面向组件的软件测试技术
Testing for Component-based Software

白晓清[1,2],蓝秋萍[1],李怀忠[1]
Bai Xiaoqing[1,2],LAM C. Peng[1],Li Huaizhong[1]
(1. School of Computer and Information Science, Edith Cowan University, Mt. Lawley, WA 6050, Australia; 2. 广西大学电气工程学院,广西南宁 530004)
(1. School of Computer and Information Science, Edith Cowan University, Mt. Lawley, WA 6050, Australia; 2. College of Electrical Engineering, Guangxi Univ., Nanning, Guangxi, 530004, China)

摘要:使用组件,特别是第三方商业组件集成开发而成的软件越来越多,面向组件的测试技术的需求越来越大。UML是目前软件开发业应用最广泛的可视化标准建模语言,它能应用于整个软件开发过程的各阶段。UML的许多特点与面向组件的软件测试需求相吻合。基于UML的软件测试技术在对面向组件的软件开发具有广阔的研究和应用前景。

关键词:软件 测试技术 组件技术 UML
中图法分类号:TP311.5

Abstract: It is a trend to develop software systems using components, especially using Commercial Off-The-Shelf (COTS) components. Such a trend calls for the development of component-based software testing techniques, since the traditional testing techniques can not be fully adapted to test component-based software systems. The UML has emerged as an industrial standard for modeling software systems, and can be used across the whole development lifecycle of a software system. An overview of current works on UML-based software testing for component-based software development is presented. It is shown that the UML-based software testing technique can be widely used in both the academic research and the industrial applications.

Key words: software, testing, component techniques, UML

随着人们对软件质量和生产速度要求的不断提高,软件开发方法和技术的更新成为当今软件工程界的热门研究课题。目前基于组件的开发方法和技术是从面向对象的开发方法发展而来的,其特点是能够实现软件的大粒度复用,"即插即用",缩短软件开发周期,降低维护成本[1,2]。众所周知,软件测试是软件开发过程的重要组成部分,大部分软件开发公司或机构花费在软件测试阶段的费用占总开发费用的40%以上[3]。基于传统的面向过程和面向对象的软件测试技术已被深入研究和应用[4]。面向组件的软件测试技术,特别是基于统一建模语言(Unified Modeling Language,UML)的组件测试技术是目前软件测试界的研究热点,本文对此进行详细的比较和研究。

1 组件技术

面向组件的开发技术是从面向对象的开发过程发展而来。组件和对象虽有相似之处,也存在差异。因此,面向组件的软件测试技术与传统的面向对象软件测试技术有所不同。

1.1 组件的基本概念

目前软件工程界对组件没有统一的定义,我们不妨这样理解"组件"[1]:组件是带一定独立功能的,独立发布的,遵循某个组件标准的,支持第三方集成的二进制软件单元。它有如下特点:(1)即插即用。组件能方便地由第三方集成于软件系统中,不需要修改代码和重新编译;(2)以接口为核心。组件的接口和实现分离,通过接口实现与其他组件或系统交互,具体的实现被封装在组件内部,系统组装者只关心接口,不必知道实现细节;(3)标准化。组件的接口被严格标准化,这实际上是组件技术成熟的标志之一。

2003-04-21 收稿,2004-02-11 修回。

目前主要组件标准有 Microsoft COM/DCOM，Java 的 JaveBeans 和 EJB，以及 OMG 组织的 CORBA。(4)组件市场和组件库。在全球化的组件市场中或由开发人员开发积累的组件库中有大量成熟组件，软件系统组装者能从中挑选合适的组件。组件提供者对组件的拥有版权，对组件的质量负责，并且提供详细的组件使用文档。

因此，组件比面向对象技术中的对象，具有更高的使用度，更灵活的生产方式，更容易理解和分发。组件分为开发人员根据特定的组件标准专门开发或把原有的程序代码封装而成的内部组件和由独立的第三方机构开发的商业组件两种。

1.2 基于组件的软件开发过程

基于组件的软件开发过程与传统的面向过程或面向对象的软件开发过程有所不同。传统的软件开发过程是围绕用户需求进行需求分析和设计，软件完全根据用户的需求定制，称之为基于需求的软件开发过程[3]。而基于组件的软件开发过程，由于通常从商业市场上购买的组件并不是为某一个固定用户定制的，或者组件实现的功能并不是完全与用户要求吻合的，因此，基于组件的软件开发过程是在用户需求、软件设计、项目管理和组件市场之间的一个平衡过程。传统软件开发过程是对图 1 表示的子开发过程的迭代[3]，而基于组件的软件开发过程是对图 2 表示的子开发过程的迭代[2]。

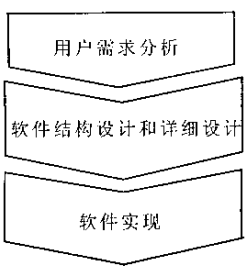

图 1 传统软件开发迭代子过程

1.3 软件开发过程中的软件测试

在软件开发过程中，软件缺陷具有放大效应，即如果一个软件缺陷在开发过程的早期没有被发现并纠正，那么随着开发过程的进行，它给整个软件系统带来的副作用将呈指数级增长[3]。所以，在软件测试中有一句格言“尽早测试、经常测试、充分测试”[5]。因此，软件测试不是独立于软件开发过程的，而是与软件开发过程紧密相关并与之组成一个有机的整体。传统的面向过程的软件测试一般分为 5 个阶段，即单元测试，集成测试、系统测试，回归测试和接受测试[6]。一般而言，经过这一系列的测试后系统就可以分发并使用了。虽然，许多面向对象软件测试技术与传统测试技术相同，或者可以从传统测试技术演化而来，但是面向对象软件的测试与传统的面向过程软件的测试之间依然存在不同之处的，特别是面向对象的编程语言中的继承，封装或多态等特征给传统测试技术带来了新的挑战[4]。基于这些特征，面向对象的软件一般进行 6 个阶段的测试：模型测试，类测试(代替传统的单元测试)，交互测试(代替传统的集成测试)，系统测试，接受测试和发布测试[5]。随着 Internet 的广泛应用和商业组件市场的快速发展，从面向对象的编程技术发展而来的基于组件的软件开发技术越来越多地应用于软件系统的开发。基于组件的软件开发技术最大特点是用于构成软件系统的基础是由第三方提供的无源码的，基于不同平台的，不同技术开发的组件[7]，因此，面向组件的软件测试技术与面向对象的软件测试技术的要求有所不同。面向组件的软件测试技术主要从以下两个方面分类：对组件开发者而言是通过对组件的测试提供组件的质量证明和增加组件使用者使用该组件的信心[8]；对组件使用者而言是保证各组件间能正常有效地配合工作的集成测试和是否能正确完成系统要求的系统测试[4]。同样的，在基于组件的软件开发过程中，测试也存在于每个迭代子过程中，目的是保证该迭代子过程中系统实现的功能与用户的业务需求模型一致。所以在确定具体组件之前，通过一系列测试活动，帮助开发人员从市场上(或组件库中)大量符合条件的组件选取满足用户的业务需求的组件，并通过集成测试验证组件之间能正常集成；一旦确定具体组件，进行的则是测试组件集成之后系统是否符合用户的系统业务需求[2]。

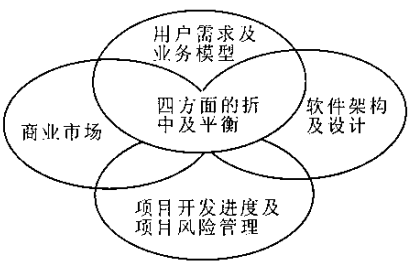

图 2 基于组件的软件开发迭代子过程

2 面向组件的软件测试策略和测试技术

通常软件测试可以分成两大类[3,9]：基于需求的测试和基于程序代码的测试。按照这种分类方法，功能测试技术及基于状态的测试技术属于前一类，而结构测试技术和故障测试技术属于后一类。这些测

试技术都能用于面向组件的测试。表 1 列举了它们的特点,使用阶段和不足之处。

对于传统的软件测试阶段而言,单元测试的目的是保证每个程序单元都符合它的说明,尽可能地揭示程序单元内部的缺陷[3,6]。通常单元测试是在单元开发时进行的,所以一般都与程序代码有关。对于基于组件的软件开发过程而言,单元测试主要指组件开发者通过测试对组件本身的测试,目的有两个:一是提高用户对组件质量的信任程度;二是帮助用户减少测试费用。由于组件开发者能够使用组件的源代码,所以可以使用结构测试技术和故障测试技术[9],另外功能测试技术和基于状态的测试技术也常常被组件开发者用于单个组件的测试。

软件系统的各部分工作正常并不等于整个系统组合在一起后协同工作正常,所以集成测试的目的就是尽可能的发现存在于这些协作关系中的软件故障[3,6]。这些软件故障一般都发生软件各部分的行为交互上,特别是对基于组件的软件系统,这个问题更加突出。所以从组件使用者的角度来看,测试目的就是验证挑选出来的组件之间能够和谐的工作[6]。然而,传统的集成测试技术却不太适合面向组件使用者的组件集成测试。原因有:(1)由于组件的版权限制,组件使用者几乎不可能获得组件的源代码,因此,基于软件代码的技术,如结构测试技术和故障测试技术在这里无法直接使用;(2)对基于组件的系统,即使组件是带源代码的(如某些开放源代码的组件),由于这些组件有可能是在不同开发平台或使用不同程序语言开发的,结构测试技术和故障测试技术也依然无法直接使用;(3)由于组件常常不是根据某个具体最终用户开发的,它提供的功能常常会比组件使用者要求的多。在这种情况下用传统的测试覆盖率来衡量集成测试结果是没有意义的,因为有可能组件提供的某些功能在该系统中根本不会被使用。例如,在结构测试中的基于程序控制流图的技术评估标准是对程序中所有控制流的测试覆盖率能达到某个特定的百分比,但是在一个基于组件的系统中评估这样的测试覆盖率应该要排除组件中在目标系统中不使用的功能部分,否则即使一个设计得很完善的测试集也可能只获得较低的测试覆盖率[10]。因此,面向组件的集成测试一般采用功能测试技术,基于状态的测试技术或经过改进的结构测试技术。

系统测试是证明被测系统完成了用户需求定义的功能为主[3,6]的对整个系统的测试活动,一般也是采用功能测试技术和基于状态的测试技术。

3 UML 与面向组件的软件测试

UML 是目前软件开发业应用最广泛的可视化标准建模语言[11]。它不仅在工业界得到广泛支持,而且已经被 OMG 组织(Object Management Group)采纳作为业界标准。这实际上是软件界的第一个统一的建模语言。人们对 UML 的研究和应用重点不仅放在软件系统设计和开发方面,在软件开发过程中的软件测试部分也有不少研究和应用。

3.1 UML

UML 是第三代的用于为面向对象开发的软件产品进行说明、可视化和编制文档的方法。从 1997 年 11 月开始,UML1.1,UML1.3 和 UML1.4 陆续被 OMG 采纳成为正式技术标准。UML2.0 也已于 2003 年 6 月在法国巴黎召开的 OMG 技术会议上被宣布为正式技术标准[11]。随着 UML 新版本的不断推出,UML 对软件开发过程的支持会越来越好。

表 1 测试技术及特点

测试技术	技术特点	缺陷	应用实例
功能测试(也称黑盒测试)	测试者完全根据软件需求产生测试用例及测试数据,与软件具体实现无关。因此,高质量的,与用户需求相吻合的需求说明是该技术应用的基础。	(1)不能保证系统的每个关键部分都被测试到;(2)如果需求说明没有规格化或不完全,据此产生的测试用例集的质量会受到影响。	(1)等价划分选取测试集;(2)根据边界值选取测试集;(3)根据因果图选取测试集;(4)根据输入值分类划分选取测试集;(5)根据测试习惯和分布随机选取测试集;(6)根据测试规模或测试预算选取测试集。
基于状态的测试	该技术的基础是把待测程序单元或系统的所有状态用状态图表示出来,根据状态图产生测试执行序列以验证被测系统是否能够正确执行。	存在状态爆炸问题。	有限状态机。
结构测试(也称白盒测试)	测试者需要根据程序内部实现细节产生测试用例。	产生的测试用例有可能是系统不可能执行到。	(1)基于程序控制流图的技术;(2)基于程序数据流图的技术。
故障测试	根据软件开发过程中经常发生的错误选择产生测试用例,目的是发现系统是否避免了这些典型错误,以证明系统的正确性。	与使用的编程语言平台或编程环境有关。	(1)在程序中种植错误点的技术;(2)错误突变技术。

UML 并不是具体的软件开发方法，软件开发方法需要定义软件开发的步骤和内容，而 UML 只定义了图及图的意义。所以任何平台和方法都可以使用 UML。UML 作为一种语言由图和元模型组成，使用的是一个 4 层建模概念框架，即图是 UML 的语法，UML 语义定义在该 4 层框架中[11]，分别是元元模型层、元模型层、概念模型层和用户模型层。由于 UML 用于描述模型，根据该框架，它用模型来描述系统的基础结构，静态特征，以及行为或动态特征。从不同的视角为系统的架构建模以形成系统的不同视图，包括用例视图、逻辑视图、并发视图、组件视图和展开视图等。另外 UML 具有可扩展机制，可以根据需要定义其它视图。每种 UML 视图都是由一个或多个图组成，一个图实际上是系统架构从某个侧面的表示，所有图一起组成系统完整视图。目前 UML 提供 9 种不同的图，分为两大类，一是静态图，描述的是系统的静态结构，包括用例图、类图、对象图、组件图和配制图；另一类是动态图，是对系统动态行为的建模，包括序列图、协作图、状态图和活动图。使用不同的 UML 图可以从各角度描述软件系统的内在信息。由于软件测试的主要目的是尽可能的寻找软件系统执行过程中的缺陷或错误，所以大多数的软件测试技术是基于 UML 动态图为主，辅以静态图提供必要的基本信息的[12]。

UML 的应用贯穿于整个系统开发的各阶段，在需求分析阶段，使用用例视图表示客户需求；分析阶段，用逻辑视图和动态视图来描述；设计阶段，把分析阶段的结果扩展成技术解决方案，具体产生构造阶段的详细规格说明；构造阶段，根据详细规格说明生成程序代码；测试阶段，根据不同测试阶段使用不同的 UML 图，以判断被测试的部分或软件系统是否与设计相符或有哪些偏差。对于面向组件的软件测试也可以按照传统方法分为 3 个测试级别：单元测试、集成测试和系统测试。UML 图可以分别用于不同的测试阶段。

3.2 基于 UML 的单元测试

单元测试的目的是保证每个程序单元都符合它的说明，尽可能地揭示程序单元的内部缺陷[3,6]。单元测试是在单元开发时进行的，一般与程序代码有关。UML 中的类图显示系统中类之间的交互关系，体现在类图中包含该类的有关接口属性，方法和与别的类之间的重要关系。而状态图则是一个状态机[11]，具体提供了该类（或一组类）的动态行为信息。基于状态的测试技术在软件测试中研究较多而且应用广泛，它的基本原理是：假定系统设计和被测系统都是满足一些合理假设的有限状态机，则可以根据一定的输入判断系统设计和被测系统的行为是否相同[3]。目前许多基于 UML 的单元测试技术是利用类图和状态图直接产生状态机，再利用状态机生成测试用例[13~17]。

其中有代表性的是 Vieira 等[16]提出的基于需求的单元测试技术，采用的方法是从 UML 状态图设计产生测试用例。该测试方法过程如下：首先确定需要测试的 Java 类和描述该类行为的状态图，再直接把状态图映射成有限状态机；在这个有限状态机上，由测试者决定每个状态的访问次数，某个特别的状态何时需要重新访问以及需要重新访问的频率，自动产生测试脚本；最后运行该测试脚本，评估结果以取得测试的错误发生率。但由于这种根据设计模型获取的系统或对象状态机多数是平坦的，存在状态爆炸问题[14]，所以多数基于 UML 的单元测试技术的研究重点放在如何减少测试用例的产生数量和优化测试过程。

3.3 基于 UML 的集成测试

集成测试的目的是尽可能地发现存在于协作关系中的软件故障[3,6]。这些软件故障一般都发生在软件各部分的行为交互上，特别是对基于组件的软件系统问题更加突出。一般面向组件的基于 UML 的集成测试技术主要是根据 UML 的动态图，再辅以从用例图或系统级的类图中提取的必要的结构信息。采用的测试技术大致分为两类：基于状态的测试技术和基于事件信息流的测试技术。这里事件信息流测试技术类似于结构测试技术中的程序控制流或数据流技术，不同的是这里的事件信息流信息是从 UML 图（如序列图、合作图等）中获得，而不是从程序源代码中获得。

由于状态图本身是一个状态机，所以基于状态机的集成测试主要是从状态图中提取测试相关信息的[15,18~20]。其中 Jean 等[15]提出的集成测试方法比较典型，具体方法是先从各 UML 状态图产生一个系统级的全局组合状态图，之后根据从底向上的集成策略对各部分集成，然后分别测试，以发现软件各部分（包括接口部分）合作是否会产生故障。这种基于状态机的集成测试技术同样存在状态爆炸问题，其研究重点也是如何减少测试用例的产生数量和优化测试过程。

在大多数并行程序中，并不是每个事件都会引起程序的状态变换，更多的是按时间排序的并行事

件序列。UML 中的序列图和合作图反映的是外界与对象之间或对象于对象之间的为完成所需功能而传递消息的关系,二者不同之处在于序列图是以时间排序的,而合作图着重于对象之间的关系。所以,很多基于事件流的测试技术是以序列图和合作图为基础的[21~29]。其中 Hoijin[21]用 UML 的序列图和合作图构造一个 UML 测试模型,用于面向组件的软件系统的集成测试。其测试单元称为 ASF (Automatic System Function),定义为一个从输入到输出的完整功能块,用运行于一个组件中的最大消息集合表示。这些 ASF 可以从表示正常事件流和异常事件流的序列图和合作图中提取。提取出来的 ASF 集合构成一个 UML 测试模型。根据由底向上的集成策略和一定的覆盖选择准则(如全选准则)产生测试用例集。但该技术没有考虑序列图或合作图间的关系。Fraikin 等[22]作了改进,他们认为各序列图中如果存在相同的对象,则该对象就是不同序列图的组合点。但这仅表示了各序列图之间的对象关系,其他的,如控制流关系等并不能被表示,即使用序列图和合作图的集成测试技术从根本上不能完全表示待测系统之间的关系,使该测试技术存在先天缺陷。作为改进,Ye[23]采用其他 UML 图(活动图)来反映这种关系,并称之为连接图。如何解决该问题成为基于序列图和合作图集成测试的研究重点。

回归测试是指应对软件修改后的部分进行测试,以保证修改后没有引入更多错误,这也是把回归测试归入集成测试的重要原因[3]。UML 用例图、合作图和序列图都能用于回归测试[25,27]。其中 Ye 等[25]提出从合作图提取包含数据流和控制流的测试框架,以评估新旧组件之间的相似性。

3.4 基于 UML 的系统测试

系统测试是指测试整个系统,以证明被测系统完成了需求定义的功能[3,6],因此,也称功能测试。凡是能表示软件系统功能的 UML 图都能用于系统测试,如用例图、类图(这里指系统级类图)、状态图、序列图、合作图和活动图。由于需要全面测试系统的功能,应根据 2 种以上 UML 图进行测试集的设计,这与用 UML 对系统建模的方式是一致的,因为 UML 的每种图描述的只是系统的某个方面,几种图组合起来才能完整的描述整个系统。系统测试采用的测试技术大致分为:基于消息序列路径的测试技术[30~33]和基于场景的测试技术[34~38]。它们属于功能测试技术和基于状态的测试技术。

在第一类测试技术中,Briand 等[33]采用的基于用例图、活动图和序列图的系统测试技术比较有代表性。具体方法如下:首先把系统级的活动图直接映射成图,对该图按深度优先搜索自动产生用例序列,其中守卫条件从活动图中取得。由于每个用例与一个序列图相关,可以从序列图中产生规则表达式(其中的各项表示的是该用例的一个或一组场景)。对整个用例序列实例化后,用这样的规则表达式表示序列的执行路径。这时一个规则表达式就是一个消息序列路径。再根据产生测试数据的一般方法,如等价划分法和边界取值法等自动产生测试用例集。

一个序列图表示的就是一个场景[11],所以许多基于场景的 UML 系统测试技术很自然的采用了序列图。如 Jeremiah 等[37,38]采用的就是基于场景的测试技术,这里的测试场景是直接从序列图中提取的。该方法首先从序列图中产生测试需求,即取得一组序列图中的某个对象发出的所有消息。然后由测试者对每个方法的调用参数赋具体的值和填写该测试场景的预期值,运行后评估测试结果。由于序列图无法描述各序列图之间的相互关系,一些研究者[34,35]转而采用活动图来进行系统测试的设计。活动图本质上是一个状态机[11],能够描述各活动图之间的复杂关系,表示的是系统功能。在活动图中定义了表示功能的工作流和这些工作流由哪些活动组成,在哪里开始,到哪里结束等等,实际上这就是系统的典型场景的组合。例如刘敏等[34]提出的系统测试技术就是运用 UML 活动图生成测试场景,再依据测试场景进行测试用例的自动生成。

4 结束语

使用组件,特别是第三方商业组件集成开发而成的软件越来越多,面向组件的测试技术的需求越来越大。UML 是目前软件开发业应用最广泛的可视化标准建模语言,它能应用于整个软件开发过程的各阶段。UML 的许多特点与面向组件的软件测试需求是相吻合的。本文分析和研究目前面向组件的软件开发中测试阶段利用 UML 的研究现状及研究重点,认为该技术具有广阔的研究和应用前景。

参考文献:

1 Wallnau, Kurt C Scott Hissam, Robert Seacord. Building systems from commercial components. Addison-Wesley, 2002.

2 Cecilia Albert, Lisa Brownsword. Evolution process for integration COTS-based systems (EPIC): an overview. Technical Report CMU/SEI-2002-TR-005, 2002.

3 Binder R V. Testing object-oriented systems - models, patterns and tools. Addison-Wesley, 1999.

4 Chan W K, Chen T Y, Tse T H. An overview of integration testing techniques for object-oriented programs, in Proceedings of the 2nd ACIS Annual International Conference on Computer and Information Science (ICIS 2002). Mt Pleasant Michigan, 2002. 696～701.

5 Sykes David A, McGregor. John D. A practical guide to testing object oriented software. Addison-Wesley, 2001.

6 Beizer B. Software testing techniques. New York: Van Nostrand Reinhold, 1990.

7 Weyuker E J. Testing component-based software: a cautionary tale. Software, IEEE, 1998, 15(5): 54～59.

8 Li H, Lam C P, Bai X. Promoting confidence in software components: A UML approach. in Proceeding of NASAC, 2003. 228～233.

9 Auri Marcelo Rizzo Vincenzi, José Carlos Maldonado, Márcio Eduardo Delamaro. Component-based software: an overview of testing. In: Alejandra Cechich. Mario Piattini, Antonio Vallecillo (Eds). Component-Based Software Quality-Methods and Techniques. Lecture Notes in Computer Science 2693 Springer, 2003. 99～127.

10 Rosenblum David S. Adequate testing of component-based software. Technical Report TR UCI-ICS-97-34, University of California, 1997.

11 OMG, UML 1.4 specifications, http://www.omg.org.

12 Williams Clay E. Software testing and the UML, presented in the International Symposium on Software Reliability Engineering (ISSRE'99). Boca Raton, 1999. 1～4.

13 **李留英, 王 戟, 齐治昌**. UML Statechart **图的操作语义. 软件学报**, 2001, 12: 1864～1873.

14 **李留英, 王 戟, 齐治昌**. UML Statechart **的测试用例生成方法. 计算机研究与发展**, 2001, 38(6): 691～697.

15 Jean Hartmann, Claudio Imoberdorf, Michael Meisinger. UML-based integration testing, in Proceedings of ISSTA, Portland, Oregon, 2000, 8: 60～70.

16 Vieira M, Dias M, Richardson D J. Object-oriented specification-based testing using UML Statechart diagrams, in Proceedings of ICSE2000-Workshop on Automated Program Analysis, Testing, and Verification, Limerick, Ireland, 2000, 6: 796.

17 Kim Y G, Hong H S, Bae D H, et al. Test cases generation from UML State diagrams. IEE Proceedings-Software, 1999, 146(4): 187～192.

18 Briand L C, Labiche Y, Wang Y. Toward a comprehensive and systematic methodology for class integration testing. Technical Report TR SCE-03-02, Carleton University, 2003.

19 Hanh Vu Le, Kamel Akif, Yves Le Traon, et al. Selecting an efficient OO integration testing strategy: An experimental comparison of actual strategies. in Proceedings of ECOOP2001 - Object-Oriented Programming: 15th European Conference, Budapest, Hungary, Lecture Notes in Computer Science 2072, Springer, 2001. 381～401.

20 Ye Wu, Chen Mei-Hwa, Jeff Offutt. UML-based integration testing for component-based software, in Proceedings of ICCBSS, Canada, 2003. 251～260.

21 Hoijin Yoon, Byoungju Choi, Jin-Ok Jeon. A UML-based test model for component integration, presented at APSEC'99-Workshop on Software Architecture and Components, Takamatsu, Japan, 1999.

22 Falk Fraikin, Thomas Leonhardt. SeDiTeC-testing based on sequence diagrams. in Proceedings of ASE'02, September, Edinburgh, UK, 2002. 261～266.

23 Simon Pickin, Claude Jard, Thierry Heuillard, A UML-integrated test description language for component testing. in Proceedings of pUML2001-Practical UML-Based Rigorous Development Methods -Countering or Integrating the extremists, Toronto, Canada, 2001. 208～223.

24 Aynur Abdurazik, Jeff Offutt. Using UML collaboration diagrams for static checking and test generation. in Proceedings of UML '00, York, UK, 2000. 383～395.

25 Ye Wu, Jeff Offutt. Maintaining evolving component-based software with UML. in Proceedings of CSMR'03, Italy, 2003. 133～142.

26 Eun Man Choi, Anneliese von Mayrhauser. Testing object-oriented systems using extended use-cases. in Proceedings of PDPTA 2000, Las Vegas, June 24-29, 2000. Retrieved from http://www.dvo.ru/

27 Tsai W T, Paul R, Zhibin Cao, et al. Adaptive scenario-based testing using UML, presented at OMG's Third Workshop on UML (tm) for Enterprise Applications: Model Driven Solutions for the Enterprise, October 2002, Francisco, CA. http://www.omg.org/news/meetings/workshops/uml-2002.htm.

28 Bertolino A, Basanieri F. A practical approach to UML-Based derivation of integration tests. in Proceedings of QWE2000, Brussels, Belgium, Paper 3T (CD-ROM), 2000.

29 Basanieri F, Bertolino A, Marchetti E, et al. An automated test strategy based on UML diagrams, presented at Ericsson Rational User Conference. Upplands Vasby, Sweden, 2001.

30 Benoit Baudry, Yves Le Traon, Gerson Sunyé. Testability analysis of a UML class diagram. in Proceedings of IEEE Metrics02, Ottawa, Canada, 2002. 54～65.

31 Jeff Offutt, Aynur Abdurazik. Generating tests from UML specifications. in Proceedings of UML99, Fort Collins, 1999. 416～429.

32 Kim Y, Carlson C R. Scenario based integration testing for object-oriented software development. in Proceedings of the Eighth Asian Test Symposium, Shanghai, China, 1999. 283～288.

33 Briand L, Labiche Y. A UML-Based approach to system testing. Software and Systems Modeling Springer, 2002, 1(1): 10～42.

34 刘 敏, 金茂忠, 刘 超. 基于 UML 活动图模型生成测试场景的设计. 计算机工程与应用, 2002, 12: 122～124.

35 张 楣, 刘 超, 孙昌爱. 基于 UML 活动图模型的测试用例生成技术研究. 北京航空航天大学学报, 2001, 27(4): 433～437.

36 Ben Lieberman. UML activity diagrams versatile roadmaps for understanding system behaviour. http://www.therationalegdge.com/content/apr_01/t_UML_bl.html.

37 Jeremiah Wittevrongel, Frank Maurer. Using UML to partially automate generation of scenario-based test drivers. in Proceedings of OOIS '01, University of Calgary, Canada, 2001. 303～306.

38 Jeremiah Wittevrongel, Frank Maurer. SCENTOR: scenario-based testing of E-business applications. in Proceedings of 10th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, Massachusetts, 2001. 41～48.

（责任编辑：邓大玉）

（上接第 56 页）

5 异或运算取反法

经过分析异或运算法可知，实现两变量的交换时也可采用如下方法：

```
sub swap(a,b)
  a=not(a xor b)
  b=not(a xor b)
  a=not(a xor b)
end sub
```

该方法与异或方法实质上是一样的，虽然比上面的方法要多计算一个过程，但不失为一种好方法，该方法也适合于整数、字符型、浮点型和布尔型，但不能用于其复杂的数值类型，如字符串类型。

6 结束语

以上这种方法都是免用第三变量来实现两变量的交换，减少了内存资源和提高了执行效率。但这几个方法仍然存在一些不足之处，对一些复杂的数据类型仍然行不通。这就要求我们在设计程序的，针对不同的变量类型，采用不同的方法，实现内存资源的优化和提高整个程序的执行速度，使程序达到最优化的目的。

参考文献：

1 白中英. 数字逻辑与数字系统. 第 2 版. 北京：人民邮电出版社，1999. 9.

（责任编辑：黎贞崇）