量子程序中的不稳定性识别

Lei Zhang MahsaRadnejad信息系统统系 计算机工程系 马里兰大学巴尔的摩县分校 伊斯法罕分校，伊斯兰阿萨德大学 美国巴尔的摩 伊朗伊斯法罕
leizhang@umbc.edu radnejad@khuisf.ac.ir

Andriy Miranskyy
*De*计算机科学系
*T*多伦多都会大学 多伦多，加拿大 avm@torontomu.ca

arXiv:2302.03256v2 [cs.SE] 7 Jul 2023

摘要——近年来，软件工程师一直在探索协助量子软件程序员的方法。本文旨在继续这一探索，并考察量子软件程序员是否也会遇到困扰经典程序的一些问题。具体而言，我们研究间歇性失败测试，即不稳定测试，是否会影响量子软件开发。

为探究不稳定性，我们对 14 个量子软件代码库进行了初步分析。随后，我们识别出不稳定测试，并对其原因及修复方法进行分类。

我们在 14 个量子软件代码库中的 12 个里发现了不稳定测试。在这 12 个代码库中，与不稳定测试相关的问题所占百分比的下限为每个代码库 0.26% 至 1.85%。我们识别出 46 份不同的不稳定测试报告，归纳出 8 组原因和 7 种常见解决方案。此外，我们注意到，量子程序员没有采用软件工程师近期提出的某些不稳定测试应对措施。

这项工作可能会引起从业者的兴趣，因为它为量子程序中不稳定测试的解决提供了有用的见解。研究人员也可能会发现本文颇有帮助，因为它提供了有关量子软件中不稳定测试的定量数据，并指出了新的研究机会。

I. 引言

在相同代码上运行的测试有时会产生不同的结果，即有时显示“通过”，有时显示“失败”。这类测试称为不稳定测试。不稳定测试会提供误导性信号，从而对开发人员产生负面影响。可以将不稳定测试视为测试缺陷，因为它们会产生非确定性结果。这类测试会消耗大量资源。在谷歌，2014年，160 万次测试失败中有 7.3 万次（4.56%）是由不稳定测试导致的 [1]；2017年，420 万次测试中有 1.5% 是不稳定测试 [2], [3]。

不稳定测试可能由两个因素导致：源代码本身的不确定性，或测试本身的不确定性 [1], [4]–[8]。量子程序本质上具有不确定性。随机性来源多种多样。例如，它可能由量子系统的物理性质（如量子不确定性 [9, 第 1] 章）或硬件问题（如测量错误或网络问题）导致，也可能由两者共同导致（如量子退相干 [10, 第 7] 章）。在经典计算机（CC）上模拟量子计算机（QC）时，会使用伪随机数生成器（PRNG）来模拟这些随机性来源。所有这些来源产生的随机性会导致输出值呈现一定分布，进而可能造成测试不稳定。

据我们所知，目前尚无关于量子程序中不稳定测试的量子软件工程研究。因此，我们旨在初步分析量子程序中的不稳定性及其根本原因。为此，我们将尝试回答以下三个研究问题（RQs）。

研究问题1：量子程序中的不稳定测试有多普遍？研究问题2：量子软件中的不稳定性由什么导致？研究问题3：量子程序员如何修复不稳定测试？

本文有两项主要贡献。首先，我们对量子程序中的不稳定测试进行了初步研究。具体而言，我们调查了 14 个量子软件的代码和缺陷跟踪仓库；其中 12 个软件至少有一个不稳定测试（总计 46 份独特的不稳定测试报告）。我们估计，这 12 个软件中至少有 0.26% 至 1.85% 的问题报告与不稳定测试有关。其次，我们识别并归纳了量子软件不稳定性的八类原因，以及七种常见的修复方法。我们发现，不稳定性最常见的原因是随机性，而最常见的解决方案是为伪随机数生成器固定种子。此外，量子程序员没有采用软件工程师为应对不稳定测试而开发的一些最新应对措施（例如， [7], [8]）。我们公开了包含量子不稳定测试、修复方法及其类别的数据集，网址为 https://doi.org/10.5281/zenodo.7888639。

本文组织如下。第II节介绍我们对现有量子程序不稳定性开展的初步实证研究。第III节通过分析不稳定测试报告，寻求回答我们的三个研究问题。第IV节讨论效度威胁。第V节介绍量子程序测试以及经典程序中不稳定测试的相关工作。第VI节总结全文，并概述我们长期的研究目标。

II. 实证研究：数据收集

为回答我们的研究问题，我们开展了一项实证研究，重点关注 GitHub 上的开源量子程序，因为开源项目具有透明的开发历史和缺陷报告。我们通过三个步骤，从开源量子项目中收集不稳定测试报告。

首先，我们选择了四个热门量子平台（即 IBM Qiskit [11], Microsoft Quantum Development Kit [12], TensorFlow Quantum [13], 和 NetKet [14]），并确定了 14 个具有活跃贡献和缺陷报告的代码仓库。尽管这并非所有量子软件的穷尽列表，但这些平台代表了一些最活跃、最受欢迎的开源量子生态系统。

其次，我们针对 14 个 GitHub 代码仓库搜索了已关闭的 GitHub issue，使用了 10 个关键词（即“flaky”、

表I：包含不稳定测试的量子软件代码库统计信息。最右侧三列分别为：已关闭问题报告的数量（第 T 列）、已关闭不稳定测试报告的数量（第 F 列），以及与不稳定测试相关的报告百分比（第 P 列），计算结果为 $F/T \times 100\%$。

平台	存储库	语言	T	F	P
Qiskit	qiskit-terra	Python	2,810	25	0.89%
Qiskit	qiskit-aer	Python	558	3	0.54%
Qiskit	qiskit-nature	Python	287	1	0.35%
Qiskit	qiskit-experiments	Python	255	1	0.39%
Qiskit	qiskit-ibm-runtime	Python	217	2	0.92%
Qiskit	qiskit-ibm-provider	Python	184	2	1.09%
Qiskit	qiskit-machine-learning	Python	378	1	0.26%
微软	qdk-python	Python	64	1	1.56%
微软	QuantumLibraries	Q#	136	2	1.47%
微软	量子	Q#	108	2	1.85%
TensorFlow	量子	Python	192	1	0.52%
NetKet	netket	Python	295	5	1.69%
总计			5,484	46	

“flakiness”、“flakey”、“occasion”、“occasional”、“intermit”、“fragile”、“non-deterministic”、“nondeterministic”和“brittle”）。我们只关注已关闭报告，因为这些报告已由开发人员验证。

最后，至少由两名作者交叉审查这些问题及相关的拉取请求和代码提交，判断它们是否与不稳定性有关，并确定原因类别。如果审查人员之间存在分歧，则会进行联合审查，以达成一致。在本研究中，使用这 10 个关键词进行搜索返回了 253 份问题报告，其中 46 份经人工标注为不稳定测试报告（经过交叉审查）。随后，我们剔除了两个没有经过验证的不稳定测试的代码仓库（即 qiskit-finance 和 qiskit-optimization），最终得到 12 个代码仓库。

根据我们的人工检查，不稳定测试报告的常见情形是：一份问题报告附带一个修复不稳定性的拉取请求。不过，还有另外三种情形：1）多个拉取请求与一份不稳定测试报告相关，例如回移植拉取请求，我们将其视为一份不稳定测试报告；2）没有问题报告、但解决了不稳定性的拉取请求；3）已关闭但没有关联拉取请求的不稳定性问题报告，例如某个不稳定测试在没有详细讨论或相应提交的情况下得到解决。为简单起见，我们将上述四种情形统称为“不稳定测试报告”。

III. 分析与结果

本节将通过分析获得的不稳定测试报告，来回答我们的三个研究问题。

A. 研究问题1：量子程序中的不稳定测试有多普遍？

表I展示了我们检测到的量子程序代码库及不稳定测试报告的统计数据。数据最后更新于2023年1月12日；因此，统计数据今后可能会发生变化。

如表I所示，我们在这12个代码仓库中检测到46份不稳定测试报告。在所有代码仓库中，核心 Qiskit 组件 qiskit-terra 的不稳定性报告最多（即25份），因为该项目规模较大。不稳定性报告数量最多（即25份）的代码仓库是核心 Qiskit 组件 qiskit-terra，这与项目规模有关。不稳定性的平均百分比从 0.26%（qiskit-machine-learning）到1.85%（Microsoft/Quantum）不等。由于我们的关键词列表并不全面，不稳定性百分比表示不稳定测试报告数量的下限（详见第IV节的进一步讨论）。换句话说，实际的不稳定测试可能比我们观察到的更多。

最后，比较量子计算和经典计算机程序中不稳定测试的频率较为困难。如第 I 节所述，谷歌报告的是测试用例和测试失败的百分比，而不是不稳定测试报告的百分比；其他作者通常也报告类似的统计数据（参见 [5] 综述）。因此，无法进行直接比较。我们未来的计划包括将不稳定测试报告映射到测试用例，并开展此类比较。然而，即使无法进行直接比较，我们仍然可以明确回答研究问题1：量子程序中存在数量可观的不稳定测试。

B. 研究问题2：量子软件中的不稳定性由什么原因导致？

研究问题2通过手动分析所有不稳定测试报告、对其进行分类，并将汇总结果列于表 II 中来回答。原因共有八个类别，另有两个特殊类别，即“其他”和“未知”。我们发现有一份不稳定性报告同时包含与随机性和浮点运算相关的问题及提交记录。因此，表 II 中不稳定测试报告的总数为 47，而不是 46（表 I 中不稳定测试报告的总数）。

我们的原因与 CC 软件相关原因之间有哪些差异？文献中已有多项针对 CC 软件中不稳定测试的分析 [1], [4]–[7]。例如，Luo 等人 [1] 确定了 CC 软件不稳定性的 10 个原因：异步等待、并发、测试顺序依赖、资源泄漏、网络、时间、输入输出、随机性、浮点运算和无序集合。 [1] 中的大多数不稳定测试与 Java 项目和分布式系统（例如 Apache HBase 和 Hadoop）有关。我们与其研究共有五个类别：并发、网络、随机性、浮点运算和无序集合。

[1] 中不稳定性的主要原因是异步等待和并发。我们的不稳定测试报告中，最主要的原因是随机性（21%）和软件环境（15%）。为什么会出现这种差异？

研究人员已将随机性确定为通用 CC 软件中不稳定性的非主要原因 [1], [5], [6]。一项调查 [5] 表明，在 1% 到 5% 的案例中（具体取决于软件），不稳定测试由随机性导致，而异步等待和并发始终是主要原因。

我们的研究结果更接近于那些关注概率编程和机器学习软件的研究：其中60%的不稳定测试可能是由随机性导致的 [7]。由于伪随机数生成器广泛用于量子计算编程（见第 III-B1节），随机性对量子程序中的不稳定测试报告有着显著影响。

让我们更详细地考察表 II 中列出的八种原因（以及两个特殊情况）。

表 II：基于不稳定测试报告的原因类别和修复模式数量。

原因类别	修复模式								总计	百分比
	修复随机种子	更改软件环境	设置为单线程	调整容差	添加异常处理程序	同步	使用键排序	其他		
随机性	9							1	10	21%
软件环境		4						3	7	15%
多线程			2					4	6	13%
浮点运算				3				2	5	11%
可视化								3	3	6%
未处理异常					3				3	6%
网络						1			1	2%
无序集合							1		1	2%
其他								7	7	15%
未知								4	4	9%
总计	9	4	2	3	3	1	1	24	47	100%
百分比	19%	9%	4%	6%	6%	2%	2%	51%	100%	

1）随机性：如上所述，使用伪随机数生成器是量子程序不稳定性报告最常见的原因（占 21%）。伪随机数生成器每次运行都会产生不同的输出，这可能导致测试不稳定。鉴于大多数测试用例都在量子计算机的模拟器上执行，而这些模拟器严重依赖伪随机数生成器来模拟量子计算机的非确定性，因此这成为不稳定测试报告的首要原因也就不足为奇了。

伪随机数生成器用于量子程序及其相关测试套件。我们的第一个示例（清单 1）展示了在 qiskit-terra 的问题报告 #3533 中发现的量子程序问题。双量子比特分解器（详见 [15]）的构造函数在第 6 行和第 7 行包含一个非确定性组件。该分解器在生成同一酉门的受控版本时可能产生随机结果。例如，清单 2 在第 1 行创建一个 UnitaryGate 实例，而第 5 行的相等操作会随机返回“True”或“False”。

```
1 类 TwoQubitWeylDecomposition:
2     ...
3     def __init__(self, unitary_matrix):
4         ...
5         for _ in range(100):
6             M2real = np.random.randn()*M2.real \
7                 + np.random.randn()*M2.imag
```

清单 1：量子程序中可能导致不稳定测试的随机性示例。

```
1 uni = 酉门([[1, 0, 0, 0],
2
3
4
un
```

清单 2：用于复现不稳定测试的测试代码。

```
1 定义测试_追加_电路(自身，数量_量子比特): 2         ...3 首先_
电
在
```

清单 3：测试用例代码中一个与伪随机数生成器相关的不稳定测试示例。

清单 3 中的第二个示例（基于 qiskit-terra 中的问题报告 #5217）展示了一个测试用例问题。用于检查追加量子电路是否正常工作的测试逻辑 test_append_circuit 是正确的。然而，第 3 行和第 6 行导致测试出现不稳定性，因为函数 random_circuit 会使用随机选取的种子（默认情况下）生成随机量子电路。因此，测试会因随机性而偶尔失败。我们将在第 III-C1 节讨论如何修复清单 1 和清单 3 中的不稳定性。

2）软件环境：此类别包括由特定软件或库依赖问题导致的不稳定测试。例如，NetKet 中的拉取请求 #1369 讨论了一个仅在 GitHub Actions 的 Python 3.10 软件环境中出现的不稳定测试。该拉取请求开头包含以下评论。

> “不知道为什么，但在 GitHub runner 的 Python 3.10 环境中，这个测试一直失败。我从未在本地复现过，所以我认为这不是真正的失败。只需简化 [sic] 该测试以避免这种情况。”

修复此问题的一种常见方法是更改软件环境，具体内容将在第 III-C2 节中介绍。

3）多线程：此类不稳定测试由多线程问题引起，例如并发和过载。例如，qiskit-terra 中的问题报告 #5904 描述了一个由并行构建导致地址冲突而引发的不稳定测试。此示例中的代码见清单 4，其中显示 sphinx-build 在 N 个进程上并行生成文档，N 是中央处理器的数量（即通过参数 -j auto 指定）。虽然并行化提高了吞吐量，但当 sphinx-build 的多个任务相互竞争时会导致错误。第 III-C3 节给出了针对这一原因的修复模式。

```
1 命令 = sphinx-build -W -b html -j
a
```

清单 4：一个与多线程相关的不稳定测试示例。

4）浮点运算：如果不稳定测试是由于舍入误差等挑战而发生的，我们就将其定义为与浮点运算相关的不稳定测试。例如，在 NetKet 的拉取请求 #1147（清单 5）中，硬编码的容差值 10^{-5}（如第 5 行所示）会导致测试用例间歇性失败。

```
1 定义测试_vmc_函数: 2     ha、sx、ma、sampler、driver
=
dr
```

```
approx(ma.expect(ha).mean, abs=1e-5)
```

清单 5：一个与浮点运算相关的不稳定测试示例。

为了解决这个问题，开发人员会修改断言以调整容差、对实际值进行舍入，或彻底移除不稳定测试用例。我们将在第 III-C4 节详细介绍开发人员如何应对这一挑战。

5) 可视化：这类不稳定测试与图像生成有关。例如，qiskit-terra 有一个测试管理器，可按顺序安排可视化测试，并允许这些测试彼此通信。qiskit-terra 中的问题 #3283 报告了测试管理器的可视化器存在不稳定测试，原因是参考索引已过时。

在这一类别中，我们无法确定一种反复出现的修复模式。三个报告（均与 qiskit-terra 有关）各自都有独特的解决方案。

6) 未处理异常：这类不稳定测试是由未能妥善处理异常的代码导致的。例如，清单 6 展示了 Microsoft/QuantumLibraries 中 #398 的问题报告里的堆栈跟踪。在这种情况下，如果“number”不在预期范围内，测试就可能偶尔失败。我们将在第 III-C5 节探讨针对这一原因的修复模式。

```
未处理异常。
Mi
nu
```

清单 6：未处理异常示例。

7) 网络：此类别中的不稳定测试是由网络相关问题导致的，例如网络或服务器不稳定。例如，qiskit-ibm-runtime 的问题 #584 报告了一个由超时和套接字连接问题导致的不稳定测试。如清单 7 所示，test_websocket_proxy 测试失败，因为作业在 WebSocket 连接建立之前就已完成。第 III-C6 节介绍了修复此原因的一种方法。

```
失败：测试_WebSocket_代理（测试.集成。测试_结果.
Te
```

清单 7：网络相关不稳定测试示例。

8) 无序集合：这一类不稳定测试特有于 Python（不过，理论上也可能出现在其他语言中）。从 Python 3.3 到 3.7 [6]，字典（哈希映射或哈希表）以无序集合的形式实现。当测试存在顺序依赖时，测试结果可能变得非确定性，从而导致不稳定性。例如，qiskit-terra 中的拉取请求 #8627 揭示了一个不稳定测试：该测试使用插入顺序来比较两个字典，因此会产生非确定性结果。第 III-C7 节将探讨修复此原因的一种方法。

9）其他：如果不稳定测试报告中，某个原因只出现过一次，且该原因在以往研究中尚未分类（例如， [1], [5], [6]），则将其纳入此特殊类别。

例如，在清单 8（Microsoft/Quantum 中的问题报告 #62）中，左双引号与微软之间有一个空格，这会意外地向测试中注入一个随机值，并导致间歇性失败。

```
fixedSeeds.Add(" Microsoft.Quantum.Tests.
    RobustPhaseEstimationTest", 2020776761);
```

清单 8：一个拼写错误导致不稳定测试。

另一个例子是在 Jupyter Notebook 中保留本地输出，并阻止不稳定测试在持续集成流水线中运行（TensorFlow/quantum 中的拉取请求 #453）。

最后一个例子见于 qiskit-aer 的拉取请求 #795。它涉及量子计算的配置变更（因为工程师会不断调整计算机的配置）。每当配置发生变化时，工程师都会更新这台量子计算模拟器的配置文件。然而，测试用例的预期值是硬编码的，因此导致测试用例间歇性失败。为解决这个问题，测试人员开始从配置文件中动态读取配置参数，而不是将其硬编码。

10）未知：在这种特殊情况下，如果关于不稳定测试原因或修复的信息不足，则将其原因归类为未知；也就是说，若没有详细的讨论或相应的提交，就无法确定不稳定测试的原因。例如，qiskit-machine-learning 的问题报告 #185 中提供了不稳定测试堆栈跟踪。由于信息不足，该报告后来被关闭。

C. RQ3：量子程序员如何修复不稳定测试?

表 II 展示了我们发现的不稳定测试报告对应的七种修复模式（基于与不稳定性相关的提交和拉取请求）。这些模式涵盖了 49% 的修复报告。其余修复方法都是一次性的，归入“其他”修复模式（51%）。没有对应修复模式的唯一原因类别是“可视化”，因为不存在反复出现的模式，而且我们只在 qiskit-terra 中观察到与可视化相关的不稳定测试。

1）固定种子：当用于初始化伪随机数生成器的种子固定时，更容易将变量与常量进行比较。因此，与随机性相关的不稳定测试报告中，有 10 例中的 9 例（即所有不稳定测试报告的 19%）通过将随机种子值设为常量来修复（或者结合降低收敛阈值，正如 qiskit-terra 的拉取请求 #8820 所示）。例如，为避免清单 3 中的不稳定性，清单 9（qiskit-terra 中的拉取请求 #5599）展示了一种解决方案：将默认的非固定随机种子值替换为固定种子值。

```
- 第一个_电路 = 随机_电路（量子比特数_量子比特[0],深度）+ 第一
```

虽然通过固定伪随机数生成器的种子可以控制不稳定性，但这种方法可能会降低测试的有效性，因为它限制了可能暴露真实错误的执行路径 [8]。此外，伪随机数算法发生变化后，测试用例可能会再次变得不稳定。

我们尚未发现针对此问题的稳健修复方法，包括多次运行测试用例以计算成功和失败执行的分布[1]，然后执行分布分析，以评估代码的新更改是否会改变该分布（详情请参见 [8] ）。这可能是由于需要更高的计算和执行时间。此外，程序员可能不希望从头开始构建这样的测试框架，也没有意识到类似的测试框架（例如 FLEX [8], ）已经存在。

最后，我们发现有一个不稳定测试通过将伪随机数生成器替换为确定性公式得以修复（qiskit-terra 的拉取请求 #3585，如清单 10 所示）。

```
- 对于_ 在range(100)中:  - M2实部 =
np
np

```

清单 10：清单 1 中消除随机性来源的示例。

2）更改软件环境：通过更新或更改开发环境，可以消除不稳定测试。在与软件环境相关的七份不稳定测试报告中（见表 II），有四份（占所有报告的 9%）通过升级或更改依赖项得到修复。清单 11 展示了一个修复示例：移除对特定 numpy 版本的依赖（更多详情见 qiskit-terra 的拉取请求 #4656）。

```
- numpy!=1.19
```

清单 11：修复库依赖的示例。

其余三份报告则分别通过更改持续集成流水线的配置（Microsoft/qdk-python 中的问题 #319）、简化测试用例（netket 中的拉取请求 #1369），或因依赖项发生变化而直接关闭报告（qiskit-aer 中的问题 #1466）得到处理。

3）设置为单线程：通过将线程数限制为 1，可以解决与多线程相关的不稳定测试。设置为单线程解决了六项与多线程相关的不稳定测试中的两项（占所有报告的 4%）。例如，在 qiskit-experiments 的拉取请求 #780 中，开发人员发现，由于某些环境禁用了多线程，同时运行多个测试时偶尔会出现超时问题。虽然禁用多线程可以解决不稳定性，但可能会带来性能开销。另一个例子见清单 12（qiskit-terra 中的拉取请求 #6539），其中展示了通过禁用并行化来修复清单 4 中的不稳定测试。

[1]尽管我们看到了一种将测试运行三次，并在测试至少通过一次时判定成功的策略（netket 中的拉取请求 #724）。

```
- sphinx-build -W -b html -j auto docs/ docs/_build/
    html {posargs}
+ sphinx-build -W -b html docs/ docs/_build/html {
    posargs}
```

清单 12：清单 4 中不稳定测试的修复。

4) 调整容差：对于与浮点运算相关的不稳定测试，可以调整容差。五份报告中有三份（占所有报告的 6%）通过这种方式得到修复。例如，可以手动增大容差，例如将容差值加倍，如清单 13 所示。然而，硬编码的宽松容差可能会导致假阴性测试结果增加，从而影响程序的正确性。

```
- np.testing.assert_allclose(result.values,
    [-1.307397243478641], rtol=0.05)
+ np.testing.assert_allclose(result.values,
    [-1.307397243478641], rtol=0.1)
```

清单 13：qiskit-terra 的拉取请求 #8820 中容差增加的示例。

设置动态容差可能是更稳健的解决方案。例如，要修复清单 5 中所示的问题，开发人员会动态计算容差级别，如清单 14 第 2 行和第 3 行所示。

```
- 断言 driver.energy.mean  == approx(ma.expect(ha).mean,


```

清单 14：清单 5 中增加容差的示例。

开发人员还会对实际值进行舍入，以应对数值错误（例如，qiskit-terra 的拉取请求 #4835）；如果某个不稳定测试用例被认为并非必要，也会直接将其删除（例如，qiskit-terra 的拉取请求 #8582）。

5）添加异常处理程序：对于因未处理异常导致的不稳定测试，可以通过添加异常处理程序或移除异常来缓解（所有报告中的三例，即 6%，都是通过这种方式修复的）。例如，微软/Microsoft/QuantumLibraries 的开发人员在拉取请求 #399 中添加了一个 if 条件，以移除任何意外的负系数（见清单 15）。

```
- 原地应用异或(keepCoeff[idx],keepCoeffRegister);+ if
(k
ke

```

清单 15：清单 6 中不稳定测试的修复。

6）同步：我们观察到一份网络相关的不稳定测试报告（即所有报告的 2%），涉及 qiskit-ibm-runtime 中的问题 #584 和拉取请求 #588。在此，回调函数测试依赖于套接字测试。然而，有时回调测试会在套接字测试之前完成，从而导致不稳定性。开发人员增加回调测试的迭代次数，让套接字测试有足够的时间先完成（见清单 16）。这可能不是最佳解决方案，因为迭代次数是硬编码的。理想的解决方案是

将回调测试与套接字测试的完成进行同步。

```
- 作业 = 自身。_运行_程序（服务，迭代次数=1，回调函数=结果
=1
```

清单 16：清单中不稳定测试的修复方法 g 7.

7）使用键排序：对于由无序 Python 字典导致的不稳定测试，开发人员可以使用键值进行排序，而不是使用插入顺序。在 Python 3.8 及更高版本中，字典会保留顺序，因此升级 Python 环境可以解决该问题。然而，升级 Python 可能导致依赖问题。一份不稳定测试问题报告（即所有报告的 2 %）涉及这一原因及修复方法。

IV. 效度威胁

有效性威胁根据 [16], [17] 进行分类。

内部效度与构念效度。数据采集和清理是容易出错的过程。数据由人工收集。几位共同作者分别检查了不稳定测试的问题报告搜索结果，随后共同审阅并讨论了研究结果，最终确定了清单。用于筛选问题报告的 10 个关键词（第 II 节讨论）可能导致假阴性。因此，我们低估了与不稳定测试相关的问题报告数量。然而，即使是这组有限的关键词，也证明了量子计算程序中存在不稳定测试。未来，我们将把这项研究扩展到自动化方法和工具，以确定这些报告是否与不稳定性相关。

外部效度与结论效度。一般而言，软件工程研究受到现实世界变异性的影响，而泛化问题只能得到部分解决 [18]。我们需要对理论总体进行泛化，并理解其架构相似关系，以建立理论 [18]。尽管本研究使用了 14 个开源项目，但我们的发现未必能推广到其他项目。不过，基于这项试点研究的发现，我们可以通过精心设计的受控实验，对其他量子软件产品开展同样的实证考察。通过此类未来研究，我们希望相关领域的研究者能够扩展我们的原因分类体系，并识别相关模式，最终形成量子计算中不稳定测试的一般理论。

V. 相关工作

关于量子程序测试与调试的文献日益增多。由于量子力学的特性，量子程序难以测试 [19]。软件工程原则正被应用于量子程序的测试与调试 [19]–[21]；关于量子软件工程研究的全面概述，请参见 [22], [23] 。

学术界采取了多种方法来应对这一挑战。通过在代码中添加断言检查 [24]–[27] ，可以简化量子计算程序的测试；在某些情况下，也可以引入调试技巧，例如提取经典信息 [20]。识别量子程序中的缺陷模式有助于缺陷分析与分类 [28]。我们还可以调整经典的模糊测试技术 [29] ，或执行基于属性的测试 [30]。此外，还可以在经典计算（CC）模拟器中调试量子程序（Qiskit 和 Q# 等框架都 readily 提供此选项），但它们只能用于规模较小的问题 [19], [31]。

据我们所知，目前尚无针对量子程序中不稳定测试的研究。正如第III-B节所述，Luo 等人 [1] 总结了经典计算机软件中导致不稳定性的10种常见原因，其中有四种与我们的研究重合。 [5]–[7], 也提出了经典计算软件的类似分类体系，可与我们的研究互为补充。

VI. 结论与未来工作

本文考察了14个量子程序中的不稳定性。我们在其中12个程序中检测到46份不稳定测试报告，这意味着这些程序的问题报告中至少有0.26%至1.85%与不稳定性有关。此外，我们归纳出量子计算代码中导致不稳定测试的八类原因，以及七种常见的修复方法。最后，我们观察到，量子程序员仅采用了近期开发出来的、用于处理不稳定测试的部分软件工程技术。我们希望这些发现能帮助研究人员和开发人员在设计和测试量子程序时降低不稳定性风险。

这项工作有多种扩展方式。未来，我们将探索更多代码仓库和关键词，以识别量子计算中的更多不稳定测试模式。此外，我们计划计算受不稳定性影响的测试用例比例，以便对经典计算程序和量子计算程序进行直接的统计比较。

我们的长期目标是：1）基于本文所识别的缺陷模式，将检测量子程序中不稳定测试的手动方法实现自动化；2）基于我们所识别的修复模式，开发能够自动修复量子计算不稳定测试的方法和工具。对于第一个目标，我们将针对每个量子不稳定测试类别开发一个自动化工具，并将这些工具集成到一个涵盖最常见不稳定测试类别的框架中。对于第二个目标，我们将采取与第一个目标类似的路径，即先开发独立的自动化修复方法和工具，随后构建一个更全面的框架来修复不稳定测试。这两个目标将有机结合，为缓解量子程序中的不稳定性提供高效的解决方案。

我们希望业界能够利用我们的研究成果，改进从缺陷报告文本中检测不稳定性的方法，自动识别不稳定测试，提出缓解策略，并丰富不稳定测试数据集。此外，通过提高人们对经典程序与量子程序之间相似之处和差异的认识，我们希望鼓励量子程序员采用软件工程领域开发的一些工具，以提升量子软件质量。

参考文献

[1] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov, 《不稳定测试的实证分析》，载于第22届ACM SIGSOFT软件工程基础国际研讨会论文集，2014年，第643–653页。[2] J. Micco, 《谷歌持续集成测试的现状》，2017年。 [在线]。可访问：https://storage.googleapis.com/pub-tools- public-publication-data/pdf/45880.pdf

[3] A. Memon, Z. Gao, B. Nguyen, S. Dhanda, E. Nickell, R. Siemborski, and J. Micco, 《驯服谷歌规模的持续测试》，载于《2017年IEEE/ACM第39届国际软件工程会议：软件工程实践分会（ICSE-SEIP）》论文集。IEEE，2017年，第233–242页。[4] M. Eck, F. Palomba, M. Castelluccio, and A. Bacchelli, 《理解不稳定测试：开发者视角》，载于《2019年第27届ACM欧洲软件工程会议与软件工程基础研讨会联合会议》论文集，2019年，第830–840页。[5] O. Parry, G. M. Kapfhammer, M. Hilton, and P. McMinn, 《不稳定测试综述》，《ACM软件工程与方法论汇刊》（TOSEM），第31卷，第1期，第1–74页，2021年。[6] M. Gruber, S. Lukasczyk, F. Kroiß, and G. Fraser, 《Python中不稳定测试的实证研究》，载于《2021年第14届IEEE软件测试、验证与确认大会（ICST）》论文集。IEEE，2021年，第148–158页。[7] S. Dutta, A. Shi, R. Choudhary, Z. Zhang, A. Jain, and S. Misailovic, 《检测概率与机器学习应用中的不稳定测试》，载于第29届ACM SIGSOFT国际软件测试与分析研讨会论文集，2020年，第211–224页。[8] S. Dutta, A. Shi, and S. Misailovic, 《FLEX：通过更新断言边界修复机器学习项目中的不稳定测试》，载于《2021年第29届ACM欧洲软件工程会议与软件工程基础研讨会联合会议》论文集，第603–614页。[9] D. C. Marinescu, 《经典与量子信息》。Academic Press，2011年。[10] M. A. Nielsen and I. L. Chuang, 《量子计算与量子信息：十周年纪念版》。剑桥大学出版社，2010年。[11] M. Treinish, J. Gambetta et al., 《Qiskit/qiskit：Qiskit 0.39.5》，1月。2023 [ ] 10 5281 7545230 。在线。[12] 2023 [可访问：https://doi.org/ . /zenodo。微软，《Q#与量子开发工具包》。在线]。可访问：https://azure.microsoft.com/en-us/resources/development-kit/quantum-computing [13] TensorFlow，《TensorFlow Quantum》，2023年。 [在线]。可访问：https:// www.tensorflow.org/quantum [14] NetKet，《NetKet——量子物理机器学习工具箱》，2023年。 [在线]。可访问：https://www.netket.org/ [15] B. Kraus and J. I. Cirac, 《使用双量子比特门实现纠缠的最优生成》，《物理评论A》，第63卷，第062309页，2001年5月。 [在线]。可访问：https://link.aps.org/doi/10 1103 63 062309 。/[16] PhysRevA。C. Wohlin, P. Runeson, M. Host, M. Ohlsson, B. Regnell, and A. Wessl´en, 《软件工程实验》，计算机科学丛书。Springer Berlin Heidelberg，2012年。[17] R. Yin, 《案例研究：设计与方法》，应用社会研究方法丛书。SAGE出版社，2009年。[18] R. J. Wieringa and M. Daneva, 《软件工程理论泛化的六种策略》，《计算机编程科学》，第101卷，第136–152页，2015年4月。[19] A. Miranskyy and L. Zhang, 《量子程序测试》，载于《2019年IEEE/ACM第41届国际软件工程会议：新思想与新兴成果（ICSE-NIER）》论文集。IEEE，2019年，第57–60页。[20] A. Miranskyy, L. Zhang, and J. Doliskani, 《你的量子程序没有错误吗？》，载于《ACM/IEEE第42届国际软件工程会议：新思想与新兴成果》论文集，ICSE-NIER '20。ACM，2020年，第29–32页。[21] ——, 《量子软件的测试与调试》，arXiv预印本 arXiv:2103.09172，2021年。[22] J. Zhao, 《量子软件工程：现状与展望》，arXiv预印本 arXiv:2007.07047，2020年。[23] M. De Stefano, F. Pecorelli, D. Di Nucci, F. Palomba, and A. De Lucia, 《量子编程的软件工程：我们走了多远？》，《系统与软件杂志》，第190卷，第111326页，2022年。[24] Y. Huang and M. Martonosi, 《用于验证模式并发现量子程序错误的统计断言》，载于《第46届国际计算机体系结构研讨会》论文集，ISCA'19。美国计算机协会，2019年，第541–553页。[25] G. Li, L. Zhou, N. Yu, Y. Ding, M. Ying, and Y. Xie, 《基于投影的运行时断言：用于测试和调试量子程序》，《ACM编程语言论文集》，第4卷，OOPSLA期，第150:1–150:29页，2020年。[26] J. Liu、G. T. Byrd 和 H. Zhou，"量子计算中的动态运行时断言量子电路"，载于《第25届编程语言与操作系统架构支持国际会议论文集》，系列：ASPLOS'20。美国计算机协会，2020年，第1017–1030页。[27] S. Ali、P. Arcaini、X. Wang 和 T. Yue，"评估输入和输出覆盖准则在量子程序测试中的有效性"，载于《2021年第14届IEEE软件测试、验证与确认大会（ICST）》。IEEE，2021年，第13–23页。[28] P. Zhao、J. Zhao 和 L. Ma，"识别量子程序中的缺陷模式"，载于《2021年IEEE/ACM第2届量子软件工程国际研讨会（Q-SE）论文集》。IEEE，2021年，第16–21页。[29] J. Wang、M. Gao、Y. Jiang、J. Lou、Y. Gao、D. Zhang 和 J. Sun，"Quanfuzz：量子程序的模糊测试"，arXiv预印本 arXiv:1810.10310，2018年。[30] S. Honarvar、M. R. Mousavi 和 R. Nagarajan，"q# 中量子程序的基于属性的测试"，载于《IEEE/ACM第42届软件工程国际会议研讨会论文集》，2020年，第430–435页。[31] M. P. Usaola，"量子软件测试"，载于《第1届量子软件工程与编程国际研讨会短文论文集》，系列：CEUR研讨会论文集。CEUR-WS.org，2020年，第57–63页。