# 突变是测试实验的适当工具吗？

J.H. Andrews 西安
大略大学计算机科学系
加拿大，伦敦
andrews@csd.uwo.ca

L.C. BriandY.Labiche
软件质量工程实验室 卡尔顿大学系统与计算机工程系
加拿大，渥太华
{briand, labiche}@sce.carleton.ca

## ABSTRACT
测试技术的经验评估在软件测试研究中起着重要作用。一种常见的做法是手动或通过使用突变算子来检测故障。后者允许系统的、可重复的播种大量的故障；然而，我们不知道以这种方式获得的经验结果是否会导致有效的、有代表性的结论。本文基于一些具有全面的测试用例库和已知故障的程序来研究这一重要问题。结论是，根据目前可获得的数据，使用突变算子产生的结果是值得信赖的（生成的突变体与真实故障相似）。然而，突变体似乎与手工播种的故障不同，后者似乎比真实故障更难检测。

## 类别和主题描述符
D.2.5 [**测试和调试**]

## 一般条款
实验，验证。

## 关键词
真正的过失，手工播种的过失，突变体

## 1. 简介
实验是软件测试研究的一个重要组成部分。通常情况下，实验是用来确定两种或多种方法中哪种方法在执行某些测试相关活动时更有优势。例如，人们可能对比较几种用于推导测试用例的测试标准的故障检测效果感兴趣，并为此进行了实验。测试实验通常需要一组具有已知故障的主题程序。这些主题程序应该足够大，以符合实际情况，但又不至于大到使实验不可行。至于故障，一种技术处理给定故障的能力应该能够准确预测该技术在实际程序中的性能。


*ICSE'05*，2005年5月15-21日，美国密苏里州圣路易斯。Copyright 2005 ACM 1-58113-963-2/05/0005...$5.00.

测试实验设计中的一个问题是，具有真实故障的适当规模的真实程序很难找到，也很难进行适当的准备（例如，通过准备正确和故障的版本）。即使有实际故障的程序，这些故障的数量往往也不足以使实验结果达到统计学意义。因此，许多研究人员采取了在正确的程序中引入故障以产生故障版本的方法。

这些故障可以通过手工引入（例如，实验者可以请有经验的工程师来做），或者通过自动生成代码的变体。一般来说，我们认为自动生成的变体是对代码应用运算符的结果。以这种方式使用的运算符被称为突变运算符，产生的错误版本被称为突变体，一般技术被称为突变或突变体生成。

突变体生成的主要潜在优势是可以精确地描述突变算子，从而提供一个定义明确的、故障播种过程。这有助于研究人员复制他人的实验，这是良好的实验科学的必要条件。虽然可以说手工引入的故障更真实，但最终还是要靠主观判断来确定某个故障是否真实。突变体生成的另一个重要优势是，可以生成潜在的大量突变体，增加了所获结果的统计学意义。

然而，一个重要的问题仍然存在。我们如何知道检测（或"杀死"）突变体的能力是否是对实际性能的准确预测；也就是说，在评估测试技术的故障检测效果时，突变体的外部有效性是什么？对这个问题的回答将对我们如何进行测试实验，从而对实验结果的有效性产生重要影响。

本文的主要贡献是比较测试套件对手工播种的、自动生成的和真实世界的故障的检测能力。我们的主题程序是一组广泛使用的带有手工播种的故障的程序，以及一个广泛使用的带有真实故障的程序。我们使用突变测试文献中的一组标准突变算子从主题程序中生成突变体。我们的结果表明，生成的突变体与真正的故障相似，但与手播故障不同，而且手播故障比真正的故障更难检测。

## 2. 相关的工作

使用突变体来衡量测试套件充分性的想法最初由DeMillo等人[6]和Hamlet[12]提出，并由Offutt等人[20]进行了广泛的探讨。Offutt[17]显示了对突变测试基本前提之一的经验支持，即检测简单故障（如由突变引入的故障）的测试数据集将检测复杂故障，即几个简单故障的组合。

Frankl和Weiss[10]、Thévenod-Fosse等人[24]和Hutchins等人[14]，以及此后的许多研究人员都使用了有缺陷的程序变体进行了实验。Frankl和Weiss使用了9个Pascal程序，每个程序都有一个现有的故障，而Hutchins等人在使用的7个程序中手工播种了130个故障。Thévenod-Fosse等人使用突变算子在四个小型C程序中自动播种故障。一般来说，这些实验遵循的模式是生成一个大型的测试用例"测试池"，在测试池中的所有测试用例上运行所有的故障版本，观察哪些测试用例检测到了哪些故障，并使用这些数据来推断从测试池中抽取的特定测试套件的故障检测能力（例如，满足特定覆盖标准的测试套件）。

尽管突变体生成最初是作为测试策略的一部分提出的，但请注意，Thévenod-Fosse等人将其作为一种生成实验用错误版本的方法。其他这样做的研究者包括Kim等人[15]、Memon等人[16]、Andrews和Zhang[1]以及Briand和Labiche[2]。

最后，Chen等人[5]在他们的实验中同时使用了手工播种的故障和生成的突变体。他们指出，手工播种的故障只是可能的故障的一个子集，并提出了故障是否具有代表性的问题，但由于这不是他们研究的重点，他们没有进一步探讨。

最后，据我们所知，还没有任何经验性研究通过与真实故障上获得的结果相比较来直接评估突变体或手工播种的故障的使用。

## 3. 实验材料和程序

在这一节中，我们描述了我们所进行的实验、受试程序和与之相关的人工制品（错误的版本和测试案例），以及我们用来产生突变体的突变算子。

在本文的其余部分，我们使用手工播种的故障（例如，由有经验的工程师制作的）和真实故障这两个术语(例如，在软件开发过程中发现的），或者只是在没有歧义时的故障，而不是由突变算子产生的故障，后者被称为突变体。

表1中列出了主题程序的属性摘要。这里的"NLOC"是净代码行数，即除去注释后的非空白代码行。"#条件"是指C条件结构（if、while、case等）和二进制逻辑运算符（&&和||）的数量，作为代码复杂性的一个粗略指标。

### 3.1 实验的定义

为了实现导言中所述的目标，我们分析了测试套件的检测率。我们使用八个主题程序（表1），这些程序的故障和大量的测试用例池是可用的。请注意，这些测试池是可比较的，因为它们确保了几个结构标准得到满足（第3.2节）。然后，通过对每个主题程序的大型测试池的随机抽样，形成测试套件（第3.4节）。然后，我们创建这些程序的突变体版本（第3.3节），并在可用的故障版本和突变体上执行这些测试套件，记录被杀死的故障，以计算测试套件的故障检测率。这些比率在突变体和故障之间进行比较，以确定它们是否相似。由于每个测试套件可能产生不同的检测率，我们为每个主题程序获得了两个分布，分别为故障和突变体。这些分布的中心趋势是通过统计推理测试来比较的。因此，无效假设（$H_0$）被表述如下。故障集和突变体集之间的检测率没有差异。备选假设（$H_a$）是：故障和突变体之间的检测比率存在差异。必须根据现有的数据来解释不同的故障和突变体分布的差异，还需要调查不同主题程序的结果的可能差异。

### 3.2 主题方案

我们使用了一组用C语言编写的八个著名的主题程序（表1）。第一个是通常被称为Space的程序，由欧洲航天局开发，首次由Frankl等人用于测试策略评估目的[9, 25]。最后7个是所谓的西门子程序套件，带有手工播种的故障，首先由Hutchins等人[14]用来比较基于控制流和基于数据流的覆盖标准。所有八个程序的工件（包括故障版本和测试用例）后来都被其他研究人员修改和扩展，特别是Rothermel和Harrold[23]以及Graves等人[11]。我们选择这些程序是因为它们的成熟度。

**表1.课题项目描述**

| | 主题方案 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 准则 | 空间 | 打印机 | 打印机2 | 替换 | 时间表 | 时间表2 | Tcas | 托特信息 |
| NLOC | 5905 | 343 | 355 | 513 | 296 | 263 | 137 | 281 |
| # 条件 | 635 | 94 | 81 | 99 | 37 | 51 | 25 | 46 |
| 测试池大小（#测试案例） | 13585 | 4130 | 4115 | 5542 | 2650 | 2710 | 1608 | 1052 |
| 故障的数量（版本） | 38 | 7 | 10 | 32 | 9 | 10 | 41 | 23 |
| 编译的突变体的数量 | 11379 | 582 | 375 | 666 | 253 | 299 | 291 | 516 |

相关的人工制品，并由于其历史意义。Do等人[8]报告说，在过去十年中，有17篇引人注目的实验性软件工程论文使用了西门子套件和/或空间。

空间与38个故障版本（真正的故障）相关联，其中33个来自最初的Vokolos和Frankl的研究，5个是后来被其他研究人员发现的。每个错误版本对应于 "在测试和程序的操作使用过程中"被纠正的错误[25]。纠正了所有故障的 "黄金"版本被认为是产生了正确的输出。相比之下，西门子套件程序的错误版本是由
"十个不同的人手工制作的，他们大多对彼此的工作一无所知；他们的目标是产生尽可能真实的错误"[14]。

每个主题程序的测试池（所有测试用例的集合）是由最初的研究人员和后来的研究人员按照各种功能（黑箱）测试技术和结构测试覆盖标准构建的。Harder等人[13]给出了这些测试池建设的全面历史。西门子程序的测试池开发如下：黑盒技术类别划分[21]被用来创建第一套测试用例，然后根据几个结构性覆盖标准（所有语句、所有边、所有定义-
使用对）进行扩展。程序空间的测试池的构建遵循类似的程序，只是原始测试集是通过随机输入选择产生的。

## 3.3 突变操作符

为了生成目标程序的突变体，我们使用了Andrews和Zhang首次使用的突变体生成程序[1]来生成C语言编写的代码的突变体。为了从源文件中生成突变体，我们依次考虑了每一行代码，并分别应用了四类
"突变操作符"（只要可能）。换句话说，对一行代码有效应用突变算子，就会产生另一个突变体。这四类突变运算符是

- 用0、1、-1、((C)+1)或((C)-1)替换一个整数常数C。
- 用同一类别的另一个运算符替换一个算术、关系、逻辑、位逻辑、增减或算术赋值运算符。
- 在`if`或`while`语句中否定这个决定。
- 删除一个声明。

前三类运算符是从Offutt等人的研究[18]中提取出来的，他们研究的是如何确定一组 "足够的"突变运算符，也就是说，一组运算符S，使得由S形成的突变体的测试套件倾向于杀死由非常广泛的运算符形成的突变体。这些算子经过调整，可以在C语言程序中工作，而不是原始研究中的Fortran。第四种运算符也出现在[18]中，它被添加进来，因为原始研究[1]中的主题程序（期间首次使用了突变体生成程序）包含大量的指针操作和字段赋值语句，这些语句不会受到任何充分突变运算符的影响。

大约有8.4%的突变体没有被编译出来。表1中列出了每个主题程序的突变体的编译数量。对于太空程序来说，产生的突变体太多，在测试套件中全部运行是不可行的。

因此，我们对产生的每10个$^{th}$
突变体进行了测试套件。因为每行产生的突变体数量不遵循任何模式，不会与每10个$^{th}$
突变体的选择相互影响（它只取决于该行的结构），这相当于从所有可能的突变体的均匀分布中随机选择了10%的突变体。此外，这也确保了整个源代码中都有错误的种子（而不是简单的几个函数/程序）。

## 3.4 分析程序

我们在本节中对我们所使用的分析程序进行了简要的描述和论证。更多细节将在下一节报告结果时介绍。

第一步是生成和编译突变体，并在整个测试池中运行所有突变体和错误版本。然后，所有没有被任何测试案例杀死的突变体都被认为是等同于原始程序。虽然这可能不是每个突变体的情况，但它被认为是一个足够好的近似值，而且在任何情况下，它是处理大量突变体时的唯一选择，因为自动识别等价突变体是一个不可判定的问题[3, 6, 19]。

对于每个主题程序，通过随机抽样（不重复）可用的测试库，随机形成5000个测试套件。我们需要产生大量的测试套件，以获得能够很好地接近基本理论分布的故障/突变体检测率的样本分布。如下文所述，大量的样本还可以提高统计测试的能力并促进其使用。随机选择而不是由覆盖标准（如结构）驱动的选择被认为提供了足够的可变性。要做的一个决定与测试套件的大小有关。我们希望这在我们的分析中是一个常数，以避免模糊我们正在分析的趋势：例如，更好的故障/突变体检测效率可能仅仅是由于测试集的大小，而不是由于故障和突变体之间（可能的）差异。测试集中太少的测试用例会导致较小的故障/突变体检测率，并使我们无法观察到差异，而测试集中太多的测试用例会减少变化。我们用不同的规模进行分析，以确定它将如何影响结果，范围从10到100个测试案例。由于在不同的测试套件规模下没有观察到重大的差异，而且100个测试套件的规模导致合理的故障/突变体检测率和良好的可变性（100个对应于最小的测试池的10%--表1），我们决定只报告100个测试套件的结果。

然后，我们确定池中的哪个测试案例检测到了哪个突变体和故障。接下来，我们计算了所有测试套件的故障检测率，绘制了每个主题程序的突变体和故障的检测率分布，然后比较它们的中心趋势。

概括地说，对于每个测试套件S，该程序产生了两个摘要数据。Dm(S)是S检测到的突变体数量，Df(S)是S检测到的故障数量。考虑到目标程序的非等价突变体数量Nm和非等价故障数量Nf，我们计算每个测试套件S的*突变检测率*Am（S）为Dm(S)/Nm，*故障检测率*Af(S)为Df(S)/Nf。

我们首先希望对每个科目的程序P进行如下的无效假设$H_0$（P）：即Am和Af比率的平均值

检验结果显示，P的值是相同的。为了做到这一点，我们采用了一个标准的统计检验：配对*t*检验[22]，原因将在第4节进一步详述[1]。如果我们拒绝$H_0$，因为故障分布和突变体分布之间的均值差异在统计学上和实践上都是显著的[2]，那么接下来的问题就是为什么？此外，如果不同程序的结果不一致，我们还需要确定最合理的解释是什么。有许多可能的原因，包括受试程序的特征、测试套件和故障播种方式的差异。所有受试程序的唯一共同点是我们生成突变体的方式。

为了解释我们的结果，并为了在第4节中进一步解释的原因，我们还研究了杀死每个故障和突变体的测试用例的百分比。对于每个突变体M和故障版本F，我们计算了K(M)(resp. K(F))，即杀死它的测试案例的数量。然后，对于每个突变体M，我们计算出杀死突变体的难易程度E(M)，即E(M)=K(M)/T，其中T是程序的测试用例的总数。(我们以类似的方式计算每个错误版本的E(F)。)然后，对于每一个程序，我们都得到了一个分布，其中每个观察值都对应着一个突变体或故障，我们可以比较这些分布在不同主题程序中的平均值。

在分析方面，我们首先进行了方差分析（ANOVA），以评估程序平均值之间差异的总体统计意义。如果有意义，我们接着用独立样本的*t*检验来比较每个（空间、西门子程序）对的分布平均值。为此，我们求助于邦费罗尼程序[22]，以确保在进行大量比较时有较低的I型错误风险：在我们的案例中，有28个比较[3]。因为我们在这里处理的是较小的样本（样本大小由数量决定）。

[1]当处理大型数据集时（作为经验法则，超过100个观测值），该测试对偏离匹配对*t*检验所依据的正态性假设（假设对之间的差异为正态分布）非常稳健，甚至在显示极端分布偏斜的情况下也是如此。这一点很重要，因为这种违反假设的情况在现实世界的数据集中很常见。还要注意的是，在我们的案例中，我们的观察值中不会有极端的异常值，因为我们要处理的是0和1之间的比率，而且我们将有5000个观察值可以使用。然而，为了安全起见，我们使用了另一种等效的非参数测试（Wilcoxon匹配对符号等级测试）来仔细检查结果。一般来说，在使用大样本时，配对*t*检验也特别适合，因为它考虑到了差异的大小，因此比同等的非参数检验更有力。

[2]在处理大样本时，即使是微小的均值差异也会导致统计学上的显著结果。然而，同样重要的是要确定一个统计学意义上的结果是否会有任何实际意义，也就是说，如果平均值的差异大到值得考虑。这就是通常所说的"实际"意义，有时也表示为"临床"意义。

[3]我们对每个（空间，西门子程序）对进行Am(S)和Af(S)的比较，因此产生了(7*8)/2=14个比较。同样地，比较E(M)和E(F)会产生14个额外的比较。

的突变体和故障，与上面的测试套件相反），我们需要用同等的非参数测试，即曼-惠特尼测试来检查不显著的结果[20, 22]。事实上，在小样本上，违反*t检验的*假设会导致第二类错误的增加，也就是错误地推断出平均值的差异是不显著的。

在我们所有的统计测试中，我们使用的显著性水平=0.05，因此意味着有5%的机会犯第一类错误，或者换句话说，有5%的风险将一个差异认定为统计学上的显著性，而实际上它并不显著。

## 3.5 对有效性的威胁

本节讨论了对我们实验有效性的不同类型的威胁，按照优先级递减的顺序：*内部*、*外部*、*构造*和*结论有效性*[4, 26]。

与内部有效性有关的一个问题是由于我们没有经过任何修改就重复使用了在以前的实验中被广泛使用的程序。我们所使用的程序、测试池和故障并不是根据任何特定的标准来选择的，除了它们是精心准备的和具有历史意义的。然而，我们不能保证测试池具有相同的检测能力和覆盖范围，尽管关于它们的文献报道似乎表明它们是以类似的方式产生的，并且相当全面，因为它们涵盖了一些主要的结构覆盖标准，正如第3.2节中所报告的那样。

我们对太空有真正的故障，但对其他的故障，我们只是由有经验的工程师手工播种的"现实的"故障。我们也不能指望这些程序具有相似的复杂性，不管我们如何定义和衡量它。实际上，它们的规模（表1）和控制流的复杂程度差别很大。预计我们的研究结果也会因所选择的突变算子而不同；如第3.3节所述，我们使用的突变算子是根据现有文献选择的，以便成为一个最小但足够的集合。

另一个可能的威胁涉及到我们对所有程序的唯一测试套件大小的报告，无论其复杂性如何，以及从测试池中随机选择的测试案例。然而，100的规模加上随机选择被认为是足够的（足够高的检测率和足够多的测试套件的可变性）。此外，还使用了其他的测试套件规模，但没有导致不同的结果，所以这里没有报告。

外部有效性与我们将实验结果推广到工业实践的能力有关。虽然我们只有程序空间的真实故障，但实验中使用的仪器是真实的工业程序，有真实的故障或由有经验的工程师手动播种的故障。尽管如此，由于只使用了一个有真实故障的程序，在其他已发现真实故障的主题程序上重复这项研究将非常重要。

结构有效性涉及到我们定义测量的方式，以及它是否测量了我们真正想要捕获的属性：测试集的检测能力和故障的可检测性。这一点在上面讨论测试集的大小和它们的构造时已经详细说明了。

结论的有效性与受试者的选择、数据收集、测量的可靠性和统计检验的有效性有关。这些问题在实验的设计中已经得到了解决。

**表2.描述性统计 - 突变体和故障的检测率（测试套件大小100）。**

| 主题方案 | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 统计数据 c | 空间 e | 替换e | 打印机 | 打印机2 | 时间安排 e | 附表2 | Tcas | 总计 |
| 变种人--Am(S) | | | | | | | | |
| 中位数 | 0.75 | 0.93 | 0.98 | 0.99 | 0.96 | 0.96 | 0.91 | 0.99 |
| 平均值 | 0.75 | 0.93 | 0.97 | 0.99 | 0.96 | 0.96 | 0.90 | 0.99 |
| 90% | 0.78 | 0.95 | 0.99 | 0.99 | 0.98 | 0.97 | 0.94 | 0.99 |
| 75% | 0.77 | 0.94 | 0.98 | 0.99 | 0.97 | 0.97 | 0.93 | 0.99 |
| 25% | 0.74 | 0.93 | 0.97 | 0.98 | 0.94 | 0.95 | 0.89 | 0.98 |
| 10% | 0.72 | 0.92 | 0.96 | 0.98 | 0.93 | 0.95 | 0.86 | 0.98 |
| 闵行区 | 0.65 | 0.88 | 0.94 | 0.93 | 0.91 | 0.91 | 0.77 | 0.94 |
| 最大 | 0.82 | 0.98 | 1 | 1 | 0.99 | 0.99 | 0.97 | 1 |
| 故障--Af(S) | | | | | | | | |
| 中位数 | 0.76 | 0.68 | 0.57 | 0.90 | 0.67 | 0.67 | 0.76 | 0.65 |
| 平均值 | 0.77 | 0.67 | 0.63 | 0.93 | 0.66 | 0.63 | 0.79 | 0.89 |
| 90% | 0.82 | 0.77 | 0.86 | 1 | 0.78 | 0.78 | 1 | 0.96 |
| 75% | 0.79 | 0.74 | 0.71 | 1 | 0.78 | 0.78 | 0.95 | 0.91 |
| 25% | 0.74 | 0.61 | 0.57 | 0.90 | 0.56 | 0.56 | 0.68 | 0.87 |
| 10% | 0.71 | 0.55 | 0.29 | 0.80 | 0.44 | 0.44 | 0.61 | 0.83 |
| 闵行区 | 0.53 | 0.35 | 0.14 | 0.60 | 0.33 | 0 | 0.34 | 0.65 |
| 最大 | 0.97 | 0.93 | 1 | 1 | 1 | 1 | 1 | 1 |

尽管最好是所有程序都有手选和真实故障。

我们的分析和解释必须考虑到上述所有因素，以确保所有合理的解释都得到考虑。

## 4. 分析结果

按照第3.4节所述的程序，我们首先比较了测试套件在突变体和故障方面的检测率（第4.1节）。在第4.2节中，我们确定了可能的现象，这些现象可以解释第4.2节中观察到的趋势。4.1.为了研究最合理的解释，我们随后比较了不同程序中杀死故障和突变体的测试用例的百分比，从而考察它们的*可检测性*，即它们的检测难度（第4.3和4.4节）。在本节的最后，我们讨论了我们的结果的含义（第4.5节）。

### 4.1 比较突变体和故障的检测分布情况

我们分析的第一步是比较每个主题程序的突变体和故障检测率*Am*和*Af*的分布情况。目的是确定它们之间是否存在统计学上和实践上的重大差异。我们在表2中提供了每个主题程序的这些分布的描述性统计（有两位数的精度）。请注意，这些是我们在选定的测试套件时得到的结果

大小为100。其他测试套件大小（10、20、50）也得到了类似的结果，因此在此不作报告。

从表2中，我们可以很容易地观察到，一般来说，突变体往往比故障更容易发现。这一趋势的唯一例外是空间，在那里可以观察到突变体和故障分布之间没有实际的显著差异（例如，它们的中位数分别为0.75和0.76）。虽然由于空间的限制，这里不能显示所有的分布，但图1和图2显示了空间和替换程序的分布直方图，以及标准的量化箱形图，以可视化分布的主要描述性统计。箱形图显示了中位数、平均值、各种量级（1%、10%、25%、75%、90%、99%）以及最小和最大。

我们现在要检查差异是否有统计学意义。为了做到这一点，我们进行了配对*t*检验[7, 22]。回顾一下，在这些分布中，我们比较了5000个测试套件的突变体（Am）和故障（Af）检测率。在每个分布中，观察结果很可能是相关的，因为它们是成对的，也就是说，*Am*和*Af*分布中的一个观察结果对应于同一个测试套件。这就是为什么需要这样一个配对测试，而不是独立样本测试。

表3显示了每个主题程序的*t检验*结果，即成对的故障/突变体检测比率的平均差异、*t*比率和*p*值（这是一个双向检验，因为我们事先不能确定差异的方向）。结果显示，所有的差异都有统计学意义，因为所有的*P*值都接近或等于0。

尽管平均差异为1%，但这并不具有实际意义，因为任何测试技术评估都不会因为如此小的差异而导致不同的结论。对于其余的主题项目，平均差异从6%到34%不等，平均为22%。与"空间"相比，这具有实际意义，因为根据这样的范围，如果一个人使用突变体来评估一种测试技术，它在检测故障方面可能会比使用种子故障看起来更有效。

然而，重要的是要记住，只有"太空"的故障是在测试和运行中检测到的真实故障。因此，在太空和其他项目之间观察到的差异的一个合理的原因是，后者的种子故障比太空的真实故障更难检测。如果这被认为是最合理的原因，那么对太空故障的测试结果就会更加可信，并且应该只根据这个项目的结果来得出结论。

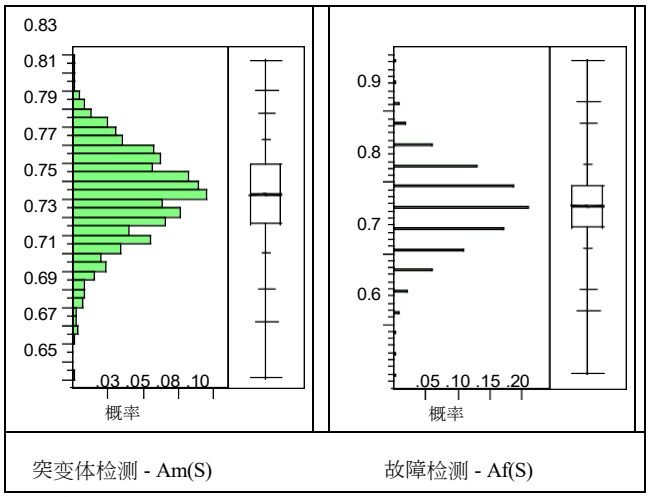


**图1.主题程序空间。检测率的分布 - 测试套件大小=100**

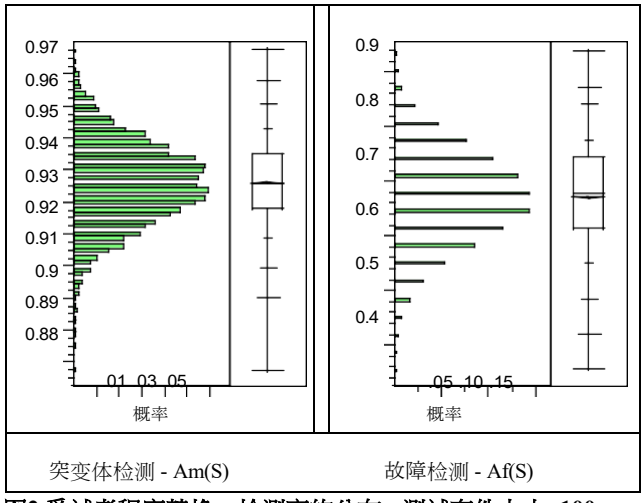


**图2.受试者程序替换。检测率的分布 - 测试套件大小=100**

**表3.匹配对的*t*检验结果 - 测试套件大小=100**

| | 匹配的配对结果 | | |
|---|---|---|---|
| 主题方案 | 平均值 琥珀(S)-琥珀(S) | *t*-比率 | *P*值 |
| 空间 | 0.014 | 16.87 | < 0.0001 |
| 替换 | -0.266 | -233.96 | 0.0000 |
| 打印机 | -0.344 | -158.2 | 0.0000 |
| 打印机2 | -0.061 | -59.39 | 0.0000 |
| 时间表 | -0.298 | -161.33 | 0.0000 |
| 时间表2 | -0.327 | -152.19 | 0.0000 |
| Tcas | -0.1128 | -57.56 | 0.0000 |
| 托特信息 | -0.1037 | -145.78 | 0.0000 |

## 4.2 其他解释

除了真实故障和种子故障的相对可检测性外，还有几个因素可以解释上一节中的数据。测试池、程序的内部特征（如规模）、与测试池和程序规模相比的测试规模套件，以及突变过程。重要的是要依次考虑每一个因素，以评估我们的第一个解释是否确实是最合理的。

测试池可以包含平均来说具有不同检测能力的测试案例。然而，关于这些程序的文献[13]表明，这些池子是按照类似的过程和相同的结构标准形成的：所有节点、所有边、所有定义-使用对。此外，为了解释我们的结果，这将意味着平均测试案例的检测能力将取决于是否考虑突变体或故障，因为与西门子的程序相比，Space对故障的检测率排名中等，对突变体的检测率较低。

另一种可能性是，与测试池的大小或程序的大小相比，*Am*和*Af*之间的差异随着测试套件的增加而增加。(Space比其他程序要大得多，而且有更大的测试池规模)。如果是这样的话，那么随着测试套件规模的减少，我们应该看到西门子程序的Am-Af更小。然而，结果显示，对于10、20和50的较小的测试套件规模是相似的：Space的mean(Am-Af)保持相似，对于其他程序，随着我们使用较小的测试套件规模，甚至有增加的趋势。附录中的图表比较了Space和Replace的四种测试套件规模的*Am*和*Af*；其他西门子程序的图表也类似。

突变过程可能在某种程度上有偏差，所以太空中的突变体比西门子程序中的更难杀死。然而，我们对太空采用了完全相同的突变过程，没有明确的理由说明结果会有不同。因此，空间比西门子程序大得多的事实更有可能解释较低的突变检测率，因为众所周知，较大的程序更难测试。然而，这种解释在我们拥有的少量受试程序中既不能确认也不能证伪，因为所有的西门子程序都是小的，而且大小相似，我们没有在连续的大小范围内的观察，直到空间。

Space程序的故障可检测性高于Siemens程序，这是我们上面已经讨论过的因素，也是最后要考虑的因素。为了进一步研究这是否是最合理的解释，我们接下来看一下故障和突变体的可检测性，即它们被从各程序的测试池中随机抽取的测试用例检测到的可能性。

假设西门子手工播种的故障比真实的空间故障更难检测，并且这不是由于不同测试池中测试用例检测能力的变化造成的，我们应该观察到以下趋势：（1）空间故障对于任何单独的测试用例来说应该比西门子故障更容易检测，（2）空间突变体对于任何单独的测试用例来说应该不比西门子程序突变体更容易杀死。我们在接下来的两节中依次探讨这些预测。

## 4.3 比较不同程序的故障可探测性

图3显示了每个主题程序的比率E(F)的分布情况，即检测每个故障F的难易程度（第3.4节）。Y轴是每个故障的计算比率，而X轴是简单的主题程序。X轴上每个主题程序的水平范围与它的相对故障样本量成正比。例如，我们可以看到，Space和Replace的样本量较大，而Schedule和Schedule2的样本量则相当小（表1）。这些不平等的样本量需要在我们的分析中加以考虑。此外，该图显示了每个程序的观察结果、总体平均数（跨程序的水平线）和每个程序的具体平均数（跨每个钻石的线）以及它们的平均数的95%置信区间（每个钻石的垂直跨度）。

图3清楚地表明，空间比率往往高于其他程序，正如我们所预期的。单向方差分析（ANOVA）[7，22]显示，总体而言，各程序均值存在统计学上的显著差异（F检验，*P*值<0.0001）。西门子程序的E(F)均值和太空故障的E(F)均值分别为0.035和0.14，从而清楚地显示出显著的差异。

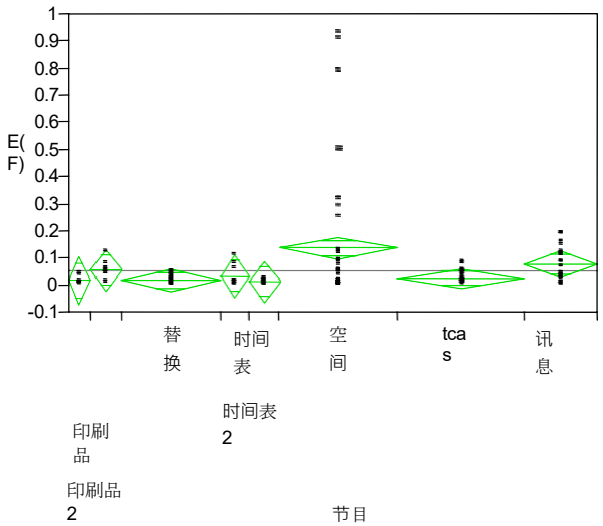


**图3.故障检测的难易程度分布 - E(F)**

将每个西门子程序与Space进行进一步的统计测试（使用Bonferroni程序[4]，用于使用*t*检验的平均值的多重比较[22]）表明，Space只与Tcas和Replace有显著差异（at = 0.05），尽管Space与所有其他程序的差异在图形上是可见的[5]。其他西门子程序缺乏统计学意义的一个可能原因是我们处理的是小样本，因为故障的数量从7个（Printtokens）到23个（Totinfo）不等。这也可能是由于Bonferroni程序倾向于保守的原因[7]。这增加了第二类错误的概率，并使合法的重要结果很可能无法被发现。换句话说，它保证了第一类错误率*最多是*在我们的例子中是0.05），但在现实中可能要少得多。

## 4.4 比较不同程序的突变体的可检测性

图4显示了E(M)的分布，即杀死每个突变体的测试用例的比率，对于每个主题程序（第3.4节）。它的结构与图3相同，只是它看的是突变体而不是故障。

与上一节中的故障一样，方差分析显示，不同程序的差异总体上具有统计学意义（F-检验，*P*值<0.0001）。

当再次使用Bonferroni程序和*t*检验更仔细地观察每个程序对时，我们发现空间的平均数明显低于所有程序，但Tcas除外（at = 0.05），证实了图4中可见的情况。因此，我们可以得出结论，不仅突变体不容易被杀死，而且个别测试案例显示，与西门子程序相比，在Space和Tcas中检测突变体的难度较小。对于太空，如上所述，这可能是由于这个程序的规模大得多（见表1）。

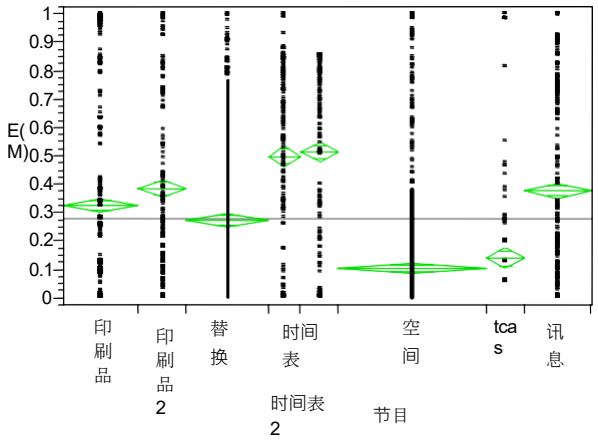


**图4.突变体检测的难易程度分布 - E(M)**

[4]这就要求用比率除以平均比较的数量作为显著性阈值，在我们的例子中是28。

[5]相等的非参数曼-惠特尼测试[7，22]也能 得到同样的结果。其他的多重平均数比较程序，如Tukey's方法，不能使用，因为我们在各程序中的样本量不相等[22]。

这些突变体和故障的反向趋势支持了我们的解释，即不同程序之间的差异不是由于不同的测试案例检测能力造成的，因为这将显示出突变体*和*故障的一致趋势，即空间测试案例检测到更多的突变体和故障。因此，图3和图4及其相关的统计测试结果支持我们的猜想，即对第4.1节中的趋势最合理的解释是，空间的故障往往更容易检测，因为它们是真正的故障，而不是像西门子程序那样由人设计的故障。

## 4.5 讨论

根据上述结果，我们可以推断，测试套件突变体检测率与Space中的故障检测率相当，而对其他程序来说却有很大不同，这是由于在其他程序中播种的故障在检测难度上可能不具有代表性。毕竟，它们不是真正的故障，而且人类可能很难衡量种子故障的可探测性。

首次报告这些程序的论文[14]实际上包含了这个结论的佐证。Hutchins等人明确指出，从最初由开发者提出的大量故障中，被350个或更多的测试用例（来自他们的原始测试池）检测到的故障被丢弃[6]
。这支持了我们上面的猜想，即Space的结果应该被赋予更多的可信度。如果是这样的话，那么我们可以得出结论，基于这里提出的突变算子的突变体，确实提供了代表真实故障的测试有效性结果。

这也意味着任何关于人种故障的研究都应该被仔细解释，因为我们看到，通过使用这里的例子程序--
这些程序在以前的研究中被广泛使用[8]--
人们在评估测试技术的故障检测率时将获得保守的结果。换句话说，这将导致低估测试技术的有效性。在衡量不同测试技术的成本效益时，这一点尤其重要。如果技术A比技术B能检测出更多难以检测的西门子故障，但A比B更昂贵，我们实际上可能没有证明A的相对成本效益。

在一些实验环境中，我们可能希望集中在难以检测的故障上，因为这些故障是程序员最难处理的。在这方面，西门子套件中播种的故障比我们在这里使用的全套突变体要好。然而，我们认为，研究人员应该有可能以选择困难的西门子故障的同样方式来选择困难的突变体，即通过排除那些被大量测试案例杀死的突变体。如果研究人员这样做了，那么他们就有可能以精确的方式描述整个突变体的选择过程，从突变算子到简单的突变体排除，从而允许其他研究人员进行复制。

[6]他们报告说，大约38%的故障是由于这个原因被丢弃的，18%的故障是由于太难检测这个补充原因而被丢弃的。

## 5. 结论

本文的主要贡献是对一个基本的假设进行了彻底的调查，这个假设是迄今为止许多实验性软件测试研究的基础。由人工（例如，由有经验的工程师）或突变操作者产生的故障是真实故障的代表。事实上，大多数实验结果都依赖于种子突变体[1, 2, 15, 16, 24]--使用突变算子或由有经验的开发人员选择的故障[10, 14]。此外，这个问题在未来的研究中不可能消失，因为真实的故障通常太少，不适合进行统计分析的实验。因此，即使对一个特定的主题程序有真实的故障，我们也经常被迫播种额外的故障。

因此，一个基本的问题是，突变体或手动播种的故障是否可能产生结果，例如在检测能力方面，是否能代表人们在真实故障上获得的结果。我们的分析表明，当使用精心选择的突变算子并去除等效突变体后，突变体可以很好地表明测试套件的故障检测能力。此外，它显示了使用人类选择的故障的危险性，因为在本文研究的系统中，它导致了对测试套件的故障检测能力的低估。由人类选择的故障的另一个问题与随着时间的推移建立一个关于测试技术效率的实验结果的必要性有关。要做到这一点，研究人员必须进行复制。当故障是根据主观的、未定义的过程来选择时，这是很难实现的。

我们还应该注意到，尽管我们在本文中的重点是表明我们生成的突变体并不比真正的故障更容易检测，但我们的数据也表明，它们并不比真正的故障更难检测到。这一事实支持了作为突变测试理论基础的
"有能力的程序员假设"；也就是说，程序员制造的故障会被杀死突变体的测试套件检测出来。

未来的研究当然需要复制我们在本文中报告的研究。我们这样做的目的是为了便于精确地复制我们的分析，从而将未来的结果与我们的结果进行比较。在面向对象的系统中进行类似的研究也是很重要的，因为这些系统在工业系统中占的份额越来越大，有合适的突变操作符。最后，由于本文的结果报告的是随机选择的同等规模的测试套件（尽管它们所建立的测试池满足结构覆盖率标准），所以研究根据不同标准（如代码覆盖率标准或运行状况标准）选择的测试套件的结果是否相同也很重要。

## 6. 鸣谢

非常感谢Gregg Rothermel和Hyunsook Do的有价值的讨论和建议，以及给我们提供的西门子和太空程序和所有相关的人工制品。也感谢所有多年来从事和改进这些主题程序和人工制品的研究人员。我们还要感谢Mike Sowka对本文草稿的审核。这项工作得到了加拿大研究主席（CRC）资助的部分支持。杰米

Andrews、Lionel Briand和Yvan
Labiche得到了NSERC业务资助的进一步支持。

## 7. 参考文献

[1] J. H. Andrews and Y. Zhang, "General Test Result Checking with Log File Analysis," *IEEE Transactions on Software Engineering*, vol. 29 (7), pp.634-648, 2003.

[2] L.Briand, Y. Labiche和Y. Wang, "使用模拟来经验性地调查测试覆盖率标准，" *Proc. IEEE/ACM国际软件工程会议*，爱丁堡，第86-95页，2004年5月。

[3] T.A. Budd和D. Angluin，"正确性的两个概念及其与测试的关系"，*Acta Informatica*，第18卷（1），第31-45页，1982。

[4] D.T. Campbell和J. C. Stanley, *Experimental and Quasi-Experimental Designs for Research*, Houghton Mifflin Company, 1990.

[5] W.Chen, R. H. Untch, G. Rothermel, S. Elbaum and J. von Ronne, "Can fault-exposure-potential estimates improve the fault detection abilities of test suites?", *Software Testing, Verification and Reliability*, vol. 12 (4), pp.197-218, 2002.

[6] R.A. DeMillo, R. J. Lipton and F. G. Sayward, "Hints on Test Data Selection:对实践中的程序员的帮助"，*IEEE计算机*，第11卷（4），第34-41页，1978。

[7] J.L. Devore, *Probability and Statistics for Engineering and the Sciences*, Duxbury Press, 5th Edition, 1999.

[8] H.Do, G. Rothermel and S. Elbaum, "对软件测试和回归测试技术的受控实验的基础设施支持，" 俄勒冈州立大学，科瓦利斯，俄勒冈州，美国，技术报告04-06-01，2004年1月。

[9] P.G. Frankl和O. Iakounenko，"测试有效性的进一步经验研究"，*Proc. 6th ACM SIGSOFT软件工程基础国际研讨会*，Orlando（美国佛罗里达州），第153-162页，1998年11月1-5日。

[10] P.G. Frankl和S. N. Weiss, "所有用途和所有边缘充分性标准的有效性的实验比较，" *Proc. 4th 测试、分析和验证研讨会*，纽约，第154-164页，1991年。

[11] T.L. Graves, M. J. Harrold, J. -M.Kim, A. Porter and G. Rothermel, "An Empirical Study of Regression Test Selection Techniques," *ACM Transactions on Software Engineering and Methodology*, vol. 10 (2), pp.184-208, 2001.

[12] R.G. Hamlet, "借助编译器测试程序"，*IEEE Transactions on Software Engineering*, vol. 3 (4), pp. 279-290, 1977.

[13] M.Harder, J. Mellen and M. D. Ernst, "Improving Test Suites via Operational Abstraction," *Proc. 25th International Conference on Software Engineering*, Portland, OR, USA, pp. 60-71, May, 2003.

[14] M.Hutchins, H. Froster, T. Goradia和T. Ostrand, "基于数据流和控制流的测试充分性标准的有效性实验"，*Proc. 16th IEEE国际软件工程会议*，Sorrento（意大利），第191-200页，1994年5月16-21日。

[15] S.Kim, J. A. Clark and J. A. McDermid, "Investigating the Effectiveness of Object-Oriented Testing Strategies with the Mutation Method," *Software Testing, Verification and Reliability*, vol. 11 (3), pp.207-225, 2001.

[16] A.M. Memon, I. Banerjee and A. Nagarajan, "What Test Oracle Should I use for Effective GUI Testing?" *Proc. IEEE International Conference on Automated Software Engineering (ASE'03)*, Montreal, Quebec, Canada, pp.164-173, October, 2003.

[17] A.J. Offutt，"软件测试耦合效应的调查"，*ACM软件工程和方法学的互动*，第1（1）卷，第3-18页，1992年。

[18] A.J. Offutt, A. Lee, G. Rothermel, R. H. Untch and C. Zapf, "An Experimental Determination of Sufficient Mutation Operators," *ACM Transactions on Software Engineering and Methodology*, vol. 5 (2), pp.99-118, 1996。

[19] A.J. Offutt和J. Pan, "Detecting Equivalent Mutants and the Feasible Path Problem," *Software Testing, Verification, and Reliability*, vol. 7 (3), pp.

[20] A.J. Offutt和R. H. Untch, "Mutation 2000:Uniting the Orthogonal," *Proc.Mutation*, San Jose, CA, USA, pp.45-55, October, 2000.

[21] J. Ostrand和M. J. Balcer，"指定和生成功能测试的类别-分区方法"，*ACM通讯*，第31卷（6），第676-686页。
1988.

[22] J.Rice, *Mathematical Statistics and Data Analysis*, Duxbury press, 2nd Edition, 1995.

[23] G. Rothermel和M. J. Harrold，"安全回归测试选择技术的经验研究"，*IEEE Trans. on Software Engineering*，第24卷（6），第401-419页，1998。

[24] P.Thévenod-Fosse, H. Waeselynck and Y. Crouzet, "An experimental study on software structural testing: deterministic versus random input generation," *Proc. 21st International Symposium on Fault-Tolerant Computing*, Montreal, Canada, pp. 410-417, June, 1991.

[25] F.I. Vokolos和P. G. Frankl, "文本差异回归测试技术的经验评估，" *Proc. IEEE国际软件维护会议*，美国马里兰州贝塞斯达，第44-53页，1998年3月。

[26] C.Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell and A. Wesslen, *Experimentation in Software Engineering - An Introduction*, Kluwer, 2000.

## 8. 附录

图5和图6显示了*Am*和*Af*的平均值如何作为测试套件大小的函数而变化。我们可以观察到，无论规模大小，*Af*和*Am*对Space来说都非常相似。然而，对于Replace来说，和其他程序一样，差异很大，并且随着测试套件规模的减少而趋于增长。

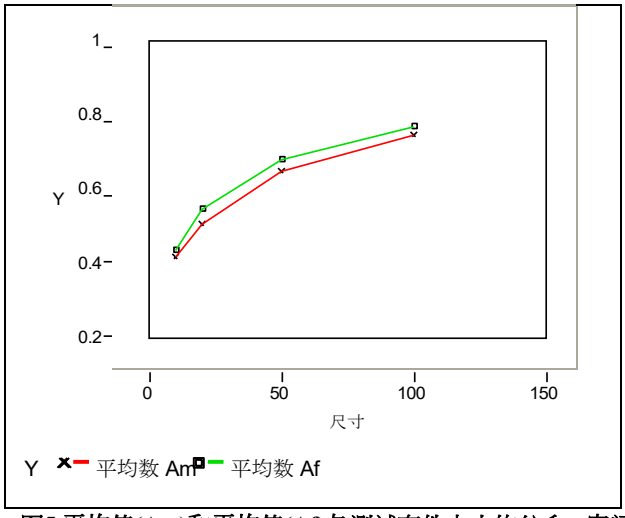


**图5.平均值(Am)和平均值(Af)与测试套件大小的关系 - 空间**

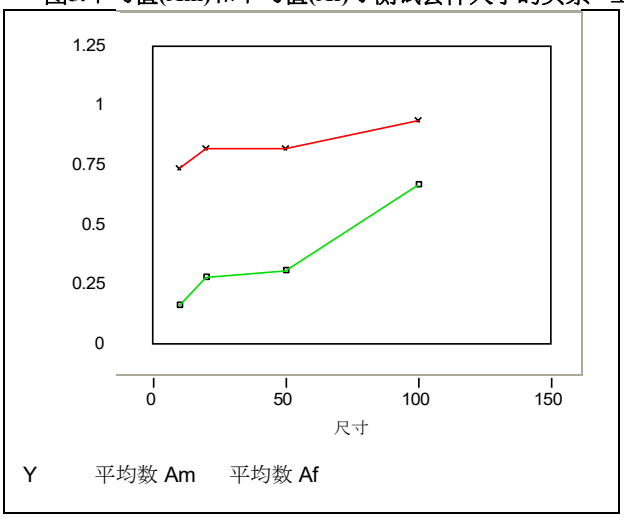


**图6.平均值(Am)和平均值(Af)与测试套件大小的关系 - 替换**