计算机科学1993Vol.20№.3

软件测试的研究及发展趋势

彭建军 （北京航空航天大学软件工程所 北京100083）

摘 要

本文探讨了软件测试的理论基础，综述了围绕寻找接近理想化测试的目标，迄今软件测试的方法与技术，软件测试的准则、策略、步骤，以及国内外研究、工具开发的动态。特别指出了形式化方法对软件测试的影响，提出了当前软件测试应研究的问题与方向，主张在软件开发与测试中寻找强语义的定义的方法，采用前端测试的原则，开发自动化的测试工具，进行软件测试。

在软件开发中，软件测试费用几乎占整个软件开发费用的50%。由此，带动了软件测试理论与技术的发展。但是，软件测试理论中尚存在许多问题未予解决，而且软件测试技术也仍处于原型研制的实验阶段，还未达到实用化。因此，软件测试的研究进展已引起人们的日益关注。

一、软件测试的理论基础

在软件开发中我们往往面对这样的问题：对一输入定义域为D的程序p，其期望结果为f(D)，执行结果为$\overline{p}(D)$，那么$f(D) \stackrel{?}{=} \overline{p}(D)$，即对任意输入$d \in D$，期望结果是否与具体执行结果$\overline{p}(d)$相等。为了回答这一问题，可以用形式化的数学方法，将有关问题的规格说明或程序描述为一些断言及推导规则，证明程序满足规格说明并可运行终止。但是，程序证明尚有许多困难，如：规格说明的形式化；程序断言的产生；证明过程中人工干预的可识别性等等。

J. B. Goodenough和S. L. Gerhart于1975年提出了软件测试的可靠性和有效性（确认）的概念：C是程序p上的测试数据的选择推测，T, T_1, T_2为测试数据集。C是可靠的，即对$\forall T_1, T_2 \in D$，$f(T_1) = \overline{p}(T_1)$ iff $f(T_2) = \overline{p}(T_2)$；C是有效的，即若有某些$d \in D$，使$f(d) \neq \overline{p}(d)$，那么必$\exists T \in D$，使$f(T) \neq \overline{p}(T)$。可靠性定义是指一数据选择准则是可靠的，当且仅当该准则产生的测试数据集使程序运行的输出结果正确；有效性定义是指一数据选择准则是有效的，当且仅当对程序中的每一错误均存在着能显示该错误的测试数据集。其基本定理为：如果一测试数据集的选择准则C是可靠与有效的，那么由C所选的任何测试均是理想的测试，即对程序p，测试数据集T，$\exists d \in D$，$f(d) \neq \overline{p}(d)$，iff $\exists T$，$f(T) \neq \overline{p}(T)$。这就说明，任何一个有错误的软件，都可用测试去检查错误，并确定错误的位置。这些定义和定理，形成了软件测试的理论基础，使人们可围绕测试数据选择准则的可靠性和有效性问题，构造测试数据，去分析与评估软件。然而，寻找理想化的测试尚存在如下问题：

- 可靠性及有效性定义在程序的整个域上，这只能用“穷尽”的方式来证明。
- 所有定义仅与单个程序有关，对于那些互相依赖的程序，将会影响其可靠性及有效性。
- 有效性及可靠性不能保证在程序变换的过程中都保持不变。

因此，寻找既可靠又有效的测试选择准则实际上是不现实的。关键的问题是，怎样才能找到最接近于理想化的测试，尽可能地发现软件中存在的错误。

G. J. Myers[2]指出：程序测试是为了

彭建军 硕士生，研究方向为软件工程及软件测试。

发现错误而执行程序的过程，他对发展软件测试技术作出了贡献；B. Beizer[3,4]对软件测试的理论建立做了卓著的工作，他认为作为质量保证的组成部分，软件测试的目标实际上应能清楚地确定软件中存在的错误；R. A. Demillo[5]等综合了软件测试的理论、技术、测试工具的发展状况；W. E. Houden[6]则在对程序测试与分析的集成化研究的基础上，用形式化方法对软件测试理论进行了描述，从功能测试的角度指出了软件测试的发展方向。

二、软件测试的一般方法与技术

2.1 测试准则

根据软件工程的瀑布模型，软件测试包括设计阶段及单元和编码的测试。测试通常采用黑盒（功能）测试和白盒（结构）测试。

黑盒测试仅注意软件的外部特性，考虑程序的输入/输出是否与规格说明相一致，而不考虑程序的内部特征。具体实现的方法有：

• 等价类划分。即将程序的输入域划分为不同等价类，再从每一个等价类中产生一测试用例。

• 边值分析。在等价类划分后，对某一等价类的输入条件进行边值分析，如对输入域的值范围、个数，输出域的边值的影响的分析等，给出边值的测试用例。

• 因果图法。因果图是一种形式语言，由规格说明的自然语言转换而成。它实际上是一数字逻辑回路，仅用简单的布尔逻辑（与、或、非）来表达输入条件（原因）及输出条件（结果）的关系。因果图除能系统地选择一组高效的测试用例外，还可指出程序规范中的不完全性和二义性。

• 随机测试法。使用随机测试可实际进行程序的测试，获得输入值的期望的运行时间分布。随机测试往往也用于实时测试等。

• 程序变异法。程序变异是指一程序p的变异体为m(p)，是一组程序的集合，其元素是对p的数据流或控制流作变异变换而产生的p的衍生体，称为变异因子。区别p与m(p)的元素的测试，即称为变异测试[7]。

• 属性文法测试法。编译中属性文法一般分为综合属性和继承属性。属性文法的测试方法就是利用属性的值控制产生式中规则的选择，控制使用动作子程序去实现文法到串的翻译，从而产生具有语义的输入输出测试用例。这一方法多用于编译程序的测试。

白盒测试法，是检查程序的内部结构，并由程序的逻辑派生出测试数据。有许多决定程序逻辑覆盖的准则，通常把它们分为控制流与数据流两类：

a. 控制流准则

• 语句覆盖。每条语句至少执行一次。

• 分枝判定覆盖。每条分枝至少执行一次。

• 条件覆盖。每个条件至少执行一次。每个条件存在于分枝的判定之中。

• 条件/判定覆盖。要求判定中的每个条件至少出现一次，每个判定的可能结果至少出现一次。

• 多重条件覆盖。每个判定中条件结果的所有可能组合至少出现一次。

• 路径覆盖。程序中的每一路径至少遍历一次。由于贯穿程序的独立的逻辑路径数相当庞大，故仅能选择一定的路径进行测试。

• 路径前缀覆盖。即对每一个有效的路径p可确定其最小子路径q，q所达的判定结点其分枝并未完全覆盖，由此再选择未覆盖的分枝的路径进行测试。该概念与算法由R. E. Prather和J. P. Myers提出[8]。

• 基本分枝准则。当一分枝无论何时被执行时，另一分枝也要被执行，这后一分枝即为基本分枝。Takeshi Chusho给出了确定基本分枝的方法，即靠删除非基本分枝的弧而实现[9]。

• 子域测试。将程序的输入域划分为路径域，路径域上的输入均使程序的控制流经

过程序流程图中的同一路径或相关路径；或根据软件规格说明、算法及数据结构等隐含的共同性质，将输入域划分为问题域。如规格说明将输入定义为整数时，可划分问题域为奇数和偶数；而从路径域考虑又可分为负整数和正整数，使路径有不同的流向；零则是一种特殊情况。按路径或问题划分的子域而形成的类即可构造测试用例[10]。

由上述测试方法，人们提出了相应的覆盖准则。如语句覆盖率$C_0=100\%$，分枝覆盖率$C_1\geqslant 85\%$时，即认为测试是理想的，软件错误可查出近90%，且时间与空间的消耗均是允许的。这一覆盖标准在软件工程的测试中颇为实用，为大多数软件测试人员所遵循。但上述测试覆盖的方法也存在许多问题，如程序的多入口多驱动点及例外子程序问题等。

b. 数据流准则　数据流分析在于建立程序中变量的关系。程序执行中变量的活动有三种方式：定义、引用、未定义。其可能出现的错误有：未定义而引用；定义和未定义；定义后再定义；定义但未引用。数据流测试即通过程序在一定路径上的执行，来发现变量活动序列的异常。数据流分析中有下述方法：

·用状态图表示变量异常。定义四种状态：U(未定义)，D(定义未引用)，R(定义并引用)，A(异常)。每一变量的初态为U。如图1，变量异常为ur，du，dd，这些错误可在程序执行中探测。

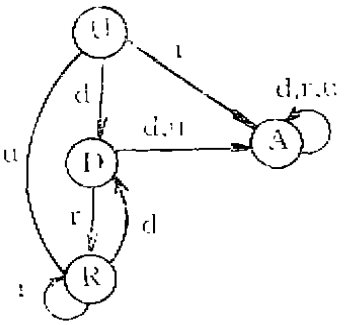

图1 变量状态图

·程序域的子域分界探测。如前所述，将程序域划分为若干子域后，分析这些子域的边界，找出产生正确或错误执行的测试数据。

·输入变量的追踪。每一输入变量的值经程序执行修改后最终作为程序的输出，它可在程序中分为定义、计算引用和谓词引用，故可对输入变量进行分析。

2.2 测试用例生成

测试过程自动化可大大减少软件测试开销，而测试用例生成是测试自动化的关键。这一技术的实现或部分实现，将大大改变靠人的直觉、经验产生测试用例的方法，而使软件测试的效率获得显著提高。国内外对于这一问题的研究均予以高度重视。

测试用例包括测试数据、期望结果、驱动程序和桩程序。测试数据生成一般分为三类：路径生成，即按语句、路径或分枝覆盖准则产生测试数据；规格说明生成，即按规格说明中定义的数据类型、域等生成，或按前面所述的属性文法生成；随机地产生测试数据。测试数据生成之后，再根据情况生成相应的桩程序或驱动程序。通常采用的是路径测试数据生成的方法，它又分为符号执行和程序直接执行两种。

符号执行即静态地分析给定路径的有关变量和谓词，对路径上的赋值语句的左部变量及分枝谓词的变量，依次用输入的符号表达式进行替换，使该路径的分枝谓词为有关输入的符号值的等式或不等式方程组，对这些路径的限制条件的方程组用线性规划的方法求解，即产生测试数据，使指定路径得以执行。但并不是所有的路径都是可执行的，故有些谓词方程组可能是不相容的。

符号执行对路径可采用向前遍历(替换)或向后遍历（替换）的方法，它们产生的谓词方程组是一样的。所不同的是，向前替换要记录中间的符号值。

但符号执行也有循环变量、数组下标不能确定，子程序调用等问题而使符号执行过

于复杂化（即组合爆炸）或不能执行，这些都尚待解决。

程序直接执行方式则根据有关信息直接产生测试数据，或在不充分的条件下，经过程序的多次循环探测，并对测试结果评价之后而最终确定测试数据。一般地，这种方法应首先产生输入变量的随机值。

2.3 测试的策略、步骤及工具开发动向

仅仅考虑单个模块的测试是不够的，故还要考虑多个模块的协调一致性，要进行集成环境下的测试。通常的测试策略为：

• 由底向上的模块测试。从程序（软件）的末端模块开始进行测试。其优点是不需要产生桩模块，而且驱动模块产生也较为容易。但测试初期缺乏对整个程序（软件）的整体认识。

• 由顶向下的模块测试。即从程序的开头进行测试，但要产生模拟被调用的下层模块的桩模块。其优点是以集成的而不是孤立的观点来进行程序测试。桩模块有时产生较困难，如桩模块不能模拟数据流；输入模块未加入时等困难。可遵循关键模块、输入输出模块优先加入的原则。

• 集成测试的目的在于软件系统的协作开发，主要根据需求和规格说明的功能进行测试。可先单元测试再集成测试；或先集成测试，再对那些发现问题的模块进行单元测试，这样可提高测试的效率。

软件测试一般分为静态分析、动测态试、

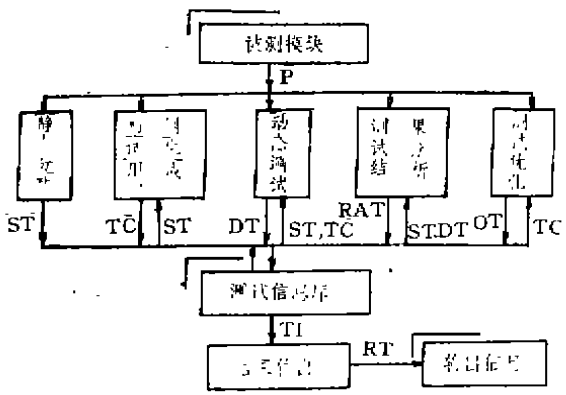

图2 软件测试的DFD图

测试用例生成、测试结果分析和测试优化等几个阶段。其DFD图如图2所示。

其中P：被测程序，ST，静态分析结果表，DT：动态分析结果表，TC：测试用例，RAT：测试结果分析表，OT：测试优化表，TI：测试信息表，RT：结果信息。

静态分析的任务是自动地分析被测软件，主要用于程序逻辑（如分枝、路径）和编码检查，一致性检查，接口分析，I/O规格说明分析，数据流分析，类型分析及单元分析，错误检查等，静态分析一般可检查出程序中30%～70%的错误。静态分析不执行被测程序。静态分析为动态测试及其他测试工具做必要准备。

测试用例生成，产生适当的可执行的测试用例，供动态测试使用。

动态测试进行结构和功能测试，包括覆盖分析、执行轨迹跟踪、定量检测、实时性能检测、性能模拟、资源使用情况分析、符号执行、断言检查及限制条件评价等。动态测试用插桩技术来记录程序在测试数据下运行的情况，并提供执行结果的动态信息。

测试结果分析即分析静态分析和动态测试产生的各种结果，确定测试准则（如覆盖准则）的满足程度，发现程序中错误的性质及其位置，协助用户控制测试过程，产生测试用例。测试结果分析通常有数据流分析（变量的定义、引用）和Slice（切片）分析（用简化的符号运算分析语义异常）。

测试优化。主要有优选测试用例，如用最少的测试用例达到最大的覆盖率；回归测试，对已修改过的软件进行复测，检查软件变动后的影响，辅助用户优化、挑选测试用例，达到最佳测试准则。

国内外在这些方面开发了许多工具。静态工具有：由Osterweil，L．J.和Fosolick，L．D.等研制的DAVE——FORTRAN程序的有效性错误探测和文档系统，由R．Walker开发的FACES——FORTRAN自动编码评价系统，以及由MCDonnell Douglas自动

化公司开发的JOYCE——FORTRAN程序的自动化静态分析器等。

动态工具有：Boyer,R. S.等人开发的SELECT——理想性测试的符号执行系统；Voges, U.等人开发的SADAT——静态和动态分析与测试系统；以及L. STueki研制的DYNA——FORTRAN动态分析器。

其他还有编译的测试数据产生器GEN-TEXTS，可产生PASCAL,SIMULA67等编译的测试；A. G. Duncan等研究的属性上下文无关文法可被用于产生任何0型语言的原理，亦可应用于任何0型语言的程序编译测试；R. A. Demillo等研制的TEC/1自动变异系统实现了FORTRAN的程序变异检测。

国内也有许多软件测试的开发与研究。北航承担的C语言软件测试工具CSTT及Ada软件质量保证系统ASQAS 均有静态分析、动态测试、测试结果分析等功能，ASQAS还有实时性能检测与死锁检测的功能。这两个项目均属国家"七五"攻关项目，并继续被列为"八五"项目，开展实用化和其他功能的开发。

北大率头研制的JB/WS集成化软件工程支持环境中的微机软件工具包JB/386，具有如C程序测试工具CPTT等众多的测试工具。

清华大学计算机系1989年研制的COBOL测试工具系统COSTE,利用弱变异等方法支持COBOL程序的测试。

三、软件测试的发展趋势

随着对软件开发要求的日益提高，传统的非形式化的软件测试技术已面临着挑战。传统的软件开发不能精确地表达软件开发的思想与功能，数学化、逻辑化或图形的高级抽象实现均较困难，其测试也是依赖于后端测试的。因此，传统软件工程尽管采用了阶段评审与测试手段，但依然不能显著提高软件质量。要使软件工程活动更加自动化，必须执行前端测试的原则，从需求规格说明设计一开始就实现形式化，以使验证和确认（V&V）方法渗入到软件开发的各个环节从而提高软件开发效率，保证软件产品的质量。形式化方法用代数语义、指称语义或操作语义形式化地定义软件功能及操作，实现由需求规格说明到设计继而到编码的转换，并可证明转换是正确的。不仅是程序而且程序设计及软件工程活动本身也应作为形式化的对象。事实上，除了软件设计与编码外，已经在其他软件开发中大量使用了形式化技术。例如CASE工具DFDM,可支持用户交互地画出有关问题的DFD图，并可自动检查数据流向、数据词典的数据，检查数据与说明是否匹配等；项目管理中项目的建立维护，术语的增、删，数据库管理等都使用了形式化方法。特别地，由于机器行为与程序语义之间的等价性，故可用程序语义的抽象，产生不受特定机器的细节影响的形式化通用方法。故我们可根据语义用CASE工具去检查规格说明与设计，由控制逻辑表达式的语义去模拟系统的举止，并进行证实等等。

特别要指出的是，形式化方法的所有构成技术与工具都是形式化的，并且这些方法与技术的使用均可被形式化地证明。

目前用于软件开发的形式化方法有VDM方法，Z方法和Petri网等。VDM方法规范具有对对象的语法域抽象，对象含义的语义域抽象，以及将语法对象映射到语义对象的操作抽象，即阐明函数；Z方法以集合论为基础，由公理表示对象的关系的图解是其主要特点。Z方法通过定义图解及对其进行不同组合的方式建立大型的规格说明；Petri网则是一种快速原型方法。例如用VDM或Z方法为模块中的每一项目准备规格说明，然后按照规格说明写出程序代码，再根据输入断言和定理证明编码的正确性。形式化方法具有以下性质：

- 良好的语义定义功能。
- 先进的过程描述，如何用启发式或清单的形式表达过程。

（转第66页）

的选择、评价方法来进行选择评价，另一方面则通过method域中的联想方法对可移动、可转动等功能特征进行联想，获取更深一层的设计约束和相关知识。重复上述过程，直到原型空间中所有叶节点均为基本型为止，这时系统中以设计语境的形式给出了设计结果的一个完整的原型描述。再调用原型落实操作，得到最后的设计结果输出给用户，同时将新的实例加入原型知识库中的"椅子"原型中（这步操作由学习器完成）。

五、结　束　语

从人工智能角度看，设计也是一类特殊的问题求解。目前在人工智能领域中，比较有代表性的问题求解策略有两种——基于模型的推理（Model-Based Reasoning，简称MBR）和基于事例的推理（Case-Basec Reasoning，简称CBR）。MBR使用由问题领域知识所构作的模型来进行推理，每一次求解都根据这个统一的模型从头做起，以前的求解经验不能为当前问题的求解所利用，因此适宜于解决新问题。CBR则基于以前的经验——事例，事例中包含有问题说明、解决方案等信息，在进行问题求解时，首先找出跟当前问题相关的事例，然后调整事例中的解决方案，使得它们能解决当前的问题[22]，因而适于求解一些频繁遇到的相似问题。然而，对于复杂的新问题，尤其是设计问题，MBR和CBR都无能为力，这是因为MBR随着问题复杂性的增加而变得不可控制，而CBR则因为事例的检索开销和方案调整的开销十分巨大，也是显得十分低效。原型中既包含有以前的设计经验——设计要素和设计实例，又限定了设计模式。在进行基于原型的推理时，一方面，原型限定的设计模式降低了设计问题的复杂性，另一方面又可根据原型中的设计要素和设计实例来利用以前的设计经验。可以说，基于原型的推理综合了MBR和CBR的优点。

在实际生活中，设计思维方式是多种多样的，人们对客观世界的看法也是多种多样的。即使对同一设计要求，不同设计师的设计结果也各不相同。因此，不存在一个标准的设计思维过程。但是，设计思维过程与设计风格是紧密相关的。具有同一种设计风格的设计师对特定领域中的"设计世界"的看法是基本一致的，他们的设计思维方式也大体相同[21]。而设计原型是对专业设计领域内种种设计要素的分类概括，原型中的联想、选择、评价方法所构成的设计推理机制描述了一类设计思维方式。因此，原型可以很好地模拟设计风格。

从知识工程角度来说，原型集知识表示及处理为一体，在结构形式上表现为一种面向对象的知识表示模式。随着面向对象技术的发展，常规性设计可望通过这样的系统实现完全自动化。而且，现有的AI技术如搜索、匹配、回溯、继承等可得到充分的应用。尽管在原型推理方法中隐含着联想、类比等正在探索之中的方法，但原型的引入明显地弥补了一直存在于常规性设计与创造性设计之间的断层。我们相信，随着对形象思维研究的深入[23]，在不久的将来一定能设计出一个真正具有创造性行为的ICAD系统。（参考文献共25篇略）

（接第71页）

• 利用V&V技术可对规格说明、概要设计及编码等正确性进行测试。

• 丰富的形式化支持工具。

形式化方法的使用对软件测试的自动化产生了直接的根本性影响。采用形式化方法，将使测试工具有如下特点：

• 测试用例自动生成，这一软件测试中最为重要的问题可望由形式化的方法的应用得以实现。

• 软件测试活动的自动化管理，包括测试定义，结果信息显示，测试库管理和回归测试的控制等。

• 测试过程自动化。如测试用例的选择，期望结果与实际结果的比较等。

软件测试尚待解决的问题是：如何寻找具有强语义的形式化方法？如何在软件开发的前端应用形式化方法与技术？开发何种支持形式化的测试工具？形式化方法是否支持原有的特定方法及语义？如何寻找既形式化又易于理解的测试工具？解决了这些问题，软件测试才有希望达到可行性、经济性、健壮性及灵活性的目标。（参考文献共12篇略）