一种 Java 程序度量工具的设计实现

李 诺, 金茂忠, 刘 超
(北京航空航天大学软件工程研究所, 北京 100083)

摘 要: IEEE 软件工程术语标准中定义度量(metric)为对一个系统、构件或具有的某个给定属性的度的一个定量测量. 为辅助这种定量的测量, 设计了一种针对 Java 程序的度量工具. 首先给出一套度量 Java 程序的度量算法的详细定义, 然后重点阐述如何通过分层的体系结构实现这些度量算法. 该设计实现的度量工具具有良好的可扩展性, 在实际应用中能确实帮助改进程序质量.

关键词: 面向对象度量; 度量算法; Java 度量工具设计
中图分类号: TP311. 5 文献标识码: A 文章编号: 0372-2112 (2004) 12A-175-05

A Java-Oriented Measuring Tool: Design, Implementation and Application

LI Nuo, JIN Mao-zhong, LIU Chao
(*Software Engineering Institute, BeiHang University, Beijing* 100083, *China*)

Abstract: IEEE nomenclature and criterion of software engineering defines that Metric is the quantificational measurement of a system, component, or a degree with any specifically attribute. In order to assist such quantificational measurement, a Java-oriented measuring tool was designed. Firstly a series of definition of algorithms of measurement for Java was given. Then the paper discussed how to implement these algorithms by a layered architecture. The measurement tool designed by this way was extensible. It can help to improve the quality of programs.

Key words: object-oriented metric; arithmetic of measurement; design of Java measurement tool

1 引言

Lord Kelvin 曾经说过: 当你能够测度你所说的, 并将其用数字表达出来, 你就对它有了一些了解; 但当你不能测度, 不能用数字表达时, 你对它的了解就很贫乏. 在过去的十年中, 软件工程界认可了 Lord Kelvin 的话[1]. 软件度量正是一种通过定量的方法衡量程序质量的技术, 它已经由一个模糊而深奥的专业转变为软件工程中至关重要的一部分[2]. 它帮助许多优秀的软件开发者确认设计是否是高质量的, 是否可以进一步优化; 帮助客户检查最终产品是否满足需求, 是否具有合格的质量; 帮助测试人员寻找测试重点, 指导工作方向.

Java 语言是纯粹的面向对象的语言, 近年来被广泛使用. 对 Java 程序的度量必须采用面向对象系统的技术度量, 考虑面向对象系统的局部化、封装、信息隐蔽、继承和对象抽象等特征. 当然, 对于 Java 程序中一个方法的度量还可以沿用传统的度量方法, 但是需扩充其定义. 为辅助人们度量 Java 程序的质量, 本文通过新的设计思路实现了一种 Java 程序度量工具.

2 度量算法

面向对象程序的基本单位是类, 因此主要关注对类规模、类层次、类协作及个体类的度量. 本文采用行数度量法度量程序规模; 采用 DIT(Depth of Inheritance Tree) 和 NOC(Number Of Children) 度量法度量类之间的继承层次复杂性; 采用 RFC(Response For a Class) 度量法度量类之间的通信复杂性; 采用 WMC(Weighted Methods per Class) 度量法度量类的逻辑复杂性; 采用 McCabe 和 Halstead 度量法度量方法复杂性.

2.1 程序规模度量

通过程序规模度量程序的复杂性, 最简单的方法就是统计程序的源代码行数. Lipow 曾指出少于 100 条语句的小程序的出错率与可执行代码行数是线性相关的, 随着程序的增大, 出错率以非线性方式增长. 可见代码行数度量法是一个简单粗糙但直观的度量方法. 本工具度量的代码行数是指 Java 语法所定义的可执行语句数.

2.2 类继承度量

Java 语言规定类之间采用单继承机制, 任何 Java 程序的类继承树都只有一个根结点 java.lang.Object. 无论是 J2SE 提供的类, 还是程序员开发的类都继承自它. 因为本工具度量的是开发者所写程序的复杂度, 不关心 J2SE 提供的类之间的关系, 所以定义根结点为 J2SE 提供的类. 本工具使用 DIT 和 NOC 两种度量法度量 Java 程序的类继承关系:

收稿日期: 2004-09-02; 修回日期: 2004-11-18
基金项目: 国家"863" 高技术研究发展计划基金项目(No. 2001AA113100)

DIT——继承树的深度, 指“从结点到树根的最大距离”[1].

NOC——直接从属于某类的子类的总数.[1]

2.3 类通信的度量

类响应集合 RFC 包含类中被调用的其他类的方法, 度量了潜在的类之间的通信. 计算公式如下: 设某类有方法 M_1, …, M_n; $\{R_i\}$= 方法 i 调用的方法集; $\{M\}$= 类的所有方法的集合; $RS=\{M\}\cup_{all_i}\{R_i\}$; $RFC=|RS|$

2.4 类的逻辑复杂性的度量

类的加权方法数 WMC 体现了类是否包含许多方法, 或者类中方法的逻辑是否很复杂. 所以采用 WMC 来度量类的逻辑复杂性. 设某类有方法 M_1, …, M_n, C_1, …, C_n 分别为 M_1, …, M_n 的复杂度. 则: $WMC=\sum_{i=1}^{n}C_i$. 在本工具中, C_1, …, C_n 采用 McCabe 复杂度(在 2.5.1 节中介绍).

2.5 方法的逻辑复杂性的度量

对方法复杂性的度量采用两种传统的度量方法——McCabe 和 Halstead 度量法, 下面针对 Java 程序对其作详细定义:

2.5.1 McCabe 度量法

McCabe 度量法以模块的控制流程图为基础确定模块的结构复杂性. 其计算公式为: $V(G)=m-n+p$, 其中 $V(G)$ 是强连通有向图 G 中的环数; m 是 G 中弧数; n 是 G 中结点数; p 是 G 中强连通分量个数. 一个正常程序的程序图总是连通的, 从出口点到入口点画一条虚弧就可使之强连通. 所以对于单入口单出口模块, $V(G)$ 的值就是程序中包含的判断语句个数加 1[3]. 对于判定点作如下定义: if, while, do…while, for, case

2.5.2 Halstead 度量法

Halstead 软件科学是软件规模的度量, 根据软件词汇量的信息预测软件的长度, 并由此分析出软件级别等度量. Halstead 有一系列度量值(如表 1 所列), 全部通过代码中出现的不同操作符个数($n1$), 不同操作数个数($n2$), 操作符总数($N1$)和操作数总数($N2$)计算得出.

表 1 Halstead 度量值计算公式[2]

名 称	计 算 公 式
实际的 Halstead 长度	$N=N_1+N_2$
预测的 Halstead 长度	$L=n_1\log_2 n_1+n_2\log_2 n_2$
程序词汇表	$n=n_1+n_2$
程序量	$V=N\log_2 n=(N_1+N_2)\log_2(n_1+n_2)$
程序难度	$D=(n_1/2)(N_2/n_2)$
程序级别	$L_e=V/D$
程序工作量	$E=V/L_e$
错误个数	$B=V/3000$
编程时间	$T=E/S=E/(stroud\times 3600\times 8)$ ($stroud=18$)

根据“Telelogic Tau Logiscope 5.1 Audit Basic Concepts”[4] 中关于 C++ 语言中操作符操作数的定义, 定义 Java 语言中的操作数为常量、变量和非 Java 基本类型的类型名, 操作符的定义则如表 2 所列:

表 2 Java 操作符的定义

类 别	操 作 符
一元操作符	+ - ++ -- ! ~ throws new () instanceof
二元操作符	+ - * / % 《 》 》》 & | ^ && || 〉〈 ≤ ≥ == !=
三元条件操作符	?:
赋值操作符	= * = /= %= += -= ≫= ≪= &= ≙ |=
结构控制	if else while() do…while() for(;;) switch case default {} break continue return label
异常	throw () try {} catch (){} finally{}
声明	public protected private static abstract final native synchronized class strictfp void 基类 变量声明 {} ~“类定义” ; ~“空声明” { …, …, …} ~“初始化列表”
其他操作符	(…) ~ 映射 {} 块 ; ~ 空语句 [] . this super 方法调用

3 Java 程序度量工具设计思想

为实现上述复杂度算法, 设计静态分析器(如图 1 所示)以 Java 源文件为输入, 实现各种复杂度计算后, 将结果以 XML 格式存入信息库等待输出. 本工具集成在北航软件工程研究所开发的 QESAT/Java 软件分析与测试工具中, 复杂度计算结果将在 QESAT/Java 的图形界面中显示, 这里只重点介绍静态分析器的设计.

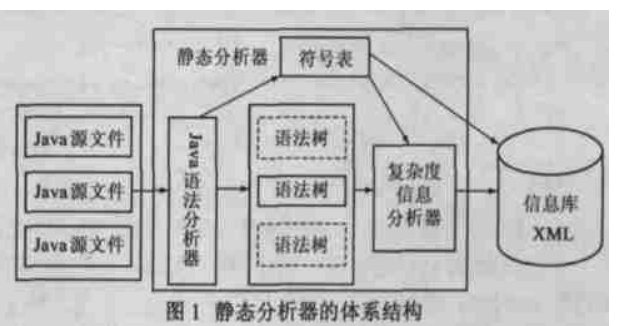

图 1 静态分析器的体系结构

3.1 Java 语法分析器

语法分析器分析 Java 源文件, 构造符号表和语法树. 它通过由 JavaCC[5] 根据 Java 语法自动生成的 Java 语法分析器改造而成. JavaCC 是 Sun 公司为促进 Java 语言的普及和配套工具的方便开发提供的一款免费软件, 是 $LL(K)$ 语法分析器的生成工具. 所以本工具中使用的语法分析器采用的是自顶向下分析的递归子程序法. 如图 2 所示, 该语法分析器主要由字符流管理器(JavaCharStream)、词法分析器(JavaParserTokenManager)和语法分析器(JavaParser)组成. Token 抽象输入文件的组成单位. 一个单词就是 Token 类的一个实例, 具有属性

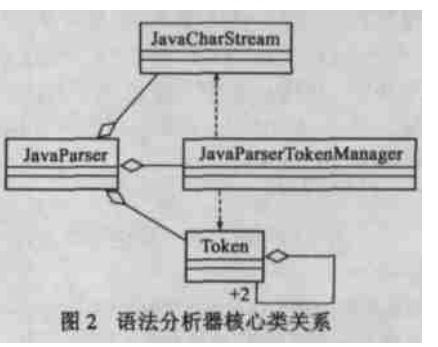

图 2 语法分析器核心类关系

和方法, 其属性将用于语法分析过程中语法成分的判别.

首先字符流管理器识别从源程序中读取 Unicode 转义序列进行词法变换, 产生相应的输入字符序列传给词法分析器. 然后由词法分析器识别其中的行终结符, 把输入序列分成行, 从而将输入字符序列转变成为包括输入字符和行终结符的输入元素. 此后非空白符和非注释的输入元素传给语法分析器. 语法分析器针对非左递归文法采用递归下降分析方法和提前扫描, 将读取的词法符号组合起来进行语法规则匹配, 为整个被分析程序建立语法树, 在某条规则匹配之后, 插入语义代码进行静态信息的提取、分析和记录.

3.2　语法树和符号表

语法树中各种语法结点之间父子关系代表着被测源文件的语法结构. 将语法结点按 Visitor 模式[6]设计, 从而使复杂度信息分析器对语法树进行遍历的过程中方便访问所有结点.

图 3 展示了语法结点的类关系. SimpleNode 对应 Visitor 模式中被访问的抽象元素对象, 其子类 ASTXxxx 对应被访问的具体元素. 这些结点中都包含一个特殊的方法 Accept(aVisitor) 用来接受访问者.

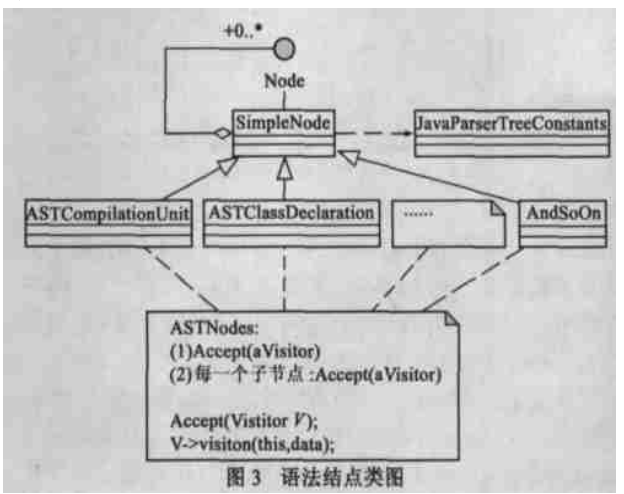

图 3　语法结点类图

语法分析器没有记忆功能, 除了匹配词法和语法, 不知道它分析过的代码包含了什么内容, 但是在计算某些复杂度时必须获取被分析源文件中的各种定义信息和引用信息, 所以需要用符号表记录被分析过代码中的定义信息. 大规模的程序通常包含数千个不同的标识符, 考虑到符号表的检索效率, 本工具采用以哈希表为主的存储结构实现符号表. 对于各标识符定义中具有多个值的属性采用了链表存储结构. 图 4 以一个典型的包定义和其中包含的标识符定义为例展示符号表的组织结构.

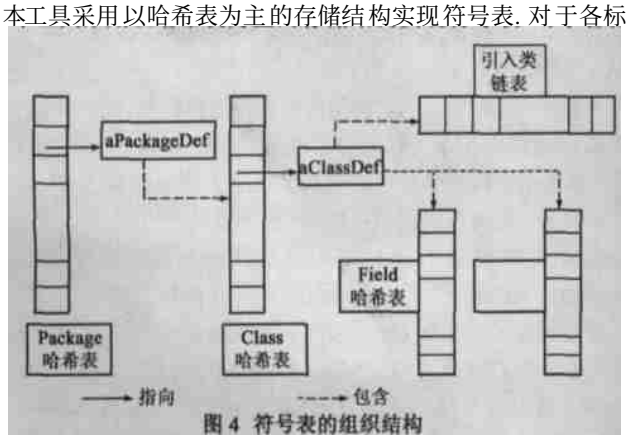

图 4　符号表的组织结构

符号表外有一层包装类定义操作符号表的接口, 实现语法分析器和符号表之间的交互, 定义向语法文件中添加信息的方法. 在分析过程中, 语法分析器创建一个符号表实例, 并调用符号表外包装类提供的方法记录所有定义信息. Java 语言自身的特征使得所有被测源文件可共享同一个符号表实例, 无需考虑语法分析器实例与被测源文件的关系是一比几.

在语法分析器分析 Java 源文件的过程中, 难以同时构造符号表和语法树. 因为在遍历语法树的过程中要分析识别变量操作就必须使用符号表中标识符定义信息, 而构造符号表的前提又是分析所有的待测 Java 源文件, 且语法树的复杂结构难以存储. 所以本工具先单独进行符号表构造的语法分析过程, 再生成语法树, 提供进一步分析所需信息.

3.3　复杂度信息分析器

复杂度信息分析器遍历语法树各结点, 同时依据符号表中的标识符定义信息以分支为基本单位提取其中的语法、变量信息, 识别本文第 1 部分介绍的与计算复杂度相关的各种信息, 遍历结束后利用获得的信息计算复杂度的值. 复杂度信息分析器对语法树的遍历方式采用 Visitor 模式, 从而将复杂的语法分析器实现代码与提取复杂度信息的代码分离, 使程序结构简单, 代码组织清晰, 提高静态分析器的可读性和可维护性. 且这种机制增强了静态分析器功能的可扩充性, 不会影响其他功能的实现代码.

如图 5 所示, 以 HalsteadVisitor 为例, 它通过继承抽象的访问者接口 JavaParserVisitor 成为一个可以访问语法树的具体访问者, 完成统计被测程序中操作符, 操作数的功能. OtherFunctionVisitor 代表其他访问者(譬如 McCabeVisitor), 或是其他任何需要在遍历语法树过程中访问各语法结点的访问者. 这种设计模式的方便之处在于每次扩充功能都不需要改变操作的接口, 对其他功能代码没有影响.

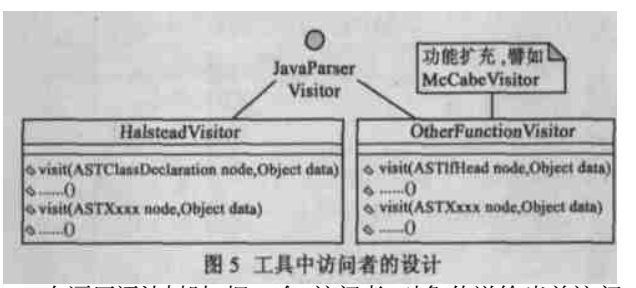

图 5　工具中访问者的设计

在遍历语法树时, 把一个“访问者”对象传递给当前访问的语法结点. 当一个语法结点“接受”该访问者时, 该语法结点向访问者发送一个包含自身类信息的请求. 该请求同时也将该语法结点本身作为一个参数. 然后访问者将为该元素执行相应的操作, 统计计算复杂度所需要的信息.

4　Java 程序度量工具的应用

下面通过对北航软件工程研究所开发的另一个项目中某些模块的度量, 展示本工具对辅助程序员提高软件质量所起

的作用. 度量的程序分别由甲、乙、丙三名程序员开发. 他们分别负责该项目中的某些逻辑类似的模块, 所以推测它们的复杂度信息不会相差甚远.

图 6 为本工具分析被测源文件后通过 QESAT/Java 框架显示度量结果的窗口. 用户通过在项目视图或者类视图中选择类或方法的名称, 就可以在信息视图中看到该类或方法的各种度量值. 为分析方便, 将静态分析得出的数据导入 Excel, 生成一系列图表. 图 7、图 8、图 9 分别展示了甲、乙、丙各自开发的 graphic 包、video 包、message 包中各个类的复杂度信息. 分别包括类的可执行语句数, DIT、NOC、RFC 和 WMC 值, 依次用纵纹, 黑色, 白色, 横纹和斜纹垂直柱表示, x 轴每个类名后面括号中的数字表示该类所包含的方法数.

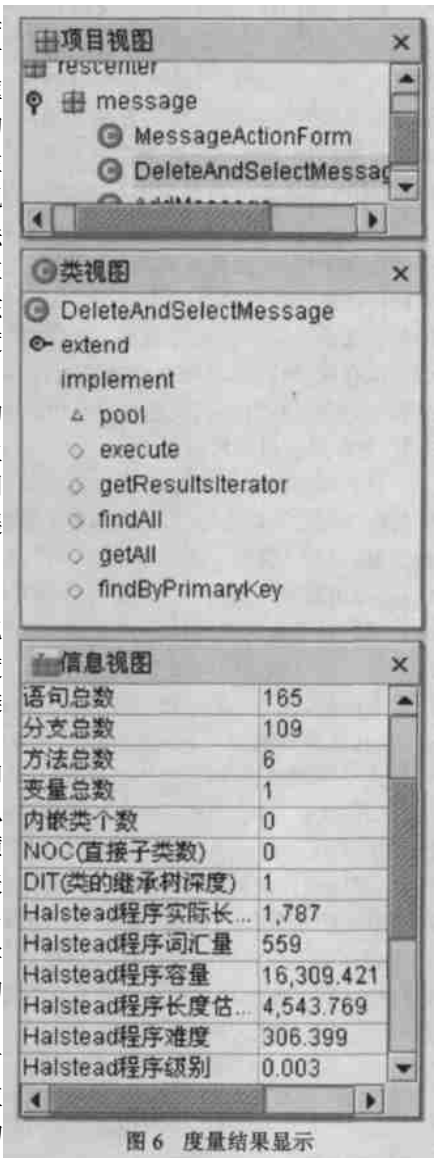

图 6 度量结果显示

由图可见所有包各个类的 DIT 值均为 1, NOC 值均为 0, 进一步检查发现它们都继承自同一个类, 而这些类实现的功能各不相同没有交叠, 说明在设计系统框架时进行了合

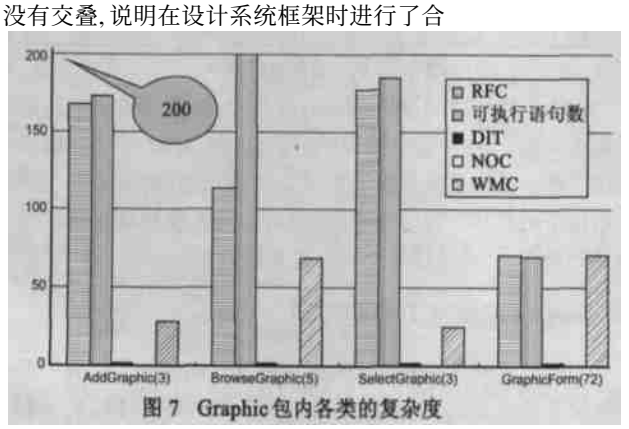

图 7 Graphic 包内各类的复杂度

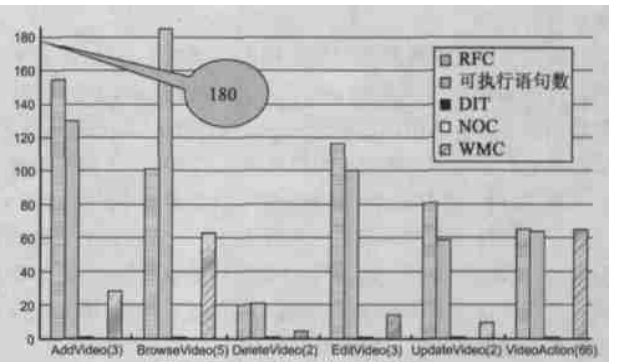

图 8 Video 包内各类的复杂度

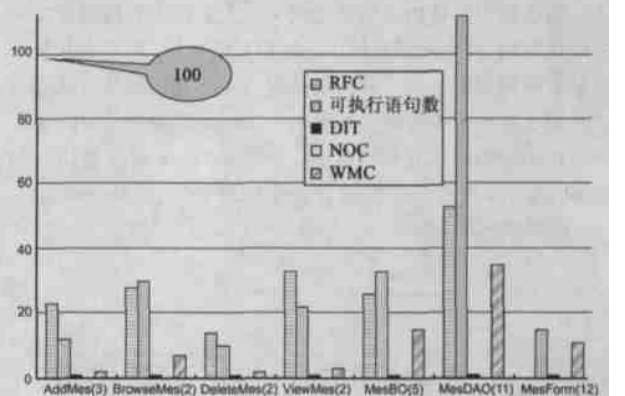

图 9 Message 包内各类的复杂度

理抽象. 但三名程序员所用类的个数、每个类的程序规模、RFC 和 WMC 的值却大相径庭. 甲的程序只有 4 个类, 每个类的规模都很大, 50% 的 WMC 值在 70 左右, 远高于乙、丙的 WMC 值. 类似的程序逻辑却产生完全不同的度量结果迫使三个人一起讨论他们的代码.

从程序规模来看, 三者的总代码量一致, 只是分布不同. 各自的 XxxFrom 类的功能是设计系统总体架构时预先定义的, 根据各 XxxFrom 类的 WMC 与方法个数比可以推断这些 XxxFrom 类代码合理. 其余类都实现对特定资源的增删改查操作. 甲将所有与检索条件有关的操作放入 SelectGraphic 中实现, 乙将其分解在不同类中实现, 丙不但将这些功能分解, 而且将数据库操作分层. 丙程序的结构虽然增加了许多类, 但是提高了代码的灵活性, 可扩展性, 不仅思路清晰而且充分实现代码复用. 每个类职责明确便于维护, XxxDAO 可以被其他需要操作同一个数据库的类调用, 一旦发现问题只需修改一个 DAO 文件.

于是甲乙将各自的代码按丙的设计思路修改. 图 10 展示了 Graphic 包中各个类修改前后程序规模和 WMC 值的变化, 具体实现增删改查操作的类复杂度普遍下降 60% 以上, 大幅度降低这些类潜在错误的可能, 且利于错误定位, 方便调试. 但由图 10 可见在获得良好类结构代码的同时也导致了 GraphicDAO 的复杂度极高, 下面进一步分析该类的结构. 图 11 描述的是 GraphicDAO 中各方法的 Halstead 程序难度、程序词汇表和 McCabe 复杂度. McCabe 值全部在 8 以下, 约 43% 的 McCabe 值为 1, 只有 FindAll 方法 McCabe 值为 45, 且其 Halstead 复杂度几乎是其他方法的 3 倍. 但程序员甲表示该方法逻辑

复杂,并且没有继续分解的必要,那么该方法应成为测试人员测试的重点.

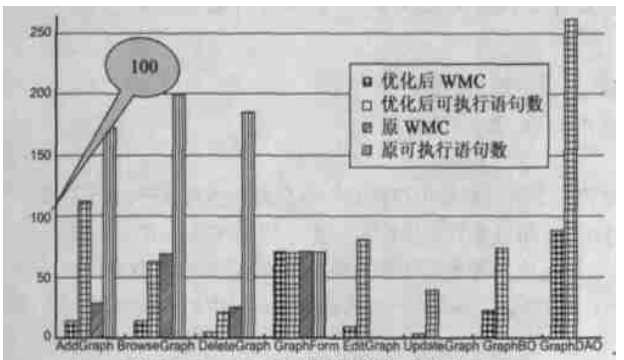

图 10　Graphic 包各类修改前后复杂度对比

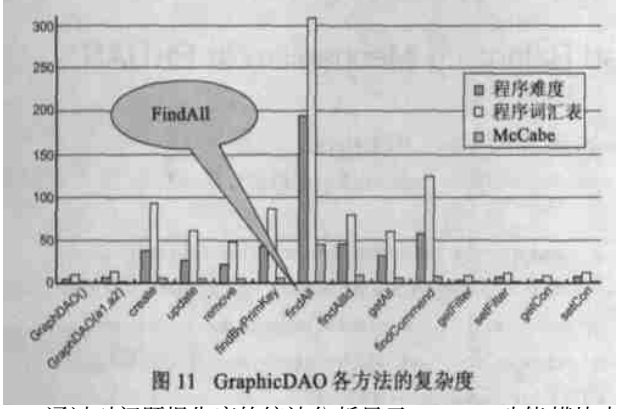

图 11　GraphicDAO 各方法的复杂度

通过对问题报告库的统计分析显示: Graphic 功能模块中 80% 的错误源于 GraphicDAO 类,而 GraphicDAO 类中 56% 的错误源于 findAll 方法. 但由于对类结构进行上述优化,更正这些错误时只修改 GraphicDAO 一个文件即可,大大减少了甲的工作量,并且有效避免修改多个文件时二次引入错误.

5　总结与展望

为帮助软件工程师们度量软件,近几年国内外出现了一些相关工具,包括武汉工业大学用 VC++ 开发的计算 McCabe 复杂度和 Halstead 复杂度的程序度量工具; 北航软件工程研究所研制的针对 C++ 、ADA 语言的度量工具 SafePro/ Eval; 以及国外的针对 java 语言的度量工具 Jstyle 等. 但是以上工具或者只是针对结构化程序度量,或者忽略传统的复杂性度量,并且均不具备跨平台性.

本工具是在一个良好的、可复用的框架下使用 Java 语言实现的. 分层的体系结构将语法分析, 信息存储及复杂度计算功能分离,使本工具不依赖于特定的语法分析器, 利于功能升级. 语法分析器是由 JavaCC 自动生成的标准语法分析器, 增强工具性能可比性. 复杂度信息分析器对语法树的遍历采用 Visitor 模式,计算各种复杂度的功能类相互独立, 便于今后新功能的扩充. 语法树的信息组织结构与符号表相结合方便各种语法、定义信息的提取, 且两者互不干扰,可根据 QESAT/ Java 其他功能的需要增加存储的信息, 为其他模块复用.

由实例分析可见,将程序的度量信息展示给程序员,辅助他们考虑程序结构是否合理, 是否可以优化. 同时引导测试人员根据度量值选择测试重点.

目前本工具实现了传统的程序规模、McCabe 和 Halstead 度量, 以及 CK 度量套件中的 DIT, NOC, WMC 和 RFC 四项度量指标. 下一步在充实符号表记录的信息后, 就可以实现计算 CK 度量套件中的 LCOM 和 CBO 两项度量指标. 另外目前度量结果显示的方式还不够直观,需要手工将数据导入 Excel 画图, 可以考虑提供自动画图的功能,以增加工具的可用性. 基于本工具分析代码是否合理很大程度上依赖于程序员的经验, 并没有成型的流程供初学者参考,所以利用本工具在实际应用中分析程序, 统计分析度量数据和测试报告, 并详细调查汇总程序员的使用经验,总结出一套完整全面的度量流程将是非常有意义的工作.

参考文献:

[1] Roger S Pressman. 软件工程: 实践者的研究方法[M]. 梅宏, 译. 北京: 机械工业出版社, 1999. 53. 464- 465.
Roger S Pressman. Software Engineering: A Practitioner's Approach [M]. Translated by MEI Hong. Beijing: China Machine Press, 1999. 53. 464- 465.

[2] Norman E Fenton, Shari Lawrence Pfleeger. Software Metrics: A Rigorous & Practical Approach[M]. Beijing: Tsinghua Press, 2003. 3. 250.

[3] 夏红霞, 童维农, 邹承明, 鄂勇辉, 钟珞. 软件复杂性度量系统的研制[J]. 计算机应用, 2000, 20(4): 16.
XIA Hongxia, TONG Weinong, ZOU Chenming, E Yongui, ZHONG Luo. Development of a software complexity evaluation system[J]. Computer Applications, 2000, 20(4): 16.

[4] Telelogic. Telelogic Tau Logiscope 5. 1 Audit-Basic Concepts[EB/OL]. http://www.telelogic.com/support, 2002- 01.

[5] Sun. Java Compiler Compiler[EB/OL]. https://javacc.dev.java.net/, 2003- 12- 8.

[6] Erich G, 等. 设计模式[M]. 李英军, 等, 译. 北京: 机械工业出版社, 2000. 218- 228.
Erich G, et al. Design Patterns: Elements of Reusable Object-Oriented Software[M]. Translated by LI Yingjun, etc. Beijing: China Machine Press, 2000. 218- 228.

作者简介:

李　诺　女, 1981 年生于北京, 博士研究生, 主要研究方向为软件工程. E-mail: seraphic@sei.buaa.edu.cn.

金茂忠　男, 1941 年生于上海, 教授, 博士生导师, 主要研究方向为软件工程, 编译技术.

刘　超　男, 1958 年生于北京, 教授, 北航软件工程研究所所长, 主要研究方向为软件工程.