and B is the representation of the probability of being 1.
- Sphere: the result can be represented as point on the Bloch sphere, as shown in Figure 1, that is a representation of the state of a qubit as the bottom pole represent 1 and the top pole represent 0.
- Histogram : As shown in Figure 3, the result could be represented as the probabilities of the different outcomes.

As shown in Figure 3, the result is represented in a histogram: The X-axis represents the possible answers (e.g.: 100 means that $qubit_2 = 1$, $qubit_1 = 0$, $qubit_0 = 0$) and the Y-axis

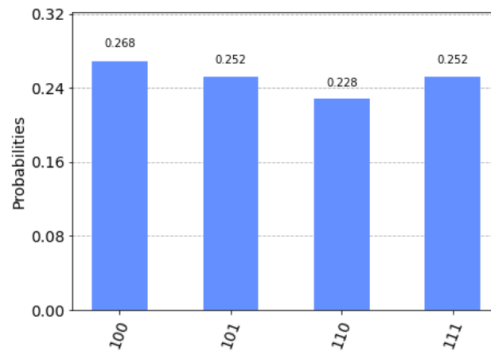


Figure 3: Qiskit histogram representation

represents the probability of being this answer. Moreover, in Qiskit the circuit in Figure 4 is the equivalent to the circuit of the code in Figure 2. The circuit simply teleports the value of qubit 0 after the first barrier to qubit 2 and stores the response in the classical bit 2.

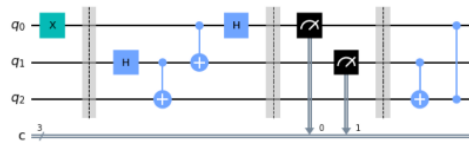


Figure 4: Qiskit circuit representation

6. An FSM Procedure for Quantum Computer using Qiskit

The measurement objective, scope, and elements must be specified and clearly defined in order to adequately measure the functional size of the software implemented with Qiskit. A set of measurement criteria must be included in the FSM method to be applied to the modelled functional requirements to determine their functional size. These criteria must be clear and consistent in order to provide reliable results, simplify the measurement process, and enable automation of the method.

6.1. The measurement strategy phase

Purpose: The purpose of this FSM procedure is to apply the COSMIC approach to the Qiskit software, i.e., to determine the size of any system based on the functional requirements of the Qiskit software.

Scope of Measurement: The scope of this FSM is at the subsystem level of the Qiskit software.

Functional Users: The functional users are all systems that interact (send or receive data) with the specified software.

6.2. The mapping phase

Functional Process: a functional process is a Quantum Gate and receives, manipulates, and transfers data groups when triggered.

Boundaries: Between any external system (functional user) and the program to be monitored there is a boundary. A boundary exists between any two systems (peer components in the same layer).

Data groups: Accurate identification of data group movements in any functional process begins with correct identification of data groups. A single data group is sent over a single line, which is assumed a priori.

Table 1 shows the principles for distinguishing functional users, software boundaries, and functional processes. The principles for identifying data group movement are shown in Tables 2.

Using the principles in Tables 2 and 3, the data group movements of each functional process are identified in this step. After all data movements in a functional process have been identified, each data movement is assigned a standard value of 1 CFP. The final step is to combine the data in order to determine the functional size of each functional process (Rule 9). The functional sizes of the functional processes are then summed to obtain the functional size of the program under consideration (Rule 10).

The Read (R) and Write (W) data movements are bound to the memory Reads and the memory writes respectively (rules 7 and 8).

7. Applying the FSM to Qiskit Example

3 Inport blocks, 3 Outport blocks, 7 Elementary blocks (1 X gate, 2 H gates, 3 CX gates, and 1 CZ gate), 3 Data Store Write blocks, and 3 Data Read blocks are used to represent the simple Qiskit example. All data transfers discovered in this functional process are listed in Table 4. The implemented code in Figure 2 has a total size of 20 CFP.

Table 1
Qiskit/COSMIC Mapping Rules

COSMIC element	Rule number	Rule number
Functional process (FP)	1	Identify 1 functional process for each quantum gate
Boundary	2	Identify 1 boundary between two functional processes interacting with each other
Boundary	3	Identify 1 boundary between any external system interacting with the system to be measured

Table 2

Rules to Identify Data Movements of a Functional Process

COSMIC element	Rule Number	Rule description
Data group movements	4	Identify 1E for each QUBIT connected via a line to a quantum gate
Data group movements	5	Identify 1E for each gate connected via a line to a Functional process
Data group movements	6	Identify 1X for each line to a Functional process
Data group movements	7	Identify 1W for each Quantum Measure identified in this FP
Data group movements	8	Identify 1R for each read from a classical bit

8. Design of the Automation Tool

8.1. Introduction

An automation tool should assist in the application and use of the chosen FSM approach. With Qiskit software, a prototype was created to automate the measurement of the functional size of the requirements.

8.2. Algorithm

The automation tool's algorithm must step by step follow the recommended FSM method. In other words, each functional process (FPdata)'s movements are recognised and given a numerical value of 1 CFP each. Finally, the sizes of all the FP's recognised data motions are combined into a single functional size number.

At first the prototype takes Python file or text file as input.

1. Search for any circuit in the input and identify each as a system
2. Search in each circuit for quantum gates and identify each as a functional process
3. For each functional process identify one Entry data movement
4. For each functional process identify one Exit data movement
5. Search in each circuit for Quantum measure commands and identify each as Data Write
6. Search in each circuit if there is any read from a classical bit and identify each as Data Read
7. Assign one CFP for each Data Entry
8. Assign one CFP for each Data Exit

Table 3

Rules for Obtaining the Functional Size of the Functional Processes and the Whole Software.

COSMIC element	Rule number	Rule description
Functional Process (FP)	9	Aggregate the CFP related to the data movements identified in a specific FP to obtain the functional size of that process
Whole Software	10	Aggregate the CFP related to the data movements of (identified in) the functional processes of (identified in) the whole system to obtain the functional size of that system.

9. Assign one CFP for each Data Read
10. Assign one CFP for each Data Write
11. In order to calculate the CFP for single Functional process Aggregate the CFP related to the data movements identified in a specific FP to obtain the functional size of that process.
12. In order to calculate the CFP for Whole Software Aggregate the CFP related to the data movements of the functional processes of the whole system to obtain the functional size of that system.

8.3. Measuring the simple Qiskit example using the prototype

The measurement result of the Qiskit example achieved using the prototype is shown in Figure 5. It identified seven functional processes in the circuit; in addition to 7 Entries, 7 Exits, 3 Writes, and 3 Reads. The total size of the circuit is 20 CFP.

9. Conclusion

In the near future, quantum software should be widely used in research and the industry. Quantum Computers are of great importance for cybersecurity, notably cryptography, in the context of the internet of things (IoT).

Many measurement procedures based on measurement methods and international standards have been proposed in the literature to obtain the functional size of software for "classical computers".

Table 4
Functional Size of Qiskit Example

No. of the rule applied	Type(s) of block(s) identified	Name(s) of the block(s) identified	Data movement type	CFP value
4	Gate	X	E	1
6	Gate	X	X	1
5	Gate	H	E	1
6	Gate	H	X	1
4	Gate	H	E	1
6	Gate	H	X	1
5	Gate	CX	E	1
6	Gate	CX	X	1
4	Gate	CX	E	1
6	Gate	CX	X	1
5	Gate	CX	E	1
6	Gate	CX	X	1
5	Gate	CZ	E	1
6	Gate	CZ	X	1
7	Measure	qubit 0	W	1
7	Measure	qubit 1	W	1
7	Measure	qubit 2	W	1
8	Read result	qubit 0	R	1
8	Read result	qubit 1	R	1
8	Read result	qubit 2	R	1
				Total: 20 CFP

In this paper, a functional size measurement (FSM) method procedure for Quantum Computers is presented. This procedure is based on the latest version of the COSMIC ISO 19761 measurement method and on the Qiskit tool. The rules of this procedure cover the measurement of quantum computing software projects and can be considered as a set of rules that allow mapping the conceptual elements of Qiskit to the concepts of COSMIC. The procedure proposed attempts to fill in the research gap identified in the FSM procedures scientific literature for Quantum Computers Software.

Finally, measurement with automated tools eliminates measurement variances caused by the interpretation of different measures, which can lead to different measurement results for the same set of requirements. For this reason, a tool that automates the measurement procedure while ensuring the accuracy of measurement results is useful and can benefit organizations in terms of reducing the workload of measurement specialists as well as eliminating measurement delays. This work also provides a basis for developing other types of automated measurement

```

1 DATA E
1 DATA X
1 DATA E
1 DATA X
1 DATA E
1 DATA X
1 DATA E
1 DATA X
1 DATA E
1 DATA X
1 DATA W
1 DATA R
1 DATA W
1 DATA R
1 DATA E
1 DATA X
1 DATA E
1 DATA X
1 DATA W
1 DATA R
total CFP = 20      {E:7 X:7 R:3 W:3}

```

Figure 5: The measurement results by the automation tool

tools for quantum computing software.

References

- [1] Adarsh Kumar, Carlo Ottaviani, Sukhpal S. Gill, and Rajkumar Buyya, "Securing the future internet of things with post-quantum cryptography," SECURITY AND PRIVACY, vol. 5, no. 2, 2021.
- [2] Hassan Soubra, and Alain Abran. "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, pp. 122-128. 2017.
- [3] Soubra, Hassan and Toufik Azib. "Functional Size Measurement for Energy Needs early Estimation in Autonomous Drones." In IWSM-Mensura, pp. 48-53. 2018.
- [4] Hassan Soubra, Alain Abran, and Mehdi Sehit. "Functional size measurement for processor load estimation in AUTOSAR." In Software Measurement, pp. 114- 129. Springer, Cham, 2015.
- [5] Hassan Soubra, and Khaled Chaaban. "Functional size measurement of electronic control

units software designed following the autosar standard: A measurement guideline based on the cosmic iso 19761 standard.” In 2012 Joint Conference of the 22nd International Workshop on Software Measurement and the 2012 Seventh International Conference on Software Process and Product Measurement, pp. 78-84. IEEE, 2012

- [6] Hassan Soubra, Laury Jacot, and Steven Lemaire. ”Manual and Automated Functional Size Measurement of an Aerospace Realtime Embedded System: A Case Study based on SCADE and on COSMIC ISO 19761.” International Journal of Engineering Research and Science Technology 4, no. 2 (2015)
- [7] Robin Blume-Kohout and Kevin C. Young, “A volumetric framework for quantum computer benchmarks,” Quantum, vol. 4, p. 362, 2020.
- [8] Salonik Resch and Ulya R. Karpuzcu, “Benchmarking quantum computers and the impact of Quantum Noise,” ACM Computing Surveys, vol. 54, no. 7, pp. 1–35, 2022.
- [9] Jianjun Zhao, “Some size and structure metrics for Quantum Software,” 2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE), 2021.
- [10] Michael A. Nielsen and Isaac L. Chuang, Quantum Computation and Quantum Information. Cambridge University Press, 2010.
- [11] Yazhen Wang and Hongzhi Liu, “Quantum Computing in a statistical context,” Annual Review of Statistics and Its Application, vol. 9, no. 1, pp. 479–504, 2022.