基于统计测试的净室策略研究及应用

国防科技大学计算机学院　林星、徐锡山
厦门大学软件学院　林凡

摘 要:

净室（cleanroom）软件工程是一种应用数学和统计学方法以经济有效的方式开发高质量软件的工程技术。净室软件提出的基于使用模型的统计测试，是统计方法在软件测试上的一种应用。借助马可夫模型分析，作者在厦门某银行的凭证管理系统的设计中，采用的 Discriminant 值对使用链和测试链进行定量分析，结果表明该算子具有良好的验证效率，证明了测试用例的充分性。

关键词： 净室 使用模型 统计测试 马可夫模型 Discriminant 值

1. 引言

净室软件提出的基于使用模型的统计测试，是统计方法在软件测试上的一种应用。首先，要构造出被测软件的使用模型，它描述了被测软件的使用方式；然后根据使用模型随机产生测试用例并实施测试；最后，按照数学和统计学模型对测试结果进行分析，获取软件的质量度量。测试用例设计是从初态开始经过若干个中间状态到达终态的状态和边的序列。在辅助测试工具的帮助下可以轻而易举地产生大量的测试用例。

但是，问题的关键在于测试样本空间的控制，以及达到充分测试数量的阈制。统计测试通过比较测试过程中的使用链和测试链的差异程度来解决这个问题。本文通过马可夫模型和 Discriminant 值分析厦门某银行凭证系统的使用模型，有效解决了测试链覆盖率和测试效率问题，证明采用统计测试等净室软件工程手段，可以有效的提高软件自动化测试的覆盖率的同时，提高测试效率。

2. 净室软件工程理论

净室软件工程包含如下三项关键技术[1]：基于统计方法的过程控制下的增量式开发、基于函数的规范、设计和验证以及统计测试和软件证明。这些技术可以分开或合起来使用，它们的采用可以有效的改进软件生产过程。

2.1. 函数理论

净室软件开发方法基于数学中的函数理论[1]。一个函数定义了一个从定义域到值域的映射，定义域中的每一个元素都可在值域中找到唯一的元素与之对应。一个特定的程序也描述了一个从定义域（程序所有可能输入序列的集合）到值域（所有对应于输入的输出集合）的映射。因此，程序的规范就相当于函数的规范，描述了一个程序的输入到输出的对应关系[6]。

一个定义明确的函数包括：完备性（completeness）、一致性（consistency）和正确性（correctness）。因为一个程序规范描述了一个函数，所以它必须是完备的、一致的和正确的[1]。

2.2. 统计理论

净室测试方法基于统计学[1]。在在软件测试中，源于软件的特殊性，统计方法表现得更为重要。在净室软件工程中，用于采样的全体（population）是软件所有可能使用方式的全集，集合中的每个元素代表软件系统的一种可能的使用情况。由于实际上不可能对庞大的总体进行所有路径和所有条件的完全测试，因此必须利用统计学方法，通过抽样采样测试的样本，来度量软件的整体性能和正确性。

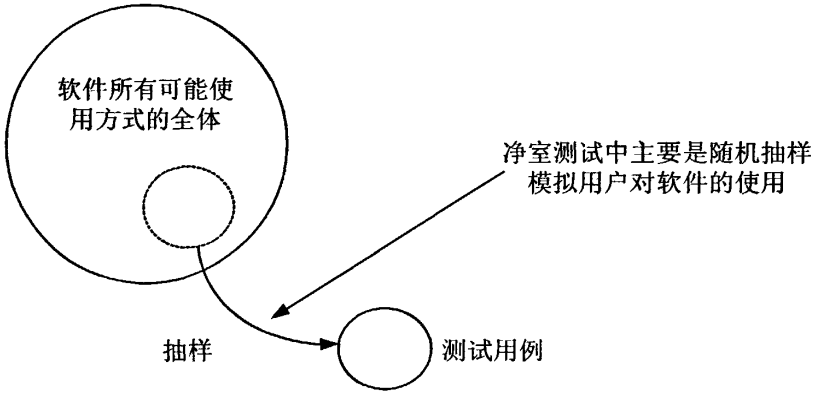

图 1 净室测试中的统计抽样

2.3. 正确性验证

在盒子结构的规范与设计中，黑盒被定义来记录所需的行为，状态盒是黑盒的细化，定义了所需的状态数据，明盒是状态盒的细化，定义了所需的处理过程。每一种盒子结构在小组开发评审时都要经过正确性验证。小组采用基于函数理论的推理来验证每一细化步骤相对于前一步骤的正确性。也就是说，开发小组确认在每一步中定义的激励-响应映射规则在后续步骤中得到保持。

明盒过程可能包含无限多的路径，这些路径不可能通过基于路径审查或软件测试来得到全部的检查。但正确性定理是基于验证单个控制结构而不是跟踪路径的，因为过程包括有限的控制结构，所以正确性定理将验证转变为有限情形的检查，并从逻辑上验证了所有可能的使用情况[1]。而一个可行而有效的验证过程使得开发组可以根据规范彻底验证软件的正确性。验证步骤对减少错误非常有效，这也是净室小组能够提高软件质量的主要因素。

3. 统计测试和软件证明

净室测试方法采用统计学的基本原理，当总体太大时必须进行抽样，用样本的性能推断总体的性能。

3.1. 使用模型定义

统计测试时首先要构造出被测软件的使用模型，它描述了被测软件的使用方式；然后根据使用模型随机产生测试用例并实施测试；最后，按照数学和统计学模型对测试结果进行分析，获取软件的质量度量，在统计测试中是软件的可靠性。统计测试和回归测试、基于结构的测试是兼容的，它们可作为净室统计测试的补充。大量的实践表明，基于使用模型的统计测试更经济有效，并且能获得软件的高可靠性[1]。

使用模型基于软件规范而不是代码，代表了系统使用中所有可能的事件及其发生概率。因此通过使用模型的建立可以在工程的早期阶段避免出现影响全局的软件规范上的缺陷。具体说来，基于使用模型的统计测试有如下主要优点[1]：使用模型是系统规范的外部视图，便于需求确认；可以辅助资源分配和测试计划的制定；可以借助使用模型自动生成测试用例；便于测试聚焦；有助于量化测试管理等。

3.2. 使用模型的表示和建立

在净室软件工程中，使用模型是伴随着顺序规范（Sequence Specification）的过程建立起来的。首先，列出软件所有可能的外部输入；然后，对每种长度的输入序列都用这些可能的输入进行组合，并去掉不可能的和等价的输入序列，直到没有新的输入序列产生为止；最后，这些完全不同的输入序列形成了使用模型的状态空间。状态间的转移概率是通过老版本软件或类似软件的使用记录来获得的，如果不存在类似的产品或没有使用记录，则由领域专家协商解决，也可以使每个状态的出边全都等概率转移。

使用模型可以用有向图、表格或矩阵的方式来描述，用有向图表示的优点是直观易懂，但通常只用于小型系统或大型系统的高端表示。用表格和矩阵表示的时候，行和列代表状态，表格（矩阵）的单元值代表了状态间的转移概率，这种方法的优点是比较容易描述大的系统，但不够直观。

通过 Markov 理论对其进行分析，这使得 Markov 链使用模型在净室软件开发的实际应用中具有明显的优势。以 Markov 链表示的使用模型由状态和边组成。状态表示软件使用过程中的内部环境，边表示状态间的转移关系。每条边都有一个激励输入与之对应，表明在当前状态下输入这种激励使软件转移到下一个状态。每条边都有一个转移概率，表示状态转移发生的可能性。特定状态的所有出边的转移概率之和应该为 1。

用 Markov 过程来描述软件的使用方式，任何下一个发生的事件只和当前的状态有关，不涉及历史信息。因此，软件的使用模型可以表示为有穷状态、离散参数的马尔可夫链，用马尔可夫理论来建立和分析使用模型。

每一个使用模型都有唯一的初态和终态。初态是使用模型的初始状态，它是每一次软件使用的开始。终态是使用模型的终止状态，它是软件每一次使用的终结。软件的每一次使用（或者说每一次操作）都从初态开始经过若干个中间状态，最后到达终态，这便形成了一个测试用例。

图 2 描述了一个简化后的银行凭证系统使用模型，它描述了软件使用的通常的情况，即：启动->使用->终止。其中节点代表状态，弧线代表边，每条边都标有激励和发生概率。状态 Invocation 是初态，状态 Termination 是终态。

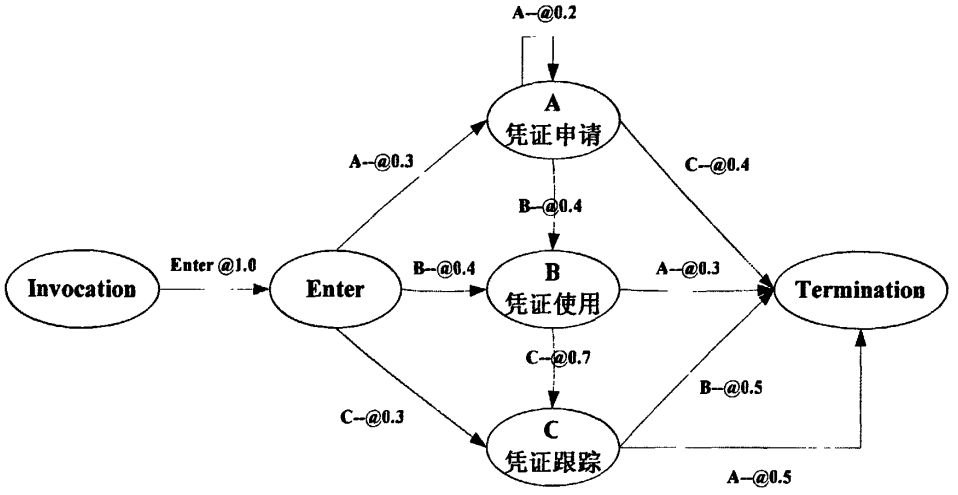

图 2 凭证系统的使用模型范例

转移矩阵 $P=[p_{i,j}]$，其中 $p_{i,j}$ 为从状态 i 转移到状态 j 的概率。图 2 所示使用模型的转移矩阵为：

$$P=\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.3 & 0.4 & 0.3 & 0 \\ 0 & 0 & 0.2 & 0.4 & 0 & 0.4 \\ 0 & 0 & 0 & 0 & 0.7 & 0.3 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

使用模型对应的转移矩阵

$$S=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0.3 & 0.4 & 0.3 \\ 0 & 0.2 & 0.4 & 0.4 \\ 0 & 0.3 & 0 & 0.7 \\ 0 & 0.5 & 0.5 & 0 \end{bmatrix}$$

模型的激励矩阵

其中，各行和各列都代表状态，从上到下（从左到右）的顺序为：Invocation、Enter、A、B、C、Termination。转移矩阵的起始状态删除后，可以获得对应的缩减矩阵，并根据模型的激励概率，获得激励矩阵。

激励矩阵 $S=[s_{i,j}]$，其中 $s_{i,j}$ 为在状态 i 时选择激励 j 的概率。对图 2 所示的使用模型来说，它的激励矩阵为右图所示。

其中，行代表状态，从上到下的顺序为：Invocation、Enter、A、B、C、Termination，列代表激励，从左到右的顺序为：enter、a、b、c。在本文后面部分我们都用 S 来标记激励矩阵。

另外，我们还假定初始状态的索引为 1，终止状态的索引为 n，使用模型的激励数为 s。因此，转移矩阵 P 和限制矩阵 $P|_k$（$1 \leqq k \leqq s$）都是 n×n 维矩阵，缩减的转移矩阵是（n−1）×（n−1）

维矩阵，激励矩阵 S 是（n－1）×s 维矩阵。E 代表单位矩阵，Q^{-1} 代表 Q 的逆矩阵，X_d 代表 X 的对角矩阵。

通过对使用模型进行静态分析可得出多种有关使用模型的静态参数。这些参数在软件的整个生命周期都可使用，有助于验证使用模型的正确性、更好的理解系统的使用方式以及辅助制定测试计划。主要的静态参数如下：

- ✧ 模型的复杂度。
- ✧ 测试用例输入序列的平均长度。
- ✧ 长时间运行中各状态的占有率。
- ✧ 单个测试用例中各状态的发生概率。
- ✧ 特定状态发生前的平均测试用例数。

4. 样本与总体

统计测试产生软件所有可能的使用的一个子集，并以这个子集所表现的性能为依据来考虑整体使用的性能。也就是说，用样本来描述总体。

4.1. 软件使用的随机特性

通常来说用户对软件的使用总会表现出一定的随机特性，从外部看来它表现在不同操作发生的概率各不相同，从内部看来则表现在一次使用中软件状态变化的不确定性。这种由于不同的输入处于不同的状态以及不同的出现频度，其表现出一定的统计特性，便可以借助使用模型是对软件形态进行精确刻划。

对输入序列长度没有限制的软件，理论上来说有无限多种不同的输入序列。比如一个软件如果有两种用户输入：a 和 b，那么可能的输入序列有：a，b，aa，ab，ba，bb，aaa，aab，aba……。

因此净室软件测试的焦点从发现错误转移到了软件质量证明。问题的关键在于如何描述使用总体以及如何抽取测试用例。抽取的测试用例子集必须最大程度的保留总体的特征，只有那样才能保证我们从测试环境得出的软件质量度量能够代表软件实际使用时的质量度量。

4.2. 测试充分性度量

测试的方法多种多样，关键在于：什么时候可以停止测试。作为一种主流的随机测试手段，统计测试的目标是模拟用户对软件的使用方式，以一种最有效的方式发现那些对软件可靠性影响最大的软件缺陷，并度量软件的可靠性。

因此，充分性判定方法就应该是度量测试使用和实际使用的差异程度，以确保我们从测试使用中计算出的可靠性能代表软件实际使用时的可靠性。基于这一点，统计测试中测试充分性度量主要集中在比较测试链和使用链的相似性，并以量化数据描述它们的相似程度。

在统计测试过程中会产生一条测试链和一条使用链。测试之初，我们把使用模型称为使用链，这是相对于测试过程中的测试链来说的。测试链是从使用链产生的，它把使用链各边所对应的转移概率替换为一个初值为 0 的计数器。随着测试的进行，测试链会越来越接近使用链，当它们之间的差异足够小的时候我们就认为测试充分可以停止测试。

使用模型是软件使用过程中软件形态的精确刻划。通过对使用模型的分析，可以获得多种有关

使用模型的静态参数。这些参数描述了软件使用的某些特性，可以用来更深入地了解软件的使用方式，更好地控制软件的开发和测试过程。

使用模型的静态参数主要分为四类：与模型相关的静态参数、与状态相关的静态参数、与边相关的静态参数和与激励相关的静态参数。

统计测试的充分性判定方法就应该是度量测试使用和实际使用的差异程度，以确保我们从测试使用中计算出的可靠性能代表软件实际使用时的可靠性。基于这一点，统计测试中测试充分性度量主要集中在比较测试环境和使用环境的差异，并定量的描述它们之间差异的大小。

在统计测试过程中会产生一条测试链和一条使用链。测试之初，我们把使用模型称为使用链，这是相对于测试过程中的测试链来说的。测试链是从使用链产生的，它把使用链各边所对应的转移概率替换为一个初值为 0 的计数器，随着测试的进行，每当一个测试用例经过该边时计数器就加 1，根据每一条边的计数器的值计算出该边的相对转移概率就形成了测试链。如果产生失效的话，则在测试链中添加一个失效状态，同时添加一条从失效源状态到失效状态的边和从失效状态到失效结束状态的边。在测试过程中，测试链代表了测试环境，使用链代表使用环境。随着测试的进行，测试链会越来越接近使用链，当它们之间的差异足够小的时候我们就认为测试充分可以停止测试了。测试链和使用链的差异程度可以用欧几里得距离或 Discriminant 值来进行度量，它们的计算方法和各自的优缺点如下。

5. 银行凭证系统的测试用例生成

在使用模型的帮助下，可以以手动或自动的方式产生测试用例。前面说过，测试用例是从初态开始经过若干个中间状态到达终态的状态和边的序列。产生测试用例时，从初态开始，在每一个状态都生成一个 0～1 间的随机数，根据这个数选择这个状态的一条出边，转移到下一个状态，直到到达终态。这样产生的测试用例是随机的，符合用户的使用习惯。下面就是一个根据凭证系统的使用模型（图 2）随机产生的测试用例：

表 1 根据图 2 使用模型产生的一个随机测试用例

Stimulus No.	Stimulus	Next State
1	ENTER	Enter
2	A	A
3	B	B
4	C	C
5	A	Termination

这些测试用例可直接应用于被测软件，或作为测试自动化工具的输入实施自动的软件测试。自动测试可以极大的提高软件测试的效率，根据不同的工具类型可能需要修改测试用例的格式或添加一些额外的信息，以满足测试自动化工具的需要。

5.1. 应用 Discriminant 值

我们通过 discriminant 值来比较测试链和使用链，它是两个随机过程似然度的期望值，在这里简记为 D(U,T)。其中 U 代表使用链，T 代表测试链，它的计算方法如下：

$$D(U,T)=\lim_{n\to\infty}\frac{1}{n}\left[\log\left(\frac{\Pr[x_0,x_1,\ldots x_n\mid U]}{\Pr[x_0,x_1,\ldots x_n\mid T]}\right)\right]$$

其中 $x_0, x_1, \ldots x_n$ 是通过使用链产生的输入序列，这个公式等价于

$$D(U,T) = \sum_{i=1}^{u} \pi_i \sum_{j=1}^{u} u_{i,j} \log\left(\frac{u_{i,j}}{t_{i,j}}\right)。$$

其中 $u_{i,j}$ 和 $t_{i,j}$ 分别为使用链和测试链中从状态 i 到状态 j 的转移概率，π_i 为状态 i 长时间运行中的占有率（Long Run Occupancy）即长时间运行中各状态所占有的比例。可以看到测试链和使用链越接近，D(U,T)就越接近于 0，当它小于一个特定值的时候，我们认为测试变得充分，可以停止测试了。它可以提供比较精确的结果，但我们可以看出它并不总是可计算的，在测试过程中只有在使用模型的所有边都覆盖以后它才有意义。因此产生了一种变体的计算方法，使得在任何情况下 D(U,T)都是可计算的：

$$\hat{D}(U,T) = \sum_{i=1}^{u} \pi_i \sum_{j=1}^{u} u_{i,j} \log\left(\frac{u_{i,j}}{\varepsilon - \varepsilon(\mathrm{sgn}(t_{i,j})) + t_{i,j}}\right)$$

其中，ε 为一个很小的正数；sgn(x)为符号函数，当 x=0 时 sgn(x)=0，当 x<0 时 sgn(x)=−1，当 x>0 时 sgn(x)=1。显然，在测试过程中当所有边都覆盖以后 D(U,T) = $\hat{D}(U,T)$。下表是图 2 使用模型在测试过程中的状态和边的覆盖率以及 discriminant 值。我们可以看到每一次失效都使 D(U,T)值明显增大，这使得我们不得不追加更多的测试用例才能达到预期的 discriminant 值。

表 2 凭证系统使用模型在测试过程中的状态和边的覆盖率及 discriminant 值

Script No.	Result	State Coverage	Arc Coverage	D(U,T)
1	Pass	0.6666667	0.27272728	3.5349884
2	Pass	0.8333333	0.54545456	2.5790153
3	Pass	1.0	0.8181818	0.50386137
4	Pass	1.0	0.90909094	0.29729524
5	Pass	1.0	0.90909094	0.2894273
6	Failed	1.0	0.90909094	0.35086015
7	Pass	1.0	0.90909094	0.32058263
8	Pass	1.0	0.90909094	0.33343184
9	Failed	1.0	0.90909094	0.3719081
10	Pass	1.0	0.90909094	0.38963783
11	Pass	1.0	0.90909094	0.38262

				13
12	Pass	1.0	0.90909094	0.33822182
13	Pass	1.0	1.0	0.08055764

5.2. 结论

由于欧几里得距离的不稳定，在本文中用 discriminant 值来度量测试的充分性。但它只是定量的描述测试链和使用链的差异程度，并不能告诉我们究竟它为多大时才可以停止测试。这个基准值必须由测试人员自己给出，它与使用模型的规模和特点有关，通常要依靠实践经验。但有一点可以肯定，停止测试的时候 discriminant 值随测试进行的变化很小，不能给测试过程添加新的信息，否则不能停止测试。另外，要决定是否可以停止测试还必须考虑软件的可靠性是否达到预期的指标。

通常来说，一个中等项目大概有 100－500 个状态上千条边，大点的项目有上千个状态上万条边，因此我们很难把这种项目的使用模型完全放在一个图里面，因此有必要引入子模型。另一方面，引入子模型有利于模型的分层和问题的简化，我们可以先对系统高层建模，形成描述系统顶层结构的使用模型，再对其进行扩展分别对各部分建立子模型。采用基于使用模型的统计测试办法，可以有效的对测试规模和效率进行控制，达到正确性验证所需要的效果。笔者在银行凭证系统的开发过程中，通过净室工程的分析和测试方法，实践证明取得了良好的软件质量和测试效率的效果。

6. 参考文献

[1] 贲可荣等译，《净室软件工程：技术与过程》，电子工业出版社 2001。

[2] Stacy J.Prowell,Carmen J.Trammell, and Richard C.Linger,Jesse H.Poore. Cleanroom Software Engineering: Technology and Process. Addison-Wesley 1998.

[3] S.J.Prowell. Computations for Markov Chain Usage Models. Software Quality Research Laboratory.

[4] James A.Whittaker and Michael G..Thomason. A Markov Chain Model for Statistical Software Testing. IEEE Transactions on Software Engineering,vol 20, October 1994, pp.812-824.

[5] Kirk Sayre. Improved Techniques For Software Testing Based on Markov Chain Usage Models. Ph.D. Dissertation,The University of Tennessee:Knoxville, December 1999.

[6] Harlan D.Mills,Michael Dyer, and Richard C.Linger. Cleanroom Software Engineering. IEEE Software vol.4 September 1987:19-87.

[7] J.H.Poore,Harlan D.Mills, and David Mutchler. Planning And Certifying Software System Reliability. IEEE Software vol.10 January 1993:88-99.

[8] Richard C.Linger. Cleanroom Process Model. IEEE Software vol.11 March 1994:50-58.

[9] S.J.Prowell and Jesse H.Poore. Sequence-Based Software Specification of Deterministic Systems. Software Practice and Experience vol.28 March 1998:329-344.

[10] P.A.Hausler.R.C.Linger, and C.J.Trammell. Adopting Cleanroom software engineering with a phased approach. IBM System Journal VOL 33,No 1,1994:89-109.

[11] Gwendolyn H.Walton, J.H.Poore, and Carmen J.Trammell. Statistical Testing of Software Based on a Usage Model. Software Practice and Experience January 1995.

[12] J.G.Kemmeny and J.L.Snell, Finite Markov Chains, Springer-Verlag: New York, NY,1976.