计算机科学 1999Vol.26№.4

面向对象软件测试过程研究*)

Research of Testing Process of Object Oriented Software

顾玉良　王立福

（首都师大计算机系　北京 100037）（北京大学计算机系）

Abstract This paper prompts a kind of hierarchical structure for OOT based on features of OO software, and then a model of software testing process. Software testing process is a set of testing enactment and related activities, including structural design of testing enactment, modeling of testing enactment, testing enactment and management, etc., to support testing of large software project.

Keywords OOT, Software testing process, Testing enactment

1 引言

软件测试是软件质量保证的重要方面，高质量的软件测试过程有助于提高测试的效率。目前，对测试过程建模的粒度选择、测试对于开发过程变化的适应性等方面还缺乏研究，已有的一些过程模型在形式化和可操作性支持等方面还不够。通过研究软件测试过程，使之具有良好的表示和实施可操作性等，这对于软件测试，尤其是大型软件项目的测试具有重要的意义，并有助于在软件工程环境中支持形成完整的测试解决方案。

McGregor 提出了一种测试过程模型—迭代增量模型[3]，该模型是测试过程的一个简单描述。Poston 给出了基于 OMT 方法的测试过程模型[6]，该模型考虑，测试过程与开发过程在时间上是可以重叠的，并强调从开发过程的结果中捕获用于生成测试用例所需的信息。Perry 给出了支持原型开发的测试过程模型[4]。Kit 和 Poston 根据瀑布开发模型给出相应的测试过程模型[2,5]。在开发过程中，存在许多可以为测试所用的信息，如果开发过程充分有序、规约（文档）严格、信息表示适度，则测试过程与开发过程可以相互支持。

2 测试层次结构

软件层次测试（Hierarchical Testing Approach）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，这样就构成了一个层次结构。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]。根据测试层次结构，面向对象软件测试总体上呈现从单元级、集成级到系统级的分层测试，测试集成的过程是基于可靠部件组装系统的过程。

2.1 测试次序

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分别可以自然地构造相应的层次结构，这些层次结构的语义决定了自然的测试次序：

• 对于继承结构，自然的测试次序是父类在先子类其后。父类可以看作其子类的公共部分，在父类是"测试安全"的前提下，子类测试可以关注子类独有的部分以及父类和子类之间的交互。

*）本项研究得到国家"九五"重点科技攻关项目资助。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。

·对于组装结构，自然的测试次序是部分类在先整体类其后。在部分对象类是"测试安全"的前提下，整体对象类的测试可以关注各个部分类是否能够按规约进行组装。

·对于类簇包含关系，自然的测试次序是先测试组成类簇的各个部件，而后对这些部件进行集成测试。

2.2 测试层次结构的生成

给定一个对象关系模型，相应的测试层次结构按如下步骤构造而成：

(1)根据对象关系模型，构造一个有向图 G：

$G=(V,E)$，其中，$V=\{V_1,V_2,\cdots,V_n\}$，$n>0$，E 是有向边的集合，每个结点代表相应的类，每个有向边连接其中两个结点。

构造 G 的方法是，如果对象关系模型中的两个类 A、B 存在组装关系，A 是整体类，则生成与 A、B 对应的结点 $V_i,V_j\in V$，及有向边 $e=(V_i,V_j),e\in E$；如果两个类 A、B 存在继承关系，A 是子类，则生成与 A、B 对应的结点 $V_i,V_j\in V$，及有向边 $e=(V_i,V_j),e\in E$。

(2)在 G 中消除循环，形成若干不相连的层次结构。由于只保留了继承组装关系，G 可能由一些没有关联的有向子图构成，其中还存在循环，但测试总是有序的，所以消除 G 中的循环，使得 G 由一些没有关联的层次结构构成。

(3)把以上形成的不相连的层次结构按类簇包含关系连接成一个层次结构。对以上生成的结构，如果若干类存在一个类簇包含关系，则生成一个对应的类簇结点 $V_i\in V$，对其包含的每个类(或类簇)$V_j\in V$，生成有向边 $e=(V_i,V_j),e\in E$，直至该生成过程遍历了所有类簇包含关系。这样，G 就成为一个层次结构。

在测试层次结构中，叶子结点代表一个类(部分类或继承层次中的祖先类)，中间结点和顶层结点可以是一个整体类、一个子类或者一个类簇，一个类或类簇可以参于多个类簇之中。

可以利用测试层次结构对相关信息进行组织，每个结点表明了该层次的测试范围，即从该结点出发沿有向边可以到达的类或类簇；对相关的部件进行组织，包括部件测试计划、测试模型、测试用例集、测试标准和采用的测试策略等；记录结点当前的测试状态，如已执行过的用例和已达到的覆盖、发现的故障、执行的时间等。

大型软件项目的开发一般采用分而治之的策略，典型地，可以用项目工作分解结构(Work Breakdown Structure)进行规划。对于软件测试而言，可以利用项目工作分解结构，在项目级规划测试的层次。不同级别的测试形成了一个层次结构，称为测试实施活动结构，与测试层次结构相对应。

3 测试过程

目前，已经提出了一些软件测试过程的模型，软件测试过程一词也已经被许多人使用，但还没有关于软件测试过程的确切定义。本文认为，从测试的角度，可以对软件开发过程中所有的测试实施活动，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)进行组织、从而形成测试过程。从软件开发过程角度来看，测试实施活动是开发过程中的活动；从测试过程角度来看，测试实施活动也是测试过程中的活动。测试过程中的活动相互协作，完成整个软件项目的测试。

3.1 过程模型

小组(个人)级测试层次结构对应小组(个人)开发的软部件，可以实施单独的测试；然后，这些部件在集成时进行中间级的测试；最后，在项目级完成整个系统的测试。在这里，测试实施包括小组(个人)级、中间级及项目级的测试。

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。约束主要包括如下类型：

(1)角色约束。测试的角色是测试人员及相关的人员和组织。人员总是有限的，需要在不同的活动之间进行分配，是对测试的一种约束。在开发过程中，有关人员可能没有按计划到位，从而拖延测试进程。

(2)资源约束。包括设备约束、费用约束、时间约束、信息约束和代码约束等。资源是有限的，它们被分配到不同的测试之中，是对测试的一种约束。其中，代码之间的约束是一种重要的约束，在并行实施的大型软件项目开发过程中，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。

(3)工具约束。每个测试需要相应的工具支持，

它们被分配给不同的测试实施活动。

根据测试层次结构，组织整个项目中的所有测试实施活动，形成与测试层次结构一致的结构，活动的约束由相应的角色、资源和工具表示。测试实施活动集成在开发过程之中，通过管理这些约束，使测试过程能较好地适应变化。同时，测试层次结构适应面向对象软件开发的特点，支持测试复用。

软件测试过程模型可以表达成一系列活动的集合，用以达到特定的目标，给实际的测试过程提供支持。基于测试层次结构，图1给出了一个软件测试过程的模型，主要包括如下活动：

(1)测试实施设计。其主要任务是，根据项目级软件过程和测试层次结构，生成描述不同测试实施活动和它们之间的关系结构，称为测试实施活动结构。测试实施活动是项目级软件过程中的活动，其中与测试实施活动相关的信息被记录在测试实施活动结构中，并且动态地跟踪项目级软件过程中与测试实施相关的变化。

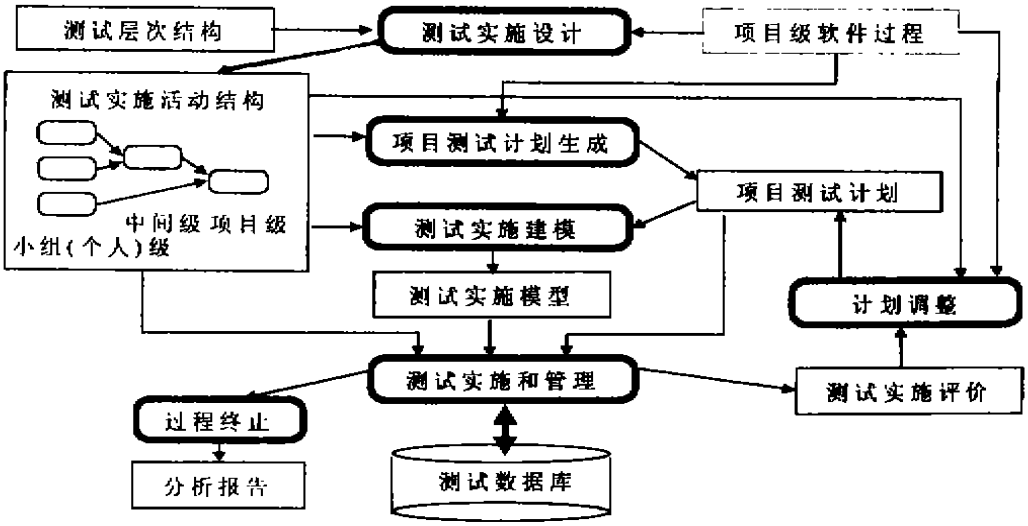


图1 测试过程模型

(2)测试实施建模。其主要任务是，对测试实施活动结构中描述的活动进行建模，生成实际的活动模型。

(3)测试实施和管理。其主要任务是，根据给定的测试实施模型、测试实施活动结构和项目测试计划，具体实施各个测试。该活动动态地启动一组测试实施活动，包括小组(个人)级、中间级和项目级的测试。

(4)项目测试计划生成。其主要任务是，生成项目测试计划，用以在项目级描述所有测试的范围、方法、资源和进度等，典型地，包括测试方法的选择、测试设备和支持软件的准备、配置管理、风险管理(预算、资源、安排、培训)等。

(5)计划调整。其主要任务是，根据测试实施过程中的问题和相应的评价，调整测试计划的有关内容，如资源、进度等。

(6)过程终止。在完成所有测试实施活动后，过程终止，并给出相应的分析报告。

3.2 人员组织

测试过程的实施涉及四类基本角色，即开发人员、测试分析人员、测试实施人员和测试支持人员。一个人可以担当多个角色，一个角色可由多个人担当。根据软件开发组织的组织策略，测试组织可以有多种形式。

测试分析人员分析产品以决定测试需求，不要求开发背景，可以既是测试者也是开发者。测试分析人员必须理解测试观点，熟悉整个项目测试计划。

开发人员有责任为测试过程创建认同的输入。开发人员与测试分析人员之间的协作是非常重要的，理解测试观点的开发人员可以建立具有良好可测试性的软件，有助于降低测试代价。开发人员必须具有责任心，保证测试建立的客观性。

测试实施人员需熟悉被测试部件，可以是另一个开发者或独立测试者。测试实施人员按照测试计

划中的要求实施测试，处理测试过程中发生的问题，与测试分析人员协商，并建立相应的文档。

支持人员包括测试管理人员和信息维护人员。测试管理人员对测试过程中的相关信息进行处理，协调以上三类角色之间的关系，保证测试过程中文档标准的实施，保证所有活动结果的客观性。

3.3 测试实施和管理

测试实施和管理主要包括以下内容：

(1)实施准备。其主要任务是准备测试所需的活动模型、角色、资源和工具等。

(2)控制和管理。主要包括如下活动：

· 启动：测试实施人员根据给定的项目测试计划、测试实施模型和测试实施活动，实例化一个(组)测试并启动之。

· 监控：执行相应的测试，监督测试的执行，包括测试日历管理和特殊事件的处理(如应急情况处理)等，提供进展的报告，调查、分析和解决过程执行中发现的问题等。

· 分析：对测试实施活动进行评估和度量，通常当活动与事先安排不相符合时，需要生成问题和评价报告，从而可能需要调整测试计划。

· 审查及问题处理：在测试实施过程中分析和排除问题或不一致的情形，其目的是提供一种适时的、可信赖的文档手段。

在测试过程中，需要进行测试复用和数据维护，包括数据的准备、数据组织和存储、数据查询、测试层次结构及相关部件(如测试模型、测试用例集、测试结果及相关文档等)的维护、测试状态信息的管理等。测试过程的活动对于开发过程有着监督、审查和管理的作用。

4 测试实施

测试实施活动可以进一步细化为更小粒度的活动的集合。这些活动按照一定的程序、特定的时间和约束完成相应的任务，产生相应的结果。

这里使用 SN(Slang Net)[1]对测试实施活动进行可视化建模。SN 的定义如下：$SN=(P,T,A)$。其中，P 表示位置的集合，T 表示变迁的集合，A 表示流关系集合。SN 满足下面的性质：

$P\neq\varnothing, T\neq\varnothing, A\neq\varnothing$

$\forall t\in T, {}^{\bullet}t\neq\varnothing, t^{\bullet}\neq\varnothing$

图 2 给出了一个测试实施模型，对活动的约束(角色、资源和工具约束)在模型的外部表示。根据项目测试计划和企业内部的文档规范，制订对应于测试层次结构的部件级测试计划。然后，可以进行多次测试设计、测试执行和测试结果分析的循环。在完成测试，并满足测试覆盖标准后，生成测试报告，并结束该测试活动。测试实施主要包括如下活动：

(1)测试计划(部件级)。整个项目的测试计划由一组部件测试计划组成，可以利用测试层次结构来组织，从而形成部件测试计划的分级结构。测试计划的结果是部件测试计划，它详细规定相应的测试要求，包括测试的目的和内容、方法和步骤以及评价测试的标准等。

测试计划在需求分析后期启动，测试分析人员与开发人员等进行充分的讨论，形成部件测试计划的初始内容。

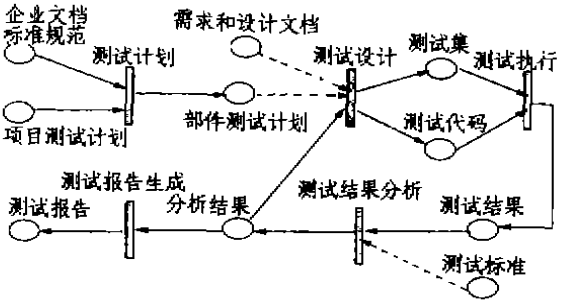
(2)测试设计。包括对被测对象进行测试建模和测试用例生成。根据不同的测试特征，可以建立多个测试模型。当测试过程中发现软件错误后，软件规约可能需要修改，从而需要修改行为(功能)模型；程序修改后，结构模型发生变化，相应地，导致其它修改(如对测试用例集的修改)。测试实施和控制活动维护所有的修改信息。

测试用例生成的目标是根据测试模型产生一个合适和可用的测试用例集和相关信息(称为测试集)。即使是小程序也可能需要大量测试用例，需要考虑是否具有可复用和可验证性，分配相应的资源。为了适应开发过程的变化，测试用例需要与开发过程同步地维护。测试用例生成与测试集的维护是紧密相关的，如果软件发生更改，则必须首先定位到需要修改的测试用例。

(3)测试执行。首先，测试启动根据当前需要执行的测试用例集，在测试数据库中获取相应的测试数据和测试脚本。在执行测试中，依次准备测试脚本、对应测试数据和预期结果，测试执行的结果加入到数据库中，如此循环以至规定的测试脚本执行完毕。

(4)测试结果分析。测试结果提供了故障分析的原始数据。当测试执行结束后，需要对测试所产生的结果进行分析，对于发现故障和没有发现故障两种情形分别处理。当执行一组用例发现故障后，故障分析确定故障发生的原因、故障的影响范围、故障对于系统运行所产生破坏性的程度等，把代码及文档提

交开发组进行修改。在修改完成后，分析修改所产生的影响，决定是否需要进行回归测试，相应的策略和采用的覆盖标准，并修改测试用例，生成新的测试用例。如果没有发现故障，则需要分析当前达到的测试覆盖是否满足客户对软件可靠性的需求。



其中的建模元素说明如下：

○	位置
□	变迁，表示活动
▣	变迁，包含外部可执行工具
⋯→	只读流关系，表示在触发变迁时，源位置中的标码不移动
——→	流关系

图 2　测试实施模型

(5)测试报告生成。测试分析报告说明对测试结果的分析、经过测试证实了软件具有的能力和欠缺限制，并给出结论性的评价意见，它既是对软件质量的评价(如测试有效性评价、测试覆盖评价、错误评价)，也是决定能否交付用户使用的依据。

参 考 文 献

1 Bandinelli S, Fuggetta A. Computational Reflection in Software Process Modeling, the SLANG Approach. In: Proc. of the 15th International Conf. on Software Engineering. 1993

2 Kit E. Software Testing in the Real World. Addison-Wesley Publishing Company, 1995

3 McGregor J D, Korson T D. Integrated Object-Oriented Testing and Development Processes. Communication of ACM, 1994, 37(9): 59～77

4 Perry W. Effective Methods for Software Testing John-Wiley & Sons, 1995

5 *Poston R M. T: Automated Software Testing Workshop Programming Environments.* Tinton Falls Inc., NJ., 1990

6 Poston R M. Automated Testing from Object Models. Communication of ACM, 1994, 37(9): 49～57

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

(上接第 56 页)

参 考 文 献

1 Malsburg C. Self-Organization of Orientation Sensitive Cells in the Striate Cortex. Kybernetik, 1973, 14: 85～100

2 Grossberg S. Adaptive Pattern Classification and Universal Recoding, I: Parallel Development and Coding of Neural Feature Detectors. Biological Cybernetics, 1976, 23: 121～134

3 Carpenter G A, Grossberg S. A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine. Computer Vision, Graphics, and Image Processing, 1987, 37: 54～115

4 Carpenter G A, Grossberg S. The ART of Adaptive Pattern Recognition by a Self-Organizing Neural Network. Computer, 1988, 21: 77～88

5 Carpenter G A, Grossberg S. ART2: Self-Organization of Stable Category Recognition Codes for Analog Input Patterns. Applied Optics, 1987, 26(23): 4919～4930

6 Carpenter G A, et al. ART2-A: An Adaptive Resonance Algorithm for Rapid Category Learning and Recognition. In: Proc. of the Intl. Joint Conf. on Neural Networks (IJCNN-91), Seattle, USA, IEEE/INNS Inc., 1991(2): 151～156

7 Carpenter G A, Grossberg S. ART3: Self-Organization of Distributed Pattern Recognition Codes in Neural Network Hierarchies. In: Proc. of the Intl. Neural Network Conf (INNC-90), Paris, France, Kluwer Academic Publishers, 1990(2): 801～804

8 Carpenter G A, et al. ARTMAP: A Self-Organizing Neural Network Architecture for Fast Supervised Learning and Pattern Recognition. Same to [6], 1991 (1): 863～868

9 Carpenter G A, et al. Fuzzy ART: An Adaptive Resonance Algorithm for Rapid, Stable Classification of Analog Patterns. Same to [6], 1991(2): 411～416

10 *Carpenter G A, et al. Fuzzy ARTMAP: A Neural Network Architecture for Incremental Supervised Learning of Analog Multidimensional Maps.* IEEE Trans. on Neural Networks, 1992, 3(5): 698～713

11 陈光乾，周戎等．一种新的自适应谐振算法 FTART. 软件学报，1996, 7(8): 458～465

12 陈兆乾，李红兵等．对 FTART 算法的研究及改进 软件学报，1997, 8(4): 259～265

13 Serrano-Gotarredona T, Linares-Barranco B. An ART1 Microchip and Its Use in Multi-ART1 Systems. IEEE Trans. on Neural Networks, 1997, 8(5): 1184～1194