Vol.2020-QS-1 No.1
2020/10/16

量子ソフトウェア工学の現状について

趙　建軍[1,a)]

概要：量子ソフトウェアは，量子コンピューティングシステムの可能性を最大限に引き出すために重要な役割を果たしているため，近年注目が高まっている．本稿では，「量子ソフトウェア工学」という用語を定義し，量子ソフトウェアのライフサイクルを紹介する．それに基づいて，量子ソフトウェアの設計，テスト，保守で利用可能な技術を調査し，量子ソフトウェア工学の課題を提示する．

キーワード：量子ソフトウェア工学，量子ソフトウェアのライフサイクル，量子ソフトウェアの設計，量子ソフトウェアのテスト，量子ソフトウェアの保守

1. はじめに

量子コンピューティングは，多くの分野でブレークスルーが期待される急速に発展している分野である [1]．量子コンピューティングは，量子力学の原理を利用して情報を処理し，特定のタスクを古典的なコンピューティングよりもはるかに高速に実行できる可能性を秘めている．量子コンピューティングの基本は，量子ビット（qubit）です．古典的なコンピュータビットとは異なり，量子ビットは 0 と 1 の重ね合わせである状態を割り当てることができる．古典的なアルゴリズムと比較して，量子アルゴリズムは指数関数的な加速で特定の問題を解決する可能性を持っている．

古典的なコンピューティングのように，量子コンピューティングは，多くの分野のアプリケーションに適用可能であり，広い影響を持つだろう．量子コンピュータが古典コンピュータを凌駕すると期待される応用には，二種類がある．一種類は，大量の並列計算を必要とする最適化, 暗号化, 及び機械学習という問題である．その他の応用としては，自然界の量子問題を効率的かつ正確にシミュレーションする必要がある問題への応用が挙げられる．

量子ハードウェアの急速な発展に伴い，量子デバイスは量子サービス（QaaS）を介して研究者や専門家に提供されるようになった．今こそ，量子技術の恩恵を享受するために，量子ソフトウェア工学に注力すべきであると考えている．一方，量子コンピューティングの様々な応用領域では，量子ソフトウェア工学の方法論や技術が緊急に必要とされている．そのため，量子ソフトウェアを効率的に開発するための方法，ツール，プロセスを考案することに焦点を当てた量子ソフトウェア工学のためのコミュニティを構築することが急務であると考えている．

量子ソフトウェア開発技術（quantum software development, QSD）は, 量子ソフトウェアのライフサイクル全体を通して量子ソフトウェアを作成するための手段を提供することを目的としている．多くの量子プログラミング手法が利用可能であり，例えば，Scaffold [2]，Qiskit [3]，Q# [4]，ProjectQ [5] と Quipper [6] 等がある．さらに，量子ソフトウェア開発の初期段階でも応用が進んでいる [7], [8]．ソフトウェア技術者は，各段階で様々な技術を利用することができるが，量子ソフトウェアシステムの設計には大きな課題がある．各開発段階で，ソフトウェアエンジニアは開発するアプリケーションに最適な量子ソフトウェア技術を採用する必要がある．技術の選択は，システム要求，組織慣行，ツールや開発環境によって課される制約，量子プログラミングの性質など，様々な要因によって決定される．これは，複数の技術が各段階で互いに連携して採用される可能性があることを意味している．量子ソフトウェア開発技術の成熟に伴い，量子ソフトウェアシステムの開発を支援するためのガイドラインが必要とされている．

本稿では，主に量子ソフトウェア工学の問題に焦点を当てた過去の研究をまとめ，設計，テスト，保守の 3 つの側面に分けて文献を整理した．これまでにも，量子ソフトウェア工学のいくつかの側面に関連した調査が行われてきたが，量子ソフトウェアの設計，テスト，保守を含む量子ソフトウェア開発の調査に焦点を当てた先行研究はない．本稿の主な内容は以下の通りである．

- 量子ソフトウェア工学を定義し，量子ソフトウェア開

[1] 九州大学
Kyushu University
[a)] zhao@ait.kyushu-u.ac.jp

発を支援するための量子ソフトウェアのライフサイクルモデルを紹介する．
- 量子ソフトウェアの設計，テスト，保守など，量子ソフトウェア工学に関する研究を調査する．
- 量子ソフトウェア工学の今後の課題，有望な研究の方向性を議論する．

以下，2 節で，量子プログラミングを簡単に紹介する．3 節で量子ソフトウェア工学の定義，量子ソフトウェア開発のための量子ソフトウェアのライフサイクルについて提案し，簡単に紹介する．4 節から 6 節まで，量子ソフトウェアの設計，テスト，保守の現状を概観する．7 節では，量子ソフトウェア工学について，これからの課題と機会を議論する．最後に 8 節でまとめについて述べる．

2. 量子プログラミング

量子プログラミングとは，特定の計算結果を得るために実行可能な量子コンピュータプログラムを設計し，構築するプロセスです．量子プログラミングの取り組みは，他の量子ソフトウェア開発技術よりも先行しているため，まず量子プログラミングに焦点を当てる．本節では，量子プログラミングの概念，言語，意味論を簡単に紹介する．Ying [9] の優れた本では，量子プログラミングの基礎をより詳細に説明している．

2.1 概念

量子プログラムはコードブロックで構成されており，それぞれが古典的な構成要素と量子的な構成要素を含んでる．量子演算は，オペランドのノルムを保持して可逆的に演算を行う *unitary* 演算と，確率的に演算を行う *non-unitary* 演算に分けられる．量子コンピュータ上で実行される量子プログラムは，量子演算を実行するために量子ビットの量子レジスタを使用し，量子ビットの状態の測定値を記録し，量子演算子を条件付きで適用するために古典ビットのレジスタを使用する．従って，典型的な量子プログラムは，通常，2 種類の命令で構成されている．一つは，古典ビットの状態を操作して条件文を適用する古典命令と呼ばれるものである．もう一つは，量子ビットの状態で動作し，量子ビットの値を測定する量子命令と呼ばれるものである．

2.2 言語

初期の量子プログラミング言語開発は，Deutsch [10] によって提案された量子チューリングマシン (QTM) モデルの探索に焦点を当てていったが，量子コンピュータをプログラミングするための実用的なツールには至らなかった．このような状況から，量子回路モデルは，急速に量子プログラミングの原動力となった．回路設計だけではなく，実用的な言語として構築するために，Knill [11] は，量子プログラミングのための疑似コード概念と，量子システムが古典的なコンピュータによって制御される量子ランダムアクセスマシン（QRAM）のモデルを提案した．このモデルは，その後の量子プログラミング言語の設計に影響を与えた．Omer [12] は，1998 年に C のような構文を持つ最初の実用的な量子プログラミング言語 QCL を開発した．それ以来，量子コンピュータをプログラミングするための様々なタイプの言語パラダイムの観点から，多くの量子プログラミング言語が設計され，実装されてきた．例えば，命令型言語は Scaffod [2], Q# [4], $Q|SI\rangle$ [13], ProjectQ [14] と Qiskit [15] がある．関数型言語は Quipper [6]，LIQUi|⟩ [16] がある．量子プログラミング言語の詳細については，Ying [9] の本を参照してください．

2.2.1 意味論

量子プログラミングの意味論も広く研究されており，最近では多くのアプローチが提案されている [9], [17], [18]．これらの方法は 3 つの種類に分類できる，即ち，操作的意味論，公理的意味論，及び表示的意味論である．量子プログラミング言語の意味論の詳細な議論については，量子プログラミング言語に関する本 [9] を参照することができる．

3. 量子ソフトウェア工学

本節では，量子ソフトウェア工学を定義し，量子ソフトウェア開発のためのライフサイクルを紹介する．

3.1 量子ソフトウェア工学の定義

量子プログラミング言語はエキサイティングな発展を遂げているが，量子ソフトウェア開発における問題の主な原因はコーディングではない．要求や設計の問題の方がはるかに一般的であり，修正にはコストがかかる．従って，量子ソフトウェア開発技術の焦点は，量子プログラミングの問題に限定されるべきではなく，量子ソフトウェア工学の他の側面にも焦点を当てる必要がある．量子ソフトウェア開発手法は，分析や設計の複雑さを克服し，分析や設計の再利用を実現するための有望な手段である．もし，量子ソフトウェア開発が量子プログラミング以上のものであることが認められるのであれば，量子ソフトウェア開発の他の側面に対応するために，ライフサイクルを含めた全く新しいアプローチを採用しなければならない．

ソフトウェア工学は，行動科学，工学，プロジェクト管理，コンピュータサイエンス，プログラミング，コスト管理，数理科学にルーツを持つ問題解決プロセスである．[19] によると，ソフトウェア工学のプロセスは次のように定義される．

”A software process is defined as a coherent set of policies, organization structures, technologies, procedures and artifacts that are needed to conceive, develop, deploy, and

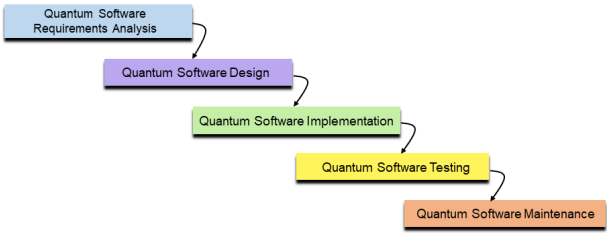

図 1 量子ソフトウェアのライフサイクル

maintain a software product."

IEEE では，古典的なソフトウェア工学について，以下のように定義している．

"(1) The application of a systematic, disciplined, quantifiable approach to development, operation and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1)."

Fritz Bauer [20] は，ソフトウェア工学の定義を次のように述べている．

"The establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines."

その後，多くのソフトウェア工学の定義が開発されてきたが，Bauer の定義は現在でも広く受け入れられており，本論文における量子ソフトウェア工学の定義の基礎となるものである．

本稿では，量子ソフトウェアの定義として，実行可能な量子プログラムだけでなく，量子ソフトウェアの開発，運用，保守に必要なライブラリやドキュメントを定義している．このような広い視野で量子ソフトウェアを定義することで，量子ソフトウェア開発プロセスの重要な一部として，タイムリーな文書化を考慮する必要性を強調したいと考えている．古典的なソフトウェア工学の定義に触発され，量子ソフトウェア工学を次のように定義する．

"Quantum software engineering is the use of sound engineering principles for the development, operation, and maintenance of quantum software and the associated document to obtain economically quantum software that is reliable and works efficiently on quantum machines."

本定義では，量子ソフトウェア開発における 3 つの重要な課題を取り上げたいと思う．第一に，量子ソフトウェアの開発には，"健全な工学的原則 "を適用することが重要である．第二に，量子ソフトウェアは "経済的に "構築されなければならない．最後に，量子ソフトウェアは "信頼性の高い "量子マシン上で "効率的に "動作する必要がある．

3.2 量子ソフトウェアのライフサイクル

ソフトウェアライフサイクルモデルは，高品質なソフトウェアの設計と構築を支援するソフトウェア開発プロセスのための参照モデルとして定義することができる．いくつかのソフトウェアライフサイクルモデルは，古典的なソフトウェア開発のために，提案されている落水型モデル，プロトタイピングモデル，およびらせん型モデルを含む．それらの中で，古典的なソフトウェア開発のための最も広く受け入れられたモデルは，落水型のライフサイクルモデルであり，他のモデルはこれを改良したものが多い．

本稿では，量子ソフトウェアシステムの設計と構築を支援するために，量子ソフトウェア開発プロセスの逐次的なライフサイクルモデルを提案する．量子ソフトウェアのライフサイクルモデルは，システムレベルから始まり，要求分析，設計，実装，テスト，保守と進んでいく．更に，このモデルは，必要に応じて，より多くのフェーズを追加することができるという意味で，拡張性もある．**図 1** は，次のようなフェーズからなるライフサイクルを示している．

- 量子ソフトウェアの要求分析（Requirements analysis）
- 量子ソフトウェアの設計（Design）
- 量子ソフトウェアの実装（Implementation）
- 量子ソフトウェアのテスト（Testing）
- 量子ソフトウェアの保守（Maintenance）

次に，量子ソフトウェア開発の観点から，ライフサイクルモデルの各フェーズを簡単に紹介する．

3.2.1 要求分析

量子ソフトウェアのライフサイクルモデルは，利害関係者が目標を達成するために開発が必要なソフトウェアの要求を議論する要求分析フェースから始まる．要求分析フェーズでは，各要求の詳細を把握し，作業範囲と各要件をどのように満たすかを全員が理解できるようにすることを目的としています．この分析により，測定可能な量子ソフトウェア要求のセットが作成され，サプライヤーの視点から，利害関係者の要件を満たすために量子ソフトウェアシステムが持つべき特性，属性，機能および性能要求を特定する．量子ソフトウェアの設計，実装，テスト，保守などのライフサイクルの後の段階では，量子ソフトウェアのライフサイクルを通して要求分析が継続されることを前提としている．

3.2.2 設計

量子ソフトウェアの設計は，ライフサイクルの第二段階であり，ユーザーの要求を適切な形に変換するプロセスであり，量子ソフトウェアの実装（コーディング）のプログラマを支援する．

古典的なソフトウェア設計のように，量子ソフトウェア設計は，アーキテクチャ設計と詳細設計のように 2 つの段階を分けることもできる．アーキテクチャ設計は，量子ソフトウェアのコンポーネント，機能，相互作用（インター

フェース）を定義し，量子ソフトウェアシステムの開発のためのフレームワークを確立する．詳細設計は，量子ソフトウェアコンポーネントの内部（アルゴリズム，処理ロジック，データ構造，データ定義）を記述するために，アーキテクチャ設計を洗練させ，使用する．詳細設計は，実装（コーディング）に十分な記述ができた時点で完了となる．

3.2.3 実装

要求と設計活動が完了した後，ライフサイクルの次のフェーズは，量子ソフトウェアの実装または開発である．このフェーズでは，開発者は前のフェーズで議論した要件と設計に基づいてコーディングを開始する．また，開発者は，各コンポーネントの単体テストを実施して，自分が書いた新しいコードをテストし，お互いのコードをレビューし，ビルドを作成し，量子ソフトウェアを環境に展開する．この開発サイクルは，要件が満たされるまで繰り返される．

3.2.4 テスト

テストは，顧客にソフトウェアを提供する前の量子ソフトウェアのライフサイクルの最後の段階である．テスト中，テスターは，要件に基づいて量子ソフトウェアのテストを開始する．目的は，ソフトウェアの欠陥を見つけることと，量子ソフトウェアの要件分析段階のドキュメントに基づいて，ソフトウェアが期待通りに動作するかどうかを確認することである．

3.2.5 保守

量子ソフトウェアのメンテナンスは，量子ソフトウェアのライフサイクルの最終段階であり，量子ソフトウェア製品の納品後に行われるすべての変更と更新を表している．

4. 量子ソフトウェアの設計

量子アルゴリズムの開発は，量子コンピューティングの本質的な特徴である重畳やエンタングルメントのため，古典的なものに比べて非常に難しいと考えられています．そのため，量子アルゴリズムの新しい設計原理が強く求められている．ソフトウェア設計は古典的なソフトウェア工学の重要な段階として認識されているが，量子ソフトウェア設計の原理と方法論に関する研究はまだ始まったばかりである．

4.1 量子ソフトウェアのモデリング

統一モデリング言語（UML）は，古典的なソフトウェア工学の分野でよく知られている汎用的なモデリング言語である．古典的なソフトウェア開発ライフサイクルの設計を可視化するための標準的な方法を提供する．最近，Pérez-Delgado と Perez-Gonzalez [7] は，量子ソフトウェアシステムをモデル化するために UML を拡張する手法を提案した．この拡張は，2 種類の UML 図（クラス図とシーケンス図）を対象としている．

4.2 量子ソフトウェアの仕様

量子コンピュータは，コンピュータ科学者やソフトウェア開発者よりも物理学者になじみの深い量子力学に依存している．従って，量子コンピュータについて推論する前に，その基礎となる理論を知っておく必要がある．量子力学の原理を伝えることは，その反直感的な性質のため，簡単なことではない．

最近，Cartiere [21] は，量子アルゴリズムのための正式な仕様言語を定義する作業を提案した．その目的は，量子コンピューティングの基本的な表記法を表現するために使用できる形式仕様言語を導入することであり，量子コンピューティング科学者や量子ソフトウェア技術者にとってより適したものとなっている．この言語は，古典的ソフトウェアシステムの形式仕様言語として研究されている Z 言語をベースにしている．この言語は，量子プログラムの記述をサポートするためのユニタリ変換を行う Identity ゲート，Pauli-X ゲート，Phase Shift ゲート，Pauli-Z ゲート，Hadamard ゲート，C-NOT ゲートなどの基本的な量子ゲートを表現することができる．Cartiere はまた，この言語を使って，Deutsch-Jozsa アルゴリズムのような簡単な量子アルゴリズムを記述する方法を示した．

4.3 量子ソフトウェアのパターン言語

Leymann [22] は，量子アルゴリズムの開発を支援するためのパターン言語を提案した．また，量子ソフトウェア開発において繰り返し発生する問題の解決策を文書化する必要性があり，量子アルゴリズムで使用される基本的な ”トリック ”を非公式にまとめた [23], [24] で述べられている．本研究の主な貢献は，これを体系化し，量子アルゴリズムのためのソフトウェア工学分野の課題とすることである．Leymann は，主にゲートモデルに基づく既存の量子アルゴリズムから，10 種類の基本パターンを同定した．各パターンは，*name*, *intend*, *icon*, *problem statement*, *context*, *solution*, *know uses*, and *next* の 8 つの要素で記述されている．

5. 量子ソフトウェアのテスト

量子コンピュータは古典コンピュータと比べて，量子重ね合わせ，量子もつれ，量子複製不可能などの性質が非常に異なる [23] ので，量子ソフトウェアの動作を予測することは非常に困難である [25]．そのため，新しい量子ソフトウェアのテストやデバッグ技術の開発が必要とされている．最近では，バグの発見や量子ソフトウェアのテスト・デバッグのための研究が盛んに行われている．本節では，広義の意味での量子ソフトウェアテストの現状を，量子ソフトウェアのテストとデバッギングの観点から概観する．

5.1 テスト

テストとは，プログラムを実行してエラーを発見することであり，ソフトウェア開発における品質保証を支援する上で非常に重要なプロセスである．本節では，量子ソフトウェアをテストするための最新の技術を簡単に紹介する．

Miranskyy と Zhang [26] は，量子プログラムの検証と検証と同様に，ホワイトボックステストとブラックボックステストに関連するいくつかの課題を提案した．その結果，既存のソフトウェアエンジニアリング手法のいくつかは，量子コンピューティング領域に容易に移行可能であることが示された（例：コードレビュー）．また，他の手法を量子コンピューティング領域に適用することは困難である（例：対話型デバッグ）．残りは導入しなければならない（例えば，検証プログラムの複雑さに依存した配置）．量子ソフトウェアのための特定のテスト方法（または戦略）を提案するのではなく，量子ソフトウェアのためのソフトウェア工学の実践を定義しようとした．この定義が手元にあれば，学術界や産業界のソフトウェア工学の専門家は，量子ソフトウェアの魅力的な分野である．また，量子コンピューティングの研究を，量子ソフトウェアのライフサイクルの残りのフェーズと同様に，他のテスト分野にも拡大していく．

Wang ら [27] は，coverage-guided fuzzing (CGF) と呼ばれる古典的なソフトウェアテストの既存技術を，量子ソフトウェアのテストに適応させた．彼らは，量子ソフトウェアのための検索ベースのテスト入力生成器である `QuanFuzz` を提案した．`QuanFuzz` の基本的な考え方は，量子ソフトウェアのテスト入力を評価するために量子的に敏感な情報を定義し，より高いカバレッジを持つテストケースを生成するために行列生成器を使用することである．

Honarvar ら [28] は，Q#で書かれた量子ソフトウェアのためのプロパティベースのテスト手法である `QSharpCheck` を提案した．このために，Q#プログラムのプロパティを指定するテスト仕様言語を定義した．この言語では，テスト，テストプロパティ名とパラメータ，割り当てと設定，関数呼び出し，アサートとデアロケーションの 4 つの部分で表現される．また，テスト仕様言語に関連付けられた後条件やアサーションの種類の基礎となるアサーションの種類をいくつか特定した．これらの特性を基に，様々なタイプのテストケースを生成し，テストケースを実行して Q#プログラムをテストし，出力結果を確認することで，プログラムに問題がないかどうかを確認することができるようになった．

5.2 デバッグ

量子プログラミングにおいて，デバッグの方法論や技術は非常に重要である．本節では，量子プログラムのデバッグ手法を概観する．

5.2.1 デバッグの戦術

古典的なプログラムのデバッグでは，「Brute force」，「Backtracking」，「原因除去」，「原因消去」などの一般的な手法が広く使われている．これらのデバッグ手法は，量子プログラムのデバッグをサポートするためにも利用できる．

Miranskyy ら [29] は，量子プログラムのデバッグ戦術を分析し，確立された古典的なデバッグ技術を量子領域に適用する可能性について議論した．その結果，古典的なデバッギングの中でも，上述したように，主に 3 つのタイプのデバッギング手法を検討し，*backtracking* の手法，特にコードレビューや検査に基づく手法は，量子デバッグにも適用できる可能性があると結論付けた．

5.2.2 アサーションベースのデバッグ

最近，量子プログラムをデバッグするためのアサーションベースのアプローチがいくつか提案されている．Huang と Martonosi [30], [31] は，量子プログラム（またはアルゴリズム）の実装経験に基づいて，量子プログラムにおけるいくつかのタイプのバグをまとめた．また，バグのない量子プログラムを開発するために，プログラミングやアサーションの観点から，バグを防ぐための防御戦略を提案した．さらに，Huang と Martonosi は，いくつかの古典的観測値に対する統計的検定に基づいた量子プログラムのための統計的アサーションを提示した．これにより，量子プログラムの状態が古典的な状態，重ね合わせ状態，量子もつれ状態のいずれかで期待される値と一致するかどうかを決定することができる．これに基づいて，量子ソフトウェアのアサーションを，古典的なアサーション，重ね合わせアサーション，及びもつれアサーションの 3 つの種類に分類した．また，既存の量子プログラミング言語である Scaffold [2] に量子アサーションを指定する機能を追加し，量子プログラムシミュレータ QX [32] でアサーションを確認するツールを開発した．

Zhou，Byrd と Liu [33], [34] は，Huang と Martonosi のアサーションベースアプローチの重大な制限は，デバッグ中の各測定はプログラムの実行を停止しなければならず，実際の計算結果を測定する場合にはアサーションは実行の集約を必要とすることを観察した．この制限を克服するために，量子誤り訂正（QEC）と非破壊識別（NDD）に触発され，量子ソフトウェアの実行時アサーションチェックを支援するための適切な量子回路を構築する手法を提案した．この手法の基本的なアイデアは，テスト対象の量子ビットの情報を間接的に収集するためにアンシラ量子ビット（いくつかの追加量子ビット）を使用し，テスト対象の元の量子ビットを直接測定することではなく，それらのアンシラ量子ビットを測定することで，実行時のアサーションチェック時にプログラムの実行が中断されることを避けることができる．

6. 量子ソフトウェアの保守

量子ソフトウェア保守の主な目的は，納品後にソフトウェアアプリケーションを修正・更新し，障害を修正し，性能を向上させることである．この分野の研究は，最近では，主に既存の古典的なソフトウェアシステムを新しい量子アルゴリズムと統合するためのリエンジニアリングに焦点を当てている．

量子コンピュータの開発は大きく進展しているが，短期的には（初期コストの高さなどの理由から）すべてのことに量子コンピュータを利用できないことが明らかになっている．その代わりに，古典的なコンピュータからクラウド上の遠隔地にある量子コンピュータに特定の呼び出しを行い，いくつかの重要な問題を解決するために量子コンピュータを利用することが一般的である [35]．この場合，多くの企業では，最初の量子アルゴリズムや将来の量子アルゴリズムを既存の古典的な情報システムと統合して移行する必要がある．そのため，量子コンピューティングの移行や，古典的なソフトウェアと量子ソフトウェアの共存に関連する問題に対処するために，リエンジニアリングを再検討する必要がある．

この問題を解決するために，P'erez-Castillo [36] は，古典情報システムを既存または新規の量子アルゴリズムとともに再構築し，古典情報システムと量子情報システムの両方を組み合わせた対象システムを提供するソフトウェア近代化アプローチ（モデル駆動型リエンジニアリング） [37] を提案した．古典ソフトウェア工学におけるソフトウェアモダナイゼーション手法は，業務知識を保持したままソフトウェアの移行・進化を実現できる有効な仕組みであることが証明されている．提案するソリューションは，既存のよく知られた標準規格である Unified Modelling Language (UML) や Knowledge Discovery Metamodel (KDM) に基づいた体系的なものである．この解決策は，新たな量子情報システムの開発を減らすことができるという利点がある．

リエンジニアリングに関するもう一つの問題は，量子計算プリミティブ（例えば，量子ソフトウェアコンポーネント）を既存の古典的なソフトウェアシステムにどのように統合するかということである．量子コンピュータ (QC) は，従来の技術や手法とは大きく異なるため，既存のソフトウェアシステムに統合するためには，アルゴリズムの実装レベルでこの問題を解決するだけでなく，ソフトウェア工学で研究されている多くのより広範な問題を含んでいる必要がある．Krüger と Mauerer [38] は，量子アニーラー (QA) に実装されたブール満足度問題 (SAT) のための量子ソフトウェアコンポーネントを，既存のソフトウェアシステムにどのように拡張するかについての事例研究を行った．この事例研究では，量子コンポーネントの関連する品質尺度について議論し，数学的には等価だが構造的には異なる SAT を QA に変換する方法が，これらの品質に大きな違いをもたらすことを示した．

7. 今後の課題について

本節では，量子ソフトウェア工学の分野での課題について議論する．

7.1 量子ソフトウェアの設計

7.1.1 量子アーキテクチャ記述言語

アーキテクチャ記述言語 (ADL) [39] は，ソフトウェアシステムのアーキテクチャを表現するために使用することができる形式的な言語である．それらは，特定のソースモジュールの実装の詳細よりも，全体的なアプリケーションの高レベルの構造に焦点を当てている．これまでにいくつかの ADL が提案されており，古典的なソフトウェアアーキテクチャの形式的な表現と推論をサポートするためのものがある．最近のソフトウェアアーキテクトは，量子ソフトウェアシステムのアーキテクチャを指定する際に，量子問題を適切に扱うためのツールを持っていない．現在の ADL は，量子コンポーネントと古典コンポーネントとの接続を指定するためのプリミティブを提供していない．ADL を使用して，ソフトウェアアーキテクトは，コンポーネントを使用してシステムの機能性を指定し，コネクタを使用してコンポーネント間の相互作用を指定することができる．量子システムのアーキテクチャレベルでの設計のためには，現在の ADL を量子アーキテクチャ記述言語（qADL）に拡張し，量子ソフトウェアシステムのアーキテクチャを正式に記述する必要がある．

7.1.2 量子デザインパターン

デザインパターンとは，解決すべき繰り返しのデザイン問題，その問題に対する解決策，そしてその解決策が機能する文脈を記述したものである [40]．デザインパターンは長い間進化してきたものであり，ソフトウェア開発中に直面する特定の問題に対する最良の解決策を提供する．デザインパターンは，オブジェクト指向の分野で非常に有用であることが証明されており，検証されたコンポーネントの再利用性を通じて，アプリケーションの優れた設計を達成するのに役立ってきた．これらの経験から，量子ソフトウェアのための優れた再利用可能なデザインの要素を特定し，量子デザインパターンを用いてその経験を形式化することが重要であることがわかった．量子デザインパターンが特定されれば，量子ソフトウェアの開発に貢献することができ，量子ソフトウェアのための量子パターンカタログが特に有用になると期待している．

7.2 量子ソフトウェアの信頼性

本節では，量子ソフトウェアの信頼性に関するテスト，

デバッグ，及び検証を含めて議論する．

7.2.1 テスト

量子ソフトウェアシステムの体系的なテストは，量子ソフトウェアの構造的・行動的特性を反映した断層モデルに基づいて行われなければならない．量子プログラムをテストするための基準と戦略は，断層モデルの観点から開発されなければなりません．古典的なテストと同様に，量子ソフトウェアのテストでは，量子ソフトウェアの効果的で効率的なテストフレームワークを構築するために，以下の問題を厳密に探求し，それぞれの問題に対する答えをする必要がある．

- 量子ソフトウェアのテストカバレッジ基準をどのように定義するのか？
- 量子ソフトウェアのテストケースをどのように自動的に生成するのか？
- 量子ソフトウェアのテストデータの品質をどのように評価するのか？

7.2.2 デバッグ

量子計算の確率的な性質から，量子ソフトウェアのデバッガは，2 つの量子レジスタが同一であると仮定できないことを常に念頭に置いておく必要がある．このことから，より多くの研究が必要であることは明らかである． 最近，いくつかの研究が行われ，量子ソフトウェアのデバッグについていくつかの有望な初期結果が得られた．しかし，量子コンピューティングに適したデバッグ技術はまだ明らかにされていない．その結果，新たな手法の開発が必要とされている．一方，アサーションベースのデバッグは量子ソフトウェアのデバッグにおいて有望な方法のように思われるが，古典コンピューティングにおける異なる種類のデバッグ技術，例えば，対話型デバッグ，静的・動的解析，コードレビューと検査などまた，更なる調査の価値があると思われる [41]．これらの古典的なデバッグ技術は量子領域にも適用できるのか?

7.2.3 検証

形式検証は，量子プログラムにおいて重要な役割を果たしている．量子プログラムは，量子重ね合わせや量子もつれ，量子複製不可能などの特徴があるため，デバッグやテストが困難である．量子プログラムの正しさを保証するためには，新しい検証手法を開発する必要がある．現在，量子プログラムの検証には大きな進展があった [42], [43], [44] が，Rand*et al.* [44] が指摘したように，エラーを処理し，実行するハードウェアに関するエラーが発生しやすい量子プログラムを検証する新しい検証手法が必要である．さらに，量子コンパイルを検証する方法も必要である．

7.3 量子ソフトウェアの保守

ソフトウェア保守は，ソフトウェア開発時には欠かせない作業である．古典的なソフトウェアの保守方法，技術，ツールは十分に研究され，確立されているが，量子ソフトウェアの保守に関する研究はまだ始まったばかりである．量子ソフトウェアの保守プロセスは，少なくとも以下の 3 つの主要な問題に対処する必要があると考えている．

- 既存の量子ソフトウェアをどのように理解するのか？
- 既存の量子ソフトウェアをどのように変更するのか？
- 修正された量子ソフトウェアをどのように再検証するのか？

8. おわりに

短い歴史の中で，量子ソフトウェア開発は量子プログラミング言語の出現によって推進されてきた．量子ソフトウェア開発のためには，完全なソフトウェア工学の規律を確立することが不可欠である．本稿では，量トウェアのライフサイクルにおける量子ソフトウェアの設計，テスト，保守などの分野について，量子ソフトウェアの工学的な支援の現状を調査した．また，この分野における課題についても議論した．

参考文献

[1] Martonosi, M. and Roetteler, M.: Next Steps in Quantum Computing: Computer Science's Role, *arXiv preprint arXiv:1903.10541* (2019).
[2] JavadiAbhari, A., Patil, S., Kudrow, D., Heckey, J., Lvov, A., Chong, F. T. and Martonosi, M.: ScaffCC: Scalable compilation and analysis of quantum programs, *Parallel Computing*, Vol. 45, pp. 2–17 (2015).
[3] Research, I.: Qiskit, *Accessed on: April, 2020*, (online), available from ⟨https://qiskit.org⟩ (2017).
[4] Svore, K., Geller, A., Troyer, M., Azariah, J., Granade, C., Heim, B., Kliuchnikov, V., Mykhailova, M., Paz, A. and Roetteler, M.: Q#: Enabling Scalable Quantum Computing and Development with a High-level DSL, *Proceedings of the Real World Domain Specific Languages Workshop 2018*, pp. 1–10 (2018).
[5] Team, P.: ProjectQ, *Accessed on: April, 2020*, (online), available from ⟨https://projectq.ch/⟩ (2017).
[6] Green, A. S., Lumsdaine, P. L., Ross, N. J., Selinger, P. and Valiron, B.: Quipper: a scalable quantum programming language, *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation*, pp. 333–342 (2013).
[7] Pérez-Delgado, C. A. and Perez-Gonzalez, H. G.: Towards a Quantum Software Modeling Language, *First International Workshop on Quantum Software Engineering (Q-SE 2020)* (2020).
[8] Cartiere, C. R.: Quantum Software Engineering: Bringing the Classical Software Engineering into the Quantum Domain, Master's thesis, Department of Computer Science, University of Oxford (2013).
[9] Ying, M.: *Foundations of Quantum Programming*, Morgan Kaufmann (2016).
[10] Deutsch, D.: Quantum theory, the Church–Turing principle and the universal quantum computer, *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, Vol. 400, No. 1818, pp. 97–117 (1985).
[11] Knill, E.: Conventions for quantum pseudocode, Technical report, Los Alamos National Lab., NM (United

States) (1996).
[12] Ömer, B.: A procedural formalism for quantum computing, Master's thesis, Department of Theoretical Physics, Technical University of Vienna (1998).
[13] Liu, S., Wang, X., Zhou, L., Guan, J., Li, Y., He, Y., Duan, R. and Ying, M.: $Q|SI\rangle$: A Quantum Programming Environment, *Symposium on Real-Time and Hybrid Systems*, Springer, pp. 133–164 (2018).
[14] Steiger, D. S., Häner, T. and Troyer, M.: ProjectQ: an open source software framework for quantum computing, *Quantum*, Vol. 2, p. 49 (2018).
[15] 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., Chow, J. M., Córcoles-Gonzales, A. D., Cross, A. J., Cross, A., Cruz-Benito, J., Culver, C., González, S. D. L. P., Torre, E. D. L., Ding, D., Dumitrescu, E., Duran, I., Eendebak, P., Everitt, M., Sertage, I. F., Frisch, A., Fuhrer, A., Gambetta, J., Gago, B. G., Gomez-Mosquera, J., Greenberg, D., Hamamura, I., Havlicek, V., Hellmers, J., Łukasz Herok, Horii, H., Hu, S., Imamichi, T., Itoko, T., Javadi-Abhari, A., Kanazawa, N., Karazeev, A., Krsulich, K., Liu, P., Luh, Y., Maeng, Y., Marques, M., Martín-Fernández, F. J., McClure, D. T., McKay, D., Meesala, S., Mezzacapo, A., Moll, N., Rodríguez, D. M., Nannicini, G., Nation, P., Ollitrault, P., O'Riordan, L. J., Paik, H., Pérez, J., Phan, A., Pistoia, M., Prutyanov, V., Reuter, M., Rice, J., Davila, A. R., Rudy, R. H. P., Ryu, M., Sathaye, N., Schnabel, C., Schoute, E., Setia, K., Shi, Y., Silva, A., Siraichi, Y., Sivarajah, S., Smolin, J. A., Soeken, M., Takahashi, H., Tavernelli, I., Taylor, C., Taylour, P., Trabing, K., Treinish, M., Turner, W., Vogt-Lee, D., Vuillot, C., Wildstrom, J. A., Wilson, J., Winston, E., Wood, C., Wood, S., Wörner, S., Akhalwaya, I. Y. and Zoufal, C.: Qiskit: An Open-source Framework for Quantum Computing, (online), DOI: 10.5281/zenodo.2562111 (2019).
[16] Wecker, D. and Svore, K. M.: LIQUi—⟩: A software design architecture and domain-specific language for quantum computing, *arXiv preprint arXiv:1402.4467* (2014).
[17] Gay, S. and Mackie, I.: *Semantic Techniques in Quantum Computation*, Cambridge University Press (2010).
[18] Cho, K.: Semantics for a quantum programming language by operator algebras, *New Generation Computing*, Vol. 34, No. 1-2, pp. 25–68 (2016).
[19] Fuggetta, A.: Software process: a roadmap, *Proceedings of the Conference on the Future of Software Engineering*, pp. 25–34 (2000).
[20] Naur, P. and Randell, B.: Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th-11th October 1968 (1969).
[21] Cartiere, C. R.: Quantum Software Engineering: Introducing Formal Methods into Quantum Computing (2016).
[22] Leymann, F.: Towards a Pattern Language for Quantum Algorithms, *International Workshop on Quantum Technology and Optimization Problems*, Springer, pp. 218–230 (2019).
[23] Nielsen, M. A. and Chuang, I.: Quantum computation and quantum information (2002).
[24] Lipton, R. J. and Regan, K. W.: *Quantum algorithms via linear algebra: a primer*, MIT Press (2014).
[25] Chong, F. T., Franklin, D. and Martonosi, M.: Programming languages and compiler design for realistic quantum hardware, *Nature*, Vol. 549, No. 7671, pp. 180–187 (2017).
[26] Miranskyy, A. and Zhang, L.: On testing quantum programs, *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, IEEE, pp. 57–60 (2019).
[27] Wang, J., Gao, M., Jiang, Y., Lou, J., Gao, Y., Zhang, D. and Sun, J.: QuanFuzz: Fuzz Testing of Quantum Program, *arXiv preprint arXiv:1810.10310* (2018).
[28] Honarvar, S., Mousavi, M. and Nagarajan, R.: Property-based testing of quantum programs in Q#, *First International Workshop on Quantum Software Engineering (Q-SE 2020)* (2020).
[29] Miranskyy, A., Zhang, L. and Doliskani, J.: Is Your Quantum Program Bug-Free?, *arXiv preprint arXiv:2001.10870* (2020).
[30] Huang, Y. and Martonosi, M.: QDB: From quantum algorithms towards correct quantum programs, *arXiv preprint arXiv:1811.05447* (2018).
[31] Huang, Y. and Martonosi, M.: Statistical assertions for validating patterns and finding bugs in quantum programs, *Proceedings of the 46th International Symposium on Computer Architecture*, pp. 541–553 (2019).
[32] Khammassi, N., Ashraf, I., Fu, X., Almudever, C. and Bertels, K.: QX: a high-performance quantum computer simulation platform, *Proceedings of the Conference on Design, Automation & Test in Europe*, pp. 464–469 (2017).
[33] Zhou, H. and Byrd, G. T.: Quantum Circuits for Dynamic Runtime Assertions in Quantum Computation, *IEEE Computer Architecture Letters*, Vol. 18, No. 2, pp. 111–114 (2019).
[34] Liu, J., Byrd, G. T. and Zhou, H.: Quantum circuits for dynamic runtime assertions in quantum computation, *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, pp. 1017–1030 (2020).
[35] Stepney, S., Braunstein, S. L., Clark, J. A., Tyrrell, A., Adamatzky, A., Smith, R. E., Addis, T., Johnson, C., Timmis, J., Welch, P. et al.: Journeys in non-classical computation II: initial journeys and way points, *The International Journal of Parallel, Emergent and Distributed Systems*, Vol. 21, No. 2, pp. 97–125 (2006).
[36] Pérez-Castillo, R.: Reengineering of Information Systems toward Classical-Quantum Systems, *International Workshop on the Quantum Software Engineering & Programming* (2020).
[37] Seacord, R. C., Plakosh, D. and Lewis, G. A.: *Modernizing legacy systems: software technologies, engineering processes, and business practices*, Addison-Wesley Professional (2003).
[38] Krüger, T. and Mauerer, W.: Quantum Annealing-Based Software Components: An Experimental Case Study with SAT Solving, *International Workshop on the Quantum Software Engineering & Programming* (2020).
[39] Clements, P. C.: A survey of architecture description languages, *Proceedings of the 8th international workshop on software specification and design*, IEEE, pp. 16–25 (1996).
[40] Gamma, E., Helm, R., Johnson, R. and Vlissides, J.: *Design Patterns Elements of reusable object-oriented sofware*, Addison Wesley (1995).
[41] Mosca, M., Roetteler, M. and Selinger, P.: Quantum Programming Languages (Dagstuhl Seminar 18381), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2019).

[42] Chadha, R., Mateus, P. and Sernadas, A.: Reasoning about imperative quantum programs, *Electronic Notes in Theoretical Computer Science*, Vol. 158, pp. 19–39 (2006).
[43] Ying, M.: Floyd–hoare logic for quantum programs, *ACM Transactions on Programming Languages and Systems (TOPLAS)*, Vol. 33, No. 6, pp. 1–49 (2012).
[44] Rand, R., Hietala, K. and Hicks, M.: Formal Verification vs. Quantum Uncertainty, *3rd Summit on Advances in Programming Languages (SNAPL 2019)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2019).