

Toward Entanglement Dependence Metrics for Quantum Software Complexity

Haibo Yu 

Kyushu Sangyo University, Fukuoka, Japan

Jianjun Zhao 

Kyushu University, Fukuoka, Japan

Abstract

Quantum software is often described using circuit-level metrics, including the number of qubits, the number of gates, circuit depth, the number of two-qubit gates, and the number of measurement operations. These metrics are useful for estimating program size and execution cost, but they do not directly describe whether and to what extent qubits become structurally related through possible entanglement during program execution. This paper argues that entanglement dependence provides a basis for measuring a quantum-specific form of software complexity. The proposed metrics capture possible entanglement-related structural dependence, not physical entanglement observed in a particular execution. Based on this notion, we define a focused set of six metrics that characterize the amount, density, local connectivity, global scope, program span, and modular crossing of entanglement-related dependencies. We also analyze these metrics using Weyuker's properties for evaluating software complexity measures. The paper presents the metrics as a measurement framework and research direction for empirical quantum software engineering. It discusses how future studies can validate the metrics and examine their usefulness for testing, debugging, maintenance, and program comprehension of quantum software.

2012 ACM Subject Classification Software and its engineering → Software testing and debugging; Hardware → Quantum computation

Keywords and phrases Quantum software engineering, quantum program analysis, entanglement dependence, software metrics, software complexity

Digital Object Identifier 10.4230/LIPIcs.ESEM.2026.79

Category Emerging Results, Vision & Reflection Track Paper

Funding *Haibo Yu*: JSPS KAKENHI Grant No. JP24K14908

Jianjun Zhao: JSPS KAKENHI Grant Nos. JP26K02892 and JP26K23820

1 Introduction

Software complexity metrics have long been used to support program comprehension, testing, debugging, maintenance, and empirical analysis [6, 22]. For classical software, many metrics have been proposed to measure different aspects of complexity, such as program size, control-flow structure, data-flow relations, coupling, and cohesion [3, 8, 9, 11]. Although no single metric can fully explain software complexity, these metrics provide useful evidence for comparing programs and studying software engineering tasks. Quantum software also needs suitable complexity metrics. Current metrics for quantum programs are often defined at the circuit level, including the number of qubits, the number of gates, circuit depth, the number of two-qubit gates, and the number of measurement operations [4, 5, 21]. These metrics are useful because they are related to program size, hardware requirements, and execution cost on current quantum devices. However, they mainly describe circuit size and gate structure, and do not directly describe whether and to what extent qubits become structurally related through possible entanglement during program execution.



© Haibo Yu and Jianjun Zhao;
licensed under Creative Commons License CC-BY 4.0

20th International Symposium on Empirical Software Engineering and Measurement (ESEM 2026).

Editors: Robert Feldt, Maria Paasivaara, Daniel Mendez, Stefan Wagner, and Marvin Muñoz Barón;

Article No. 79; pp. 79:1–79:13



Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Entanglement is one of the main sources of such relations [12]. When qubits become entangled, the state of one qubit cannot be understood independently from the state of the others. This has direct implications for software engineering. A developer may not be able to reason about one qubit locally after it has become entangled with other qubits [14, 19]. A fault in an operation that creates or propagates entanglement may affect several qubits and may only be observed later through a measurement operation. These cases suggest that entanglement can introduce a form of structural complexity that is not well captured by simple size-based or circuit-level metrics.

In this paper, we use entanglement dependence as a structural notion. The proposed metrics do not measure physical entanglement observed in a particular execution. Instead, they measure possible entanglement-related dependence induced by program operations and represented in a program dependence structure. This distinction is important because the goal is to support software engineering tasks, such as program comprehension, testing, and debugging, rather than to characterize the exact quantum state produced in one run.

Recent work on quantum program dependence analysis has introduced entanglement dependence as a quantum-specific form of program dependence [20]. Entanglement dependence captures possible state influence among qubits caused by operations that may create or propagate entanglement. This notion provides a possible bridge between quantum state relations and software structure. If entanglement dependencies can be represented in a program dependence graph, they can also be counted, compared, grouped, and related to software engineering tasks. This paper explores this idea from a measurement perspective. We argue that entanglement dependence can serve as a basis for defining metrics for quantum-specific software complexity. The proposed metrics are intended to complement existing circuit-level metrics, not to replace them. While circuit-level metrics describe size and gate structure, entanglement dependence metrics aim to describe whether, where, and to what extent possible entanglement-related dependencies appear in a quantum program, including their amount, density, local connectivity, global scope, program span, and modular crossing.

This paper does not report an empirical validation study. We do not claim that the proposed metrics are already validated. Instead, we make a first step toward a measurement framework for entanglement-related complexity. To make the proposal precise, we define a focused set of six metrics over a dependence-graph representation of quantum programs. We also analyze these metrics using Weyuker’s properties for evaluating software complexity measures [18]. This analysis helps clarify what the metrics can capture and what still needs empirical validation.

This paper makes the following contributions.

- It identifies entanglement dependence as a basis for measuring a quantum-specific form of software complexity.
- It defines a focused set of six metrics for characterizing entanglement-related dependencies in terms of their amount, density, local connectivity, global scope, program span, and modular crossing.
- It analyzes these metrics using Weyuker’s properties for software complexity measures and outlines a research agenda for their empirical validation and use in quantum software engineering activities.

The rest of this paper is organized as follows. Section 2 reviews background and related work. Section 3 motivates the problem and introduces entanglement dependence. Section 4 defines the metrics and illustrates their use. Section 5 presents the theoretical analysis. Section 6 discusses the research agenda and limitations. Section 7 concludes the paper.

2 Background and Related Work

Quantum programs are usually represented as quantum circuits [12]. A quantum circuit operates on qubits, which are the basic units of quantum information. A qubit can be transformed by quantum gates and later read by a measurement operation. Single-qubit gates act on one qubit, while two-qubit gates, such as controlled gates, act on two qubits and may create or propagate entanglement. When qubits are entangled, the state of one qubit cannot be fully described independently of the state of the others [12]. This property is important for quantum computing, but it also makes program comprehension, testing, and debugging different from those for classical programs.

Software complexity metrics have been studied for several decades [6, 22]. Classical metrics capture different aspects of software complexity, such as program size, control-flow structure, information flow, coupling, and cohesion. For example, McCabe’s cyclomatic complexity measures control-flow complexity [11], Halstead’s software science metrics characterize program size and vocabulary [8], Henry and Kafura’s metrics consider information flow between modules [9], and object-oriented metrics capture design properties such as coupling and cohesion [1, 3]. Weyuker proposed a set of properties for evaluating software complexity measures [18]. These properties provide a way to reason about whether a metric can distinguish programs, react to composition, and capture structural differences. This paper uses these properties as a theoretical lens for analyzing the proposed metrics.

Quantum software metrics have also begun to receive attention. Sicilia *et al.* discussed quantum software measurement as part of quantum software engineering [16]. Prior work has proposed size and structure metrics for quantum software at different abstraction levels [21]. Other studies have considered metrics for quantum circuit understandability [4] and quantum software quality [5]. Functional size measurement has also been adapted to quantum software through Q-COSMIC [17]. Together, these studies show that quantum software can be measured from several useful perspectives, including size, structure, understandability, quality, and functional size. However, these perspectives do not directly characterize whether entanglement-related dependencies are localized, dense, widely connected, long-ranged, or crossing modular boundaries. This paper focuses on this missing measurement perspective by studying complexity from the viewpoint of entanglement dependence.

Program dependence analysis provides another structural view of software [7]. In classical software engineering, dependence relations have been used for slicing, testing, debugging, optimization, maintenance, complexity measurement, and program comprehension [2, 10, 13, 15]. For quantum programs, dependence analysis needs to consider quantum-specific features, including state evolution, measurement operations, and entanglement. Recent work has introduced quantum program dependence graphs and identified entanglement dependence as a quantum-specific dependence relation [20]. This paper builds on that idea and studies how entanglement dependence can be used as a basis for measuring an entanglement-related aspect of quantum software complexity.

3 Motivation and Entanglement Dependence

We use a small example to illustrate why circuit-level metrics may be insufficient for measuring quantum software complexity. Figure 1 shows two quantum programs that both use four qubits and a small number of gates. Program *A* prepares two independent Bell pairs, while program *B* uses the same four qubits in a simple entangling chain, creating a connected entanglement-dependence structure. At the circuit level, both programs are small and have similar values for common metrics such as qubit count, controlled gates, and circuit depth. However, their entanglement-dependence structures are different.

Program A: h q[0] cx q[0], q[1] h q[2] cx q[2], q[3]	Program B: h q[0] cx q[0], q[1] cx q[1], q[2] cx q[2], q[3]
--	---

■ **Figure 1** Two small quantum programs with different entanglement-dependence structures.

In Program *A*, the entanglement-dependence structure is local. The first two operations create a possible entanglement relation between $q[0]$ and $q[1]$, while the last two operations create another possible relation between $q[2]$ and $q[3]$. These two pairs are independent. Reasoning about one pair does not require reasoning about all four qubits together. In Program *B*, the entanglement-dependence structure is more connected. The first controlled operation may create an entanglement relation between $q[0]$ and $q[1]$. The following controlled operations may propagate this relation to $q[2]$ and $q[3]$. As a result, the program may require reasoning about a larger connected group of qubits. A fault in an earlier operation may also influence later qubits through the propagated entanglement-related dependence.

This example shows a gap between circuit-level metrics and entanglement-related structure. The number of qubits and the number of controlled gates provide useful information, but they do not directly tell whether entanglement is separated into small local groups or spread across a larger part of the program. A program with localized entanglement dependence may be easier to understand, test, and debug than a program where entanglement dependencies form a large connected structure. Entanglement dependence captures this kind of possible state influence at the program level. It is a form of program dependence that arises from possible entanglement among qubits. In a quantum program, operations such as controlled gates may create or propagate entanglement. After such operations, the state of one qubit may no longer be reasoned about independently from the state of other qubits. The dependence described here is structural and conservative: it indicates that program operations may create or propagate entanglement-related influence, not that physical entanglement is necessarily observed in every execution.

Entanglement dependence is different from classical data dependence. Classical data dependence usually describes how the value of one variable is computed from, or affected by, the value of another variable. Entanglement dependence does not mean that one qubit stores or copies the value of another qubit. Instead, it means that the quantum states of the involved qubits may become structurally related through quantum operations. Such a relation can influence later computation and measurement results, even when there is no classical value flow between program variables.

Entanglement dependence is also different from simply counting two-qubit gates. A two-qubit gate is an operation in a circuit, while an entanglement dependence is a relation induced by the possible state influence of that operation. The same number of two-qubit gates may lead to different dependence structures. Several two-qubit gates may create independent local relations, or they may connect many qubits into a larger entanglement-dependence structure. Therefore, measuring entanglement dependence can provide information that is not directly available from gate counts alone.

For measurement purposes, entanglement dependence is useful because it connects quantum state relations with program structure. Once such dependencies are represented as edges in a dependence graph, they can be measured in terms of their amount, density, connectivity, scope, and span, and then studied in relation to software engineering tasks.

A program where these dependencies form a large connected component may be harder to understand or test than a program where they are separated into small local components. Similarly, dependencies that persist across many statements or circuit layers may make debugging and fault localization more difficult.

This view treats entanglement dependence as a structural element of quantum software. It does not replace existing circuit-level metrics. Instead, it provides an additional layer of measurement. Circuit-level metrics describe properties such as circuit size, depth, and gate structure, while entanglement dependence metrics describe how qubits and program elements are connected through possible entanglement-related dependencies.

4 Entanglement Dependence Metrics

This section defines six metrics based on entanglement dependence. The goal is not to provide a complete metric suite, but to define a focused set of metrics that are precise enough for theoretical analysis and future empirical validation. For ease of presentation, we label the proposed metrics as $M_1, \dots, M_6$.

Let P be a quantum program and let $Q(P)$ be the set of qubits used in P . We assume that a dependence analysis produces a dependence graph $G_P = (V, E)$, where V is a set of analysis nodes and E is a set of dependence edges. Depending on the level of analysis, a node may represent a quantum operation, a program statement, a subroutine element, or a qubit-level abstraction. Let $E_{ent} \subseteq E$ denote the set of entanglement-dependence edges. Each edge $(u, v) \in E_{ent}$ indicates that node v may be affected by a possible entanglement-related state influence created or propagated through node u .

Assumptions for constructing the dependence graph

The proposed metrics depend on the entanglement-dependence graph used as input. This paper does not prescribe one fixed construction algorithm, because such a graph may be built at different abstraction levels and for different quantum programming languages. In general, the graph can be obtained from a quantum program dependence analysis that tracks operations capable of creating or propagating possible entanglement-related state influence. For a simple circuit without measurement operations, classical control, or subroutine boundaries, the resulting entanglement-dependence edges may be close to the topology induced by two-qubit gates. However, entanglement dependence is not intended to be only a count of such gates. In more general quantum programs, the dependence graph may also reflect propagation through later operations, interaction with measurement operations and classical control, and dependencies that cross modular boundaries. A conservative analysis may include possible dependencies that do not occur in every execution, while a more precise analysis may compute fewer edges but may be more expensive or language-specific. Therefore, metric values should always be interpreted together with the dependence model and analysis assumptions used to construct G_P and E_{ent} .

The metrics defined below are independent of a particular implementation of the analysis. They assume that G_P and E_{ent} have already been constructed, and then measure different structural aspects of the resulting entanglement-dependence information.

We use several auxiliary functions when the corresponding information is available. Let $pos(v)$ denote the position of node v in the program or circuit, such as its statement index or circuit layer. If the program is organized into subroutines, let $sub(v)$ denote the subroutine that contains node v .

Metric M_1 (entanglement dependence count)

Metric M_1 counts the number of entanglement-dependence edges:

$$M_1(P) = |E_{ent}|.$$

It measures the amount of entanglement-dependence structure in a program. A larger value means that more program elements may be connected through possible entanglement-related dependencies.

Metric M_2 (entanglement dependence density)

Metric M_2 normalizes the number of entanglement-dependence edges by the total number of dependence edges:

$$M_2(P) = \frac{|E_{ent}|}{|E|}.$$

When E is empty, we define $M_2(P) = 0$. This metric describes how much of the dependence structure is related to entanglement. It complements M_1 by reducing the direct effect of program size.

Metric M_3 (maximum entanglement degree)

For a node $v \in V$, let its entanglement degree be

$$edeg(v) = |\{u \in V \mid (u, v) \in E_{ent} \vee (v, u) \in E_{ent}\}|.$$

Metric M_3 is the maximum entanglement degree:

$$M_3(P) = \max_{v \in V} edeg(v).$$

It identifies the node that is most highly connected through entanglement dependence. Such a node may be a potential hotspot for testing, debugging, or program comprehension.

Metric M_4 (maximum entanglement cluster size)

Let $G_{ent} = (V, E_{ent})$ be the subgraph induced by entanglement-dependence edges. We ignore edge directions when computing connected components. Let $C_1, C_2, \dots, C_k$ be the connected components of G_{ent} that contain at least one entanglement-dependence edge. Metric M_4 is the maximum entanglement cluster size:

$$M_4(P) = \max_{1 \leq i \leq k} |C_i|.$$

If E_{ent} is empty, we define $M_4(P) = 0$. This metric describes the largest connected region of entanglement dependence in the program. It helps distinguish localized entanglement-dependence structure from a structure that connects many nodes.

Metric M_5 (maximum entanglement span)

For an entanglement-dependence edge $(u, v) \in E_{ent}$, its span is defined as

$$span(u, v) = |pos(v) - pos(u)|.$$

Metric M_5 is the maximum entanglement span:

$$M_5(P) = \max_{(u,v) \in E_{ent}} \text{span}(u, v).$$

If E_{ent} is empty, we define $M_5(P) = 0$. This metric describes the longest program range of entanglement-related state influence. Long spans may make reasoning and fault localization more difficult because the effect of an earlier operation may appear much later.

Metric M_6 (cross-subroutine entanglement dependence)

For programs organized into subroutines, Metric M_6 counts entanglement-dependence edges that connect nodes in different subroutines:

$$M_6(P) = |\{(u, v) \in E_{ent} \mid \text{sub}(u) \neq \text{sub}(v)\}|.$$

This metric measures the amount of entanglement dependence that crosses modular boundaries. It may be useful for studying modular reasoning, integration testing, and maintenance of quantum software.

Table 1 summarizes the proposed metrics.

■ **Table 1** Proposed entanglement dependence metrics.

Metric	Intended meaning
M_1	Amount of entanglement-dependence structure
M_2	Relative density of entanglement dependence in the dependence graph
M_3	Highest local connectivity through entanglement dependence
M_4	Largest connected region of entanglement dependence
M_5	Longest program range of entanglement-related state influence
M_6	Entanglement dependence across subroutine boundaries

These metrics should be interpreted together rather than in isolation. Metric M_1 measures the amount of entanglement-dependence structure, but a high value may simply reflect a large program. Metric M_2 provides a normalized view of how much of the dependence graph is related to entanglement. Metric M_3 identifies locally central nodes, while M_4 captures the largest connected entanglement-dependence structure. Metric M_5 describes how far entanglement-related influence may extend in the program, and M_6 captures whether such influence crosses subroutine boundaries. Together, these metrics provide a focused view of entanglement-related complexity.

Applying the Metrics to the Example

We use the example in Section 3 to illustrate how the proposed metrics can expose differences that are not fully captured by circuit-level metrics. This analysis is not an empirical evaluation. It only shows how the metrics can be applied once entanglement-dependence relations are identified.

For simplicity, we analyze the example at the qubit-relation level. This is a simplified view of the dependence graph that keeps only the qubit-level entanglement-dependence relations needed to illustrate M_1 , M_3 , and M_4 . We treat each possible entanglement dependence relation between two qubits as an undirected edge. For readability, we write q_i for $q[i]$ in the graph notation. Program A creates two independent Bell pairs, while Program B uses the same four qubits in a simple entangling chain, creating a connected entanglement-dependence structure. The simplified qubit-level entanglement-dependence graphs are:

$$G_{ent}^A = (\{q_0, q_1, q_2, q_3\}, \{(q_0, q_1), (q_2, q_3)\}), G_{ent}^B = (\{q_0, q_1, q_2, q_3\}, \{(q_0, q_1), (q_1, q_2), (q_2, q_3)\}).$$

In each graph, the first component is the set of qubit nodes, and the second component is the set of undirected entanglement-dependence edges. Thus, G_{ent}^A contains two separate connected components, one consisting of q_0 and q_1 and the other consisting of q_2 and q_3 , while G_{ent}^B contains one connected component over all four qubits, formed by the chain (q_0, q_1) , (q_1, q_2) , and (q_2, q_3) .

Table 2 shows the values of three metrics that can be directly illustrated by this example. Program A has two entanglement-dependence edges, two connected components, and each qubit is connected to only one other qubit. Program B has three entanglement-dependence edges, one connected component containing all four qubits, and two middle qubits with degree two.

■ **Table 2** Illustrative metric values for the motivating example.

Metric	Program A	Program B
M_1	2	3
M_3	1	2
M_4	2	4

The example shows that the two programs have different entanglement-dependence structures. Program A has localized entanglement dependence, while Program B has a connected entanglement-dependence structure that spans all four qubits. Thus, even when programs are small and have similar circuit-level metric values, their entanglement-dependence structures may differ.

This example does not compute all metrics. Metric M_2 requires the full dependence graph, not only the entanglement-dependence edges. Metric M_5 requires positional information such as statement indices or circuit layers. Metric M_6 is relevant only when the program is organized into subroutines. These metrics can be applied in the same way once the corresponding information is available.

The main point is that entanglement-dependence metrics make structural differences explicit. A simple count, such as M_1 , shows that Program B has more entanglement-dependence edges. Metrics M_3 and M_4 further show that Program B has higher local connectivity and a larger connected entanglement-dependence structure. Such differences may matter for program comprehension, test design, debugging, and fault localization.

5 Theoretical Analysis of the Metrics

We analyze the proposed metrics using Weyuker’s properties for evaluating software complexity measures [18]. These properties were originally proposed for software complexity metrics in general. In this paper, we apply them to metrics over quantum programs. The goal is not to prove empirical usefulness, but to clarify the formal behavior of the metrics before empirical validation.

Let $\mathcal{P}$ be the set of quantum programs under consideration. A complexity metric is a function

$$\mu : \mathcal{P} \rightarrow \mathbb{R}_{\geq 0},$$

where $\mathbb{R}_{\geq 0}$ denotes the set of non-negative real numbers. In this section, μ denotes an arbitrary metric, while $M_1, \dots, M_6$ denote the concrete metrics defined in Section 4. We write $P;Q$ for sequential composition, $P \equiv Q$ for semantic equivalence, and $\rho(P)$ for a program obtained from P only by consistent renaming of qubits, variables, or statement labels.

Tables 3 and 4 summarize the Weyuker properties used in this paper and the satisfaction results, respectively. Here, “Y” means that the property holds under the metric definitions used in this paper, and “N” means that it does not hold in general. For properties stated using “there exist”, such as W_1 , W_3 , W_4 , W_6 , W_7 , and W_9 , a “Y” means that suitable programs or contexts can be constructed under the metric definitions. It does not mean that every program pair or every composition has that behavior.

■ **Table 3** Weyuker properties used in this paper [18].

Property	Name	Form used in this paper
W_1	Non-coarseness	There exist programs P, Q such that $\mu(P) \neq \mu(Q)$.
W_2	Finiteness	For every value c , only finitely many programs P satisfy $\mu(P) = c$.
W_3	Non-uniqueness	There exist different programs P, Q such that $\mu(P) = \mu(Q)$.
W_4	Design sensitivity	There exist semantically equivalent programs P, Q such that $P \equiv Q$ and $\mu(P) \neq \mu(Q)$.
W_5	Monotonicity under composition	For all P, Q , $\mu(P) \leq \mu(P; Q)$ and $\mu(Q) \leq \mu(P; Q)$.
W_6	Interaction sensitivity	There exist programs P, Q, R_a, R_b such that $\mu(P) = \mu(Q)$, $\mu(P; R_a) \neq \mu(Q; R_a)$, and $\mu(R_b; P) \neq \mu(R_b; Q)$.
W_7	Order sensitivity	There exist programs P, Q such that Q is obtained by permuting the order of parts of P and $\mu(P) \neq \mu(Q)$.
W_8	Renaming invariance	For every consistent renaming ρ , $\mu(P) = \mu(\rho(P))$.
W_9	Non-additive composition	There exist P, Q such that $\mu(P; Q) > \mu(P) + \mu(Q)$.

■ **Table 4** Satisfaction of Weyuker properties.

Metric	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9
M_1	Y	N	Y	Y	Y	Y	Y	Y	Y
M_2	Y	N	Y	Y	N	Y	Y	Y	Y
M_3	Y	N	Y	Y	Y	Y	Y	Y	Y
M_4	Y	N	Y	Y	Y	Y	Y	Y	Y
M_5	Y	N	Y	Y	Y	Y	Y	Y	Y
M_6	Y	N	Y	Y	Y	Y	Y	Y	Y

All six metrics satisfy W_1 , W_3 , W_4 , W_6 , W_7 , W_8 , and W_9 . They satisfy W_1 because programs with and without entanglement dependence can have different values. They satisfy W_3 because different programs may have the same metric value. For example, two different circuits may both contain two independent entanglement-dependence edges, and therefore have the same value for M_1 . They satisfy W_4 because semantically equivalent programs may use different decompositions and induce different entanglement-dependence structures. They satisfy W_6 because two programs with the same metric value may interact differently with suitable contexts on either side of a composition. For example, a context may connect

to qubits that are structurally independent in one program but already connected through entanglement dependence in another program. They satisfy W_7 when reordering parts of a program changes the dependence structure or the span of dependencies, and they satisfy W_8 because consistent renaming does not change the dependence graph up to isomorphism.

None of the metrics satisfies W_2 over an unrestricted program set. This is expected because the proposed metrics measure only entanglement-related structure. Many programs can differ in parts unrelated to entanglement dependence, such as additional single-qubit gates or classical computations, while keeping the same metric value.

For W_5 , only M_2 does not satisfy the property in general. Metric M_2 is a density metric. When a program is composed with another program containing many non-entanglement dependencies, the ratio of entanglement dependencies may decrease. In contrast, M_1 , M_3 , M_4 , M_5 , and M_6 are absolute or maximum-based metrics, and their values do not decrease under composition that preserves existing dependence edges.

All six metrics satisfy W_9 because composition can introduce new entanglement dependencies across the composition boundary. Because W_9 is stated using “there exist”, it does not require every composition to have a value greater than the sum of the values of its parts. For example, composing two independent Bell-pair programs without any cross-boundary entangling operation may not increase M_4 beyond the values of the parts. This does not contradict W_9 . Thus, the metric value of the composed program can be greater than the sum of the values of the two parts. There exist compositions in which later operations connect previously separated entanglement-dependence components, introduce additional cross-boundary edges, or create longer dependence spans. In such cases, count-based metrics such as M_1 and M_6 may increase beyond the sum of the separate values, density-based metric M_2 may become larger after composition, and maximum-based metrics such as M_3 , M_4 , and M_5 may also exceed the sum of the corresponding values of the parts. This reflects the non-local and context-dependent nature of entanglement dependence.

This analysis does not replace empirical validation. It only clarifies the formal behavior of the proposed metrics.

6 Research Agenda and Limitations

The proposed metrics need empirical validation before they can be used as evidence in quantum software engineering. The theoretical analysis in Section 5 clarifies their formal behavior, but it does not show whether they are useful for explaining or predicting the difficulty of software engineering tasks such as testing, debugging, maintenance, and program comprehension. We outline several directions for future validation.

- **Comparison with Circuit-Level Metrics.** Future studies should compare M_1 – M_6 with circuit-level metrics, including the number of qubits, the number of gates, circuit depth, the number of two-qubit gates, and the number of measurement operations. If they capture different properties, they can complement existing metrics and provide a more detailed view of quantum software complexity.
- **Testing Difficulty.** Programs with high entanglement-dependence density, high local connectivity, large connected entanglement-dependence components, or long entanglement-dependence spans may require more careful test design, test input preparation, or oracle construction. Future studies can examine whether M_2 , M_3 , M_4 , and M_5 are associated with fault detection, mutation score, coverage achievement, or test generation cost.
- **Debugging and Fault Localization.** When a failure is observed in a measurement result, the cause may come from an earlier operation that created or propagated entanglement. Empirical studies can examine whether M_3 , M_4 , and M_5 are related to fault-localization accuracy or debugging effort.

- **Program Comprehension.** Entanglement-related complexity may also affect program comprehension. Controlled experiments or user studies can examine whether the proposed metrics are related to the time needed to understand a program, the accuracy of answers to comprehension questions, or the difficulty perceived by developers.
- **Design and Maintenance.** The metrics can be used to compare different implementations of the same quantum algorithm, different circuit decompositions, or different modular designs. Metric M_6 is especially relevant for studying whether entanglement dependence crosses subroutine boundaries.
- **Tool Support and Datasets.** Future work also needs tools for extracting entanglement dependence and computing the proposed metrics. Benchmark datasets should include small educational programs, standard quantum algorithms, and real quantum software projects.
- **Metric Correlation and Redundancy.** Future studies should also examine the relationships among the proposed metrics. Some metrics may be correlated on certain classes of programs because they are computed from the same entanglement-dependence subgraph. For example, programs with many entanglement-dependence edges may also tend to have higher local connectivity or larger connected components. However, this need not always be the case: a program with many small independent components may have a high value of M_1 but a small value of M_4 , while a program with fewer edges may still contain one large connected component. Empirical studies are needed to determine when the metrics provide complementary information and when some of them are redundant.

Overall, the next step is to move from metric definition to empirical evidence. The main questions are whether the metrics can be computed reliably, whether they provide information different from existing metrics, whether they provide complementary information to each other, and whether they help explain software engineering effort or difficulty. These directions should be interpreted alongside the limitations of the current study, as outlined below.

- **Lack of empirical validation.** The proposed metrics have not yet been empirically validated. Future studies are needed to examine whether they are related to testing effort, debugging difficulty, fault-localization accuracy, program comprehension, or maintenance cost.
- **Dependence on the underlying analysis.** The metrics depend on how entanglement dependence is defined and extracted. Different dependence models may produce different entanglement-dependence edges. This issue is central because all metric values are computed from the dependence graph and its entanglement-dependence edges. For example, a conservative analysis may include possible dependencies that do not occur in every execution, while a more precise analysis may be harder to compute. Therefore, any use of the proposed metrics should clearly state the underlying dependence model and analysis assumptions.
- **Structural dependence rather than physical entanglement.** The metrics describe structural dependence rather than physical entanglement observed in a particular execution. An entanglement-dependence edge means that one program element may be affected by a possible entanglement-related state influence created or propagated through another program element. It does not by itself prove that a specific run produces entanglement between the corresponding qubits. This distinction is important for programs with conditional behavior, measurement operations, input states, or classical control.

- **Abstract definitions.** The current definitions are still abstract. They are defined over a dependence-graph representation of quantum programs, but this paper does not provide a complete algorithm for constructing such graphs for all quantum programming languages. In practice, the construction may depend on the language, the intermediate representation, the circuit model, and the treatment of measurement and classical computation.
- **Limited scope.** The proposed metrics focus only on entanglement-related complexity. Quantum software complexity also includes other aspects, such as control structure, measurement behavior, classical–quantum interaction, noise sensitivity, resource usage, and maintainability. We also define only an initial set of six metrics in this paper. Other useful variants, such as average degree, average span, or normalized cross-subroutine dependence, are left for future work. The proposed metrics should therefore be used as a complement to existing circuit-level and software engineering metrics.

7 Conclusion

This paper presented an initial measurement framework for quantum software complexity based on entanglement dependence. Existing circuit-level metrics, including the number of qubits, the number of gates, circuit depth, the number of two-qubit gates, and the number of measurement operations, are useful for describing program size and execution cost. However, they do not directly describe whether and to what extent qubits become structurally related through possible entanglement dependence during program execution. To address this gap, we defined a focused set of six entanglement dependence metrics that characterize different aspects of entanglement-related complexity, including amount, density, local connectivity, global scope, program span, and modular crossing of entanglement-related dependencies. We also analyzed the metrics using Weyuker’s properties for software complexity measures. The analysis shows that these metrics can capture structural information not directly captured by existing circuit-level metrics. The proposed metrics are not yet empirically validated and should be viewed as a starting point for empirical quantum software engineering. Future work should develop tool support, apply the metrics to real quantum programs, and study their relationship with testing, debugging, maintenance, and program comprehension.

Data Availability Statement

This paper does not report an empirical study and does not produce a new dataset or software artifact. All examples used in the paper are fully described in the text.

References

- 1 Lionel C. Briand, John W. Daly, and Jurgen K Wust. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on software Engineering*, 25(1):91–121, 1999. doi:10.1109/32.748920.
- 2 Jingde Cheng. Complexity metrics for distributed programs. In *Proceedings of 1993 IEEE International Symposium on Software Reliability Engineering*, pages 132–141. IEEE, 1993. doi:10.1109/ISSRE.1993.624282.
- 3 Shyam R. Chidamber and Chris F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994. doi:10.1109/32.295895.
- 4 José A. Cruz-Lemus, Luis A. Marcelo, and Mario Piattini. Towards a set of metrics for quantum circuits understandability. In *Quality of Information and Communications Technology*, pages 239–249. Springer, 2021. doi:10.1007/978-3-030-85347-1_18.

- 5 José A. Cruz-Lemus, Moisés Rodríguez, Raúl Barba-Rojas, and Mario Piattini. Quantum software quality metrics. In *Quantum Software*, pages 125–142. Springer, 2024. doi:10.1007/978-3-031-64136-7_6.
- 6 Norman Fenton and James Bieman. *Software metrics: a rigorous and practical approach*. CRC press, 2014.
- 7 Jeanne Ferrante, Karl J Ottenstein, and Joe D Warren. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 9(3):319–349, 1987. doi:10.1145/24039.24041.
- 8 Maurice H. Halstead. *Elements of Software Science*. Elsevier, 1977.
- 9 Sallie Henry and Dennis Kafura. Software structure metrics based on information flow. *IEEE Transactions on Software Engineering*, SE-7(5):510–518, 1981. doi:10.1109/TSE.1981.231113.
- 10 Susan Horwitz, Thomas Reps, and David Binkley. Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(1):26–60, 1990. doi:10.1145/77606.77608.
- 11 Thomas J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, SE-2(4):308–320, 1976. doi:10.1109/TSE.1976.233837.
- 12 Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.
- 13 Karl J Ottenstein and Linda M Ottenstein. The program dependence graph in a software development environment. *ACM Sigplan Notices*, 19(5):177–184, 1984.
- 14 Simon Perdrix. Quantum entanglement analysis based on abstract interpretation. In *International Static Analysis Symposium*, pages 270–282. Springer, 2008. doi:10.1007/978-3-540-69166-2_18.
- 15 Andy Podgurski and Lori A Clarke. A formal model of program dependences and its implications for software testing, debugging, and maintenance. *IEEE Transactions on Software Engineering*, 16(9):965–979, 1990. doi:10.1109/32.58784.
- 16 Miguel-Angel Sicilia, Marçal Mora-Cantallops, Salvador Sánchez-Alonso, and Elena García-Barriocanal. Quantum software measurement. In *Quantum Software Engineering*, pages 193–208. Springer, 2022. doi:10.1007/978-3-031-05324-5_10.
- 17 Francisco Valdes-Souto, Hector G. Perez-Gonzalez, and Carlos A. Perez-Delgado. Q-COSMIC: Quantum software metrics based on COSMIC (ISO/IEC 19761), 2024. arXiv:2402.08505.
- 18 Elaine J. Weyuker. Evaluating software complexity measures. *IEEE Transactions on Software Engineering*, 14(9):1357–1365, 1988. doi:10.1109/32.6178.
- 19 Shangzhou Xia and Jianjun Zhao. Static entanglement analysis of quantum programs. In *2023 IEEE/ACM 4th International Workshop on Quantum Software Engineering (Q-SE)*, pages 42–49. IEEE, 2023. doi:10.1109/Q-SE59154.2023.00013.
- 20 Haibo Yu and Jianjun Zhao. The quantum program dependence graph and its uses in quantum software development. In *Proceedings of the 2025 IEEE/ACM 6th International Workshop on Quantum Software Engineering (Q-SE)*, pages 48–55. IEEE, 2025. doi:10.1109/Q-SE66736.2025.00014.
- 21 Jianjun Zhao. Some size and structure metrics for quantum software. In *Proceedings of the 2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE)*, pages 22–27. IEEE, 2021. doi:10.1109/Q-SE52541.2021.00012.
- 22 Horst Zuse. *Software complexity: measures and methods*, volume 4. Walter de Gruyter GmbH & Co KG, 1990.