计算机科学2002Vol.29№.10

一种基于UML的类复杂性度量方法

A Metrics Suite for Class Complexity Based-on UML

张 涌 陶 隽 钱乐秋

(复旦大学计算机系 上海200433)

Abstract The complexity of software products is an important aspect in software measurement field. The complexity has a close relationship with the developing cost, time spending and the number of defects which a program may exist. OOA and OOD have been widely used, so the measurement of object-oriented software products is a indispensable part of object-oriented software engineering. UML is a modeling language for OOA and OOD, it has been accepted as an industry standard. This paper presents a suite of class complexity metrics based on UML class diagram and state diagram, and verifies them with a suite of evaluation rules suggested by Weyuker.

Keywords Complexity measurement, Object-oriented, UML

一、简介

自从面向对象的分析与设计方法问世以来,由于其拥有众多的优点,已经得到了广泛的应用。面向对象的分析与设计方法与传统的软件开发方法有许多不同之处,在软件开发过程、分析与设计技术、度量以及测试和维护方面都存在很大的差异。软件产品的度量一直是软件工程研究人员研究的一个热点问题。对软件产品进行度量可以让我们对项目开发的进度,开发成本等方面有一个定量的认识。软件产品复杂性的度量是软件度量的一个重要方面,它直接关系到软件开发费用的多少,开发周期的长短和软件内部潜伏错误的多少,同时它还间接度量了软件产品的可理解性。此外复杂性度量可以为我们的测试和维护工作的安排提供一个依据。例如,对于复杂性较高的模块我们应该投入更大的精力去测试和维护,对于特别复杂的模块,我们应该考虑采取一定的方法对它进行分解,使其复杂性降低,这样会减少代码的出错率,从而减少测试和维护费用。

统一建模语言(UML)由Rational公司提出[1],它结合了Rumbaugh、Booch和Jacobson提出的三种面向对象分析与设计方法的优点,被OMG接受而成为事实上的面向对象建模语言的工业标准,并得到了广泛的应用。本文讨论在软件的分析和设计阶段利用UML对类的复杂性进行度量,这种早期的度量可以给我们的项目开发带来一些有用的信息,避免在软件开发的后期对软件的开发方案和设计原则进行重大的调整。下面先介绍对象度量的一些相关工作,然后讨论类的复杂性度量;最后对该复杂性度量使用一定的度量理论进行评价。

二、面向对象度量的一些相关工作

面向对象的软件开发方法与传统的软件开发方法之间存在较大的差异,因此我们要根据面向对象系统的特点来定义新的度量准则,或者对传统的软件度量准则进行扩展以适应面向对象系统的度量。许多研究人员对面向对象的度量方法进行了研究。Morris最早提出了对面向对象软件开发度量的一些度量准则[2],但这些准则没有经过验证。Moreauhe和Dominick提出了对面向对象图形信息系统进行度量的三个度量准则[3],缺点是他们没有提供形式化和可测试的定义。Chidamber和Kemerer提出了对面向对象设计进行度量的六个准则[4],它们是(1)类中方法权重(Weighted Method per Class, WMC):类中所有方法的复杂性之和。(2)继承树的深度(Depth of Inheritance Tree, DIT):类的继承树的深度,在多重继承的情况下,它是从叶子结点到根结点的路径最大长度。(3)子类的个数(Number of Children, NOC):在继承的情况下,一个类的直接子类的个数。(4)类间的耦合(Coupling Between Object Classes):某一个类与其他类之间的耦合计数。(5)类的响应的个数(Response For a Class, RFC):接受外部的消息而在类中激发的方法数。(6)类中方法内聚的缺失(Lack of Cohesion in Methods),他还采用了两个项目中的类库利用该度量方法得到一些经验数据。Basili等人[5]对Chidamber的度量方法进行了验证。Li[6]指出了Chidamber度量准则中的一些缺点,并提出了另外一套度量准则(1)祖先类的个数(Number of Ancestor Classes, NAC):一个类所继承的全部祖先类的数目。(2)子孙类的个数(Number of Descendent Classes, NDC):一个类在继承树中所有子孙类的个数。(3)类中局部方法的个数(Number of Local Methods, NLM):在类中定义的且可由外部访问的方法数(即可见性为Public的方法数)。(4)类方法的复杂性(Class Method Complexity, CMC):类中所有方法的结构复杂性之和。(5)抽象数据类型之间的耦合(Coupling through Abstract data Type, CAT):在类的数据成员申明中作为抽象数据类型被使用的类的个数。(6)消息传递之间的耦合(Coupling Through Message passing, CTM):从类中发送出的不同类型的消息减去传送到在类的内部生成的对象的消息。

三、基于UML的类复杂性度量

UML是一个对面向对象分析和设计方法进行建模的描述语言,它包括用例的描述,类的静态描述,类间关系的静态描述,类的动态描述,类之间交互的动态描述以及系统描述等各个组成部分。本文主要讨论对类的复杂性进行度量,因此主要关注UML中的类图和状态图。

UML中的类图主要是对类进行静态的描述,它描述类的属性以及方法。如图1所示,从中我们可以导出两个度量准则:(1)类中属性的个数;(2)类中成员函数(方法)的个数。

张 涌 博士研究生,主要研究领域为软件复用、软件测试。陶 隽 硕士研究生,主要研究领域为软件复用、软件构件库。钱乐秋 教授,博士生导师,主要研究领域为软件复用、软件测试。

度量准则1 类中数据成员的加权值(Weighted Number of Data Member,WNODM):在面向对象领域,类中的数据成员叫做类的属性。

定义 类的数据成员的加权值被定义为从父类中继承的数据成员的加权值和各类特有的数据成员个数之和。

$NOA=w_i p+q$ p 为继承的数据成员数目,q 是该类中特有的数据成员数目,w_i 为继承的数据成员对类复杂性的贡献的权重,它应该取小于1的正数。

注意在类的继承关系中,父类的可见性为 Public 和 Protect 的数据成员在子类中可用,而可见性为 Private 的数据成员在子类中不可用,因此 p 只包括那些可见性为 Public 和 Protect 的数据成员。

度量准则2 类中成员函数的加权值(Weighted Number Of Member Function,WNOMF):

定义 我们把 WNOMF 定义为从父类中继承的成员函数的加权值和该类中特有的成员函数个数之和。

$WNOMF=w_{if}If+f+w_{pv}Pf+w_vVf$ If 是从祖先类中继承的实成员函数的个数,f 是该类中新添的成员函数的个数,Pf 是该类继承的父类中纯虚函数的个数,纯虚函数必须要在子类中实现;Vf 是该类中继承的虚函数的个数,在这里 Vf 指的是在子类中重新实现的虚函数。w_{if} 是继承的成员函数对类复杂性的贡献的权重,w_{pv} 是纯虚函数对类复杂性的贡献的权重,w_v 是虚函数对类复杂性的贡献的权重,w_{if},w_{pv},w_v 是小于1的正数,并且 $w_{if}<w_v<w_{pv}$。

注意在类的继承关系中,父类的可见性为 Public 和 Protect 的成员函数在子类中可用,而可见性为 Private 的成员函数在子类中不可用,因此 If 只包括那些可见性为 Public 和 Protect 的成员函数。

根据人们的直觉来看,某一事物的特性越多,所拥有的功能越多,它也就越复杂,这两个简单的度量准则与人们的直觉观念相一致。

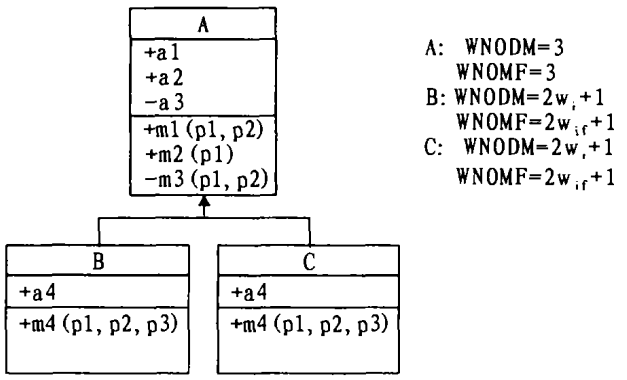

图1 UML 类图的例子以及其数据成员和成员函数的加权值

但类不仅仅是一个静态的事物,它还会接受外部的或内部的事件,并根据这些事件产生特定的动作,引起它的状态发生改变,这就是类的动态特性。在面向对象的软件开发方法中,对象和类的所有属性的当前值代表了类的一个状态[7]。UML 中的状态图可以描述对象的状态及其在对象的生存期内发生改变的过程这一动态行为。图2显示了一个简单的自动咖啡贩卖机的状态图。

UML 的状态图中的状态包括基本状态以及组合状态,其中组合状态又可包括基本状态和组合状态,组合状态包含其它的状态作为其子状态。一个组合状态可分为 or-组合状态和 and-组合状态。一个 or-组合状态是它的各个子状态之间存在不可兼或的关系的组合状态。一个 and-组合状态说明它的各个子状态是可以并发的。在图2中,状态 CVM 包含 OFF 和 ON 两个状态,OFF 和 ON 之间是不可兼或的关系,因此 CVM 是一个 or-组合状态,同理 Coffee 及 Money 状态也都是 or-组合状态。相反在 ON 状态中,包括 Coffee 和 Money 两种状态,它们可以是并发的,因此 ON 状态是 and-组合状态。UML 的状态图中的状态可以有与其相关的活动;其中有三个缺省的事件:entry,exit 和 do。entry 事件代表在进入该状态时所产生的特定活动,exit 事件代表在离开该状态时所产生的特定活动。do 事件代表在给定的状态中要执行的特定活动。例如,在 BUSY 状态中分别有 entry,exit 和 do 事件所产生的活动。

UML 中的状态转移用状态之间的箭头来表示。并被记为 event[guard]/action send,其中 event 代表何种事件会产生状态转移,[guard]表示发生状态转移所要符合的条件,action 表示状态转移时发生的动作,send 表示该状态转移会发送何种消息。我们可把发生状态转移的事件分为外部事件及内部事件,一个外部事件是由对象所处的环境或者是其它的对象产生的事件,一个内部事件是由对象本身所产生的事件。在图2中,dec 和 finish 事件都是内部事件,其它的事件均为外部事件。

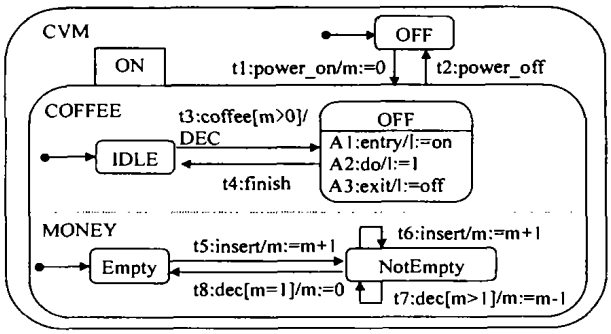

图2 一个 UML 状态图的例子(其中 m 代表 money,l 代表 light)

由于 UML 状态图中存在这种分层和并行结构,因此对它进行直接度量会产生困难,因为这些状态在 UML 状态图中本身就可能存在包含和被包含的关系,因此直接度量可能会产生二义性,例如:ON 状态可以看作一个状态,也可以看作几个状态。我们要把 UML 状态图转化为扩展有限自动机的形式,除去这种分层和并发结构,然后可对其进行度量。下面我们先描述如何把 UML 状态图转化为扩展有限自动机的形式,然后再讨论度量准则。

根据文[8]中的描述,我们使用以下概念:令 States 和 Trans 分别是 UML 状态图中所有的状态和状态转移的有限集合。并令 $\rho:States\to 2^{States}$ 是某个状态的子状态集合的函数。

1. $\rho^*(s)$代表一个状态 s 的传递闭包,它是一个状态的集合。

2. $\rho^+(s)$代表状态集合 $\rho^*(s)-\{s\}$

3. 存在一个唯一的状态 $r\in States$,使 $\rho^*(r)\subseteq States$,它叫做根状态。

4. 对于一个状态 $s\in States$,我们使用 entry(s),do(s)和 exit(s)来分别代表 s 状态的 entry,do,exit 活动。

5. 我们把一个系统可同时拥有的状态的最大集合称为系统的一个配置(Configuration),记为 C。即若 $C \subseteq States$,且(1) C 包含根状态;(2)对每一个 and-组合状态 s,或者状态 s 和它的子状态都在 C 中,或者它们都不在 C 中;(3)对每一个 or-组合状态,或者状态 s 和仅有一个它的子状态在 C 中,或者 s 以及它的所有的子状态都不在 C 中;则称 C 为一个配置。

6. 对于一个状态转移 $t \in Trans$,我们使用 source(t)和 destination(t)来分别表示 t 的源和目的状态集合;并且使用 event(t),guard(t),action(t)和 send(t)来代表状态转移标志上的各个组成部分。

为了消除 UML 状态图的嵌套和并发结构,我们引入扩展有限状态机(Extended Finite State Machine,EFSM)来作为转化的一种中间形式。一个扩展有限状态机是一个三元组〈GStates,C_0,GTrans〉,Gstates 是状态的集合,为了和 UML 状态图中的状态相区分,我们称之为全局状态,$C_0 \in GStates$ 是一个初始全局状态,GTrans 是状态转移的集合,同理我们称之为全局状态转移。从一个给定的 UML 状态图,我们可用以下方式构造出扩展有限状态机。

首先,全局状态的集合 GStates 对应于 UML 状态图中的配置集合。例如,在图2中有五个配置,它们组成了图3的 EFSM 的全局状态。EFSM 中的一个全局状态转移 $gt \in GTrans$ 是一个五元组〈C,e,g,a,C'〉。其中 $C, C' \in GStates$,它们分别代表全局状态转移的源和目的全局状态。e 代表引起转移的事件,g 代表状态转移发生所应具有的前置条件,$a = A_1(t) \cup A_2(t) \cup A_3(t)$,其中 $A_1(t) = \cup_{s \in c} entry(s)$,$A_2(t) = action(t)$,$A_3(t) = \cup_{s \in c} exit(s)$。例如对于 UML 状态图的配置 C_1={CVM,OFF},C_2={CVM,ON,COFFEE,IDLE,MONEY,EMPTY}以及 power_on 事件,我们可得到一个全局状态转移 $gt_1 = (C_1, power\_on, true, money = 0, C_2)$。对于 UML 状态图中的 do 活动,我们可以引入一个特殊的状态转移$(C, null, true, \cup_{s \in c} do(s), C')$,由此我们可以把图2的 UML 状态图转化为扩展有限状态机,如图3所示。

在得到了扩展有限状态机之后,我们需要对该扩展有限状态机进行剪枝,因为在上面的状态组合过程中,我们组合出所有可能的状态组合情况,但其中有些状态是不可达状态,我们需要把这些不可达状态删除,在度量时不计入其中。如图3所示,C_3即为一个不可达状态,因为从 C_2到 C_3的状态转移 gt_3:coffee[m>0]/1:on 的发生条件是 money>0,而 C_3的状态为 money=EMPTY,它们之间相互冲突,因此 C_3为不可达状态,则我们要从该图中剪去这一状态,得到图4。进行剪枝过程的算法如下:

```
prunning(Node)
Begin
  If all gs∈GStates are visited then return
   For each gt=(C,,e,,g,,a,,C,)∈GTrans do
   Begin
   If C,=Node and C, is not visited then
     If s∈C, is not consistent with g, then
        mark C, as infeasible
        For each gt=(C,,e,,g,,a,,C,)∈GTrans do
            If C, is infeasible or C, is infeasible then
              delete gt
     else
        make C, as a child of Node
     End
     For each child cNode of Node do
        Prunning(cNode)
   End
```

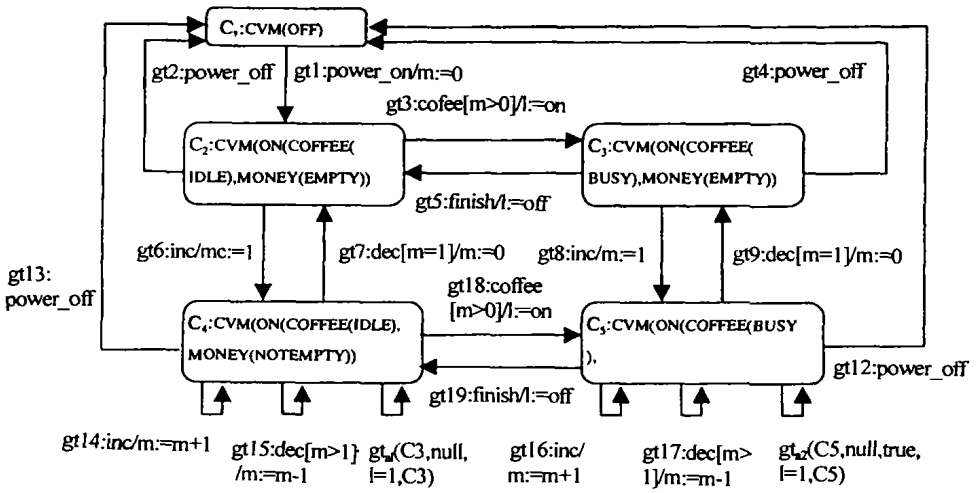

图3 EFSM 的例子

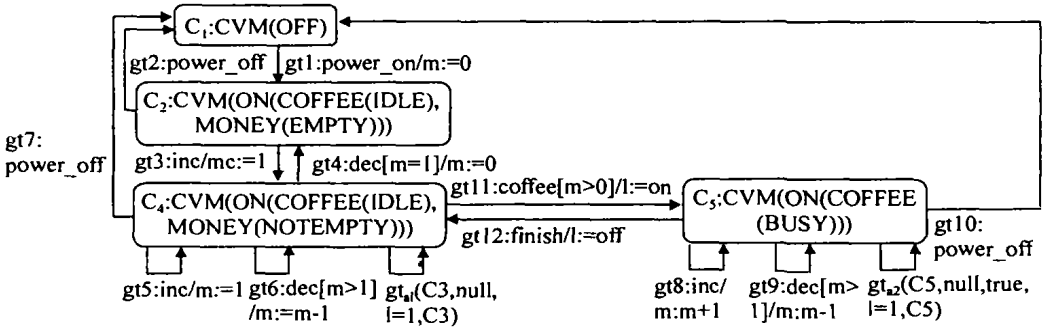

图4 经过剪枝后的 EFSM

通过把 UML 中的状态图转化为扩展有限自动机的形式后，我们可以根据该扩展有限自动机推导出一些度量准则：(3)类中全局状态的个数；(4)类中状态转移的个数；(5)扩展有限状态机中的环路数。

度量准则3 类中全局状态的个数(Number Of Global States，NOGS)：

定义 NOGS＝转化后的扩展有限状态机中的状态数。在上例为4。

度量准则4 类中状态转移的个数(Number Of Transations，NOT)：

定义 NOT＝转化后的扩展有限状态机中的状态转移数。在上例中为13。

度量准则5 有限状态机的环路复杂性(Cyclomatic Number Of States Machine，CNOSM)：

根据 Berge 的概念[9]，一个图的环路数 V(G)＝e－n＋2p，其中 e 为状态图的边数，n 为状态图的节点数，p 是图中连通分量的个数。一个图的复杂性可以由其环路数来度量，上面的扩展有限状态机的表现形式也是一个图，因此我们可以通过环路数来度量其复杂性，从而间接地度量类的复杂性。从上例中我们可以看到，每一对相邻的状态之间都存在两个边(即两个状态转移)，这意味着状态的转移是可逆的，但如果对其计算环路数，那么计算过程就会因为这些可逆的状态转移的存在而变得复杂，为了简化对扩展有限状态机的环路数的度量，我们把扩展有限状态机转化为无向图并去掉自环路，这样保持了图的复杂性，但又使度量过程不至于太复杂。

定义 CNOSM＝e－n＋2p，e 是扩展有限状态机中的边数，n 是节点数，p 是连通分量的个数，因为扩展有限状态机中的各个状态之间都是可达的，因此 p 总为1，上面的公式可以简化为：CNOSM＝e－n＋2。

对于上例来说，其环路数为3。

四、对以上度量准则的评价

Weyuker[9]提出了对复杂性度量进行评价的九个特性，在这些特性被提出来之后，有许多研究人员对其提出了批评，Fenton 提出 Weyuker 的特性不能确定对复杂性有一个单一的确定的认识[10]，Zuse 批评 Weyuker 提出的特性与尺度理论不一致。Cherniavsky 和 Smith 提出 Weyuker 的特性只是一个好的度量准则所具有的必要条件而不是充分条件[4]。但 Weyuker 的形式化的分析方法包容了以前非正规的对复杂性度量进行评价的特性，并为我们对复杂性度量进行评价提供了一定的理论基础，因此在本文中我们仍然采用 Weyuker 的评价特性对本文提出的度量准则进行评价。Chidamber 和 Kemerer[4]提出在评价面向对象的复杂性度量时，要对 Weyuker 提出的度量准则进行调查，从而给出以下六个评价特性。

Property1：给定类 P 和 Q 以及度量准则 f，存在 $f(P)\neq f(Q)$。它指出对于一个度量准则来说，并不是每一个类的复杂性度量都产生相同的结果，否则将失去度量的意义。

Property2：存在不同的类 P 和 Q，使 $f(P)=f(Q)$。

Property3：给定两个类设计，P 和 Q，它们都提供相同的功能，但并不能因此得出 $f(P)=f(Q)$。即设计的细节信息会对度量结果产生影响。

Property4：对任意的类 P 和 Q，必须有：$f(P)\leqslant f(P+Q)$ 和 $f(Q)\leqslant f(P+Q)$(P＋Q 指 P 和 Q 的合并)。它指出两个类的合并之后的类复杂性不应小于其中任一个类的复杂性。

Property5：$\exists P,\exists Q,\exists R$，若 $f(P)=f(Q)$ 并不一定能得到 $f(P+R)=f(Q+R)$。这一特性指出 P 和 R 之间的交互与 Q 和 R 之间的交互可能会产生不同的结果，因此导致其复杂性也不一样。

Property6：$\exists P,\exists Q$，使 $f(P)+f(Q)<f(P+Q)$，这一特性指出若 P 和 Q 合并后，它们之间的交互可能会使复杂度提高。

对于特性1和特性2来说，本文提出的五个度量准则很明显都能满足，因此在下面的讨论中我们不对这两个特性进行讨论。下面我们就对本文提出的复杂性度量准则进行评价。

度量准则1 类中数据成员的个数(Weighted Number Of Member Data，WNOMD)

对于特性3，我们假定 P 和 Q 是两个完成相同功能的类，但其数据成员可能会不一致，比如说某一数据成员并不是类的属性，而是为了完成某一功能所加入的变量(如信号量等)，而另一个完成相同功能的类则不使用这样的方式来完成相应的功能，它们两者的度量结果可能会产生差异，因此度量准则1满足这一特性。

对于特性4，我们把 P 和 Q 进行合并得到一新类，则其数据成员的个数不会小于合并前的两个类，度量准则1满足这一特性。

对于特性5，假定现有三个类 P，Q，R，WNOMD(P)＝WNOMD(Q)，若 P 和 R 的数据成员的交集不为空，而 Q 和 R 的数据成员的交集为空，则会产生 WNOMD(P＋R)＜WNOMD(Q＋R)的情况，所以该特性被满足。

对于特性6，该特性得不到满足。因为对于 P 和 Q 的并，令 I(X)代表类的数据成员，若 $I(P)\cap I(Q)=\varnothing$，则 $WNOMD(P)+WNOMD(Q)=WNOMD(P+Q)$，若 $I(P)\cap I(Q)\neq\varnothing$，则 $WNOMD(P)+WNOMD(Q)>WNOMD(P+Q)$。

度量准则2 类中成员函数的个数(Weighted Number Of Member Functions，WNOMF)

对于特性3，我们假定 P 和 Q 是两个完成相同功能的类，但其成员函数可能会不一致，例如若 P 类中的成员函数 A 调用了成员函数 B 从而完成某一个操作；但在类 Q 中为了完成相同类型的操作，我们可能不把 B 设计为成员函数，而是把它放在了成员函数 A 中，这样虽然它们实现了相同的功能，但它们两者的度量结果可能会产生差异，因此度量准则2满足这一特性。

对于特性4，我们把 P 和 Q 进行合并得到一新类，则其成员函数的个数不会小于合并前的两个类，度量准则2满足这一特性。

对于特性5，假定现有三个类 P，Q，R，WNOMF(P)＝WNOMF(Q)，若 P 和 R 的成员函数的交集不为空，而 Q 和 R 的成员函数的交集为空，则会产生 WNOMF(P＋R)＜WNOMF(Q＋R)的情况，所以该特性被满足。

对于特性6，该特性得不到满足。因为对于 P 和 Q 的并，令 M(X)代表类中的成员函数，若 $M(P)\cap M(Q)=\varnothing$，则 $WNOMF(P)+WNOMF(Q)=WNOMF(P+Q)$，若 $M(P)\cap M(Q)\neq\varnothing$，则 $WNOMF(P)+WNOMF(Q)>WNOMF(P+Q)$。

度量准则3 类中全局状态的个数(Number Of Global States，NOGS)

对于特性3，我们假定 P 和 Q 是两个完成相同功能的类，则其状态也应该完全相同，否则这两个类不会是相同的类，因此度量准则3不满足这一特征。

对于特性4，我们把P和Q进行合并得到一新类，考虑极限情况$S(P)\equiv S(Q)$，则它们合并以后的全局状态的个数应该与合并以前的每一个类的全局状态的个数相等，所以在两个类合并以后，其全局状态的个数至少与合并以前的每一个类的全局状态个数相等，因此度量准则3不满足这一特性。

对于特性5，假定现有三个类P，Q，R，NOGS(P)＝NOGS(Q)，若P和R的全局状态的交集不为空，而Q和R的全局状态的交集为空，则会产生NOGS(P+R)＜NOGS(Q+R)的情况，所以该特性被满足。

对于特性6，若有两个类P和Q，若P和Q的属性交集为空，则P和Q将具有单独的状态空间，假设它们都有n个状态，则NOGS(P)＝NOGS(Q)＝n，NOGS(P)＋NOGS(Q)＝2n，但它们的并的状态空间是P和Q的状态的一个组合，有可能存在n^n种状态，即NOGS(P＋Q)＞NOGS(P)＋NOGS(Q)，所以该特性得到满足。

度量准则4 类中状态转移的个数(Number Of State Transitions，NOST)

对于特性3，我们假定P和Q是两个完成相同功能的类，其状态转移的个数必定相同，若状态转移的个数不相同，则它们实现的功能不会完全一样，因此度量准则4不满足这一特性。

对于特性4，我们把P和Q进行合并得到一新类，考虑极限情况$S(P)\equiv S(Q)$，则它们合并以后的类的状态转移的个数应该与合并以前的每一个类的状态转移的个数相等，所以在两个类合并以后，其状态转移的个数至少与合并以前的每一个类的状态转移个数相等，因此度量准则4满足这一特性。

对于特性5，假定现有三个类P，Q，R，NOST(P)＝NOST(Q)，若P和R的状态转移的交集不为空，而Q和R的状态转移的交集为空，则会产生NOST(P＋R)＜NOST(Q＋R)的情况，所以该特性被满足。

对于特性6，通过度量准则3对特性6的满足情况的证明，我们显然可以得到度量准则4对于特性6来说也满足。

度量准则5 状态机中的环路数(Cyclomatic Number Of State Machine，CNOSM)

对于特性3，我们假定P和Q是两个完成相同功能的类，如前所述其状态数与状态转移的个数必定相同，因此其状态图的环路数也必定相同，因此度量准则5不满足这一特性。

对于特性4，我们把P和Q进行合并得到一新类，其状态数和状态转移的数目都会增加，因此其状态图的环路计数也会得到增加，因此度量准则5满足这一特性。

对于特性5，假定现有三个类P，Q，R，CNOSM(P)＝CNOSM(Q)，若P和R的状态图的交集不为空，而Q和R的状态图的交集为空，则会产生CNOSM(P＋R)＜CNOSM(Q＋R)的情况，所以该特性被满足。

对于特性6，由度量准则3和度量准则4对于特性6的满足情况，我们可以得到在两个类的状态的交集为空时，并且当原来两个类的状态之间存在状态转移，合并后的状态图的环路数就可能大于原来类的状态图的环路数的和。因此度量准则5满足特性6。

从以上的讨论我们可以看出，度量准则1和2不满足特性6；度量准则3，4和5不满足特性3。对于特性6来说，它的出发点是考虑类之间的交互可能会导致它的复杂性的提高，它着重强调类之间的动态特性，但度量准则1和2度量的是类的静态特性，因此这种不满足的情况是合理的；对于特性3，它的出发点是对于传统的程序设计方法来说，对完成同一功能的两个程序，设计的细节信息会导致它们之间的复杂性的不同，但对于类的设计来说，这种设计的细节信息只会对类内部的方法的复杂性产生影响，但对于更高层的抽象的类来说，这种影响是不存在的，若有两个相同的类，则它们的状态，以及状态转移的个数肯定是相同的，否则这两个类是决不可能相同的，因此这种不满足的情况出现，并不是度量准则的缺陷，而是这些复杂性度量评价准则存在不同的适用条件。

结论 本文提出了一种根据UML对类的复杂性进行度量的方法，提出了五个度量准则，并用Weyuker提出的复杂性度量的评价标准对其进行了评价。该方法应用于面向对象分析和设计阶段，并可与某些利用UML的Case工具相集成，在设计阶段自动给出度量的结果供设计人员或项目管理人员在进行决策时使用。由于UML得到了工业界的认可并被广泛使用，因此这套度量方法的应用前景十分广阔。本文仅讨论了对类的复杂性度量的准则，对于某些类组合成的子系统或系统的复杂性并没有进行讨论，由于类之间存在各种各样的动态关系(消息传递，激活等)，因此在以后的工作中我们可以进一步对类间的动态关系进行研究，归纳出一定的度量准则，使该理论能得到进一步的补充和完备，另外度量准则1和度量准则2中的各个权值本文中并没有给出，因为这些权值的确定需要大量的统计和调查工作，这也是我们今后所要解决的问题之一。

参考文献

1 UML proposal to the Object Management Group, Version 1.1, Rational Corporation, Sept. 1997. http://WWW.rational.om/uml

2 Morris k. Metrics for object-oriented software development: [Master thesis]. MIT Sloan school of Management, Cambridge, MA, 1988

3 Moeau D R, Dominick W D. object oriented graphical information systems: Research plan and evaluation metrics. The Journal of Systems and Software, 1989, 10: 23～28

4 Chidamber S R, Kemerer C F. A metrics suite for object oriented desigh. IEEE Transaction on Software Engineering, 1994, 20(6): 476～493

5 Basili V L, Briand L, Melo W L. A validation of objet oriented metrics as quality indicators. IEEE Transaction on Software Engineering, 10: 751～761

6 Li Wei. Another metrics suite for object-oriented programming. The Journal of System and Software, 1998, 44: 155～162

7 邵维忠，杨芙清著. 面向对象的系统分析. 清华大学出版社，广西科学技术出版社，1998

8 Pnuell A, Shalev M. What is in a step: on the semantics of statecharts. In: Proc. of intl. conf. on theoretical aspects of computer science, Lecture Notes in Computer Science, vol. 298, Spring-Verlag, 1991. 245～264

9 Weyuker E J. Evaluating software complexity Measures. IEEE Transaction on Software Engineering, 1988, 14(9): 1357～1365

10 Fenton N E. Software Metrics: A rigorous approach. New York, Chapman & Hall, 1991