CBS 测试策略研究*)

曹严元

(西南大学计算机与信息科学学院 重庆 400715)

摘 要 基于构件的软件系统(CBS)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,给出相应的测试策略,并提出改进的建议。

关键词 CBS,软件测试,测试方法,测试策略

1 引言

使用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,由于对软件功能性、开发效率、质量、可靠性、可移植性等方面的良好支持而受到越来越多软件开发组织的重视。基于构件的软件(CBS)开发方法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。同时,CBS也引入了新的问题,即测试构件的问题。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。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;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]。构件软件带来的主要测试问题在于构件提供者和使用者之间缺乏必要的信息交换。本文针对构件软件测试问题中的这个主要问题给出相应的解决方法和策略,并通过对比和评价,在最后给出一些改进的建议。

2 构件软件存在的测试问题

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。

2.1 构件的上下文相关开发问题

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,一旦实际的上下文环境发生变化,构件必须重新进行测试。

2.2 构件使用者对构件提供者的依赖

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。

2.3 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。

3 构件软件测试策略

针对构件提供者和使用者之间缺乏必要的信息交换的问题,给出相应的测试策略和方法。这些方法可以被归为两类:

第一类,关注引起交换信息缺乏的原因,主要目的是避免交换信息缺乏的发生。

第二类,关注发生交换信息缺乏问题时的处理,主要目的是处理导致的问题。

3.1 构件的元数据(meta-data)方法

构件元数据方法[2]通过在构件中增加信息,使构件提供者和使用者之间有足够的信息交换,该方法旨在避免交换信息缺乏引起的测试问题。

构件元数据方法对构件进行增强,把构件使用者需要的信息以元数据的形式加入到构件里,扩展了用于构件分析和测试的信息,使之包含语义信息。元数据的提供机制具有灵活性,采用开放形式提供,使得在必要的时候可以进行变更,元数据也可以不必被包装进构件并能实现按需(on-demand)生成和远程存储。构件提供者可以根据构件使用者现行的需求选择信息构造元数据。元数据具有以下三个特性:由构件提供者生成、按标准格式组织、可由工具

*)西南大学青年基金项目(SWNUQ2005011)

支持处理。

使用元数据方法，构件提供者可以对构件使用者提供必要的测试活动的支持。例如提供者可以提供构件的抽象级别上的数据流图信息，构件使用者根据数据流图可以不用获得构件源代码，解决测试中依赖源代码的很多问题。

元数据描述了构件的静态和动态两方面特性，可提高测试的精度；不仅提供语法信息而且包含了语义信息；灵活的机制使得可以根据构件使用者的需求进行变更；但所有构件提供者之间要达成元数据描述标准还有一定的难度；该方法目前只适用在小型软件测试中；而且仅支持从开发者向使用者的单向信息流表示。

3.2 反射包装器(Reflective wrapper)方法

反射包装器方法[4]使用包装器这种特殊的体系结构实现构件提供者到构件使用者的信息交换，是一种避免交换信息缺乏引起测试问题的策略。

信息流和元数据方法一样，仅支持从构件提供者向构件使用者的单向信息流。构件提供者包装信息到构件中，构件使用者可以通过人为方式或工具恢复出信息。包装器一般不会改变构件原有的接口，包装的主要是一些能够为测试提供信息的方法(如构件的规格说明、构件的行为描述以及质量保证等)或者是限制使用的某些功能，构件使用者可以根据恢复出的信息，便于测试的实施。图 1 是一个基于包装器构件测试框架。

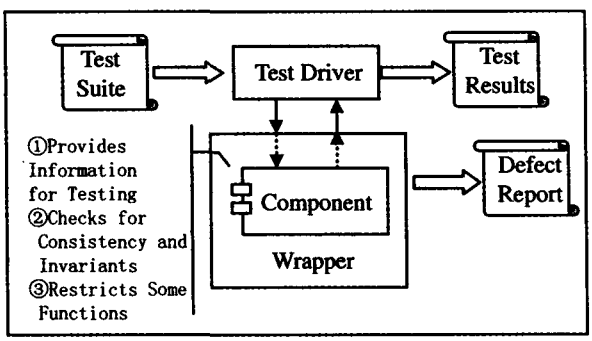

图 1 基于包装器的构件测试框架

与其他方法不同的是，包装器对用户是透明的，通过执行构件的相同的接口和构件的规格说明保持一致。包装器还具有相当的灵活性，当测试完成后，可以卸去包装器从而减少构件运行时占用的资源。

3.3 Retro 构件(Retro-components)方法

Retro 构件方法[3]增强的构件信息包括两方面信息，除了提供给使用者已测试的历史记录和进一步测试的推荐信息外，使用者的测试和运行信息可以反馈给构件开发者以便质量的进一步改进。回顾方法也是避免交换信息缺乏的一种策略，相对元数据方法，它可实现构件提供者和使用者之间的双向信息交换。

Retro 构件方法中信息的交换有两种，一方面，信息的交换可以是静态的，Retro 构件可以提供给构件使用者测试引导、充分性准则的使用和测试历史，此外静态信息还可以包含对进一步测试的推荐信息；另一方面，信息交换也可以是动态的，Retro 构件具有计算测试用例集充分性和收集描述它的使用信息的能力。Retro 构件能够自主地在测试和运行构件时收集信息，这些信息提供给构件生产者和使用者，使构件生产者对构件的使用情况更加了解，构件使用者在缺乏源代码情况下可使用源码覆盖分析方法。但由于 Retro 构件方法没有形成构件标准，构件提供者不一定提供该方面支持。

3.4 CTB(Component test bench)方法

CTB 方法[5]和前面几种方法一样，也是对构件增加用于分析和测试的附加信息。该方法是一种避免交换信息缺乏引起测试问题的策略。

与前面的方法不同的是，该方法提供的信息由某种规范来约束，称为测试规格。其中描述了构件的行为、每个行为提供的接口以及接口的适当的测试用例集。用户可以在实际系统中使用这个规范进行测试。测试规格以 XML 的形式提供，可以不受平台的限制。另外可以由第三方工具执行测试规格，对测试规格可以进行读取、解释和修改。

该方法提供了一系列构件测试工具，可以把 XML 形式的规范转化为 C 或 JAVA 执行。可以以三种方式生成测试，手工的、计算机辅助的和自动化的，比较灵活；使用 XML 语言描述测试规格，可移植性好；支持符号执行，可以通过符号执行得到测试输出而无需实际运行测试。但存在的不足是符号执行速度比较慢。

3.5 BIT(Built-in test)方法

BIT 方法是一种新的构件规范，基本思想是构件提供者在构件中预置测试脚本并设置相应的测试接口，构件使用者可调用该接口实施构件的自行测试，脚本通常是测试用例或能够产生测试用例的语句。该方法是旨在处理交换信息缺乏引起的测试问题的一种策略。

```
Class BIT-INT-Stack {
  int stack [ N ] ;
  int stack-index ;
  BIT-INT-Stack ( ) ; //The constructor
  ~ BIT-INT-Stack ( ) ; //The destructor
  //Normal member functions
  int BITs-Stack-Empty ( ) ;
  int Pop ( ) ;
  int Push (int element) ;
  //Built-in test implementation
  void BIT-INT-Stack-Test1 ( … )
{ … } ;
  void BIT-IN T-Stack-Test2 ( … )
  } ; //The IN T-Stack component
```

图 2 一个内建式测试构件的例子

文[6]中的 BIT 方法，通过在构件中增加能包含或产生测试用例的附加成员方法来增强可测试

性，如图2的示例代码，构件可以运行在两种模式下，标准模式(normal mode)和维护模式(maintenance mode)，在标准模式下，构件内建测试信息对构件使用者是透明的，在维护模式下，构件使用者可以调用能驱动测试的代表性方法执行测试，前提是构件必须被假定为一个类，所内建的用于测试的方法可以被子类继承。

文[7]针对EJB构件，内建能和外界工具交互的接口和用于追踪的函数来确保使用者在执行测试时能方便地观测构件行为，并实现了两个测试辅助工具：Test Agent和Tracking Agent。其可观测性强，利于使用者检测出构件的异常行为，但只限定于特定的构件。

为克服内建式测试用例存储开销过大的问题，文[8]提出了"Component+"的测试方法，将要开发的构件分为：有内建测试能力的构件(BIT构件)；能访问BIT构件接口和包含测试用例的构件；用于运行时失效恢复等用途的构件。这种分类开发构件的方法一定程度上实现了开发和维护的分离，不会增加BIT构件在运行环境中的资源消耗，但会加大使用者部署测试的难度。

内建式的思想可增强构件的可维护性，它的用处可以不局限于测试。但这种方法需要源代码的支持。

3.6 Testable beans方法

文[7]中提出的方法，通过改进构件的可测试性来处理缺乏交换信息引起的测试问题，该方法提出的具有可测试性的构件应具有支持测试执行和测试观察的能力。

一个testable bean应具有下列特性：

(1)一个testable bean是可配置和执行的。它可以像一个一般构件一样被使用，不需要特殊地提供操作。

(2)一个testable bean是可跟踪的。构件使用者可以在测试时观察它的行为。

(3)一个testable bean执行测试接口。它执行一个一致和良好定义的测试接口。

(4)一个testable bean必要时可和外部测试工具进行交互。

testable bean的测试接口和一般构件的接口对构件使用者来说是有明显区别的。测试接口声明三个方法，一个方法初始化测试，被测试的方法以及测试用例；第二个方法执行初始化的测试，第三个方法对测试给出评价。这些方法被测试接口声明并可以被testable bean自己执行，也可以被外部的工具执行。

3.7 程序分片、控制依赖分析和数据流测试方法

文[9]方法首先由构件提供者对构件进行充分测试，提供summary information；构件使用者通过得到的summary information可以进行有效的系统测试，不需要得到构件的源代码。举出三个应用方向，程序分片(program slicing)、控制依赖性分析和数据流测试。根据构件提供者提供的不同附加信息，构件使用者可以进行不同领域的分析。

该方法中summary information并不属于任何现有构件规范标准，因此要得到构件提供者的支持难度比较大。

结论 目前CBS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。主要的解决思路是由构件提供者提供更多的附加信息以便使用者进行测试。本文给出的几种测试策略针对构件软件测试的主要问题，提出增强信息交换的方法和机制。但都有各自的应用环境和针对性，也存在局限性。

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)内建式测试中如何进行交换信息的表达、理解和提取等问题需进一步明确化以便实用。(3)结合程序切片技术研究回归测试用例的选择和优化。(4)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 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

2 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

3 Liu C, Richardson D. Software components with retrospectors. In: International Workshop on the Role of Software Architecture in Testing and Analysis, 1998. 63～68

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

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

6 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), IEEE Computer Society Press, 1999. 186～189

7 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

8 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

9 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

10 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

11 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