变异测试应用研究综述

靳琦清 201250063、闻博 201250065、何沛阳 201250192、华广松 201840309
南京大学 软件学院, 江苏 南京 210093

摘要：在软件开发和维护过程中，测试必不可少。变异测试作为一种经典的软件测试，经过国内外学者四十多年的研究，已经在许多方向有了实际应用。本文旨在对变异测试近十年来的应用工作进行梳理和归纳，寻找变异测试应用的效果、优点和不足，总结变异测试应用的现状和前景。
关键词：变异测试应用；软件测试；综述

1. 引言

变异测试是一种基于缺陷的软件测试技术，在近四十年得到国内外学者的广泛关注，并取得了一些研究成果。对已有的研究工作进行总结，将其分为变异测试原理、优化和应用三个模块，变异测试（Mutation Testing）（有时也叫做“变异分析”）是一种在细节方面改进程序源代码的软件测试方法。这些所谓的变异，是基于良好定义的变异操作，这些操作或者是模拟典型应用错误（例如：使用错误的操作符或者变量名字），或者是强制产生有效地测试（例如使得每个表达式都等于 0）。目的是帮助测试者发现有效地测试，或者定位测试数据的弱点，或者是在执行中很少（或从不）使用的代码的弱点。变异测试通过选择一些变异操作，并对于每一个可执行代码段依次把它们应用在源代码中。对程序使用变异操作的结果叫做一个突变异种。如果测试单元可以察觉到错误（即：一个测试失败了），那么就说该突变异种被杀害了。弱的突变测试（弱的突变覆盖）只要求满足第一个条件。强的突变测试要求满足两个条件。强突变更有效，因此它保证测试单元可以真实的捕捉错误。弱突变近似于代码覆盖方法。它只需较少的计算能力来保证测试单元满足弱突变测试。

变异测试作为一种软件检测技术，在不同和领域都存在大量实际应用，本综述试图总结研究近十年变异测试应用的各类文献资料，对它们分别进行分类和分析，并将其研究的要点提取出来，以对各项研究进行区别与联系。经过分类，我们将变异测试的应用分为优化回归测试、缺陷定位和自动修复、AI 测试应用、面向特定编程语言的应用以及其他应用五个大类。

接下来的部分如下组织：第 2 节介绍综述中文献的检索、筛选和分类信息；第 3 节重点介绍在不同方向和领域中变异测试的应用；第 4 节对本文内容进行总结。

2. 文献检索

2.1 检索方法

为了对变异测试应用进行较全面的总结，我们首先在谷歌学术（Google Scholar）中使用关键词进行搜索，随后人为检查搜索结果确定符合主题的相关文献，最后阅读相关文献的参考文献部分抽取剩余相关工作。由于本综述的研究主题为变异测试应用，我们首先在谷歌学术中测试了关键词“mutation test”的搜索结果，发现这些词只能检索出极小部分参考文献，为了扩大文献精确度，我们使用“mutation test apply”、“mutation”、“mutation software test”等作为最终的检索关键词在 DBLP 上进行搜索。为了寻找变异测试在具体某

一领域或某一方向的应用，我们还使用“regression test”、“TCP”、“fault localization”、“automated repair”、“deep learning”、“deep neural network”等作为辅助关键词进行搜索。

2.2 文献筛选

在使用 2.2.1 中关键词进行搜索得到相关文献候选集后，我们对其中每篇文献进行人工审核，删除不符合研究主题的文献。在本调研中，我们只关注变异测试在近五年或者十年内的最新应用研究。同时，鉴于许多文献都是优化变异测试技术本身的问题，本文主要关注基于变异测试原理、技术或者思想，从真实场景反馈的问题出发，研究其应用现状及进展， 为未来研究提供重要的实践评估标准。更进一步地，本文重点关注变异测试在具体应用中发挥的作用问题，冗余的相关研究不纳入本文的调研范围。排除冗余应用相关研究主要出于如下考虑：本文重点变异测试在不同领域的应用，作为报告了同一应用的冗余报告，研究其中的一份，有一定代表性。

2.3 文献分类

通过文献检索和筛选，最终获取到 29 篇文献用于本文研究，文献发表时间为 2010-2022 年。在这 29 篇论文中，会议论文 19 篇，期刊 10 篇。其中，有 4 篇重点讨论了优化回归测试应用；有 8 篇主要论述了缺陷定位和自动化修复；有 6 篇试图将变异测试用于 AI 测试；有 9 篇讨论了面向特定编程语言的应用；最后有 2 篇论文讨论了移动应用。

3. 变异测试应用

3.1 优化回归测试

回归测试是软件开发和维护过程中保障软件质量的一种重要手段，用于保障代码修改的正确性，并避免代码修改对被测程序的其他模块造成副作用。目前常见的回归测试技术包括：失效测试用例的识别和修复技术、测试用例选择技术、测试用例优先级排序技术、测试用例集约简技术和测试用例集扩充技术等。许多研究人员正在考虑如何应用变异测试技术来优化回归测试。

Rudolf 等人[1]调查了变异测试在提高安全关键软件系统中单元测试质量方面的适用性和实用性。在改进测试时，在代码中发现了两个新的错误，并且优化后的代码测试覆盖率符合预定要求。为此，他们又评估了现有测试的故障检测有效性，并逐一识别了测试中的缺陷。结果表明，变异分析对于改进验证安全关键软件有效性的过程是十分有用的。变异测试能够评估已经达到全覆盖的测试套件的质量。此外，变异测试提供了测试用例中很难发现的缺陷的提示，这些反馈可以直接用于修改和增强测试。

由于回归测试中可能会聚集大量的测试用例，如果乱序执行或全部执行都有可能产生不必要的大量时间浪费，因此需要对测试用例的执行顺序进行调度，以便尽早发现后一个版本的错误，这就是软件演进中的测试用例优先化（TCP）。由于测试用例优先级排序需要用程序的故障作为参考，但是真实的错误数据通常不足以支撑此排序，为此，研究人员开始使用变异体这种可以大量生成的人工错误来代替真实错误数据。例如，Y.Lou 等人[2]提出了一种针对软件进化的测试用例优先排序方法，该方法首先利用基于早期版本与后期版本差异的变异故障来模拟软件进化过程中发生的真实故障，然后根据测试用例的故障检测能力(基于变异故障定义的故障检测能力)来调度测试用例的执行顺序。

使用变异体作为替代的基本假设是，在杀死大量变异的优先级测试用例集和检测大量真实错误的用例集之间有很强的相关性。但是，目前还不清楚在多大程度上，变异体上的 TCP 性能可以代表在实际故障上实现的性能。为了回答这个问题，Qi 等人[3]进行了一次实证研究，

比较 TCP 技术应用于实际故障和变异故障的性能。他们的研究范围包括八个经过充分研究的 TCP 方法，35k+变异错误，以及在 Defects4J 数据集中来自五个 Java 系统的 357 个真实的错误。他们的研究结果表明，所研究的 TCP 技术在变异体上的相对性能可能与在真实故障上的性能没有很强的相关性，这取决于主题程序的属性。这表明，在某些情况下，在一组变异体上表现最好的技术在实际应用到实际故障时可能并不是最好的技术。他们还说明了这些相关性对于由不同算子产生的变异体是不同的，这取决于所选择的算子是否反映了主题程序的典型错误。这突出了为特定程序域开发变异运算符的重要性，尤其是对 TCP 而言。

针对 TCP，Donghwan 等人[4]提出了一种新的测试用例优先化技术——一种称为变异区分图(MDG)的图形模型，它结合了基于变异和基于多样性的方法，展示了基于变异的优先级标准在什么时候执行得很差以及为什么执行得很差。该图形模型演示了关于变异杀死和区分的故障检测测试用例的分布，变异区分旨在将一个变异的行为与另一个变异区别开来，而不是与原始程序区别开来。实验结果表明，基于变异的优先级排序至少在 88.9%和 77.8%的故障上比随机优先级排序和基于覆盖的优先级排序更有效或同样有效。

3.2 缺陷定位和自动修复

在整个调试活动中，缺陷定位一直是一个代价昂贵的阶段。传统的人工定位需要测试者努力理解被测程序的复杂内部逻辑，以及测试通过和失败的运行之间的差异，这无疑是一个耗时耗力的工程，特别是对经验不足的开发者而言。

由此，自动缺陷定位技术得到了广泛的研究，目前最流行的方法是基于频谱的缺陷定位（SBFL）。该方法使用程序谱（即测试套件执行的信息概要），根据预测的包含缺陷的风险对程序语句进行排序。然后，开发人员将按照给定排名中语句的顺序检查被测程序，希望在排名顶部附近会遇到错误语句。但是这样的方法消耗较多程序运行时间成本和资源成本，且测试用例的数量和质量对缺陷定位性能影响较大。此外，此方法因为使用的模型（排名的线性检验）不具备足够的现实性和较低的准确性，也一直受到批评。

因此，有很多人开始考虑其他方法的缺陷定位，基于变异测试思想的缺陷定位正是目前许多人的研究方向。

Thierry Titcheu Chekam 等人[5]研究发现，只有在一个很高的变异评分水平以上，变异评分和缺陷提示之间才有很强的联系。在这个水平以下，变异评分与故障揭示则完全不相关。在实践中，这意味如果一个测试套件不能达到相对高的变异分数，则很容易受到噪音的影响，测试人员不应该对基于它们的测试有信心。相对应的，同样的研究表明，强变异充分测试套件能够揭示至少 90%的程序错误。

Mike Papadakis 等人[6]考虑了一种基于变异分析的缺陷定位方法，通过人为地向代码中注入缺陷来探索程序的行为，并获得一些可用于自动故障定位的信息，该方法与现有的基于频谱的缺陷定位方法进行比较，结果显示明显优于后者。又由于应用变异分析需要消耗大量的计算资源，他们将选择性变异技术应用到缺陷定位问题，并确定了故障定位背景下具有代表性的变异算子集，来减少变异体数量。最终实验的结果表明，该方法能够将涉及的变异体数量减少 80%，而不会损失故障定位精度。

在之后，Mike Papadakis 等人[7]在另一些研究中提出了一种基于变异分析的故障定位方法——Metallaxis，其创新之处在于它使用了变异体，并将它们与错误的程序位置联系起来。因此，大部分因测试失败而死亡的变异体提供了一个关于故障位置的良好指示。使用 Metallaxis 的实验表明，它比基于语句的方法更有效。即使在有助于降低变异成本的技术（如变异采样）的情况下，这也是正确的。此外，受控实验的结果表明，使用变异作为测试技术有利于缺陷定位过程。因此，通过使用基于变异的测试套件而不是基于块或基于分支的测试

套件，可以显著改进缺陷定位。他们还证明了 Metallaxis 具有良好的可扩展性。

Seokhyeon Moon 等人[8]提出了一种新的基于变异测试的故障定位技术——MUSE, MUSE 使用变异分析来唯一地捕捉单个程序语句和观察到的故障之间的关系，而不受来自块结构的共享排名的强制影响。其基本思想是，由于错误的程序可以通过修改错误的语句来修复，因此修改错误的语句将使更多失败的测试用例通过。相比之下，由于改变正确的语句会引入新的错误语句，因此改变正确的语句将使更多通过的测试用例失败。

Seokhyeon Moon 等人还为缺陷定位技术提出了一种新的评估指标，LIL（位置信息损失），其利用信息理论，通过定位技术测量断层的真实位置和预测位置之间的信息损失。LIL 可以应用于任何故障定位技术，并描述任意数量缺陷的定位。最终，使用传统的消耗指标和提出的 LIL 指标，针对五个真实世界程序的 14 个错误版本评估 MUSE。结果表明，平均而言，MUSE 比目前最先进的 SBFL 技术 Op2 准确约 25 倍。

一般情况下，等价变异体被认为是导致变异测试应用成本加剧的主要原因之一，很多研究都在努力尝试如何减少或消灭等价变异体，然而许多研究人员已经开始将这视为某些情况下的优势。Paolo Arcaini 等人[9]发现，等价变异体可以用于识别、检测和消除静态异常（一种可以在不影响工件语义的情况下被删除的异常），换句话说，等价变异体可以被用于提高代码可读性和达到重构目的。

通过对这些文献的深入分析，可以发现将变异分析引入缺陷定位问题，在准确性上都明显优于目前主流的基于频谱的缺陷定位技术（SBFL）。然而两种方法都会消耗大量的计算资源，对于基于变异分析的缺陷定位，这是由于变异测试本身要求生成大量变异体（其中大部分都为无效的等价变异体）所造成的，因此研究人员也在积极探究使用各种技术，如使用选择性变异技术，来降低变异分析的资源消耗。

自动修复也一直是代码工程师们渴求的一项技术，因为传统的修复不仅费时费力，还可能引入更多新的错误，变异测试或变异测试的思想可以应用于此领域。Shin Hwei Tan 等人[10]在结合了变异分析和其他搜索算法之后，提出了一种回归修复技术——relifix。该方法通过使用一组代码转换来自动地更改语句的语法信息，然后通过搜索代码转换运算符来完成回归修复。经过实验对比，该方法比另一个回归修复方法 GenProg 能修复更多的错误，并且不会引入新的错误。

Ali Ghanbari 等人[11]则提出了在 JVM 字节码级别运行的实用摘要自动修复技术——PraPR，该方法使用一组传统的变异算子，在 JVM 字节码级别进行变异操作，最终实现摘要自动修复。实验表明，该方法于其他摘要自动修复技术相比，可以修复更多错误，并且速度要快 10 倍以上，同时还不存在在数据集上过度拟合的问题。

Satish Chandra 等人[12]提出了一种基于变异测试的调式技术——Angelic debugging，其思想是执行某种形式的数据状态变异，不断搜索编辑空间中所有能修复失败测试而不影响通过测试的变异程序，以纠正程序执行。最终他们在 Java PathFinder 模型检查器之上实现了该方法，实验表明，该方法在加速程序调试方面具有应用前景。

3.3 AI 测试

AI 技术在近十年来得到了迅速的发展，并且在许多领域都得到了应用，然而对于 AI 的测试与经典的软件测试差别较大。经典的程序设计中，输入的是规则（程序）和需要这些规则处理的数据，系统输出的是答案。因此，经典的软件测试，我们可以通过对比答案来验证和确认程序的正确性。然而，数据驱动的 AI 系统，不是通过编写明确的逻辑，而是通过数据来训练程序，输入是数据和从这些数据中预期得到的答案，AI 系统输出的是规则，这些规

则可应用于新的数据，自动计算出答案。可见，数据驱动的 AI 系统是一种新的编程范式，这种编程范式给 AI 测试带来了极大的挑战。

具体而言，AI 测试面临的挑战有（1）AI 系统输出结果很难预测；（2）AI 系统测试通过的准则很难确定；（3）AI 系统的输出结果随时间变化；（4）AI 系统需要更高效的持续测试方法；（5）AI 系统的性能依赖于数据。

AI 系统对数据的依赖性极高，而变异测试正是通过变异算子生成大量的变异体（变异体为程序或测试数据），因此许多研究人员想到将变异测试的思想应用到 AI 测试之中，并提出了很多新的方法。

Lorenz Klampf 等人[13]研究了变异测试在神经网络背景下的适用性。他们考虑了不同的神经网络实现，并利用可用的变异测试工具进行了实验评估，以检查是否可以使用普通的变异测试（即未用于神经网络的变异测试）来获得关于测试这些神经网络实现的程度的有价值信息。特别是，利用了神经网络的评估方法，其中利用可用的测试数据来计算变异分数，即检测到的变异和神经网络库实现中引入的所有变异的分数。此外，还研究了测试数据的大小对变异得分的影响。Lorenz Klampf 等人进行了考虑 MLP 和卷积神经网络的实验，表明变异分数不足以找到足够数量的变异体。此外，还扩展了实验，涵盖了不同大小的测试数据集的情况，也表明测试数据不足以测试神经网络。最终，得出结果为有必要为神经网络库设计专门的测试套件，以大幅提高变异分数。

Lei Ma 等人[14]提出了一个专门用于深度学习（DL）系统的变异测试框架，用以测量测试数据的质量。为此，Lei Ma 等人首先定义了一组源级变异算子，以向 DL 源（即训练数据和训练程序）注入故障。然后，又设计了一组模型级变异算子，该算子在没有训练过程的情况下直接将故障注入 DL 模型。测试数据的质量可以通过对注入故障的检测程度的分析来评估（为此还提出了两种 DL 特异性变异测试指标）。最终，在两个公共数据集（MNIST 和 CIFAR-10）和三个 DL 模型上证明了所提出的变异测试技术的有用性。

Qiang Hu 等人[15]提出了个用于前馈神经网络（FNN）和递归神经网络（RNN）的变异测试框架——DeepMutation++。DeepMutation++为 DeepMulation 中引入的 FNN 模型引入了八个模型级算子，并进一步提出了九个专门用于 RNN 模型的新算子。特别是，为了迎合 RNN 的特点，DeepMutation++支持静态变异生成以整体分析测试数据，以及动态变异生成以在运行时检测测试输入的脆弱片段。与深度神经网络（DNN）的结构覆盖标准不同，DeepMutation++能够直接提供关于 DNN 对输入的鲁棒性的反馈。 最终，Qiang Hu 等人证明了 DeepMutation++在 DNN 稳健性分析和测试数据脆弱片段检测的两个典型场景中的有用性：1）基于 MNIST 数据集的 FNN（即 LetNet-5）图像处理，2）基于 IMDB 数据集的 RNN（即 LSTM 和 GRU）文本情感分析。还发现，基于变异测试定义的度量可以是 DNN 鲁棒性的一个重要指标，与 DNN 抵抗对抗性攻击的鲁棒性有很强的相关性。

在主动自动机学习中存在着两种查询，成员查询和等价查询,其中成员查询对于学习黑盒系统很容易实现。等价查询则更难实现，一些研究提出了一致性测试算法，但这些算法存在两个问题，一是要求要求黑盒系统的状态数有一个固定的上限（通常是未知的），二是构建的测试套件的大小在这个范围内是指数级的。因此实现等价预言可以被视为自动机学习的真正瓶颈。Bernhard K. Aichernig 等人[16]提出了一种基于变异测试的等价查询的实现，更具体地说是基于模型的变异测试，其基于新的目标不是试图证明等价性，而是快速找到反例的思想，尽量减少执行等价查询的测试次数。实验中将随机测试与变异分析相结合，以实现测试的高变异性，并适当解决覆盖率问题。为了证明其有效性，本实验将该方法应用于学习实验，并将其性能与成熟的测试技术——部分 W 方法进行比较。该评估表明，该方法显著降低了

学习成本。在多次实验中，将成本降低了至少一个数量级。

变异测试思想在 AI 领域的另一个应用是用来检测 DNN 中的对抗样本攻击。研究表明，即使是训练有素的 DNN 也容易受到对抗样本的攻击，尤其是当 DNN 应用于分类任务时。对抗样本通常是通过对抗性扰动来制作的（即对原始样本进行细微修改，使 DNN 模型对样本进行错误标记），而这样的制作很容易完成。

现有的防御策略主要集中于增加攻击者制作对抗样本难度和基于训练辅助模型检测对抗样本两个思路进行，然而这些防御策略依赖于可用的对抗样本，因此通常仅限于防御特定攻击。

Jingyi Wang 等人[17]发现在 DNN 上施加随机变异，对抗样本比正常样本敏感得多。于是他们提出了一个灵敏度度量，用实证表明正常样本和对抗样本具有可区分的灵敏度。然后，整合了统计假设检验和模型变异检验，通过测量其灵敏度来检查输入样本在运行时是否可能是正常或敌对的。他们在 MNIST 和 CIFAR10 数据集上评估了该方法。结果表明，该方法可以有效、准确地检测由最先进的攻击方法生成的对抗样本。

Jingyi Wang 等人[18]在另一篇研究中提出了 nMutant，一种在运行时有效检测对抗样本的方法。通过对 MNIST 和 CIFAR10 数据集的实证研究，表明该方法对许多现有的攻击方法（例如 FGSM、C&W、JSMA 和 Blockbox）是有效的。并且该方法工作时不需要任何底层 DNN 系统的知识，因此可以潜在地应用于广泛的系统。与现有的防御策略相比，它具有合理的可扩展性和可靠性。

3.4 面向特定编程语言的应用

Mirshokraie 等人[19]提出了一组 JavaScript 变异算子，以此来检测 JavaScript 中的常见错误(例如删除 this 关键字、更改 setTimeout 函数、将 undefined 替换为 null)。实验结果表明，这些算子在产生非等价变异体方面是有效的。

Delgado-Pérez 等人[20]对 C++的类级变异运算符进行了评估。根据得到的结果，他们提出了一个 C++变异工具——MuCPP，该工具通过使用 Clang API 遍历每个翻译单元的抽象语法树来生成变异，最终产生了值得信赖的效果。

Alberto 等人[21]研究了形式化模型的变异测试方法，他们介绍了一种将变异测试应用于 Circus 规范的方法，并对变异操作符进行了广泛研究。prahamontripong 等人[22]则设计并研究了一组针对 web 应用的变异运算符。

目前的许多变异测试工具主要适用于一些如 C 和 Java 的主流语言程序，由于这些工具的一个关键元素是变异运算符列表（它定义了修改源代码和生成变异的方式），这些变异运算符本质上是特定于编程语言的，因此能用于一些小众语言的变异测试工具并不多。目前，正有许多研究人员在研究一些针对小众语言的变异测试工具。

Arzu Behiye Tarımcı 等人[23]针对 PL/SQL (Procedural Language/Structured Query Language)，这一业界采用的但很少被研究关注的动态编程语言，设计了一种变异测试工具，muPLSQL，来帮助变异的生成和测试执行的自动化。具体的，他们提出了 44 个适用于 PL/SQL 的变异运算符，其中 17 个用于 PL，21 个用于 SQL，6 个专门用于 PL/SQL。同时还将此工具设计为可扩展的，用于合并新的变异运算符。为了评估该工具和变异测试的适用性和有用性，本实验还进行了一个工业案例研究，使用 muPLSQL 对业务支持软件系统进行变异测试和分析。研究中使用了系统的 19 个对象，共包含了 8206 行 PL/SQL 代码。最终 muPLSQL 总共产生了 5939 个变异体，存活变异体数量为 680，PL/SQL 特异性变异算子产生了 320 个变异体，其中 46 个在测试执行中存活。对活变种的手动检查发现，有 112 个缺失的测试场

景和数据验证应纳入现有测试套件。此外，在检查过程中发现了 8 个源代码错误。

Lorena Gutiérrez-Madroñal 等人[24]将进化变异测试（EMT）应用于 Esper EPL 编程语言。测试这种编程语言的重要性是不可低估的，它非常适合事件驱动的物联网系统。考虑到许多物联网系统的生存或死亡取决于其对事件的快速反应，测试系统能否根据需求触发预期响应是至关重要的。为了将 EMT 与 Esper EPL 结合使用，使用 GAmera 工具分析了该技术的行为。GAmera 是一个包含应用 EMT 的遗传算法的工具。由于遗传算法的实现，该算法可以在不同的编程语言中使用。开发了一个桥接器来连接涉及的系统:MuEPL 和 GAmera。根据这两种系统的特点实现了桥。这意味着由于 GAmera 的适应性，它可以与任何编程语言一起使用。

对于 NodeJS 后端运行环境的变异测试工具非常稀少，但是随着 NPM 的使用者越来越多、其上的模块也越来越多，一个模块上的 bug 可能会导致巨大的影响。Rodríguez-Baquero 等人[25]开发了面向 NodeJS 的变异测试工具，并将其用到了 NPM 最热门的且自带测试用例的前 20 个模块中的 12 个。

Pablo G´omez-Abajo 等人[26]则提出了一个名为 WodelTest 的框架，来减少创建变异测试工具的工作量。为了达到这个目的，它遵循了一种模型驱动的方法，在高级描述中合成变异测试工具。该描述使用领域特定语言 Wodel 来定义和执行模型变异。Wodel 是独立于语言的，因为它允许为元模型定义的任何语言创建变异操作符。WodelTest 通过生成一个变异测试环境，将被测程序解析为一个模型，应用变异操作符，并根据生成的变异对测试套件进行评估，提供了丰富的变异测试度量集合。测试报告了基于为 Java 和 Atlas 转换语言创建变异测试工具的方法评估。

Daniel Fortunato 等人[27]则开始研究变异测试在量子计算领域可能的应用。尽管在经典计算领域，测试已经得到了广泛的研究，并提出了许多方法和工具，但量子程序（QPs）的此类方法仍处于起步阶段，大多数更容易在反直觉的量子编程中犯错误。其中一个问题就是，QPs 必然是概率性的，不可能在不中断执行或不影响其结果的情况下进行检查。因此，在量子计算领域，确保 QPs 的正确实现更加困难。

通过重点研究最流行的用于量子计算的开源全栈库，IBM 的量子信息软件包（Qiskit），Daniel Fortunato 等人利用语法等效门的概念，提出了五个为 QPs 定制的新颖的变异算子；以及一个新的基于 Python 的可以自动执行 QPs 的变异测试的工具集，名为 QMutPy，并且完成了其在 24 个真实 QPs 上的有效性和效率的实证评估；同时还详细讨论了如何扩展 QPs 测试套件以杀死更多的变异体，从而检测更多的 bug。实验结果表明，QMutPy 可以生成揭示错误的量子变异体，并暴露在实验中使用了真实 QPs 的测试套件中出现的一些问题。

3.5 其他应用

变异测试应用的一个具有挑战性的软件领域是移动应用。虽然移动设备和随之而来的应用程序已成为现代计算的支柱，但其开发中使用的框架和模式使测试和验证变得尤为困难。作为帮助衡量和确保移动测试实践有效性的一步，Kevin Moran 等人[28]引入了 MDroid+，一个用于 Android 应用程序变异测试的自动化框架。MDroid+包括来自十种经验衍生的 Android 故障类型的 38 个变异算子，自动化检测潜在变异位置和生成变异的过程，并通过可扩展的体系结构促进新操作符的添加和现有操作符的维护。对 MDroid+与其他流行的 Java 语言变异测试工具进行了评估，结果显示，MDroid+生成的不可编译和微不足道的变异更少。

Jose Miguel Rojas 等人[29]则为变异测试进行了游戏化，开发了一个在线游戏——CODE DEFENDERS，旨在让学习和应用变异测试变得有趣。在游戏中，玩家扮演攻击者的角色，目

的是创建最微妙的非等价变异体；或者是防御者的角色，目的是创建强大的测试来杀死这些变异体。这种方法的好处是多方面的:游戏可以发挥教育作用，让学习者以一种有趣的方式参与变异测试活动。有经验的玩家将创造出强大的测试套件，能够检测出其他玩家能够想到的最微妙的漏洞。

3. 总结

本综述总结研究了近十年变异测试应用的各类文献资料，对它们分别进行分类和分析，并将其研究的要点提取出来，以对各项研究进行区别与联系。通过这次调查我们发现，近年来，变异测试应用方面的研究十分活跃，论文期刊层出不穷，而且迭代十分迅速，不断有新技术、新工具涌现出来。由此可以看出人们对这方面的兴趣始终保持在高位，变异测试的价值依然不容小觑，有待进一步发掘。

变异测试首先可以用来对回归测试进行优化。众所周知，回归测试是软件迭代、版本变更过程中的一个重点，未经处理的回归测试的用例集可能会显得十分庞大臃肿，存在许多无效或效果并不显著的用例。为此人们提出测试用例优先级等方法，这些方法可以显著提高用例集的有效性，但是也出现了一些问题，比如现实中的故障数量可能数量不多，不足以支撑优先级排序，于是有人想到使用变异产生大量人工故障作为替代。为此，有团队研究了变异体和现实故障在 TCP 排序结果上是否有相关性，并指出相关性未必很强，因此这种应用可能有待优化，对特定程序要用特定变异算子才能提高有效性。有的团队就通过建模表明在满足一些条件时，基于变异的 TCP 在大多数情况下都会更有效。

在调试过程中，缺陷的精准定位和修复往往需要人们付出很大代价，费时又费力。为了自动定位缺陷，有基于频谱等热门技术，但是缺点明显。有研究者便将目光转向基于变异的缺陷定位。有研究表明，在变异评分很高的水平下，变异评分与缺陷揭示有很强的正相关性。许多研究者发现，变异体的死亡往往与故障相关，由此可以推导出可能的故障位置。通过对比发现，这种基于变异的缺陷定位技术比传统的基于频谱的技术准确率高上数倍。在缺陷修复方面，也有研究表明，变异可以应用于缺陷语句或测试数据从而实现自动修复，这些自动修复方法各有不引入新错误、速度加快、不产生过拟合等新特性。

AI 测试在近十年有着广泛的应用，并且与传统软件测试方法有很大的不同。由于 AI 的一大特点是其对训练数据的高依赖性，因而能生产大量变异体的变异测试就有可能应用于 AI 测试领域。尽管经典的变异测试可能难以直接应用于 AI 测试，但研究人员利用变异测试的思想，在包括深度学习测试、主动自助机学习测试、监测深度神经网络的对抗样本攻击等方向都实现了一定的成果。

变异测试在面向特定的编程语言如 JS, C/C++, Java 等主流语言时同样有广泛应用。其中，变异测试工具的关键在于特定于一门语言的变异算子。它可以基于语言的特性来定义修改源代码和生成变异的方式，用来检查语言中的常见语义语法错误和基本函数的使用情况。由于这种变异工具往往不能直接嫁接到其他语言上，目前研究的重点开始转向一些小众语言，为它们的测试提供一种优化方案。近来甚至有研究在量子程序上产生变异算子，以期检测出更多 bug。

变异测试还有一些其他应用。移动应用测试方面，有人提出了用于安卓应用程序的变异测试框架；还有研究者将变异测试游戏化，旨在提高变异测试的趣味性，寓教于乐。

变异测试已经被研究人员证实为一个有效的测试方法，在众多领域也得到了充分应用。我们有许多挑战需要面对，例如前文提到的突变体和真实故障的相关性问题。但是，变异测试新应用、新工具的出现与改进总是能够振奋人心，不断赋予这项技术以长足的生命力，未来的研究也一定会为变异测试创造出更多更宏大的应用场景。

参考文献

[1] Rudolf Ramler, Thomas Wetzlmaier, and Claus Klammer. 2017. An empirical study on the application of mutation testing for a safety-critical industrial software system. In Proceedings of the Symposium on Applied Computing (SAC '17). Association for Computing Machinery, New York, NY, USA, 1401–1408.

[2] Y. Lou, D. Hao, L. Zhang, Mutation-based test-case prioritization in software evolution, in: 26th IEEE International Symposium on Software Reliability Engineering, ISSRE 2015, Gaithersbury, MD, USA, November 2-5, 2015, 2015, pp. 46-57. doi:10.1109/ISSRE.2015.7381798.

[3] Luo Q, Moran K, Poshyvanyk D, et al. Assessing test case prioritization on real faults and mutants[C]//2018 IEEE international conference on software maintenance and evolution (ICSME). IEEE, 2018: 240-251.

[4] Shin D, Yoo S, Papadakis M, et al. Empirical evaluation of mutation-based test case prioritization techniques[J]. Software Testing, Verification and Reliability, 2019, 29(1-2): e1695.

[5] Chekam T T, Papadakis M, Le Traon Y, et al. An empirical study on mutation, statement and branch coverage fault revelation that avoids the unreliable clean program assumption[C]//2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE). IEEE, 2017: 597-608.

[6] Papadakis M, Le Traon Y. Effective fault localization via mutation analysis: A selective mutation approach[C]//Proceedings of the 29th annual ACM symposium on applied computing. 2014: 1293-1300.

[7] Papadakis M, Le Traon Y. Metallaxis-FL: mutation-based fault localization[J]. Software Testing, Verification and Reliability, 2015, 25(5-7): 605-628.

[8] Moon S, Kim Y, Kim M, et al. Ask the mutants: Mutating faulty programs for fault localization[C]//2014 IEEE Seventh International Conference on Software Testing, Verification and Validation. IEEE, 2014: 153-162.

[9] Arcaini P, Gargantini A, Riccobene E, et al. A novel use of equivalent mutants for static anomaly detection in software artifacts[J]. Information and Software Technology, 2017, 81: 52-64.

[10] Tan S H, Roychoudhury A. relifix: Automated repair of software regressions[C]//2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. IEEE, 2015, 1: 471-482.

[11] Ghanbari A, Zhang L. PraPR: Practical program repair via bytecode mutation[C]//2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE, 2019: 1118-1121.

[12] S. Chandra, E. Torlak, S. Barman, R. Bod´ık, Angelic debugging, in: Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011, 2011, pp. 121–130. doi:10.1145/1985793.1985811

[13] L. Klampfl, N. Chetouane and F. Wotawa, "Mutation Testing for Artificial Neural Networks: An Empirical Evaluation," 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS), 2020, pp. 356-365.

[14] Ma L, Zhang F, Sun J, et al. Deepmutation: Mutation testing of deep learning systems[C]//2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE). IEEE, 2018: 100-111.

[15] Q. Hu, L. Ma, X. Xie, B. Yu, Y. Liu and J. Zhao, "DeepMutation++: A Mutation Testing

Framework for Deep Learning Systems," 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2019, pp. 1158-1161.
[16] Aichernig B K, Tappler M. Efficient active automata learning via mutation testing[J]. Journal of Automated Reasoning, 2019, 63(4): 1103-1134.
[17] Wang J, Dong G, Sun J, et al. Adversarial sample detection for deep neural network through model mutation testing[C]//2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). IEEE, 2019: 1245-1256.
[18] Wang J, Sun J, Zhang P, et al. Detecting adversarial samples for deep neural networks through mutation testing[J]. arXiv preprint arXiv:1805.05010, 2018.
[19] S. Mirshokraie, A. Mesbah, K. Pattabiraman, Guided mutation testing for javascript web applications, IEEE Trans. Software Eng. 41 (5) (2015) 429–444.
[20] P. Delgado-Prez, S. Segura, I. Medina-Bulo, Assessment of c++ object oriented mutation operators: A selective mutation approach, Software Testing, Verification and Reliability 27 (4-5) (2017) n/a–n/a.
[21]A. D. B. Alberto, A. Cavalcanti, M. Gaudel, A. Simao, Formal mutation testing for circus, Information & Software Technology 81 (2017) 131–153. doi:10.1016/j.infsof.2016.04.003..
[22] U. Praphamontripong, J. Offutt, L. Deng, J. Gu, An experimental evaluation of web mutation operators, in: Ninth IEEE International Conference on Software Testing, Verification and Validation Workshops, ICST Workshops 2016, Chicago, IL, USA, April 11-15, 2016, 2016, pp. 102–111.
doi:10.1109/ICSTW.2016.17.
[23] Tarımcı A B, Sözer H. Mutation testing of PL/SQL programs[J]. Journal of Systems and Software, 2022, 192: 111399.
[24] Gutiérrez-Madroñal L, García-Domínguez A, Medina-Bulo I. Evolutionary mutation testing for IoT with recorded and generated events[J]. Software: Practice and Experience, 2019, 49(4): 640-672.
[25] Rodríguez-Baquero, Diego, and Mario Linares-Vásquez. "Mutode: generic javascript and node. js mutation testing tool." Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2018.
[26] Gómez-Abajo P, Guerra E, Lara J, et al. Wodel-Test: a model-based framework for language-independent mutation testing[J]. Software and Systems Modeling, 2021, 20(3): 767-793.
[27] Fortunato D, Campos J, Abreu R. Mutation Testing of Quantum Programs: A Case Study With Qiskit[J]. IEEE Transactions on Quantum Engineering, 2022, 3: 1-17.
[28] Moran K, Tufano M, Bernal-Cárdenas C, et al. Mdroid+: A mutation testing framework for android[C]//2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion). IEEE, 2018: 33-36.
[29] Rojas J M, Fraser G. Code defenders: a mutation testing game[C]//2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW). IEEE, 2016: 162-167.