一组基于三级度量模型的面向对象度量准则

A Criteria Suite for Object Oriented Software Measurement

李 茜 凌 辉 许晓春 徐永森 徐家福

(南京大学计算机科学与技术系 计算机软件新技术国家重点实验室 南京210093)

Abstract As object-oriented analysis and design techniques are widely used, the demand on assessing the quality of object-oriented software substantially increases. There are several standards in the measurement of software quality. Internal attribute and external attribute are on different levels. The former describes the structure complexity of the software itself. The latter describes the relationship between software and its environment, which is often related to special applications. In this paper, we propose a criteria suite for object-oriented software measurement based on the ISO9126 quality model to link the internal attribute and the external attribrte.

Keywords Object-oriented, Criterion, Internal attribute, Extenal attribute, MOOD Metrics

1 引言

软件是信息技术的核心,软件的质量则是人们最关心的因素。软件度量作为刻画、评估和预示软件质量的一个必要和重要的方面,在70年代就开始被研究工作者所关注。从传统的结构化设计到面向对象理论的产生,一方面不断出现新的软件度量指标,如著名的C&k度量体系,MOOD度量体系;另一方面,对度量指标本身的评价也提出了相应的理论,包括应满足的性质、形式化评估软件度量性质的准则等。随着面向对象技术的广泛应用,传统的度量不再适用于面向对象的某些概念,如类、继承、封装和消息传递,需要有新的度量方法学、度量指标,以更好地刻画、评估和预示面向对象系统的质量。

软件的度量有不同的标准,可以分为内部属性和外部属性。内部属性反映软件本身的结构复杂性;外部属性描述了软件与环境之间的关系。软件度量的意义就在于在实际应用中帮助管理人员控制、安排软件开发并利用反馈信息对软件进行改善,从而提高软件质量。这种指导作用只能由外部属性提供,而内部属性作为评估过程的基石,对应用者来说却是很不直观的,因而度量研究的任务之一就是建立外部属性和内部属性之间的合理联系。

本文基于ISO9126三级度量模型,提出了一组评估面向对象软件的度量准则,以期作为连接软件内部属性和外部属性的桥梁。

本文下面分如下几个方面论述:第二部分介绍相关的度量理论以及本文所提出的度量准则中会涉及到的度量指标,并讨论了软件的内部属性和外部属性在度量中的不同功用;第三部分从面向对象软件的继承、封装、多态、耦合特性上,分析了MOOD度量中相对应的标准所存在的问题,提出了一组更完善的度量准则;第四部分总结了本文的工作,并对今后的研究提出一些建议。

2 相关度量理论

2.1 度量理论的发展

以指导软件开发与应用为目标的度量理论的研究,要求具有完善的理论基础,其体系必须是实际可用的。这也正是人们常对软件度量提出的质疑之处。Prather和Weyuker认为传统的度量没有合理的数学性质,因而不能揭示正常的预示行为[15]。Zues和Fenton等人认为度量指标应建立在度量理论基础之上,并运用度量理论检验给定的度量指标是否适合特定的环境[14]。

Baker和Fenton等人提出了对软件进行度量应有的四个步骤[11,14]:

(1)确定要进行评估的软件属性;

(2)对这些属性建立相应的经验系统;

(3)定义形式化的度量指标,把经验关系系统映射到形式关系系统;

(4)对度量指标进行评估。

在论证软件度量的可用性方面,以往的研究工作提出了度量指标应有的一些性质,其中较为公认的是MOOD工程组1994年提出的软件度量中度量元应符合以下七条准则[2]:(1)要求形式化定义;(2)与大小无

关的度量元应该与系统大小无关;(3)无度量尺度或使用统一的度量单位;(4)在软件生存周期中早期可用;(5)向下可扩展;(6)易计算;(7)与语言无关。

但是仅仅上述的性质是不够的,需要有严密的形式化度量理论的提出。目前普遍被人们所接受并使用的是 Weyuker 提出的一组形式化评估软件度量性质的准则[7]。

性质1 存在 P、Q,|P|≠|Q|。

性质2 对于非负数 c,只有有限个程序具有度量值 c。

性质3 存在 P、Q,(P≠Q&|P|=|Q|)。

性质4 存在 P、Q,(P≡Q&|P|≠|Q|)。

性质5 对任意 P、Q,(|P|<=|P;Q|&|Q|<=|P;Q|)。

性质6 存在 P、Q、R,(|P|=|Q|&|P;R|≠|Q;R|);存在 P、Q、R,(|P|=|Q|&|R;P|≠|R;Q|)。

性质7 存在程序 P 和 Q,其中 Q 是 P 中语句重排后组成的程序,但|P|≠|Q|。

性质8 如果 P 是 Q 重命名,则|P|=|Q|。

性质9 存在 P、Q,(|P|+|Q|<|P;Q|)。

Weyuker 的这组准则在将度量纳入形式化范畴方面起到了很大的推进作用,但是并不能说完美。Cherniavsky 和 Smith 认为该准则给出的只是衡量良好度量指标的必要而非充分条件[16]。Fenton 认为它与衡量准则不一致[14],但是目前从度量理论的研究需要来看 Weyuker 的这组准则仍称得上是评估度量指标的良好出发点。

2.2 著名度量体系

2.2.1 对传统的结构化软件的度量 1977年 Halstead 提出软件科学(Software Science)度量[17],就程序大小、程序级别、程序难度、智力花费、估计可能的错误数量等方面做了讨论。McCabe 的环记数(Cylomatic Number)[18], Albrecht 的函数点(Function Point)[19]也都是针对结构化软件提出的度量。在国内,也有这方面的研究,南京大学徐家福先生曾对程序复杂度的量度给出更合适的定义[20],并对当时的复杂性度量方法进行了系统的分析[21]。此外,中国船舶工业总公司709所的王振宇在对树枚举理论研究的基础上,也就平均复杂度分析进行了讨论[22]。

2.2.2 C&K 度量 随着面向对象概念的提出,度量理论的研究也进入了一个新的阶段,传统的度量指标已不能符合面向对象软件的特点,需要有新的评价准则和新的度量方法学。Chidamber 和 Kemerer 于1991年提出了六条使用于面向对象设计度量的基本度量元[1],它们至今还常用作进一步研究的基石。

1)类方法的加权和 WMC(Weighted Metrics per Class),设类 C 中定义了方法 $M_1, M_2, \cdots, M_n$,C_i 是方法 M_i 的复杂度,则

$$WMC=\sum_{i=1}^{n} C_i$$

2)继承树的深度 DIT(Depth of Inheritance Tree):类在继承树中的最大深度。根结点的值取0,以下各级依次递增。

3)子类数目 NOC(Number Of Children):类的直接子类数目。

4)对象的耦合度 COB(Coupling between Object Class):与类无继承关系的耦合对的数目。

5)类响应数目 RFC(Response For a Class):设 Ms(C)是类 C 中所有方法的集合,Mr(C)={M_i|M_i 被 M_i 调用,$M_i \in Ms(C)$},则

$$RFS=|Mr(C)|+|Ms(C)|$$

6)类方法缺乏内聚度 LCOM(Lack of Cohesion in Method):设类 C 中定义了方法 $M_1, M_2, \cdots, M_n$。对于每个 $i(1 \leqslant i \leqslant n)$ I_i 是 M_i 方法中使用的实例变量的集合,令

$$P=\{(I_i, I_j) \mid I_i \cap I_j = \phi\}$$

$$Q=\{(I_i, I_j) \mid I_i \cap I_j <> \phi\}$$

$$LCOM=\begin{cases} |P|-|Q|, & \text{if } |P|>|Q| \\ 0, & \text{其它} \end{cases}$$

2.2.3 MOOD 度量 MOOD 工程组1994年首次提出 MOOD 度量以及第2.1节中所述的度量指标应符合的七条准则[2]。这组指标着重从四个方面对面向对象软件进行了度量。

1)从封装的角度:两个度量指标 MHF(Method Hiding Factor)和 AHF(Attribute Hiding Factor)的讨论侧重于信息隐蔽,值为系统中各数据成员/方法的信息隐蔽因子之和与所有数据成员/方法的比值。

2)从继承的角度:两个度量指标 MIF(Method Inheritance Factor)和 AIF(Attribute Inheritance Factor)的值为系统中继承得到的数据成员/方法与所有可用的数据成员/方法的比值。

3)从耦合的角度:CF(Coupling Factor)的值为系统中实际存在的类耦合对数与最大可能的类耦合对数的比值。

4)从多态的角度:PF(Polymorphism Factor)的值为系统中重定义(Override)的方法数与最大可能的方法重定义次数的比值。

2.3 软件的内部属性和外部属性

在将近二十年的度量理论研究中,研究工作者提出的各种度量指标,就其考核的层面来讲,不尽相同。如上文提到的 C&K 度量中的指标,是软件结构本身一些数据的统计,它们并不能明显反映出软件的质量

好坏。MOOD 度量中的指标是在语言机制的层面上，已经可以给出一些实用的信息。但从度量的应用角度来看，复杂度、易维护性等才是最直接的指标。这也就是我们所说的软件的内部属性和外部属性。

内部属性通常描述软件结构的复杂性，仅与软件本身有关。如对文本、数据流、控制流等方面的特性。内部属性一般具有清晰的定义并能进行客观的度量。

外部属性则涉及到软件和环境等外部因素的关系。如复杂度、易理解性、易维护性等。外部属性的度量往往与具体的应用要求相联系。

外部属性的实用性要建立在内部属性的客观性基础之上，因而如何建立内部属性和外部属性之间的合理联系应成为度量研究的重点之一。

ISO9126软件度量三级模型中，度量的评估分成三个级别：基本度量元级(metrics)、度量准则级(criterion)和度量因素级(factor)。基本度量元级对应于软件的内部属性，若干个基本度量元经过组合可定义出一个度量准则。度量因素级对应于软件的外部属性，度量因素由若干个度量准则组合而成。由于软件的外部属性与应用相关，这个级别的定义往往是由某个具体应用的评审者给出。这样，度量因素级别就显得尤为重要。一方面，只有确定了度量准则，对度量元进行选择时才能有的放矢；另一方面，合理的度量准则，可以为在应用中度量因素的定义提供良好的参考。下面本文就根据面向对象软件度量的特点提出了一组应用无关的度量准则。需指明的是，下文所用术语都遵照了ISO9126三级度量模型中的定义。

3 适用于面向对象度量的一组度量准则

对象在面向对象(OO)系统的构建中最为重要。类结构的好坏直接影响到程序的质量。类作为定义对象的模板由两部分构成：数据成员和方法。数据成员定义了对象的状态，而方法则定义了类可能的行为，封装机制限制了对对象数据的访问，使得信息隐蔽成为可能。良好的 OO 设计风格要求所有的数据成员只能通过类中定义的方法，使用定义一致的接口来访问。继承机制使得类的定义不用从头开始，而是可以直接获得已有类的属性。多态机制也是 OO 设计的重要特点，是指有多种定义的方法使用一个统一的接口。对于重定义可分为两类：一类是扩充功能型的重定义；另一类是完全改变功能的重定义。在这二者中，前一种类型的重定义应予提倡。面向对象软件的这些机制，也正是我们在度量过程中最需要关心的。因而，本文分别就继承、封装、多态和耦合特性提出相关的度量准则。

以往所使用的度量指标，对于不同的应用其数值的范围往往无法控制，难以据此对系统进行正确评估，因此本文采用比值的形式以克服这种缺陷。所有提出的度量准则均以比值形式描述，分子是在系统中实际使用到某个方面度量的数值，分母是其可能的最大值。这样每个度量准则的值都是在0—1之间，取0表示系统在某个特性上没有体现，取1表示在某个特性上达到了最大的使用程度。

由于在 ISO9126三级度量模型中，度量准则是通过若干基本度量元经过四则运算组合而成，在这里首先列出所有使用到的基本度量元。为了便于论述，这里凡是基本度量元的意义与 MOOD 中所用到的一致，就保持与其相同的命名规则。

3.1 若干基本度量元

本节中涉及的基本度量元有以下几个：

·TC—系统中类的总数；

·DC(C_i)—类 C_i 的所有子类(直接子类和间接子类)数目；

·Md(C_i)/Ad(C_i)—在类 C_i 中定义的方法/数据成员数目；

·Mn(C_i)/An(C_i)—在类 C_i 中定义的新方法/新数据成员数目；

·Mi(C_i)/Ai(C_i)—类 C_i 中继承得来的方法/数据成员数目；

·Ma(C_i)/Aa(C_i)—类 C_i 中可用的方法/数据成员数目；

·Mo(C_i)/Ao(C_i)—类 C_i 中被重载的方法/数据成员数目；

·LCOM(C_i)(Lack of Cohesion in Method)—类 C_i 的方法缺乏内聚度因子。

基本指标中存在如下关系：

$Md(C_i)=Mn(C_i)+Mo(C_i)$

$Ad(C_i)=An(C_i)+Ao(C_i)$

$Ma(C_i)=Md(C_i)+Mi(C_i)$

$Aa(C_i)=Ad(C_i)+Ai(C_i)$

3.2 继承性准则 IC(Inheritance Criterion)

先考虑 MOOD 度量体系中提出的 MIF(Method Inheritance Factor)指标和 AIF(Attribute Inheritance Factor)指标：

$$MIF=\sum_{i=1}^{TC}Mi(C_i)/\sum_{i=1}^{TC}Ma(C_i)$$

$$AIF=\sum_{i=1}^{TC}Ai(C_i)/\sum_{i=1}^{TC}Aa(C_i)$$

MOOD 的这两个指标之值在0～1之间，但是这个定义却无法达到最大值1。试看下例：

```
class P{public int x; public void y();}
class Q extends P{   }
class R extends Q{   }
```

这个列子是一个树型的继承关系，Q 和 R 除了从父类中继承得到的属性之外都没有再定义新的属性。

这本应该是MIF/AIF指标取得最大值的情况，也就是说其值应该为1，但是按照MIF的定义计算得到：

Mi(P)=0，Mi(Q)=1，Mi(R)=1 Total=2

Ma(P)=1，Ma(Q)=1，Ma(R)=1 Total=3

于是MIF=0.666，同样AIF=0.666，显然这是不符合实际的，出现这种偏差，是因为父类的属性不是由继承得到，在MIF/AIF的分子部分中没有体现。又考虑到对于继承特性，将方法与数据成员区分开来的意义不大，所以我们着眼于任意两个类之间的关系，对这个指标进行了改进，提出考核系统继承特性的度量准则IC，定义如下：

$$IC=\frac{\sum_{i=1}^{TC}DC(C_i)}{TC(TC-1)/2}$$

这个准则从类的角度来度量系统的继承特性，计算的是类与类之间的继承关系。当系统中任意两个类都存在继承关系时，IC取到最大值。如上例所描述的情况，当整个系统的继承结构为直线型时，各个类的子类数目分别为0，1，2，…，TC－1，此时IC的值为1。当系统内没有任何继承关系时，IC取得最小值0。准则IC的值越大，系统的继承性越好，说明系统结构越紧密，模块重用越充分；但是另一方面，随着继承层次的加深，软件的易理解性同时会降低。

3.3 封装性准则EC(*Encapsulation Criterion*)

封装作为规范软件设计的有效手段之一，对信息隐蔽和整体性两方面都提出了要求。但是，在涉及到类的划分时，我们通常只是根据经验来判断类与类之间的关联是否合理，而始终没有给出形式化的评价标准。对封装性的度量也始终局限在信息隐蔽一个角度。例如MOOD定义了如下的MHF(Method Hiding Factor)、AHF(Attribute Hiding Factor)：

$$MHF=\frac{\sum_{i=1}^{TC}\sum_{m=1}^{Md(C_i)}(1-V(Mmi))}{\sum_{i=1}^{TC}Md(C_i)}$$

$$AHF=\frac{\sum_{i=1}^{TC}\sum_{m=1}^{Ad(C_i)}(1-V(Ami))}{\sum_{i=1}^{TC}Ad(C_i)}$$

其中，V(Mmi)/V(Ami)是方法/数据成员的可见因子，其取值在0－1之间，公有成员的值为1，私有成员的值为0，保护成员的值是可以访问的类的数目与系统内类的总数的比值。

MHF和AHF评估了方法和数据成员的隐蔽特性。但是就"封装"而言，其目的是限制与对象属性有关的操作只能通过有限的途径来实现，也就是只能使用对象的方法来获得，这正是良好的程序设计风格所要求的。而类的状态是由数据成员来体现的，所以在这里MHF指标并不具有太大的意义。再看下例：

```
系统1由一个类构成
    class P{
        private int x,y,z;
        public method1(int x,int y){};
        public method2(int z){};
}
系统2也由一个类构成
    class Q{
        private int x,y,z;
        public method1(int x,int y){};
        public method2(int x,int z){};
}
```

按照MOOD的定义，类P和类Q的AHF值相同。但是很明显，类Q的封装性要优于类P。在类P中，method1和method2所使用的参数并没有交集，也就是说类P中方法之间的关联是松散的，应该可以将其分成两个类，因而用AHF来评估系统的封装特性显然是不全面的。

我们认为在评估系统的封装特性时，不能忽视整体性因素，故引入C&K的类方法缺乏内聚度度量元，与可见因子共同来对程序的封装特性进行评估，定义准则EC如下：

$$EC=\frac{\sum_{i=1}^{TC}((1-lcom(C_i))*\sum_{m=1}^{Ad(C_i)}(1-V(Ami)))}{\sum_{i=1}^{TC}Ad(C_i)}$$

$$lcom(C_i)=\frac{|LCOM(C_i)|}{Ma(C_i)*(Ma(C_i)-1)/2}$$

其中，lcom(C_i)是类C_i的方法缺乏内聚度因子，其分子是C&K的类方法缺乏内聚度度量元，分母是类中所有可用方法的二元组数目。

这里不仅考虑了数据成员的隐蔽特性，同时加入了类方法缺乏内聚度因子以评估类的内部结构的整体性。lcom(C_i)的值越大，说明类的结构越松散，类的封装特性就越差，应将该类分成几个类；lcom(C_i)的值越小，类的内聚度就越大，封装性越好。具有较好封装特性的系统，更易于理解和维护，也才算是具有良好面向对象设计风格的系统。

3.4 多态性准则PC(Polymorphism Criterion)

MOOD工程组对多态的度量指标PF(Polymorphism Factor)是这样定义的：

$$PF=\frac{\sum_{i=1}^{TC}Mo(C_i)}{\sum_{i=1}^{TC}[Mn(C_i)*DC(C_i)]}$$

这个指标并不能完全符合MOOD工程组的七条准则。如下图所示之例，PF就不满足该组准则的第五条(向下可扩展性)。

在P、Q、R组成的子系统中

Mo(P)=1，Mo(Q)=2，Mo(R)=2

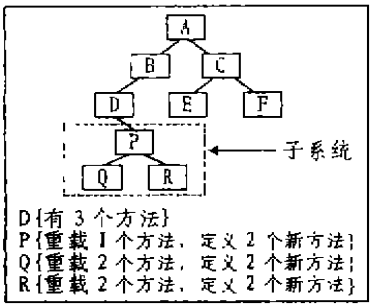
(Mn(P)=2*DC(P)=2)=4

(Mn(Q)=2*DC(Q)=0)=0

(Mn(R)=2*DC(R)=0)=0

得到PF的值为(1+2+2)/(4+0+0)=5/4>1，超出了应符合的取值范围，这是由于MOOD度量被提出时，C++是主流的面向对象程序设计语言，还没有出现

后来的 Java 语言、C++ 的语言机制中对标准类库的严格依赖，使得当时提出的 PF 定义没有考虑到系统的向下扩展，从而也就不能满足 MOOD 准则中的与语言无关的要求。



鉴于这个原因，我们将分母的计算部分改进为仅与继承关系中的子类相关，提出 PC 作为衡量软件多态特性的度量准则，其定义如下：

$$PC=\sum_{i=1}^{TC}Mo(C_i)/\sum_{i=1}^{TC}Mi(C_i)$$

其中，分子是系统中被重载的方法总数，分母是各个类通过继承得到的方法之和，也是可以被重定义的方法的最大数目。仍然使用上图之例，我们来看 PC 的计算结果。

$Mo(P)=1, Mo(Q)=2, Mo(R)=2$

$Mi(P)=3, Mi(Q)=5, Mi(R)=5$

得到 PC 的值为(1+2+2)/(3+5+5)=5/13。只有当 P、Q、R 各自对继承得到的每一个方法都进行了重定义时，PC 才能取到最大值1；在没有任何方法被重定义的情况下，PC 取最小值0。可以看出，PC 的定义克服了 MOOD 指标在向下扩展要求上的缺陷，符合 MOOD 的七条准则。面向对象技术的"多态"概念为软件提供了更为灵活的重用手段，使得一个对象可以有不同的状态，从而在不同的环境下被使用。因而我们这里定义的准则 PC 也是衡量软件可重用性的一个因素，PC 的值越大，说明软件的可重用性越好。

3.5 系统耦合性准则 CC(Coupling Criterion)

对耦合性的度量我们沿用 MOOD 的评估方法，CC 定义如下：

$$CC=\frac{\sum_{i=1}^{TC}[\sum_{j=1}^{TC}is\_client(C_i,C_j)]}{TC(TC-1)}$$

其中，分子是系统中存在的类的耦合对数目，分母为最大的类的二元组数目。当类 C_i 与 C_j 存在耦合关系时，$is_client(C_i,C_j)$取1；当类 C_i 与 C_j 不存在耦合关系时，$is_client(C_i,C_j)$取0。

对于一个结构设计良好的系统，准则 CC 的值越小越好，因为系统中过多的耦合关系，会使系统的结构复杂，也不利于模块化设计和模块的重用，类越独立，在应用中的可重用性才会越好。同时耦合关系越多，对系统中进行部分修改时就越敏感，维护起来就越困难。

结论 本文中，我们从度量的不同层次分析了软件的内部属性和外部属性及二者之间的关系，指出建立外部属性和内部属性之间的合理联系是度量研究的重要任务之一。然后，基于 ISO9126三级度量模型，本文提出了一组适用于面向对象软件度量的度量准则，并分别就继承特性、封装特性、多态特性和耦合特性四个方面加以讨论。由于构造这些度量准则的基本度量元在理论上都是相当成熟的，经过四则运算的组合，我们提出的四个度量准则保持了这些基本度量元的性质，符合 MOOD 评估准则和 Weyuker 的形式化判定准则。

目前，面向对象软件度量的理论研究还不成熟，从理论到实际应用也同样有很长的一段路要走。这里，我们认为面向对象软件度量还有待于在以下几个方面进行进一步研究：

·对软件属性清晰严格的形式化定义：特别对软件的外部属性，如复杂度、易理解性、易维护性等还多属于经验性的评价，正如 Fenton 所指出的"对许多软件属性，我们还只有非常粗糙的经验关系系统"[12]。

·动态度量：目前对软件度量的研究还仅限于静态度量，但是在应用领域特别是实时系统的度量中，更要求能对系统进行动态的测试及评估[11]。

·实践应用：将度量应用到软件工程实践中去，用实际数据检验其有效性、合理性和实用性，以指导软件管理人员的工作。度量在不同面向对象环境的应用还可以检验面向对象环境和语言，从而有助于建立面向对象环境和语言的基准评价。

参 考 文 献

1 Chidamber S R, Kemerer C F. A Metrics Suite For Object-Oriented Design. IEEE TSE, 1994, 20(5)476～493

2 Brito F, Abreu E. MOOD-Metrics for Object-Oriented Design. In: OOPSLA'94 Workshop on Pragmatic and Theoretical Directions in Object-Oriented Software Metrics. Portland, OR, 1994

3 Harrison R, Counsell S J, Nithi R V. An Evaluation of The MOOD Set of Object-Oriented Software Metrics. IEEE TSE, 1998, 24(6)

4 Mayer T, Hall T. Measuring OO Systems: A Critical Analysis of the MOOD Metrics. IEEE TSE, 1999. 108～117

5 Bakasubramanian N V. Object-Oriented Metrics. IEEE TSE, 1996. 30～34

6 Kim E M, et al. Analysis of Metrics for Object-Oriented Program Complexity. IEEE TSE, 1994. 210～207

(下转第43页)

Environment)是一个针对C++语言的程序分析和理解工具，总体结构如图2。COD布局算法用在图形生成部分，得到的一个例子C++程序的类继承图，当鼠标指向某一个类时，在文本窗口显示出这个类的基本信息。

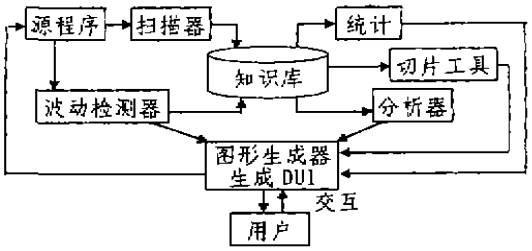


图2 OOPSE总体结构

结论 本文提出了一种层次结构图形的布局算法，对于典型的面向对象语言程序的二维图形生成，此种实现方法的思想都是可应用的，其最大优点是由于将布局阶段的工作提出来单独完成，在前端有一个数据源接口，后端有一个绘图接口，优化不仅和语言无关同时占用计算资源极少，效率很高，响应时间极短，然而由于毕竟是基于算法的布局，对数据源的数据结构还是有一定要求，并且在绘图接口还没有应用处理大数量结点的方法，这也是今后改进的一个主要方向。

参 考 文 献

1 Meyer B. Competitive Learning of Network Diagram Layout. In: Proc. 1998 IEEE Symposium on Visual Languages

2 Lin T, Eades P. Layout Creation Methods For Software Visualization. Software Visualization, World Scientific Publishing Co Pte Ltd, 1996. 61～82

3 Cross J H, Hendrix T D Language Independent Program Visualization. Software Visualization, World Scientific Publishing Co. Pte Ltd, 1996. 27～45

4 Bertolazzi P, et al. Parametric graph drawing. IEEE Trans. on Software Engineering, 1995, 21(8): 662～673

5 Eades P, Sugiyama K. How to draw a directed graph. Journal of Information Processing, 1991. 424～437

6 Eades P. A heuristic for graph drawing. Congressus Numerantium, 1984, 42: 149～160

7 Lange D B, Nakamura Y. Program Explorer A Program Visualizer for C++. USENIX Association, Conference on Object-Oriented Technologies(COOTS) 1995. 26～29

8 Oudshoorn M J, et al. Aspects and Taxonomy of Program Visualization. Software Visualization. World Scientific Publishing Co. Pte. Ltd., 1996. 3～26

9 Van Wijk J J. Scientific Visualization. Computer Systems and Software Engineering, Kluwer Academic Publishers, 1992. 387～396

10 Sugiyama K. A cognitive approach for graph drawing. Cybernetics and Systems: An International Journal, 1987, 18, 447～488

11 DiBattista G, et al. Algorithms for drawing graphs: an annotated bibliography. Journal of Computational Geometry Theory and Applications, 1994, 4: 235～282

12 Price B, et al. An Introduction to Software Visualization. Software Visualization Programming as a Multimedia Experience, The MIT Press, 1998. 3～28

(上接第29页)

7 Weyuker E Evaluating Software Complexity Measures. IEEE Trans. On Software Eng, 1998, 14: 1365～1375

8 Kolling M, Risenberg J. Support for Object-Oriented Testing. TOOLS 28, Noverber 1998. 23～26

9 MacDonald A, Carrington D. Object-Oriented Design. TOOLS 28, Noverber 1998. 23～26

10 Periyasamy K, Liu X. A New Metrics Set for Evaluating Testing Effort for Object-Oriented Programs. IEEE TSE, 1999. 84～93

11 Yacoub S M, et al Dynamic Metrics for Object Oriented Designs. IEEE Trans. Sotware Eng, 1999. 50～61

12 Fenton N E. Software Measurement: A Necessary Scientific Basis, IEEE Trans. Software Eng, 1994, 20(3): 199～206

13 Baker A L, et al. A Philosophy for Software Measurement, J. System Software, 1990, 12: 227～281

14 Fenton N E. Software Metrics. A Rigorous Approach. Chapman and Hall, London, 1991. 337

15 Kearney J K, et al. Software Complexity Measurement. Comn. ACM, 1986, 29: 1044～1050

16 Zues H. Software Complexity. Mesures and Methods. Walter de Gruyter, Berlin, 1990. 605

17 Halstead M H. Elements of Software Science. New York: Elsevier North—Holland, 1997

18 Macabe T J. A Complexity Measurement. IEEE Trans. Software Eng., 1976, 2(4): 308～320

19 Albrecht A J. Measuring Application Development Productivity, In: Proc. Join SHARE/GUIDE/IBM Application Development Symposium. 1979. 34～43

20 徐家福，张幸儿. 程序复性度量. 南京大学学报，1979

21 袁峰，陈佩佩，徐家福. 程序复杂性量度的一些分析. 计算机应用与软件，1984，6

22 王振宇. 树的枚举与算法复杂性分析. 国防工业出版社