基于模型的软件测试方法研究

Research on The Software Testing Based on Testing Model

吴 艳 张 惠 （浙江工业大学 之江学院 浙江 杭州 310024）

摘 要： 随着面向对象软件开发技术的广泛应用和软件测试自动化的要求，基于模型的软件测试逐渐得到了软件开发人员和软件测试人员的认可和接受。基于模型的软件测试是软件编码阶段的主要测试方法之一，具有测试效率高、排除逻辑复杂故障测试效果好等特点。但是误报、漏报和故障机理有待进一步研究。对主要的测试模型进行了分析和分类，同时，对故障密度等参数进行了初步的分析；最后，提出了一种基于模型的软件测试流程。

关键词： 软件测试 软件模型 基于模型的测试 静态分析

1 引言

基于模型的软件测试技术是针对软件中的一些常见的软件模型而提出的一种测试技术，如故障模型、安全模型和死锁模型等等。与形式验证排除故障不同，基于模型的软件测试技术首先提出软件模型，然后通过检测算法进行检测，如果检测算法是完全的，则能够从软件中排除该类模型[1]。

近年来，基于模型的软件测试技术得到快速发展，大量的软件测试工具被研制出来并可以自动检索软件中的故障，在对一些大型商业软件和开源软件的测试中发现了大量的以前测试没有发现的软件故障和安全隐患。例如，SDV 在对 Windows 操作系统 126 个使用多年的驱动程序测试中发现了 65 个故障，包括 12 个严重故障[2]；EXPLODE 在一些常用的文件存储系统发现了大量的严重故障[3]；MC 在 Linux 和 OpenBSD 软件中发现了近 500 个故障[4]以及 100 多个安全漏洞[5]。FindBugs 在 Eclipse、J2SE 和 Jboss 等开源软件中发现上百个的故障[6]。

我国有不少高校和研究所从事软件测试及相关理论方面的研究工作，但是，尚未见国产的基于模型的商业化软件测试工具，目前国内测试中心使用的相关测试工具都是从国外购置的。近年来，和国外合作利用基于模型的静态分析测试工具对 IBM、HP、BOEING 等知名企业已经长期运行的数百万行商业代码进行测试和分析，发现了数百个的故障。同时，在军队预先研究等项目资助下自主开发了 C/C + + 的故障查找工具 TGSOFM，对国内 7 个软件约 32 万行、2 类故障进行测试，发现了约 300 个故障[7]。

2 基于模型的软件测试技术

2.1 技术特点

与其他测试技术相比，基于模型的软件测试技术具有以下特点：

（1）根据被测试应用程序的分析设计模型及其生成测试模型、产生测试用例和进行测试结果评价。

（2）大大提高了测试自动化水平以及测试效率。在内存为 1G，CPU 为 1.8G 的 PC 机上，Findbugs 对 Eclipse、J2SE 和 Jboss 等开源软件进行分析，所耗时间不超过 65min，其中，J2SE 中的 rt. jar 分析，该程序包有 13083 个类，约 40M 大小，所耗时间只需 45min。在内存为 512M，CPU 为 2.4 G 的 PC 机上，TGSOFM 对约 万行的 VC 程序进行故障分析所耗时间只需 2min 左右[6]。

（3）部分解决了测试失效辨识问题，往往能发现其他测试技术难以发现的故障，保证了软件质量。

（4）有利于测试用例的重用，并可以应用成熟的理论和技术获得比较完善的分析结果。

2.2 存在的主要问题

基于模型的软件测试技术不能代替已有的其他测试技术，它是其他测试技术的一个有效补充。基于模型的软件测试技术虽然今年来得到广泛的研究，但是

有一些问题依然没有得到很好地解决：

(1) 误报问题。误报问题是基于模型的软件测试技术一个共性问题。通常基于模型的软件测试技术都属于静态分析技术,由于某些故障的确定需要动态的执行信息,因此,对于基于静态分析的测试工具来说,误报问题是不可避免的。为此,Cadar 等人提出,将动态测试与静态测试有机结合来解决误报问题[8]。

(2) 漏报问题。漏报问题主要由模型定义和模型检测算法引起。目前软件模式没有一个规范统一的定义形式。不同的故障查找工具都给出自己所能检测的模型定义,这些定义一般是一些自然语言的描述;同时,针对具体的软件模型,不同的测试工具一般设计自己的检测算法,从检测的效率和实现的复杂性上考虑,不同的算法给出不同的假设以降低计算复杂性,这导致对于相同的模型,用不同的测试工具进行检测得到的故障结果集合很大不同。文献[9]使用一些常用的 Java 程序故障自动分析工具对 Eclips 和 JDK 等开源软件进行测试,发现不同的测试工具得到的检测结果集差别较大;使用 TGSOFM 和 Reasoning 公司的工具对 4 个软件项目约 32 万行源代码关于内存泄漏和空指针引用故障进行测试,发现的故障集合有所不同。

(3) 模型机理。由于编程过程中,程序员具有较强的个体性,因此,软件模型是多种多样的。通常软件中的故障主要出自于程序员,如错误的理解造成的、二义性造成的、疏忽造成的和遗漏造成的。

例 某 C/S 结构、采用 Delphi 语言开发的数据库类软件,该软件大约 30 万行,分 7 个子系统、20 个模块,测试结果如表 1 所示[7]：

表 1　故障统计表

阶段	故障原因				
	错误的理解	二义性	疏忽	遗漏	合计
需求错误	0	0	5	2	7
概要设计错误	0	0	0	0	0
详细设计错误	0	0	4	1	5
编码错误	0	1	32	13	66
测试错误	2	6	0	0	8

在该例子中,由于疏忽造成的错误占错误总数的大约 63%,如果不考虑测试引入的错误,则疏忽造成的错误占错误总数的大约 80%。虽然单个软件具有很强地个体性,但是,大量的测试数据统计分析可以发现有些软件模型是具有共性的,比如说内存泄漏故障模型、非法指针引用故障模型等。

3　软件模型分类

软件模型又很多,目前常用的就有数百种。软件模型通常可以分为以下 7 类。

(1) 故障模型。该类模型是引起错误的常见软件模型,应该尽量避免,如内存泄漏故障(MLF)、使用空指针故障(NPDF)、数组越界故障(OOBF)、非计算类故障(ILCF)、使用未初始化变量的故障(UVF)、不完备的构造函数故障(ICF)以及操作符号故障(OAE)等。

表 2 是对 3197302 行、美国生产的 C++程序 4 类故障的测试结果[6]。可以看出,4 类故障总的故障密度为 0.48/KLOC,也就是说,大约 2000 行 C++的代码有一个故障。

表 2　Reasoning 公司测试工具 IQA 的测试结果

故障类型	故障数目	故障密度
MLF	561	0.16/KLOC
NPDF	380	0.12/KLOC
OOBF	167	0.05/KLOC
UVF	453	0.14/KLOC

(2) 安全漏洞模型。该类模型为他人攻击软件提供可能。而一旦软件被攻击成功,系统就可能发生瘫痪,所造成的危害可能更大,因此,此类漏洞应当尽量避免,如 缓冲区溢出漏洞模型、被感染数据漏洞模型、竞争条件漏洞模型以及风险操作 – 随机数漏洞模型等。从统计的数据来看,在上述所给出的故障类型中,其故障密度一般在 1 个故障/1 ~ 2KLOC。因此,测试这些故障是很有价值的。

(3) 差性能模型。该模型在软件被动态运行时效率比较低下,主要包括调用了不必要的类构造方法、空字符串的比较、参数为常数的函数定义、创建不必要的对象以及声明未使用的属性及方法等。

(4) 并发故障类型。由于线程启动的任意性和不确定性使用户无法确定所编写的代码具体何时执行而导致对公共区域的错误使用,如死锁等。

(5) 不良习惯模型。该模型主要是由于程序员编写代码的不好习惯造成的一些错误。包括文件的空输入、垃圾不及时、类和方法等命名不合法、对象序列化、参数传递不一致和代码安全性问题等。

(6) 代码国际化模型。该模型主要是在语言进行国际化的过程中,可能造成本地设置和程序需求不符的情况下,造成匹配错误。

(7) 易诱骗代码模型。该模型主要指代码中容易引起歧义的、迷惑人的编写方式。比如无意义的比较,永远是真值的判断,条件分支使用相同的代码,声明了却未使用的域等。

4 基于模型的软件测试流程

基于模型的软件测试过程中从源代码输入开始,经历预编译、词法分析、语法分析与语义处理、抽象语法树生成、控制流图生成和故障扫描等几个步骤,最后生成故障报表。监测流程如图1所示。

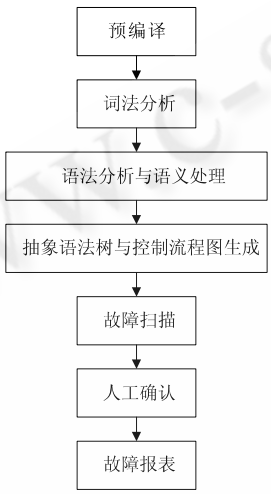

图1 测试过程

各个操作步骤如下所示:

(1) 预编译。由于源程序中存在宏定义、文件包含和条件编译等预处理命令,因此,在进行词法分析前必须进行预编译,将宏展开,这样有利于变量的查找。

(2) 词法分析。将预编译阶段产生的中间代码进行分解,形成各种符号表,为语法分析做准备。符号表的结构主要有 标识符表、类型表、关键表、常数表、运算符表和分界符表。

(3) 语法分析和语义处理。这一步主要是将输入字符串识别为单词符号流,并按照标准的语法规则,对源程序进一步分析,区分出变量定义、赋值语句和函数等等。语法分析的结果是生成语法树,并提供对外的接口。此外,通过语法树可以生成程序的控制流图和变量的定义使用链,为下一步的故障查找做准备。

(4) 抽象语法树生成。语法分析和语义处理之后生成抽象语法树,源程序中的所有语句都作为抽象语法树中的结点。抽象语法树是后续操作的基础,含有后续处理所需要的各种信息,如语句类型、变量名及类型等。

(5) 控制流图生成。在抽象语法树的基础上生成控制流图,控制流图必须能反映源程序的结构。

(6) 故障扫描。故障扫描是测试过程的关键步骤,首先定义测试模型,然后在控制流图上遍历对测试模型进行匹配,从而生成故障报表。工具所能检测的故障的集合取决于定义的测试模型集合。

(7) 人工确认。由于误报的存在,因此,需要对生成的故障进行人工确认。人工判定是基于模型的软件测试技术的主要消耗步骤。

5 结束语

面向模型的软件测试技术是测试效率很高的测试方法,是目前其他测试方法无法替代的。当然,它也不能替代已有的其他测试技术,因为,软件中的故障仅仅只依靠该方法进行测试是不行的。随着软件模型类型的逐步增多,测试算法的逐步成熟,以该方法为基础所开发的测试系统必然会在市场上广泛在我国,该技术的研究则刚刚起步,但已经收到了很好的效果。

参考文献

1 颜炯, 王戟, 陈火旺. 基于模型的软件测试综述. 计算机科学, 2004, (2).

2 Ball T, Bountimova E, Cook B. Through static analysis of device drivers. EuroSys, 2006:73 – 85.

3 Yang J, Sar C, Engler D. Explode:a lightweight, general system for finding serious storage sys errors. Proceedings of the USENIX Symposium on Operating System Design and Implementation (OSDI) Seattle, Washington, 2006.

4 Engler D, Chelf B, Chou A, et al. Checking system rules using system – specific, programmer – written compiler extensions. Proceedings of the Fourth Symposium on Operating Systems Design and Implementation, San Diego, CA, 2000.

(下转第 68 页)

（上接第 89 页）

5 Ashcraft K，Engler D. Using programmer – written compiler extensions to catch security holes. IEEE Symposium on Security and Privacy，Oakland，California，2002.

6 Hovemeyer D，Pugh W. Finding bugs is easy. ACM SIGPLAN Notices，2004，39（12）：92 – 106.

7 **杨朝红，宫云战，肖庆，毕学军. 基于模型的软件测试. 北京化工大学学报**，2007，（34）：85 – 88.

8 Cadar C，Canesh V，Pawlowski P M，et al. Engler EXE：Automatically generating inputs of death. 13th ACM Conference on Computer and Communications Security，2006.

9 Rutar N，almazan C，Foster J S. A comparison of bug finding tools for Java. Proceedings of the 15th IEEE International Symposium on Software Reliability Engineering，Saint – Malo，France，2004.