一种基于程序切片技术的软件测试方法

A Software Test Approach Based on Program Slicing Technique

李必信[1] 方祥圣[2] 袁 海[1] 郑国梁[1]

(南京大学计算机软件国家重点实验室 南京210093)[1](安徽经济管理学院 合肥230055)[2]

Abstract It is well acknowledged that quality of software has a higher priority than the performance and functions of software. One of the ways to get high-quality software is to get more efficient software-testing techniques. Theory and technology of software quality assurance are an important part of software developing methodology and software engineering. Software testing plays a key role in software quality assurance. The purpose of the essay is to search for new software testing method and to solve some problems in testing of object-oriented program. We also try to amend some deficiency in the traditional test method for structured programs. By the idea of program slicing, we can disassemble the source code of a program into several slices following certain rules. Instead of testing the whole program, we can test these slices. We can also guarantee the equivalence of the two ways. Testing on the base of program slicing has several advantages than the one simply using data flow analysis and control flow analysis. The first, because a program equals to the union of its slices, to test all of the slices makes a complete test of the program, and to test each slice which is related to the interested variables is actually a complete test of the requirement test. Then we solve the problem of sufficiency in traditional structured program testing and object-oriented program testing as well. The second, program slicing technique can be applied to the testing of both structured programs and object-oriented ones.

Keywords Software testing, Object oriented, Program slicing

1 引言

软件测试是人们发现、纠正、预防软件错误以及完善软件功能的重要手段[1]。软件测试的目的就是为了发现程序中的错误。对于传统程序设计语言书写的软件，软件测试人员普遍接受三个级别的测试：单元测试、集成测试和系统测试。无论在哪个级别上进行测试，其测试过程均为输入测试数据、处理和验证输出结果三个步骤。目前面向对象软件开发技术发展迅速，但面向对象软件测试技术的研究还相对薄弱。例如，对面向对象的程序测试应当分为多少级尚未达成共识。基于结构的传统集成策略并不适于面向对象的程序。这是因为面向对象的程序的执行实际上是执行一个由消息连接起来的方法序列，而这个方法序列往往是由外部事件驱动的，在面向对象语言中，虽然信息隐藏和封装使得类具有较好的独立性，有利于提高软件的易测试性和保证软件的质量，但是，这些机制与继承机制和动态定连给软件测试带来了新的课题，尤其是面向对象软件中类与类之间的集成测试和类中各个方法之间的集成测试具有特别重要的意义，与传统语言书写的软件相比，集成测试的方法和策略也应该有所不同。从目前的研究现状来看，研究较多地集中在类和对象状态的测试方面。面向对象程序设计的继承和动态定连所带来的多态性对软件测试的影响，文献中虽然有所论及，但是不仅缺乏针对这一特点的测试方法，而且还有许多问题有待进一步的研究。软件测试中的另一个重要问题是测试的充分性问题，朱鸿博士[2]证明了充分性准则对软件测试的揭错能力具有重要影响。对传统语言的软件测试已经存在多种充分性准则，但对面向对象的软件测试，目前尚无普遍接受的充分性准则。对这些方面的深入研究将会产生真正对软件测试的理论与实践有指导意义、有影响的成果。

2 传统测试技术分析

2.1 传统测试技术概述和分类

软件测试包括静态测试和动态测试两种，前者的基本特征是在对软件进行分析、检查和测试时不实际运行被测试的程序。动态测试的基本特征是通过运行软件来检查软件的动态行为和运行结果的正确性。因此，所有动态测试都必须包括三个基本要素：被测试程

序、测试数据、软件需求和规约[1]。根据在软件开发过程中所处的阶段及其作用，动态测试可分为以下几类：单元测试、集成测试、系统测试、验收测试和回归测试。其中回归测试是在软件维护阶段，对软件修改之后进行的测试。其目的是检验对软件进行的修改是否正确。这里，修改的正确性有两重含义。一是指所作的修改达到预定的目的；二是指不影响软件其他功能的正确性。

传统的软件测试方法分为黑盒测试和白盒测试。按照选择或产生测试数据所根据的信息来源，软件测试方法可分为以程序为基础的测试、以需求和功能规约为基础的测试、程序和需求相结合的测试、以界面为基础的测试四种。按照选择或产生测试数据及判断测试充分性的方法，软件测试方法可以分为结构性测试、排错性测试、分域测试、功能测试。

一个成功的测试包括两个主要方面：一是软件在所有(或足够多的)测试数据上是正确的；二是测试数据是充分的，即软件在测试数据上的表现能够充分反映软件的总体表现。检查软件在测试数据上的行为的正确性称为测试的"先知者问题"。判断一个测试数据集合是否充分称为充分性问题。

2.2 结构性测试技术的缺陷分析

较成熟的结构性测试方法有以程序为基础的控制流测试和数据流测试。主要包括语句测试、分支测试、路径和定义—引用关系测试等。在结构性测试技术中，程序结构的流图被作为程序结构的一种模型，该模型奠定了结构性测试基础。

但是，实际维护工作中，我们经常遇到这样的问题：对一个软件进行大规模测试后，对软件进行了某种小的修改，这种修改势必影响到程序的其他部分，如果采用上面提到的传统方法，必须重新对程序进行大规模测试。为此，我们从程序切片的角度考虑这个问题：首先找到程序新旧版本之间的不同之处。比较它们的切片和依赖图，对那些具有相同切片的结点不用考虑，将那些在新版本依赖图中出现，而在旧版本依赖图中不出现的结点标记出来，称为"影响点"，计算它的静态切片和前向切片，并取两者的交集。这样，如果要对程序进行重新测试，仅需对此交集进行测试，因而减少了工作量。同样，在程序测试过程中，最常见的工作莫过于发现一个错误并找出所有与错误有关的语句，利用程序切片技术可容易做到这一点。例如使用动态切片，可以根据输入得到语句数比源程序少得多的切片集，使程序的错误定位比原来容易。在大型软件项目的调试过程中，经常会遇到对某些输入产生正确结果，对另一些输入则产生错误结果的情况。如果采用多次运行程序，逐条语句跟踪的测试方法，将会耗费大量的时间。而采用程序切片技术，我们可以构造输入语句的前向切片和输出语句的动态切片，并取两者的交集，从而极大地缩小考察的程序范围。

3 基于切片的软件测试方法

基于切片的软件测试、调试及维护的研究已经引起人们的注意，其中研究较多的有 M Kamkar，J. R.-Lyle 和 M. J. Harrold 等，他们的研究成果可参见文[2～7]。我们在这里主要讨论了基于切片的软件测试的一般原理、方法和性质，同时给出了此类软件测试工具设计和实现的基本思想。

```
(1) read(n);
(2) i:=1;
(3) sum:=0;
(4) product:=1;
(5) while i<=n do
    begin
(6)       sum:=sum+i;
(7)       product:=product*i;
(8)       i:=i+1;
    end
(9) write(sum);
(10) write(product);
```

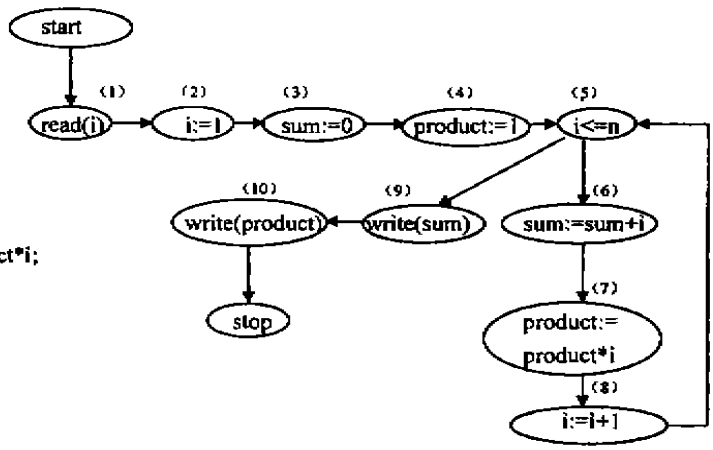

图1 一个例子程序 P(左)及其 CFG(右)

3.1 程序切片的基本知识

程序切片有两类定义：1)Weiser 定义的切片是一个可执行的程序，是从源程序中移去零条或多条语句来构造的；2)另一种定义是程序中语句和控制谓词组成的一个子集，这些语句和控制谓词直接或间接影响在切片准则计算的变量的值，这类切片不必构成可执

行的程序。

为了计算程序切片，必须考虑各种流依赖和控制依赖，因此先讨论一下数据依赖和控制依赖是必要的。数据依赖和控制依赖是根据程序的控制流图(CFG)来定义的。CFG 中包含的结点表示所有语句和谓词，从结点 i 到结点 j 的一条边表示从前者到后者的可能的控制流。CFG 中包含的分别标记为 Start 和 Stop 的特殊结点表示程序的开始和结束。集合 Def(i)和 Ref(i)分别表示在 CFG 的结点 i 定义和引用的变量。

数据依赖有流依赖、输出依赖和反依赖等。根据是否被作为迭代的结果，流依赖能够进一步分为带循环的依赖和不带循环的依赖两种。但为了切片的目的，可不必区分有循环的流依赖，考虑流依赖即可。直观地讲，如果在程序的某个执行过程中，在 i 计算的值在 j 被使用，则称语句 j 流依赖于语句 i，在没有别名的情况下，可以形式地定义流依赖如下：

定义1(数据依赖) 存在一个变量 x 使得：①$x \in REF(i)$；②在一条从 i 到 j 的路径上没有插入对 x 的另外定义。则称语句 j 流依赖于语句 i，也就是说，x 在结点 i 的定义是为了结点 j 的一个到达定义(reaching definition)。

控制依赖通常是根据后控制关系来定义的，如果所有从 i 到 Stop 的路径 Path 都经过 j，则称 CFG 中结点 i 是后受制于 j。

定义2(控制依赖) 称一个结点 j 控制依赖于结点 i，如果存在一条从 i 到 j 的路径 Path 使得 j 后控制 Path 中除 i 和 j 之外的每个结点，i 不受 j 的后控制。

3.2 基于程序切片的软件测试

基于程序切片的软件测试是一种以程序或程序和需求相结合为基础的测试，它根据计算程序的不同切片来缩小软件测试的范围，并提高软件测试的效率。同时，由于程序切片考虑程序存在的各种依赖关系(而不仅仅是数据依赖和控制依赖)，所以使得测试的准确性得到提高。另外，任何一个程序可以和一组程序切片的并集等价，测试每个切片实际就是测试了整个程序，从而保证了测试的充分性。下面，我们详细介绍利用切片技术进行软件测试的基本过程，以图1中程序 P 为例。P 有两个输出变量 sum 和 product，以 sum 和 product 为切片变量来分别计算切片，可以得到如下两个切片 S(9,sum)和(10,product)：

```
(1) read(n);                    (1) read(n);
(2) i:=1;                       (2) i:=1;
(3) sum:=0;                     (3) * * * * * * * *;
(4) * * * * * * * *;            (4) product:=1
(5) while i<=n do               (5) while i<=n do
  begin                           begin
(6)   sum:=sum+i;               (6)   * * * * * * * * * * * * * *;
(7)   * * * * * * * *           (7)   product:=product*i;
(8)   i:=i+1;                   (8)   i:=i+1;
  end                             end
(9) write(sum);                 (9) * * * * * * * *;
(10) * * * * * * * * * * * *    (10) write(product);
```

图2 程序 P 关于切片准则(9,sum)(左)和(10,product)(右)的切片

根据程序切片的定义，S(9,sum)由程序 P 中所有与 sum 有关的语句和谓词组成，故 S(9,sum)={1,2,3,5,6,8,9}。同样，S(10,product)={1,2,4,7,8,10}。P=S(9,sum)∪S(10,product)。

现在，如果我们在维护过程中需要对源程序进行某些修改，例如假设修改了 sum 的初始值，为了找出这种修改可能影响到的语句，我们不必对源程序进行分析，而只要分析切片 S(9,sum)就可以了。可见，分析的范围缩小了，效率提高了。

基于切片技术进行软件测试的基础模型是各种依赖图。如果对方法进行测试，可以利用方法依赖图(MDG，Method Dependence Graph)；如果对类进行测试，可以利用类依赖图(ClDG，Class Dependence Graph)；如果测试控制依赖关系，可以利用控制依赖图(CDG，Control Dependence Graph)；如果测试数据依赖关系，可以利用数据依赖图(DDG，Data Dependence Graph)；如果测试消息依赖关系，可以利用消息依赖图(MeDG，Message Dependence Graph)等等。下面以数据依赖图为例来讨论这种测试的相关特性。

定义3(切片变量) 把程序的所有有意义的输出变量都称为切片变量。有意义是指：对这些输出变量计算切片是有意义的。例如图1中的 SumN 和 ProdN 是切片变量，而 N，I 等就不是切片变量。

定义4(数据依赖图 DataDG) 数据依赖图是一个二元组(D，E)，其中结点 D 是程序中所有数据的集合(包括变量，常量的定义和引用等.)，E 表示结点之间的流依赖(包括定义引用，赋值，值传递等)和控制依赖关系。如图3所示。

图3中每个结点中三元组元素(n，v，o)表示数据，其中 n 表示语句号，v 表示某个数据标记，o 表示数据 v 的第几次出现。例如，三元组(3，sum，1)表示数据 sum 第1次出现在第3条语句中。数据依赖图的构造和

数据切片的计算可参见文[1]。

如果我们在第6行对 sum 进行了某种修改，则计算〈6，sum〉的前向切片，如图4所示。阴影部分表示切片准则〈6，sum〉前向切片。

切片准则〈6，sum〉的静态数据切片和前向数据切片的交集就是关于切片准则〈6，sum〉的前向数据切片。我们只需测试程序的第6行和第9行就可以了。而实际上对第6行中 sum 的某种修改确实只影响第9行的 sum，而对其他地方没有任何影响。

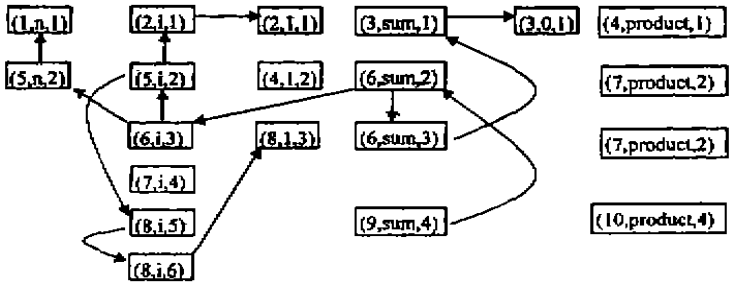

图3　图1中程序 P 的一个数据依赖图及其关于切片准则〈6，sum〉的静态数据切片

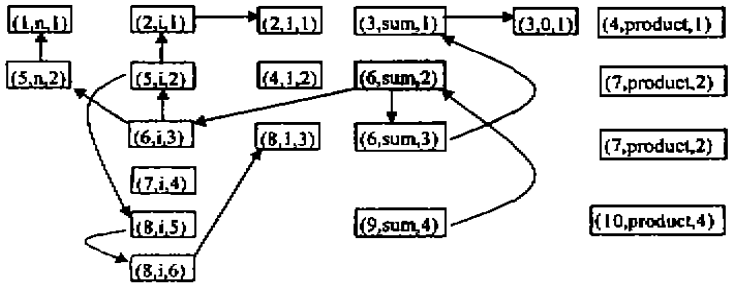

图4　程序 P 关于切片准则〈6，sum〉的前向数据切片

3.3　基本性质

令 P 表示某个程序，S(n，V，static)表示关于切片准则〈n，V，static〉的静态程序切片，S(n，V，dynamic)表示关于切片准则〈n，V，dynamic〉的动态程序切片，S(n，V，forward)表示关于切片准则〈n，V，forward〉的前向程序切片，S(n，V，backward)表示关于切片准则〈n，V，backward〉的后向程序切片。

定义5(分解切片)　如果从程序 P 可以根据切片变量计算切片 S(n，V_1，X)，S(n，V_2，X)，…，S(n，V_n，X)等，其中 X 表示 static 或 dynamic。令 D 为分解切片，则 D 满足：

$$D=P-\bigcup_{i=1}^{n}S(n,V_i,static)\quad 或$$

$$D'=P-\bigcup_{i=1}^{n}S(n,V_i,dynamic)$$

性质1(覆盖定理)　任何一个程序都可以分解为一组程序切片和分解切片的并。即

$$P=(\bigcup_{i=1}^{n}S(n,V_i,static))\cup D\quad 或$$

$$P=(\bigcup_{i=1}^{n}S(n,V_i,static))\cup D'$$

性质2　测试程序 P 等价于测试程序 P 的每个程序切片和分解切片。也就是说，如果对程序的每个切片都进行了测试相当于对该程序进行了充分性测试。

由性质1知，任何一个程序都可以分解成一组程序切片的并，而这些切片都是根据某个切片变量和切片准则计算出来的，对某个切片变量修改不会影响到其他切片变量的切片。因为，根据切片的定义，所有能够影响到的语句、谓词等都被包含到该切片变量的切片中。

性质3　维护测试(在维护过程中对程序作了小小的修改，然后进行的测试)只需对相关切片变量的静态切片和前向切片的交集进行测试即可。如图3，4中的例子。

性质4　如果 $P=(\bigcup_{i=1}^{n}S_i)\cup D$，且 $\sum_{i=1}^{n}S_i\cap D=\phi$，$S_i\cap S_j=\phi(i\neq j)$，则程序 P 具有切片独立性。这时，程序 P 可以切片划分成 S_1，…，Sn 和 D。如果 $D=\phi$，则 P 具有严格的切片划分，这种程序的测试具有如下特点：测试整个程序 P 只需分别测试每个切片即可，且这种测试方式保证了测试的完备性和充分性。

性质5　令 # 为取元素个数的运算，例 #(1，2，3)

=3，则#(S(n,V,static)∩S(n,V,forward))≥1，#(S(input,V,static)∩S(output,V,forward))≥1。其中S(input,V,static)表示输入语句的静态切片，S(output,V,forward)表示输出语句的动态切片。

4 软件测试工具的设计和实现

4.1 软件测试的基本步骤

测试的结果为了找错，找到错误后的行动即为排错，并最终使整个系统正确。通常人们依据测试程序是否被执行而把测试工具分为静态分析和动态测试两种。单一的测试工具具有其自身的局限性，只能测出某几类错误，而随着软件的规模越来越大，复杂性越来越高，单一的测试工具越来越不能满足软件生产的要求，集成测试系统的研究已成必然趋势。

软件测试的基本步骤：1)测试用例生成；2)查找和定位错误；3)调试程序；4)回归测试。

4.2 软件测试工具的设计思想和功能描述

该测试工具设计的基本思想是：(1)利用黑盒测试方法确定输出结果和期望结果，检查有没有错误发生，如果发现错误，则利用程序切片技术来定位错误，确定错误的影响范围，从而达到排除错误和修复程序的目的。(2)对Java源程序按数据级、方法级(过程级)分析各种依赖关系和流关系。(3)在不同层次上进行数据流和控制流及依赖关系分析，建立各种流图和依赖关系图。(4)利用分层切片算法或一般程序切片算法(如图形可达性算法和两阶段图形可达性算法等)结合各种切片准则计算各个层次的切片，以便确定引起错误的根源和错误的影响范围。(5)排错和修复程序，进行回归测试，直至没有发现错误为止。(6)利用报告生成模块自动生成测试报告和相关技术文档。

该测试工具由以下几个模块组成：黑盒测试模块、切片获取模块(包括词法语法分析，构造抽象语法树，生成各种依赖图以及计算切片等功能)，测试模块(包括测试用例生成，静态分析和动态测试等功能)，错误发现、定位、排错和修复模块，回归测试模块以及报告生成模块等，如图5所示。其中：

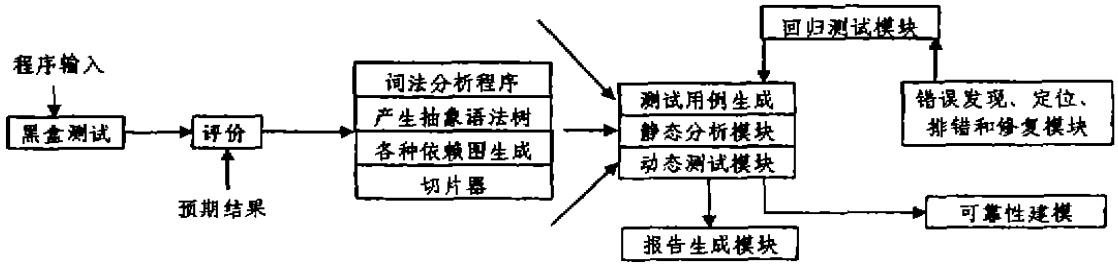

图5 基于切片技术的软件测试流程图

黑盒测试模块：确定输出结果和期望结果，来检查有没有错误发生。

切片获取模块：用于计算程序的各种切片。其中：1)词法和语法分析：目的是检查源程序的词法和语法错误。2)产生抽象语法树：利用Lex/Yacc生成抽象语法分析树。3)各种依赖图生成：利用抽象语法树构造各种依赖图。4)切片器：用于产生各种程序切片。

测试模块：基于切片的结果测试程序。其中：1)测试用例生成模块：根据各程序段，模块或子系统入口处的临界条件设计和生成各种情况下的测试用例。2)静态分析模块：利用静态程序切片技术，理解被测软件的基本结构，使用户能更好地对被测试软件进行分析和测试。该模块主要融入各种静态分析手段和方法，其中包括：①数据流分析；②控制流分析；③依赖关系分析；④分析函数的各种静态调用关系以及对全局变量的引用等。3)动态测试模块：利用动态程序切片技术，根据用户的输入分析和测试程序的各种数据结构的正确性和合理性，测试各模块功能的实现情况、测试模块组装以后形成的较大模块(或子系统)的功能实现情况和完成指标，测试系统的运行情况以及适应环境的情况等等。它在被测试源程序被选定的函数的各个结点前插入一个探头函数，该探头函数的作用是搜集动态测试中的统计信息，并不影响原函数的运行，并且能够将这些信息以文件的方式存下来，让用户通过该工具提供的阅读功能了解这些统计信息。

错误发现、定位、排错和修复模块：根据程序段、模块、子系统、系统的输出来发现错误，利用后向切片技术定位错误，用前向切片技术界定错误的影响范围，然后排错并修复程序。

回归测试模块：对程序修改以后，再进行回归测试。因为对程序的任何修改都可能引入新的错误，用以前测试用的用例进行回归测试，有助于发现由于修改程序而引入的新错误。回归测试要反复进行直至没有发现错误为止。

报告生成模块：生成各种类型的测试情况报告，帮

(下转第112页)

此，对元系统的节点管理就更显困难。在设计中，每个节点的安全性必须得到加强，至少不能破坏或降低节点原有的安全特性。

·支持高效容错功能：设计元计算系统应该考虑系统的容错功能、系统例外处理功能和软件调试与排错功能。

·支持统一资源管理：由于异构环境的节点各式各样，元系统必须支持多种体系结构，在元系统中，最好没有中心节点，整个系统是全分布的。通过中间件来保证异构硬件和软件的互操作性，进行资源的统一管理和调度。

·支持应用程序在元系统上合理调度：在元系统中，除了考虑指派任务给机器之外，还要考虑机器的异构特性，实现任务有选择的映射和调度。一个应用程序分解的子任务间往往存在依赖关系，依赖任务在异构机器上的调度是元系统设计的重点。

结论 元计算是一个相对较新的研究领域，开发一个高效、好用的元计算系统还存在一定的困难。一个元计算系统必须和同构系统一样，具有友好的人机交互界面、统一的编程环境、可靠的通信协议和高效的任务调度算法，并提供与流行的并行编码软件如 PVM 的接口。开发一个基于校园网络的元计算系统是我们的第一研究目标。

参 考 文 献

1 Grimshaw A，Ferrari A，et al. Metasystems. Communications of the ACM，1998，41(11)：46～55

2 Smarr L，Catlett C E. Metacomputing. Communications of the ACM，1992，35(6)：44～52

3 Messina P，Culler D，et al. Architecture. Communications of the ACM，1998，41(11)：36～45

4 Freund R F. Heterogeneous Processing. Computer，1993，26(6)：13～17

5 DeFanti T A，Foster I，et al. Overview of the I-WAY：Wide area visual supercomputing. International Journal of Supercomputer Applications and High Performance Computing，1996，10(2/3)：123～130

6 Foster I，Kesselman C. Globus：A metacomputing infrastructure toolkit. International Journal of Supercomputer Applications and High Performance Computing，1997，11(2)：115～128

7 Grimshaw A S，Wulf W. The Legion Vision of a Worldwide Virtual Computer. Communications of the ACM，1997，40(1)：39～45

8 Steen M，Homburg P，et al. The Architectural Design of Globe：A Wide-Area Distributed System：[Internal report IR-422]. Vrije Universiteit，March，1997

9 Foster I，Kesselman C，et al. Nexus：Runtime Support for Task-Parallel Programming Languages，Argonne National Laboratories. http://www.mcs.anl.gov/nexus/paper/

10 Grimshaw A S. Easy to Use Object-Oriented Parallel Programming with Mentat，IEEE Computer，May，1993. 39～51

（上接第101）

助测试和调试人员分析错误结果和原因，为进一步修改和完善软件提供技术支持。

结束语 基于切片技术的软件测试是一种全新的软件测试技术，它把对整个程序的测试转化为只对某个程序切片的测试。与传统的测试方法相比，它具有效率高、准确性好等优点。基于切片的测试用例设计与切片准则有关，测试模块内切片也就是进行单元测试，测试用例的设计只与该模块有关；测试模块间切片也就是进行组装测试，测试用例的设计必须考虑到模块之间的联系；测试软件体系结构切片也就是进行确认方面的测试，测试用例的设计须符合需求分析的要求；接口测试（系统测试）可通过计算接口切片的方法进行相关测试，当然是通过接口描述语言规约切片以后进行测试。程序切片可用于错误查找，定位。

参 考 文 献

1 朱鸿，金凌紫. 软件测试和质量保障技术. 科学出版社，1997

2 Podgurski A，Clarke L. A formal model of program dependence and its implications for software testing，debugging，and maintenance. IEEE Transactions on Software Engineering，1990，16(9)：965～979

3 Harman M，Danicic S. Using program slicing to simplify testing. Software Testing，Verification and Reliability，1995，5：143～162

4 Kamkar M. Interprocedural Dynamic Slicing with Applications to Debugging and Testing：[PhD thesis]. Linkoping University. S-581 83 Linkoping，Sweeden，1993

5 Kamkar M，Fritzson P，Shahmehri N. Interprocedural dynamic slicing applied to interprocedural data flow testing. In：Proc. of the Conf. on Sofrware Maintenance -93，1993. 386～395

6 Gupta R，Harrold M J，Soffa M L. An approach to regression testing using slicing. In：Proc. of the IEEE Conf. on Software Maintenance，1992. 299～308

7 Bates S，Horwitz S. Incremental program testing using program dependence graphs. In：Conf. Record of the Twentieth ACM Symposium on Principles of Programming Languages，ACM，1993