维普资讯 http://www.cqvip.com

软件的易测试性分析方法述评*)

刘菲菲 单锦辉 姜 瑛
(北京大学软件研究所 北京100871)

摘 要 软件测试是软件工程领域中重要组成部分。随着软件规模的不断扩大，测试工作的复杂性也不断升高，而改善、提高软件易测试性则是降低测试复杂性的有效手段。为达到这一目标，首先要能对软件的易测试性进行准确的度量。这种度量结果除了作为软件度量的一个量化指标外，还应能为改善易测试性提供指导、为测试的设计提供有效帮助。本文对现有软件易测试性分析方法进行归类，将已有技术大致归并为基于复杂度分析、基于信息论分析、基于PIE技术分析和基于UML类图分析四种基本类型。简要介绍了每一类方法，对这些方法的特点与不足进行分析比较，并探讨今后的研究方向。

关键词 软件易测试性，软件复杂性度量，信息流图，PIE技术

A Review of Approaches for Software Testability Analysis

LIU Fei-Fei SHAN Jin-Hui JIANG Ying
(Software Engineering Institute, Peking University, Beijing 100871)

Abstract Software testing is an important part in the domain of software engineering. With the enlargement of software scale, software testing becomes more complex. The improvement of software testability is an efficient method to reduce the testing complexity. The precise measurement of software testability is the first step to achieve the above goal. Besides as a quantified guideline, the measurement result should be direction for the improvement of software testability and be helpful to testing design. In this paper, the existing analysis methods of software testability are classified roughly as analysis based on software complexity, analysis based on information theory, analysis based on PIE technology and analysis based on UML class graph. This paper briefly introduces some representative methods of each class, compares the merits and demerits of these methods. Finally, future research directions are discussed.

Keywords Software testability, Software complexity metric, Information transfer graph, PIE technology

1 引言

随着软件产业的不断发展，软件不断向系统化、集成化发展，规模越来越大，软件测试的重要性也越来越显著。但软件测试本身是具有很大复杂性的工作，从软件设计到开发的整个过程中，40%的资源是用于测试[1]。如何在保证软件质量的前提下，有效降低软件测试的开销?更有效的测试方法固然重要，但如果软件自身更容易被测试，同样也可有效加快工程进度。软件的易测试性度量正是在这样的一个前提下被提出。

1991年Freedman提出把软件易测试性定义为可控制性和可观察性的集合[2]，其中可控制性主要指能更好地通过软件的输入来控制它的输出；可观察性主要指通过输出能更好地分析测试的结果。随时间推移这一概念不断丰富完善。目前软件易测试性度量主要包含以下几个方面：可操作性、可控制性、可观察性、简单性、适宜性(suitability)、稳定性等。将这些属性归结到一点，就是高易测试性意味着更容易在测试活动中发现软件中所包含的错误。

从软件的易测试性被提出以来，一方面它逐步成为衡量软件产品质量优劣的一个重要尺度；另一方面，软件的设计人员也通过新的易测试性设计方法改进软件过程，提高软件质量。不论是以上哪一方面，合理并有效的易测试性分析都是重要的一环。不同于普通的质量度量，这种分析不能只是一种定性的评判。我们需要一些可直接观测的数据。在某些情况下，这些数据并不一定要很精确，但必须具有直观性，从而为软件的改进提供较为可靠、有指导性数值依据，为软件测试的设计提供有效帮助。而且这种分析在软件生产过程中，开始得越早，对于节省软件投入，提高效能所发挥的作用也就越大。

如何进行易测试性的分析，以及制定怎样的一个度量标准，从20世纪90年代初就有很多学者做了这方面的尝试[2~4]。但这些方法之间存在明显的差异性，因为每种方法可能针对的是不同软件属性(当前主要是针对可控制性和可观察性)。加之各种软件之间结构的差异，设计方法的差异，不同设计层面的差异，测试方法的差异等，区分就更加明显。其中一些比较典型的技术，大致可以分为以下四类：

第一类：引入新的度量标准，对软件的复杂性进行分析。主要代表有：文[5]提出的基于程序流图的方法和文[6]提出的基于数据流的方法。

第二类：基于信息流图，按信息论理论，以模块的输入输出信息为依据对可控制性和可观察性进行定量计算，从而确定其易测试性。主要在文[8~10]中提出。

*)本文受国家"八六三"高技术研究发展计划项目(2001AA113070)、国家自然科学基金项目(编号：60373003)、国家重点基础研究发展规划(973计划)项目(编号：2002CB31200003)、中国博士后科学基金项目(编号：2003034077)资助。**刘菲菲** 硕士研究生，主要研究方向为软件测试。**单锦辉** 博士，主要研究方向为软件测试、构件技术、形式化方法。**姜 瑛** 博士研究生，主要研究方向为软件工程、软件测试、软件构件技术。

维普资讯 http://www.cqvip.com

第三类:以 PIE(propagation analysis, infection analysis and execution analysis)技术为基本原型,针对程序中的某一处进行分析,计算在该处故障能被检测到的概率。文[11～14]中所提出的方法都属于这一类。

第四类:基于 UML 类图的分析技术。它主要以 UML 类图为主要分析对象,找出并消除构件中类之间的复杂依赖关系,以提高软件的易测试性[15,16]。

除以上几种大的方法类外,也有一些从其它角度对易测试性分析进行研究的工作,如:对软件全局易测试性和局部易测试性之间相关性进行形式化分析[17]、基于神经网络对易测试性进行分析[7]等。下面简要介绍上述四类主要技术的基本实现方法,并进行对比分析。

2 基于复杂度度量的方法

受软件度量学的影响,最先出现的一类易测试性分析方法与软件复杂性度量的思路基本一致,并部分借鉴了其相应的方法(McCabe 的度量方法和 Nejmeh 的度量方法)。主体思想都是基于某种流图,对图中各种结构信息(点、边、路径数、循环数等)进行统计计算。这些信息的统计可以以模块为单位,不依赖于程序的执行和测试策略,所得出的结果可用于不同程序结构的优劣比较,找出软件中的重点测试部位,指导软件的改进。

基于软件复杂度形式的度量方法与传统的复杂性度量在使用方法上有着一定相似性,但由于两者所度量的目标不同,所以在度量标准的选择上又有比较大的差别。如按 McCabe 的程序复杂度度量,将一个多重嵌套的选择结构变为一个顺序化的选择结构序列可有效降低程序复杂度[18]。但从测试的角度来看,这样的修改是否合适,还需要根据实际情况做进一步分析。无疑当测试以分支覆盖为标准时,多重嵌套结构需 $N+1$ 条路径才能达到覆盖,而顺序化结构则只需2条;但如果以路径覆盖为测试标准,在没有循环的情况下,前者只要 $N+1$ 条路径就可以达到路径覆盖,而后者需要 2^N 条(N 是选择结点的个数)。由此可以看出对于软件的易测试性必须制定更适合的度量标准。

文[5]中定义向量 $\alpha_D=(\overline{M(D)}, P(D), V(D), \varepsilon(D))$ 为程序 D 的测试性矢量,其中 $\overline{M(D)}$ 是 D 的平均路径长度,$P(D)$ 是路径总数,$V(D)$ 是圈复杂度,$\varepsilon(D)$ 是循环数。同时,各路径平均循环数 $CP_D=\frac{\varepsilon(D)}{P(D)}$ 和各路径平均圈复杂度 $VP_D=\frac{V(D)}{P(D)}$ 同样也可以作为易测试性度量的指标,以上两个值越大,则说明 D 越难测试。

文[6]中则指定了以数据流图中边的数目、点的数目、变量定值的数目、定值-引用对(d-u pairs)的数目等为指标,并将这些值的倒数值作为易测试性分析结果。

3 基于信息论的分析方法

该方法在文[8～10]中提出,主要综合了两个具体的技术,分别是:信息转换图分析技术[8]和基于信息论的易测试性计算[9,10]。

3.1 信息转换图

该技术最早在硬件测试中得以应用,并取得良好效果。其核心是在一个信息转换图的基础上提取图中的结构信息,用以对软件测试的过程进行辅助。信息转换图模型由结点、转换和边三种成份组成。其中结点代表了操作,功能调用或是声明。系统的输入输出也分别是一结点,称为终端结点。在图中,结点一般用圆形来表示。转换的模式有三种:(1)连接(junction)模式:一个数据目的结点的执行要同时使用多个数据源结点的输出信息;(2)归因(attribution)模式:一个数据目的结点执行只使用多个结点中的某一个数据源信息;(3)选择(selection)模式:数据源结点将自己的输出数据只送入多个数据目的结点中某一个。边则用于连接结点与转换模式。

信息转换图中既可以表示控制流也可以表示代表变量定值与使用关系的数据流,或是两者的综合。而当其用于定义信息路径时,则更有效的是对流的分析。一个流就是图中从输入到输出的一条信息路径,它包含一组结点、转换和边。可以把流看成是信息转换图的一个子图,因为它代表了一个可脱离系统其它部分而独立运行的基本功能。

3.2 基于信息转换图的分析

尽管信息转换图本身会作为后续信息论技术计算的依据,但其自身已包含了很多与易测试性相关的信息。在文[8]中对此进行了描述,主要有以下两点:

(1)信息转换图中所包含的流的个数与环计数度量和 N 路径度量具有以下关系:

$$V-NF_{min}=N_{seq}-1,\ NP=NF_{total}$$

其中:V 代表环计数、NP 代表 N 路径数、NF_{min} 代表图中为达到结点覆盖所需最小的流数目、N_{seq} 代表连续的、非嵌套的控制结点的数目、NF_{total} 代表图中所包含流的总数。

这使得信息转换图中的流的个数同时可以作为程序复杂度度量的标准。而这种复杂性也可以看作易测试性度量的一个方面。

(2)有助于生成合理的测试用例,因为在信息转换图中流的覆盖包含代码覆盖和控制流图的分支覆盖,并实际代表了程序的某一种功能,所以是一种更严格的测试标准。

3.3 基于信息论的易测试性计算与分析

基于信息论理论,以交互中所包含的信息量为度量标准是这一分析方法的核心。其主要思想是:以图中的各个结点为单位,分别计算该结点在不同信息流中的可控制性和可观测性(具体的计算公式见文[10])。

因为对流的分析可以用来决定测试用例的生成,每一个模块在具体的流中对应的可控制性和可观测性也就反映了测试用例针对这个模块可能含有的错误,在最好的情况下的揭示能力。一般而言,针对计算结果大致有以下几个方面的工作可做[10]:

(1)以达到模块覆盖为标准,找出测试能力比较强的一组流,并生成相应的测试用例。

(2)各个流中易测试性都比较差的模块,是测试的薄弱环节,要进行一些有针对性的测试。

(3)当使用层次式设计模型时,针对高层中易测试性相对较弱的模块,要对其下一层次的细化结构进行重点测试。

(4)对以上方法均不便于解决的问题,可通过部分调整或修改程序结构的方法。如修改分支结构,在适当位置添加输入或输出结点等方法。从而改变信息转换图,提高易测试性。

4 基于 PIE 技术的分析方法

PIE 技术最早是由 Voas 等人提出[3],它基于故障/失效模型,估计程序中某一处的 P、I、E 三种可能性:

维普资讯 http://www.cqvip.com

E:在假定输入呈均匀分布的情况下,执行某部分程序的可能性

I:某部分程序中如果存在变异,该变异会错误地改变数据状态的可能性

P:如果数据状态被错误地改变,这种改变会影响到程序输出的可能性

对这三种可能性分别进行分析,分析结果决定该处的易测试性。该方法主要用结构分析的方法对程序的语句进行定量计算[11]。

'E'是对执行的可能性的估计,其上限为1。顺序语句因为一般都要执行,所以其 E 值为1。而分支语句一般要对其是否执行进行估算,或以平均值来计算。在分支汇合点则要取各分支 E 值的总和。

'I'是对产生错误的可能性进行分析。对于输入或输出语句,每一个输入/输出变量都有一个 I 值,一般为1。表达式语句的计算则较为复杂,语句的 I 值取决于其中所包含的运算符、变量、常量的数目和类型。而这每一个元素都会有自己的 I 值,加权平均后得到语句的 I 值。操作数为常量时 I 值为1,为变量时,取最后一次对该变量进行定值的语句的 I 值。运算符的 I 值见表1。

表1 运算符 I 值计算表

操作符	Iw
+,−,×,÷	1
>,<,=,≥,≤,≠	DRR * 或1
a mod b(b 是常量)	(b−1)/b
a div b(b 是常量)	([a/b]−1)/[a/b]
其它(如:Trunc,round,div,mod,…..)	1/2

* DRR 是输入域基数同输出域基数的比率

对'P'值进行估计时,追踪语句所赋值的变量在后续执行过程中的使用,直到输出,中间记录经过的所有语句。这些语句 I 值之积就是 P 值。如果无法到达输出结点,则 P 值为0。

当计算出语句1的 E_1、I_1、P_1后,三者的乘积就是易测试性值。

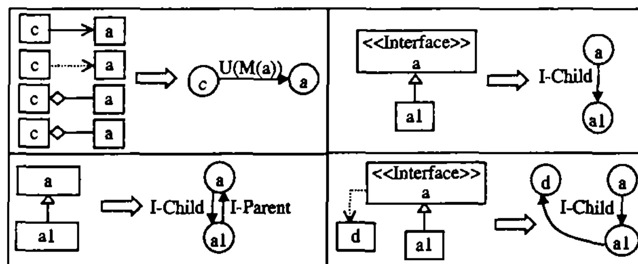
与文[14]中所提出的方法相比,上述方法侧重于结构上的分析。而文[14]中提出的方法更多基于统计的方法,通过对程序进行插装或变异体的执行,根据结果的统计规律,近似估计三种值。执行的程序变异体越多,得出的估计结果也就越准确。

5 基于 UML 类图的分析方法

现有基于类图的易测试性分析主要是针对类图中所反映的类之间的相互关系,相互关系越复杂,意味着程序结构也就越复杂,也更容易隐藏程序错误。合理地分析和度量这种相互关系,并以分析结果为指导,改善系统结构,降低复杂度,就是这类方法的主要目标。下面对这类方法进行简单描述。

文[15]中,作者认为按类图进行测试设计时,最应注意的是类交互(class interaction)—当两个类之间存在着两条以上的依赖路径时,就将这样的一种拓扑结构称为类交互。因为这样的结构将是测试的难点。文中首先提出了一种模型—(Class Dependency Graph, CDG)—用于捕获类交互。该模型可按表2中的规则由 UML 类图得到:

表2 从 UML 类图到 CDG 图的基本转换形式



在得到 CDG 图后,按一定的规定与算法,度量整个软件中类交互结构的数量,最后根据度量结果提出设计上的改进意见,以降低其复杂度。加入易测试性分析后的软件开发工作流程如图1所示:

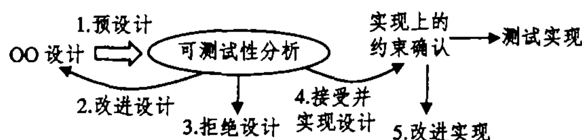


图1 加入可测试性分析后的工作流图

在设计的改进方面,作者提出了以下几种方法:

(1)在允许的情况下减少继承的使用,尤其是复杂的继承关系,用接口关系进行替代。

(2)定义并使用衍型(stereotypes),如:《create》、《use》、《use-consult》、《use-def》等对类之间的关系进行更清楚的表示。

6 对比与分析

对比几种易测试性的分析方法,可以看出:定量计算对易测试性给出了严格的度量,它的结果对测试用例的生成与选择、测试结果的分析、系统结构的优化都有比较好的辅助作用。并能反映出很多无法从各种图形表面直观获取的内在信息,这是它们共同的优势所在。但相比较而言,它们之间也有很大的不同。

基本方法中的第一类方法,主要是从传统的软件复杂度的度量方法转化而来。虽然软件复杂度与易测试性有一定必然联系,但完全靠复杂度来说明易测试性则并不充分。这种方法忽略了要针对测试这一特殊的要求,无法得出各种复杂结构对测试所带来的具体影响,从所得出的结果也无法得到对于改进程序易测试性有针对性的具体方法。而在第二、三类方法中,这一问题明显得到了比较好的解决。不论是基于信息流图,还是基于 PIE 技术,一方面它们都对软件的易测试性提出了一套具体的、量化的评测指标;另一方面它们都有另一目标,就是以具体的量化数据来分析测试中的难点,找出薄弱环节,使测试不再盲目,更有针对性,从而达到更好的测试效果。

虽然基于信息论的方法与基于 PIE 技术的方法在目标上有着明确的相似,但它们实现目标的方法却各有特色。基于信息论的方法以构成软件的模块为最小计量单位(最小的模块可以是过程调用,也可以是单个语句),考查每个模块对信息、数据的影响,但不考虑代码实现的细节。使得这种方法可以适用于软件设计的不同层次,对软件的整体设计会有很深刻的影响。而通过增加辅助的输入输出信息来改变信息流的方法,一方面可以在代价很小的情况下,有效改善软件的易测

维普资讯 http://www.cqvip.com

试性;另一方面,它也与测试技术中的BIT(build-in test)技术不谋而合,为该技术的实现提供了有效的指导。

基于PIE技术的方法与前者相比则更多的侧重于软件的代码实现。该方法严格分析了代码中每一个细小元素(包括变量,运算符,表达式等)自身的正确性及它们的累积效应给整个程序的正确性所带来的影响。这样的措施虽然使分析变得更严格,但同时也将方法的使用范围限定为白盒分析。如果要将这种方法在更高的设计层次上使用,就必须对三种分析提出新的合理的量化标准,这一点在实现上有一定的难度。

纵观前三类基本方法发展历程,可以看出它们都明显带有结构化程序设计的烙印,与之相比基于UML类图的易测试性分析方法更多地强调和运用了面向对象的设计工具。对软件的复杂度,尤其是与测试相关的复杂性给出了新的度量标准。从根本上讲它只是第一类用复杂度分析来度量软件的易测试性思想在面向对象领域的一个延伸。其标准的选择具有一定的合理性和充分的可操作性。由于与相应的面向对象的开发工具相结合,使得这种方法更容易通过相应的自动化工具来辅助实现。表3对几类方法进行了综合对比。

表3 四种分析技术性质对比表

性质 \ 分析技术	基于复杂性的分析技术	基于信息论的分析技术	基于PIE的分析技术	基于UML类图的分析技术
作为易测试性度量标准	√	√	√	√
分析基础	代码级实现	·代码级实现·系统分析模型·系统设计模型	代码级实现	·OO分析模型 ·OO设计模型
分析的对象	数据流、程序流	信息流、数据流	程序流	类图
分析粒度	模块	语句、代码模块、系统模块(均以特定的信息流为背景)	语句、代码模块	类
对软件测试的辅助	无	辅助选择测试重点、辅助生成测试路径	辅助选择测试重点、辅助分析错误位置	辅助选择测试重点
以何种方式指导软件易测试性改进	无	适度增加或修改数据输入与输出	无	·减少类交互 ·使用衍型重新表示类关系

结束语 Voas J.M提出"今后的十年将是提高软件质量的十年"[19]。在对软件质量要求越来越高,测试工作量越来越大、越复杂的情况下,具有良好易测试性的软件对节省开发投资,缩短开发周期,降低维护费用等方面都有着重大意义。现有的易测试性分析方法针对传统的软件开发模式已基本体现出这种优势。

但在将来的软件工程中,基于复用的构件的开发将成为主流。与传统的软件测试相比,构件测试有着自身的固有特点:(1)不能对构件的执行环境和用户的使用模式进行完全准确的预测,故构件开发者不能完全、彻底地对构件进行测试,并且很难确定何时结束测试;(2)构件复用者和第三方测试人员通常无法得到构件的源代码及详细的设计知识,通常只能对构件进行黑盒测试,使得构件执行过程中的一些错误被隐藏。

构件测试的这种情况使良好的易测试性成为构件高可靠性和可信任的重要保证。那么如何在没有源代码的情况下对构件,甚至是整个系统的易测试性进行分析,现有方法尚未明确解决这一问题。我们认为按信息论理论,以软件系统内各构件为基本模块建立信息流图,进行易测试性分析,并以各构件之间的交互复杂度为辅助度量,将可能是解决这一问题的一个有效途径,我们将加强这方面的研究。

参考文献

1 Pressman R S. Software Engineering. New York: McGraw-Hill, 1992
2 Freedmam R S. Testability of software components. IEEE Tran. Software Eng, 1991, 17(6): 553～564
3 Voas J M. PIE: A dynamic failure-based technique. IEEE Trans. Software Eng, 1992, 18(8): 717～727
4 Voas J M, Miller K. Software testability: the new verification. IEEE Software,May, 1995. 17～28
5 崔伟宁,宫云战,崔培枝.基于程序流图的一种软件测试性计算方法.2000年全国测试学术会议(CTC'2000),北京.280～284
6 Yeh Pu-Lin, Lin Jin-Cherng. Software testability measurements derived from data flow analysis. In: Proc. of the 2nd Euromicro Conf. on Software Maintenance and Reengineering. Mar. 1998. 96～102
7 Khoshgoftaar T M, Allen E B, Xu Z. Predicting testability of program modules using a neural network. In: Proc. of 3 rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology,Mar. 2000. 57～62
8 Le T Y, Robach C. From hardware to software testability. In: Proc. of International Test Conf. Oct. 1995. 710～719
9 Le T Y, Robach C. Testability measurements for data flow designs. In: Proc. of 4th Intl. Software Metrics Symposium. Albuquerque, NM, USA . Nov. 1997. 91～98
10 Nguyen T B, Delaunay M, Robach C. Testability analysis for software components. In: Proc. of Intl. Conf. on Software Maintenance,2002. 422～429
11 Lin Jin-Cherng, Lin Szu-Wen. An analytic software testability model. In: Proc. of the 11th Asian Test Symposium 2002 (ATS '02). 278～283
12 Lin Jin-Cherng, Lin Szu-Wen, An Ian-Ho. estimated method for software testability measurement. In: Proc. of 8th IEEE Intl. Workshop on Software Technology and Engineering Practice:[incorporating Computer Aided Software Engineering]. London UK. July. 1997. 116～123
13 Lin Jin-Cherng, Lin Szu-Wen, Huang Louis. An approach to software testability measurement. In: Proc. of Asia Pacific Software Engineering Conf. and Intl. Computer Science Conf. Hong Kong,Dec. 1997. 515～516
14 宫云战,刘海燕,万琳,杨朝红.软件测试性的分析与设计技术研究.2000年全国测试学术会议(CTC'2000,北京).271～274
15 Baudry B, Le T Y, Sunye G. Testability analysis of a UML class diagram. In: Proc. of 8th IEEE Symposium on Software Metrics, 2002. 54～63
16 Jungmayr S. Testability measurement and software dependencies. In: Proc. of the 12th Intl. Workshop on Software Measurement. Magdeburg, Germany,Oct. 2002. 179～202
17 Le T Y, Ouabdesselam F, Robach C. Analyzing testability on data flow designs. In: Proc. of 11th Intl. Symposium on Software Reliability Engineering, ISSRE 2000. San Jose, CA, USA,Oct. 2000. 162～173
18 Parther R E. An axiomatic theory of software complexity measure. Comput. J., 1984, 7(4): 340～347
19 Whittaker J A, Voas J M. 50 Years of software: Key principles for quality. IT Professional, 2002, 4(6): 28～35