计算机科学 1999Vol. 26№ 11

面向对象软件测试的关键问题及对策

Key Problems and Considerations for Object-oriented Software Testing

顾玉良 王立福 吕慧颖

(首都师大计算机系 北京大学计算机系 北京 100037)

Abstract Although some popular methods for object-oriented software development have been extensively used, theories and practice of OOT are not mature. Methods for structural testing have many suitability problems in dealing with complexity of software and issues resulted from object-oriented methods. Based upon the study in [1], the paper discusses some key issues of OOT and relevant solutions. Then, it provides a technical framework for OOT, which includes technique aspects and process aspects of OOT to support life cycle of OOT.

Keywords Object-oriented software testing, Test modeling, Test adequacy standard, Test process, Software testability, Framework

1 引言

目前,面向对象软件开发方法占主导地位,但是,面向对象软件测试的研究却远远滞后于软件工程实践的需要,软件规模和复杂程度不断增加、应用软件及其运行环境,如操作系统、异构平台、分布计算环境等方面,已经大大地复杂了,传统的结构化软件测试技术受到了前所未有的挑战。

尽管如此,结构化测试方法的许多思想依然是可用的,但是需要克服其局限性,其主要表现在:(1)通常只关注一些局部或某个层次的问题、并作了过分的简化;(2)很少考虑被测对象的可控制性和可观察性问题。

Binder 和 Barlay 等研究了面向对象特征,如封装、多态和动态绑定、继承等,认为这些特征的引入增加了测试的复杂性[2,3]。另一些研究则考察如何利用继承进行子类的测试,Fielder 认为在父类充分测试的基础上,被子类继承的方法只需最小测试[4]。Harrold 则主张,在测试子类时、只需对被继承的属性和新的属性之间的交互进行测试[5]。但是,并没有给出关于最小测试范围、交互的判定标准和相应规则。

从软件工程的角度,对于软件测试,需要一组相关的技术、从单元级、集成级和系统级的软件测试周期内研究这些技术、它们的适用性以及它们之间的协作问题,但很多研究忽略了这些、从而难以满足软件工程实践的实用性需求。

2 测试层次

软件层次测试基于测试复杂性分解的思想,是软件测试的一种基本模式、即从最小部件(单元)的测试开始,逐步组合,直至完成整个系统的测试,这样就构成了测试的层次。传统的层次测试基于功能模块的层次结构;在面向对象软件测试中,继承和组装关系刻画类之间内在的层次,它们既是构造系统结构的基础,也是构造测试层次的基础。Seigel 描述了从类、基础部件、系统部件到应用系统的面向对象软件的层次测试[7],测试集成的过程是一个基于可靠的部件组装系统的过程。

一个类簇由一组相关的类、类树或(和)类簇组成。类的继承关系、组装关系以及类簇包含关系等可以自然地构造相应的层次结构,它们的语义决定了自然的测试次序,从而生成相应的测试层次结构[1]。这里,自上而下、自下而上和两者结合的测试都可用,一个被测单元或部件可以是一个类,一组类或者一个程序框架,在其中可以包含未经测试并且本级测试不准备测试的类或一些测试桩类。

某些类之间的高耦合性要求对某些类树或类簇作为最小单元进行测试。对于类树,通过底层类的实例进

顾玉良 博士,主要研究领域为软件工程和软件测试。王立福 教授,主要研究领域为软件工程。吕慧颖 讲师,主要研究领域为软件工程和软件测试。

行测试，但子类可能重置父类中的方法，那么父类中对应的方法就不能被测试到，这时需要对类树进行分割；如果对某个类簇进行测试时，其中某些类无法测试到，则需要分割该类簇。

许多对象的某些行为可能只在特定的环境中表现，那么对它们的测试不能离开该环境。由于可控制性和可观察性的原因，即使某些类或类簇进行了单元测试，但它们依赖于环境的部分很难独立于特定环境而被测试。某些类无法进行独立的单元测试，只能在某个集成测试层次上进行测试，甚至可能到系统测试时才可能进行测试，如具有图形界面的面向对象软件中的界面类[3]。

所以，在确定测试层次结构[3]后，并不是每一个结点都需要实施独立的测试，因为对类或类簇进行包装以产生可执行测试代码通常会付出相当大的代价。选择测试层次的因素主要包括组织内部标准相关的因素（如测试代价）和软件特性相关的因素（如软件结构复杂性、可控制性和可观察性、对象交互的方式等）。

3 测试模型及测试建立

被测对象的模型是建立和执行测试的基础，模型中包含了需要测试什么和如何建立测试的信息。在面向对象软件开发过程中，大都在分析设计文档中给出了目标系统的模型，而且是从多个角度给出模型，有静态模型，也有动态模型。现有的开发建模方法旨在建立目标系统的模型表示，使得程序编写人员能理解系统，正确实现其功能。测试的关键问题是如何发现故障和提高发现故障的能力，但是这些模型并不提供便于建立测试的规则。它们不包括测试用例的形式和如何建立测试的知识，有时会有一些，但不完整。所以，需要对这些模型进行转换，才能用于测试。在实际的软件开发过程中，所给出的模型又常常是不完备的，所以测试所用的模型又常常需要增补一些信息。

3.1 选取建模范围

从可测试性角度考虑，在被测部件中，并非所有的对象是对等的，在建立测试时所选取的建模的侧面和采用的建模方法也不同。所以，需要关注不同类型的对象所具有的不同特征，以决定测试建模的范围。测试建模的范围主要依据以下几方面来确定：

- 不同特性对象的划分；
- 不同的测试特征；
- 不同测试特征的抽象级别。

图1表示一个通用的测试支持系统结构。在被测部件中，从测试角度来看，存在两类对象：边界对象和内部对象（非边界对象）。

（1）边界对象（B对象），测试驱动器与被测部件，目标系统运行环境与被测部件的边界上存在一类对象，称为边界对象，可以是用户界面对象，也可以是程序内部对象（如一般的C++对象），典型的是界面类对象。边界对象具有如下特征：

- 对象对于测试驱动器是可识别或定位的，
- 在测试执行过程中，实例数量有限。首先只有数量有限才可能识别和定位，其次在测试执行过程中所消耗的存储资源较少，以避免非期望的环境例外，
- 对象是消息可达的，
- 对象通常有较好的可控制和可观察特性。

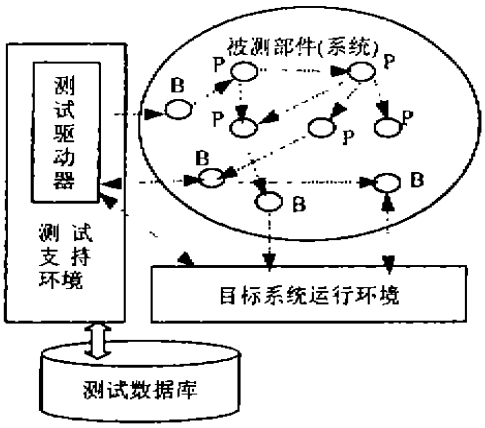

图1 测试支持系统结构

（2）内部对象（P对象），对于内部对象，有时可以有一些观察特性，有时可能无法观察，通常不能或不能直接地定位，所以也就无法直接地控制。

在确定了边界对象和内部对象后，需要有不同建模方法的支持。

3.2 决定建模类型及方法

为了检查不同类型的错误，面向对象软件测试不依赖于单一模型，因而需要从不同的侧面建模被测对象。总的来说，模型可以分为（功能）行为模型和结构模型两大类，根据不同的软件类型和可测试性目标，可以采用不同的方法进行测试建模[8-10]。

一般地说，面向对象软件比结构化软件更加强调被测对象的行为。从类到类簇，从类簇到整个系统，可以用相似的观点看待它们：都是一个相对独立的系统，通过内部交互和系统边界的交互而提供外部所需要的功能，从而为单元测试、集成测试和系统测试提供一致的行为视角。在单元测试中，测试对象是一个程序单元，包括一组程序对象类。在对程序单元测试时，需要建立测试能力，使该程序单元具有明确可识别的测试接口。在集成测试中，测试对象由一组已经经过测试的

程序单元和一些未经测试的程序单元构成，包括一组程序对象类，集成视图关心不同的程序单元之间的交互测试，相应的测试覆盖标准与单元测试不同。在系统（或确认）测试中，测试对象是整个应用系统，包括相应的运行环境组成一个整体。

对于不同的测试级别，有不同的测试目标，相应测试建模的范围和建模的方法需要考虑这些目标，必须考虑运行平台对于测试的支持程度。

3.3 建立附加的可测试特性

根据测试模型生成的测试用例覆盖需要测试的软件特征，测试用例的执行首先要求被测对象具有用例所规定的可控制性和可观察性，它们是可测试性的两个关键刻面[7]。如果不能控制输入，就不可能确认由什么导致了输出；如果不能观察被测对象的状态或输出，就不可能确认一个输入是如何被处理的。被测对象与测试系统之间的关系是造成可控制性和可观察性问题的关键因素之一，因为测试系统可能难于控制被测对象，被测对象的中间输出可能难于观察，图形接口可能降低被测对象的可控制性和可观察性，存取和跟踪输入输出在复杂的运行环境中存在很多困难，等等。

在面向对象程序中建立控制和观察机制可能的几个方法是：

- 在被测试的类中建立特定的方法，如：初始化方法、输出方法等；
- 生成被测试类的子类，并在子类中建立特定的方法，以避免直接改变被测试类；
- 对于被测试的对象，利用其自身方法的组合，通过外部的消息发送和输出检查达到控制和观察的目的[8]。

在测试时，需要使对象达到特定状态的机制，通常可以采用如下两种途径：

- 在对象内部，建立状态建立/复位机制，但这种方法存在局限性，因为一个对象的状态可能与其它对象有很多联系，建立这种机制的代价可能超过它带来的好处，所以，即使在类测试中也不是普遍适用的，在类簇测试中则基本不适用。
- 从对象实例化开始，由测试系统保证在被测对象的外部直接或间接提供触发消息，使对象达到特定的状态。这一途径可以保证对象内部状态及整个被测部件的一致性和完整性。

4 测试评价标准

从测试模型可以导出测试评价的标准。对于面向对象软件，需要根据软件项目质量标准的总体策略，从单元测试、集成测试和系统测试的不同层次上综合地建立完整和适用的测试评价标准。

4.1 影响测试评价的基本考虑

测试评价标准试图达到对于特定测试特征的规定的覆盖级别。在面向对象软件测试中，针对对象的封装特性，宜采用功能技术，但针对结果检查，如检查对象的状态，识别测试执行过程中的某一个对象，观察某些执行序列，需要直接或间接采用结构技术。所以，结构测试技术和功能测试技术通常被综合使用。

在结构化程序测试中，结构覆盖标准（分支类和数据流类）是比较成熟的标准，但是，在面向对象程序中，结构覆盖标准需要考虑对象边界。子类化、多态性、对象聚合和包容特性、不同类型的对象之间的交互协作特性等也都是重要的因素。

其中，在考虑结构覆盖时如何看待测试路径是决定一个测试覆盖标准的基础。测试用例对应一条测试路径，如何从路径的角度看待面向对象软件的测试用例，可以采用两种策略，一种是破封装策略，另一种是分级策略。破封装策略忽略对象边界，把消息视为跨越对象边界的"调用"。其缺点是：

- 需要静态地确定接收消息的对象所属的类，以决定其引用的方法。
- 由于对象系统不是按功能分解的，从功能角度看待对象系统的测试路径时，其长度相对较长，组合问题比过程式软件更复杂。
- 降低测试可复用性。

由此可见，破封装策略对于面向对象软件测试不是一种好的策略。分级策略则基于面向对象的封装特征，在对象之间，对象通过消息进行交互，在对象内部，各方法通过相互调用交互。在测试角度，把消息和调用明确区分，消息表达对象之间的交互，调用则体现对象内部的交互，这样可以部分地避免上述问题，并且使技术复杂性得到分解。

4.2 三类覆盖

以下三方面的覆盖是需要的，它们组成了对于面向对象软件测试的覆盖的三个维。

- *结构覆盖* 可以包括从程序的代码到分析模型中的结构模型，但主要是程序代码级结构模型。对于面向对象程序的结构覆盖，尽管传统的语句和分支覆盖等依然有用，但对于传统的数据流方法进行扩展则更具有潜力[1]，结构覆盖在单元级、集成级所采用的策略有比较大的区别，但通常不在系统测试级别上运用。
- *功能覆盖* 考虑一个被测部件的功能，即从被测部件的输入输出侧面，依据输入域与输出域的数据关联，检查被测部件的功能是否符合规范。功能覆盖可以用在单元级、集成级和系统测试级。
- *数据覆盖* 经常被混同于功能覆盖。事实上，数据覆盖关注数据域的特征，在实际实施时，经常会考虑

输入域与输出域的数据关联，这时它与功能覆盖类似，但在测试数据选择时其目标依然有区别。可以考虑被测部件的输入输出接口的数据域，也可以考虑对象内部的属性值。

在建立这三类覆盖标准时，需要考虑单元级、集成级、系统级覆盖之间的一致性和回归测试的覆盖策略。需要在不同的测试级别上研究不同覆盖类型的综合运用的方法。

5 测试层次与过程

根据测试层次结构确定相应的测试活动，并生成相应的层次。如图 2 所示，一个圆角的矩形框表示一次测试活动，框中的名字是活动名，一个活动可以是单元测试、集成测试或系统测试，带箭头的实线表示活动之间的层次关系，框的嵌套关系正好对应测试层次关系。带箭头的虚线表示在实际测试中存在回归测试。

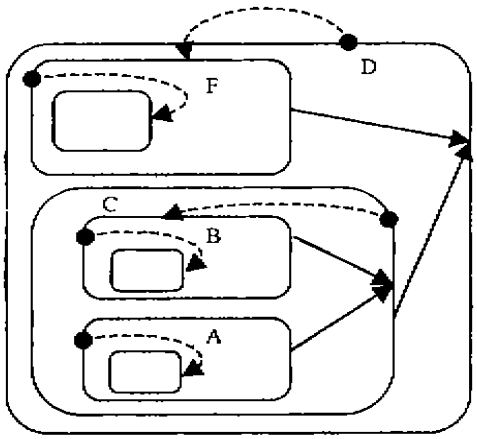

图 2 面向对象软件的测试层次

我们可以把这些测试活动安排在整个项目的测试过程之中[1]，测试过程包括了所有的测试实施活动（上下文清楚时简称测试活动），以及一些相关活动（如测试实施设计、测试实施建模、组织执行测试实施等）。在测试过程中，不同的测试在一定条件下可以并行实施，它们之间通过特定的约束进行同步。约束主要包括角色约束、设备约束、费用约束、时间约束、信息约束、代码约束和根据约束等。其中，代码之间的约束是一种重要的约束，因为在并行实施的大型软件项目开发过程中，不同的软部件之间不可避免地存在某些子项目的拖延，一些公共部件的开发和测试可能逾期，对已经测试完的部件，可能在其后的集成测试或系统测试中，又发现了独立测试时没有发现的故障，从而需要修改并且进行回归测试。

每一个测试活动包括两个时间段 T_1 和 T_2。T_1 用于进行预定的测试（单元级、集成级和系统级），T_2 则用于回归测试，T_2 的启动时间被延期的主要因素包括编码的延期及前级测试的延期；T_1 被延长的主要因素是测试时资源要求不满足，从而导致 T_2 的启动时间被延期；T_2 被延长的主要因素包括回归测试和资源的限制。其中，回归测试给一个测试活动带来了关键的不确定因素，在图 2 中，这种不确定被封装。当某一级活动发现软件故障，并负责定位故障单元，然后从相应单元测试开始，并选择一些级别进行回归测试。这样，相应的影响封装在本级，以控制对下一级的影响。

6 技术框架

综上，面向对象软件测试需要考虑多方面的技术内容，这里给出一个支持面向对象软件测试的框架，考虑完成整个项目测试的主要方面，规定一组适用的、协调的技术及原则，技术之间应该满足的基本关系或约束，包括过程和技术两方面的内容，是对多种技术的有机组织。在这里，技术和过程并非截然地加以区分，在考虑过程方面时，主要涉及活动的划分和组织，进度、资源和人力工具等如何配置，其中会运用相应的技术。通过这个框架，对于面向对象软件支持从单元、集成到系统诸测试的整个周期。

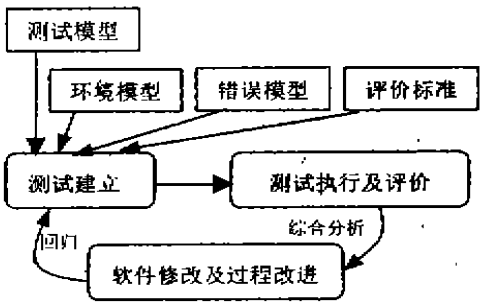

图 3 框架包含的技术及过程方面

如图 3 所示，软件测试需要考虑的关键因素，从其采用技术和与软件开发的关系来看，大致可以分为两个方面的内容：

- 在技术方面，根据工程质量目标和产品的特性，建立合适的测试模型，并根据模型生成合适的测试评价标准和一个适用的用例集；测试建立还包括测试环境的建立，测试驱动器的生成等，然后进行测试实施，并施用相应的软件标准。
- 在过程方面，通过对测试评价的结果进行综合分析，考虑相关因素，如进度、目标可达程度、市场、风险等等，决定修改有错软件的策略和进行回归测试的策略，相应地改进软件过程。

我们可以从以下几个方面说明该框架：

·过程 指软件测试活动的组织实施方面，以及测试活动与软件过程的其它活动之间的交互。对应于系统各组成部件的结构，测试活动(单元、集成和系统测试)的依赖关系组成一个层次结构，这在面向对象软件测试和结构化测试是相似的。由于面向对象软件开发适应于增量和迭代的过程，软件测试的可复用性、测试过程对于整个软件过程演化的适应性、测试过程的目标一致性和有效性非常重要。

考虑测试活动与软件过程的其它活动之间的交互时，回归测试的策略和软件测试结果的分析模型是关键的因素。软件质量保证的资源使用的类型也影响这两种因素，决定了两类测试过程。其中，可靠性驱动的过程要求通过软件测试使被测软件达到可靠性目标，时间和费用增加到测试上直至达到可靠性目标，具有较高可测试性的系统则可以降低增加到测试上的时间和费用。资源有限过程把测试看作一种在可允许的资源范围内尽可能地消除软件错误的过程，具有较高可测试性的系统可以在预算范围内进行更有效的测试，使得被测软件达到更高的可靠性指标。

·测试模型 模型及其可用性在测试分析和测试用例设计中是一个关键因素。它涉及从关于所需系统能力的自然语言语句到详细的形式规范、期望结果描述的清晰性，需求的客观性、可达性、可量化性，规范的完全性，表示的可跟踪性，一个软件部件是否实现了其规范，规范是否实现了其需求等等。

一个面向对象程序实现特征决定可观察性和可控制性，包括结构、故障敏感性、外部接口、决定性、例外处理，特殊性能等等。内部测试机制提供了清晰的测试和应用功能的分离，系统地给类加入设置/重置、报告、断言，以使测试接口标准化和系统化，从而使附加的开发代价最小。

·评价标准 它根据测试模型产生。整个测试项目中可能使用多种不同类型的模型，需要研究这些模型之间的关系，以生成一个从单元测试，集成测试和系统测试的一致的测试评价标准。

·运行环境 其特性对于测试建立是一个重要因素。它决定了在被测系统中哪些对象对于测试是可以控制的，哪些对象是可见的，其检查的效果是否可以表示运行用例的正确性。运行环境是否支持测试接口的标准化和系统化等等。

·测试工具 测试支持环境是有效测试所需要的，包括多种类型的工具集以支持不同的测试技术和过程方面。合适和可用的测试集(用例和相关信息)，包括用例集及其是否具有复用和可验证特性，相关的文档(测试计划，测试设计，测试用例，测试过程，测试结果，测试执行日志，测试事件报告，测试总结等)，需要测试工具的有力支持。

7 结论和进一步的研究

在软件工程领域，面向对象软件测试是一个重要的研究方向。本文在[1]的基础上，在面向对象软件完整测试周期中，研究面向对象软件测试的几个关键问题，并组织在一个技术框架，讨论面向对象软件测试的技术和过程问题，以为面向对象软件测试完整解决方案提供一个基础。

软件测试依据测试模型和特定的、被认为是好的、适用的测试评价标准(实际是覆盖标准)，运用于测试过程之中。特别需要研究的是在整个项目的测试过程中，在某些测试的级别上，典型的是在系统级测试和以后的性能测试、安全测试、适用性测试、用户测试等，需要研究一些用于对测试结果进行综合分析的数学模型，由此判定一个软件是否可以投放。

面向对象软件测试的问题是由于软件复杂性问题而引起的，客观存在的复杂性很难通过技术手段降低，新的开发技术并不能解决软件质量问题，软件测试技术受到软件固有的特性和开发技术的制约，实际上是软件测试技术与目前软件复杂的计算模式的不匹配，由此，软件可测试性可以看作是开发技术与测试技术之间的桥梁，系统地研究软件可测试性成为解决目前测试技术存在问题的关键途径之一。DFT(为测试而设计)[7]是一个与开发过程一致以使软件测试在可靠性驱动或资源有限过程中是最为有效的技术，其基本动机是考虑通过在开发过程中和软件本身提供可测试性，一方面可以降低测试技术难度从而可以降低测试本身的代价，另一方面，开发过程为之付出较小的代价，使最终的代价最小。

参 考 文 献

1 顾玉良. 面向对象软件测试技术研究:[北京大学博士学位论文]. 1998

2 Binder R V. Testing Object-Oriented Systems: A Status Report. American Programmer. April 1994

3 Barbey S, Strohmeier A. The problematics of testing object-oriented software. Building Quality into Software, 1995. 410～425

4 Fielder S P Object-Oriented Uint Testing. Hewlett-Packard Journal, 1989(April): 69～74

5 Harrold M J, Offutt A J, Tewary K. Incremental testing of object-oriented class structures. PSEIC'92 68～80

6 Seigel S. Object Oriented Software Testing. John-Wiley & Sons, Inc., 1996

7 Binder R V. Design for Testability in Object-Oriented Systems, 1994. 87～101

8 Frankl P G, Doong R K. The ASTOOT approach to testing object-oriented programs. ACM Trans. on Software Engineering Methodology, 1994, 3(2): 101～130

9 Binder R V. The FREE approach for system testing. Object Magazine, 1996, 5(9): 73～81

10 Hsia P Testing of Object-Oriented Programs. YIGA'97 Seminar