计算机科学 1996 Vol. 23 No. 3

面向对象设计中软件度量学的进展*)

弓惠生
(中国科学院软件所 北京 100080)

摘 要 This article offers the situation of the software metrics study in the Object-oriented design, analyses and updates the software complexity metric used in this field, and explained how to use software metrics in Object-Oriented System.

关键字 Object-oriented design, Software metrics, Software complexity metrics.

一、引 言

软件度量学作为软件工程的一个研究方向，其目标是用定量方法管理和控制软件开发过程，评价软件质量，合理地分配资源。二十多年来在软件度量的理论基础、软件度量方法、验证、应用模式等方面已有不少结果。

近年来面向对象技术的兴起，在面向对象分析、设计、面向对象程序设计语言(OOP)、DBMS 和工具上有所发展和突破。面向对象技术采用了数据抽象(分层的数据抽象)、封装、继承、多态性、信息隐藏、重用机制等，为提高软件的可重用性，增强可维护性、可靠性，提高生产效率等方面提供了潜在的可能。上述各种机制在传统的软件开发中是不完善或缺少的，因此自然地提出了在面向对象方法中进行软件度量的研究。

本文将概述软件度量学在面向对象方法中的研究概况，并对适用于面向对象方法的度量方法进行综述、分析和初步改进，阐述了在面向对象开发方法中使用软件度量的问题。

二、为什么要对面向对象方法进行软件度量研究

面向对象方法的中心是用对象(数据与操作的集合)来模拟现实世界，常用的传统设计方法是面向功能分解，数据与操作是分开的。面向对象方法引入了新的机制，如对象、继承、封装等，因此现有的软件度量不完全适用于面向对象技术。

传统的度量方法适用于对象内部特定的方法，因为它就是普通的函数或过程。如果用语句行数(LOC)来度量整个对象不大合适，因为继承性提供了重用，在正常情况下属于对象自身的代码不会太多。

正确使用面向对象技术将使对象间的耦合进一步减少。若仅把类看作是模块不完全恰当，因为功能性模块间的耦合关系表现在接口上，主要是参数传递(传值或传地址)，对全程量的访问以及模块间的调用关系。Henry[5]等提出的信息流度量较好地反映了这种模块间的耦合关系。对象间的耦合主要表现为，通过继承、通过消息的传递与接收，通过对抽象数据类型的引用带来的耦合，因此需要新的度量来反映这些关系。当然，为传统度量提出的目标和方法在这里依然有效。

三、在面向对象方法中对软件度量的研究

文[4]中对此做了较详细的论述，综述如下：

①认为软件复杂性度量缺乏坚实的理论基础，对面向对象方法中的度量研究应建立严格的理论基础，并对此做了初步建议；

②对用面向对象技术开发的特定软件(图形信息系统)提出非数学描述的度量方法；

③用对象和方法的简单计数建立成本估算模型；

④在 C++ 环境下测量继承性度量，并以一实验系统用自动工具收集数据。

Chidamber 和 Kemerer[3,4]基于继承树提出适用于面向对象设计的一组软件复杂性度量，并对用 C++ 和 Smalltalk 编写的两个类库进行了分析。

*)国家自然科学基金资助课题。弓惠生 副研究员，主要从事软件质量学研究与 MIS 系统开发。

Li 和 Henry[5]检测了 Chidamber 提出的度量在预测软件维护工作量中的作用。

四、适用于面向对象设计的软件度量

面向对象方法的主要关键是将具体问题分解成一系列的对象和相关的类。类分两种，一种是基础类，这些类在以后的应用中将反复使用，如栈、队、链表、树等常用数据结构以及图形处理、窗口管理、自然语言处理等，它们将构成可重用的类库(构件库)；第二种是针对具体应用问题的类，尽量使这些类的设计具有可重用性，且也应遵从传统的结构化设计的原则。

1. 类设计度量[4]

(1)DIT(Depth of the Inheritance Tree)。指对象所属类在继承树中的深度(层数)，其中树根为 0，(在多继承情况，DIT 为当前结点至根的最大长度)。分析如下，DIT 数值大，则表示：1)该类维护强度大，因为它有较多的超类，将继承较多的方法与数据。另外有的 OOP(如 C++)为提高效率，在一定条件下允许绕过超类的方法去访问超类的非公有成员，则有违封装性，将给维护带来困难，此时应在效率与可维护性间进行选择。2)表明其超类有较大的重用性。3)表示对象间有较大的继承性耦合。

事实上，在多继承情况为表示复杂性，DIT 应取从该结点至根的弧长之和。

(2)NOC(Number Of Children)。指直接子类数目。NOC 为正整数或 0，其意义如下：1)NOC 值大可能表示该类具有较大的重用性。2)NOC 大可能表明该类抽象不当或子类使用不当，应重新检查设计。3)NOC 大表示该类在设计中有较大影响，此类应作为测试重点。4)表示对象间继承性耦合。

由 NOC 定义知 NOC 仅涉及一级子类，而没有顾及子类的下属类，为表明该类的全部影响范围应采用全部子类数目。

(3)RFC(Response For a Class)。定义如下，RFC＝|RS|，其中 RS 为该类的响应集。响应集是指当被测量类的一个对象接收一个消息时，有一个或多个方法准备响应，此时类选中一个执行，因此，

$$RS = M \cup \sum_{i \in M} R_i$$

其中 M 为被测类的方法集(函数成员集)，R_i 为方法 i 调用的方法之集合(为简单起见，可仅考虑方法调用的第一级，略去深度嵌套调用)。

分析表明：1)RFC 大可能是被测类方法多，因此复杂性大。2)RFC 表示对该类测试和调试的难度。3)RFC 可作为选取测试驱动类的一种判别值。选择 RFC 大的类作为测试驱动类，可得到较高的测试覆盖率。4)对 C++一类语言(面向过程语言加入面向对象机制)对象间通讯除消息传递外，还有函数调用另一途径，因而 RFC 相对要小一些，而纯面向对象语言，由于只有消息传递一条途径，因而 RFC 值相对要大一些。

(4)LCOM(Lack of Cohesion in Methods)。指类内聚缺乏度。设类 C 有 n 个方法 $m_1, m_2, \cdots, m_n$，令 I_i 表示方法 m_i 所访问的 C 的数据成员集合，令 $P=\{(I_i, I_j) | I_i \cap I_j = \varnothing\}$ $Q=\{(I_i, I_j) | I_i \cap I_j \neq \varnothing\}$。注意 $(I_i, I_j)=(I_j, I_i)$ 为无序对(如果 $I_1, \cdots, I_n$ 都为空集，则 $P=\varnothing$)，定义

$$LCOM=\begin{cases} |P|-|Q| & \text{当 } |P|>|Q| \text{ 时} \\ 0 & \text{其他} \end{cases}$$

如果 $I_i \cap I_j = \varnothing$，称相应方法 m_i 和 m_j 为无关方法对，否则称为相关方法对。

类是数据成员及方法封装的集合，LCOM＞0 表示无关方法对数量超过相关方法对。LCOM 表明类内聚缺乏度，但 LCOM＝0 并不能说明被测类的内聚性好。

因为 $|P|+|Q|=n(n-1)/2$，n 为被测类中方法数量，令 $LCOM1=(2|P|)/(n(n-1))$ 代表类内聚缺乏度更为合理，$0 \leqslant LCOM1 \leqslant 1$。

类内聚性是面向对象方法所要求的，它增强了封装性，高值的 LCOM 表明该类可能提供了不相关的功能。可用 LCOM 识别那些试图达到多目标的类，这样的类容易发生错误，测试困难，最好是进行分解。

(5)CPC(Complexity Per Class)。指类的复杂性。

CPC1＝该类数据成员复杂性＋方法成员复杂性。例如 CPC1＝D＋M(其中 D 和 M 分别为该类数据成员及方法成员数目)。

CPC2＝C1＋C2＋…Cn，其中 n 为该类有 n 个方法 M1，M2，…，Mn，C_i 为方法 M_i 的复杂性度量值，此即 WMC(Weight Method Per Class)[3]。

CP3＝NOM(Number Of Methods)该类方法数量。

CPC 综合地表示类的复杂性，可以作为开发和维护类工作量的指示器。

NOM 大，潜在地增加了应用特殊性，限制重用，实践表明大多数类的方法都较少(10 个左右)。在应用中类提供特定的服务与特定的抽象，使其结构相对简单。

实践表明方法数量大的类，大多是独立的，不重用的、"短线投资"的。

在类设计中对 CPC 与 DOC 都大的类应特别注意对它的测试。

(6)CBO(Coupling Between Object Classes)。类 C 的 CBO=与类 C 相耦合的类之数目。所谓类 C 与类 C1 相耦合是指，类 C 中的方法使用类 C1 中成员或反之(C 和 C1 无继承关系)。

CBO 应尽可能小，对 CBO 大的类应予以严格测试。对于 Smalltalk 语言 CBO 意义不大，因为它将控制结构及简单变量类型都看作是类，都将计入 CBO。

2. 对象间的耦合度量

按面向对象方法提供的对象间的通讯机制可分为三种：

(1)通过继承性带来的耦合。继承性为软件重用提供了有力的手段，但是也潜伏着破坏封装和数据隐藏的危险，例如 C^{++} 中提供的保护方式(Protected)，是以增加复杂性(子类访问基类中的非公有成员)来换取程序执行效率。这种耦合可用前面提出的度量 DIT 和 NOC 来测量。

(2)通过消息传递带来的耦合 MPC(Message Passing Coupling)。消息传递是指当一个对象需要其他对象为它服务时，前一对象向后一对象发送消息，具体动作是在后一对象(的类)中进行。对 C^{++} 就是带选择机制的函数调用，具体方式是：

对象标识符.方法名(参数表)

类 C 的 MPC=类 C 向其他类对象发送消息的数目。MPC 表示一个类中方法的实现对其他类的依赖性，但 MPC 没有包含类接收消息的数量，即该类对其他非继承类的影响。

(3)通过数据抽象耦合度量 DAC(Number of ADTs defined in a Class)。类可以看作是抽象数据类型(ADT)的具体实现，如果在类 C 中说明的变量具有类类型，就引发了 C 与另一类间的耦合。对某些 OOP 有可能违背封装的原则，如 C^{++} 的友元允许直接访问 ADT 的私有性质，也是一种以增加复杂性换取效率。

类 C 的 DAC=C 中具有说明为类类型变量的数目。DAC 大表明类间耦合高。

3. 类继承树可重用度量 ITR(Inheritance Tree Reuse)

在面向对象设计中，确认类之后，类主要按继承关系组织起来构成继承树(严格地讲是图)。方法应尽量放在继承树的高层类中，使共享这些方法的子类增多，重用得到更好的支持。我们采用前面的 DIT 和 NOM 来测量。

为了得到更好的重用，应满足 DIT 小时(高层)NOM 相对大或 DIT 大时(低层)NOM 相对小。

令 a=类继承树中 DIT 最大值；b=相应于 DIT 最大值类的 NOM 值；d=根类 NOM 值。有：

$ITR=(d-b)/a$

为提高可重用，应满足 ITR>0，且 ITR 值较大为好。

五、小结

实践表明用面向对象方法开发的软件具有更好的可靠性和可重用性。由于 OO 方法发展不够成熟，实际开发中多采用与结构化方法相混合的办法，只是在设计和编程阶段使用 OO 方法。在设计阶段主要是确定类及类间关系，最好将类组成类库以支持重用，在类库中，类主要是以继承关系组合起来的，此时可用前面的类设计度量来定量地辅助完善类的设计，类间的非继承关系可用对象间耦合度量以增加内聚，减少耦合。至于在实现阶段，主要是类中方法的编程，可采用传统的软件复杂性度量方法，降低方法实现的复杂性。

采用面向对象方法缩小了分析、设计、实现各阶段间的鸿沟，像 C^{++} 语言将类定义和类实现完成于同一形式的不同文件中，这有利于在面向对象设计中应用软件度量。

分析表明面向对象方法中软件度量研究还仅仅开始，这里提供的度量大都基于继承树这一模型，也显得过于粗糙，有待在模型与方法上进一步研究。

参 考 文 献

[1] T. Dillon, P. Tan, Object-Oriented Conceptual Modeling, Prentice Hall, 1993

[2] 杨芙清、陈钟、章远阳，面向对象程序设计，北京大学出版社，1992

[3] R. Chidamber, F. Kemerer, Towards a metrics suite for Object-Oriented Design, OOPSLA' 91

[4] R. Chidamber, F. Kemerer, A Metrics Suite for Object Oriented Design, IEEE Trans., Softw., Eng., Vol. 20, No. 6, 1994

[5] W. Li, S. Henry, Object-Oriented Metrics that Predict Maintainability, J. Systems& Software, Vol. 23, 1993

[6] 弓惠生，软件设计复杂性度量，计算机研究与发展，1992. 3