第7卷第3期
2006年6月

解放军理工大学学报(自然科学版)
Journal of PLA University of Science and Technology

Vol 7 No. 3
Jun. 2006

文章编号: 1009-3443(2006)03-0236-06

基于构件的软件测试模型及方法

奚和平

(解放军理工大学 指挥自动化学院, 江苏 南京 210007)

摘 要: 采用基于构件的软件设计方法是软件设计的新趋势。介绍了构件及基于构件的软件(CBS)的概念、特点以及由此引发的测试问题。概述了构件软件测试方法的现状, 阐述了基于构件的软件黑盒测试方法及模型、白盒测试方法及模型, 讨论并比较了两种方法的使用场合和异同之处。在此基础上, 列举并比较了几种常见的自动测试方法和工具的基本原理。对构件软件的集成测试和性能测试等问题进行了讨论和分析, 给出了构件软件测试的存在问题和研究方向。

关键词: 构件; 基于构件的软件; 测试充分性标准

中图分类号: TP311 **文献标识码**: A

Models and methods of component-based software

XI He-ping

(Institute of Command Automation, PLA Univ. of Sci & Tech., Nanjing 210007, China)

Abstract: Using component-based software designing method is a new trend of software designing. The concept characteristics and testing problems of component and component-based software (CBS) were introduced. The current status of the testing methods of CBS is summarized. Black-box testing methods, white-box testing methods for CBS were described comprehensively, and their methods used and difference were discussed. Furthmore, some common test automation and tools were listed and compared. The problems of integration testing and performance testing of CBS were analysed. Finally existing problem of CBS and the future research of CBS were pointed out.

Key words: component; CBS (component-based software); test-adequacy criteria

软件测试正越来越得到人们的重视。专家学者们在软件领域做了大量的研究工作[1~3], 提出了许多测试方法和测试策略。但软件测试是计算机工程中比较难的一步, 测试的充分性问题、剩余缺陷数估计问题、测试用例自动生成等问题, 目前还没有完善的解决方法。

继面向对象的设计方法之后, 基于构件的软件设计方法正在逐渐成为新的趋势[4]。由于构件的特点, 使得基于构件的软件开发更具优势, 但也带来了分析、设计、实现、测试和维护的一系列问题。

收稿日期: 2006-02-23.

作者简介: 奚和平(1967-), 女, 副教授; 研究方向: 指挥自动化系统理论与技术; E-mail: xuning168@vip.sina.com.

1 构件的定义和特点

结合W. B. Alan[5]、S. Clemens[6]、T. Lewis[7]和B. Ugo[8]等学者的定义, 可以综合出构件的基本定义: 构件是近乎独立的、可替换的、满足一定功能的模块; 构件之间通过接口进行通信; 构件通常是经过编译的二进制代码或机器码, 甚至是可执行的。它具备以下特点: 构件是封装了数据和实现方法的软件模块, 只有接口是可见的, 只能通过接口与外界交互; 构件可以增加新的接口, 但原有接口的定义不能改变; 构件是经过编译的二进制代码, 对使用者源代

码是不可见的; 系统中的构件是可替换的, 不需要重新编译。

按照构件化程序设计的思想, 复杂的应用程序被设计成一些小的、功能单一的构件模块, 这些模块可以运行在同一机器上, 也可以运行在不同的机器上, 每台机器的运行环境可以不同, 甚至可以是不同的操作系统。为了实现这样的应用软件, 构件需要一些细致的规范, 只有构件程序遵守这些共同的规范, 构件软件才能正常运行。目前流行的构件规范主要有 CORBA (common object request breaker architecture)[9], COM (component object model)[10] 和 EJB (enterprise java bean)[11]。CAI[12]给出了 3 种规范的比较。所谓构件软件则是基于构件技术开发的软件。目前的应用已有很多, 如微软的 OLE 和 Active 控件技术。

2　构件及构件软件的测试问题

M. J. Harrod[13]认为应该从构件提供者和构件使用者两个不同的角度来看待构件软件的测试问题。构件的提供者认为构件相对于使用构件的环境是独立的, 所以要用上下文独立的方式测试构件所有的功能。相反, 构件使用者开发的应用程序提供了构件的运行环境, 所以构件使用者不把构件看成独立的单元, 仅仅考虑与应用程序相关的构件功能。另一个重要的区别是提供者有构件的源代码, 而使用者则通常没有构件的源代码。

B. Adrita[14]指出当构件失效产生的后果大于测试的费用时就要进行测试, 构件测试的最终目标是检查构件是否达到规范和满足功能的需求, 设计规范中的结构关系是否正确体现出来。

构件测试包括以下问题[14, 15]:

(1) 测试充分性判据 (test adequacy criteria) 的可扩展性。由于复杂性问题和组合爆炸问题, 对小规模程序适用的判据对大规模程序不一定适用。

(2) 测试数据的产生。由于同样的原因, 难以产生合适的测试输入, 使得对低层次元素 (如分支、定义使用对, 需求功能可看作是高层元素) 难以达到较高的覆盖率。

(3) 如何配置构件的测试环境。对单个构件进行测试的环境, 与构件在实际系统中运行的环境可能不同。所以在测试构件时应该考虑模拟真实环境, 如构件的竞争和死锁和多线程等。但是由于构件的复用, 难以完全模拟构件使用的所有真实环境。

(4) 构造测试驱动器和打桩技术。传统的技术是面向特定的工程, 但是构件的多样性和其功能的专用化使得传统的技术达不到应有的效力。

(5) 构件测试的可重用性。对构件的测试应该是可重用的。

测试构件软件系统存在如下问题[14, 15], 或者说是构件使用者面临的测试问题:

(1) 测试充分性判据的可扩展性问题依然存在。

(2) 构件的测试顺序。如果软件采用分层结构, 可以先测试底层构件, 因为不需要其他构件提供服务; 然后再测试高层构件, 所调用到的其他构件都是经过测试的。如果不是分层结构, 就可能难以确定测试的顺序。

(3) 冗余测试问题。通常先对构件进行单独测试, 然后使用相同的充分性判据进行集成测试, 这就导致有些测试是重复的。

(4) 源代码是否可用。源代码可用与否导致不同的系统测试方法。

(5) 编程语言、操作系统平台、硬件结构的混杂性 (heterogeneity)。系统使用的构件可能是用不同语言编写的, 运行在不同的平台上, 这就要求测试方法和工具与平台和语言无关。

(6) 测试分布式软件时的监控问题。分布式系统比集中式系统复杂得多, 其监控和数据收集存储机制也更为复杂。

(7) 事件重构。分布式系统中经常有并发处理和异步通讯, 对整个运行环境难以完全控制。这就导致程序的执行情况难以重现, 因此只测试构件不足以说明系统的可靠性。

3　构件软件测试方法研究现状

国外在构件软件的测试技术领域已有一些研究, 各种方法的比较, 如表 1 所示。详细内容可参考文献[16]。关于测试的充分性, E. J. Weyuker 定义了测试充分性条件的 10 个公理[17], 其中: 8 条适用于所有软件, 包括构件软件和非构件软件; 另外 2 条专门针对基于构件的软件, 称为反解构公理 (axiom of antidecomposition) 和反构造公理 (axiom of anticompostion)。Weyuker 的公理是正确的, 但对基于构件的软件来说并不是太适用。原因是: 公理是基于面向过程的软件结构, 而不是面向对象的; 公理描述的是最坏的情况, 而不是一般性的情况; 公理中的测试充分性要求达到 100%, 但实际测试中很少能达到 100%。

目前国内在基于构件的软件开发方面已开展了

表1 构件测试方法的比较

Tab 1 Comparisons of testing methods on component-based software

编号	测试技术	适于构件提供者还是使用者	系统测试还是单元测试	有无自动化工具支持
1	Certification of component	提供者	都是	有
2	The component meta-data way	提供者	系统测试	开发中
3	component test bench(CTB)	提供者	单元测试	有
4	UML Integration test	使用者	系统测试	无
5	Component interaction test(CIT)	提供者	都是	有
6	Built in tests	使用者	都是	有
7	Component software flow graph(CIG)	使用者	系统测试	无
8	Retrospectors	都是	都是	有
9	Component interaction graph	使用者	集成测试	开发中
10	TDS	都是	都是	有
11	An approach user program slicing controldependence analysis and data-flow testing	都是	都是	无
12	Sequence generation technique	都是	单元测试 集成测试	开发中

一些工作,如北京航空航天大学的刘超等提出了一种白盒测试工具的设计方法,就是基于构件的思想。在软件测试工具的开发方面,暨南大学的CHEN Huoyan[18]提供了基于代码插装的类簇级测试工具,北京航空航天大学的CHEN Xuesong 等提供了构造运行剖面和产生测试数据的计算机辅助工具。但是对基于构件的软件测试技术研究与测试工具开发方面的研究还很少见到。

4 构件软件测试模型及方法

4.1 构件软件黑盒测试模型及方法

根据IEEE定义,黑盒测试又称为功能测试,含义为测试过程中忽略系统或组件的内部机制,只关注对应于用户选择的输入和执行条件而产生的输出。测试执行是为了评价一个系统或组件与指定的功能需求的符合性。构件软件黑盒测试的问题主要表现在构件需求规格说明、构件确认历史记录、构件定制等3个方面。

需求规格说明是测试的基础,构件是有规范的接口标准和清楚定义外在联系的一组单元。因此,在构件规范说明中应该对什么是构件接口以及如何使用接口有详细的描述。事实上,不同开发商在定义构件规范上使用着不同的定义语言,因此构件规格说明的最大问题就是规范描述语言的选择[19]。目前不能广泛采用统一规范语言的原因是人员培训问题、软件购买能力以及规范语言只有有限的工具支持等因素。

虽然规范的需求描述语言为开发方测试提供了基础,但是它并不提供双方在测试确认方面的信息。XML(extensible markup language)作为一种表示层次性数据的通用脚本语言,可以很好的满足上述要求。XML的优势具体表现为:表示数据功能强大、处理简单、在几个领域有由XML发展而来的特定语言,如:XMI、BML、UXF等。

当一个构件运用到新的环境中时,通常需要对原构件的需求说明作修改。构件定制主要功能在于消除两份规格说明的不同。它存在的主要问题是重新开始需求说明,花费大;通过定制来重用构件提供者的测试成果存在测试充分性问题。构件定制主要分为以下3种:

(1)通用的定制

构件需求规格说明中有限制,而用户需求规格说明中无限制。

(2)专门的定制

构件需求规格说明无限制,而用户需求规格说明中无限制。

(3)重构的定制

构件需求规格说明与用户需求规格说明对于某些输入域有改变。

常用的黑盒测试方法有:随机测试、等价类划分测试、边界值测试、基于判定表的测试以及变异测试等。结合传统的测试方法以及构件需求规格说明、构件确认历史记录、构件定制等基本问题可以实现构件软件的黑盒测试技术。

4.2 构件软件白盒测试模型及方法

白盒测试方法可以分为逻辑驱动测试和基本路径测试两种类型。逻辑驱动测试主要是测试程序的覆盖率,以程序内在逻辑结构为基础。包括以下7种类型:语句覆盖、判断覆盖、条件覆盖、判定—条件覆

盖、条件组合覆盖、路径测试以及基本路径测试。

上述白盒测试方法大都可以用于基于构件的软件测试中, 但是有两点在基于构件的软件测试中需要注意:

(1) 传统的软件开发应用背景都是固定的, 有详细的需求说明, 而基于构件的软件开发其背景是独立的。当构件组合时就涉及到背景的转变, 因此, 测试过程中需要考虑更多的不确定因素。

(2) 基于构件的软件测试比传统的白盒测试有更高的要求。因为很多构件问题只有当构件发布后在实际使用中才能发觉, 因此构件的测试要求更高。

基于白盒的构件测试主要有两个因素: 测试构件时考虑该构件可能涉及到的尽可能多的内外环境因素。构件发布时, 要符合严格的测试标准。

4.3　构件软件自动测试方法及工具

随着构件软件的复杂性增加, 基于构件的自动测试生成变得越来越复杂。原因是: 构件支持 GUI 使其增加了难度; 构件支持多媒体使其增加了复杂度; 分布式的构件更增加了其通信等方面的难度。因此, 作为一个好的基于构件的自动测试工具, 下面的功能是测试人员真正需要的: 定义好的黑盒测试模型和充分的测试标准、系统的方法和工具支持基于领域的测试生成。目前一些基于构件的测试工具已经开发, 有代表性的主要有两种: 基于规约的黑盒测试工具——ADL scope 和用于 JAVA 的工具集——Java Compatibility Test Tools。

S. Zweben 等定义了一个为单个构件提供黑盒测试模型的流图模型[20]。流图是一个有向图, 每个顶点代表构件的一个操作, 每条有向边代表从顶点 A 至顶点 B 执行操作的可能性。构造流图的步骤如下:

(1) 从规约中定义构件的初始操作, 用一个初始点表达该操作;

(2) 定义构件的结束操作, 用一个结束点表达该操作;

(3) 定义所有可能的操作, 用顶点表示, 连接各个可能发生操作的节点。

4.4　构件软件的集成测试

集成测试对于保证构件之间的正确通信非常重要[21~23]。集成测试中, 交互相关的缺陷可以分为: 程序相关的缺陷是指构件之间的缺陷; 程序无关的缺陷是指交互操作相关的缺陷。构件之间的缺陷是指即便单个构件被完全测试, 构件之间还是有缺陷的。很多缺陷都可以归结为构件之间的缺陷, 比如死锁、活锁等。在单元测试很难描述的缺陷, 却是集成测试的主要目标[24, 25]。

集成测试需要把很多构件放在一起, 需要考虑的是: 一是各个构件的顺序如何安排; 二是如何测试新增加的构件。目前主要有基于功能分解的集成测试和基于调用图的集成测试。

当执行构件集成测试时, 前面提到的方法也可以使用, 比如功能分解法。但一些更有效的方法, 比如调用图方法, 由于构件实现代码的不可见性就不能使用了。为解决该问题,M. J. Harrold 等人[26]提出了一种通过分析构件提供者和使用者两方面资源的技术。S. Ghosh 和 A. P. Mathur[27]讨论了分布式构件系统的集成测试问题。

由于没有源代码, 虽然可以很容易得到接口和事件信息, 却不易掌握上下文相关关系和内容相关关系。而作为一种描述、构造、可视化软件系统的语言,UML 具有很多优点, 能以最小的代价, 提供足够的信息去决定如何执行集成测试以及测试的停止标准。在测试时, 测试人员通常会把精力放在接口与事件上, 但是这些接口与事件如何交互、集成之后会有那些潜在的风险则常常会被忽略。为了确定上下文相关关系, 张书杰、于学军[25]等提出了基于协作图和顺序图的方法对测试因素进行描述。通过使用包含类的实例的UML 协作图和顺序图, 就可以描述各交互之间的关系了。需要指出的是, 这种方法不仅能作为构件发布的指导性方法, 还能作为构件质量评估的指导性方法。

4.5　构件软件的性能测试

构件软件的性能测试和度量必须要满足: 正确的评估模型、定义好的指标、高效的性能数据采集和监控技术、有效的性能分析设备和工具。在过去的十几年里, 性能测试的方法已创造了很多种, 归结起来主要有:

(1) 基于状态的方法: 主要应用有限状态机的方法, 它主要关注构件间的状态行为。

(2) 面向用户的方法: 主要关注系统用户的行为性能, 通过用户操作手册衡量。

(3) 基于场景的方法: 主要基于不同层次下系统功能场景, 比如系统反映时间、事务延迟、吞吐量、可靠性及有效性等。

(4) 基于架构的方法: 主要基于系统架构以及系统功能和设计需求。目前主要把系统划分成批-序列/管道(batch-sequential/pipeline)、并行/过滤管道(parallel/pipe-filter)、错误容忍(fault tolerance)、

调用与返回(call and-return),计算系统每一类别状态的指标然后综合起来计算系统的整体性能。这种方法有2个问题:如何获得正确的事务概率;如何综合成整体。

(5)基于事务的方法:运用事务模型表述系统事务、时序以及之间的关系,事务图和Petri网为常用工具。

对于一个给定的构件,构件性能模型是指一个定义好的表示性能属性、参数、动作以及构件和系统架构之间联系的模型。一个好的模型应该提供定义和选择性能评估策略及指标的基础,使它能够帮助工程师理解和定义构件化系统的性能测试和评估问题。下面介绍两种性能评估模型:基于构件的场景模型和基于构件的事务模型。

基于构件的场景模型主要基于事件的场景图,每个场景图有3种元素:构件点集、构件交互事件、构件交互时序。通常构件场景图CSD可以定义为CSD(N,E,O)。其中:N为构件点集;E为交互事件;O为E的时序集合。

基于构件的事务模型主要基于事务图,每个事务图有3种元素:构件点集、每个构件内事务集、连接事务的数据或消息。通常一个事务图CTD可定义为一个四元组(N, T, NT, L),N为构件点集,T为构件级的事务集合,NT是构件事务间的关系(T是构件N的事务),L是事务间数据和消息的集合(一个构件传递至另一构件)。

M. Strembeck和U. ZDUN[28]提出了改进的基于场景的方法,将测试定义为给定场景的正规化和可执行的描述。在连续的修订中从用例场景里派生出各种测试,通过把各个用例与测试信息相连接嵌入到构件当中,从而作为构件测试的元数据来对进行测试,降低了因对重要场景描述不充分导致出现测试遗漏的风险。

基于构件的性能测试依赖于各个独立构件的性能,构件的使用者必须在组装和配置构件时检查和评估那些可重复使用的构件。在基于构件的软件系统中,性能测试是软件质量保证过程中一项非常重要的步骤。为了给构件用户提供性能信息,比如构件的性能边界、容量限制等,构件的开发方必须评估构件的性能以便发现构件的性能问题和容量限制。

5 结 论

5.1 研究存在的问题

对于构件软件的测试,最大的困难在于源代码未知。在这种情况下,一种普遍的思路是要求构件开发者提供必要的构件分析数据,方便构件使用者进行测试,但是到目前为止还没有形成统一的标准。目前各种方法的本质还是借鉴传统的测试技术,如各种覆盖技术、数据流、控制流、面向对象的测试、分布式系统测试,然后根据构件的特点进行改进。不少方法还停留在理论研究阶段,有些方法进行了实证研究,但还缺乏专门的测试工具的支持。

部分学者[8,29]在研究构件软件的集成测试和系统测试问题时都假设构件已经经过了充分的单元测试,但并没有给出构件充分测试的定义和条件,自然就没有手段去保证这一假设成立。目前的构件软件测试技术是由传统的软件测试技术发展来的,重点在于研究构件源代码不可见性带来的测试问题。主要的解决思路是由构件的提供者提供更多附加信息以便进行测试。但是没有充分考虑构件软件的其他特点:语言无关性、平台无关性、版本不确定性,而这些性质带来了一系列测试问题。

5.2 研究发展的趋势

从研究方法上,构件化软件测试应以方法研究、实证研究和工具开发并重,相互印证,相互补充,相互促进。以方法研究为起点,选取适当的软件测试和技术应用于实证研究,从实证研究中改进软件测试技术,提出新的技术研究主题,并为工具开发提供有用信息;而所开发的工具则为方法研究和实证研究提供帮助。在测试技术和方法上应发展和完善构件测试的理论基础,改进传统测试方法;研究构件软件的测试充分性条件;编写可测试的构件,把测试机制引入构件编写的规范中;解决跨平台、跨语言的测试问题。在工具开发上,建立通用的、可重用的测试平台;实现自动确定集成测试顺序的工具;实现自动化的测试用例生成工具。

参考文献:

[1] WALDK, STAMATIS V. A survey of software functional testing techniques[C]. Binghamton: NYS, 1988.

[2] PHYLLIS G F, ELAINE J W. An applicable family of data flow testing criteria[J]. IEEE Trans on Software Engineering, 1988, 14 (40): 1483-1498.

[3] SAYRE K, POORE J H. Partition testing with usage models[J]. Information and Software Technology, 2000(42): 845-850.

[4] POUR G. Component-based software development ap-

proach: New Opportunities and Challenges[C]. California: Eiffel Software, 1998.

[5] ALAN W B, KURT C W. The current state of CBSE [J]. IEEE Software, 1998, 15 (5): 37-46.

[6] CLEMENS S. Component software beyond object oriented programming [M]. Boston: Addison Wesley, 1997.

[7] LEWIS T. The next 10, 000 years, part Ⅱ[J]. IEEE Computer, 1996, 29(5): 78-86.

[8] UGO B, CARLO G, ALESSANDRO O, et al. A framework for testing object-oriented components [C]. Los Angeles: Association for Computing Machinere, 1999.

[9] OMG. The common object request broker: architecture and specification [DB/OL]. ftp://ftp.omg.org/pub/docs/formal/01-12-01.pdf. 1994.

[10] Microsoft. The component object model specification. Version 0.9 [DB/OL]. http://www.microsoft.com, 1996.

[11] SUN. Enterprise javaBeans TM specification, Version 2.0 [DB/OL]. ftp://ftp.java.sun.com/pub/cjb/947q9tbb/ejb-2-0-fr2-spec.pdf, 1998.

[12] CAI Xia, MICHAEL R L, KAM F W et al. Component-based software engineering: technologies, development framewords, and quality assurance schemes [C]. Singapore: APSEC, 2000.

[13] HARROD M J, DONGLIN L, SAURABH S. An approach to analyzing and testing component-based systems[C]. Los Angeles: ACM, 1999.

[14] ADRITA B. Software component testing strategies [DB/OL]. http://www.ics.uci.edu/~abhor/ics221/comp-test.htm, 2001.

[15] SUDIPTO G, ADITYA P M. Issues in testing distributed component-based systems[C]. Los Angeles: ACM, 1999.

[16] 景　涛, 白成刚, 胡庆培, 等. 构件软件的测试问题综述[J]. 计算机工程与应用, 2002, 38(24): 2-3.

[17] WEYUKER E J. Axiomatizing software test data adequacy [J]. IEEE Transaction on Software Engineering, 1986, 12(12): 1128-1138.

[18] CHEN Huoyan. A dynamic approach for object-oriented cluster-level tests by program instrumentation[C]. Nashville: IEEE, 2000.

[19] ADDY E A, SITARAMAN M. Formal specification of COTS-based software: a case study[C]. Los Angeles: ACM, 1999.

[20] ZWEBEN S, HEYM W, KIMMICH J. Systematic testing of data abstractions based on software specifications[J]. Journal of Software Testing, Verification, and Reliability, 1992, 4(1): 39-55.

[21] BLACKBURN M. Mars polar lander fault identification using model-based testing [C]. Greenbelt, Maryland: IEEE CS Press, 2002.

[22] GARLAN C, ALLEN R, OCKERBLOOM J. Architectural mismatch or why it's hard to build systems out of existing parts[C]. Seattle, Washington: ACM, 1995.

[23] WEYUKER E J. Testing component-based software: a cautionary tale[J]. IEEE Software, 1998, 15(5): 54-59.

[24] 许　静, 陈宏刚, 王庆人. 软件测试方法简述与展望[J]. 计算机工程与应用, 2002, 39(13): 1-3.

[25] 张书杰, 于学军, 阎健卓, 等. 基于构件软件系统集成测试的初步研究[J]. 北京工业大学学报, 2004, 30(2): 3-5.

[26] HARROLD M J, LIANG D, SINHA S. An approach to analyzing and testing component-based systems [C]. Los Angeles: IEEE Reliability Society and IEEE Computer Society, 1999.

[27] GHOSH S, MATHUR A P. Issues in testing distributed component-based systems[C]. Los Angeles: IEEE Reliability Society and IEEE Computer Society, 1999.

[28] STREMBECK M, ZDUN U. Scenario-based component testing using embedded metadata [C]. Erfurt, Germany: University of Ulm 2004.

[29] HOIJIN Y, BYOUNGJU C, JIN-OK J. A UML-base test model for component integration test [DB/OL]. http://selab.ewha.ac.kr/HoijinYoon/Papers/wsac.pdf, 2000.

(责任编辑: 程　群)