模型驱动的自动化测试架构

周景才,杨家红,陈毅波
ZHOU Jing-cai,YANG Jia-hong,CHEN Yi-bo

湖南师范大学 工学院 电子工程系,长沙 410081
Department of Electronic Engineering,College of Polytechnic,Hunan Normal University,Changsha 410081,China
E-mail:jhyang3668@vip.sina.com

ZHOU Jing-cai,YANG Jia-hong,CHEN Yi-bo.Testing architecture based on model driven.Computer Engineering and Applications,2010,46(2):66-68.

Abstract: Today,many software company want to cut down the development cycle but it usually makes the software has a poor quality.So,how to solve this problem has became very important.In this paper,an automation testing architecture based on model driven is recommended according to MDT.By using a serials algorithms,the architecture transforms the UML system design model into U2TP testing design model directly.Then,the U2TP testing model can be used to generate test case automatically based on some testing strategy and software testing engineering method.Experiment results demonstrate that the architecture make it became possible to reuse testing resource and design testing case in the early of a project.So,it's fairly efficient for solving "how to cut down the development cycle" problem.

Key words: Unified Modeling Language(UML);Model Driven Testing(MDT);system under testing;test case;test logic;system design model;testing design model

摘 要:如何在确保软件质量的前提下有效缩短上市周期的问题日益显得重要。在实际工作中基于MDT的思路研究出了一种基于模型驱动的自动化测试架构。该架构通过算法直接将UML系统设计模型转换成U2TP测试设计模型,然后由测试设计模型根据测试策略和测试工程方法自动生成测试用例,实现了测试资源重利用和测试活动的前移从而有效缩短了测试周期。

关键词:统一建模语言;基于模型驱动测试;被测系统;测试用例;测试逻辑;系统设计模型;测试设计模型
DOI:10.3778/j.issn.1002-8331.2010.02.021 **文章编号**:1002-8331(2010)02-0066-03 **文献标识码**:A **中图分类号**:TP311.56

1 引言

软件测试在软件开发中占有非常突出的重要位置。然而为缩短产品的开发周期加快产品的上市时间,各商家又不得不从测试环节中节省时间。所以如何在确保软件质量的前提下缩短测试周期便成了各测试领域专家研究的重点。

受MDT思想的启法,从实现测试资源重利用及测试前移的目标出发,提出了一种新的自动化测试架构。该架构的主要思想就是通过测试模型来驱动测试,将原来的测试用例设计与维护工作转移到测试模型的设计与维护,并通过模型转换、用例自动生成、测试数据自动生成等技术实现测试活动前移和测试资源的重利用。支撑工具TestStudio1.0的试点反馈证实,该架构对缩短开发周期有明显的改进作用。

2 概述

2.1 模型驱动测试概述

随首UML 2.0 Testing Profile的发布,模型驱动测试作为一种全新的测试思想已由前期的理论探索论证转入了发展与实践阶段。模型驱动测试就是通过对SUT的功能与系统结构进行分析,然后结合测试策略构建起全面、清晰的测试模型,最后通过测试模型自动生成测试用例驱动测试人员完成SUT的测试。模型驱动测试的优点主要有两方面:

(1)测试模型为用户提供了更加清晰、准确和系统的测试设计[1]。无论是基于数据驱动的还是基于关键字驱动的自动化测试技术展现给用户的都是一个个互不相关的测试用例[2],用户很难从生成的测试用例中理解整个测试系统的结构与测试重点。

(2)减少了测试用例维护工作,实现了测试资源的重利用,有效缩短了测试周期。建立起测试模型后,测试系统设计师可以在测试需求或SUT的系统设计规格发生改变后通过调整测试模型(而不是个性测试用例)来适应变化。

2.2 目前软件测试主要存在的问题

在目前的自动化测试活动中,一般都是由TSE(测试系统工程师)首先根据测试需求设计出测试方案,然后根据测试方案设计测试用例,最后由测试工程师根据测试用例实现自动化

基金项目:湖南省自然科学基金(the Natural Science Foundation of Hunan Province of China under Grant No.08JJ3131)。
作者简介:周景才(1982-),男,硕士生,主要研究领域:软件测试工程方法、自动化测试技术;杨家红(1968-),通迅作者,男,工学博士,教授,主要研究领域:网络系统安全、模式识别与图像处理。
收稿日期:2008-10-10 **修回日期**:2008-11-18

测试脚本并执行测试。在整个的测试活动中,如果测试需求发生变化或者被测试系统的实现规格发生更改都不可避免地引起整个测试项目组成员从上至下的修改。因此如何快速响应变化实现测试资源重利用就成了目前测试工程方法中迫切需要解决的问题。

另外,在目前的测试活动中,用例维护也是消耗时间的一个重要环节。无论是基于数据驱动的测试技术还是基于关键字驱动的自动化测试技术,都面临着一个棘手的问题,那就是如何有效地维护测试用例。因为基于上述技术的测试用例都不是根据测试设计自动生成的,当发生变化时需要手工地修改与维护。

3 基于模型驱动的自动化测试架构

3.1 测试架构

MDT-TA(Model Drive Testing-Testing Architecture)的系统结构如图 1 所示,该系统结构将所有模块分成三层。

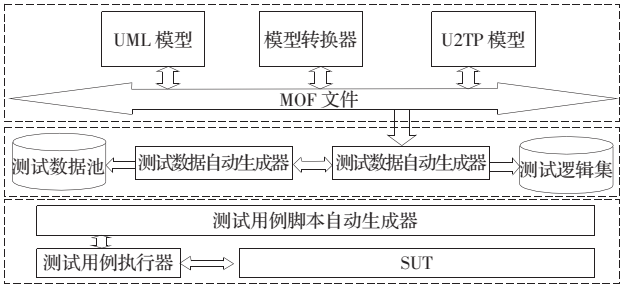

图 1 MDT-TA 系统架构图

(1)模型处理层——核心层

该层是整个测试架构中的核心层。本层提供可视化的建模工具来支持系统设计建模和测试设计建模。当系统设计模型设计完成后,“模型转换器”根据模型转换算法自动将系统设计模型转换成测试设计模型。

(2)测试用例抽象层

为了增加该测试架构的可扩展性,在测试架构中提炼出了测试用例抽象层。该层根据已建立的测试模型和测试策略,自动生成测试逻辑和测试数据池,而不直接生成具体的测试用例脚本。具体的实现与执行将由测试人员根据系统的要求在第三层的用例实现与执行层进行适配。

(3)测试用例实现与执行层

有了测试逻辑和测试数据后,系统就可以根据 SUT 的特性选用一种最合适的测试语言生成具体的测试脚本实施测试。该层的功能就是用测试数据实例化测试逻辑从而形成测试用例,然后生成测试脚本。

3.2 模型自动转换技术

在文献[3]中描述“软件的测试内容与方法由被测系统决定”,文献[4]中则称“测试模型与系统设计模型之间有一种天然的联系”,由此可见测试设计模型与系统设计模型之间存在着一种强关联。该文的模型转换技术就是通过解析系统设计模型与测试设计模型之间的关联关系来实现从 UML 模型到 U2TP 模型的自动转换[5]。图 2 展示了设计模型与测试活动的对应关系[6]。

在 UML 建模语言中,用例是来描述业务功能的,它的执行有明确的前因和后果,一般由几个有序的步骤来完成[7]。由用例组合而成的用例图明确地划定了系统所能提供的服务,而系统

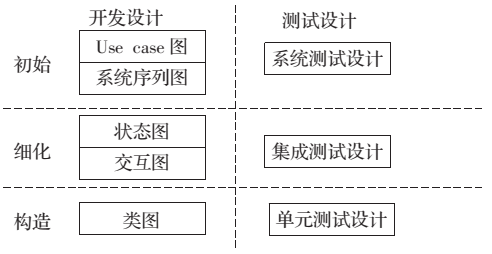

图 2 系统设计模型与测试活动对应关系图

序列图则清晰地描述出了每一个功能的详细操作步骤和激励。该文的模型转换技术就是通过分析用例图和系统序列图来构建系统测试模型,构建过程如图 3 所示。

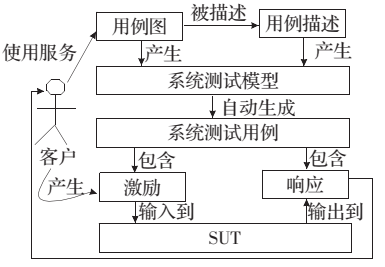

图 3 基于用例的系统测试构造模型图

生成测试系统模型的过程如下:

(1)根据用例图分析出所有的参与者。

(2)根据用例描述图分析出参与者的活动和系统活动。

(3)根据参与者活动和系统活动的基本流程和备选流程生成系统测试模型。

下面通过 ATM 系统的实现例来演示模型转换技术。图 4 和表 1 分别为 ATM 机系统的用例图和用例描述表。

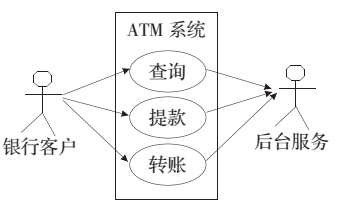

图 4 ATM 系统用例图

表 1 用例描述

过程描述	步骤	参与者活动	系统活动
基本流程	1	用户插入信用卡	读取信用卡号,校验信用卡合法
	2	输入密码	验证密码
	3	查询	从数据库服务器中查询当前账户的余额,然后显示
	4	退出系统,取回信用卡	注销登录,退出信用卡
备选流程	1(a)	用户插入无效信用卡	系统显示错误并退出信用卡,用例结束
	2(a)	输入错误密码	系统显示错误并提示用户重新输入密码,重新回到基本流步骤 2;如果连续三次输入错误,用例结束

模型转换器通过扫描分析系统设计模型的 MOF 文件,得到当前系统参与者的活动和系统活动。然后将系统活动集转换成测试模型中的状态图,同时将参与者的活动转换成激励,每一状态迁移时输出的信息作为测试结果仲裁的输入。基于用例图自动生成的 ATM 系统测试模型如图 5 所示。在生成的测试模型中存在两个激励和三个观察点,通过对激励点的不同组合就可以完成对 ATM 系统查询功能的系统测试。

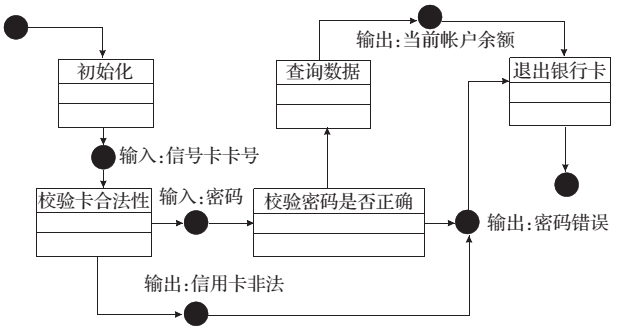

图 5 ATM 系统的系统测试模型图

3.3 测试逻辑自动生成技术

为了实现与平台无关和提高系统的可扩展性,提出了“测试逻辑”的概念,其定义如下:

定义 1 测试逻辑

(1)有特定的测试目的且该目的在当前测试套中是唯一的。

(2)不含测试数据。

(3)是一系列的有序的测试步骤集合。

从测试模型生成测试逻辑的过程中可以根据测试策略选用不同的算法来实现。结合深度优先算法和广度优先算法,从测试模型状态图的初始状态开始遍历,将遍历过程中产生的每一条完整的状态转换路径都记录下来,这样每一条完整的路径就是一个测试逻辑。

算法描述如下:

步骤 1 检查出当前状态图的所有环回路径并在相应的节点做上标记。

步骤 2 采用深度优先算法打上一级步骤编号。

步骤 3 采用广度优先算法打上二级步骤编号,见图 6。

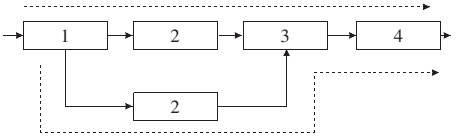

图 6 深度优先搜索算法打上二级步骤编号

步骤 4 根据编号生成测试逻辑。

由图 7 可以得到以下测试逻辑:TestLogic(1.1,2.1,3.1,4.1);TestLogic2(1.1,2.2,3.1,4.1)。

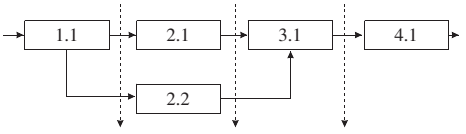

图 7 广度优先搜索编号示意图

3.4 测试数据自动生成技术

测试数据的好坏直接决定了测试用例的质量。测试数据如果过多,则会加重对测试资源的消耗(包括人力资源);如果过少,又有可能导致覆盖不全面,容易造成漏测。该文的测试数据自动生成技术是针对系统测试用例而提出的,所以它只关注激励数据而不关注程序实现本身。

算法描述如下:

步骤 1 获取测试模型中的所有激励(即标识为“IN”的变量)与约束,见表 2。

步骤 2 根据激励的合法性约束划分出有效等价类和无效等价类的边界,见表 3。

表 2 “密码”约束规则

(1)密码不能为空
(2)密码为 6 位数字
(3)密码不能以 0 开头

表 3 等价类划分边界条件表

	有效等价类	等价类编号	无效等价类	等价类编号
密码	非空	1	为空	4
	数字	2	非数字	5
	6 位数字	3	小于 6 位数字	6
			大于 6 位数字	7

步骤 3(可选) 如果激励的约束中确定了值的范围、个数或者先后顺序,则标识该激励的边界值。

表 4 激励边界值表

	上点	离点	内点
密码长度	6	7	NULL

步骤 4 根据测试数据自动生成算法,生成有效等价类解集和无效等价类解集。在生成的测试数据中,根据测试策略中的权值表(该表从 XML 模板文件中读取,表中的权值因测试的目的不同而判定每一级的条件不同。例如在性能测试策略中,靠近阀值或极限值的数据的优先级别为最高;而在基本功能测试中,为上点、离点或内点的数据的优先级要高于阀值)设定每一个测试数据的优先级。

表 5 等价类分类表

有效解集			覆盖等价	无效解集			覆盖等价
编号	值	优先级	类编号	编号	值	优先级	类编号
1	123456	3	1,2,3	4		1	4
2	111111	3	1,2,3	5	Aa2222	1	5
				6	333	1	6
				7	33333333	1	7

4 结论与展望

目前基于 MDT-TA 指导开发的全流程测试管理与执行工具 TestStudio 已进入用户体验阶段。从用户反馈的结果看,一方面证实了该架构在实现测试前移缩,短开发周期和降低维护成本的优势;另一方面也暴露出来了该架构的不足之处:

(1)只能证实,不能证伪。如果系统设计模型错误,则测试模型无法自动检测出系统设计模型的错误。

(2)当前的 UML 语义还不足已完全支撑测试模型中需要的信息[8]。

(3)生成的 U2TP 测试模型可以非常方便地生成 TTCN 测试用例,但对 JAVA、C++语言的支持不够。U2TP 模型中的部分概念无法直接转换成对应的 JAVA 或 C++代码。

因此,下一步需要研究的重点主要是下面几个方向:

(1)研究测试设计模型的自动测试技术。

(2)扩展 UML 和 U2TP 语义,使其满足测试建模的需要。

(3)研究将状态图、类图生成测试模型的算法,使当前的测试模型不仅仅覆盖系统测试而且还可以覆盖集成测试和单元测试。

(4)扩展测试逻辑和测试数据自动生成算法。

(下转 131 页)