21世纪大学计算机专业教材

UML

自动化测试技术

徐宏喆 陈建明 等

西安交通大学出版社
XI'AN JIAOTONG UNIVERSITY PRESS

UML 自动化测试技术

徐宏喆　陈建明

张昊翔　刘海军　夏蔚然

西安交通大学出版社

·西安·

内容提要

UML 自动化测试技术是一种新兴的软件测试方法，国内外多所大学和研究机构都在进行这方面的理论研究。本书是一本系统介绍使用 UML 模型实现软件自动化测试的计算机技术著作。书中对软件自动化测试的要求和 UML 模型的可测性进行了分析，从软件自动化测试和 UML 建模语言入手，引入 UML 自动化测试技术的原理，然后通过实例，分别从单元测试、集成测试、系统测试、Web 系统测试和实时系统测试五个方面说明自动化测试的方法，并且列出了一些自动化工具以及其实现方案，最后综述了该技术的发展趋势。

本书适合作为高等院校计算机、软件工程等专业本科高年级教材，亦可供从事软件质量保证和软件测试等相关技术研究的研究生、教师、技术人员参考。

图书在版编目(CIP)数据

UML 自动化测试技术/徐宏喆，陈建明等编著. —西安：西安交通大学出版社，2006.8

21 世纪大学计算机专业教材

ISBN 7-5605-1674-2

Ⅰ.U… Ⅱ.①徐… ②陈… Ⅲ.①软件—测试—高等学校—教材 ②面向对象语言，UML—程序设计—高等学校—教材 Ⅳ.TP31

中国版本图书馆 CIP 数据核字(2006)第 101401 号

书　　名：UML 自动化测试技术
编　　著：徐宏喆　陈建明等
出版发行：西安交通大学出版社
地　　址：西安市兴庆南路 25 号(邮编：710049)
网　　址：http://press.xjtu.edu.cn
电　　话：(029)82668357　82667874(发行部)
　　(029)82668315　82669096(总编办)
电子邮箱：eibooks@163.com
印　　刷：陕西江源印刷科技有限公司
版　　次：2006 年 8 月第 1 版　2006 年 8 月第 1 次印刷
开　　本：787 mm×1 092 mm　1/16
印　　张：18.125
字　　数：435 千字
印　　数：0001～3000
书　　号：ISBN 7-5605-1674-2/TP·324
定　　价：24.00 元

版权所有　侵权必究

前　言

软件工程研究的重要问题是软件的质量和可靠性。通过软件测试来保证软件的质量和可靠性是一个行之有效的方法。软件测试是一项繁琐而复杂的工作，又是高强度的脑力劳动，同时测试的费用占到整个开发费用的一半左右。传统的方法是由测试人员手工输入测试数据、记录测试结果并进行比较分析。这种方法不但无法保证测试的科学性与严密性，而且重复的测试工作需要耗费大量的时间、成本、人力和物力资源。此外，作为软件项目经理和软件开发人员都面临着进度不断加快、要求使用资源最小的挑战。为了实现这个目标，企业必然要转向自动化测试。

一些测试专家是如此评价测试自动化的：

已有相当多的测试专家令人信服地重复实施着软件测试过程自动化的工作。参加软件测试的大部分人员都认为测试过程的自动化不仅是我们期望的，而且实际上也是当前市场的必然要求。

测试自动化是一项投资。最初的投资可能很多，但投资的回报也很丰厚。自动化测试运行超过 15 次以后，测试就是免费的了。

目前，软件自动化测试技术发展非常迅速，成熟的自动化测试工具也层出不穷，包括 IBM 公司的 Rational Robot，Mercury Interactive 公司的 LoadRunner，Compuware 公司的 Test-Partner等。很多企业在测试工作中使用了自动化测试工具，在一定程度上提高了测试工作的效率，节省了测试工作的成本。但是，目前这些自动测试工具都是对测试执行、结果捕获与分析、结果验证和报告等方面的支持，对测试设计的支持还非常少。

测试设计可以通过两种方法实现：基于代码的测试设计与基于规格说明的测试设计。前一种方法是一种基于白盒测试的方法，通过代码导出测试用例实现自动化相对容易一些；后一种方法测试的层次较高，而且强调尽早测试尽早发现问题。因此，与基于代码的测试相比，提高基于规格说明的测试设计及执行的自动化具有更大意义。

要解决以上问题，首先必须对规格说明进行形式化，准确和严格的语义才能方便计算机的处理。目前对软件规格说明的形式化方法和工具有 Z，VDM，B，Prolog，Pert 网等。通过它们进行自动化测试也有很多的研究成果，但是这些方法在实际应用中碰到了困难，很难得到推广。

基于 UML 的自动化测试技术是一种新兴的软件测试方法。国内外多所大学和研究所都在进行这方面的理论研究，而且也得到了 IBM 等大公司的支持，一些基于 UML 的自动化测试工具也处在研发当中。可以看出，UML 自动化测试技术将成为软件测试的发展趋势和主流。但是，目前国内外除了一些文献对该技术进行讨论，还没有一本全面介绍这种技术的专著。于是，我们在收集、整理大量相关文献资料的基础上，结合我们在应用研究中获得的成果，撰写了这本介绍 UML 自动化测试的专著。

书中总结了很多研究者的科研成果，提出了很多利用 UML 实现测试自动化的系统化的

方法。虽然 UML 具有很多良好的特性，但是如何让它支持自动化测试，还需要进行如下研究工作：

（1）研究 UML 模型的可测试性，给出严格的测试模型，在适当的情况下对 UML 模型进行扩展。

（2）研究从可测试的 UML 模型中寻找测试序列和测试数据，实现测试设计和执行的自动化方法。

（3）研究如何设计和实现基于 UML 的自动化测试工具及如何实现该工具与其他工具的集成。

以上这些方面是本书所要讨论和解决的问题。本书从软件自动化测试和 UML 建模语言入手，引入 UML 自动化测试技术的理论，然后分别从单元测试、集成测试、系统测试、Web 系统测试、可靠性测试、回归测试 6 个方面来详细说明如何利用 UML 来进行自动化测试，最后，给出若干自动化工具以及实现自动化测试工具的方案。

由于篇幅有限，本书不可能涵盖软件测试及自动化的所有方面，很多方法和技术的讨论在其他相关书籍中可以找到。对本书的讨论范围做如下说明：

（1）本书重点讨论测试设计的自动化，对于测试执行、结果捕获与分析、结果验证等方面不作为讨论的重点。

（2）软件测试的范围广泛，包括功能测试、负载压力测试、安全测试、可靠性测试、易用性测试、文档测试等等。但是由于 UML 模型主要对软件功能进行建模，无法涉及其他方面，因此除了功能测试以外的其他测试将不在本书中讨论。

（3）基于 UML 的软件测试都是假设系统通过面向对象技术设计和开发的，但是本书并没有强调面向对象技术的细节和一些语言特性，以及对这些方面的测试，因此如何实现对继承、虚函数、抽象类、模板等测试将不在本书中讨论。

（4）本书介绍的自动化测试的方法对自动化测试工具的开发具有很大的指导意义。但是，如何利用相关的工具和技术实现自动化测试工具不是本书讨论的重点。

（5）公司引入自动测试是需要一定代价的。是否引入自动化测试，如何有效地实施自动测试，如何考虑自动化测试风险，如何计算自动化测试的成本……之类的问题本书将不做出回答。

本书按照循序渐进的方式进行编写，下面是章节的划分和主要内容：

第 1 章，软件测试及自动化。该章首先介绍了软件测试的定义，相关基本概念，测试的目的等。然后重点介绍了软件测试的方法。最后，说明了自动化测试的相关概念、分类以及自动化测试的过程。对软件测试和自动化测试比较了解的读者可以略过此章。

第 2 章，UML 建模语言。本书主要论述如何使用 UML 进行软件测试的技术，因此对 UML 语言进行详细讨论是必不可少的。本章包括三个部分：UML 简介、测试使用的模型以及对象约束语言。其中测试使用的模型主要包括用例图、顺序图、协作图、状态图和活动图。而为了提高模型的可测试性，需要通过对象约束语言（OCL）来辅助建立测试模型。

第 3 章，基于规格说明的测试和 UML 方法。基于代码和基于规格说明的测试是不同的，这一章中首先介绍了两者的差别。接下来详细介绍了基于规格说明的测试，而 UML 则是它的一个分支。然后，对基于 UML 的测试方法、测试阶段、测试过程和自动化框架进行说明。最后，说明了测试过程中面临的三个主要的问题。

第 4 章，基于 UML 的单元测试。单元测试是在单元级别对软件代码进行测试。本章首

先介绍了一些传统的测试方法。然后，说明面向对象的单元测试方法，包括类测试和使用状态图进行测试的方法。最后给出了一种单元测试自动化的解决方案。

第 5 章，基于 UML 的集成测试。完成单元测试后，就要进行集成测试。通过 UML 的协作图和状态图可以实现集成测试。本章详细讨论了基于协作图的测试方法和基于状态图的测试方法。

第 6 章，基于 UML 的系统测试。本章首先介绍了一般的系统测试方法，包括判定表法和场景测试方法，它们是 UML 测试方法的基础。接着，介绍了用例图和活动图相接合的测试方法。最后，详细讨论了 TOTEM 方法，该方法是一种适用性较强的系统测试方法，而且具有较高的自动化程度。

第 7 章，基于 UML 的 Web 应用系统测试。由于 Web 系统测试和系统测试还是有所不同的，所以本书单独通过一章来对 Web 测试的方法进行讨论。该章首先介绍了 Web 应用测试的方法，而后介绍了通过 UML 进行 Web 系统建模的方法，它也是 Web 系统测试的关键内容。接着讨论了基于 OOWTM 的测试方法，从对象模型、行为模型和结构模型对其进行详细说明。最后，讨论了 Webcomp 的测试方法。

第 8 章，基于 UML 的可靠性测试。本章首先介绍了软件可靠性测试的相关概念、测试的过程和测试的方法。然后介绍了基于使用模型的测试方法，这也是目前使用较多的测试方法。接下来，介绍了一种基于用例模型的测试方法。但是，这种方法的自动化程度不是很高，因此最后又提出了一种改进的方法，并对其进行了详细的论述。

第 9 章，回归测试和测试用例优选。虽然本章看似和 UML 关系不大，但是其中提到的测试优选方法和覆盖准则可以在其他章节中使用。本章首先对回归测试进行了介绍，然后介绍了基于 UML 的回归测试方法。接着，介绍了测试覆盖的准则和测试用例优选技术。最后，对基于风险的测试用例选择方法进行了简要的说明。

第 10 章，基于 UML 的自动化测试架构和工具。本章详细讨论了三种测试工具，包括 AGEDIS、PrUDE、TDE/UML，但是介绍的侧面有所不同。通过对它们的研究，可以详细了解自动化测试工具的架构、功能、用户接口、集成方法等。

本书可以作为软件质量保证和测试类的专著，可以作为研究生和本科生高年级的教材来使用。此外，对于一些从事软件测试工作的人员，本书也是一个不错的选择和参考。

徐宏喆和陈建明完成本书的策划，并组织编写了全书。刘海军完成了第 1 章、第 2 章和第 10 章的编写工作，夏蔚然完成了第 4 章和第 5 章的编写工作，张昊翔完成了第 3 章、第 6 章、第 7 章、第 8 章、第 9 章和第 10 章的编写工作。此外，王伟强、谢华伟、杨潇、翁绍榕、冯波、贺宏波、薛健、高鹤等也都参与很多相关工作。在此，我们对本书编写和出版作出努力的工作人员表示感谢。

书中包含了我们在软件测试行业的一些经验，其中一些观点和结论都是我们的一些研究成果和经验之说，是否正确还需要读者的批评。通过撰写这本书，希望将我们的研究成果和实际经验与广大的读者分享，共同促进该技术的研究发展和应用。由于成书仓促，书中难免有误，希望读者指正。

作者

2006 年 3 月于西安

目　录

前言

第 1 章　软件测试及自动化 …… (1)

1.1　软件测试 …… (1)

1.2　软件测试的目的 …… (3)

1.3　软件测试方法 …… (4)

1.3.1　白盒测试 …… (4)

1.3.2　黑盒测试 …… (5)

1.3.3　ALAC 测试 …… (5)

1.3.4　单元测试 …… (5)

1.3.5　集成测试 …… (6)

1.3.6　系统测试 …… (8)

1.4　自动化测试 …… (9)

1.4.1　自动化测试及分类 …… (9)

1.4.2　自动化测试过程 …… (10)

本章小结 …… (21)

习题 …… (21)

第 2 章　UML 建模语言 …… (22)

2.1　UML 简介 …… (22)

2.1.1　面向对象的开发方法 …… (22)

2.1.2　UML 的发展 …… (23)

2.1.3　UML 的含义 …… (24)

2.1.4　UML 组成 …… (25)

2.2　测试使用的模型 …… (27)

2.2.1　用例图 …… (27)

2.2.2　顺序图 …… (30)

2.2.3　协作图 …… (32)

2.2.4　状态图 …… (33)

2.2.5　活动图 …… (35)

2.3　对象约束语言 …… (37)

2.3.1　初识 OCL …… (37)

2.3.2　OCL 与 UML 模型元素 …… (38)

2.3.3　基本类型和值 …… (41)

2.3.4　对象及其属性 …… (44)

2.3.5　集合操作 …… (45)

2.3.6　消息 …… (47)

本章小结 …… (47)

习题 …… (47)

第 3 章　基于规格说明的测试和 UML 方法 …… (48)

3.1　基于代码与基于规格说明 …… (48)

3.2　基于规格说明的测试 …… (49)

3.2.1　方法分类 …… (49)

3.2.2　响应系统的测试 …… (50)

3.2.3　基于断言的方法 …… (50)

3.3　基于 UML 的测试方法 …… (55)

3.3.1　测试阶段与方法 …… (56)

3.3.2　测试过程 …… (58)

3.3.3　UML 自动化测试架构 …… (59)

3.4　基于 UML 测试中的若干问题 …… (60)

3.4.1　面向路径的测试数据自动生成 …… (60)

3.4.2　测试预言、期望结果的自动生成 …… (60)

3.4.3　回归测试和测试选择 …… (61)

本章小结 …… (61)

习题 …… (62)

第 4 章　基于 UML 的单元测试 …… (63)

4.1　单元测试概述 …… (63)

4.1.1　单元测试 …… (63)

4.1.2　极限编程中的单元测试 …… (64)

4.2　传统的单元测试 …… (64)

4.2.1　路径测试 …… (64)

4.2.2　条件测试路径选择 …… (69)

4.3　面向对象的单元测试 …… (71)

4.3.1　类测试模型 …… (71)

4.3.2　构建类测试用例 …… (71)

4.3.3　根据类图和 OCL 生成测试用例 …… (72)

4.3.4　根据状态转换图构建测试用例 …… (77)

4.4　往返路径测试方法 …… (79)

4.4.1　基于状态的测试方法概述 …… (79)

4.4.2　往返路径测试 …… (80)

4.4.3　测试生成例子 …… (81)

4.5　基于状态图的自动化单元测试 …… (84)

4.5.1　状态转换测试序列 …… (84)

4.5.2　调用序列树 …… (84)

4.5.3　构造调用序列树 …… (87)

4.5.4　获得测试约束条件 …… (90)

4.5.5 OrderSet 例子 …… (90)
本章小结 …… (93)
习题 …… (93)

第 5 章 基于 UML 的集成测试 …… (94)
5.1 集成测试概述 …… (94)
5.1.1 集成测试的目的和意义 …… (94)
5.1.2 集成测试的方案 …… (95)
5.2 基于 UML 协作图的集成测试 …… (98)
5.2.1 方法概述 …… (98)
5.2.2 UML 协作图回顾 …… (99)
5.2.3 基于协作图的协作集成测试模式 …… (103)
5.2.4 使用 UML 协作图生成测试用例 …… (104)
5.2.5 基于协作图的集成测试方法相关工作 …… (112)
5.3 基于 UML 状态图的集成测试 …… (113)
5.3.1 UML 建模组件 …… (113)
5.3.2 建立全局动作模型 …… (115)
5.3.3 测试的生成和执行 …… (131)
5.3.4 TnT 的执行 …… (134)
本章小结 …… (137)
习题 …… (137)

第 6 章 基于 UML 的系统测试 …… (138)
6.1 系统测试与方法 …… (138)
6.2 判定表法 …… (140)
6.2.1 建立判定表 …… (140)
6.2.2 判定表法举例 …… (141)
6.3 场景测试法 …… (143)
6.3.1 从用例模型生成测试用例 …… (143)
6.3.2 ATM 例子 …… (144)
6.4 基于用例图和活动图的测试方法 …… (149)
6.4.1 方法简介 …… (149)
6.4.2 在线书店例子 …… (149)
6.5 TOTEM 方法 …… (157)
6.5.1 概述 …… (157)
6.5.2 产生用例序列 …… (159)
6.5.3 确定用例场景 …… (166)
6.5.4 变量序列的产生 …… (172)
6.5.5 自动化的实现 …… (173)
本章小结 …… (177)
习题 …… (178)

第 7 章 基于 UML 的 Web 应用系统测试 …… (179)
7.1 Web 应用的测试 …… (179)
7.1.1 Web 应用的特点和故障源 …… (179)
7.1.2 Web 系统的测试内容 …… (180)
7.1.3 Web 应用的测试方法 …… (181)
7.2 Web 应用和 UML 建模 …… (181)
7.2.1 Web 应用的框架 …… (181)
7.2.2 Web 应用的组成 …… (182)
7.2.3 Web 应用的建模 …… (183)
7.3 基于 OOWTM 的测试方法 …… (189)
7.3.1 对象模型和测试方法 …… (189)
7.3.2 行为模型和测试方法 …… (190)
7.3.3 结构模型和测试方法 …… (194)
7.4 Webcomp 测试方法 …… (194)
7.4.1 Web Records 实例介绍 …… (194)
7.4.2 系统建模 …… (195)
7.4.3 基于优先权的场景测试 …… (198)
7.4.4 Webcomp 单元测试 …… (199)
本章小结 …… (200)
习题 …… (201)

第 8 章 基于 UML 的可靠性测试 …… (202)
8.1 软件的可靠性测试和方法 …… (202)
8.1.1 软件可靠性测试概念 …… (202)
8.1.2 软件可靠性测试过程 …… (203)
8.1.3 基于功能分解的软件可靠性测试方法 …… (204)
8.1.4 软件可靠性评估 …… (206)
8.2 基于使用模型的测试 …… (206)
8.2.1 使用模型和基于使用的测试 …… (206)
8.2.2 使用模型的创建过程 …… (207)
8.3 基于用例模型的测试方法 …… (213)
8.3.1 用例的精化 …… (213)
8.3.2 从用例到状态图 …… (215)
8.3.3 从状态图到使用图 …… (217)
8.3.4 从使用图到使用模型 …… (217)
8.3.5 从使用模型到测试用例 …… (218)
8.4 一种改进的测试方法 …… (219)
8.4.1 模型的定义 …… (219)
8.4.2 生成使用模型 …… (220)
8.4.3 测试举例 …… (222)
本章小结 …… (226)
习题 …… (226)

第 9 章 回归测试和测试用例优选 …… (227)
9.1 回归测试 …… (227)
9.1.1 回归测试的策略 …… (227)
9.1.2 回归测试的类型和一般步骤 …… (229)
9.1.3 选择性回归测试的 3 种标准 …… (230)

9.1.4 结构化软件回归测试技术 …… (230)
9.1.5 面向对象软件的回归测试 …… (231)
9.1.6 自动回归测试工具 …… (232)
9.2 跟踪性和测试选择 …… (233)
9.3 完全测试覆盖准则 …… (236)
9.3.1 白盒测试覆盖准则 …… (237)
9.3.2 基于 UML 的覆盖准则 …… (237)
9.4 测试用例优选技术 …… (239)
9.4.1 测试用例的优选问题 …… (239)
9.4.2 基于缺陷探测率的优选技术 …… (239)
9.4.3 基于风险的优选技术 …… (246)
本章小结 …… (248)
习题 …… (248)

第 10 章 基于 UML 的自动测试架构和工具 …… (249)
10.1 AGEDIS …… (249)
10.1.1 AGEDIS 的测试方法 …… (250)
10.1.2 AGEDIS 的体系结构 …… (250)
10.1.3 AGEDIS 的接口 …… (252)
10.1.4 AGEDIS 的工具集 …… (254)
10.1.5 AGEDIS 测试实例 …… (255)
10.2 PrUDE …… (257)
10.2.1 PrUDE 简介 …… (257)
10.2.2 PrUDE 平台的体系结构和自动化 … (258)
10.2.3 PrUDE 应用举例 …… (259)
10.3 TDE/UML …… (269)
10.3.1 TDE/UML 软件测试环境框架 …… (269)
10.3.2 TDE/UML 设计与执行 …… (272)
本章小结 …… (276)
习题 …… (276)

参考文献 …… (277)

第 1 章　软件测试及自动化

随着软件产业的日益壮大和逐步成熟，软件产业本身开始朝着软件分工的方向发展。在整个软件开发周期中，越来越多的厂商开始专注于不同的开发阶段。软件解决方案厂商凭借丰富的领域知识和软件设计经验，从事方案设计部分的工作；软件开发商以其自身的技术特点，从事软件研发工作，主要实现解决方案厂商所提供的面向某些领域的方案。近年来，用户对软件质量的要求越来越高，软件开发过程中对软件质量保证的重视程度明显提高，造就了软件测试这一分工的出现，并得到快速的发展。软件测试从最初的由软件编程人员兼职测试，发展到软件公司组建独立专职测试部门进行软件测试工作，最后出现了专门的软件测试公司。测试工作也从简单测试演变为包括编制测试计划、编写测试用例、准备测试数据、编写测试脚本、实施测试、测试评估等多项内容的正规测试。软件测试已经进入到较为规范化的阶段，对软件质量的提高有明显的帮助。

1.1　软件测试

自 20 世纪 70 年代以来，软件开发领域出现并且形成了软件生命周期的概念。软件开发过程开始划分成若干阶段进行。在不同阶段中，十分自然地具有阶段目标，对于程序正确性的要求也变得明确和严格。但程序中的错误并不一定是编码所引起的，很可能是详细设计阶段、概要设计阶段，甚至是需求分析阶段的问题所引起的。因此，必须对每一个阶段进行详细检查。这种贯穿于整个开发过程中各个阶段的复查、评估和检测活动，也就是所谓测试的活动。

目前关于“什么是软件测试”的问题，仍有很多不同的说法：

(1) 对照规格说明来检查程序。

(2) 使用各种可能方法尽力找出程序中的隐藏错误。

(3) 确认系统能够正常工作并供用户使用。

(4) 表明程序执行过程是正确无误的。

(5) 检验软件的能力。

(6) 验证软件文档。

(7) 确认开发任务已经完成。

这些说法虽然具有一定的道理，但都不能准确回答“什么是软件测试”这个问题。

1973 年，Hetzel 曾经提出，测试是对程序或者系统能否完成特定任务建立信心的过程。这种说法在开始一段时间内曾经被看作是最权威的。但后来也有人提出异议，认为不应该为一个程序建立信心或者显示信心而去做测试。于是，Hetzel 修正了自己的观点：测试的目的在于鉴定程序或者系统的属性或能力，它是软件质量的一种度量。这一说法在很大程度上依

赖了软件质量。对于很多人来说，软件质量的概念是比较抽象的，它和测试一样抽象，令人难以捉摸。尽管可以给出软件质量的确切解释，但影响软件质量的因素也并非是一定的。在某一环境下得到的高质量产品，换在另外的环境下也许就变为低质量产品了。如果将软件质量理解为“满足需求”，那么开发出来的产品如果能满足所有需求，就是高质量的软件产品。1983年，IEEE 提出的软件工程术语中给软件测试下的定义是：“使用人工或者自动手段来运行或测定某个系统的过程，其目的在于检验它是否满足规定的需求或是弄清预期结果与实际结果之间的差别。”这里强调了软件测试以检验是否满足需求为目标。

Myers 则持另外的观点，他认为：“程序测试是为了发现错误而执行程序的过程。”这一测试定义明确指出“寻找错误”是测试的目的。相对于“程序测试是证明程序中不存在错误的过程”，Myers 的定义似乎更合理。因为把证明程序无错当作测试目的是完全做不到的，而且对做好测试工作没有任何益处，甚至有害。不过，Myers 的定义所规定的范围似乎也有些狭窄，使得它受到较大的限制。如前所述，除去执行程序以外，还有许多方法去评价和检验一个软件系统。根据 Myers 的定义，测试只有在编码完成以后才能开始。另外，测试归根结底是“检测”“评价”和“测验”的意思，比“找错”的范围更广。

上述两种定义都是强调软件的正确性。有些测试专家认为软件测试的范围应该包括得更加广泛。Goodenough 认为测试除了要考虑正确性以外，还应关心程序的效率、健壮性等因素，并且应该为程序调试提供更多的信息，给出了如图 1－1 的测试组合表。

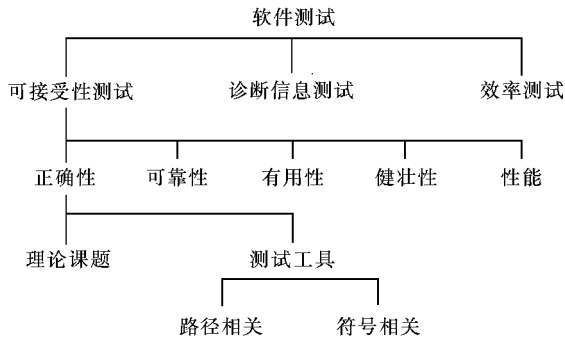


图 1－1　软件测试 Goodenough 定义表

Redwine 认为软件测试应该包含以下几种测试覆盖：

（1）功能覆盖。

（2）输入/输出域覆盖。

（3）函数交互覆盖。

（4）代码执行覆盖。

随后，软件测试专家们进一步总结原有的测试理论并加以完善，使得软件测试成为有理论指导的实践性学科。在软件测试理论迅速发展的同时，各种高级的软件测试方法也将软件测试技术提高到较高的高度。

近年来，尽管软件测试技术有了长足的进步，但总的来说，仍然和软件开发实践提出的要求有相当大的差距。测试手段的进展也远远没有达到人们想象的程度。

软件测试是十分困难的，任何开发过重要软件的人都有过这样的经历：用户报告代码中出

现隐错。当这样的情形发生时，软件开发管理者都会有同样的感受：这些隐错是如何逃过测试的呢？

事实上，软件有错的原因非常之多：

(1) 用户执行了测试所忽略的有错代码。因为开发时间的限制，完全有可能将未经过详细测试的代码发布，这是比较常见的。如果代码有错，用户当然可能碰到隐错。

(2) 用户应用了从未测试过的输入值组合。对测试人员来说可能的组合数太多而不可能全部应用。测试人员不得不决定选择哪些组合数来进行测试，有时就会作出错误的决定。成千上万的用户通过用户界面可以运行无数种不同的输入组合。不能期望有限的软件测试人员在软件发布之前检查所有的组合。这就需要测试人员对测试用例进行精心设计。

(3) 用户的操作环境从来没有经过测试。或者在测试环境下无法复制用户的硬件、外围设备、操作系统以及用户安装的应用软件。编写网络软件的公司不可能在测试实验室创建上千个网络节点，但是用户能够做出这样的环境。

这样，测试人员必须考虑软件的计算功能、输入及其组合方式以及软件规划和执行测试的操作环境，这个工作困难且费时。软件测试人员不仅需要一个好的程序员(测试时需要编写大量代码)，也需要具有较好的形式化语言、图论和算法的知识。实际上，具有创造力的测试员会把许多相关的计算原理结合到测试问题上。

1.2　软件测试的目的

软件测试的目的决定了如何去组织测试。如果测试的目的是为了尽可能多地找出错误，那么测试就应该直接针对软件比较复杂的部分或是以前出错比较多的位置。如果测试目的是为了给最终用户提供具有一定可信度的质量评价，那么测试就应该直接针对在实际应用中会经常用到的商业假设。不同的机构会有不同的测试目的，甚至相同的机构也可能有不同的测试目的，这是测试的不同区域或对同一区域的不同层次的测试所引起。

在谈到软件测试时，许多人都引用 Grenford J. Myers 在 *The Art of Software Testing* 一书中的观点：

(1) 软件测试是为了发现错误而执行程序的过程。

(2) 测试是为了证明程序有错，而不是证明程序无错。

(3) 一个好的测试用例在于它能发现至今未发现的错误。

(4) 一个成功的测试是发现了至今未发现的错误的测试。

这种观点可以提醒人们测试要以查找错误为中心，而不是为了演示软件的正确功能。但是仅凭字面意思理解这一观点可能会产生误导，认为发现错误是软件测试的唯一目的，查找不出错误的测试是没有价值的，事实并非如此。

首先，测试并不仅仅是为了要找出错误。通过分析错误产生的原因和错误的分布特征，可以帮助项目管理者发现当前所采用的软件过程的缺陷，以便改进。同时，这种分析也能帮助我们设计出有针对性地检测方法，改善测试的有效性。

其次，没有发现错误的测试也是有价值的，完整的测试是评定测试质量的一种方法。详细而严谨的可靠性增长模型可以证明这一点。例如 Bev Littlewood 发现一个经过测试而正常运行了 n 小时的系统有继续正常运行 n 小时的概率。

人们出于不同的动机去进行软件测试,其中绝大多数动机都可以用一个名称来描述。想要通过测试确定所在机构是否应该接受某个产品,这种测试称为接受测试;想要通过测试确定某个产品是否满足实现标准,这种测试称为符合性测试;想要通过测试确定某个产品是否易于使用,这种测试称为可用性测试。按照这样划分,还包括性能测试、可靠性测试、健壮性测试等等。

不难看出,诸多种类的软件测试具有一些共同特性:

(1) 每种测试都需要测试人员按照产品行为描述来实施。产品行为描述可以是书面的规格说明书、产品的需求说明书、产品文件或者用户手册,甚至可以是源代码。显然,最好尽可能地给出有关产品的准确信息。然而,在有些情形下,能得到的仅仅是可执行程序和指导产品如何运行的一般性建议。

(2) 每种测试都需要产品运行于真实或者模拟环境下。运行产品功能把测试与代码评审和审查区分开来,代码评审和审查可以在产品编译和链接之前静态实施。

(3) 每种测试都要求以系统方法展示产品的功能性,说明测试结果是肯定的(测试成功)还是否定的(测试失效),以及是否可判断其中的区别。真正区分失效测试和成功测试的关键在于:必须知道在寻找什么,并能说出何时能找到。

每种测试都包括上述特征,主要差别在于其目的和一些细节处理上。软件测试是一项细致并需要高度技巧的工作,稍有不慎就会顾此失彼,发生不应有的疏漏。软件工程的总目标是充分利用有限的人力和物力资源,高效率、高质量地完成测试。为了降低测试成本,选择测试用例时应注意遵守“经济性”的原则。

1.3 软件测试方法

软件测试的方法和技术是多种多样的。对于软件测试技术,根据不同角度,可以将测试方法分为不同种类。从是否需要执行被测软件的角度,可以分为静态测试和动态测试;从测试是否针对系统内部结构和具体实现算法的角度,可以将软件分为白盒测试和黑盒测试;从实际测试的前后过程来看,软件测试是由一系列的不同测试组成,这些软件测试的步骤分为单元测试、组装测试(集成测试)、确认测试和系统测试。下面我们简单介绍主要的测试方法。

1.3.1 白盒测试

白盒测试又称为结构测试或逻辑驱动测试,允许测试人员对程序内部逻辑结构及有关信息来设计和选择测试用例,对程序的逻辑路径进行测试。白盒测试是知道产品内部工作过程,可通过测试来检测产品内部动作是否按照规格说明书的规定正常进行,按照程序内部的结构测试程序,检验程序中的每条通路是否都能按预定要求正确工作,而不顾它的功能,白盒测试的主要方法有逻辑驱动、基路测试等,主要用于软件验证。在使用白盒测试时,测试者必须检查程序的内部结构,从检查程序的逻辑着手,得出测试数据。贯穿程序的独立路径数是天文数字。但即使每条路径都测试了,仍然可能有错误。首先,穷举路径测试决不能查出程序违反了设计规范,即程序本身是个错误的程序;其次,穷举路径测试不可能查出程序中因遗漏路径而出错;最后,穷举路径测试可能发现不了一些与数据相关的错误。

1.3.2　黑盒测试

黑盒测试又称为功能测试或数据驱动测试，把系统看成一个黑盒子，不考虑程序的内在逻辑，只根据需求规格说明书的要求来检查程序的功能是否符合它的功能说明。黑盒测试是已知产品所应具有的全部功能的情形下，通过测试来检测每个功能是否都能正常使用，在测试时，把程序看作一个不能打开的黑盒子，完全不考虑程序内部结构和内部特性，测试者对程序接口进行测试。黑盒测试只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出信息，并且保持外部信息（如数据库或文件）的完整性。黑盒测试方法主要有等价类划分、边值分析、因果图、错误推测等，主要用于软件确认测试。黑盒法着眼于程序外部结构、针对软件界面和软件功能进行测试。黑盒法是穷举输入测试，只有把所有可能的输入都作为测试情况使用，才能以这种方法查出程序中所有的错误。实际上测试情况有无穷多个，人们不仅要测试所有合法的输入，而且还要对那些不合法但是可能的输入进行测试。

1.3.3　ALAC 测试

ALAC(Act Like A Customer)测试是一种基于客户使用产品的知识开发出来的测试方法。一个复杂的软件系统可能会存在很多的缺陷，这些缺陷并非都是用户可能遇到的。ALAC 测试是站在用户的角度，将查找错误的范围缩小为用户最有可能遇到的错误范围之内，使得用户成为最大的受益者。

1.3.4　单元测试

单元测试是在软件开发过程中要进行的最低级别的测试活动，测试的对象是软件设计的最小单位——模块。单元测试的依据是详细设计，单元测试应对模块内所有重要的控制路径设计测试用例，以便发现模块内部的错误。在单元测试活动中，软件的独立单元将在与程序的其他部分相隔离的情况下进行测试。单元测试语言采用白盒测试技术，系统内多个模块可以并行地进行测试。

在一种传统的结构化编程语言中，比如 C 语言，要进行测试的单元一般是函数或子过程。在像 C++这样的面向对象语言中，要进行测试的基本单元是类。对 Ada 语言来说，开发人员可以选择是在独立的过程和函数，还是在 Ada 包的级别上进行单元测试。单元测试的原则同样被扩展到第四代语言(4GL)的开发中，在这里基本单元被典型地划分为一个菜单或显示界面。

单元测试不仅仅是作为无错编码的一种辅助手段在一次性的开发过程中使用，单元测试必须是可重复的，无论是在软件修改，或是移植到新的运行环境的过程中。因此，所有的测试都必须在整个软件系统的生命周期中进行维护。

经常与单元测试联系起来的另外一些开发活动包括代码走读(code review)，静态分析(static analysis)和动态分析(dynamic analysis)。静态分析就是对软件的源代码进行研读，查找错误或收集一些度量数据，并不需要对代码进行编译和执行。动态分析就是通过观察软件运行时的动作，来提供执行跟踪、时间分析以及测试覆盖度方面的信息。

一般认为单元测试应紧接在编码后，当源程序编制完成并通过复审和编译检查，便可开始

单元测试。测试用例的设计应与复审工作相结合,根据设计信息选取测试数据,这将增大发现各类错误的可能性。在确定测试用例的同时,应给出期望结果。

测试模块应该开发一个驱动模块(driver)和(或)若干个桩模块(stub),图 1-2 显示了一般单元测试的环境。驱动模块接收测试数据并将这些数据传递到被测试模块,被测试模块被调用后,测试模块打印结果信息。驱动模块和桩模块是测试使用的软件,而不是软件产品的组成部分,但它们需要一定的开发费用。若驱动和桩模块比较简单,实际开销相对低些。遗憾的是,仅用简单的驱动模块和桩模块不能完成某些模块的测试任务,这些模块的单元测试只能采用后面讨论的综合测试方法。

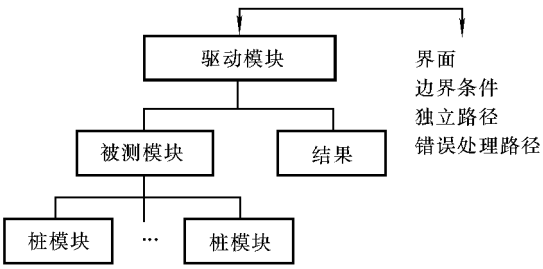


图 1-2 单元测试环境

1.3.5 集成测试

经过单元测试,一般来说每个模块都可以单独工作。但是,一旦将所有模块都集成起来形成系统后,系统却不一定能够很好地正常工作。主要原因是,模块相互调用时接口处会引入许多新问题。例如,数据经过接口可能丢失;一个模块对另一模块可能造成不应有的影响;几个子功能组合起来不能实现主功能;误差不断积累达到不可接受的程度;全局数据结构出现错误,等等。集成测试是组装软件的系统测试技术,按设计要求把通过单元测试的各个模块组装在一起之后,进行集成测试以便发现与接口有关的各种错误。

一些设计人员习惯于把所有模块按设计要求一次性全部组装起来,然后进行整体测试,这称为非增量式集成。这种方法容易出现混乱。因为测试时可能发现一大堆错误,为每个错误定位和纠正非常困难,并且在改正一个错误的同时又可能引入新的错误,新旧错误混杂,更难判断出错的原因和位置。与之相反的是增量式集成方法,程序一段一段地扩展,测试的范围一步一步地增大,错误易于定位和纠正,界面的测试亦可做到完全彻底。下面讨论两种增量式集成方法。

1. 自顶向下集成

自顶向下集成是构造程序结构的一种增量式方式,它从主控模块开始,按照软件的控制层次结构,以深度优先或广度优先的策略,逐步把各个模块集成在一起。深度优先策略首先是把主控制路径上的模块集成在一起,至于选择哪一条路径作为主控制路径,这多少带有随意性,一般根据问题的特性确定。以图 1-3 为例,若选择了最左一条路径,首先将模块 M1,M2,M3 和 M4 集成在一起,再将 M5 集成起来,然后考虑中间和右边的路径。广度优先策略则不然,它沿控制层次结构水平地向下移动。仍以图 1-3 为例,它首先把 M2,M6 和 M8 与主控模块

集成在一起，再将 M3，M5 和其他模块集成起来。

自顶向下综合测试的具体步骤为：

（1）主控模块作为测试驱动模块，把对主控模块进行单元测试时引入的所有桩模块用实际模块替代。

（2）依据所选的集成策略（深度优先或广度优先），每次只替代一个桩模块。

（3）每集成一个模块立即测试一遍。

（4）只有每组测试完成后，才着手替换下一个桩模块。

（5）为避免引入新错误，必须不断地进行回归测试（即全部或部分地重复已做过的测试）。

自顶向下集成的优点在于能尽早地对程序的主要控制和决策机制进行检验，因此较早地发现错误。缺点是在测试较高层模块时，低层处理采用桩模块替代，不能反映真实情况，重要数据不能及时回送到上层模块，因此测试并不充分。

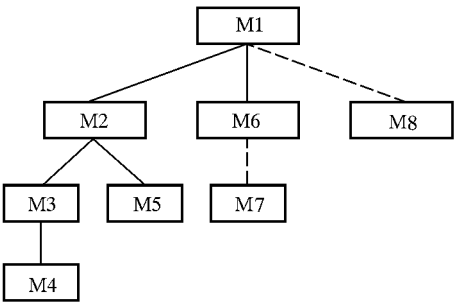


图 1－3　测试模块图（1）

2. 自底向上集成

自底向上测试是从“原子”模块（即软件结构最低层的模块）开始组装测试，因测试到较高层模块时，所需的下层模块功能均已具备，所以不再需要桩模块。

自底向上综合测试的步骤为：

（1）把低层模块组织成实现某个子功能的模块群（cluster）。

（2）开发一个测试驱动模块，控制测试数据的输入和测试结果的输出。

（3）对每个模块群进行测试。

（4）删除测试使用的驱动模块，用较高层模块把模块群组织成为完成更大功能的新模块群。

从第一步开始循环执行上述各步骤，直至整个系统构造完毕。

图 1－4 说明了上述过程。首先“原子”模块被分为 3 个模块群，每个模块群引入一个驱动模块进行测试。因模块群 1、模块群 2 中的模块均隶属于模块 Ma，因此在驱动模块 D1，D2 去掉后，模块群 1 与模块群 2 直接与 Ma 接口，D3 被去掉后，Mb 与模块群 3 直接接口，可先对 Ma，Mb 进行集成测试，最后 Ma，Mb 和 Mc 全部集成在一起进行测试。

自底向上集成方法不用桩模块，测试用例的设计亦相对简单，但缺点是程序最后一个模块加入时才具有整体形象。它与自顶向下综合测试方法优缺点正好相反。因此，在测试软件系统时，应根据软件的特点和工程的进度，选用适当的测试策略，有时混合使用两种策略更为有效，上层模块用自顶向下的方法，下层模块用自底向上的方法。

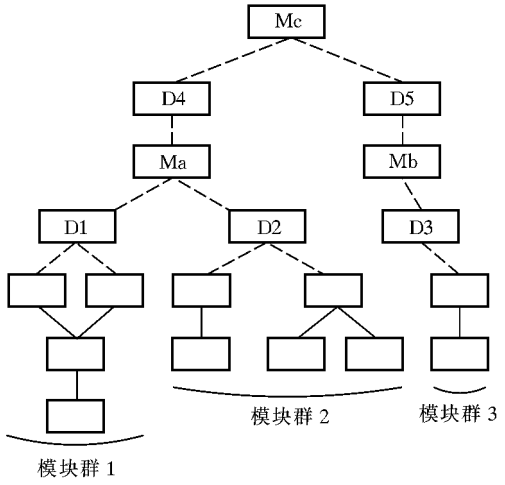


图 1-4 测试模块图(2)

1.3.6 系统测试

系统测试的目的是充分运行系统,验证系统各部件是否都能正常工作并完成所赋予的任务。以下介绍常见的几类系统测试。

1. 恢复测试

恢复测试主要检查系统的容错能力。当系统出错时,能否在指定时间间隔内修正错误并重新启动系统。恢复测试首先要采用各种办法强迫系统失败,然后验证系统是否能尽快恢复。对于自动恢复需验证重新初始化(re-initialization)、检查点(check pointing mechanisms)、数据恢复(data recovery)和重新启动 (restart)等机制的正确性;对于人工干预的恢复系统,还需估测平均修复时间,确定其是否在可接受的范围内。

2. 安全测试

安全测试主要检查系统对非法侵入的防范能力。安全测试期间,测试人员假扮非法入侵者,采用各种办法试图突破防线。例如:①想方设法截取或破译口令;②定做专门软件来破坏系统的保护机制;③故意导致系统失败,企图趁恢复之机非法进入;④试图通过浏览非保密数据,推导所需信息等等。理论上讲,只要有足够的时间和资源,没有不可进入的系统。因此系统安全设计的准则是,使非法侵入的代价超过被保护信息的价值,此时非法侵入者已无利可图。

3. 强度测试

强度测试主要检查程序对异常情况的抵抗能力。强度测试总是迫使系统在异常的资源配置下运行。例如:①当中断的正常频率为每秒 1～2 个时,运行每秒产生 10 个中断的测试用例;②定量地增长数据输入率,检查输入子功能的反应能力;③运行需要最大存储空间(或其他资源)的测试用例;④运行可能导致操作系统崩溃或磁盘数据剧烈抖动的测试用例。

4. 性能测试

对那些实时系统和嵌入式系统,软件部分即使满足功能要求,也未必能够满足性能要求。