软件分析技术进展

梅 宏[1)] 王千祥[1)] 张 路[1)] 王 戟[2)]

[1)](北京大学信息科学技术学院 高可信软件教育部重点实验室 北京 100871)

[2)](国防科学技术大学计算机学院 并行与分布处理国防科技重点实验室 长沙 410073)

摘 要 软件分析技术的研究已有较长历史,相关成果也在软件生命周期的不同阶段中得到了广泛应用.软件生命周期中不同活动所需要的软件分析技术既不完全相同,又有许多交叠,且不同的分析技术之间互相影响.文章在讨论了软件分析的基本概念之后,主要从静态分析与动态分析两个方面介绍了一些主要的软件分析技术以及部分相关分析工具.结合软件的质量问题,文章还探讨了一些分析技术与软件质量属性的相关性,以便于人们在分析特定的软件质量属性时,选取合适的技术与工具.最后,文章展望了软件分析技术的发展趋势.

关键词 软件分析;静态分析;动态分析;软件质量

中图法分类号 TP301 **DOI 号**: 10.3724/SP.J.1016.2009.01697

Software Analysis: A Road Map

MEI Hong[1)] WANG Qian-Xiang[1)] ZHANG Lu[1)] WANG Ji[2)]

[1)](*Key Laboratory for High Confidence Software Technologies of Ministry of Education, School of Electronics Engineering and Computer Science, Peking University, Beijing* 100871)

[2)](*National Laboratory for Parallel and Distributed Processing, School of Computer Science, National University of Defense Technology, Changsha* 410073)

Abstract Research on software analysis has long history. It has been widely used in many processes in software lifecycle. The software analysis technologies that are used in different processes are different, while there are many interleaves among them. This paper discusses the concept of software analysis, followed with main software analysis technologies and related tools, from view of static analysis and dynamic analysis. Some relationships between software analysis and software quality characters are introduced, so as to provide some hints when some specific software character is under analyzing. The future of software analysis is discussed in the end of this paper.

Keywords software analysis; static analysis; dynamic analysis; software quality

1 引 言

软件是一种十分特殊的人工制品:它是人类"智力活动"的产物,是对客观事物的虚拟反映,是知识的固化与凝练.尽管软件迄今已有 50 多年的发展历史,但目前人们对软件的许多认识还十分有限.例如:对任何一个给定的软件,我们能否完全了解它的特性?软件分析就是一个以软件特性为关注点的研究领域."分析",通俗来说,是以某种方式将复杂对

收稿日期:2009-08-22.本课题得到国家"九七三"重点基础研究发展规划项目基金(2009CB320703)、国家自然科学基金委员会创新研究群体研究科学基金项目(60821003)、国家自然科学基金(60725206)、国家"八六三"高技术研究发展计划项目基金(2006AA01Z175)资助.**梅 宏**,男,1963 年生,博士,教授,博士生导师,研究兴趣包括软件工程、软件复用与软件构件技术、软件生产线技术、编程语言.**王千祥**,男,1970 年生,博士,教授,研究兴趣包括软件分析与评价、软件中间件.E-mail: wqx@pku.edu.cn.**张 路**,1973 年生,博士,教授,研究兴趣包括软件测试、软件维护.**王 戟**,男,1969 年生,博士,教授,博士生导师,研究兴趣包括高可信软件、软件工程与分布计算.

象分解为更小的部分,以更好地理解该对象的过程.分析技术很早就被应用于数学、逻辑等方面的研究,近代以来逐步被更多的学科(例如化学、物理等)所大量采用.软件作为一个新发展起来的学科,在研究过程中引入分析技术是十分自然的.目前软件生命周期中的许多活动(分析、设计、实现、测试、部署、维护等)都离不开分析技术.然而,软件分析的能力是有限的:对于任何一个有一定规模的软件,希望获得关于它的完备描述通常是不现实的[1].特别是,对于自动分析而言,许多问题是不可判定的.其中最典型的例子是停机不可判定问题:不存在一个这样的算法,对于任意的图灵机以及任意的输入,可以判断该图灵机是否停机[2].但从软件分析这么多年所取得的进展可以看出,尽管软件分析的能力有限,它仍然是软件领域十分有用的技术:将程序从高级语言向机器语言的翻译过程需要分析,判断一个程序是否符合需求规约需要分析技术,想了解程序是否存在安全漏洞需要分析技术,维护过程更是需要大量的分析技术,等等.

本文将软件分析定义为"对软件进行人工或者自动分析,以验证、确认或发现软件性质(或者规约、约束)的过程或活动".下面对上述定义中几个术语进行解释.首先是"软件":软件最初主要是指程序,后来逐步扩大到文档等其它形态软件制品.软件分析也从程序分析发展到了更大的范围,例如对文档(含需求规约、设计文档、代码注释等)的分析、对运行程序的分析,等等."自动"也是很重要的概念:软件分析的历史几乎与软件的历史一样长,自从有了软件就有了软件分析.最初的分析主要是人工进行的,但人工分析往往需要花费大量的时间与精力,因此,后来人们越来越多地关注自动分析.其中,编译技术的发展大大带动了软件的自动分析技术,目前的许多分析技术都可以在编译技术中找到基本雏形.所谓"验证"(verification),是要回答"软件制品是否与软件需求规约一致"的问题,而"确认"(validation)则要回答"软件的特性是否符合用户需求"的问题.在英文中,人们经常用"Do the thing right"来解释"验证",而用"Do the right thing"来解释"确认"."发现"(discover)是指在没有事先设定软件某个性质的前提下,通过分析发现软件的某种性质.之所以强调"性质",是因为分析的结果通常表示为软件是否符合或者具有某种性质(或者规约、约束),而这种性质不是软件本身自明的.

在本文讨论的软件分析中,分析对象仅限于软件制品,不涉及对软件过程、软件人员与软件组织等的分析.目前与软件分析相关的综述性文献中,多数只对软件分析的一个子集进行比较深入的介绍.例如文献[3]集中在对源代码分析的介绍;文献[4]主要介绍形式化的分析方法;文献[5]着重从语义的角度介绍程序分析;文献[6]集中在模型为中心的程序分析上.本文在这些工作的基础之上,尝试对软件分析涉及的主要方法进行尽可能全面的总结、分类.另外,考虑到近年来软件质量为人们所热切关注,本文在介绍分析技术之外,特别关注那些与质量相关的分析技术,并结合不同的软件质量属性,探讨不同的质量属性适合运用什么类型的分析技术.最后,文章结合软件形态、软件运行环境等几个驱动力对软件分析技术的发展趋势进行展望.

2 软件分析技术

软件分析通常是另外一个更大的软件生命周期活动(例如:开发、维护、复用等)的一部分,是实施这些过程的一个重要环节:(1)在开发阶段,对正在开发的软件进行分析,以快速地开发出高质量的软件,例如:了解开发进展、预测开发行为、消除软件缺陷、程序变换等等;(2)在维护阶段,对已经开发、部署、运行的某个软件进行分析,以准确地理解软件、有效地维护该软件,从而使软件提供更好的服务;(3)在复用阶段,对以前开发的软件进行分析,以复用其中有价值的成分.上述各过程差异较大,需要的分析技术也多种多样.这导致目前的软件分析内容十分丰富,且相互之间的界限也不甚清晰:有些分析过程需要另外某个或某些分析的支持;有些分析针对具体的性质开展,而有些分析方法则可以支持多种性质的分析.

2.1 软件分析分类

为了对软件分析有个比较全面的了解,对其进行合理的分类是十分必要的.对软件分析进行分类的维度有很多.其中,分析对象是最重要的准则之一,即软件分析是对什么制品进行分析?从大的方面看,软件分析可以对代码进行,也可以对模型(需求规约、设计模型、体系结构等)、文档甚至注释进行.代码可以进一步分为源码与目标码,目标代码又具有静态与运行态两种存在方式,而运行态的代码又可以进一步区分为离线的运行与在线的运行.除分析对象维度之外,还可以从方法学(结构化软件、面向对象软件、面向构件软件等)、并行程度(串行软

件、并行软件)等其它维度划分软件分析的内容.

本文首先以分析过程"是否需要运行软件"为准则,将软件分析技术划分为静态分析技术与动态分析技术两大类,然后又在每一大类技术下面做进一步的划分.图 1 是综合考虑静态、动态分析技术给出的一个分析过程示意图.

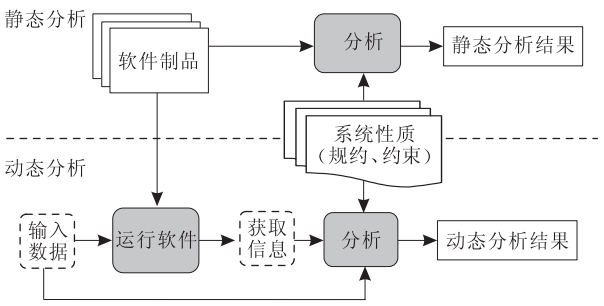

图 1　静态分析与动态分析的基本过程

2.2　静态分析

静态分析是指在不运行软件前提下进行的分析过程.静态分析的对象一般是程序源代码,也可以是目标码(例如 JAVA 的 byte code),甚至可以是设计模型等形态的制品.静态代码分析主要可以应用于如下几个过程:① 查找缺陷,以消除软件中存在的缺陷;② 程序转换,以实施编译、优化等过程;③ 后期的演化与维护;④ 动态分析,等等.

根据各种分析方法使用的广泛程度以及分析方法的相近性,本文将主要的代码静态分析划分为 4 类:基本分析、基于形式化方法的分析、指向分析与其它辅助分析(见图 2).其中,基本分析是一些常见的分析,例如语法分析、类型分析、控制流分析、数据流分析等,是多数编译器都包含的分析过程(词法分析由于相对简单而没有引入);而基于形式化方法的分析则在分析过程中大量采用一些数学上比较成熟的形式化方法,以获得关于代码的一些更精确或者更广泛的性质.指向分析多数与指针密切相关,由于在静态分析中长期受到较多的关注,因此单独作为一类.其它辅助分析则包含了一些单独分析的目的性不是很强、但可以为前面几类分析提供支持的一些分析方法.需要指出的是,这不是一个严格的分类,而仅仅是为了便于人们比较全面地了解静态分析,对一些主要的、具有共性的静态分析进行归类而得到的一个结果.

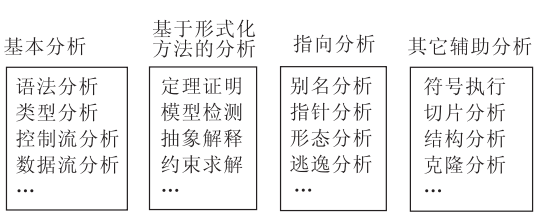

图 2　主要的静态分析技术

2.2.1　基本分析

(1)语法分析(syntax analysis).语法分析是按具体编程语言的语法规则分析和处理词法分析程序产生的结果并生成语法分析树的过程.这个过程可以判断程序在结构上是否与预先定义的 BNF 范式相一致,即程序中是否存在语法错误.程序的 BNF 范式一般由上下文无关文法描述.支持语法分析的主要技术包括算符优先分析法(自底向上)、递归下降分析法(自顶向下)和 LR 分析法(自左至右、自底向上)等.语法分析是编译过程中的重要步骤,也是多数其它分析的基础:如果一个程序连语法分析都没有通过,则对其进行其它的分析往往没有意义.

(2)类型分析(type analysis).类型分析主要是指类型检查(type checking).类型检查的目的是分析程序中是否存在类型错误.类型错误通常是指违反类型约束的操作,例如让两个字符串相乘,数组的越界访问,等等.类型检查通常是静态进行的,但也可以动态进行.编译时刻进行的类型检查是静态检查.对类型分析的支持程度是划分编程语言种类的准则之一:对于一种编程语言,如果它的所有表达式类型可以通过静态分析确定下来,进而消除类型错误,则这个语言是静态类型语言(也是强类型语言).利用静态类型语言开发出的程序可以在运行程序之前消除许多错误,因此程序质量的保障相对容易一些(但表达的灵活性弱一些).

(3)控制流分析(control flow analysis).控制流分析的目标是得到程序的一个控制流图(control flow graph).控制流图是对程序执行时可能经过的所有路径的图形化表示.通过根据不同语句之间的关系,特别是考虑由"条件转移"、"循环"等引入的分支关系,对过程内的一些语句进行合并,可以得到关于程序结构的一些结果.一个控制流图是一个有向图:图中的结点对应于程序中经过合并的基本语句块,图中的边对应于可能的分支方向,例如条件转移、循环等等,这些都是分析程序行为的重要信息.

(4)数据流分析(data flow analysis).数据流分析试图确定在程序的某一点(语句)关于各个变量的使用或者可能取值情况.数据流分析一般从程序的一个控制流图开始.数据流分析主要有前向分析(forward analysis)、后向分析(backward analysis)两种方法.前向分析的一个例子是可达定义(reaching definitions).它计算对于程序的每一点,可能

到达该点的定义的集合.后向分析的一个例子是活跃变量(live variables).它计算对于程序的每一点,程序后面的语句可能读取且没有再次修改的变量.这个结果对于消除死代码(dead code)很有用:如果一个变量在某个阶段被定义后,后面的语句一直不会用到这个定义,那么这个定义就是死代码,应该从程序中删除.基于格(lattice)与不动点(fixpoint)理论的数据流分析是目前被广泛使用的技术:首先对控制流图中的每个节点建立一个数据流等式(equations),并根据分析目标构造一个具有有限高度的格 L,然后不断重复计算每个节点的输出,直到达到格的一个不动点.许多编译器为了进行编译优化而引入了数据流分析技术.

由于上述4种基本分析是多数编译器包含的内容,因此很早就得到了较深入的研究[7].这些分析过程还有一个共同特点是分析的输入仅仅是软件代码,不需要提供图1中的"系统性质".或者说,基本分析技术所需要的"系统性质"都是最基本的性质.例如语法分析对应的"系统性质"就是编程语言的BNF范式,类型分析对应的"系统性质"是预先定义的类型约束,数据流分析对应的"系统性质"是编程语言的基本约定,等等.而下面要讲到的形式化分析、指向分析则通常要事先提供待验证的性质,例如,某个变量的取值是否在某个范围内,某两个变量名是否指向相同的内存实例,等等.

2.2.2 基于形式化方法的分析

为了提高分析的准确度,获取关于程序的更多性质,许多研究人员借用形式化方法来扩展基本分析技术.代表性技术有模型检验、定理证明、约束求解、抽象解释等.

(1)模型检验(model checking).模型检验用状态迁移系统表示系统的行为,用模态/时序逻辑公式描述系统的性质,然后用数学问题"状态迁移系统是否是该逻辑公式的一个模型"来判定"系统是否具有所期望的性质"[8].模型检验虽然在检查硬件设计错误方面简单明了且自动化程度高,然而被应用在软件程序分析与验证时却存在着难以解决的状态空间爆炸问题.另外,由于模型检验所针对的检查对象是模型而非程序本身,任何在将程序向模型转化的过程中所使用的抽象技术以及转化工作都有可能使模型与程序不一致或者存在偏差,从而导致最终的检查结果无法准确反映实际程序中存在的错误情况.更多关于模型检验的深入讨论可以参见文献[8].支持模型检验的代表性软件分析工具为SLAM[9]、MOPS[10]、Bandera[11]和Java PathFinder2[12].

(2)定理证明(theorem proving).自动定理证明通过将验证问题转换为数学上的定理证明问题来判断待分析程序是否满足指定属性[3],是众多分析方法中最复杂、最准确的方法.然而,一阶逻辑是半可判定的,理论分析结果表明,机械化的定理证明过程并不保证停机.另外,为了获取指定的属性以实现有效的证明,这些工具都要求程序员通过向源程序中添加特殊形式的注释来描述程序的前置条件、后置条件以及循环不变量.这无疑增加了程序员的工作量,也导致该方法难以广泛应用于大型应用程序.使用定理证明的代表性软件分析工具为ESC[13]和ESC/Java[14].

(3)约束求解(constraint solving).基于约束求解的程序分析技术将程序代码转化为一组约束,并通过约束求解器获得满足约束的解[15].早期的研究表明,面向路径的测试数据生成可以很好地归结为约束求解问题.后来,学者们又发现程序中不变式的分析也可以归结为约束求解问题.最新的研究表明,许多其的程序分析问题也可以归结为约束求解问题:由于从程序获得的约束通常采用一阶或二阶的形式表示,可以进一步将其转换成约束求解器可处理的形式.支持约束求解的代表性软件分析工具为SAT/SMT Solver[16].

(4)抽象解释(abstract interpretation).程序的抽象解释就是使用抽象对象域上的计算逼近程序指称的对象域上的计算,使得程序抽象执行的结果能够反映出程序真实运行的部分信息.抽象解释本质上是在计算效率和计算精度之间取得均衡,以损失计算精度求得计算可行性,再通过迭代计算增强计算精度的一种抽象逼近方法.通过不断迭代,抽象解释最终为程序建立一个抽象模型.如果抽象模型中不存在错误,就证明其对应的源程序中也不存在错误.具有抽象解释分析功能的代表性分析工具为Proverif[16]和ASTREE[18].

形式化支持的分析技术在分析软件的某个性质时,需要首先对该性质进行形式化的描述,然后将这个描述与软件制品一起作为输入提供给分析工具.其中,模型检验首先被用于对软件的模型进行分析,后来有研究人员通过从代码中提取模型,然后将模型检验技术应用于代码分析.定理证明主要对静态代码比较适用.约束求解多用于输入数据的生成.基于抽象解释理论的形式化方法是对大规模软件、硬件系统进行自动化分析与验证的有效途径之一,已

经被广泛地应用于大型软件与硬件系统的验证研究中.

2.2.3　指向分析

(1)别名分析(alias analysis).别名分析主要用于确定程序中不同的内存引用(reference)是否指向内存的相同区域.在编译过程中,这可以帮助判断一个语句将影响什么变量.例如,考虑如下的代码:…; $p.foo=1$; $q.foo=2$; $i=p.foo+3$; ….如果 p 和 q 不是别名,那么 $i=p.foo+3$;等价于 $i=4$;如果 p 和 q 是别名,那么 $i=p.foo+3$;等价于 $i=5$;这样就可以对代码进行等价优化.别名分析又可以分为基于类型的分析与基于流的分析.前者主要用于类型安全(type safe)的语言,后者则主要用于含有大量引用与类型转换的语言[19].

(2)指针分析(pointer analysis).指针分析试图确定一个指针到底指向哪些对象或者存储位置,尤其是,在某个语句处是否可能为空.由于受到可判定性问题的限制,加上分析过程中时间、存储等的限制,多数的指针分析方法都在分析过程中进行"近似"或者"简化",并导致分析结果精确性不够.实际上,上面的别名分析与下面的形态分析、逃逸分析都与指针分析密切相关.

(3)形态分析(shape analysis).形态分析主要用于发现或者验证程序中动态分配结构的性质.对于一个具体的程序,形态分析将为其构造一个形态图(shape graph),用于列出每个指针可能指向的目标以及目标之间的关系.形态分析可以认为是指针分析的一种,但比一般的指针分析精确:形态分析可以确定一个小一些但是更精确的指向集合.例如在Java 程序中,可用来保证一个排序算法正确地对列表进行排序;在 C 程序中,可以用来分析一个内存是否被正确地释放.尽管形态分析很强大,但往往需要花费较多的时间[20].

(4)逃逸分析(Escape Analysis).逃逸分析计算变量的可达边界.对于一个方法 m 中的一个变量,如果变量是在调用方法 m 时创建的,但在 m 的生命周期之外可以获得该变量,我们就说这个变量逃逸了方法 m.类似地,如果在 t 之外的一个点能通过一个引用访问到该变量,则说一个变量逃逸了一个线程 t,逃逸分析传统上被用于查找一些变量,它们只存在于为它们分配内存的方法或线程的生命周期内:前者允许变量在运行时的栈(stack)上,而不是堆(heap)上分配内存,这样就可以降低堆的碎片与垃圾回收负载.后者被用于进行优化,以避免高成本的异步操作.逃逸分析检查引用的赋值与使用(assignments and uses)以计算每个变量的逃逸状态.每个变量可以被赋予 3 个可能逃逸状态(全局逃逸、参数逃逸或者捕获)中的一个.当一个变量是全局可达的(例如被赋值给了一个静态域)时,这个变量被标记为全局逃逸;如果变量是通过参数或者被返回给调用者方法,它被标记为参数逃逸;一个不逃逸的变量被标记为捕获.逃逸分析主要用于效率分析[21].

与基本分析技术相比,指向分析通常与应用程序的某个特定性质密切相关.这类分析一般是以基本分析(尤其是数据流分析)为主要分析框架,为了提高分析精度而提出的技术,且分别结合了编程语言的不同特点.例如别名分析利用的变量引用、形态分析利用的指针等等.这也导致了这些分析技术分别适合由不同编程语言实现的程序.另外,这些分析技术尽管可以自动进行,但在分析之前通常需要较多的人工介入,例如,指定对哪些变量进行分析、提供对什么性质进行分析等等.

2.2.4　其它辅助分析

(1)符号执行(symbolic execution).符号执行通过使用抽象的符号表示程序中变量的值来模拟程序的执行[22-23].其特点在于通过跟踪被模拟的各条执行路径上变量的实际取值,把分析工作局限在实际可达的路径上,从而使得到的结果更贴近程序实际执行情况,并为程序员提供更为准确的、与检出的缺陷相关的上下文信息.但是由于需要穷举各条可能执行的路径,该技术需要处理的工作量随着程序规模的增大而呈指数级别增长.虽然符号执行方法可以被应用于大型程序的分析,其分析结果的可靠性仍依赖于所允许的分析时间和对路径及其数目的选择等方面.

(2)切片分析(slicing analysis).切片分析用于从源程序中抽取对程序中兴趣点上的特定变量有影响的语句和谓词,组成新的程序(称作切片),然后通过分析切片来分析源程序的行为[24].计算程序切片的方法主要有两种:根据数据流方程计算和根据依赖图关系计算.切片分析技术已被广泛应用于程序分析、理解、调试、测试、软件维护等过程[25-26].

(3)结构分析(structure analysis).结构分析的目标是获得程序的调用关系图(call graph),以展示程序中各个函数之间的调用关系.把程序中每个函数当作一个节点,再分析每个函数调用了哪些其它函数,并在存在调用关系的函数间建立一条边,就可

以得到调用关系图.调用关系图通常用于辅助开发人员理解程序.对于面向对象程序,程序的结构分析还包括从程序中获取类图(class diagram)等.除了一些编译器支持结构分析外,目前软件开发过程中的一些工具,例如 IBM Rational Rose 等也能够对已开发出的代码进行结构分析.

(4)克隆分析(clone analysis).代码克隆(code clone)是指软件开发中由于复制、粘贴引起的重复代码现象.研究指出,一般商业软件中存在 5%~20%的重复代码[3].由于克隆代码的普遍性以及克隆代码对代码质量的重要影响,代码克隆相关研究是静态代码分析领域近年来一个十分活跃的研究分支.主要研究内容包括:克隆代码检测、由代码克隆引起的代码缺陷诊断、通过代码重构来减少代码克隆、克隆代码跟踪、基于代码克隆的源代码演化分析等等.代码克隆分析有十分丰富的实际应用价值,比如缺陷诊断、重构、代码理解、源代码演化分析和代码剽窃检查等等.克隆代码可以分为如下 4 类:①除空格、回车以及注解之外完全相同的代码片段;②除空格、回车、注解以及变量名及常量值替换外,语法结构完全相同的代码片段;③除空格、回车、注解以及变量名及常量值替换外,语法结构基本相同,但含有少量语句的增加、删除或修改的代码片段;④两段或多段代码具有相同或相似的功能,或者说相似的输入、输出条件.其中,最后一类比较特殊,是语义(功能)相似性,其它 3 类都是文本相似性[22,27-38].

2.3 动态分析

动态分析是通过运行具体程序并获取程序的输出或者内部状态等信息来验证或者发现软件性质的过程.与静态分析相比,动态分析具有如下几方面特点:(1)需要运行系统,因此通常要向系统输入具体的数据;(2)由于有具体的数据,因此分析结果更精确,但同时只是对于特定输入情况精确,对于其它输入的情况则不能保证.

本文从运行信息的获得途径与获得时机两个方面对动态分析进行介绍.在信息获得的时机上,又根据软件是否已经上线投入使用将软件的动态分析划分为两大类:离线动态测试/验证(Offline Dynamic Testing/Verification)与在线监测(Online Monitoring).所谓离线动态测试/验证,是指在系统还没有正式上线时对软件进行运行、分析,分析过程中可以随意输入数据,并尽量模拟实际用户的操作.所谓在线监测,是指在系统已经上线后对软件系统进行分析,监测过程中一般不能随意输入数据,所有数据都是真实的.离线动态测试/验证、在线监测与运行信息获取之间的基本关系见图 3.

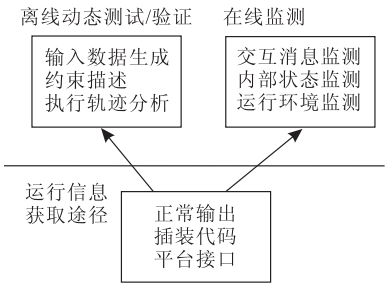

图 3 动态分析涉及的主要技术

2.3.1 运行信息的获取途径

(1)从程序的正常输出中获取信息.

每个程序在运行过程中都会产生许多输出信息.有些输出是程序运行中间或者结束时输出的正常结果,有些是一些提示信息,还有一些是日志信息.通过将最终得到的实际输出结果与事先设定的期望输出结果进行对比、分析,就可以得到关于软件的有价值信息.

(2)通过插装代码获取信息.

仅仅通过观察程序的正常输出对于了解软件的运行信息往往是不够的.例如,软件运行过程中内部变量的状态信息、某个特定类型的实例信息、模块之间的交互信息等等.这些信息对于发现缺陷以及定位缺陷特别重要.获得这些内部信息的自然方式是在软件中插装监测代码(monitoring code),通过这些监测代码就可以获得相应的信息.主要的监测代码插装方法可以分为如下 3 类:

①源码插装.这是最自然的插装方式,即在编写应用系统时,在需要监测的地方直接加上监测代码,例如,增加输出信息语句、增加日志语句等等.AOP(Aspect Oriented Programming)技术出现之后,人们发现 AOP 可以被很好地用于代码插装,以有效地分离系统的业务逻辑与监测逻辑[39].

②静态目标码插装.近年来字节码插装技术在 Java 社区中十分流行.字节码插装可以在静态直接更改中间代码文件(例如 Java 的.class 文件)或在装载时刻进行字节码插装.字节码插装所具有的执行效率高、插装点灵活、应用范围广等特点,使其被广泛应用于 AOP 等研究领域,并陆续出现了 BCEL①、

① Apache, BCEL(ByteCode Engineering Library). http://jakarta.apache.org/bcel.

Javassist①、ASM② 等多种字节码操纵工具.

③ 基于截取器(Interceptor)的获取方式.截取器处于调用者和被调用者之间,可以截获二者之间传递的消息,从而完成一些特定的处理工作.由于这种获取方式不需要直接修改目标程序,代码侵入性较弱,甚至可以在运行阶段部署,因此得到了越来越广泛的使用.Tomcat 服务器、EJB3 规范、Spring 框架中都有截取器的实现[40].

(3)通过平台接口获取信息.

向目标系统插装监测代码可以很方便地获得内部信息.如果底层的运行平台(操作系统、JVM、中间件、或者数据库管理系统等)能提供很好的支持,则许多信息获取起来就更加方便.例如,许多研究人员通过开发特殊的 JAVA 虚拟机来获取所需要的监测信息.JPF[41]、QVM[42] 等就是典型代表.目前标准的 JVM 本身也提供了许多供调试、监测的接口,例如 JVMTI③ 等等.

2.3.2　离线动态测试/验证

离线的动态测试与动态验证都需要在离线的情况下运行程序,并获取、分析运行信息,因此二者有比较密切的联系.不仅如此,为发现更多的软件缺陷,离线动态验证与动态测试都需要仔细准备输入数据.另外,如果将程序的输出与输入之间的关系看作一种约束需求,或者在分析测试结果时也收集日志等内部信息的话,二者就更接近了.从不同之处看,离线动态验证一般比测试收集更多的内部信息、关注更多的约束需求,而且往往利用一些轻权(lightweight)的形式化方法来描述这些需求.从研究内容看,离线动态测试/验证主要关注 3 个方面的内容:输入数据生成、约束描述、运行轨迹分析.

(1)输入数据生成.为了尽可能多地发现潜在的缺陷,在运行程序之前通常需要首先静态地分析目标程序,根据验证目标(什么功能、什么约束等等)辅助用户生成和选择输入数据.

(2)约束描述.利用形式化方法描述软件约束,以便于分析能够自动进行.目前多数动态验证研究人员利用线性时序逻辑(Linear Temporal Logic,LTL)来描述.

(3)运行轨迹分析.程序运行过程中产生的内部、外部数据可能是大量的,需要测试/验证目标对数据进行必要的过滤,然后分析这些数据以推断程序的执行轨迹,并进一步判断程序的执行是否遵循程序的约束.

2.3.3　在线监测

与离线动态验证相比,在线监测有几个特点:(1)系统输入由实际的真实用户与系统拥有者共同决定;(2)系统拥有者的输入是受限制的,一些在上线之前可以做的实验(例如:压力测试、安全性测试等)此时不能随意实施,否则就会威胁到系统正常的服务质量;(3)监测代码往往就是应用系统的组成部分.这使得在线监测与前面介绍的各种分析都非常的不同:前面介绍的分析技术一般都由特定的分析工具支持,分析工具是一种外部辅助工具,在系统上线后,分析工具就与系统分离了.而在线监测则可能一直伴随系统的服务过程,并且可能是在系统上线之后,在不同的维护阶段增加上去的.有研究人员因此提出了面向监测的编程(MOP)[43].

在分析机制上,在线监测的分析可以采用内联模式(inline)或者外联模式(outline).在内联监测模式中,监测代码(monitor)与被监测程序运行在相同的空间中,因此执行效率相对要高,且发现异常后响应要快.在外联监测模式中,监测代码运行在独立的运行空间中,比如另外一台机器或者 CPU.外联模式效率要低一些,但可以对多项监测内容进行综合处理,因此可以做更深入的分析.

对于在线系统,如果要监测它,就一定会影响它.如果影响过大,就可能给正常的服务过程带来负面作用.因此,衡量在线监测技术的一个重要指标是监测开销,尤其是运行开销.许多监测的时间开销甚至高于程序自身的运行开销.由于在线监测一定会对监测对象的运行产生影响,因此监测的结果也一定是被影响后系统的表现,而不是被监测对象自身的表现.针对这个现象,有些研究人员参照物理学中的测不准原理提出了软件的测不准原理[44].

需要特别注意的是,在系统上线之后,有价值的监测通常不仅仅针对软件自身,而是针对由软件、硬件组成的服务系统,甚至包括与服务系统交互的环境(用例图中的 Actor)[45].例如:客户程序的调用序列、系统的响应时间等等.在这个意义上,在线监测不仅仅是努力发现软件的缺陷,而且关注服务过程的潜在问题,例如:响应时间是否足够小?是否发现可疑的攻击行为?因此,随着 Web 服务技术、软件作为服务(Software as a Service,SaaS)的推广,在线监测正受到越来越多研究人员的关注[46].另外,

① JBOSS, Javaassist. http://www.jboss.org/javaassist.
② OW2, ASM. http://asm.ow2.org.
③ SUN, JVMTM Tool Interface (JVM TI). http://java.sun.com/j2se/1.5.0/docs/guide/jvmti/.

多核处理器的发展,也为监测提供了更多的途径:由于业务逻辑与监测逻辑相对分离,完全可以由不同的核分别进行业务处理与监测处理.

(1)对软件内部的监测.对软件内部的监测方法与运行时验证的方法比较接近,上面提到的各种插装技术对于在线监测都适用.不仅如此,对于在线系统,还可以利用在线升级(upgrading)的技术[47],或者动态编织(weaving)技术[48]等将监测代码"在线"地部署到目标系统中,以加强监测的覆盖面;或者从系统中移除监测代码,以降低监测开销.除此之外,还有许多对实例的监测,例如负责客户连接对象的实例数目、负责数据库连接的实例数目等等.

(2)对外部交互的监测.主要监测对象包括:来自客户程序的请求消息顺序是否正确?参数值是否在允许的范围内?请求者的权限是否够(与安全相关)?应答消息的参数值是否在允许的范围内?从收到请求消息到发出应答消息的响应时间是否符合要求?管理人员的操作是否合法?等等.

(3)对运行环境的监测.对运行环境的监测主要体现为对底层资源的监测,通常是独立于具体应用的监测内容,例如 CPU 的使用情况、内存使用情况、网络带宽等等.对于越来越多的嵌入式系统,由于各种资源都有较大的限制,监测就显得更加重要.

2.4 分析技术的评价

面对种类众多的软件分析技术,人们通常不仅希望能了解它们,还希望对它们进行评价、比较,以在其中选择选择合适自己当前任务的分析技术.实际上,由于这些技术错综复杂,对它们的评价本身也是一个十分值得研究的题目.目前人们比较关注的评价准则有:误报率(False Positive Rate)、漏报率(False Negative Rate)、精度(Precision)、速度(Speed),等等.

误报率与漏报率.误报是指当软件不存在某个缺陷时,分析工具报告软件可能存在某个缺陷.大量的误报将导致大量的人工分析工作:人们必须手工判断系统是否真的存在某个缺陷.漏报是指软件中存在某个缺陷,但分析工具没有报告这个缺陷.将报告结果与缺陷的实际情况进行对比,就可以得到具体的量化指标:误报率与漏报率.误报率与漏报率是目前评价分析工具的两个最重要的指标:一个工具的误报率与漏报率越小,说明这个分析工具越好.但实际情况往往是:有的方法在降低误报率方面很有效,但往往同时增加了漏报率;而有的方法在降低漏报率方面很有效,但却抬高了误报率[3].误报率与漏报率的反面说法分别是查准率(Precision)与查全率(Recall).一般来说,静态分析可以比较全面地考虑执行路径,因此可以比动态分析发现更多的缺陷,漏报率比动态分析低;但动态分析由于获取了具体的运行信息,因此报出的缺陷一般更为准确,误报率比静态分析低.

精度与速度.为了提高分析的精度,即降低误报率与漏报率,实际的静态分析工具往往综合运用多种分析技术,并需要做一些较深入的分析,这自然意味着更长的分析时间.因此,分析的精度与分析的速度往往也是一对不可兼得的矛盾体,必须在二者之间进行折中.另外,动态分析由于往往一次只关注少部分的软件性质,因此精度可以更高一些,而且受程序规模的限制较小;而静态分析在程序运行之前就可以实施,因此可以更早地发现问题.

3 软件分析与质量保障

软件分析的一个主要应用是保障软件质量:通过分析某个软件,查找出其中包含的软件缺陷,就可以让开发人员修改软件,将缺陷修复,从而提高软件质量.ISO9126 提出了一个两层、六类的质量模型,包括:(1)功能性(含适合性、准确性、互操作性、保密安全性);(2)可靠性(含成熟性、容错性、易恢复性);(3)易用性分析(含易理解性、易学性、易操作性、吸引性);(4)效率(含时间特性、资源利用性);(5)易维护性(含易分析性、易改变性、稳定性、易测试性);(6)可移植性(含适应性、易安装性、共存性、易替换性).

本文以 ISO9126 的分类为基础,从软件分析的角度出发,结合上面的质量属性,归纳出 5 种相对并列的软件质量属性:正确性(Correctness)、健壮性(Robustness)、安全性(Security)、效率(Efficiency)、易维护性(Maintenance).当然,这 5 种属性之间不是完全正交的,它们之间存在着一些不同形式的关联.本节主要介绍如何利用上一节中提到的一些分析技术对这些属性进行分析.尤其是,如何利用这些技术发现相应的质量问题.

3.1 软件正确性

"正确性"主要指程序的运行结果是否与预期值相同.如果不相同,则视为结果不正确,该程序包含一个"正确性"错误.正确性的考虑比较单纯,主要是从输入到输出这个函数映射的角度看待计算过程,特别关心输出结果的正确与否,而不考虑输入数据

的具体来源与错误输出结果的后果如何.与正确性相对的是健壮性与安全性:正确性考虑的是软件是否按照预先的设定执行,健壮性与安全性则考虑软件是否在一定条件下执行设定之外的事情.

3.2.1　动态测试/验证

动态测试/验证是发现正确性缺陷最有效的手段.对于任何一个稍具规模的软件而言,穷举所有可能输入的测试/验证都是不现实的.因此,对于发现与正确性有关的缺陷而言,测试/验证的关键在于生成有较强揭示错误能力的输入数据,并通过程序的执行最终检测与正确性相关的缺陷.基于程序分析的数据输入生成技术大体可以分为面向路径的测试用例生成[49]和面向目标的测试用例生成[50].面向路径的测试数据生成是指给定程序的一条执行路径,生成一个恰好执行该条路经的测试用例.面向路径的测试用例生成可以转化为约束求解问题.面向目标的测试用例生成是指,给定测试的某个目标(比如语句覆盖),生成一组可以达到该目标的测试用例.面向目标的测试用例生成通常以面向路径的测试用例生成为基础,其基本思路是将面向目标的测试用例生成转化成面向路径的测试用例生成甚至直接转化成约束求解问题.多数情况下,一种测试用例生成技术并不仅针对与正确性有关的缺陷,同时也会兼顾与健壮性有关的缺陷.

3.2.2　静态分析

静态分析也是发现与正确性有关缺陷的重要途径.静态分析的基本思路是建立一些与正确性有关的形式化规约,然后检查软件是否满足这些规约.静态分析可以不针对软件的特定输入发现一类缺陷,因此在很大程度上可以与测试互为补充.

3.2　软件健壮性

"健壮性"主要是指系统是否能控制软件内外客观不良事件的发生,以保持系统的基本服务质量.例如,是否会发生因死锁而导致系统不工作、是否因为内存泄漏而导致系统崩溃、是否因产生数据竞争而导致系统出现错误状态等等.健壮性与正确性之间存在一定的关联:首先,不正确的输出有可能影响包含软件的系统整体上的健壮性;其次,因系统崩溃等原因导致"得不到结果"有时也被认为是"不正确"的一种特殊表现.与"健壮性"比较接近的一个术语是"可靠性"(Reliability).后者目前被认为主要是指在规定运行环境下、规定时间内,软件无失效运行的概率,并且是评价软件正确性的一种重要途径[51].由于这个概念与正确性在内涵上重合较大,因此本文采用"健壮性"这个与正确性在内涵上重合较小的概念作为从分析角度上划分的质量属性之一.

3.2.1　动态测试/验证

动态测试/验证是发现健壮性的一个重要手段.与针对软件正确性的测试类似,这方面软件分析的重点也在于输入数据(测试用例)生成.不同的是,针对软件健壮性的生成的目标是生成不正确的输入,并检查这些输入是否会造成非期望的结果.

3.2.2　模型检验

将模型检验应用于健壮性分析的最典型例子是死锁(deadlock)检验.死锁[52]是指多个进程(线程)因相互等待而不能完成计算任务.死锁通常是进程(线程)在竞争资源时产生的:每个进程(线程)拥有一些资源同时等待其它进程(线程)拥有的某些资源,每个进程(线程)都因不能拥有所需的所有资源而无法继续进展.模型检验可以对软件可能的状态进行穷举、分析,从而判断软件是否可能进入死锁状态.

3.2.3　指向分析

软件的许多健壮性问题与指针相关.因此,指向类分析对于健壮性保障十分重要.例如内存泄漏(memory leak)问题.内存泄漏是指因内存使用后没有释放而引起的可用内存的异常消耗[53].严重的内存泄漏会导致软件系统因可用内存消耗殆尽而崩溃.静态的内存泄露检测主要检查软件中每个内存申请语句是否有对应的内存释放语句以及软件是否存在内存申请语句和内存释放语句不匹配的路径.动态的内存泄露检测主要分析内存的使用情况,检查内存的使用上是否存在一些与内存泄露以有关的现象.例如,面向对象程序中的内存泄漏往往会导致某些类的对象的数目会随软件的执行而不断增长,跟踪每个类的对象的数目可以辅助内存泄漏的检测.

3.3　软件安全性

"安全性"主要指软件是否存在安全漏洞,是否会由于人为攻击,造成信息泄露(保密性)、数据损失(完整性)或者系统不能正常工作(可用性)等不良后果.安全性与健壮性之间的主要区别在于:导致系统不安全的因素是人为因素,而导致系统不健壮的因素是软件本身.另外,二者之间也存在关联:许多系统因为健壮性存在缺陷(例如:字符串溢出)而导致安全性缺陷(恶意注入).

3.3.1　静态代码漏洞查找

(1)输入验证.这里的输入不仅仅是用户的输入,还包括配置文件、从数据库检索出的数据、命令

行参数、环境变量、网络消息等等.在一个软件系统中,接受这些输入的接口集通常被称为应用程序的攻击面.SQL 注入、脚本注入、跨站点攻击等都是这类攻击.通过对应用程序进行分析,可以自动发现应用系统的可信边界,还可以发现边界上的点是否因为缺乏有效的验证而存在安全漏洞.

(2)溢出攻击.由于缓冲区溢出很容易被攻击者用来重写内存中的数据,并非法获取一些权限,因此许多恶意者利用缓冲区溢出进行攻击.缓冲区溢出通常是由一些特殊的函数导致的.例如 C 语言中的 gets(),scaf(),strcpy(),sprintf()等.静态分析可以发挥的地方包括:分析是否使用了不必要的高风险函数、是否存在多次内存释放操作等.其中,内存释放等问题往往涉及别名分析.

(3)隐私信息.多数程序中存在需要隐藏的内容.不严谨的编程人员很容易在程序总不经意地泄露相关的信息,导致系统被攻击.例如:将私密数据放到日志中,程序中硬编码了密码信息,向浏览器返回过于详细的出错信息,随机数的产生规则过于简单,选择的加密算法不够安全等等.静态分析可以帮助编程人员发现这些漏洞.

3.3.2 安全性测试

安全性测试通过运行软件、模拟恶意用户的输入来分析系统的安全性.一些主要考虑的问题有两类:权限相关的安全性与网络相关的安全性.权限相关的安全性包括:是否明确地区分系统中不同用户权限?系统中会不会出现用户冲突?系统会不会因用户的权限的改变造成混乱?是否可以通过绝对途径登陆系统?等等.与网络相关的安全性包括:系统的补丁是否打上?模拟非授权攻击,看防护系统是否坚固?系统中是否存在某些安全漏洞?等等.

3.3.3 入侵检测

入侵检测系统(Intrusion Detection System,IDS)是一种对网络传输进行即时监视,在发现可疑传输时发出警报或者采取主动反应措施的防御手段.通过分析攻击模式(例如某个特定的操作序列)可以判断是否是一个潜在的攻击.另外,传统的身份认证与授权实际上也是以实时监测调用者的角色来保证某些特殊的操作只能由经过授权的用户来调用.类似的技术还有监测调用者的系列、与入侵操作模式进行对比等等,以发现潜在的入侵行为.

3.4 软件效率

"效率"包括时间效率与空间效率.在时间效率方面,用户的响应时间要小、吞吐量要大;所谓空间效率方面,运行过程中占用资源要少,尤其是内存资源.由于早期 CPU 的计算能力有限,内存容量也有限,因此程序的效率问题在早期很受重视:算法复杂性的研究长期以来是计算机科学的重要内容,而衡量一个算法好坏的主要标准是时间开销与空间开销.算法复杂性分析多数是手工进行的.

近十多年来,人们对效率问题的重视程度有所下降:为了提高开发效率,人们往往以牺牲软件效率为代价.例如大量系统软件、框架的引入方便了应用软件的开发,但同时导致调用层次过多,系统执行效率下降.只是由于硬件速度提升较快,盖过了软件效率的下降,使最终用户感觉系统的总体速度还是增加了.不仅如此,为了提高用户的响应时间,许多分布式的服务器软件以大量的重复冗余计算为代价,为同一个用户请求创建在多个机器上的计算,并将最先得到的结果返回给用户,而将后续得到的结果直接抛弃.实际上,忽视对运行效率的追求是 IT 系统能耗持续增长的因素之一.随着能源问题的日趋突出,这方面的研究迫切需要加强.

3.4.1 效率测试

测试是发现效率缺陷的最有效方法.效率测试通过运行软件来分析软件的时间效率与空间效率.通常这个过程需要自动化的测试工具来辅助完成:测试工具主要用来模拟多种正常、峰值以及异常负载条件,从而对系统的各项性能指标进行测试.效率测试可以在用户模拟的环境中进行.不过,由于软件的运行效率与运行环境有很大的关系,因此效率测试更需要在系统部署到实际环境、但还没有实际上线时进行.

3.4.2 效率监测

在系统实际运行过程中,对效率进行监测的主要对象包括:系统的响应时间、CPU 负载情况、内存使用、客户连接实例数目、内部类型的实例数目、数据库连接实例数目,等等.通过这些信息的获取,管理者可以对照用户需求看是否发生违反需求的情况,如果有,则可以考虑通过运行时刻调整负载、增加硬件资源等方式对系统进行调整.

3.4.3 静态效率分析

编译过程中的代码优化就是建立在静态效率分析技术的基础上.通过静态分析,可以发现程序中的从来就不会执行的代码、不会被引用的变量或者赋值操作、不需要的检查、串复制、数字转换,等等,从而可以对代码进行优化:在不改变程序语义的前提下,剔除冗余的代码,从而减少内存开销,提高执行

效率.

3.5　软件易维护性

"易维护性"主要是指系统是否易于理解、是否易于根据需求的变化对系统进行调整.另外,开发人员在维护软件的过程中,为了便于及时了解与维护任务相关的一些性质,有时也需要进行软件分析.面向易维护性分析的首要目的是对软件的易维护性进行度量.为了让维护人员更好地利用分析结果,如何将分析结果展示给维护人员也是相关分析技术的一个重要关注点.

3.5.1　易维护性度量

从软件度量的角度看,软件的易维护性是一种外部属性,不能直接进行度量.需要把易维护性表示为一组可直接度量的内部属性的函数,而这些内部属性可以通过程序分析获得.易维护性度量[54]通常考虑以下两个方面的内部属性:(1)程序的结构,主要包括传统程序度量中关注的内聚、耦合等属性;(2)程序的风格,主要包括编程的格式、命名的方式等属性.从已有文献看,易维护性度量一般采用静态分析技术.

3.5.2　系统分解

系统分解是指将大型的软件系统划分为若干子系统.对于大型软件系统的维护,维护任务通常只涉及其中很少的子系统,系统分解有助于对系统不熟悉的维护人员理解与维护任务相关的子系统.从已有文献看,系统分解一般也采用静态分析技术,其基本思路是通过对软件结构的分析,将软件划分为若干高内聚、低耦合的子部分,每个子部分对应一个子系统.

3.5.3　特征定位

由于维护任务往往针对特征展开,特征定位(feature location)[55]能够分析哪些代码与哪些特征相关,维护人员可以直接利用特征定位的结果支持程序理解.静态的特征定位抽取程序的结构信息,然后通过自动或半自动地遍历程序的结构信息找到与每个特征相关的代码.动态的特征定位针对每个特征生成输入数据,然后通过执行这些输入数据和分析执行结果获取特征与代码的关系.

3.5.4　逆向工程

由于软件文档经常被人们所忽视,因此软件维护过程中经常会面临软件文档不完整的情形.这给维护过程中所必须进行的理解软件、改进软件等活动带来了很大的困难.此时,通常可以对程序代码进行静态分析,通过数据收集、知识组织、信息浏览等一系列活动,逆向恢复出关于程序的构成成分、成分之间的关系等信息,以指导具体的维护活动.

4　未来研究展望

作为软件技术研究领域的核心内容之一,软件分析技术随着软件技术的发展而处于不断发展之中,并受到如下几方面的推动:软件分析新理论和新方法的引入与集成、软件形态的新发展、软件运行平台的新发展等等.

软件分析新理论和新方法的引入与集成.随着近年来高可信软件研究的兴起,近年来软件分析出现了一些新发展趋势,包括:(1)新理论的引入与应用.将新逻辑系统应用到软件分析上,例如面向动态结构的处理,分离逻辑的提出并应用于系统代码的分析;在软件分析中运用新的数学工具,例如将代数符号计算应用于程序终止性证明;(2)静态分析与动态分析的集成与融合.静态分析与动态分析在分析精度、分析开销、适用的软件属性等方面各有所长.一个十分自然地想法就是将二者结合起来考虑.例如,先进行静态分析,为动态分析的监测部署提供依据,以减少监测代码的部署范围,缩短离线动态验证的时间,降低在线监测的开销;或者先利用离线动态验证生成大量的执行轨迹,然后进行静态分析,以提高分析精度.

软件形态的新发展.随着软件应用领域及其需求的发展,软件的形态正在发生深刻的变化:软件的规模不断增大,人们对软件行为特性认识的愿望也日益强烈.例如网构软件作为网络时代软件的新形态,其开放、自主等形态特征将产生以往软件分析未涉及的性质[56].同时,软件性质的描述需要直观,以方便具有不同知识背景的人员描述性质.软件性质的描述还需要尽量形式化,以提高分析的自动程度.近年来,由于软件应用范围的持续扩张,编程辅助工具代码生成能力的提升、计算机网络技术的推动、开放源代码理念的被认可,软件代码量的增长十分迅速.据分析,到 2025 年人们开发的代码将达到 1 万亿行[3].软件分析也因此需要一些新的视角,例如数量巨大的代码激发了一个特殊的研究方式:基于统计、挖掘的软件分析.2004 年发起的"挖掘软件池工作会议"(International Working Conference on Mining Software Repository)着重研究如何从大量现有的代码中挖掘有价值的内容,例如挖掘不同形态制品的追踪关系、挖掘以往软件项目开发过程中

具有的特性等,以支持软件的开发与维护过程.

软件运行平台的发展.软件运行平台对软件分析技术发展的影响体现在多个方面.例如多核技术对软件分析技术发展的影响、各种数据中心/服务中心对软件分析技术发展的影响等等.以多核技术的影响为例:一方面,多核体系结构使得并行软件成为一种软件常态,而并行程序的分析受制于语义复杂、状态空间爆炸等问题,与需求差距明显.另一方面,并行性也为软件分析提供了提高分析规模和能力的空间,例如模型检验的并行化以及多核平台上的运行时验证等.

软件分析是软件技术中一个得到长期关注的研究内容,并与其它的软件技术有较多的结合.随着软件规模越来越大、越来越复杂、积累的软件越来越多,软件分析必将在软件开发、软件维护等过程中发挥越来越大的作用.

参 考 文 献

[1] Shaw M. Truth Vs. knowledge: The difference between what a component does and what we know it does//Proceedings of the 8th International Workshop Software Specification and Design. Budapest, Hungary, 1996: 181-185

[2] Zhang Ming-Hua. Calculable Theory. Beijing: Tsinghua University Press, 1984(in Chinese)
(张鸣华. 可计算性理论. 北京:清华大学出版社, 1984)

[3] Binkley David. Source code analysis: A road map//Proceedings of the Future of Software Engineering. Minneapolis, MN, USA, 2007: 104-119

[4] Dwyer Matthew B, Hatcliff John, Robby, Păsăreanu Corina S, Visser Willem. Formal software analysis emerging trends in software model checking//Proceedings of the Future of Software Engineering. Minneapolis, MN, USA, 2007: 120-136

[5] Flemming Nielson, Hanne Riis Nielson, Chris Hankin. Principles of Program Analysis. Berlin, Germany: Springer Verlag, 2005

[6] Jackson Daniel, Rinard Martin. Software analysis: A roadmap//Proceedings of the Future of Software Engineering. Limerick, Ireland, 2000: 133-145

[7] Aho Alfred V, Sethi Ravi, Ullman Jeffrey D. Compilers: Principles, Techniques, and Tools. New Jersey, USA: Addison-Wesley, 1986

[8] Clarke E M, Jr Grumberg O, Peled D A. Model Checking. Cambridge, MA: MIT Press, 2000

[9] Ball T, Rajamani S K. Automatically validating temporal safety properties of interfaces//Dwyer M B ed. Proceedings of the 8th SPIN Workshop. LNCS 2057. Springer, 2001: 103-122

[10] Chen H, Wagner D A. MOPS: An infrastructure for examining security properties of software//Proceedings of the 9th ACM Conference on Computer and Communications Security. Washengton, DC, USA, 2002: 235-244

[11] Corbett J et al. Bandera: Extracting finite-state models from Java source code//Proceedings of the 22nd ICSE. Limerick, Ireland, 2000: 439-458

[12] Visser W, Havelund K, Brat G, Park S. Model checking programs//Proceedings of the International Conference on Automated Software Engineering. Grenoble, France, 2000

[13] Detlefs D L, Nelson G, Saxe J B. A theorem prover for program checking. HP Laboratories, Palo Alto: Technical Report, 2003

[14] Owre S, Rajan S, Rushby J M, Shankar N, Srivas M K. PVS: Combining specification, proof checking, and model checking//Alur R, Henzinger T A eds. Proceedings of the 8th CAV. LNCS 1102. Springer, 1996: 411-414

[15] Gulwani S, Srivastava S, Venkatesan R. Program analysis as constraint solving//Proceedings of the SIGPLAN Conference on Programming Language Design and Implementation. Tucson, Arizona, 2008: 281-292

[16] Een N, Sorensson N. An extensible SAT-solver//Proceedings of the 6th International Conference on Theory and Applications of Satisfiability Testing. LNCS 2919. Santa Margherita Ligure, Italy, 2003: 502-518

[17] Goubaut-Larrecq J, Parrennes F. Cryptographic protocol analysis on real C code//Cousot R ed. Proceedings of the 6th International Conference on Verification, Model Checking and Abstract Interpretation. LNCS 3385. Paris: Springer-Verlag, 2005: 363-379

[18] Blanchet B, Cousot P, Cousot R, Feret J, Mauborgne L, Miné A, Monniaux D, Rival X. A static analyzer for large safety-critical software//Proceedings of SIGPLAN Conference on Programming Language Design and Implementation (PLDI). San Diego, 2003: 196-207

[19] Appel Andrew W. Modern Compiler Implementation in ML. Cambridge, UK: Cambridge University Press, 1998

[20] Sagiv Mooly, Reps Thomas, Wilhelm Reinhard. Parametric shape analysis via 3-valued logic. ACM Transactions on Programming Languages and Systems (TOPLAS), 2002, 24(3): 217-298

[21] Dufour Bruno, Ryder Barbara G, Sevitsky Gary. A scalable technique for characterizing the usage of temporaries in framework-intensive Java applications//Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE). Atlanta, Georgia, USA, 2008: 59-70

[22] Ekwa Duala-Ekoko, Martin robillard CloneTracker: Tool support for code clone management//Proceedings of the International Conference on Software Engineering (ICSE). Leipzip, Germany, 2008: 843-846

[23] Zhang Jian. Sharp static analysis of programs. Chinese Journal of Computers, 2008, 31(9): 1549-1553(in Chinese)

(张健. 精确的程序静态分析. 计算机学报, 2008, 31(9): 1549-1553)

[24] Weiser Mark. Program slicing. IEEE Transactions on Software Engineering, 1984, SE-10(4): 352-357

[25] Zhao Jianjun. Applying slicing technique to software architectures//Proceedings of the ICECCS. Montorey, CA, USA, 1998: 89-99

[26] Li Bi-Xin, Zhong Guo-Liang, Wang Yun-Feng, Li Xuan-Dong. An approach to analyzing and understanding program: Program slicing. Journal of Computer Research and Development, 2000, 37(3): 284-291(in Chinese)
(李必信, 郑国梁, 王云峰, 李宣东. 一种分析和理解程序的方法——程序切片. 计算机研究与发展, 2000, 37(3): 284-291)

[27] Chanchal Kumar Roy, James R Cordy. A survey on software clone detection research. Technical Report, School of Computing, Queen's University at Kingston, Ontario, Candada, 2007

[28] Baker B S. On finding duplication and near-duplication in large software systems//Proceedings of the WCRE. Toronto, Canada, 1995: 86-95

[29] Baker B S. Parameterized duplication in strings: Algorithms and an application to software maintenance. SIAM Journal on Computing, 1997, 26(5): 1343-1362

[30] Kamiya T, Kusumoto S, Inoue K. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. IEEE Transactions on Software Engineering, 2002, 28(7): 654-670

[31] Jiang Lingxiao, Misherghi Ghassan, Su Zhendong, Glondu Stephane. DECKARD: Scalable and accurate tree-based detection of code clones//Proceedings of the International Conference on Software Engineering(ICSE). Leipzip, Germany, 2007: 96-105

[32] Gabel Mark, Jiang Lingxiao, Su Zhendong. Scalable detection of semantic clones//Proceedings of the International Conference on Software Engineering (ICSE). Leipzip, Germany, 2007: 321-330

[33] Juergens Elmar, Deissenboeck Florian, Hummel Benjamin, Wagner Stefan. Do code clones matter? //Proceedings of the International Conference on Software Engineering (ICSE). Vancouver, BC, Canada, 2009: 485-495

[34] Jiang Lingxiao, Su Zhendong, Chiu Edwin. Context-based detection of clone-related bugs//Proceedings of the European Software Engineering Conference and Symposium on the Foundations of Software Engineering(ESEC/FSE). Chocago, IL, USA, 2007

[35] Schulze Sandro, Kuhlemann Martin, Rosenmuller Marko. Towards a refactoring guideline using code clone classification//Proceedings of the Workshop on Refactoring Tools (WRT). Nashville, Tennessee, USA, 2008

[36] Adar Eytan, Kim Miryung. SoftGUESS: Visualization and exploration of code clones in context//Proceedings of the International Conference on Software Engineering (ICSE). Minneapolis, MN, USA, 2007: 762-766

[37] Harder Jan, Gode Nils. Modeling clone evolution//Proceedings of the International Workshop on Software Clones (IWSC). Wishengton, DC, USA, 2009

[38] Livieri Simone, Higo Yoshiki, Matsushita Makoto, Inoue Katsuro. Analysis of the Linux kernel evolution using code clone coverage//Proceedings of the 4th International Workshop on Mining Software Repositories (MSR'07). Minneapolis, MN, USA, 2007: 106-115

[39] Froihofer L, Glos G, Osrael J, Goeschka K M. Overview and evaluation of constraint validation approaches in Java//Proceedings of the 29th International Conference on Software Engineering (ICSE). Minneapolis, MN, USA, 2007: 313-322

[40] Wang Qianxiang, Mathur Aditya. Interceptor based constraint violation detection//Proceedings of the IEEE International Conference and Workshop on the Engineering of Computer Based Systems. Washington DC, USA, 2005: 457-464

[41] Havelund K, Rosu G. Monitoring Java programs with Java PathExplorer//Proceedings of the 1st Workshop on Runtime Verification (RV'01). Paris, France, 2001: 97-114

[42] Arnold Matthew, Vechev Martin T, Yahav Eran. QVM: An efficient runtime for detecting defects in deployed systems//Proceedings of the 23rd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA). Nashville, TN, USA, 2008: 143-162

[43] Chen Feng, Rosu Grigore. Towards monitoring-oriented programming: A paradigm combining specification and implementation//Proceedings of the 3rd Workshop on Runtime Verification, 2003

[44] Schroeder Beth A. On-line monitoring: A tutorial. IEEE Computer, 1995, 28(6): 72-78

[45] Wang Qianxiang, Liu Yonggang, Li Min, Mei Hong. An online monitoring approach for Web services//Proceedings of the 31st IEEE International Computer Software and Applications Conference (COMPSAC 2007). Beijing, China, 2007: 335-342

[46] Raimondi F, Skene J, Emmerich W. Efficient online monitoring of Web-service SLAs//Proceedings of ACM SIGSOFT/FSE 2008. Atlanta, USA, 2008: 170-180

[47] Wang Qianxiang, Shen Junrong, Wang Xiaopeng, Mei Hong. A component-based approach to online software evolution. Journal of Software Maintenance and Evolution: Research and Practice, 2006, 18(3): 181-205

[48] Popovici Andrei, Gross Thomas, Alonso Gustavo. Dynamic weaving for aspect-oriented programming//Proceedings of the 1st International Conference on Aspect-Oriented Software Development (AOSD). Enschede The Netherland, 2002: 141-147

[49] Gupta N, Mathur A P, Soffa M L. Automated test data generation using an iterative relaxation method//Proceedings of the ACM SIGSOFT 6th International Symposium on Founda-

tions of Software Engineering. Orlando, Florida, USA, 1998: 231-244

[50] Gotlieb A, Denmat T, Botella B. Goal-oriented test data generation for programs with pointer variables//Proceedings of the 29th Annual International Computer Software and Applications Conference. Edinburgh, Scotland, UK, 2005: 449-454

[51] Zhang Xiao-Xiang et al. Encyclopedia of Computer Science and Technology. Beijing: Tsinghua University Press, 1998 (in Chinese)
(张效祥等. 计算机科学技术百科全书. 北京: 清华大学出版社, 1998)

[52] Naik M, Park C-S, Sen K, Gay D. Effective static deadlock detection//Proceedings of the 31st International Conference on Software Engineering. Vancouver, Canada, 2009: 386-396

[53] Xu G, Rountev A. Precise memory leak detection for Java software using container profiling//Proceedings of the 30th International Conference on Software Engineering. Leipzig, Germany, 2008: 151-160

[54] Oman Paul, Hagemeister Jack. Metrics for assessing a software system's maintainability//Proceedings of the International Conference on Software Maintenance, 1992: 337-344

[55] Zhao W, Zhang L, Liu Y, Sun J, Yang F. SNIAFL: Towards a static noninteractive approach to feature location. ACM Transactions on Software Engineering and Methodology, 2006, 15(2): 195-226

[56] Yang Fu-Qing, Lu Jian, Mei Hong. Technical framework for internetware: An architecture centric approach. Science in China Series F: Information Sciences, 2008, 51(6): 610-622(in Chinese)
(杨芙清, 吕建, 梅宏. 网构软件技术体系:一种以体系结构为中心的途径. 中国科学(E 辑:信息科学), 2008, 38(6): 818-828)

MEI Hong, born in 1963, Ph. D., professor, Ph. D. supervisor. His research interests include software engineering, software reuse and software component technology, software production technology, and programming languages.

WANG Qian-Xiang, born in 1970, Ph. D., professor. His research interests include software monitoring, program analysis, and middleware.

ZHANG Lu, born in 1973, Ph. D., professor. His research interests include software testing, software maintenance.

WANG Ji, born in 1969, Ph. D., professor, Ph. D. supervisor. His research interests include high confidence software and systems, software engineering and distributed computing.

Background

This paper is an overview of software analysis, for the special issue of "Software Analysis" of Journal of Computer. It is mainly supported by NSFC (National Science Foundation of China) project on Group Creative Project, which was titled as "Research on Foundation and Technologies for High Confidence Software". It is a joint work between researchers from School of Electronics Engineering and Computer Science, Peking University and School of Computer Science, National University of Defense Technology.