Quantum Software Testing
A Brief Introduction

Shaukat Ali[1, 2] and Tao Yue[1]

[1]Simula Research Laboratory, Oslo, Norway

[2]Oslo Metropolitan University, Oslo, Norway

simula

May 17, 2023 ICSE 2023 Technical Briefing

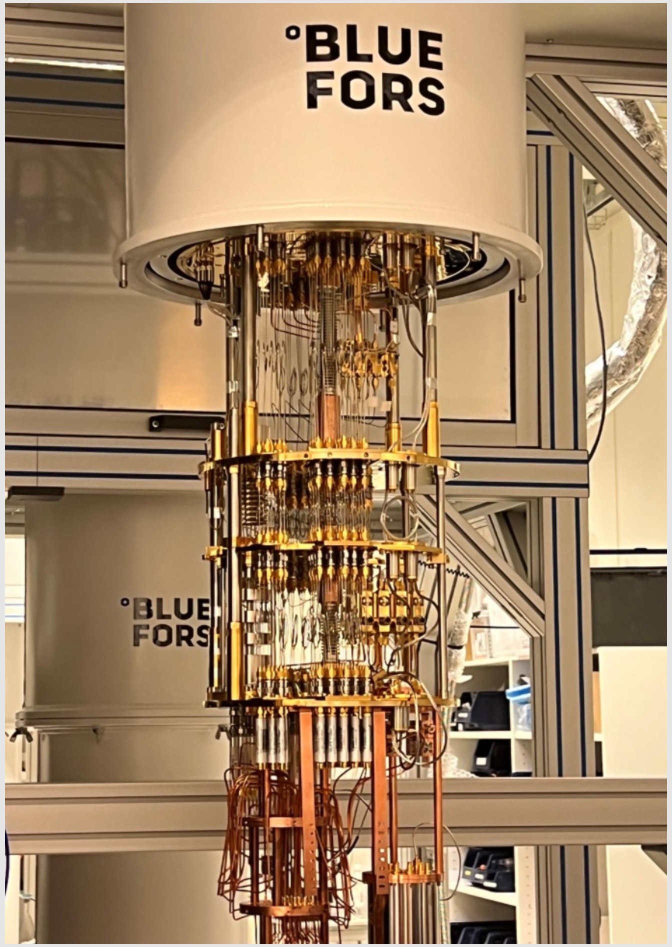
Outline

- Quantum Computing: A Brief Introduction
- Quantum Software Engineering: Our Vision
- Quantum Software Testing: the State of the Art
- Quantum Software Testing Techniques

simula

Quantum Computing (QC)

- QC promises to revolutionize classical computing.
- Quantum Computers
  - Gate-based quantum computers, e.g., IBM's Osprey, Google's Sycamore,
  - Annealing-based quantum computers, e.g., D-Wave
  - Photonic quantum computers: e.g., USIC's Jiuzhang, Xanadu's X24…
- Platforms
  - IBM Quantum Experience
  - D-Wave
  - Quantum Inspire from QuTech
  - Microsoft Quantum computing platform…
- High level programming languages
  - OpenQL by TU Delft, Q# by Microsoft, Qiskit by IBM, Cirq by Google



QAL 9000 quantum computer
Chalmers/Wallenberg Centre for Quantum Technologies, Sweden

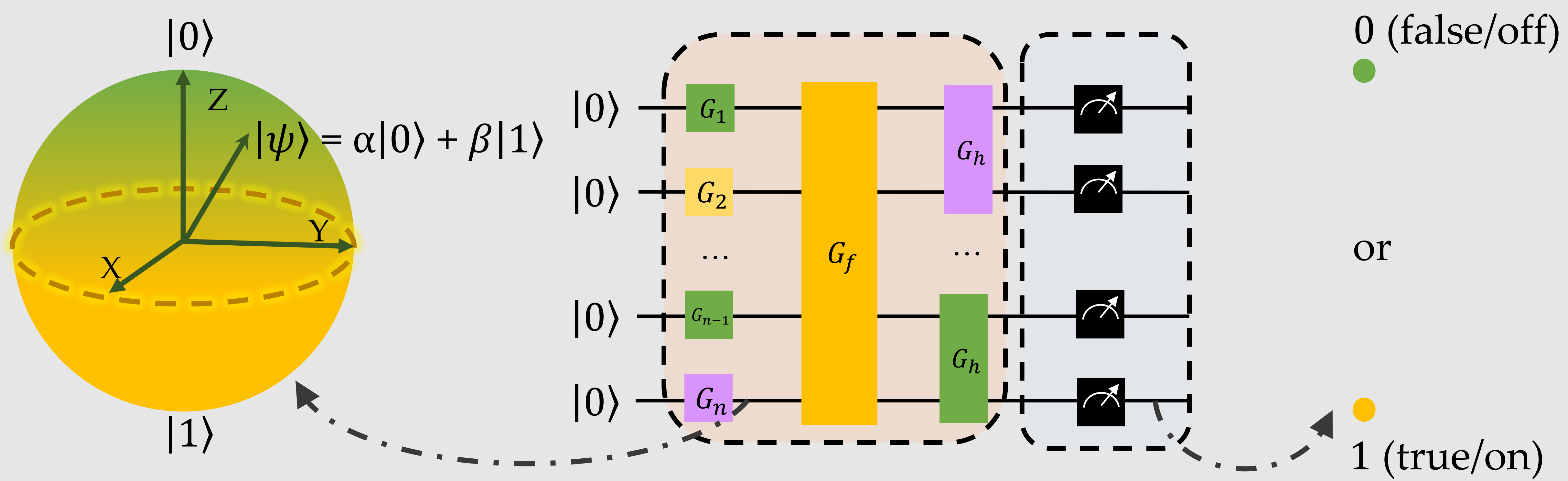
<u>QC is becoming a reality!</u>

Why quantum is different?

Qubit

Quantum Circuit

Bit



Bloch Sphere (1 qubit)

Quantum Gates

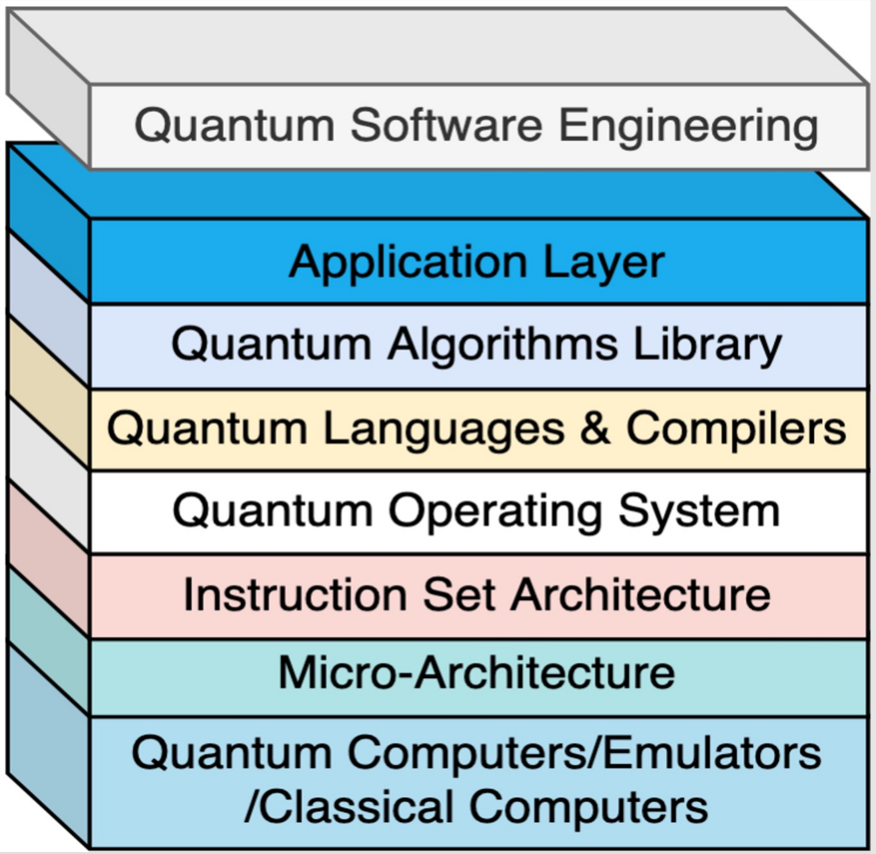
Measurements

simula

Quantum Software Engineering

simula

Quantum Software Engineering



Layered QC Architecture
(Prof. Koen Bertels's Vision)

Quantum software is at the core of the promised revolutionary QC applications.

Quantum software engineering enables cost-effective and scalable development of dependable quantum software.

simula

Shaukat Ali, Tao Yue, and Rui Abreu. 2022. When software engineering meets quantum computing. Commun. ACM 65, 4 (April 2022), 84–88. https://doi.org/10.1145/3512340

Why Quantum Software Engineering?

- **Application domains** are hard to comprehend.
- **Quantum software engineers** need to have *basic* knowledge about quantum mechanics, algorithms and their analysis, and more.
- Therefore, we need **tools**, **methodologies**, **standards**, **education**, etc. to help.

Chemistry

AI and ML

Drug Design & Development

Weather Forecasting

Financial Modeling

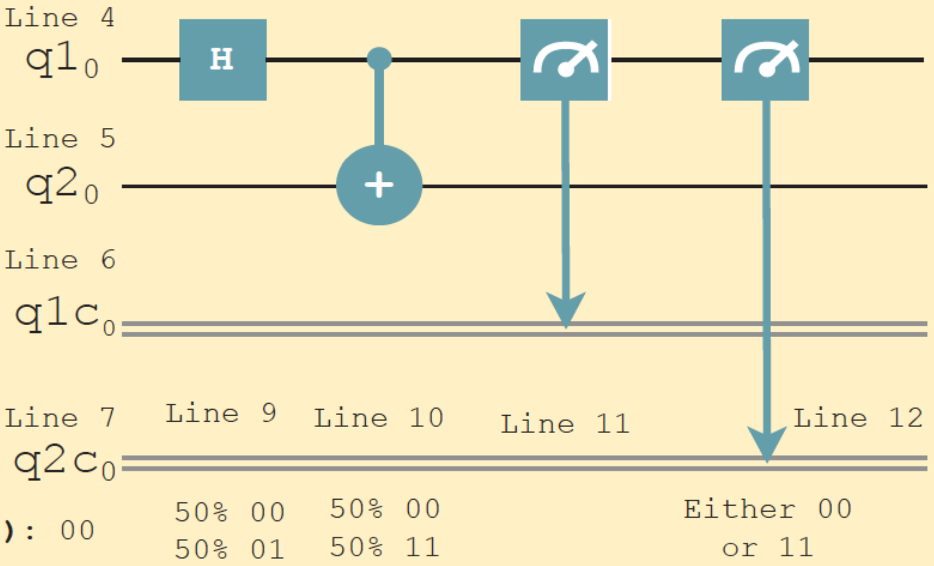
simula

Quantum program, circuit and its execution

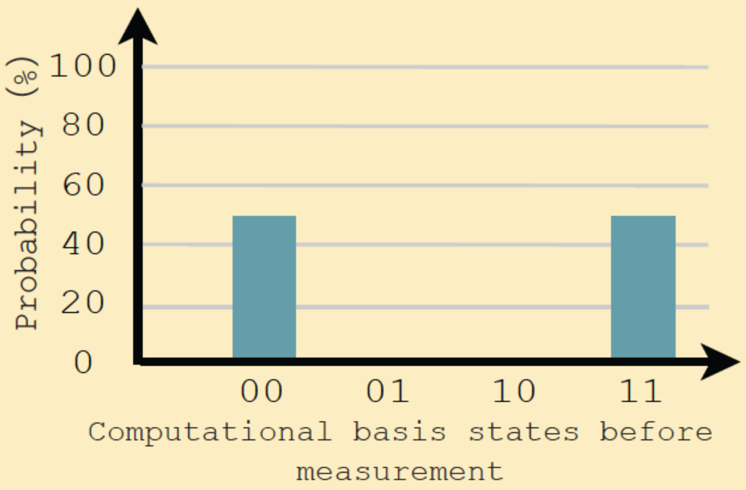
Code

```
1  OPENQASM 2.0;
2  include "qelib1.inc";
3
4  qreg q1[1];
5  qreg q2[1];
6  creg q1c[1];
7  creg q2c[1];
8
9  h q1[0];
10 cx q1[0],q2[0];
11 measure q1[0] -> q1c[0];
12 measure q2[0] -> q2c[0];
```

Quantum Circuit

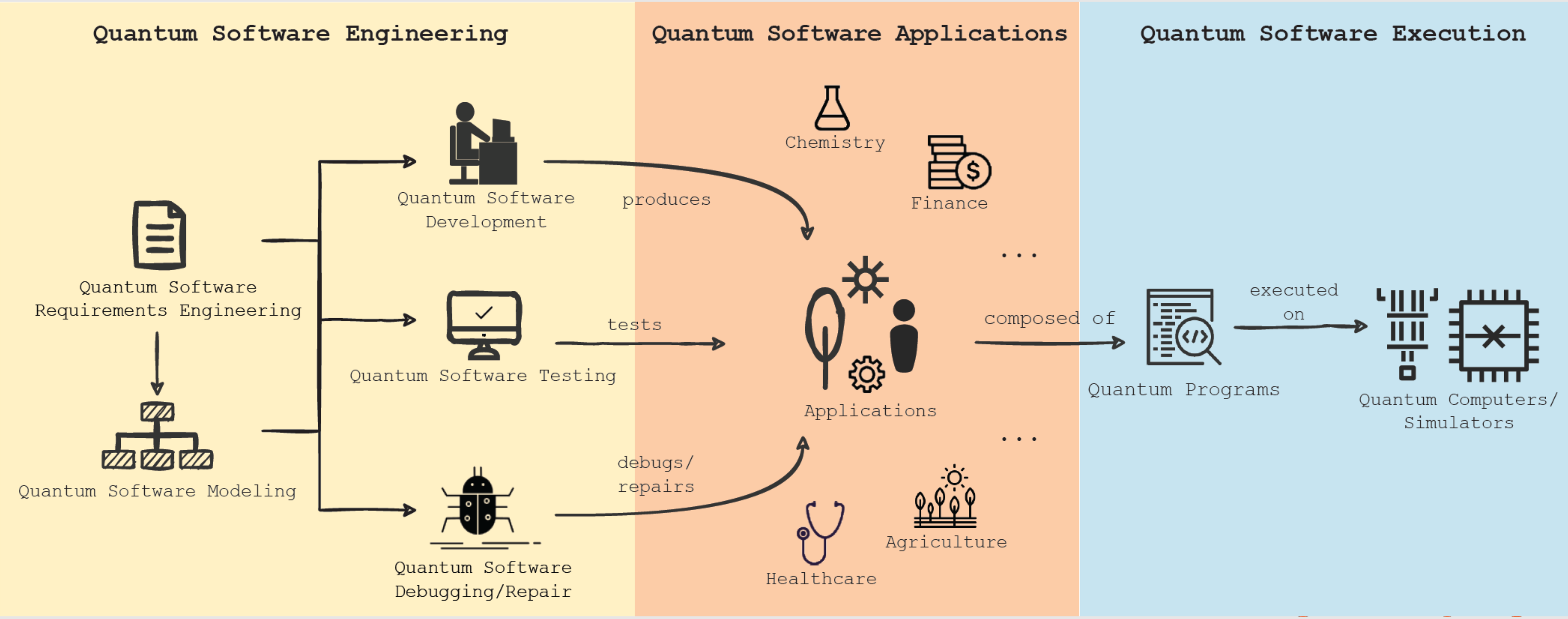


Execution Result



Description:
Lines 4-5 initialize two quantum registers (q1 and q2) in 0 states. Each register holds one qubit. **Lines 6-7** initialize two classical registers that will store the qubit values after the measurement (**Lines 11-12**). **Line 9** puts q1 in superposition with the Hadamard gate (h), whereas **Line 10** entangles q1 and q2 with the Conditional NOT gate (cx). As a result, whenever q1 and q2 are measured (e.g., in **Lines 11-12**), they will have the same values, i.e., either 00 or 11. Each value, i.e., 00 or 11 has an equal probability to be observed. Note that for simplicity, we show the states for the quantum circuit in terms of only probabilities and not as state vectors.

Developing dependable (quantum) software entails following a software development life cycle.



Shaukat Ali, Tao Yue, and Rui Abreu. 2022. When software engineering meets quantum computing. Commun. ACM 65, 4 (April 2022), 84–88. https://doi.org/10.1145/3512340

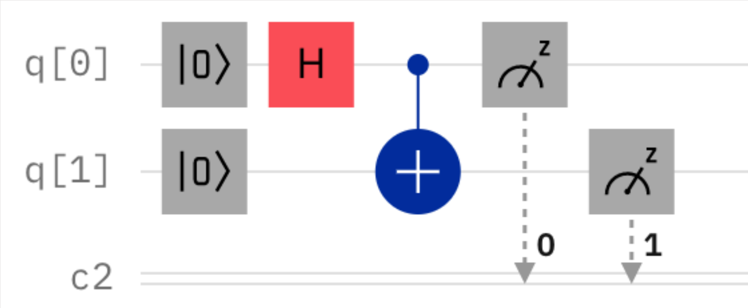
Requirements Engineering

- Quantum application **domains** are often **complex**.
- Need to ease **communication** among **stakeholders**, raise the level of **abstraction** in understanding domains and linking them to downstream activities.

Requirements engineering for quantum software is an uncharted area of research!

simula

Abstraction and Modeling



mapping

```
OPENQASM 2.0;
include "qelib1.inc";

qreg q[2];
creg c[2];

reset q[0];
h q[0];
reset q[1];
cx q[0],q[1];
measure q[0] -> c[0];
measure q[1] -> c[1];
```

From: IBM Quantum Composer

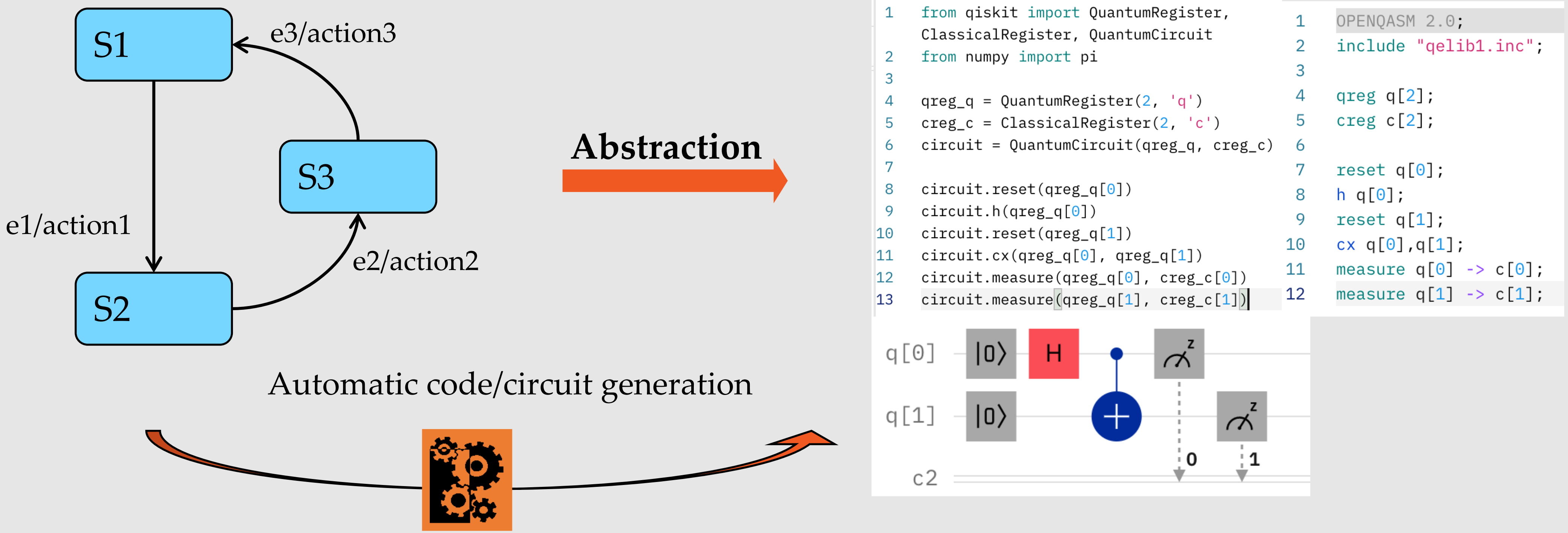
mapping

```
from qiskit import QuantumRegister,
ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(2, 'q')
creg_c = ClassicalRegister(2, 'c')
circuit = QuantumCircuit(qreg_q, creg_c)

circuit.reset(qreg_q[0])
circuit.h(qreg_q[0])
circuit.reset(qreg_q[1])
circuit.cx(qreg_q[0], qreg_q[1])
circuit.measure(qreg_q[0], creg_c[0])
circuit.measure(qreg_q[1], creg_c[1])
```

There is no abstraction!

simula

Abstraction and Modeling



Novel and intuitive **QSE methodologies**, with:

- **Quantum modeling notations**
- **V&V with quantum software models**
- **Code/circuit generation**

simula

Quantum Software Testing

- Software quality assurance
  - **Process** of ensuring that a software product meets and complies with established and standardized **quality specifications**.
- Testing
  - “Software testing is **a way to assess the quality of the software and to reduce the risk of software failure in operation**.” – from ISTQB

Quantum software testing is challenging, because:

- Computation in superpositions
- Use of advanced features (e.g., entanglement)
- Destructive measurements
- Lack of precise test oracles…

simula

Quantum Software Testing

- Research actions being taken or to be taken:
  - Define and check quantum test oracles without destroying superposition
  - Cost-effectively find test data to break a quantum program
  - Devise noise-aware testing techniques
  - Build theoretical foundations on coverage criteria, test models, and test strategies, etc.
  - Need practical applications, extensive empirical evaluations
  - Need benchmarks…

Quantum Software Testing
State of the Art

simula

Quantum Software Testing at ICSE

2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)

On Testing Quantum Programs

Andriy Miranskyy and Lei Zhang
Department of Computer Science, Ryerson University, Toronto, Canada
{avm, leizhang}@ryerson.ca

2020 IEEE/ACM 42nd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)

Is Your Quantum Program Bug-Free?

Andriy Miranskyy
Ryerson University
Department of Computer Science
Toronto, Canada
avm@ryerson.ca

Lei Zhang
Ryerson University
Department of Computer Science
Toronto, Canada
leizhang@ryerson.ca

Javad Doliskani
Ryerson University
Department of Computer Science
Toronto, Canada
javad.doliskani@ryerson.ca

simula

Measurements and Assertions

- **Output Check:** Similar to classical, e.g., observed and expected outputs are compared [2-4]
- **Statistical Assertion:** Observed and expected distributions are compared [2-4, 8,11]
- **Dynamic Assertions:** With ancilla qubits collect information during program execution about qubits for asserting [12]
- **Projection-based:** Projective measurements to reduce the number of read operations [10]

simula

Coverage criteria

- **Quito:** Coverage of inputs and outputs of quantum programs [3-4]
  - ✓Input coverage
  - ✓Output coverage
  - ✓Input-output coverage
- **QSharpTester:** Equivalent class partition of quantum variables [5]

simula

Techniques (1/3)

- **Metamorphic testing**
  - ✓Metamorphic testing of oracle quantum programs [19]
  - ✓MorphQ: Testing quantum computing platforms (ICSE 2023, presentation later today) [6]
- **Property-based testing:** QSharpCheck framework for Q# [8]
- **Fuzz testing:** QuanFuzz framework for quantum programs [9]

simula

Techniques (2/3)

- **Search-based testing**
  - ✓**QuSBT:** Maximizing the number of failing test cases in a test suite with a genetic algorithm [13-14]
  - ✓**MutTG:** Finding a minimum number of test cases to kill the maximum number of mutants with NSGA-II [15]
- **Combinatorial testing** for quantum programs [21]

simula

Techniques (3/3)

- **Quantum mutation analysis**
  - ✓*Muskit:* Mutation generation framework for Qiskit [2]
  - ✓*QMutPy:* Generates realistic mutants by following real bug patterns [1]
- **Quantum platform testing**
  - ✓*QDiff:* Differential testing of quantum software stacks [7]
  - ✓*MorphQ:* Metamorphic testing of quantum computing platforms [6]

simula

Bug repositories and benchmarks

- Bug respository for quantum computing platforms [16]
- Bug repositories for quantum programs (Bugs4Q and Qbugs) [17]
- A multi-lingual benchmark for property-based testing of quantum programs [20]

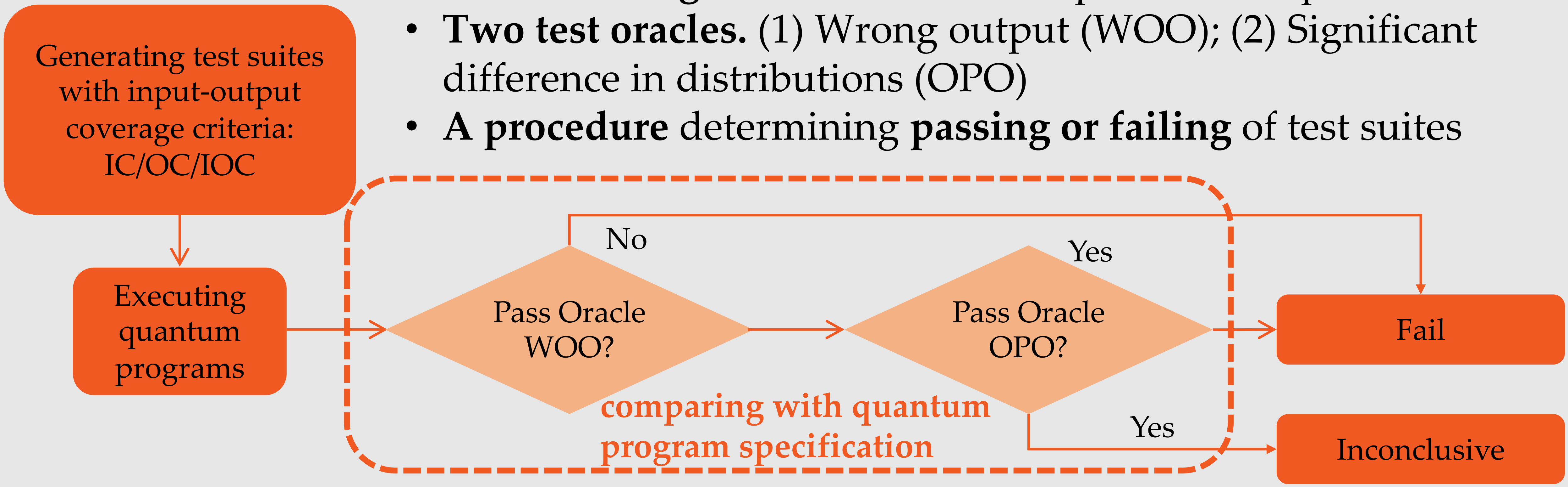
Quantum Software Testing Techniques

simula

Work 1

Quito (QUantum InpuT Output coverage)

- **Three coverage criteria** based on inputs and outputs
- **Two test oracles.** (1) Wrong output (WOO); (2) Significant difference in distributions (OPO)
- **A procedure** determining **passing or failing** of test suites



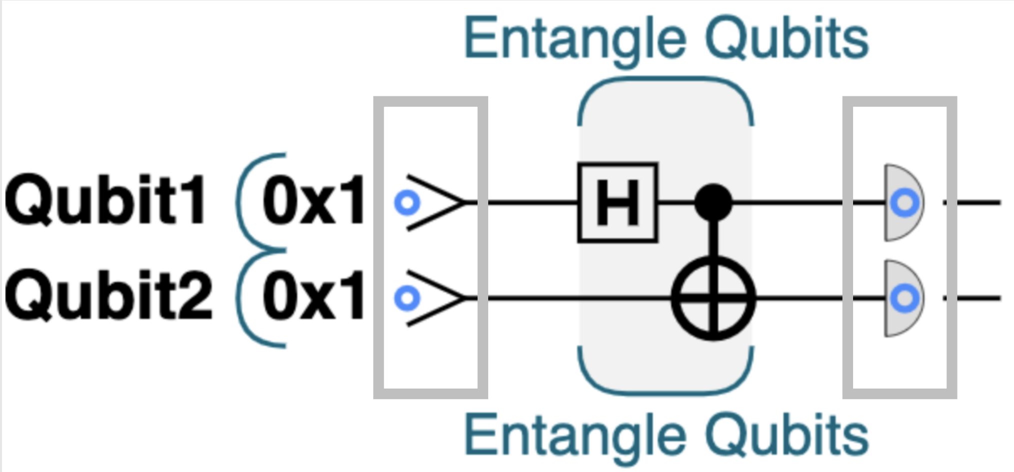
Assessment: Mutation Analysis

Assessing the Effectiveness of Input and Output Coverage Criteria for Testing Quantum Programs
[ICST 2021] [3]

simula

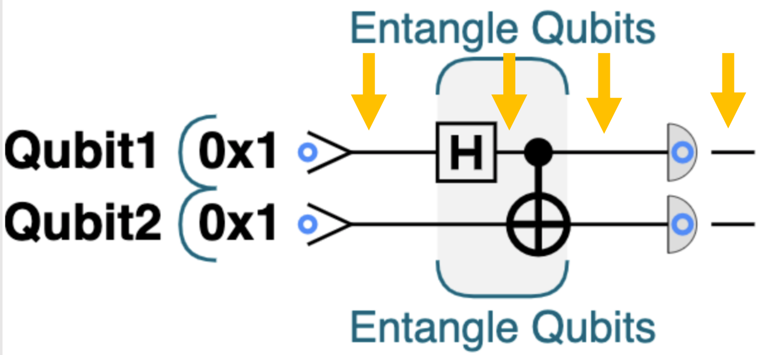
Inputs and Outputs of a Quantum Program (QP)

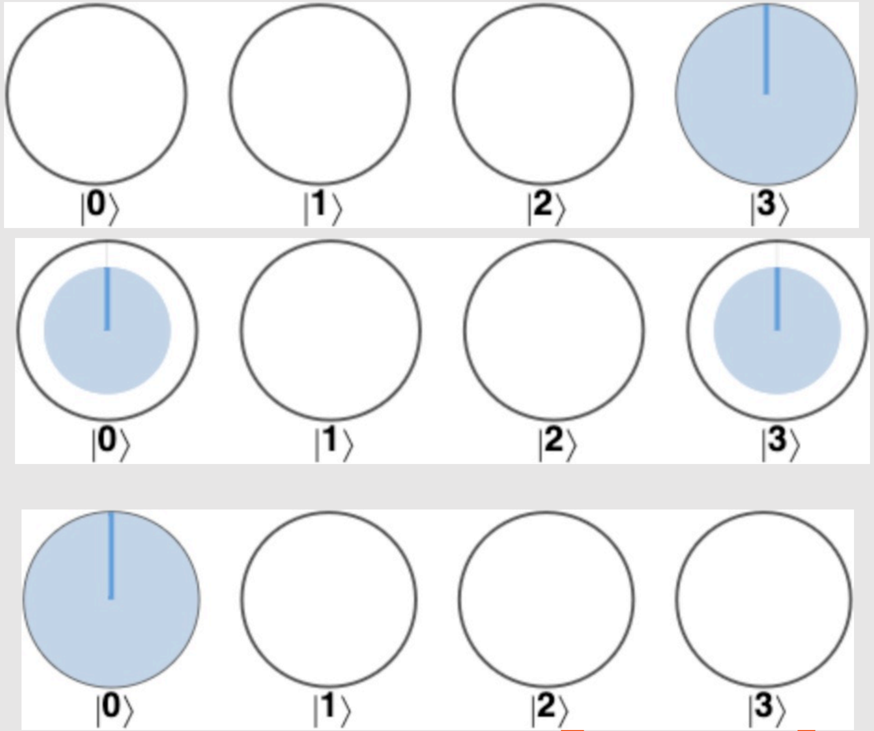
- **Inputs**
  - Values of qubits after QP initialization
- **Outputs**
  - Values of qubits obtained after measurement



simula

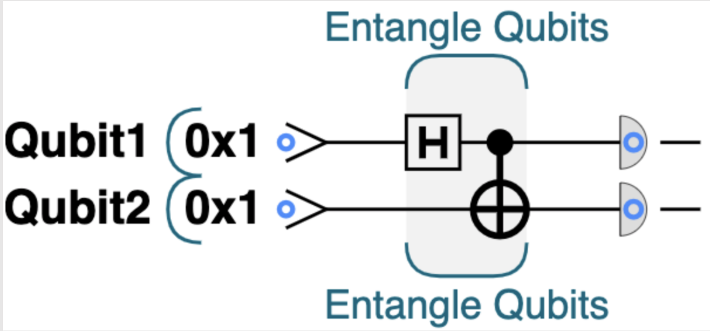
```
qc.reset(2);
var a = qint.new(1, 'a');
var b = qint.new(1, 'b');
qc.reset(2);
qc.write(0);          // Initialize with 0
qc.nop();
qc.label('entangle');
a.had();              // Hadamard Gate. Place into superposition
b.cnot(a);            // Control-NOT Gate. Entangle
qc.label();
qc.nop();
var a_result = a.read();   // The two bits will be random,
var b_result = b.read();   // but always the same.
qc.print(a_result);
qc.print(b_result);
```





[5] M. Gimeno-Segovia, N. Harrigan, and E. Johnston, Programming Quantum Computers: Essential Algorithms and Code Samples.O' Reilly Media, Incorporated, 2019. [Online].

Program Specification (PS) of a QP



- **Valid Inputs**
  - Input values that are valid according to PS
- **Valid Outputs Values**
  - Output values that can be produced with at least one valid input
- **Probabilities**
  - Given a valid input, expected probabilities of occurrence of all the valid output values

Valid Input	Valid Output 1	Probability 1	Valid Output 2	Probability 2
00	00	50%	11	50%
01	00	50%	11	50%

Quito: A Framework for Quantum Program Testing

3 Coverage Criteria

- Input Coverage
- Output Coverage
- Input-Output Coverage

2 Test Oracles

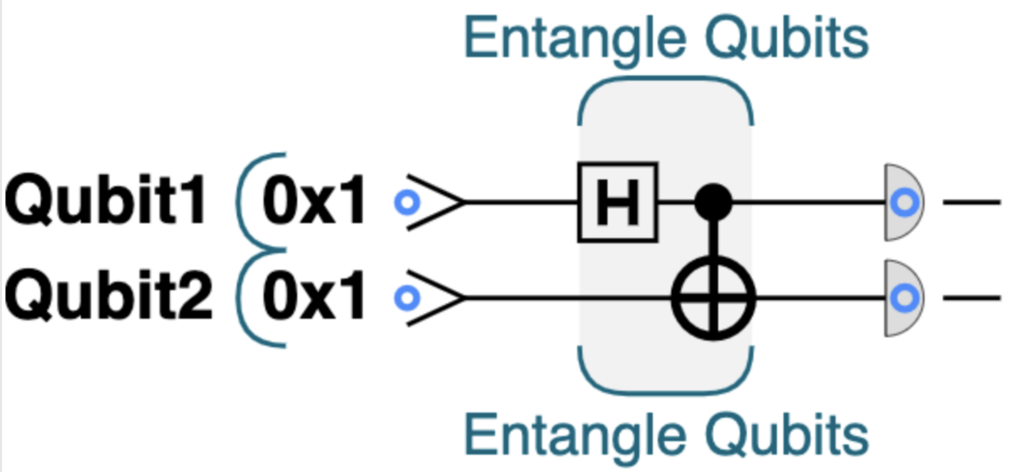
- Wrong Output Oracle
- Output Probability Oracle

Assessment

- Mutation Operators
- Mutation Analysis

simula

Input Coverage (IC)



- In one test suite, there exists a test for each valid input
- A statically generated test suite can achieve IC

One Possible Test Suite

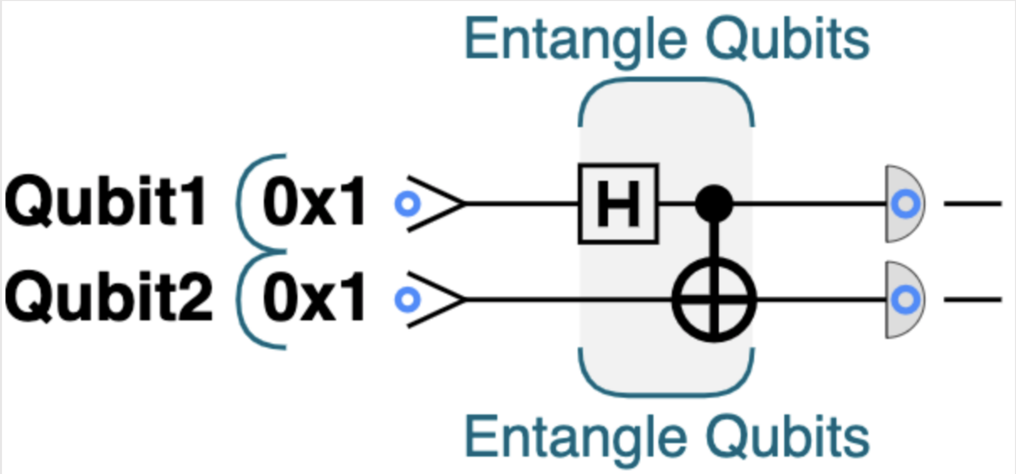
Input	Output
00	0
01	0

Program Specification for Entanglement

Valid Input	Valid Output 1	Probability 1	Valid Output 2	Probability 2
00	0	50%	11	50%
01	0	50%	11	50%

mula

Output Coverage (OC)



- In one test suite, there exists a test for each valid output.
- The criterion cannot be achieved statically.

One Possible Test Suite

Input	Output
00	00
01	00
00	11

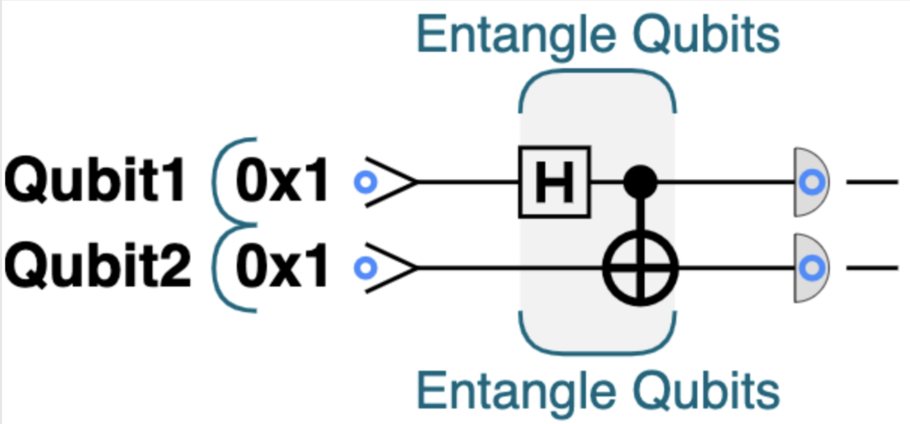
Program Specification for Entanglement

Valid Input	Valid Output 1	Probability 1	Valid Output 2	Probability 2
00	00	50%	11	50%
01	00	50%	11	50%

simula

Input-Output Coverage (IOC)

- In one test suite, there exists a test for each input-output pair.
- The criterion cannot be achieved statically.



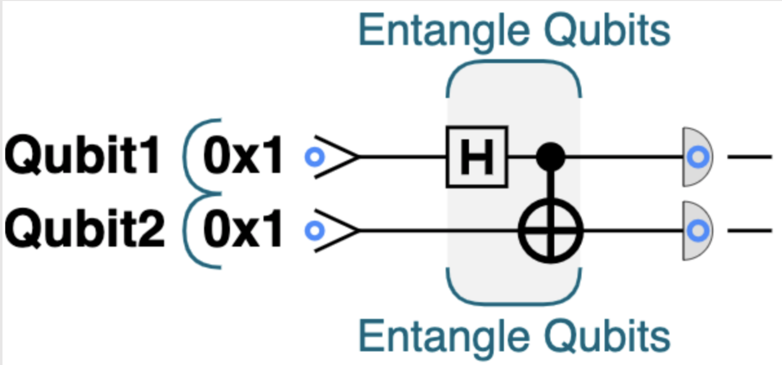
One Possible Test Suite

Input	Output
00	00
00	00
00	11
01	00
01	11

Program Specification for Entanglement

Valid Input	Valid Output 1	Probability 1	Valid Output 2	Probability 2
00	00	50%	11	50%
01	00	50%	11	50%

Test Oracle – Wrong Output Oracle (WOO)



WOO checks if the test outcome returned for a test input is invalid, which reveals a *definitely fail:* wrong outputs.

Valid Input	Valid Output 1	Probability 1	Valid Output 2	Probability 2
00	00	50%	11	50%
01	00	50%	11	50%

Input	Output
00	01

simula

Test Oracle – Output Probability Oracle (OPO)

- OPO checks if a QP returns an expected output with the expected probability.
  - **Likely Fail:** With a given confidence, multiple executions of a test show that the outputs do not occur with the expected probabilities.
  - **Inconclusive:** Multiple executions of the test do not allow to reject the null hypothesis of a statistical test.

simula

Key findings

- A **less expensive** coverage criterion (e.g., IC) may achieve higher mutation scores.
- An **expensive** coverage criterion (i.e., IOC) may increase mutation scores.
- If the fault in a program results in a **wrong output (WOO)**, it can possibly be caught with **a lower number** of test cases.
- If certain faults cannot be found with WOO, the cost of finding faults with **OPO** could be **quite higher**. However, it may be reduced with a proper budget upper limit.

simula

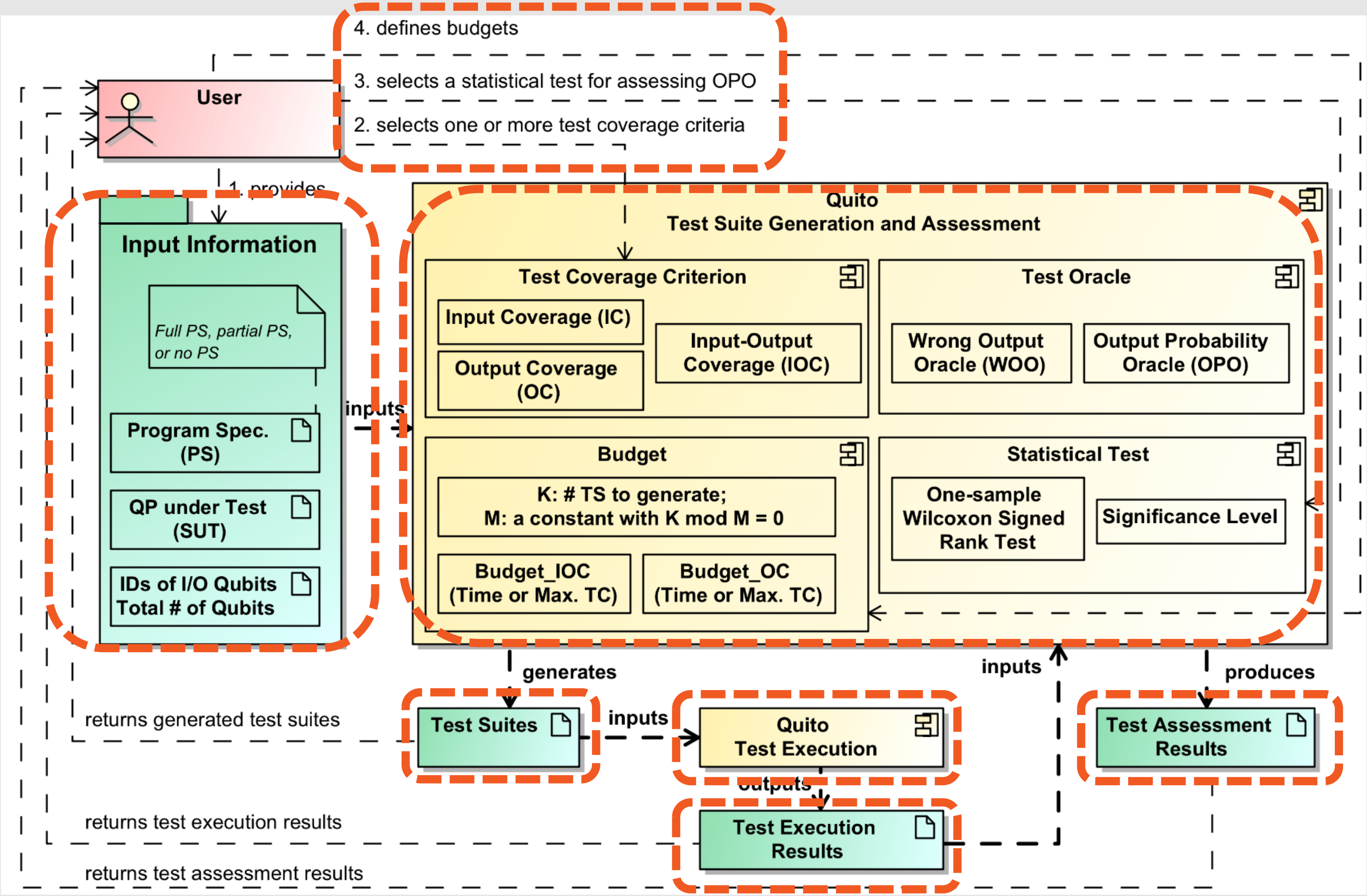
Limitations

- Quito suffer from scalability issue with an increased number of qubits.
  - 2023: Deployed Quito on an HPC platform
- Quito does not deal with phases of qubits.
  - Work in progress
- Quito's mutation analysis can be further improved.
  - 2023: Several tools are available such as Muskit, QMutPy, etc

simula

Work 1

Quito (QUantum InpuT Output coverage)



Quito: a Coverage-Guided Test Generator for Quantum Programs
[ASE Demo 2021] [4]

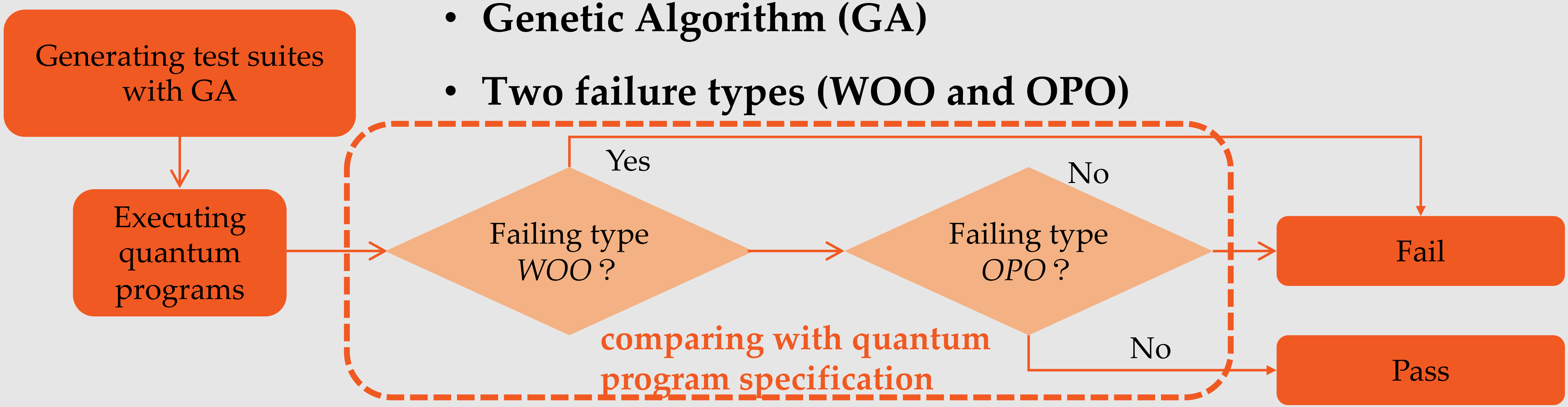
simula

Work 2

QuSBT (Quantum Search-Based Testing)

Goal: Generating a test suite with the maximum possible number of failing test cases

- **Genetic Algorithm (GA)**
- **Two failure types (WOO and OPO)**



Baseline: Random Search

Generating Failing Test Suites for Quantum Programs with Search
[SSBSE 2021] [13]

QuSBT: Quantum Search-based Testing

- **Test case generation** with a Genetic Algorithm (GA)
- **2 types of failures**
  - ✓Unexpected Output Failure
  - ✓Wrong Output Distribution Failure

simula

Test Case Generation

- Generating M search variables $x_1, \dots, x_M$, each representing one input
- Let D_I be the domain of possible valid inputs,

$$M = \lceil \beta \times |D_I| \rceil$$

- Let $ta = [\text{fail}_1, \dots, \text{fail}_M]$ be the assessments of M tests,
- Fitness function:

$$\max:\ f = \left|\left\{\text{fail}_j \in ta \middle| \text{fail}_i = true\right\}\right|$$

simula

Experiment Design

- **Frameworks:** Qiskit 0.23.2, jMetalPy 1.5.5
- **Baseline:** Random Search (RS)
- Six **benchmark** programs (e.g., quantum cryptography)
- **Faulty versions:** 30
- **Parameters:** β as 5%; 30 repetitions
- **GA:** Population size 10; termination criterion is max generation 50

simula

Experiment Design

- **Research Questions:**
  - RQ1: Does QuSBT perform better than RS?
  - RQ2: How does QuSBT perform on the benchmark programs?
- **Evaluation Metric: Number of failed tests (NFT)**
  - The best final solution
  - The best solution of each generation
- **Statistical tests**
  - The Mann-Whitney U test as the statistical test
  - The Vargha and Delaney's *A*12 statistics

simula

RQ1: Does QuSBT perform better than RS?

Comparison between GA and RS

Program	✓	≡
AS	4	1
BV	5	0
CE	5	0
IQ	4	1
QR	5	0
SM	3	2

No significant difference between GA and RS

GA is significantly better than RS

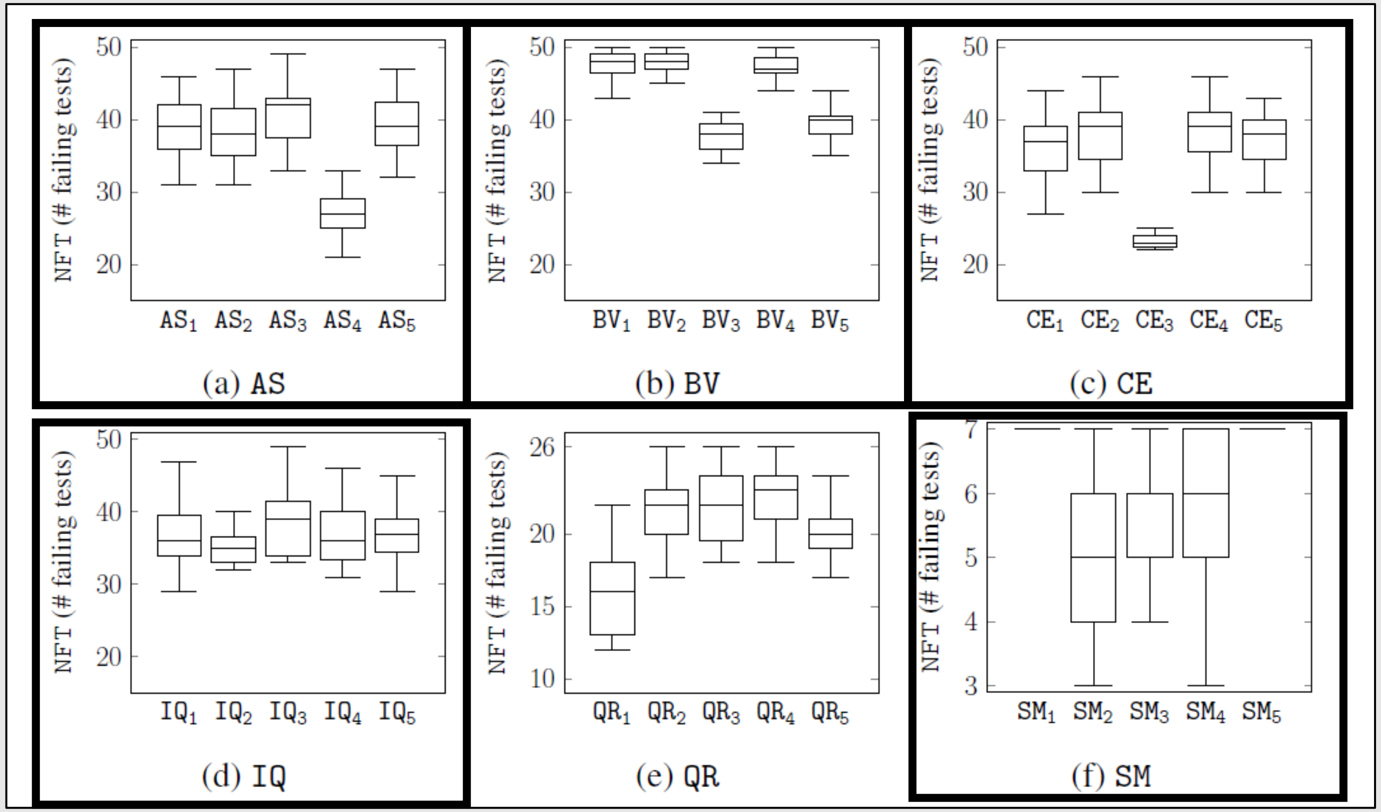
GA outperforms RS for 87% of the faulty quantum programs.

For BV, CE and QR, GA consistently performs better than RS.

simula

RQ2: How does QuSBT perform on the benchmark programs?

NFT of GA across 30 runs



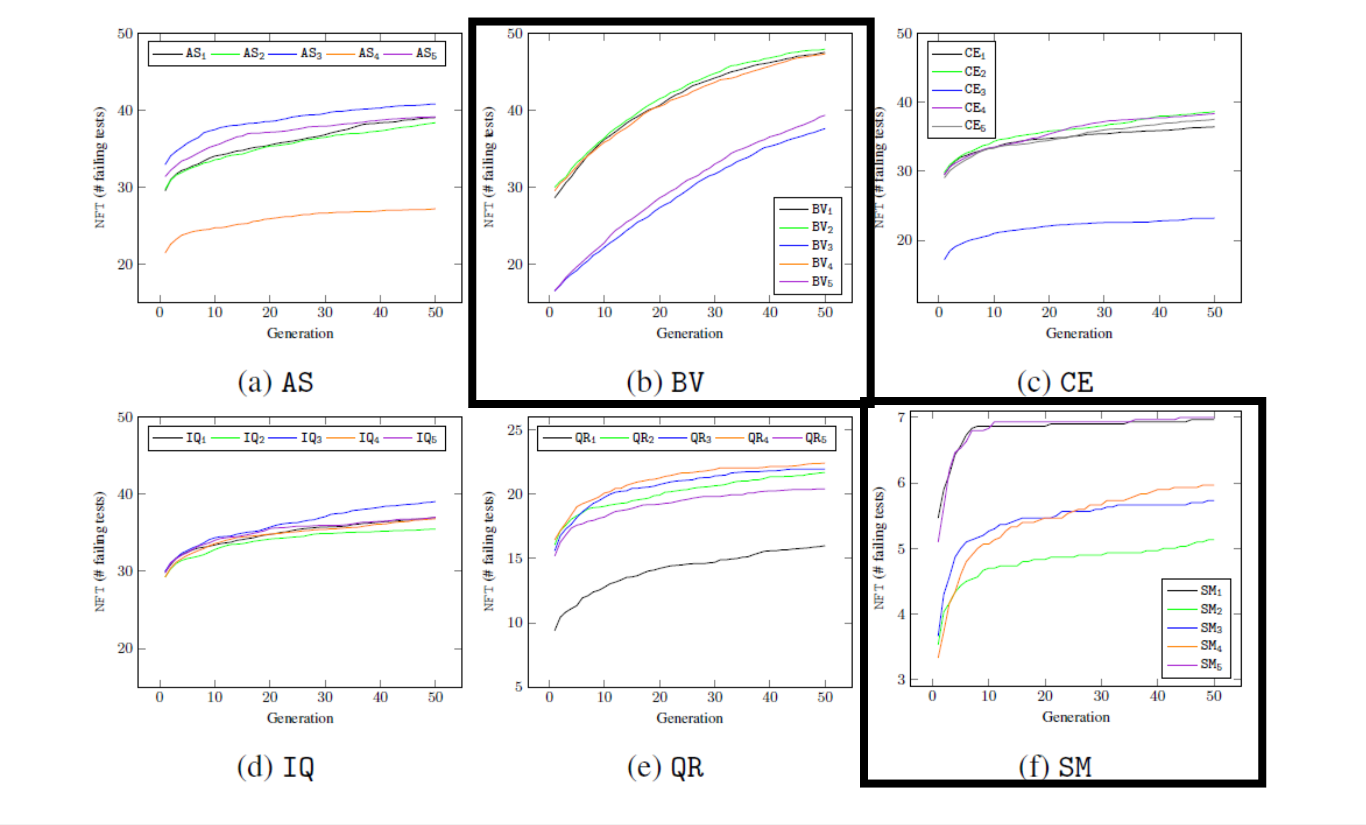
In four groups of the most complex benchmarks, the variability is usually high, but it can still find the maximum number of failing inputs in some cases.

For the small program SM, the search can always find the maximum failing inputs of the test suite for two mutants.

simula

RQ2: How does QuSBT perform on the benchmark programs?

Evolution of the fitness values over generations



(Values are averages across 30 runs of the fitness of the best individual in the population)

The first generations already find some failing tests, the values keep increasing across generations.

3 benchmarks of BV and 2 benchmarks of SM almost find all failing tests.

The numbers failing tests vary across benchmark programs, depending on the types and locations of faults.

simula

Conclusions

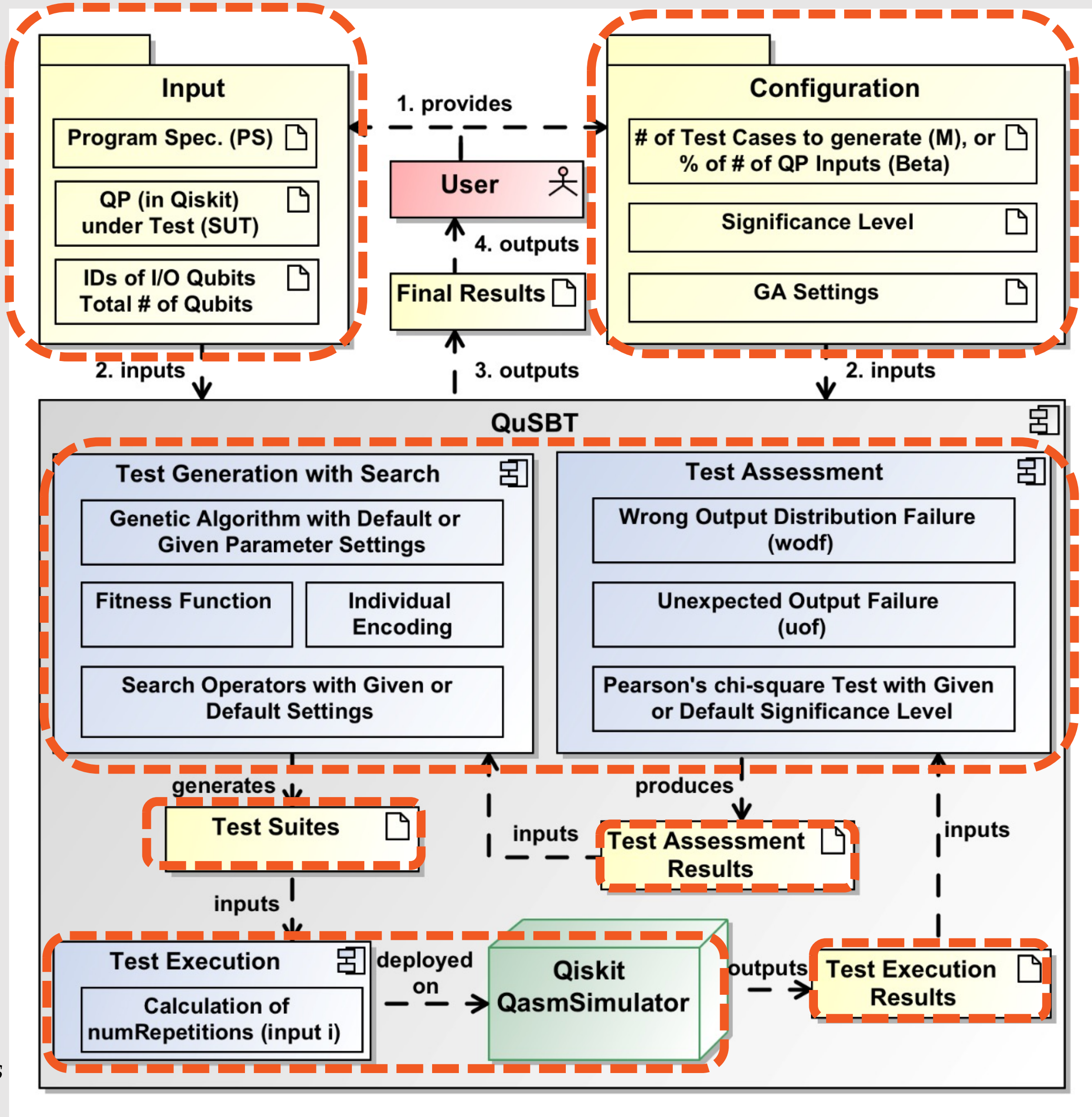
- QuSBT is a **search-based approach** for testing quantum programs with **Genetic Algorithm**, aiming at finding **as many failing tests** as possible.

- QuSBT was assessed with **30 faulty quantum programs**. **QuSBT** outperformed **Random Search** in **87%** of the programs.

simula

Work 2

QuSBT (Quantum Search-Based Testing)



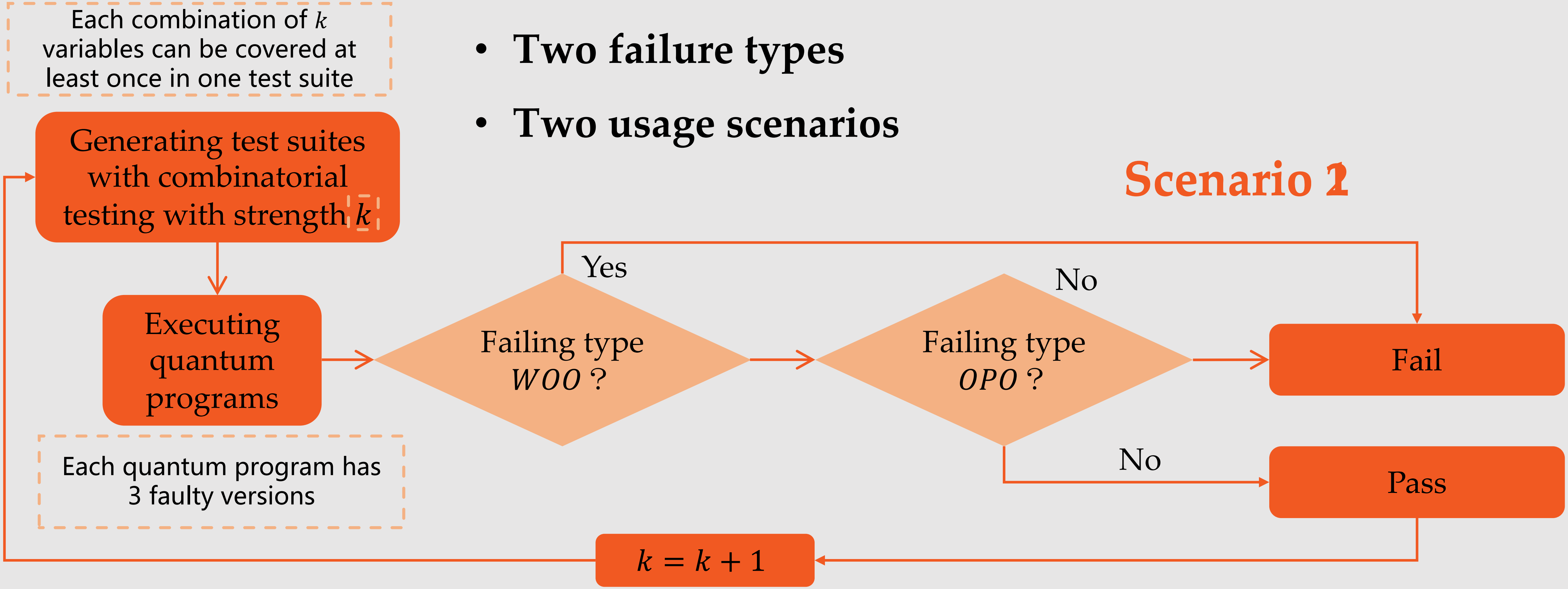
QuSBT: Search-Based Testing of Quantum Programs
[ICSE Demo 2021] [14]

simula

Work 3

QuCAT (QUantum CombinAtorial Testing)

- **Combinatorial testing**
- **Two failure types**
- **Two usage scenarios**



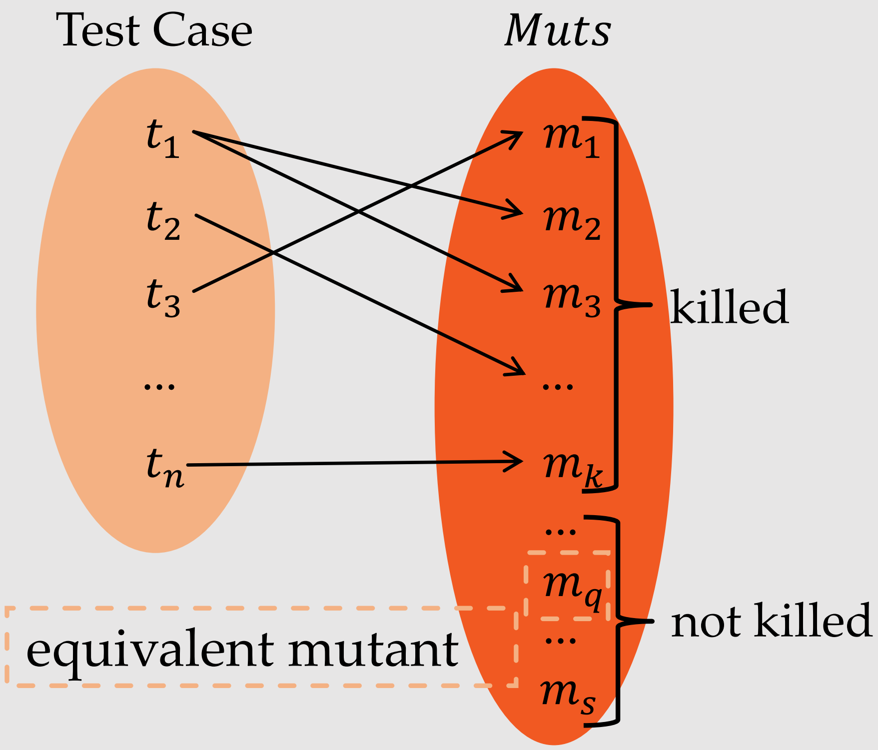
Baseline: Random Testing

simula

Application of Combinatorial Testing to Quantum Programs
[QRS 2021] [21]

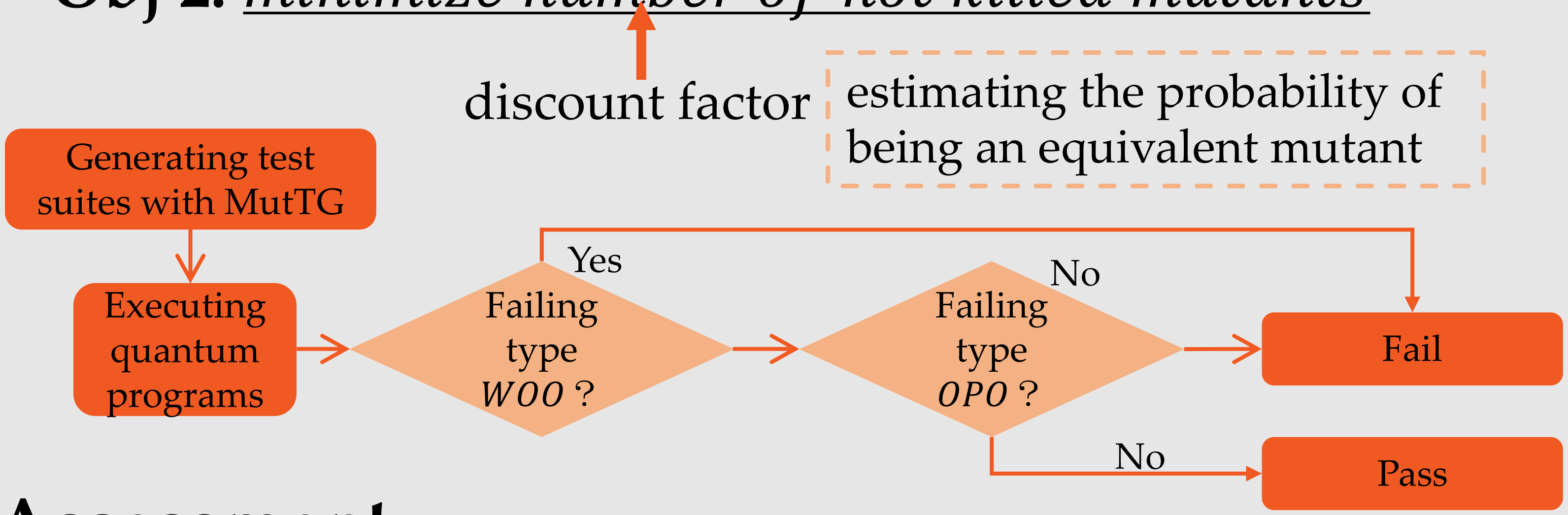
Work 4

MutTG (multi-objective search-based approach)



Obj 1: *minimize test suite size*

Obj 2: *minimize number of not killed mutants*



Assessment:

- **Baseline:** (1) random search, (2) approach without discount factor
- **Benchmarks:** *Mutants* with different difficulty levels

Mutation-based test generation for quantum programs with multi-objective search. [GECCO '22][15]

simula

Work 5

Noise-Aware Quantum Software Testing (Ongoing)

- **Problem:** Hardware Noise
  - ✓Environmental characteristics, e.g., magnetic fields, radiations, interactions of qubits with environments
  - ✓Unwanted interactions of qubits exist among themselves (crosstalk noise)
  - ✓Imprecise quantum gate calibrations

Challenges:

- Due to noise, a program can produce wrong output states or correct output states with wrong probabilities
- Does quantum program really failed or is it due to noise?

	Probability							
State	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
Ideal	0.495	-	-	-	-	-	-	0.505
Noisy	0.476	0.013	0.007	0.016	0.008	0.019	0.020	0.443

simula

Work 6

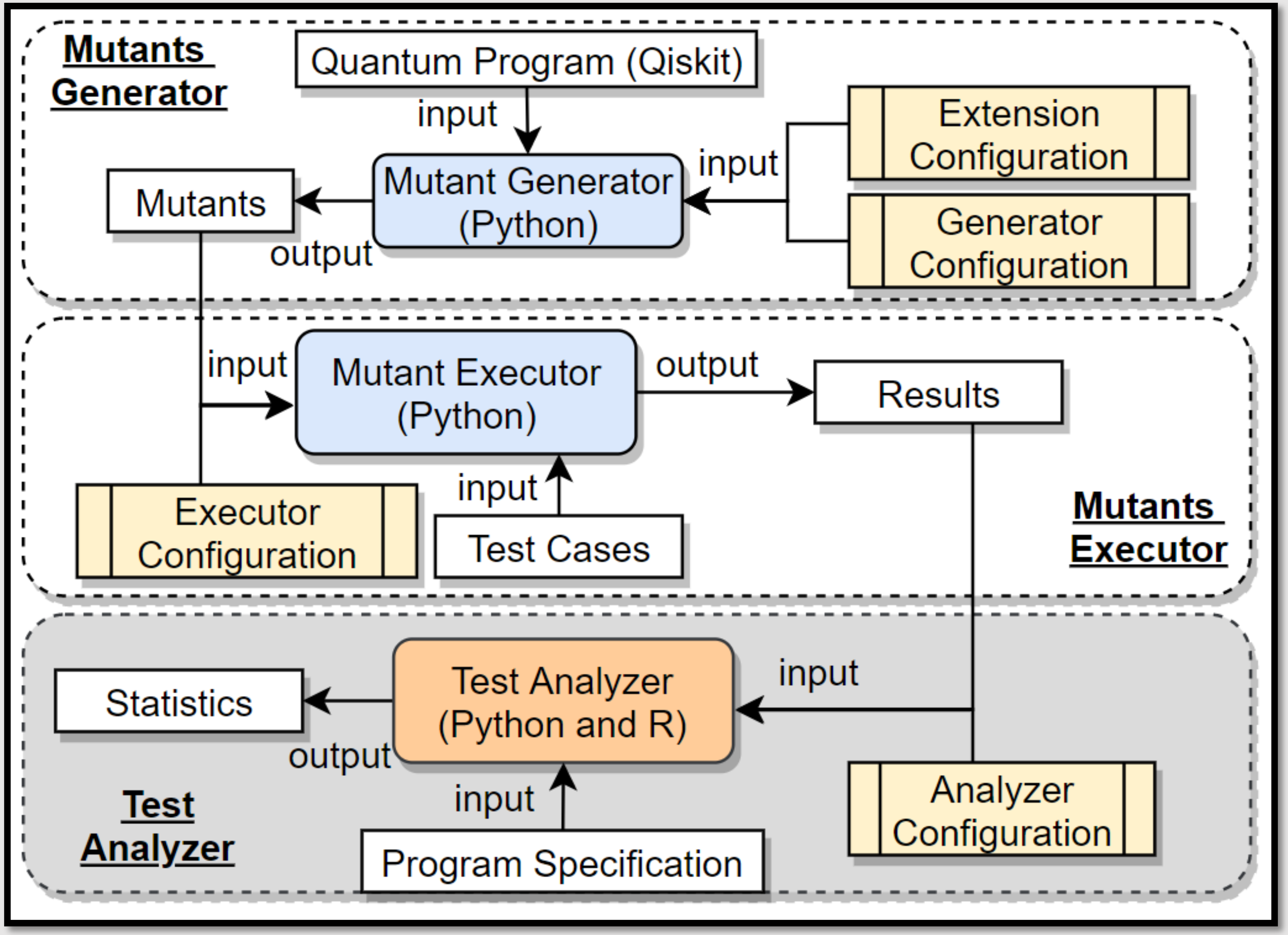
Muskit: A Mutation Analysis Tool for Quantum Software Testing

- **Problem:** Lack of bug repositories and benchmarks to assess quality of test cases generated for testing quantum programs
- **Solution:** Mutation analysis tool for quantum programs in IBM's Qiskit
- **Features**
  - ✓Mutation Operator Types: Add, Remove, Replace gates
  - ✓Mutation Selection Criteria: All, Gate selection (one qubit, two-qubit, etc), …

simula

Work 6

Muskit: A Mutation Analysis Tool for Quantum Software Testing



simula

Tools, datasets, and publications

MutTG **Quito** **QuSBT** **QuCAT** **Muskit**

Ali, Shaukat & Yue, Tao & Abreu, Rui. (2022). **When software engineering meets quantum computing**, Communications of the ACM. 65. 84-88. 10.1145/3512340.

S. Ali, P. Arcaini, X. Wang, and T. Yue, **Assessing the effectiveness of input and output coverage criteria for testing quantum programs**, the 14th IEEE Conference on Software Testing, Verification and Validation (ICST), 2021, pp. 13–23.

X. Wang, P. Arcaini, T. Yue, and S. Ali, **Quito: a Coverage-Guided Test Generator for Quantum Programs,** the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE) Tool Track, IEEE, 2021.

X. Wang, P. Arcaini, T. Yue, and S. Ali, **Generating failing test suites for quantum programs with search**, in Search-Based Software Engineering, U.-M. O'Reilly and X. Devroey, Eds. Cham: Springer International Publishing, 2021, pp. 9–25.

X. Wang, P. Arcaini, T. Yue, and S. Ali, **QuSBT: Search-Based Testing of Quantum Programs**, the 44th International Conference on Software Engineering (ICSE) Tool Track. ACM, 2022.

X. Wang, P. Arcaini, T. Yue, and S. Ali, **Application of combinatorial testing to quantum programs,** the 21st International Conference on Software Quality, Reliability and Security (QRS). IEEE, 2021.

E. Mendiluze, S. Ali, P. Arcaini and T. Yue, **Muskit: A Mutation Analysis Tool for Quantum Software Testing**, the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE) Tool Track, 2021, pp. 1266-1270

Acknowledgements

- Paolo Arcaini
- Students
  - Xinyi Wang
  - Eñaut Mendiluze
  - Asmar Muqeet
  - Tongxuan Yu

simula

References

[1] Daniel Fortunato, José Campos, and Rui Abreu. 2022. QMutPy: A Mutation Testing Tool for Quantum Algorithms and Applications in Qiskit. In Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis (Virtual, South Korea) (ISSTA 2022). Association for Computing Machinery, New York, NY, USA, 797–800. https://doi.org/10.1145/3533767.3543296

[2] E. Mendiluze, S. Ali, P. Arcaini and T. Yue, "Muskit: A Mutation Analysis Tool for Quantum Software Testing," 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 2021, pp. 1266-1270, doi: 10.1109/ASE51524.2021.9678563.

[3] S. Ali, P. Arcaini, X. Wang and T. Yue, "Assessing the Effectiveness of Input and Output Coverage Criteria for Testing Quantum Programs," 2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST), Porto de Galinhas, Brazil, 2021, pp. 13-23, doi: 10.1109/ICST49551.2021.00014.

[4] X. Wang, P. Arcaini, T. Yue and S. Ali, "Quito: a Coverage-Guided Test Generator for Quantum Programs," 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 2021, pp. 1237-1241, doi: 10.1109/ASE51524.2021.9678798.

[5] QSharpTester: Peixun Long and Jianjun Zhao. 2022. Testing Quantum Programs with Multiple Subroutines. 1 (2022), 1–14. http://arxiv.org/abs/2208.09206

[6] Matteo Paltenghi, Michael Pradel, MorphQ: Metamorphic Testing of the Qiskit Quantum Computing Platform, 2023 International Conference on Software Engineering

[7] J. Wang, Q. Zhang, G. H. Xu and M. Kim, "QDiff: Differential Testing of Quantum Software Stacks," 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 2021, pp. 692-704, doi: 10.1109/ASE51524.2021.9678792.

[8] Shahin Honarvar, Mohammad Reza Mousavi, and Rajagopal Nagarajan. 2020. Property-based Testing of Quantum Programs in Q#. In Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops (ICSEW'20). Association for Computing Machinery, New York, NY, USA, 430–435. https://doi.org/10.1145/3387940.3391459

[9] J. Wang, F. Ma and Y. Jiang, "Poster: Fuzz Testing of Quantum Program," 2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST), Porto de Galinhas, Brazil, 2021, pp. 466-469, doi: 10.1109/ICST49551.2021.00061.

[10] Gushu Li, Li Zhou, Nengkun Yu, Yufei Ding, Mingsheng Ying, and Yuan Xie. 2020. Projection-based runtime assertions for testing and debugging Quantum programs. Proc. ACM Program. Lang. 4, OOPSLA, Article 150 (November 2020), 29 pages. https://doi.org/10.1145/3428218

[11] Yipeng Huang and Margaret Martonosi. Statistical assertions for validating patterns and finding bugs in quantum programs. In Proceedings of the 46th International Symposium on Computer Architecture, pages 541–553. ACM, 2019.

[12] Ji Liu, Gregory T Byrd, and Huiyang Zhou. Quantum circuits for dynamic runtime assertions in quantum computation. In Proceedings of the Twenty-Fifth International Conference on Architectural Supportfor Programming Languages and Operating Systems, pages 1017–1030, 2020

[13] Wang, X., Arcaini, P., Yue, T., Ali, S. (2021). Generating Failing Test Suites for Quantum Programs With Search. In: O'Reilly, UM., Devroey, X. (eds) Search-Based Software Engineering. SSBSE 2021. Lecture Notes in Computer Science(), vol 12914. Springer, Cham. https://doi.org/10.1007/978-3-030-88106-1_2

[14] Xinyi Wang, Paolo Arcaini, Tao Yue, and Shaukat Ali. 2022. QuSBT: search-based testing of quantum programs. In Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings (ICSE '22). Association for Computing Machinery, New York, NY, USA, 173–177. https://doi.org/10.1145/3510454.3516839

[15] Xinyi Wang, Tongxuan Yu, Paolo Arcaini, Tao Yue, and Shaukat Ali. 2022. Mutation-based test generation for quantum programs with multi-objective search. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '22). Association for Computing Machinery, New York, NY, USA, 1345–1353. https://doi.org/10.1145/3512290.3528869

[16] Matteo Paltenghi and Michael Pradel. 2022. Bugs in Quantum computing platforms: an empirical study. Proc. ACM Program. Lang. 6, OOPSLA1, Article 86 (April 2022), 27 pages. https://doi.org/10.1145/3527330

[17] P. Zhao, J. Zhao, Z. Miao and S. Lan, "Bugs4Q: A Benchmark of Real Bugs for Quantum Programs," in 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), Melbourne, Australia, 2021 pp. 1373-1376.doi: 10.1109/ASE51524.2021.9678908

[18] J. Campos and A. Souto, "QBugs: A Collection of Reproducible Bugs in Quantum Algorithms and a Supporting Infrastructure to Enable Controlled Quantum Software Testing and Debugging Experiments," in 2021 IEEE/ACM 2nd International Workshop on Quantum Software Engineering (Q-SE), Madrid, Spain, 2021 pp. 28-32. doi: 10.1109/Q-SE52541.2021.00013

[19] R. Abreu, J. P. Fernandes, L. Llana and G. Tavares, "Metamorphic Testing of Oracle Quantum Programs," 2022 IEEE/ACM 3rd International Workshop on Quantum Software Engineering (Q-SE), Pittsburgh, PA, USA, 2022, pp. 16-23, doi: 10.1145/3528230.3529189.

[20] G. Pontolillo and M. R. Mousavi, "A Multi-Lingual Benchmark for Property-Based Testing of Quantum Programs," 2022 IEEE/ACM 3rd International Workshop on Quantum Software Engineering (Q-SE), Pittsburgh, PA, USA, 2022, pp. 1-7, doi: 10.1145/3528230.3528395.

[21] X. Wang, P. Arcaini, T. Yue and S. Ali, "Application of Combinatorial Testing to Quantum Programs," 2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS), Hainan, China, 2021, pp. 179-188, doi: 10.1109/QRS54544.2021.00029.

simula

Quantum Software Testing
A Brief Introduction

Shaukat Ali[1, 2] and Tao Yue[1]

[1]Simula Research Laboratory, Oslo, Norway

[2]Oslo Metropolitan University, Oslo, Norway

simula