一种软件代码精细分析技术

唐 英,徐良贤
(上海交通大学计算机系,上海 200030)

摘 要: 精细分析对提高关键软件的安全非常重要,并因计算量大而需要自动化.本文基于 J M Voas 的 Fault/Failure 概念模型及其 PIE 分析,提出一个实用的软件代码精细分析技术和工具.文中描述了软件代码精细分析的全过程,重点讲述"自动分析记录"的工作流程及执行、感染和传播这三个关键分析的概念性算法,并给出了工具的框架图解.此外还提出粒度可调思想,能调节分析的精度和范围,较好地解决了 Fault/Failure 模型的限制,拓展了用途.最后本文给出了一些具体的应用思路,如放置警报器、评估可靠性、配置资源、设计测试实例等,以及对面向对象软件测试的启发.

关键词: 软件测试; 可测试性: 缺陷传播; 关键软件
中图分类号: TP311.5 **文献标识码:** A **文章编号:** 0372-2112 (2002) 12A-2163-03

A Precision Analytical Technology for Software Code

TANG Ying, XU Liang-xian
(*Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai* 200030, *China*)

Abstract: Precision analysis is very important for key software to improve security, and needs to be automated because of large computing. Based on J M Voas' notional model of Fault/Failure and the PIE analysis, this paper develops a practical technology and presents an automated tool to analyze software code in detail. it describes the whole work flow of precision analytical process, especially the "automated analysis & record" phase and the notional arithmetic of three key analyses: execution, infection, propagation, and gives frame diagram of the tool. What is more, it presents the idea of adjustable granularity, which can change the precision class and analysis range, has in a way overcome the limits of Fault/Failure model and expands the possible applications. At the end, it also points out some applications such as to place assertion, estimate teliability, configure resource, design test case, and mention the elicitation to OO software testing.

Key words: software testing; testability; fault propagation; key software

1 引言

随着软件工程领域的进展,软件测试也从工程和成本的角度着手,提出了一系列新的观点和技术,从源头上堵住了大量的软件缺陷.

然而由程序员引入的缺陷仍然无法避免,它可能是一个算法设计缺陷,也可能是一个简单的代码错误,Ariane501 火箭爆炸便是典型案例[1].因此对关键软件进行精细的代码分析是必要的,而这非常费时费力,实现自动化分析才是出路[2].

另外,随着软构件思想的普及,越来越多的 COTS 上市.由于它是现成的软件,往往使用基于代码的测试技术[3].

有些学者在探索,其中 J M Voas 博士针对"软件出错的可能性是多大"这一惯常思路,反其道而提出"含有缺陷的软件不出错的可能性是多大?"并总结前人经验,初步建立 Fault/Failure 模型为:(1)必须有一个输入引起含缺陷的代码被执行.(2)执行后必须产生一个数据状态错误.(3)错误的数据状态必须传播到输出.

这分别称为执行性条件、感染性条件和传播性条件,描述了软件缺陷是怎样导致程序出错并被观察到,并指出缺陷在什么地方能够躲过测试的检验[4].

2 软件代码精细分析

2.1 软件代码精细分析概述

精细分析是在软件被编码出来后进行的细粒度深入分析工作,应先进行其他常规测试,尽量接近正确,即已经编译通过,并且在语法和语义上符合说明书,代码中大型复杂缺陷已排除.

2.2 自动分析记录的工作流程

精细分析需要实现自动化才能完成大量的计算和分析,靠人工是不现实的.因此自动分析记录是核心,其工作流程见图 2.

代码被分成一个个的点,并定义为"一段能改变数据状态

收稿日期:2002-06-10;修回日期:2002-11-21

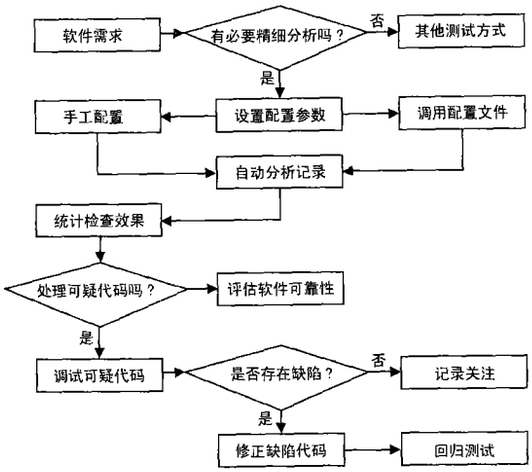

图 1 软件代码精细分析过程

(包括输入输出和程序计数器)的源代码".例如 If $a > b$ then 是一个代码点，$a := b$ 也是一个代码点，语句 read(a, b)则是两个代码点.代码点是原子性的，只要一条语句被执行，则整个代码点中所有语句都被执行，数据状态的变化只能在代码点之间被观察到.

数据状态是指一集变量与其值之间的映射.代码点前后的数据状态分别称为前状态和后状态.例如一个代码点包含变量 x 和 y，其值为 3 和 4，同时程序计数器为 100，则数据状态为$\{(x,3),(y,4),(pc,100)\}$.

已知一个代码点的正确前后状态，如果有一个输入值的正确前状态通过代码点后，变成一个与正确后状态不一致的数据状态，则该代码点包含一个缺陷(fault)，而不正确的数据状态称为错误(error)，这个过程称为感染.缺陷是否产生错误与输入值有关.

对于一个被感染的数据状态，它可能传播到输出而被看到.这就是故障(failure).它也可能在程序执行过程中消失了，叫信息丢失，这对关键软件来说也是危险的.关键软件不希望从错误的程序得到正确的输出，因为如果程序是错误的，则至少存在一个输入使该程序产生故障.

综上所述，要找到一个缺陷，则执行、感染和传播都必须出现，否则不会产生故障，无法分析和跟踪.通过统计这些数据，可以反映软件的某些特性.

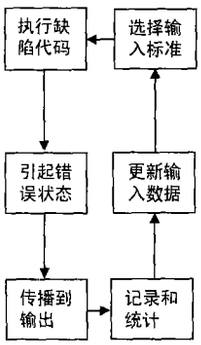

图 2 自动分析记录的工作流程

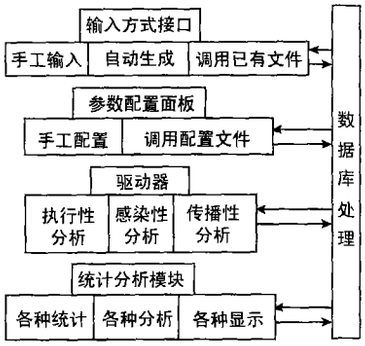

图 3 软件代码精细分析工具框架图

记录所选用的输入，可用于分析和完善随机输入的分布描述、或精简优化根据程序结构所选择的输入数据，使下一次分析更为准确.因此维护和更新一个输入数据库能够提高精细分析的效率和智能.

2.3 软件代码精细分析工具框架

现在，我们可以初步提出软件代码精细分析的框架，如图 3 所示.它以一个数据库为后台，以分析驱动器为核心，以输入接口和配置面板为交互界面，以各个统计分析和显示模块为功能，提供软件代码内部详尽的信息.在这些详尽信息的基础上，可以做出很多应用.

3 核心算法描述

假设一个程序 P 中有一个代码点 t 含有缺陷，则每次随机输入并执行后，程序会有如下四种情况之一：

(1)代码点 t 未被执行，因此输出正常；

(2)代码点 t 被执行，但未产生错误的数据状态，因此输出正常；

(3)代码点 t 被执行，并且产生错误的数据状态，但程序仍然正确输出；

(4)代码点 t 被执行，并且产生了错误的数据状态，从而导致了错误的输出.

因此可分别对程序做执行性、感染性和传播性三种分析，了解含有缺陷的代码点 t 在随机输入的情况下被发现的可能性是多大.下面给出这三种分析的概念性算法，实际的实现要复杂些.

执行性分析比较简单，就是在每个代码点后插入一个写语句，随机执行程序 n 次来统计.

感染性分析主要为代码点 t 创建一个语义变异代码点 m，在 t 的前数据状态空间中随机选取一个数据状态，然后并行执行 t 和 m，比较结果是否相同来统计.

传播性分析主要是在代码点 t 的前数据状态中随机选择一个数据状态，扰动这个数据状态中的活动变量 a 的值，然后并行执行代码点 t 后的代码，包括扰动后和未扰动的，比较其结果是否相同来统计.

以上三个分析统称 PIE 分析，由 Voas 给出和命名[4].与随机测试的区别在于 PIE 是一个个地检查代码点，而随机测试将程序当作一个唯一的黑盒，因此 PIE 分析兼有白盒和黑盒测试的优点.由于 PIE 分析不判断正确性，因此不需要预期结果，可高速自动工作.但动态测量比静态测量所需的计算高出三个数量级，例如一个简单的 PIE 分析可能就需要 200,000,000 次程序运行[5].对此，有学者研究了如何精简所需输入和运行次数，比如 Tim Menzies 等人的 HTO[6].

能最大限度地产生故障的测试方案才是最好的，这就必须让执行性、感染性和传播性都尽量大.这三个因素的乘积叫做可测试性，它是一种可能性，是一个闭区间[0,1].

4 软件代码精细分析的应用

4.1 用于不同粒度的分析

可以扩展前面关于代码点的定义，调节代码点的粒度，从

单个变量和单个子操作,到整条复杂语句、整个函数、整个模块,甚至整个程序.接受测试的代码规模也可调,可以只分析一个模块、函数,甚至语句,使精细分析工具更灵活,应用更广泛.同时,收集的信息层次丰富,对软件代码情况的把握更完整.不同粒度的分析信息有不同的含义和用途,能够拓展一系列新的应用.

粒度的扩大和缩小,也使代码分析的精度降低或提高,从而敏感度也跟着降低或提高.如果降低敏感度,则能够容忍和发现更大更复杂的缺陷,并能够应用于软件开发的更早期阶段.粒度的扩大和缩小,还能够把握耗用的时间和资源.显然粒度大的情况下能够更快地分析完毕,数据量也更少.不同粒度能够满足各种软件的不同质量需要,以合理的代价提供合理精度的分析信息.

4.2 放置触发器

触发器通过设置触发条件,展示软件执行时的内部中间状态,从而增强了软件的可观察性,并缩短路径,提高可测试性,无需覆盖天文数字般的可选路径.

问题是触发器应该放在哪里?精细分析能找出哪些代码点的执行性、感染性和传播性较低,从而针对性地放置各类触发器[7].例如,通过精细分析后,统计出每个代码点的可测试性 θ_i,并按升序或降序排列.然后根据所有代码点的整体情况以及可用资源,划定一个适当的 ε,对所有 $\theta_i < \varepsilon$ 的代码点,在其后紧跟着插入一个触发器.

4.3 其他应用

评估软件可靠性.精细分析能够对可测试性进行量化分析,并对信心有一定的影响,让测试员对程序"无错误可隐藏"达到一个可接受的自信度,并决定什么时候停止测试[8].

协助设计测试实例.精细分析只是一种分析技术,而不是测试,但可以帮助设计测试实例,从而提高测试精度和效率.

指导测试资源的分配.精细分析相当于探矿,知道哪些地方可能有金子,以及埋藏多深.从而优化配置测试资源.

容错.缺陷并不是导致软件故障的直接原因,数据状态错误才是.通过对数据状态的密切关注,可建立另一条防线来对付故障[9].

5 结语

Voas 模型是基于两个假设之上的:单个缺陷和简单缺陷.但在实际中由多种软件缺陷复合引起的程序错误和由多个代码点共同组成的大型缺陷很常见,都不能直接使用 PIE 来分析.本文提出粒度可调节的思路,使得这一分析技术有了更为广泛的适用性.

面向对象的软件具有信息隐藏、方法重写、类继承等特性,用精细分析技术来分析会很困难[10].假如把代码点改为方法,则模型可变形为"对象中含有缺陷的方法必须被调用,调用的方法必须改变对象的数据项,产生一个错误的对象状态,这个错误的对象状态必须反映到对象的输出参数,并被观察到."

尽管这个模型很简单,但其重要性和潜力远未得到认识和利用.以前许多基于代码的测试技术都只体现了这个模型的一部分.

参考文献:

[1] J L Lions. Ariane 5 Flight 501 Failure: Report of the Inquiry Board [DB/OL]. http://java. sun. com/people/jag/Ariane5. html. Paris, 1996, July 19.

[2] Elfriede Dustin, Jeff Rashka, John Pault. Automated Software Testing: Introduction, Management, and Performance [M]. Addison Wesley Longman, 1999.

[3] E J Weyuker. Testing component-based software: A cautionary tale[J]. IEEE Software, 1998.

[4] J M Voas, K Miller, J Payne. PISCES: A tool for predicting software testability[A]. In Proc. Of the Symp. On Assessment of Quality Software Development Tool[C]. New Orleans, LA: IEEE Computer Society TCSE, May 1992. 297 - 309.

[5] J Voas, G McGraw. Software Fault Injection: Inoculating Programs Against Error[M]. John Wiley & Sons, 1998.

[6] Tim Menzies, Christoph C Michael. Fewer Slices of PIE: Optimising Mutation Testing Via Abduction[DB/OL]. Kaiserslautern, Germany: SEKE'99, June 17.

[7] J Voas. Software Testability Measurement for Assertion Placement and Fault Localization [DB/OL]. AADEBUG, 1995. 133 - 144.

[8] Mark C K Yang, W Eric Wong, Alberto Pasquini. Applying Testability to Reliability Estimation [DB/OL]. http://www. stat. ufl. edu/~yang/publications/reliability. pdf.

[9] J M Voas, K Miller. Dynamic testability analysis for assessing fault tolerance[J]. High Integrity Systems Journal, 1994, 1(2): 171 - 178.

[10] James M Bieman, Sudipto Ghosh, Roger T Alexander. A Technique for Mutation of Java Objects[DB/OL]. Automated Software Engineering, 2001.

作者简介:

唐　英　男,1971 年生于江西抚州,博士研究生,南昌大学信息工程技术研究中心讲师,我国计算机专业的龚哑人博士生,主要研究领域是软件测试和软件过程,软件无障碍等.

徐良贤　男,1938 年 2 月出生于上海市,浙江宁波人,上海交大计算机系教授,博导,IEEE 计算机协会会员.