基于CBD的软件测试策略研究*)

曹严元　张为群
(西南大学计算机与信息科学学院　重庆 400715)

摘　要　基于构件开发(CBD)的软件系统被广泛应用并成为一种主流软件形态。然而,构件软件系统的异质性和实现透明性等特点给测试带来了极大的挑战。寻求高效的构件软件测试技术和开发实用的测试工具是当今软件业界亟待解决的一个课题。本文分析构件软件测试存在的主要问题,提出一个基于CBD的软件测试策略STSofCBS,建立系统化的测试策略,避免测试的偶然性带来的时间和工作量的浪费。

关键词　CBD,软件测试,自测试性,测试策略

Research for the Component-based Development Software Testing Strategy

CAO Yan-Yuan　ZHANG Wei-Qun
(College of Computer and Information Science, Southwest China University, Chongqing 400715)

Abstract　Component-based development software has been widely used in many fields and become a fairly popular software form in recent years. However, the properties of component composition, such as heterogeneity and implementation transparency, bring great challenges to the testing of component-based software systems. In this paper, aim at the main problems of testing CBD software, we research corresponding strategy, named as STSofCBS. It makes the test process systematic and saves testing effort.

Keywords　Component-based development, Software testing, Self-testability, Test strategy

1　引言

在过去的几年,基于构件的开发方法(CBD)由于其对软件功能性、开发效率、质量、可靠性、可移植性等方面的良好支持而受越来越多软件开发组织的重视。目前在构件模型、CBD技术等方面已取得了一定的进展,并与实际的应用紧密结合。基于构件开发软件系统的方法相对于传统的开发方法体现出了许多新优势,实现不同开发语言和平台的协作,使系统的开发周期更短、造价更低。

CBD方法带来了软件设计、实现、部署、验证和演化等方面的挑战。目前对于保证构件软件质量方面的关注还不多,因此我们关心基于CBD的测试方面。普遍的观点认为构件一旦经过充分测试,之后的重用就没有必要再重新测试。然而实践证明构件在最初的测试环境中通过测试并不意味着在新的集成环境中可用。因此在CBD软件的测试活动中,建立系统化的测试策略,通过定义描述整体策略和规程的计划,可以避免测试的偶然性带来的时间和工作量的浪费。本文主要研究基于CBD的软件整个测试过程的策略,把测试结合到开发过程中,使测试过程更加系统化。

本文第2节介绍构件特点及存在的测试问题,第3节针对现在构件软件存在的测试问题及挑战,提出一个系统的测试策略,指导整个测试生命周期。最后给出结论并展望未来。

2　构件软件特点及存在的测试问题

虽然对构件及基于构件开发的软件的研究已有几十年,但构件的概念一直没有统一的定义,在此我们站在测试的角度提出构件具有的属性,为CBD测试提供依据。

2.1　与测试相关的构件属性

·构件是系统独立的、可替代的部分。每个构件遵照并且提供一系列的接口。

·构件可见性(Observability),根据构件的操作行为,输入参数和输出,构件接口的设计和定义应使构件能够被观察。

·构件可追溯性(Traceability),构件具有追踪属性状态和行为的能力。

·构件可控性(Controllability),构件输入/输出、操作和行为易控制。

·构件可理解性(Understandability),构件可提供的信息和以及信息质量的好坏。

·构件有可能分布在不同的计算机中,它们之间的交互需要中间件(middleware)技术,如CORBA、DCOM等。

基于以上构件的特性,给测试CBD软件系统带来一系列的问题。

2.2　构件软件系统存在的测试问题

·构件的上下文相关开发问题

$TestS \in C$-adequate-on-$P1$,但不一定$\in C$-adequate-on-$P2$。其中$TestS$表示测试套件,C表示用来衡量测试充分性的测试标准,$P1$表示构件最初的应用环境,$P2$表示构件以后的应用环境。

对于构件提供者,在开发构件时缺乏构件将来被使用的应用环境信息。构件开发者需要在开发时预先对构件设想一个在将来被使用的应用环境,然后根据这个需求进行设计和开发。这样的上下文相关开发存在的问题是,假想的构件应用环境不一定是构件将来真正被使用的环境,构件的上下文相关开发使得对构件的测试也依赖于假想的上下文环境,一

*)基金项目:重庆市自然科学基金重点(CSTC,2006BA2003);西南师范大学科技基金项目(SWNUQ2005011)。

旦实际的上下文环境发生变化，构件必须重新进行测试。构件的提供者需要使用相当充分的覆盖准则进行测试以提高构件的可复用性；对用户使用的构件的上下文环境并不十分了解；并且不能很好地获得构件的错误报告。

·构件使用者对构件提供者的依赖

对构件的使用者而言，源代码不可见性给测试设计和用例生成带来了极大的障碍；构件使用者在测试过程中发现构件的缺陷引起的问题，然而对构件缺陷的隔离或移除需要依赖构件提供者，只有构件提供者才有构件的相关文档、测试计划和源代码，因此构件使用者通常要依赖构件提供者的维护和支持。建议在构件提供者和使用者之间建立转让契约和保护许可保证，防止提供者拒绝提供对构件的维护和支持。

·不充分的文档

对于构件使用者，在开发基于构件的系统时，缺乏相应构件的详细文档。构件提供的文档中信息可能不符合期望的语法和语义，或者是不充分的。不充分的文档带来的问题是，构件使用者必须对构件进行单元级别上的重新测试。

另外，系统中构件的异质(混杂)性不利于测试标准的统一及自动化的实现；构件版本变更快及不确定性要求对构件系统进行频繁的回归测试等[1]。构件软件开发过程中构件提供者和使用者之间缺乏必要的信息交换是给测试带来一系列问题的主要原因。

3 构件软件的测试策略

构件软件存在的测试问题给测试基于 CBD 方法开发的软件系统带来了挑战。在测试活动中，测试策略把软件测试案例的设计方法集成到一系列已周密计划过的步骤中，使测试过程系统，完整。本文提出一种系统化的测试策略 STSofCBS，指导 CBD 软件的整个测试生命周期活动。

如图 1 所示，整个测试过程迭代和递增的进行，迭代的测试适应开发过程中逐步求精的方法，使测试过程和开发过程紧密结合。

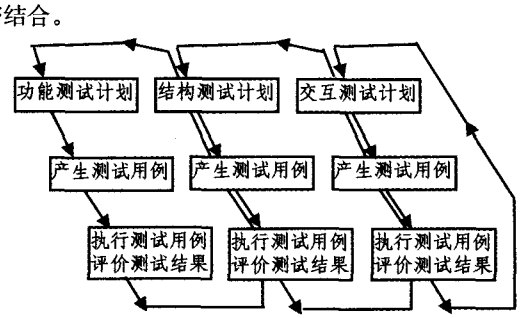

图 1

3.1 建立测试计划

STSofCBS 第一阶段建立测试计划，确定所使用的构件模型和架构；决定用于分析和测试构件系统的相关技术，测试用例的产生技术；并且要制定测试完成的标准。

测试计划在开始构建系统时就应该给出，指导后续的整个测试过程。测试计划可以是文档的形式，只是对整个测试的一个初步的规划，有可能还有进度和时间等的安排。它和开发过程的计划一致，针对 CBD 开发的特点，对于构件系统的测试在开发和构建系统的最初就开始，有利于更早的发现错误。

3.2 测试用例的产生

依据测试计划，在每次增量时关键就是测试用例的产生。对于 CBD 系统，在测试的过程中最大的问题就是构件提供者和使用者之间缺乏必要的信息交换。使用者用现有构件构建软件系统，但对于测试分析时所需的构件相关信息，如控制依赖和数据依赖、源代码等，一般不会提供给使用者，这给系统的测试带来挑战。虽然构件提供者已经对构件进行了充分的测试，但集成环境的不同要求使用者在集成构件系统之前必须对构件进行重新测试，这给系统的构建和测试增加了工作量。STSofCBS 策略第二阶段生成测试用例，又分为两个主要步骤：

步骤 1 采用自测试(self-testing)方法。

构件自身增加分析和测试方面的功能性规约，通过执行构件使用者测试过程中一些或全部活动，这种增强的构件能够测试内部的方法和行为。自测试性使得在运行构件时自动产生测试所需的技术信息，并在内部作为封装的方法使用，对于构件使用者产生信息的过程是透明的。

针对构件使用者通常不能在提供者那里获得测试所需相关信息(如源代码或其他的技术信息)的问题，自测试性允许构件使用者在没有这些信息的情况下也可以对构件进行测试，并且可以获得关于构件的测试技术信息。关于自测试方法的详细信息在文[1]中可以找到。

步骤 2 在获得测试相关信息的情况下，采用我们在前一篇文章中提出的构件交互图，构造测试模型，根据测试模型产生测试用例。

定义 1(构件交互图) 构件交互图 $G=(V,E)$，V 表示系统中的构件结点，构件接口和事件间的交互用一条边连接，以 E 表示。$V=VI\cup VE$，VI 代表构件接口，VE 代表由构件产生的事件。构件交互图中的一个路径或子路径是一个确定的有限序列$(v_0,v_1,\cdots,v_k)$，其中 $e_{i,i+1}=(v_i,v_{i+1})\in E$，$i=0,1,\cdots,k-1$。一个简单路径是路径$(v_0,v_1,\cdots,v_k)$，对于任意 i 和 j，$i\neq j$ 时 $v_i\neq v_j$。

在构件交互图中，所需的测试相关的信息：接口、事件、控制依赖关系和数据依赖关系。结合构件提供者提供的信息(如构件接口、功能描述、用户参考等)以及在自测试方法中产生的测试相关信息，可以构造构件相关的接口、事件、控制依赖和数据依赖等信息，帮助建立构件交互图模型。对于控制依赖和数据依赖关系在构件交互图中给出如下定义：

定义 2(控制依赖) 如果在图中存在一条简单路径 $v_0,v_1,\cdots,v_k$，其中 $v_i\in V$，$i=1,2,\cdots,k-1$。边 $edge=(v_i,v_{i+1})\in E$，我们就说两接口间或接口与事件间有控制依赖关系。

定义 3(数据依赖) ①如果 I_1 调用 I_1'，I_2 调用 I_2'，且 I_1'依赖 I_2'，则接口 I_1 和 I_2 有数据依赖关系。②如果 E 控制依赖于 I_1，并且 I 依赖于 I_1 或者 I_1 依赖于 I，那么接口 I 和事件 E 有数据依赖关系。③如果 E_1 控制依赖 I_1，E_2 控制依赖 I_2，并且 I_1 依赖于 I_2，那么事件 E_1 和事件 E_2 有数据依赖关系。

在构件交互图测试模型的基础上，根据一定的覆盖准则，覆盖图中的路径，产生相应的测试用例。具体方法参照文[11]。

3.3 测试执行

STSofCBS 第四阶段执行测试，根据上一步生成的测试用例执行一系列的测试活动，产生测试结果。

测试的执行可以根据产生的测试用例，指导人工的执行测试，一般在系统测试功能性时较多的采用，属于黑盒测试。在结构测试和集成测试范畴，测试的执行可采用动态技术，选

代的产生测试用例并执行，是白盒和黑盒技术的结合。可以借助自动化的工具，工具的开发是我们研究的下一步。

3.4 测试评价

STSofCBS第五阶段测试评价，根据STSofCBS第一阶段测试计划中给出的测试完成标准，构件使用者在测试执行的过程中对比测试的结果和期望结果之间的关系，对整个测试做出评价，给出测试总结和分析报告。

测试评价有时也包括对测试过程有效性的评价，可以从两方面来看。首先，在整个测试过程中测试用例的有效性，可以通过工具检查测试用例在执行过程中的覆盖率。其次，可以通过比较测试过程中注入错误的数量和测试后找到错误的量之间的关系，评价不同测试级测试的有效性。

4 实验数据

为了验证测试策略有效性和可行性，我们对两个基于构件开发的系统做了测试实践，两个系统基本信息在表1中给出。

表1

	构件	接口	事件
系统Ⅰ	18	1037	2245
系统Ⅱ	15	852	1843

系统Ⅰ是一个大型贸易管理软件的子系统，在系统中有18个构件，其中有部分是第三方构件，1037个接口；在系统发布前QA(quality assurance)测试中，72个错误被发现，系统发布后维护过程中23个错误被发现并纠正(系统Ⅱ的情况在表1中也相应给出)。通过分析错误报告和系统代码，把95个错误分为三种类型，在表2中给出。

表2

	类型A	类型B	类型C	总数
系统Ⅰ	33	38	24	95
系统Ⅱ	24	27	17	68

可以看出35%错误属于类型A，40%属于类型B，25%属于类型C。

另外我们还做了对比实验，采用传统的测试技术和本文提出的测试策略对两个系统进行测试。对于系统Ⅰ，采用传统的测试策略执行了1388个测试用例，发现了系统中72个错误；采用本文中提出的测试策略，执行了875个测试用例，发现了83个错误。结果见表3(系统Ⅱ的情况也在表3中给出)。

表3

		测试用例	类型A	类型B	类型C	总数
系统Ⅰ	错误报告		33	38	24	95
	传统测试	1388	27	29	16	72
	STSofCBS	875	26	37	20	83
系统Ⅱ	错误报告		24	27	17	68
	传统测试	1149	19	21	12	52
	STSofCBS	716	20	25	15	60

实验结果表明，对CBD软件系统进行测试，采用传统的测试可以发现76%的错误，本文提出的测试策略可以发现系统中84%的错误，并且在测试过程中发现了几个原来没有发现的新错误。

STSofCBS测试策略用比传统测试少得多的测试花费，可以发现比传统测试更多的系统错误，该方法在测试CBD软件系统时发挥了一定的作用。

结论 目前CBD软件测试重点在于构件源代码不可见性带来的测试问题，构件提供者和使用者之间缺乏必要的信息交换。本文提出的测试策略STSofCBS针对构件软件测试的主要问题，采用增加构件的自测试功能性的方法获得构件相关测试技术信息，用构件交互图作为测试模型，采用文[11]中提出的覆盖准则，产生测试用例，对CBD软件系统进行有效测试。测试策略使得整个测试过程系统化，避免了测试的偶然性带来的时间和工作量的浪费。

现有的对构件软件的测试方法和策略还在不断的探索和研究阶段，在未来的工作中，针对现有方法的局限，我们应从以下几个方面进行改进：(1)构件提供者和使用者之间交换的信息，研究统一的业内标准。以标准化方式支持构件分析数据的表示。(2)结合程序切片技术研究回归测试用例的选择和优化。(3)自动化的测试工具的研究。针对普遍使用的构件模型研制和开发自动化或半自动化的测试工具也是极具现实意义的课题。

总之，我们应以方法研究为起点，结合实证研究和工具的开发，在测试技术和方法上发展和完善构件测试的理论基础，研究测试的充分性判据，解决跨平台、跨语言的测试问题。

参考文献

1 Beydeda S. Self-Metamorphic-Testing Components. In: Proceedings of the 30th Annual International Computer Software and Applications Conference(COMPSAC'06)0-7695-2655-1/06,2006

2 Harrold M J. Testing: A roadmap [C]. In: Proc. of the Future of Software Engineering(Special Volume of the Proceedings of the Int'l Conf on Software Engineering). New York: ACM Press, 2000. 63～72

3 Orso A, Harrold M J, Rosenblum D. Component metadata for software engineering tasks. In: International Workshop on Engineering Distributed Objects (EDO), volume 1999 of LNCS, Springer Verlag, 2000. 129～144

4 Chan W, Chen T, Lu H, et al. A metamorphic approach to integration testing of context-sensitive middleware-based applications. In: International Conference on Quality Software (QSIC), 2005. 241～249

5 Edwards S H. Toward reflective metadata wrappers for formally specified software components. In: OOPSLA Workshop Specification and Verification of Component-Based Systems, 2001

6 Bundell G A, Lee G, Morris J, et al. A software component verification tool. In: International Conference on Software Methods and Tools(SMT), IEEE Computer Society Press, 2000. 137～146

7 Wang Y, King G, Wickburg H. A method for built-in tests in component-based software maintenance. In: European Conference on Software Maintenance and Reengineering (CSMR), pages. EEE Computer Society Press, 1999. 186～189

8 Cao J, Gupta K, Gupta S, et al. On building testable software component [G]. In: Proc. of the COTS-Based Software Systems (ICCBCC), LNCS2255. Berlin: Springer-Verlag, 2002. 108～121

9 Hornstein J, Edler H. Test reuse in CBSE using built-in tests [C]. In: Proc. of the Workshop on Component-Based Software Engineering, Composing Systems from Components. Los Alamitos, CA: IEEE Computer Society Press, 2002. 11～14

10 Gao J, Zhu E Y, Shim S, et al. Monitoring Software Components and Component-Based Software [C]. In: Proc. 24th Annual International Conference on Computer Software and Applications, 2000. 403～412

11 曹严元，张为群．一种基于CBD的软件测试方法．计算机科学，2005，32(2)

12 景涛，白成刚，等．构件软件的测试问题综述．计算机工程与应用，2002，24

13 Brown A W. 大规模基于构件的软件开发．赵文耘，张志等译．机械工业出版社，2003