软件测试理论初步框架

王蓁蓁

(金陵科技学院信息技术学院 南京 211169) (江苏省信息分析工程实验室 南京 211169)

摘 要 软件测试是软件开发中不可或缺的部分,也是软件工程化方法中的重要环节。目前各种软件测试技术日趋成熟,但相关的测试基本原理框架还有待开发。在前人经验的基础上,试图提出一个初步理论框架来定义软件测试的样本空间,引入反映软件某种情况(比如缺陷)的随机变量,概括白盒测试和黑盒测试的概率测度及数学期望描述。这样的构建不仅能够加深对软件缺陷存在的理论根源的理解,从而进一步提出更好的测试方法,还对发展软件测试的科学理论有所帮助。

关键词 软件测试,软件度量,程序语言,随机测试

中图法分类号 TP311 **文献标识码** A

Elementary Theoretical Framework for Software Testing

WANG Zhen-zhen

(School of Information Technology, Jinling Institute of Technology, Nanjing 211169, China)

(Information Analysis Engineering Laboratory of Jiangsu Province, Nanjing 211169, China)

Abstract Software testing is a dispensable and important part for software development and software engineering. Various techniques of software testing are nowadays refined, however, relevant testing foundations are still missing. This paper, based on experiences of pioneers, tried to provide an elementary theoretical framework in which a sample space of software testing is defined, and a random variable reflecting something (e. g. bugs) of software is introduced and the probability measure and conditional expectation about white-box testing and black-box testing are generalized. This construction aims to deepen our understanding about why software bugs exist, so that software testing can be improved. Moreover it may be beneficial for developing the scientific theory of software testing.

Keywords Software testing, Software metrics, Programming language, Random testing

1 引言

软件工程是一项具有很大风险的事业。为降低风险,人们采取了各种质量保证、质量控制措施,其中软件测试便是重要的一种手段。几十年来,软件行业已经积累了许多有关测试经验,开发了许多方法[1-17]。本文的贡献是:总结现有的软件测试经验,并在此基础上提出一个初步理论框架,该理论框架构建的思想是由软件内在特性挖掘出软件缺陷的两个最深层次的根源,然后给出解决它们的理论方法。它能解决以下几个方面问题:

(1)软件存在缺陷的理论根源。理论框架模型明确指出了软件的固有特性是以一种随机的方式产生一些不可预见的结果。当这些结果不是该形式系统所想要表达的东西时,便称之为软件缺陷。然而,现有的对软件缺陷的研究文献都是针对具体情况分析,我们的模型在总结前人软件测试经验的基础上结合数理逻辑理论,明确指出软件缺陷的根本根源。

(2)软件测试方法的理论分类。为了解决形式系统操作的不可预见问题(即上述的软件缺陷理论根源),应该提倡随机测试方法。所以模型框架理论首次将现有的软件测试方法从理论上进行了分类,并且提出要对软件测试建立一套随机测试的理论和方法。由此,我们得到:

(3)软件测试统计理论框架。该框架利用随机变量、条件数学期望、超滤等概念给出相应的模型构建。

本文第2节指出软件缺陷存在的两个理论根源;第3节总结软件测试的主要方法;第4节为第5节做准备,介绍相关的预备知识;第5节提出软件测试随机理论;最后是总结及展望。

2 软件存在缺陷的两个理论根源

2.1 形式系统操作不可预见性

Brooks在"No Silver Bullet"("没有银弹")一文中,揭示了在软件生产中存在的本质性问题。姑且不论软件在它的生命周期过程中经受的无数改变,就是软件的内在特性,诸如复杂性、无形性和不可见性,都使得软件过程的改进受到巨大的限制[1]。虽然Brooks是以软件过程的改进立论,但是他揭示出的软件内在特性却是作为形式系统的软件所固有的特征。

实际上,数理逻辑早就揭示了形式系统本身固有的局限,例如有关数学的一些形式系统不完备定理和不可判定性定理

到稿日期:2013-05-17 返修日期:2013-07-20 本文受国家自然科学基金项目(61170071),金陵科技学院科研基金(jit-b-201207)资助。

王蓁蓁(1975—),女,博士后,副教授,CCF会员,主要研究领域为软件测试、人工智能,E-mail:wangzhenzhen@seu.edu.cn。

便是数理逻辑划时代的成就。形式系统的本质在于抽象，它往往摒弃一切表面的、无关的细节，企图从根本上以一种抽象的形式，逻辑地概括它所表达的事物。正因为如此，从逻辑上看似已经完整刻画了东西，它却像 Brooks 所指出的那样，以一种不可预见性把另外一些无关的东西也概括进来或者它以一种不可预见性产生另外一些意想不到的结果。这里先举一个最简单的例子。在康托尔创立集合论学说时，集合是一些具有某特定性质的元素的聚集，人们认为它具有非常明白清晰的概念。用 P 表示某性质谓词，那么集合$\{x|P(x)\}$表示所有使性质谓词 P 成立的元素组成的集合。换句话说，对于任意给定一个条件 $P(x)$，必有一个集合 A 使得

$x\in A\leftrightarrow P(x)$

这里，“↔”表示等价。这叫做概括原理(原则)，它是集合论基本原理(原则)，在集合论中经常使用。谁也没有想到，就在这个明白如水的抽象形式里，如果把性质 P 换成不属于性质，即用$\neg(x\in x)$表示 x 不是 x 的元素，便会产生一个意象不到的结果。具体地说，利用概括原理，理应有一个集合 A 使得

$x\in A\leftrightarrow\neg(x\in x)$

用 A 代入 x 处得：

$A\in A\leftrightarrow\neg(A\in A)$

它便是罗素所发现的悖论：$A\in A$ 为真当且仅当它为假[2]。为了消除悖论，数学家意识到概括原则的使用应该受到某些限制，为此许多数学家对集合论进行了大量的研究，提出了一些更为可靠的理论，但并未从根本上解决问题。这里根本上还有另外一层意思，亦即我们不知道一个定义良好的形式系统在什么地方会出现问题。

从某种广泛的意义上说，语言也是一种形式的系统，它有严格的词法、句法甚至是篇章结构，不是人们随便想怎么说就怎么说，而别人都能理解的。但是随着环境的变化，长期的语言演变，使得至少在语言产生的原始阶段还是一种形式系统的语言早已被人们认为是一个非形式系统了。所以在唐朝的诗人王昌龄居然看到了“秦时明月汉时关”，这样富有诗意的句子，从纯形式系统角度却无法想像能够产生出来。事实上，动物的行为、人类早期的行为都是形式的，只不过基于时间的变迁和丰富的想象力，人类的许多行为才变为非形式系统。

现在再来分析几个著名的软件失败的例子。1999 年 12 月 3 日，美国航天局的火星极地登陆者号探测器试图在火星表面着陆时失踪。事后故障评估委员会在测试中发现，许多情况下，当探测器的脚迅速撑开准备着陆时，机械震动会触发着陆触点开关，设置致命的错误数据位。原来在探测器计算机中设置一个数据位来控制触点开关，原本要到脚“着地”时才关闭燃料。由于计算机在形式上分不清机械震动与着陆时的震动这两种情况，极有可能在探测器开始着陆时，计算机就关闭着陆推进器，致使火星极地登陆者号探测器飞船下坠 1800 米之后冲向地面，撞成碎片[3]。下面的例子也许更有说服力。1979 年 11 月 9 日，美国战略防空司令部收到由全球军事指挥控制系统(WWMCCS)计算机网络发出的警报，警报内容是苏联已经向美国发射导弹，这引起了混乱。幸运的是，灾难在最后一分钟内得以避免。原来计算机把模拟演习当成了真的，也就是说软件原本的设计无法从形式上区分模拟和真实，从而无法从虚假中分辨出真实[1]。

关于上述事故，人们作了许多分析，但大多忽略了一个关键点，就是形式系统无论怎样正确、完美，它自身都无法判断在另外什么地方会出现什么问题。正如佛所说：指向月亮的手指不是月亮。因此，关于软件缺陷来源的第一个结论，自然就是：既然，软件是一个形式系统，它固有的特性便是以一种随机的方式产生一些不可预见的结果。当这些结果不是人们所想要的东西，或是说它不是该形式系统想要表达的东西时，人们便称它是软件缺陷，这是软件缺陷最深处的一种根源。

2.2 编码语言的弱点

软件是以某种程序语言编码的。现存的语言非常丰富，通用的语言就有十几种。每一种语言或者说每一种编程范型都具有一定局限性，这是用该语言编写的软件之所以会产生缺陷的另一种原因。

Dijkstra 发现程序语言中含有的 goto 语句是一种固有的容易出错的程序结构。由此引发的结构化程序设计方法是软件工程发展中的一个重要里程碑[16]。同样他对 PL/I 语言的复杂性所做的批评：“我绝对无法预见，当程序设计语言(请注意，这是我们的基本工具)已经超出了我们的智力控制范围时，我们如何能够仍然牢固地将不断增长的程序置于我们的智力掌握之下。”[8]也说明了程序设计语言对运用该语言开发出的软件可能出现的缺陷具有极大相关性，即程序语言的“弱点”容易引发程序员犯错误并增加软件在其弱点处的可靠性丧失的几率。

Pierce 指出：“我们可以说一个安全的语言是保护它自己的抽象的语言。每个高层语言提供机器服务的抽象。安全性是指语言具备保证这些抽象，以及程序员用语言的定义工具引入高层抽象的完整性的能力[4]。”

虽然“安全语言”目前仍存在着很多争议，人们对语言安全存在许多不同的观点，但是这些争议都使人们注意到许多语言存在着不安全因素。例如 Modual-3 和 C ♯ 提供了一个“不安全的子语言”，用于实现低级执行时间完成的功能，如垃圾收集。这个子语言的特点可能只能用于明显注明 unsafe 的模块[4]。

《软件测试》一书中写道：“在 2002 年，Microsoft 开始主动确认通用 C 和 C++函数中容易引起缓冲区溢出的编码错误[3]。”

许多语言为了自身的安全性，不得不限制自己，例如 John Reynolds 说“类型结构是一个用来限制抽象程序的语法规则[4]。Mitchell 也指出，在程序语言的设计中，当结合了熟知的不可预见多态机制时，往往都隐含了某种“折中”。[5]

这些限制和折中恰好从另一角度反映编程语言是产生软件缺陷的另一种来源。

正如 Mark Manasse 所说：“类型理论所要解决的基本问题就是保证程序有意义。而由类型理论引发的基本问题却是有意义的，程序往往不具备其应有的意义。”例如在 C 语言中，无标记的联合类型违反了类型安全性，它允许对 $T_1\vee T_2$ 的元素作任何操作，只要对 T_1 或者 T_2 有意义就行。然而另一种理论却认为：如果只知道一个值 v 有类型 $T_1\vee T_2$，那么唯一可以对 v 进行的安全操作就是对 T_1 和 T_2 都有意义的操作(比如说，如果 T_1 和 T_2 都是记录，只有将 v 投影到它们的公用字段上才有意义)[4]。

采用面向对象范型原因很多，其中最重要的原因就是当正确使用它时，它可以解决一些传统范型遇到的问题。众所

周知，传统范型要么面向操作，要么面向属性(数据)，不会同时面向两者。这是传统范型不完全成功的重要原因。面向对象范型将属性和操作同等看待，并且由于设计良好的对象是独立的单元，其信息隐藏确保实现细节与对象外部的一切事物完全隔离，这不仅降低了软件产品的复杂度，而且简化了软件开发和维护，提高了重用度。遗憾的是，面向对象自身也有自己的问题。例如有所谓"脆弱的基类问题"，即一旦实现一个产品，对已存在的类进行修改就会直接影响继承树中它的所有子孙。另外不加约束地使用继承，会使继承树低层的对象很快变得庞大起来，这不仅引起存储问题，而且对维护也带来一定的困难。特别是它比传统范型更容易写出坏的代码[1]。

最后，还要提一下具有"非过程"特征的第四代语言(4GL)，尽管它更容易编程，存在着潜在的生产率增长，但它也存在着许多甚至是灾难性失败的危险[1]。

结论：实用的程序语言总是很大、很复杂，更何况用它编写的软件还要与编译器、操作系统以及硬件等各种形式系统配置在一起，其不可预知的不确定性确是产生软件缺陷的另一种来源。

3 软件测试的主要方法

第2节实际上已经给出软件测试方法论框架。为了解决编程语言的局限问题，应该提倡证明论方法。为了解决形式系统操作不可预见问题，应该提倡随机测试方法。现在详述如下。

3.1 审查和证明

Grady在惠普公司比较了不同类型测试技术的效率，他得到的结论是：审查是发现缺陷的最廉价、最有效的测试技术[4]。Myers曾经比较了黑盒测试、黑盒和白盒测试的结合与代码走查，得到的结论是这3项技术在发现错误方面同样有效，但代码走查比其他两项技术成本低。Hwang比较了黑盒测试、白盒测试和由一个人所做的代码阅读，发现所有3项技术同样有效，每项技术都有各自优缺点。Basili和Selby完成的一项主要试验，也是比较黑盒测试、白盒测试和一人代码阅读。他们得出的主要结论是，代码审查在检测错误方面最起码与白盒测试和黑盒测试一样成功[1]。

正确性证明是显示产品正确的一种数学技术。Dijkstra认为："程序员应让程序证明和程序一起发展"。例如在设计中应用循环时，应提出循环不变式。用这种方式开发产品时，仍然用他的话来说："提高程序的信心的唯一有效方式是对它的正确性给出有说服力的证明"[1]。

为了实施正确性证明，计算机科学研究了程序设计语言的公理语义，例如霍尔逻辑给出一组有关部分正确性的证明规则，并且建立最弱前置条件与可表达性等重要概念；Dijkstra在讨论完全正确性时把命令的含义规定为谓词转换器[7]。可惜的是，这些形式化方法过于理论性，应用起来"代价昂贵"，很难被程序员接受。于是出现了类型系统这种较为实用、完善的轻量级形式化方法。用Robin Milner的话来说："良类型程序不会出错"[4]。

结论：为了防止由于编程语言的弱点而导致软件缺陷，应该强调审查和正确性证明方法。关于正确性证明可以分为非形式证明和形式证明两种样式。前者偏重于定性非形式描述，后者偏重数学证明。

很好利用上述结论的一项开发技术是净室软件开发技术。它的一个重要特征是一个代码制品必须通过审查才编译，即一个代码制品仅在代码阅读、代码走查和审查之类的测试成功完成后才进行编译。在设计阶段，净室开发技术强调尽可能采用一些非形式化证明，而在审查者不完全相信受审查设计部分的准确性时，才给出完全的数学证明[1]。有许多证据表明，净室开发技术在一些应用领域很成功，给人留下很深的印象。

3.2 随机测试

第2节的基本思想是，无论软件编码怎样完美，由于形式系统固有的特性，总有不可预知的情况发生。实际上，这与现代物理量子理论的精神是一致的。量子世界是概率世界，软件产品同样如此，软件测试只能报告它所发现的缺陷确实存在，却不能报告软件缺陷不存在，即便对软件代码进行了严格的审查和完整的数学证明(当然这是必需要做的工作)，根据数理逻辑不可判定性和不完备性定理，只要软件产品足够复杂，人们就无法保证软件在运行时不会出现异常。回忆概率论有几个著名的论断：

1)如果事件的概率很小，那么在一次试验中，它不会出现。

2)无论事件的概率多么小，只要试验次数足够大，它必然会出现。

3)许多经验上认为是小概率的事件，往往它们的概率比较大，它们出现的可能性也比较大。

英特尔奔腾浮点除法存在缺陷，但它只有在进行精度要求很高的数学、科学和工程计算中才会导致错误，大多数用来进行税务处理和商务应用的客户根本不会遇到此类问题。然而就是这种很少见的情况却在1994年引发了一场用户与英特尔公司之间纠纷的风暴[3]。

在程序语言设计的理论中，也有资料表明[4]要使子类型检查器发散，它必须具备3个特殊性质，每一性质都不可能碰巧创造出来。这些例子说明，无论是理论研究或是软件开发，人们只偏爱前述概率论中第一个结论，对于小概率的事件放心地弃之不顾，就好像我们每天悠闲地在大街上行走，从不考虑车祸一样。遗憾的是，概率论中的第2个和第3个结论，却在日常生活中和软件产品的运行中遇到障碍时才受到重视。

要想避免意外发生，单凭"确定性"测试软件方法是不够的。这里需要某种"不确定性"测试，把它命名为随机测试。现存的软件测试方法中，例如黑盒测试、即兴测试，在一定意义上都可以认为是随机测试。特别是有种称为统计测试的测试策略就是一种随机测试。统计测试基于运行剖面的描述，运行剖面通常将可能的用户的输入组成一个空间，并在其上定义概率分布[6]。

不过，一套完整的随机测试理论和方法在软件界尚未建立。本文也只想在第4节阐述一些必要知识，在第5节给出随机测试的一个初步框架，而将它的发展和完善留给今后的工作。

日立公司很好地贯彻了上述思想并取得成效。它是日本最大的软件机构，为了向用户保证质量，它在单元测试和集成测试时采用统计测试。基于经验数据和统计分析，加强某些类型的测试，直到满足质量目标时该产品才得以通过[6]。

此外，Mills 的净室技术并入了统计测试[6]，净室技术的结果和日立公司的作法一样都提供了主要的经验证据，表明可以实际并且有效地应用统计测试。这也说明在软件测试的实践上，建立随机测试的理论和方法是有意义的。

4 预备知识

4.1 σ-代数、概率、概率空间[8]

1)σ-代数、可测空间和可测集定义

设 Ω 是样本点 w 的集，$\Omega=(w)$。Ω 的某些子集所成的集 $\mathcal{F}=\{A\}$ 如果满足下述条件：①$\Omega\in\mathcal{F}$；②若 $A\in\mathcal{F}$，则 $\bar{A}\in\Omega-A\in\mathcal{F}$；③若 $A_i\in\mathcal{F}, i=1,2,\cdots$，则 $\bigcup_{i=1}^{\infty}A_i\in\mathcal{F}$。则称 $\mathcal{F}$ 为 Ω 中的 σ-代数，这时称已确定了某 σ-代数 $\mathcal{F}$ 的空间 Ω 为可测空间，记为 $(\Omega,\mathcal{F})$，其中 $\mathcal{F}$ 中的集称为 $\mathcal{F}$ 可测集，如果没有其它的 σ-代数引起混淆时，就简称为可测集。

2)概率和概率空间定义

设 $(\Omega,\mathcal{F})$ 为一可测空间，P 为定义在 $\mathcal{F}$ 上的集函数，如果它满足下述条件：①$\forall A\in\mathcal{F}, P(A)\geqslant 0$；②对有穷或可列个集 $A_i\in\mathcal{F}, i=1,2,\cdots, A_i\cap A_j=\emptyset, i\neq j$（即 A_i 和 A_j 不相交），有 $P(\bigcup_{i=1}A_i)=\sum_{i=1}P(A_i)$；③$P(\emptyset)=0, P(\Omega)=1$。则称 P 为概率测度，简称概率。把 3 个对象 $\Omega,\mathcal{F}$ 和 P 合在一起，称 $(\Omega,\mathcal{F},P)$ 为概率空间。

3)注记

直观上，样本点是讨论的对象基元，Ω 是所有基元的集合。从 Ω 中挑选一些子集，其往往是人们感兴趣或者必须要考察的事件，这些事件组成一个集合，如果满足 1)中的 3 个条件，便称为 σ 代数。满足这些条件的 σ 代数不仅便于数学处理，而且由此推出的其他性质足以满足理论和实践的需要。概率 P 反映 $\mathcal{F}$ 中的集（即要研究的事件）出现的可能性。例如在统计测试中的运行剖面，通常将用户的可能输入空间划分为一些截然不同的类，并为每一个类赋予从该类中选取输入的概率。在那些截然不同的类上可以构造一个 σ 代数，并将为每一类赋予的输入概率扩展到这个 σ 代数，从而得到一个运行剖面的概率空间模型。

4.2 随机变量[8]

1)随机变量定义

设 $(\Omega,\mathcal{F},P)$ 为一概率空间，定义在 Ω 上的实值函数 $\xi(w)$，如果对任意实数 $x\in R_1$（R_1 表示实数空间），$(w:\xi(w)\leqslant x)\in\mathcal{F}$，则称 ξ 为随机变量。

2)数学期望定义

设 ξ 为 $(\Omega,\mathcal{F},P)$ 上的随机变量，如果 $E|\xi|=\int_{\Omega}|\xi|P(\mathrm{d}w)<\infty$，则称 ξ 绝对可积，称 $E\xi=\int_{\Omega}\xi P(\mathrm{d}w)$ 为 ξ 的数学期望（即通常意义下的均值）。

3)注记

直观上，在概率空间上考察某些随机行为，往往要对这些行为进行度量，也就是量化它们的变化。既然是随机行为，除了知道它们取值以外，还要对这些量化的变化估计它们取值的概率，这就是要知道它们取某个值或在某区间上取值的概率，而上面定义便是这个要求的抽象化、一般化。具体到软件测试，在运行剖面上考察软件运行。

4.3 滤，超滤

1. 滤定义：设 $\mathcal{S}$ 是样本空间 Ω 中的子集合簇，如果 $\mathcal{S}$ 满足下述性质：

(1)$\mathcal{S}$ 非空；

(2)若 $A\in\mathcal{S}, B\in\mathcal{S}$，则 $A\cap B\in\mathcal{S}$；

(3)若 $A\in\mathcal{S}$，且 $A\subseteq B$，则 $B\in\mathcal{S}$。

则称 $\mathcal{S}$ 为 Ω 中的一个滤（参考非标准分析）。

2. 超滤定义：如果 $\mathcal{S}$ 是样本空间中的一个滤，且对任意 $A\subseteq\Omega$，A 和 $\bar{A}$ 两集合中有且仅有一个属于 $\mathcal{S}$，则称 $\mathcal{S}$ 是 Ω 中的一个超滤。

5 软件测试随机理论

5.1 样本空间 Ω 与（缺陷）随机变量 ξ

1)（软件测试）样本空间 Ω 定义

令 $\Omega=\{w\}$ 为样本点 w 的集合，$\mathcal{F}$ 为 Ω 上的一个 σ-代数，P 为 $\mathcal{F}$ 上的一个概率测度，称三元组 $(\Omega,\mathcal{F},P)$ 为（软件测试）的样本空间，简记为 Ω。

在软件测试实践上，样本点 w 的数目是非常庞大的，所以在理论上认为 Ω 为可数无穷空间是合理的。这样可以把 Ω 上所有子集组成的集合簇看成是 Ω 上的 σ-代数。自然，Ω 中有的子集并非是人们感兴趣的事件，所以从 Ω 中挑选出有意义的事件组成 σ-代数很有必要。但选取 Ω 中所有子集组成 σ-代数在理论上是很容易的。

2)（反映软件情况）随机变量 ξ 的定义

设 $(\Omega,\mathcal{F},P)$ 为样本空间，令 ξ 为 Ω 上的随机变量，即对于任意样本点 $w\in\Omega$，$\xi(w)$ 有定义，且 $\xi(w)$ 为 $\mathcal{F}$ 可测。

要注意的是，这里并未给出 ξ 的含义，它只是反映软件的某种情况，比如缺陷，可以根据具体问题确定其实际意义。

3)（软件情况）保测映射 ξ 定义

设 $(\Omega,\mathcal{F},P)$ 为软件测试空间，$(\mathcal{S},\mathcal{G})$ 为任一可测空间，令 ξ 为 Ω 到 $\mathcal{S}$ 上的映射：$\Omega\to\mathcal{S}$，并具有下述性质：对 $\forall A\in\mathcal{F}$，有 $\xi(A)\in\mathcal{G}$，则称映射 ξ 为从 Ω 到 $\mathcal{S}$ 上的保测映射。

实际上，对软件描述的 ξ 变量，人们主要关心的是它在 Ω 上每一个可测集上的行为，即对任意 $A\in\mathcal{F}$，用 $\xi(A)$ 表示 ξ 在 A 上的软件行为。例如取 $(\mathcal{S},\mathcal{G})=(N,\mathcal{G})$，其中 $N=\{0,1,2,\cdots\}$ 为自然数（包含 0）集合，$\mathcal{G}$ 为 N 中所有子集组成的 σ-代数。如果令 ξ 表示缺陷数，若定义

$$\xi(w)=\begin{cases}1, & \text{如果在样本点 } w \text{ 处出现缺陷}\\ 0, & \text{如果在样本点 } w \text{ 处未出现缺陷}\end{cases}$$

则 $\xi(A)=\sum_{w\in A}\xi(w)$ 可以为某自然数。对于随机测试来说，用这种观点考虑软件情况比用随机变量描述软件情况更为合理。

4)注记

上述定义的实际含义是：在软件测试实践上，最常用的白盒测试方法是审查代码。这时可以把每一行代码看成一个样本点 w，其所有的样本点 w（即代码）组成样本空间（即代码空间）。这时，结构模块或是所有的代码的某种逻辑模块，组成该样本空间上的一个 σ-代数 $\mathcal{F}$。$\mathcal{F}$ 上的概率测度 P 可以这样规定：

$$\text{对}\forall A, P(A)=\frac{A\text{ 中所含的代码数}}{\Omega\text{ 中代码总数}}=\frac{A\text{ 中的样本点总数}}{\Omega\text{ 中的样本点总数}}$$

如果采取黑盒测试技术，可以把每一个输入看成是一个样本点 w，所有的样本点 w（即所有输入）组成样本空间（输入空间）Ω。Ω 上所有子集构成 σ-代数 $\mathcal{F}$，而 P 可以用经验频率

来衡量。即 $\forall A, P(A)$ 为用户使用 A 中的输入的频率。

甚至可以把一个测试用例作为一个样本点 w，所有的样本点(即所有的用例)组成样本空间 Ω(即用例空间)。相关测试用例可以组合成一个子集，所有这样的子集便组成一个 σ-代数 $\mathcal{F}$。定义概率 P 时，既可以用比例方法也可以用频率方法，视具体情况而定。

其中比例方法为：对 $\forall A, P(A)=\frac{A\text{中所含用例总数}}{\Omega\text{中的用例总数}}$；

频率方法为：对 $\forall A, P(A)$ 为 A 中用例使用的经验频率。

我们只在一般意义上定义了随机变量 ξ，只是说用它来反映软件缺陷情况，并未给出它的具体含义，因而也没有给出它的具体分布信息。

Musa、Iannino 和 Okumoto 分析了有关错误密度的可用数据。他们得出结论：当研究者测量一个代码制品的复杂度时得到的结果很大程度上可能是代码行数的反映，它又与错误数有很强的相关性。复杂性度量对通过代码行数预报错误几乎没有什么改进[1]。当用这种观点考虑问题时，我们便可以把 $(\Omega, \mathcal{F}, P)$ 选择为上述的代码样本空间，而反映软件制品的情况随机变量即可以定义为错误数 ξ，这时对 ξ 感兴趣的问题是错误率问题。

如果考虑软件的可靠性，我们采用最早、最有名的 Jelinski-Moranda 模型时[2]，可以把反映软件寿命的随机变量 ξ 取为某种随时间演变的指数分布。

另外，我们认为在有些情况下用软件保测映射比用软件随机变量更为恰当。随机变量与保测映射是有区别的。当考虑随机变量的分布时，它往往是把基于在软件测度空间 $(\Omega, \mathcal{F}, P)$ 上的考察转变为在实数空间上的考察，而保测映射有一种直接在软件测度空间 $(\Omega, \mathcal{F}, P)$ 上考察的"韵味"。

5.2 条件数学期望及其定性分析

1)条件数学期望定义

设 $(\Omega, \mathcal{F}, P)$ 为概率空间，$\Omega=(w)$，$\mathfrak{B}$ 是 $\mathcal{F}$ 的子 σ-代数(意指 $\mathfrak{B}$ 中的集合都是 $\mathcal{F}$ 中的集合，即 $\mathfrak{B}\subseteq\mathcal{F}$)，$\xi(w)$ 是某随机变量，且 $E|\xi|<\infty$。称具有下列二性质的随机变量 $E(|\xi|\mathfrak{B})$ 为 $\xi(w)$ 关于 $\mathfrak{B}$ 的条件数学期望(简称条件期望)，如果：①$E(\xi|\mathfrak{B})$ 是 $\mathfrak{B}$ 可测函数；②对任意 $A\in\mathfrak{B}$，有 $\int_A E(\xi|\mathfrak{B})P(\mathrm{d}w)=\int_A \xi P(\mathrm{d}w)$[8]。

2)定性分析

直观上，相应于软件测试，所考虑事件集合 σ-代数 $\mathcal{F}$ 过于庞大，以致于在实践上无法对它进行完整测试。自然想把它们"浓缩"到可行程度，于是从 $\mathcal{F}$ 中精选出子 σ-代数 $\mathfrak{B}$。原先在 $\mathcal{F}$ 中考察的程序行为，现在要在 $\mathfrak{B}$ 中进行考察。为了不失去程序在 $\mathcal{F}$ 中的信息，必须把程序行为平滑、平均地反映到它在 $\mathfrak{B}$ 中的行为。这就产生了条件数学期望的概念。其中条件 1 即 $E(\xi|\mathfrak{B})$ 是 $\mathfrak{B}$ 可测，表示把原本在 $\mathcal{F}$ 下考察 ξ 转换到在 $\mathfrak{B}$ 中考察 $E(\xi|\mathfrak{B})$；而条件 2 反映了对于任意 $\mathfrak{B}$ 中集合 A，$E(\xi|\mathfrak{B})$ 在 A 上平均行为与原本 ξ 在 A 中的平均行为一致。两个条件合在一起不仅表明新随机变量 $E(\xi|\mathfrak{B})$ 保留了原先 ξ 的信息，它保证在可行性约束下最大可能研究原先程序的行为，而且也可以看出命名 $E(\xi|\mathfrak{B})$ 为 ξ 关于 $\mathfrak{B}$ 的条件数学期望的理由。

在运用黑盒测试策略时，往往根据边界条件或次边界条件进行一些等价类划分，其精神就是寻找子 σ-代数 $\mathfrak{B}$，在子 σ-代数 $\mathfrak{B}$ 上考察整个软件行为。

现在提出的条件数学期望理论至少使我们加深了对黑盒测试所用的等价类划分策略的理解，它只是在平均意义上和完整测试相当。而如果子 σ-代数 $\mathfrak{B}$ 选择不当(如果太大，没有可行性；如果太小，精度不够)，它的效果就很差。极端情况，如果选择 $\mathfrak{B}$ 为 $\{\Omega, \varnothing\}$，即全集和空集这两个集合组成的子 σ-代数，那么 $E(\xi|\mathfrak{B})$ 便与 $E\xi$ 相当，它只反映 ξ 在空间上的均值这一信息，自然就不能指望 $E(\xi|\mathfrak{B})$ 做更多的工作了。

5.3 超滤模型

1. 模型定义：如果软件样本空间 $(\Omega, \mathcal{F}, P)$ 中的概率测度 P 具有下述性质：即 $\forall A\in\mathcal{F}, P(A)$ 等于 0 或 1，则称该样本空间 $(\Omega, \mathcal{F}, P)$ 为软件的超滤模型。

2. 超滤模型的定性描述：由于概率 P 具有递增性，若 $A\subseteq B$，则 $P(A)\leqslant P(B)$。因此若 $P(A)=1$，就可由关系 $P(A)\leqslant P(B)\leqslant 1$ 推出 $P(B)=1$。这样把所有 $\mathcal{F}$ 中具有概率 1 的集合聚在一起，不妨记它为 $\mathcal{S}$，它恰好满足下述关系：

(1)$\mathcal{S}$ 非空，因为 $P(\Omega)=1$，所以 $\Omega\in\mathcal{S}$。

(2)若 $A\in\mathcal{S}$，且 $A\subseteq B$，则 $B\in\mathcal{S}$，上面已证。

(3)若 $A\in\mathcal{S}, B\in\mathcal{S}$，则 $A\cap B\in\mathcal{S}$。

因为若 $A\cap B\notin\mathcal{S}$，就有 $P(A\cap B)=0$，所以 $P(\overline{A\cap B})=1$，即

$$P(\bar{A}\cup\bar{B})=P(\overline{A\cap B})=1$$

但是 $P(\bar{A})=1-P(A)=0, P(\bar{B})=1-P(B)=0$

所以 $P(\bar{A}\cup\bar{B})\leqslant P(\bar{A})+P(\bar{B})=0$。矛盾。

根据非标准分析数学理论，具有上述性质的集合簇 $\mathcal{S}$ 称为 Ω 上的一个超滤。所以在上面模型定义中称 $(\Omega, \mathcal{F}, P)$ 是超滤模型。

3. 在软件测试中的应用：如果把 $\mathcal{F}$ 取作软件所有模块产生的 σ-代数，考察软件在各个模块上是否失败(即有缺陷)，令 P 为其概率且对任意一个模块 A，$P(A)$ 非零即 1。这样 $(\Omega, \mathcal{F}, P)$ 就是一个超滤模型。显然，这个模型和前面介绍的保测模型是从不同角度符合软件测试的实际情况。更何况，由于所有具有概率为 1 的子集组成的集合簇 $\mathcal{S}$ 是 Ω 上的一个超滤，也许能将非标准分析这一强大的数学工具引入到软件测试中来。

结束语 假设程序员非常优秀，他们创造的软件产品没有编码错误，在这种理想情况下，讨论软件产品的缺陷问题，结论是：这仍然无法保证软件不会有隐藏的缺陷。这主要归因于编程语言(及其有关的软、硬件)和软件本身都是复杂的形式系统所致。为了促进软件测试工作，提出了软件测试的理论框架。这样做的目的主要是：

企图和软件测试是一项讲究条理的技术专业，并行发展出一种软件测试的科学理论。

计算机问世不久，便诞生了计算机科学。很快人们就发现在发展技术的同时并行发展理论很重要。例如有关逻辑学和程序设计语言理论的研究，不但加深了人们对现存的编程语言的理解，而且也直接促使某些语言的出现，其中 Lisp 语言受到 λ-演算理论的启发，Prolog 逻辑程序设计语言是以霍恩子句逻辑为基础的主级程序设计语言。

众所周知，在软件测试中进行统计测试并不容易，就是定

(下转第 35 页)

气 8 类情感类别进行多标签分类，取得了令人满意的结果。

文本只是利用主题特征来研究文本句的情感极性分类，目前的研究还有许多提高和改进的余地，如何以词和句的情感分析为基础研究篇章的情感问题也是今后进一步研究的方向。

参 考 文 献

[1] 赵妍妍，秦兵，刘挺. 文本情感分析[J]. 软件学报，2010，21(8)：1834-1848

[2] Hu Ming, Liu Bin. Mining and Summarizing Customer Reviews [C] // Proceedings of the 10th International Conference on Knowledge Discovery and Data Mining. 2004：168-177

[3] Dave K, Lawrence S, Pennock D M. Mining the peanut gallery: Opinion extraction and semantic classification of product reviews [C] // Proceedings of WWW-03, 12th International Conference on the World Wide Web. Budapest, HU, ACM, 2003：519-528

[4] 姚天昉，程希文，徐飞玉，等. 文本意见挖掘综述[J]. 中文信息学报，2008，22(3)

[5] Turney P D. Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews [C] // Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics. 2002：417-424

[6] Kim S M, Hovy E. Automatic identification of pro and con reasons in online reviews [C] // Dale R, Paris C, eds. Proc. of the COLING/ACL 2006. Morristown：ACL，2006：483-490

[7] 任福继，等. Document for Ren-CECps 1.0 [OL]. http://al-www.is.tokushima-u.ac.jp/member/ren/Ren-CECps1.0/Ren-CECps1.0.html，2009

[8] Blei D M, Ng A Y, Jordan M I. Latent Dirichlet allocation[J]. Journal of Machine Learning Research，2003，3：993-1022

[9] Tsoumakas G, Katakis I. Multi-Label Classification：An Overview[J]. International Journal of Data Warehousing and Mining，2007，3(3)：1-13

[10] Tsoumakas G, Vlahavas I. Random K-Labelsets：an Ensemble Method for Multilabel Classification [C] // Proceedings of the 18th European Conference on Machine Learning (ECML2007). Warsaw, Poland，2007：406-417

[11] 孙艳，周学广，付伟. 基于主题情感混合模型的无监督文本情感分析[J]. 北京大学学报：自然科学版，2013，1(49)

(上接第 16 页)

义运行剖面也没有简单的或可重复的方法。现在要具体给出样本空间$(\Omega,\mathcal{F},P)$及其上的随机变量ξ分布的构造理论，还要构造$\mathcal{F}$的子σ-代数$\mathfrak{B}$，从而研究$E(\xi|\mathfrak{B})$的条件分布，是有点困难的。但是，如果认真总结和思考数十年软件测试的经验(比如还可参看文献[9-15])，要在上面框架里做出一点成绩还是有希望的，例如我们就是在文献[17]的基础上，利用随机理论提出了随机软件错误定位方法，该方法的优势通过一些实例已经得到了验证，这也说明了我们提出的理论框架是有应用价值的。这正是我们今后研究的动力。

我们知道，在自然工程领域，只要有应用数学方法的都尽量运用，以保证工程质量。唯独软件工程，它属于社会-技术系统，再加上有时时间限制比质量限制更关乎在软件市场上的竞争力，所以软件工程界普遍认为使用数学论证方法不合算，即使 Java 语言提供了一些验证语句手段，它们在运行时往往也是"屏蔽"的(不过它们也确实起到了一定的验证作用)。因此我们强调开展软件测试关于审查和形式证明的研究，使其定位在形式系统不确定性和程序语言弱点上，这不仅方向明确而且更有实际可行性。

另外，概率论也很难"完整"地运用到软件测试中。即使统计测试也大都是关于软件可靠性方面的测试，它是基于统计学里的可靠性理论基础的。究其原因，软件测试里基于样本空间上的理论描述与基于经典概率论样本空间上的描述多少在细节上有微妙差别。为了突出差别，我们提出了样本空间的超滤模型和保测映射两种有别于经典的理论模型，希望它们更能满足软件测试随机理论的需要。对于完全的经典模型，我们也强调了条件期望理论，目前在证明等价划分的合理性上至少也看出该理论的作用。

参 考 文 献

[1] Schach S R. 软件工程—面向对象和传统的方法[M]. 邓迎春，韩松，徐天顺，等译. 北京：机械工业出版社，2007

[2] 莫绍揆. 数学基础[M]. 北京：高等教育出版社，1991

[3] Patton R. 软件测试[M]. 张小松，王钰，曹跃，等译. 北京：机械工业出版社，2007

[4] Pierce B C. 类型和程序设计语言[M]. 马世龙，眭跃飞，等译. 北京：电子工业出版社，2005

[5] Mitchell J C. 程序设计语言理论基础[M]. 许满武，徐建，袁宜，等译. 北京：电子工业出版社，2006

[6] Fenton N E, Pfleeger S L. 软件度量[M]. 杨海燕，赵巍，张力，等译. 北京：机械工业出版社，2004

[7] Winskel G. 程序设计语言的形式语义[M]. 宋国新，邵志清，等译. 北京：机械工业出版社，中信出版社，2007

[8] 王梓坤. 随机过程论[M]. 北京：科学出版社，1978：439-440，450

[9] Desikan S, Ramesh G. 软件测试-原理与实践[M]. 韩柯，李娜，等译. 北京：机械工业出版社，2009

[10] Andersson C, Runeson P. A Replicated Quantitative Analysis of Fault Distributions in Complex Software System [J]. IEEE Transactions on Software Engineering，2007，5(33)：273-286

[11] Cordy M, Classen A, Perrouin G, et al. Simulation-based abstractions for software product-line model checking [C] // Proceeding of the 2012 International Conference on Software Engineering. Zurich, Switzerland，2012：672-682

[12] 周毓明，徐宝文. 基于依赖结构分析的类重要性度量方法 [J]. 东南大学学报：自然科学版，2008，3(38)：380-384

[13] 王蓁蓁. 朴素模糊描述逻辑知识库构造及其朴素推理[J]. 应用科技，2012，39(6)：18-29

[14] Santelices R, Jones J A, Yu Yan-bing, et al. Lightweight Fault-Localization Using Multiple Coverage Types [C] // Proceedings of the 2009 IEEE 31st International Conference on Software Engineering. 2009：56-66

[15] Weimer W, Nguyen T, Goues C L, et al. Automatically Finding Patches Using Genetic Programming [C] // Proceedings of the 2009 IEEE 31st International Conference on Software Engineering. 2009：364-374

[16] Sommerville J. 软件工程[M]. 程成，陈霞，译. 北京：机械工业出版社，2008

[17] 王蓁蓁，徐宝文，周毓明，等. 一种随机 TBFL 方法[J]. 计算机科学，2013，40(1)：5-14