软件测试工具的回顾与展望

郑国梁（南京大学） 王来强（华东工学院）

摘 要

本文阐述了软件自动测试工具研究的重要性和必要性。对测试工具的发展历程进行了综述，并展望了这一领域的发展方向和存在的问题，以及我们对上述问题的初步探讨。

如何保证软件质量，不仅计算机科学的理论工作者感兴趣，而且也受到软件工程的实际工作者的日益广泛重视。近十年来，人们提出了不少用于改善软件可靠性的技术和方法。总的说来，这些方法可分为软件测试技术和软件正确性的形式证明技术两大类。

关于测试的定义是：由人工或自动方法运用或评价系统或系统组成成分的过程以验证它是否满足规定的需求，或识别出期望的结果和真正结果之间有无差别。测试只能令人信服地表明错误的存在，而不能证明程序中不存在错误。

程序正确性的证明的确是一个鼓舞人心的想法，关于这个方面的研究已有不少成果。但是，至少有三个方面的问题使得这种技术在目前不能投入实用。

（1）形式证明所需要的数学严谨性，以及涉及的大量繁复工作，令绝大多数软件工作者不能接受。例如，有人曾对一个433个语句的ALGOL程序进行证明，其证明的长度竟达46页之多[2]，证明的复杂性远远超过了被证明程序的复杂程度。

（2）缺少形式证明所基于的良好的形式规格说明方法。

（3）程序正确性的形式证明并不能保证程序在实际运行中不失败[1][3]。

因而，测试技术仍然是相当一段时间内用以确认软件的最主要的方法。

软件测试就是为了发现软件错误而运行软件的过程。尽管人们投入了大量的金钱和时间于测试，可是软件产品的质量往往使人不满意，其原因何在？如何才能有效地解决这一问题？这是广大软件工作者所关心的问题，也是软件工程学的一个重要研究方向。

调查表明，只有不到5%的程序员受过有关测试方面的训练[1]。其结果是，大多数程序员在测试过程中把诸如“程序测试是证明程序中不存在错误的过程”，“程序测试的过程是使人们确信程序可完成预期要完成的工作的过程”等谬误作为自己测试的目的，造成了事倍功半的结果。软件的测试尤其是大型软件涉及的簿记工作令人厌倦易于出错，可靠性低。

软件工程学的研究表明，解决上述问题的最好方法是软件测试自动化[4]，所以开展软件自动测试工具的研究与开发是一项很有现实意义的工作。

软件测试费用的急剧上升（占整个开发费用的50%以上）[1]，迫使人们对软件测试工作做了些努力，迄今为止已有一批形式多样的测试工具问世。通常人们根据被测程序是否被执行而把测试工具分为静态分析（静态测试）工具和动态测试（动态分析）工具两大类。

1 动态测试工具

动态测试是人们最早使用的一种测试方

法。动态测试就是利用测试实例运行被测程序以发现程序错误的过程。动态测试由三个阶段组成：生成测试实例，运行程序，检验程序的输出结果。动态测试的关键问题是如何生成测试实例。常用的两种动态测试技术是插入探针（Probe）和解释执行（interpreter）。动态测试工具的代表作有ISMS，PET，和ACES等。

通过动态测试可以测出动态死代码，循环次数不正确等错误。而且动态测试工具负责收集、整理和显示用于检测错误的信息。以及进行动态断言校对。提供的信息分为执行总结统计，控制流和数据流信息，环境信息三类。

执行总结统计包括变量的值范围，语句的执行次数和时间。控制流和数据流信息包括变量被赋值的序列和语句执行的序列。环境信息包括标识符存取，参数传递和过程求值的环境等。

动态测试的最大优点是用户可以在实际环境中观察被测程序的性态。

用于上述信息都是通过具体的实例运行而获得的，因而它们不具有普遍性。动态测试的方法决定了它不能获取与程序结构有关的全局信息。所以动态测试本身不能有效地指导自身的执行。而且动态测试费用昂贵，因此人们对静态测试寄以厚望。

2 静态分析工具

静态分析不运行被测试的程序。基本方法是：企图证明一个断言（一类弱的软件系统定理）的真实性，如果断言是真的，那么断言所提到的性质在程序中不存在，否则程序有某种错误。

早期的静态分析工具是针对程序的语法结构而设计的，它们只能检查语法错误或语言标准之间的不一致性。通常把这种工具称为代码分析工具（Tools for Code Analysis）。这种工具目前已作为编译程序或语法制导的编辑程序的一部分。

随后出现的静态分析工具是把程序控制结构作为分析对象，这种工具被称为程序结构检查工具（Tools for program structure checks）。这种工具通过对程序控制流图的分析，识别错误的或有问题的控制结构，例如不正确的循环嵌套，未被引用的标号，不可到达的语句以及没有后继的语句等。PACE是这种工具的典型。

上述二种工具都局限于程序模块内部的检查，为此人们设计了所谓模块接口检查工具（Tools for proper module interface checks）。这种工具主要用于检查过程调用时实参与形参的个数及类型是否一致。随着程序设计语言学的发展，现代程序设计语言的编译程序一般都包括这些功能。

符号执行是一种特殊的静态分析工具，符号执行的结果可用来估计覆盖的程度，设计特定数据以遵循特定的执行路径；系统地探讨一组有意义的执行路径等。著名的符号执行工具有[3][5][6]。其实现技术不外乎向前扩展（forward expansion）和向后替换（backward substitution）这两种。符号执行提供了一种介乎于个别条件测试和通用的程序正确性证明之间的手段。

符号执行的固有缺陷表现在：

- 符号执行大型程序时费用很高；
- 对非数值型应用程序的符号执行产生的结果对错误检测作用不大；
- 对于输入数据不易用符号表示的程序无能为力（如，信号处理系统）；
- 对程序正文的执行不能查出代码级的错误。

上述原因大大限制了符号执行在测试领域的进一步运用。

以数据流分析技术为基础是现代静态分析工具的重要特征。由于数据流分析考虑程序中所有执行路径的各种事件序列，故它可以推断程序中存在或不存在某些类型的错误。

数据流分析技术最早用于代码的优化。虽然它在六十年代就已提出，不幸的是，人们一直没有发现它有优化之外的应用。

DAVE[7][8]的成功展示了静态分析工具的广阔前景。各种以数据流分析为基础的静态分析工具相继问世。

DAVE是美国Colorado大学于1974年在CDC6400上开发的一个FORTRAN静态测试工具。DAVE通过对程序中的数据流进行分析跟踪，可以查出程序中使用的无初值变量，子程序接口错，COMMON块中元素对齐错(alignment)。此外，DAVE还指出程序中可疑的代码，例如，变量赋值而没有被引用(reference)，变量被连续赋值，在入口处把常量传递给子程序的出口参数。与它以前的静态分析工具一样，DAVE假定所有的程序路径都是可执行的，这是DAVE的最大缺点。研究表明DAVE测出的错误有15%是发生在不可执行路径上。

为了进一步增加数据流分析工具的功能，扩大使用范围，人们在DAVE的基础上，从纵向和横向两个方面进行了努力。

横向的努力是把数据流分析与其它分析技术相结合，以期增加工具的能力。

ASPECT是日本电报电话公用公司(Nippon Telegraph and Telephone Public Corporation)为程序设计语言Ada设计的静态测试工具。与DAVE不同的是ASPECT在数据流分析的基础上加上了值范围分析，这样ASPECT不仅能查出与数据流有关的错误，还可以查出与值范围有关的错误，更主要的是ASPECT可以测出某些不可执行的程序路径，弥补了数据流分析技术的一大缺陷。因此ASPECT不仅增加了查错功能，而且提高了测试结果的可靠性。

纵向的努力是指对数据流分析技术本身进行更深入的研究，扩大和推广其应用领域。

这个方向的努力有两个着手点。一是把数据流分析技术从测试程序错误推广到测试系统错误。例如，用数据流分析技术来检测文件操作序列错等与操作系统有关的错误，[10]讨论了这类应用。另一个出发点是从语言的角度，现代程序设计语言提供了许多FORTRAN中没有的设施，这就要求数据流分析技术适应这些新的语言特点。例如，如何对指针变量进行数据流分析，如何检测由于并发而产生的程序错等，[9][10]在这一点上作了一定的努力。

此外，测试理论的研究表明，数据流分析还可作为测试数据的选取标准。

总的说来，静态分析工具至少可以给出这样一些信息

- 语法错；
- 使用没有初值的变量；
- 子程序接口不一致；
- 不可执行的程序路径；
- 死循环；
- 冗余语句；
- 变量赋值后没有被引用或被连续赋值；
- 语言标准之间的差异；
- 各种类型语句出现的次数；
- 标识符的交叉引用表；
- 标识符在语句中的作用；
- 过程调用图和控制流程图；
- 过程的控制结构复杂性。

然而，理论上的限制和实践中的困难，使得静态分析并不能代替其它工具（如动态测试工具的存在）。所谓理论上的限制是指程序路径可执行性判定问题，实践上的困难指的是不能区分数组元素和动态链表中的各结点[4]。

3 测试工具的发展方向

由于软件规模越来越大，复杂性不断上升，单一的测试工具已不能满足软件生产的要求。集成的软件测试系统的研究已成必然趋势。因此，人们在对以数据流分析为基础的静态分析工具进行更深入的研究同时，开始了集成测试系统的研究与开发。如何设计和开发集成的软件测试工具已是当前软件工程领域的一个重要研究课题。

静态分析和动态测试是两种互为补充的测试方法。故静态分析工具和动态测试工具是集成测试系统中不可少的两个工具。为了有效地管理信息，方便各工具之间的信息交

换，提高系统的整体功效，集成测试系统还应有一个公用数据库。至于系统中还应配置那些工具，各工具之间的相互关系如何，目前还没有一般的结论。不过有一点是肯定的，那就是集成测试系统的设计应在它本身将支持的测试方法学指导下进行。

目前发表的测试系统有SADAT[11]，RXVP[12]，IVTS[13]，MAP[14]等。

SADAT系统的支撑工具包括静态分析，动态测试，路径演算和符号执行四个组成部分。系统的运行由用户输入的命令决定。各工具之间通过数据库进行信息交换。符号执行主要用于测试数据的生成。SADAT的使用经验表明集成测试系统中静态分析和动态测试是两个很有价值的工具，而利用符号执行生成测试数据则开销太大。

SADAT的静态分析是基于数据流分析技术，因此有这类工具的通病即不区分路径的可执行性，其结果自然是查错功能不强，而且结果也不可靠。

RXVP，IVTS和SADT类似，所不同的是它们考虑了再测试问题，提供了建立文档的设施。

众所周知，当找出程序错误后，需要对程序做适当的修改。这需要编辑功能的支持。其次，对程序的修改过程中，常常带入一些新的错误，这就是所谓的"波动效应"(ripple effect)，据统计已测出的程序错误中有19%是由于"波动效应"的原因[2]。显然，提供良好的维护设施应是集成测试系统的一个设计重点，而现在大多数系统在这方面缺乏考虑。

就目前的工作状况来说，人机接口界面的设计还有待努力。近几年发展起来的Smalltalk-80的人机界面的许多思想和技术，特别是多窗口设施，菜单方式和Mouse定标已引起了广泛的重视，但目前所见到的系统大都是通过命令输入方式进行对话，不易使用。

总的说来，这方面的工作刚刚开始，能够实用的系统很少，工作尚处于探索阶段。

4 一个实例

在深入研究和分析现有的一些集成测试系统基础上，我们认为集成测试系统的设计至少应遵循下列准则：

- 提供静态分析和动态测试工具；
- 使用公用数据库管理系统信息；
- 提供一致的用户友好的接口界面；
- 支持良好的软件方法学；
- 提供有效的维护设施；
- 系统具有较高的集成度。

根据上述准则，我们开发了一个针对Modula-2语言[15]的用于辅助大型软件开发的交互式的集成的软件测试系统ITEM[16]。该系统通过有效的程序内部表示形式，把软件测试、理解和维护等多种功能高度地集成为有机整体，它对整个软件生命周期都是一个有价值的软件确认和维护工具。

ITEM系统的人机界面是在面向用户方法学的指导下设计的，用户通过多窗口，多层次菜单和Mouse等先进的交互式手段操作ITEM，系统易学，易懂，易用且灵活有效。

ITEM的核心是一个基于控制流分析、数据流分析和值范围分析等多种技术的静态分析工具箱。它具有较强的查错能力，它可以识别不可执行路径，从而克服了一般静态分析工具查错结果不精确的通病，减少了测试误差，提高了结果的可靠性。ITEM可测出的数据流错误包括引用无初值的变量，变量被赋值而未被引用或被连续赋值；与控制流有关的错误有不可到达的语句，死循环，冗余语句等；与值范围有关的错误有数组下标越界，除数为0和引用值为空的指引元等。

此外，ITEM系统还提供了波动分析，观察程序正文结构和控制结构的设施。简单的编辑功能和漂亮格式加工程序以及文档生成设施使系统可作为软件理解和维护工具。

目前软件测试的理论还不十分成熟，还处于一种技巧阶段，使其从技巧上升到科学的高度将是广大计算机科学工作者努力的目标。为此有必要对现

有的软件测试技术和工具从理论上进行归纳总结，并在此基础上从理论的角度出发对软件测试方法学进行有益的探索.与此同时还要从工具的角度出发，在实践中验证方法的可行性和有效性。

参考文献

1 Beizer, B., Software Testing Techniques.

2 Ramamoorthy, C. V., and Ho, S. F., Testing large Software with automated Software evaluation Systems, IEEE Trans. Software Eng., Vol. SE-1, NO.1, March 1975.

3 Clarke, L. A., A System to Generate Test Data and Symbolically Execute Programs, IEEE Trans. Software Eng., Vol. SE-11, No.3, March 1985.

4 Miller, E., Program testing, Art Meets Theory, Computer, Vol. 10, No. 1, January 1977.

5 King, J.C., Symbolic execution and program testing, CACM July 1976.

6 Howden, W.E., Experiments with a symbolic evaluation System, Proc.1976 Nat. Computer Conf., June 1976.

7 Osterweil, L. J. and Fosdick, L. D., DAVE—A Validation Error Detection and Documentation System for Fortran Program, Software Practice and Experience, Vol.6, Oct.—Dec. 1976.

8 Osterweil, L. J. and Fosdick, L. D., Some Experience with DAVE—A Fortran Program Analyzer, in AFIPS Conf. Proc., Vol. 45, Montvale. NJ, AFIPS Press, 1976.

9 wilson, C. and Osterweil, L. J., Omega—A dataflow analysis tool for the C programming language,IEEE Trans. Software Eng., Vol. SE—11, No. 9, September 1985.

10 Richard,N.T. and Osterweil, L. J., Anomaly Detection in Concurrent Software by Static Data Flow Analysis IEEE Trans. Software Eng., Vol. SE—6, No.3,May 1980

11 Voges,U., Gmeiner, L. and Mayrhauser, A. A. V, SADAT—An Automated Testing Tool, IEEE Trans. Softwre Eng., Vol. SE—6,No. 3,May 1980.

12 Senn, E. H., Ames, R. R., and Smith, K. A., Integrated Verification and Testing System(IVTS)for HAL/S Program, in 1983 SOFTFAIR.

13 Andrews, C. L. and Dehaan, W.R., RXVP80 The Verification and Validation System for Fortran and COBOL,in 1983 SOFTFAIR.

14 Tischler, R., Schaufler, R. and Payne, C.,Static Analysis of Programs as an Aid to Debugging. SIGPLAN Notices Vol. 18, No. 8, August, 1983.

15 With, N., Programming in Modula-2, Springer—Verlag, 1984.

16 王来强 ITEM：一个集成的软件测试系统 南京大学硕士论文 1987.

（上接52页）

of the ACM, Vol.12, №.10, 1969.

〔6〕 Guttage, J.U. & Horowitz, E. &Musser, D.R.: "Abstract Data Types and Software Validation", Comm.of the ACM, Vol. 21, №.12, 1978.

〔7〕 Flon, L. & Misra, J.: "A Unified Approach to the Specification and Verification of Abstract Data Types." Proc.Specification of Reliable Software Conf., Cambridge, Mass., 1979

〔8〕 Hoare,C.A.R., "Proofs of Correctness of Data Representations", Acta Informatica, Vol.1, №.4, 1972.

〔9〕 Formal Methods of Program Verification and Specification, PRENTICE-HALL, INC., Englewood Cliff, New Jersey 07632.

〔10〕 Proceedings: 9th International conference on Software Engineering March 30-April2, 1987, Monterey, California, USA. Computer Society Press of IEEE.

〔11〕 Algebraic Approaches to Program Semantics, 1986, Springer-Verlag New York Inc.