类设计质量评估方法的研究

胡文生[1] 杨剑锋[1] 赵 明[2,3]

(贵州理工学院电气与信息工程学院 贵阳 550003)[1]

(贵州可靠性工程研究中心 贵阳 550005)[2] (耶夫勒大学技术学院 耶夫勒 SE-80176)[3]

摘 要 详细介绍了C&K度量方法,结合灰色关联分析的相关理论,提出了一种基于C&K度量方法和灰色关联分析的类设计质量评估方法。依据C&K度量阈值及可接受类的定义,可以推导出面向对象程序设计中的最佳类设计标准。将各个类与最佳类设计标准进行灰色关联分析,从而评估类设计质量的优劣。该方法为程序设计人员提供了类设计质量优劣的判定依据,并保证其在软件生命周期的早期阶段及时发现设计质量低劣的类并做相应的处理,避免后期开发的软件产品出现故障,能显著提高软件产品的可靠性和可维护性。

关键词 C&K度量方法,灰色关联分析,设计质量,评估

中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.12.029

Methodology for Classes Design Quality Assessment

HU Wen-sheng[1] YANG Jian-feng[1] ZHAO Ming[2,3]

(School of Electrical and Information Engineering, Guizhou Institute of Technology, Guiyang 550003, China)[1]

(Reliability Engineering Research Center of Guizhou Province, Guiyang 550005, China)[2]

(Department of Technology, University of Gävle, Gävle SE-80176, Sweden)[3]

Abstract This paper introduced a metric suite of C&K suggested by Chidamber and Kemerer in detail, and combined with grey relational analysis theory. A methodology for classes design quality assessment based on a metric suite of C&K and grey theory was proposed. Firstly, this methodology provides the best class design standards according to the thresholds of C&K and the definition of acceptable class. The grey relational analysis is carried out between the best class design standards and all classes of an object-oriented program, and the worst class will be found. This methodology can help the software designers to find out the flawed classes in the early phases of the software life cycle, thereby improving the reliability and maintainability of software systems.

Keywords Metric suite of C&K, Grey relational analysis, Design quality, Assessment

好软件是设计出来的,如何度量软件设计质量的优劣一直是软件工程领域中研究的热点和难点问题之一。作为软件生命周期的前期产品,软件的需求文档和设计文档是软件生命周期后续活动的根本,如果根基坏了,后续投入再多的时间和成本也是徒劳。在需求分析和设计阶段中所引入的缺陷被发现得越晚,对软件系统的危害程度就越大。James Martin等人[1-2]对软件故障的来源进行了调研,结果发现,大部分软件系统故障来自软件生命周期的前期阶段,其中需求阶段和设计阶段引入的故障占到77%以上。为了保证软件系统的质量,软件工程领域掀起了度量软件质量的研究。软件度量学最初是由Rubey和Hurtwick于1958提出的[3]。经过几十年的不断发展,已经产生了许多不同的度量方法,这些软件度量方法都是为了适应软件开发方法发展的不同时期而产生的。因此从这个意义上可将软件度量方法大体分为两类:面向过程的软件度量方法和面向对象的软件度量方法。早期软件的最主要开发方法是结构化设计方式,提倡将数据和过程分离,程序都是由各个模块组成的。为适应这种开发方法,大多传统的软件度量方法把软件模块作为研究的主要对象。这个时期最有代表性的度量方法有:基于代码行(LOC)的度量方法、Halstead提出的基于文本复杂性度量方法、McCable提出的基于程序拓扑结构的软件复杂性度量方法和Kuo-Chung提出的基于数据流的复杂性度量方法。由于结构化设计方式不适应复杂的、大规模的软件开发,因此在20世纪80年代出现了面向对象的开发方法。面向对象技术采用了封装、继承、多态等一系列新机制,因此这种设计机制完全不同于面向过程的软件设计方式,它将数据和操作过程封装在一起所构成的类作为程序设计的基础,通过继承、多态的方式来实现代码的重用。因此传统的以模块为核心的软件度量方法已经不能

到稿日期:2017-06-01 返修日期:2017-08-15 本文受贵州省科技合作计划项目(黔科合LH字[2015]7105),贵州省科技计划项目(黔科合基础[2016]1066),贵州省科学技术基金(黔科合J字[2015]2064),贵州理工学院高层次人才科研启动经费项目(XJGC20140703)资助。

胡文生(1969-),男,博士,副教授,主要研究方向为软件可靠性、可维护性,E-mail:1204489713@qq.com;杨剑锋(1987-),男,博士,副教授,主要研究方向为软件可靠性、可维护性,E-mail:332656028@qq.com;赵 明(1962-),男,教授,博士生导师,主要研究方向为软件可靠性、应用统计,E-mail:guidaiiee@hotmail.com。

完全适用于面向对象的软件系统，需要有新的度量理论和新的度量方法与之相适应。Chidamber 和 Kemerer[4] 于 1994 提出了专门针对面向对象系统的 6 个度量指标，用于描述面向对象设计的规模与复杂度，通常被称为 C&K 度量体系。后来的学者在此工作的基础上先后提出了不同的面向对象的度量方法，如 Brito 和 Abreu 等人[5] 提出的 MOOD 面向对象的度量方法、Tegarden 和 Sheetz 等人[6] 提出的面向对象的复杂性度量方法、Etzkorn 等人[7] 提出的基于类的复杂性度量方法。在上述度量方法中，预测类级别故障时，通常使用 C&K 度量集，如 Bansiya 等人[8] 基于 C&K 度量集提出类层次评估模型，Yeresime 等人[9] 将机器学习与 C&K 度量集结合起来用于预测软件系统发生故障的概率，Dubey 等人[10] 利用 C&K 度量集来评估软件系统的可维护性。

自从 Chidamber 和 Kemerer 提出 C&K 度量集后，软件工程领域的度量学得到了很大的发展。但是随后的学者都是将 C&K 度量集与软件系统的质量进行联系，或考查软件系统的可维护性，或考查软件系统的可靠性，或预测软件系统的故障情况。考虑到类是面向对象程序设计的基本单元，类设计质量的好坏在一定程度上决定了软件系统的质量。因此，本文主要研究如何利用 C&K 度量集考查向对象的程序设计过程中每个类的设计质量，基本思路是：首先计算各个类的 C&K 度量集并构造向量，综合考虑各个类的 C&K 度量集向量从而确定最佳类 C&K 度量集向量，将每个类的 C&K 度量集向量与标准类的 C&K 度量集向量进行比较，以确定设计质量较好的类和设计质量不好的类。在比较过程中使用了灰色系统理论，其中，考查两个向量的关系通常有灰色绝对关联度和灰色相对关联度两种方法。灰色绝对关联度从几何形状方面考查两个向量的相似程度，而灰色相对关联度从两个向量的空间位置考查它们的接近程度。本文既需要考查每个类设计的 C&K 度量集向量与标准类设计的 C&K 度量集向量之间的形状相似度，也要考查两者的接近程度。因此，本文利用了一个既包含灰色绝对关联度又包含灰色相对关联度的灰色综合关联度，通过灰色综合关联度的计算，从 UML 类图中找出设计质量最好的类和设计质量最差的类。该方法既可以指导软件设计人员在设计 UML 类图时，重点关注设计质量较差的类，并对它们进行改进，从而提高软件的质量，减少软件维护成本；也可以为软件管理人员做决策提供依据。

本文第 1 节主要介绍背景知识；第 2 节具体介绍由 Chidamber 和 Kemerer 提出的一套面向对象的度量集，并提出一种软件质量评估方法；第 3 节主要介绍该方法的具体应用过程；最后总结全文。

1　C&K 度量

在众多面向对象的软件度量方法中，C&K 度量体系是较早被提出的。通过分析 C&K 度量体系中的 6 个度量指标不难发现，它们都是针对类的设计质量进行度量的，符合本文对软件维护的早期预测的研究，因此有必要重点介绍 C&K 度量体系。

1.1　C&K 度量 1：每个类的加权方法（Weighted Methods per Class，WMC）

定义该指标的主要目的是度量面向对象系统的复杂性。假设面向对象系统中类 C 包含有方法 M_1，M_2，…，M_n，其中方法 M_i 的复杂性为 $C_i(i=1,2,\cdots,n)$，则类 C 的加权方法为：

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

如果将所有方法的复杂度都看作单位 1，那么类 C 的 $WMC=n$，即类 C 所包含的方法数为 n。由此可以看出，一个类所包含的方法数越多，该类就越复杂，对子类的潜在影响就会越大，对该类的开发和维护需要更多的时间和成本。因此，一个类所包含的方法数小于或等于 10 是比较合理的。

1.2　C&K 度量 2：继承树的深度（Depth of Inheritance Tree，DIT）

在面向对象的设计中，往往将组成面向对象系统的基本单元——类组成一定的层次结构，这种层次结构可以表示为一棵树，称为继承树。树中的结点表示类，一个类的 *DIT* 值为从继承树中表示该类的结点开始到树根结点的最长路径的长度。一个类的 *DIT* 值越大，表明该类在继承树中的层次越深，祖先类的数目就越多，由于牵涉到太多继承的方法，因此设计和维护该类就会变得很复杂。但是在计算一个类的 *DIT* 值时，往往会遇到文献[11]中所阐述的情况，即在面向对象的编程过程中，编译系统会提供大量的类库，所编写的程序中很多类都是从这些类库中继承的，如果将编译系统提供的全部类库也计算在继承树上，会出现这样一种情况：有些类很简单，但是它继承过来的类库有很多，从而表现出很大的继承深度；而另一些类，全部基类都是程序员自己编写的，得到的类的继承深度却很小，从而有可能导致错误的结论。因此为了避免出现这种情况，李心科[11] 提出了相对继承树的深度，即类在继承树中相对于类库中类的深度（类库中所有类的深度规定为 0）。一般来说，一个类的相对 *DIT* 值小于或等于 10 是比较合理的。

1.3　C&K 度量 3：孩子数（Number of Children，NOC）

孩子数是指在继承树中属于一个类的直接子类的数目。一个类可能有很多个子类，既有直接子类，也有间接子类。但是在计算一个类的 *NOC* 值时，将间接子类排除在外，而只计算该类的直接子类。一个类的 *NOC* 值反映了该类的潜在影响范围，*NOC* 值越大，该类的影响范围就越广，维护该类就变得越困难。

1.4　C&K 度量 4：对象类之间的耦合（Coupling between Object Classes，CBO）

当一个类的方法使用了另一个类定义的方法或变量时，就称该类与另一个类发生了耦合。一个类的 *CBO* 被用于计算与该类发生耦合的其他类的数目。“高内聚、低耦合”是面向对象程序设计最基本的原则，一个类的 *CBO* 值越小表明该类越独立，对其他类或受其他类的影响就越小，维护起来就越容易；反之，一个类的 *CBO* 值越大，在设计该类时对其他类变动的敏感性就越高，维护该类就越困难。

1.5　C&K 度量 5：类响应集（Response For a Class，RFC）

一个类的响应集是指为了响应该类对象所接受的消息而潜在执行的所有方法的数目。一个类的 $RFC=|RS|$，其中 RS 表示该类的响应集，而 $RS=\{M\}\cup\{R_i\}$，其中 R_i 为方法 i 所调用的所有方法的集合，$\{M\}$ 表示该类的所有方法的集合。该度量指标用于度量一个类与其他类之间潜在通信的情况，一个类的 *RFC* 值越大，表明该类对象在响应一条消息时，调

用的方法数越多，测试和调试该类就越复杂，维护该类就变得很困难。因此在设计类时，一般趋向于该类调用其他类的方法数越少越好。

1.6 C&K 度量 6：内聚缺乏度(Lack of Cohesion in Methods，LCOM)

假如类 C_1 有 n 个方法 $M_1,M_2,\cdots,M_n$，$\{I_i\}$ 表示方法 M_i 所使用的实例变量集。设：

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

$Q=\{(I_i,I_j)\mid I_i\cap I_j\neq\varnothing\}$

则：

$$LCOM=\begin{cases}|P|-|Q|, & \text{如果}|P|\geqslant|Q|\\ 0, & \text{否则}\end{cases}$$

$LCOM$ 值就是空的交互数减去非空的交互数，其可以把一个类中的实例变量集和该类的方法集紧密地结合在一起。该指标值用于衡量一个类中的方法的相对分散程度，反映了该类的封装性。如果一个类中有不同的方法在相同的实例变量集上完成不同的操作，则这个类是内聚的。$LCOM$ 值越大，表明该类的内聚性越差，将来出错的可能性就越大，维护该类就比较困难。

2 类设计质量的灰色评估

对于面向对象的软件，高内聚、低耦合、低复杂性是其最基本的要求。如何利用上述的 C&K 度量体系衡量面向对象软件系统的设计质量的好坏，一直是软件度量领域中研究的热点。本节主要讨论类设计质量的评估。

设 C 为要考查的系统 S 中所有类的集合，即 $C=\{C_1,C_2,C_3,\cdots,C_n\}$，$P$ 为 C&K 度量指标体系中所有指标的集合，即 $P=\{P_1,P_2,P_3,\cdots,P_6\}$，其中 P_1 表示 WMC，P_2 表示 DIT，P_3 表示 NOC，P_4 表示 CBO，P_5 表示 RFC，P_6 表示 $LCOM$。类 C_i 关于指标 P_j 的取值记为 $X_{ij}(i=1,2,\cdots,n,j=1,2,\cdots,6)$。

定义 1(度量指标的阈值) 把每个度量指标可能的取值范围分成正常和异常区域的值称为该度量指标的阈值。用 T_i 表示指标 P_i 的阈值，则 P_i 的取值范围可分成：

$M_{1i}=\{P_i\mid P_i\leqslant T_i\}$

$M_{2i}=\{P_i\mid P_i>T_i\}$

其中，$i=1,2,\cdots,6$。T_i 把指标 P_i 的取值范围分成 M_{1i} 和 M_{2i} 两个区间。至于 M_{1i} 和 M_{2i} 哪个是正常的取值区间，哪个是异常的取值区间，则取决于指标 P_i 是成本型指标还是效益型指标，成本型指标的取值越小越好，效益型指标的取值越大越好。因此当 P_i 是成本型指标时，M_{1i} 是正常的取值区间，而 M_{2i} 是异常的取值区间；若 P_i 是效益型指标，则 M_{2i} 是正常的取值区间，而 M_{1i} 是异常的取值区间。用 M_i 表示 P_i 指标的正常取值区间，有：

$$M_i=\begin{cases}M_{1i}, & \text{当 } P_i \text{ 是成本型指标时}\\ M_{2i}, & \text{当 } P_i \text{ 是效益型指标时}\end{cases}$$

定义 2(可接受的类) 如果该类的每一个度量指标取值都落在正常取值区间，那么该类的设计是可以接受的，称该类为可接受的类。

定义 3(最佳类的设计标准) 设最佳类为 $C_0(C_0\notin S)$，C_0 关于指标 P_j 的取值记为 $X_{0j}(j=1,2,\cdots,6)$，且满足：当 P_j 是效益型指标时，类 C_i 关于指标 P_j 的取值 $X_{ij}\in M_{1i}$，则 $X_{ij}^0=T$(阈值)，若 $X_{ij}\in M_{2i}$，则 $X_{ij}^0=X_{ij}$；当 P_j 是成本型指标时，类 C_i 关于指标 P_j 的取值 $X_{ij}\in M_{2i}$，则 $X_{ij}^0=T$(阈值)，若 $X_{ij}\in M_{1i}$，则 $X_{ij}^0=X_{ij}$。

于是有：

$$X_{ij}^0=\begin{cases}\max(X_{1j}^0,X_{2j}^0,\cdots,X_{nj}^0), & P_j \text{ 为效益型指标}\\ \min(X_{1j}^0,X_{2j}^0,\cdots,X_{nj}^0), & P_j \text{ 为成本型指标}\end{cases}$$

2.1 基于灰色关联分析类设计质量的评估过程

1)构造样本矩阵。类 C_i 关于指标 P_j 的取值记为 X_{ij}(其中 $i=1,2,\cdots,n$；$j=1,2,\cdots,6$)，据此可以构造出样本矩阵 $(X_{ij})_{n\times 6}$，其中该矩阵的各行代表观测的类，各列代表该类对应的度量指标的取值。

$$\begin{array}{c} \\ C_1\\ \\ C_n\end{array}\begin{array}{c}\begin{array}{ccc}P_1 & \cdots & P_6\end{array}\\ \begin{bmatrix}X_{11} & \cdots & X_{16}\\ \vdots & \ddots & \vdots\\ X_{n1} & \cdots & X_{n6}\end{bmatrix}\end{array}$$

2)设计最佳类。根据样本矩阵和各个指标的阈值，确定最佳类的设计标准。

3)构造决策矩阵。将最佳设计类 C_0 对于各个指标的值添加到样本矩阵 $(X_{ij})_{n\times 6}$ 中，从而构造出一个决策矩阵 $(X_{ij})_{(n+1)\times 6}$，该矩阵的第一行为类 C_0 对于各个指标的值，是其余各行(即各类)的参考标准。

$$\begin{array}{c} \\ C_0\\ \\ C_n\end{array}\begin{array}{c}\begin{array}{ccc}P_1 & \cdots & P_6\end{array}\\ \begin{bmatrix}X_{01} & \cdots & X_{06}\\ \vdots & \ddots & \vdots\\ X_{n1} & \cdots & X_{n6}\end{bmatrix}\end{array}$$

4)计算灰色关联度。邓聚龙教授[12]当初提出灰色关联度的计算的基本思想是：通过两个序列的相似和接近程度来判断两个序列之间的关系是否紧密。本文通过把每一个类都与最佳设计类进行关联度分析来考查类设计的优劣。与最佳设计类最接近、最相似的类是设计比较理想的类；与最佳设计类的关联度很小的类，往往是设计质量比较差的类，需要进行进一步改进。下面列举出了几种灰色关联度的计算公式：

$X_0=(x_0(1),x_0(2),\cdots,x_0(n))$

$X_1=(x_1(1),x_1(2),\cdots,x_1(n))$

$\cdots$

$X_i=(x_i(1),x_i(2),\cdots,x_i(n))$

$\cdots$

$X_m=(x_m(1),x_m(2),\cdots,x_m(n))$

1)邓氏关联度计算公式[12-13]：$\gamma(x_0(k),x_i(k))=$

$$\frac{\min\limits_i\min\limits_k|x_0(k)-x_i(k)|+\xi\max\limits_i\max\limits_k|x_0(k)-x_i(k)|}{|x_0(k)-x_i(k)|+\xi\max\limits_i\max\limits_k|x_0(k)-x_i(k)|}$$

$$\gamma(X_0,X_i)=\frac{1}{n}\sum_{k=1}^{n}\gamma(x_0(k),x_i(k))$$

其中，ξ 称为分辨系统，通常取值为 0.5。

灰色绝对关联度[12-13]：

$$\varepsilon_{0i}=\frac{1+|s_0|+|s_i|}{1+|s_0|+|s_i|+|s_0-s_i|}$$

其中，$s_0=\int_1^n(X_0-x_0(1))\,dt$，$s_i=\int_1^n(X_i-x_i(1))\,dt$，$s_0-s_i=\int_1^n(X_i^0-x_i^0)dt$。

ε_{0i}称为 X_0 与 X_i 的绝对关联度。灰色绝对关联度也反映了两个向量的几何形状的相似程度。

灰色相对关联度[12-13]：

设序列 X_0 与 X_i 的长度相同且初值皆不等于 0，X_0' 与 X_i' 分别为 X_0 与 X_i 的初值像，则称 X_0' 与 X_i' 的灰色绝对关联度为 X_0 与 X_i 的灰色相对关联度，简称为相对关联度，记为 r_{0i}。灰色相对关联度是应用灰色关联度模型定量测算变量之间关系的密切程度（或者影响大小），它描述系统发展过程中因素间相对变化的情况，也就是变化大小、方向及速度等指标的相对性。如果两个变量在系统发展过程中相对变化基本一致，则认为两者关联度较大，反之，则较小。当然，灰色相对关联度也反映了两个向量的空间位置的远近情况。

2）灰色综合关联度[12-13]：

$$\rho_{0i}=\theta\varepsilon_{0i}+(1-\theta)r_{0i}$$

其中，ε_{0i}是 X_0 与 X_i 的灰色绝对关联度，r_{0i}是 X_0 与 X_i 的灰色相对关联度，$\theta\in[0,1]$，一般情况下 $\theta=0.5$。如果对两个序列之间的绝对关联度比较关心，则 θ 值较大；如果对变化速率比较关心，则 θ 值较小。ρ_{0i}称作 X_0 与 X_i 的灰色绝对关联度，简称为综合关联度。灰色综合关联度既体现了折线 X_0 与 X_i 的相似程度，又反映了 X_0 与 X_i 相对于始点的变化速率的接近程度，是较全面地表征序列之间是否紧密的一个数量指标。

3　具体应用

本文借用了文献[11-14]给出的 Microsoft 公司写字板程序的度量结果数据（见表 1），并以此数据展开研究。

表 1　Microsoft 公司提供的一个写字板程序的度量数据

序号	类名	*WMC*	*DIT*	*NOC*	*CBO*	*RFC*	*LCOM*
1	CCSProperty Sheet	15	1	0	1	4	118
2	CMainFrame	25	1	0	4	21	134
3	CWordPadApp	32	1	0	7	13	210
4	CWordPadDoc	27	1	3	5	19	121
5	CWordPadView	51	1	3	9	43	301
6	CWordPadCntrItem	5	1	2	1	5	8
7	CColorMenu	4	1	0	2	3	2
8	CDateDialog	7	2	1	1	5	5
9	CDocOptPage	5	2	2	2	5	6
10	CFileNewDialog	5	2	1	2	5	4
11	CLocalcomboBox	6	1	0	2	3	5
12	CFontComboBox	13	2	1	2	6	22
13	CSizeComboBox	7	2	1	1	3	5
14	CFormatBar	14	1	3	3	9	25
15	CFormatParaDlg	5	2	1	2	5	4
16	CFormatTabDlg	19	2	1	2	13	101
17	CWordPadResizeBar	1	1	1	1	1	0
18	CInPlaceFrame	17	1	3	4	14	44
19	CKey	7	1	0	1	3	5
20	CListDlg	3	1	0	1	2	0
21	CUnit	3	1	0	2	2	0
22	CDocOptions	6	1	0	2	2	5
23	COptionSheet	4	2	2	2	1	0
24	CPageSetupDlg	3	2	1	2	3	1
25	CRulerBar	41	1	1	4	18	240
26	CRulerItem	20	1	0	2	12	90
27	CBigIcon	3	1	1	1	2	0
28	CSplashWnd	2	1	1	2	2	0
29	CEmbeddedItem	7	1	2	3	5	3
30	CUnitsPage	3	2	2	2	2	1
31	CFontDesc	1	1	0	3	1	0
	平均值	11.1	1.4	1.2	2.5	7.4	39.9

1）求出各个指标的阈值

软件度量指标的阈值一直是研究软件测量领域的热点问题，目前还没有很好的方法能确定一个普遍适用的软件度量指标的阈值，因此在研究各度量指标的阈值时，得到的结果往往并不一致。Lorenz[14]在 1993 年提出的 11 个面向对象度量中，规定 *WMC* 的阈值为 20，即 $0\leqslant WMC<20$，*DIT* 的阈值为 6。而 Rosenberg 等[15]将 *WMC* 的阈值定义为 100，将 *CBO* 的阈值定义为 5，将 *RFC* 的阈值定义为 100。而 Shatnawi 等[16]在对 Eclipse 的 3 个不同版本研究后得出，*WMC* 的阈值为 24，*CBO* 的阈值为 13，*RFC* 的阈值为 44。由于软件度量指标的阈值无法预先确定，从而阻止了软件度量学的进一步发展。然而越来越多的研究表明，软件度量指标的阈值应该从具体的数据集中导出，并在具体项目的局部范围内使用。Michele Lanza 等[17-18]提出了一个根据具体项目来确定 C&K 度量指标阈值的方法，该方法的基本思路如下：首先求出各个度量指标的平均值（AVG）和标准差（STDEV），平均值反映了该指标变量取值的集中趋势，而标准差反映了该指标变量的取值与其平均值的偏离程度；然后确定指标变量在该项目中的取值区间，其中最小的边界值为 *AVG*−*STDEV*，最大的边界值为 *AVG*+*STDEV*，非常高的值为（*AVG*+*STDEV*）∗1.5。通过这种方法可以确定表 2 所列的写字板程序中各个变量的取值范围。

表 2　写字板程序各个变量的取值情况

metrics	*AVG*	*STDEV*	Lower margin	Higher margin	Very high
WMC	11.4286	11.6702	−0.2416	23.0988	34.6482
DIT	1.3226	0.4752	0.8474	1.7978	2.6967
NOC	1.0645	1.0307	0.0338	2.0952	3.1428
CBO	2.5161	1.8050	0.7111	4.3211	6.4816
RFC	7.4839	8.7020	−1.2181	16.1859	24.2788
LCOM	47.0968	79.933	−32.8362	127.0298	190.5447

2）确定最佳类的设计标准

可以把各个指标的平均值作为最佳类的指标值，于是有：

$$C_0=(11.4286,1.3226,1.0645,2.5161,7.4839,47.0968)$$

对上述最佳类的设计标准必须调正，“高内聚、低耦合、低复杂性”是面向对象程序设计的基本要求。一般来说，类的 *WMC* 指标值越高，该类影响子类的程度就越高。因为该类的子类将继承它全部的方法，该类的方法数越多则该类越有可能成为针对某个专门用途设计的类，从而限制了该类的重要性，该类越复杂，发生故障的可能性就越大，所以 *WMC* 的取值越小越好。*DIT* 值和 *NOC* 值越小越好，值越小系统出现故障的概率就越小，但是 *DIT* 和 *NOC* 为 0 意味着该系统没有通过继承的方法来实现重用，重复的代码太多。而且如果 *NOC* 值越大，父类不恰当的抽象可能性就越大。如果一个类的子类数过大，那么有可能是对该类进行子类划分的一种误用，若一个类有大量的子类，则需要更多的时间和成本去测试该类所包含的方法，因此综合各方面考虑这两个值取 1 比较合适。同样地，*CBO* 的定义是与某个类相耦合的其他类的数目。一般情况下，若一个类的方法使用了另一个类中所定义的方法或变量，则这两个类是相互耦合的，根据面向对象的程序设计的原则，两个对象之间过度耦合是不利于模块化设计和软件重用的，一个类越独立，该类就越容易在另一个应用程序中重用。为了提高程序模块化设计和促进封装，对象类之间的耦合必须保持最小值，对象类之间的耦合越大，该类对设计中其他部分的变化就越敏感，因此维护起来就越困难。

对象之间的耦合越小越好，因此一般 *CBO* 取 0 时最佳。由于计算 *RFC* 值的过程既涉及到对象所包含的方法，也涉及到对象所包含的方法在执行过程中调用其他类中定义的方法，因此 *RFC* 也可以用于度量一个类与其他类之间的潜在通信情况。*RFC* 值越大，则对该类测试和调试就会变得越复杂，因为它需要测试成员花大量的时间去理解程序的结构。

一个类中方法之间的内聚是必要的，因为它能促进封装，内聚缺乏意味着该类有可能分割成了两个或更多个子类，低内聚会增加类的复杂性，从而增加开发过程中出错的可能性，因此 *LCOM* 值越小越好，其不可能取负值，所以 *LCOM* 取 0 时是最佳标准。

C_0=(11.4286,1,1,0,7.4839,0)

3)求各个类与最佳设计类 C_0 间的综合关联度的值(见表 3)

表 3 写字板程序中各个类的综合关联度

序号	类名	综合关联度	序号	类名	综合关联度
1	CCSProperty Sheet	0.558	17	CWordPadResizeBar	0.566
2	CMainFrame	0.695	18	CInPlaceFrame	0.839
3	CWordPadApp	0.673	19	CKey	0.880
4	CWordPadDoc	0.801	20	CListDlg	0.779
5	CWordPadView	0.695	21	CUnit	0.753
6	CWordPadCntrItem	0.703	22	CDocOptions	0.843
7	CColorMenu	0.763	23	COptionSheet	0.763
8	CDateDialog	0.822	24	CPageSetupDlg	0.662
9	CDocOptPage	0.685	25	CRulerBar	0.831
10	CFileNewDialog	0.721	26	CRulerItem	0.800
11	CLocalcomboBox	0.827	27	CBigIcon	0.753
12	CFontComboBox	0.916	28	CSplashWnd	0.641
13	CSizeComboBox	0.851	29	CEmbeddedItem	0.808
14	CFormatBar	0.884	30	CUnitsPage	0.662
15	CFormatParaDlg	0.721	31	CFontDesc	0.530
16	CFormatTabDlg	0.687			

从表 3 中可以得出各个类与最佳类设计标准的接近程度，从大到小的顺序为：

CFontComboBox＞CFormatBar＞CKey＞CSizeComboBox＞CDocOptions＞CInPlaceTrame＞CRoilerBar＞CLocalComboBox＞CDateDialog＞CEmbeddedItem＞CWorldPadDoc＞CRulerItem＞CListDlg＞COptionSheet＝CColorMenu＞CUnit＝CBigIcon＞CFileNewDialog＝CFormatParaDlg＞CWordPadCntrItem＞CWndPadView＝CMainFrame＞CFormatTabDlg＞CDocOptPage＞CWordPadApp＞CUnitsPage＝CPageSetupDlg＞CSplashWnd＞CWordPadResizeBar＞CCSPropertySheet＞CFontDesc。

4)分析结果

排在最后的 4 个类分别是：CSplashWnd，CWordPadResizeBar，CCSPropertySheet，CFontDesc，表明这 4 个类离最佳类设计标准相差较大。从对这 4 个类所包含的度量指标值可以发现：类 CSplashWnd，CWordPadResizeBar，CFontDesc 中的 *LCOM* 值都为 0，*WMC* 值为 1 或 2，*CBO* 值也不高，仅为 1 或 2。从表面上看，这 3 个类的内聚性比较好，复杂性不高，耦合性低，但是从这些类所包含的所有度量指标值来看这种内聚性好、复杂性不高是以牺牲方法数为代价的，也就是说这 3 个类能完成的功能很少，包含的方法数比较少，但即使完成较少的功能，它们的 *CBO* 值也不为 0，说明其严重依赖于其他类来完成功能。这表明它们的设计是较差的。类 CCSPropertySheet 的度量指标值为(15，1，0，1，4，118)，*WMC* 为 15，*LCOM* 为 118，这说明该类是比较复杂且内聚性较差的。因此该类的设计也是很差的。

从排在前面的几个类来看，类 CFontDesc 排在最后，它与标准类 C_0 的接近程度与相似程度都比较差，说明它的设计质量比较差，需要重点关注。

5)对比分析基于 UML 类图的设计质量评价方法(见表 4)

表 4 几种面向对象软件系统设计质量评价方法的对比分析

方法	模型基础	度量对象	度量方法的主要特点
文献[19]方法	C&K	UML 类图中类的复杂性	不仅度量了类的静态复杂性，而且进一步考查了软件系统的动态复杂性
文献[20]方法	C&K	UML 类图结构复杂性	通过考查类与类之间的关系，构造出“类依赖图”并提出一种度量 UML 类图结构复杂度的方法
文献[21]方法	C&K	UML 类图的复杂性	通过 UML 模型考查类、封装、继承、多态、内聚、耦合的复杂性，提出相应的度量准则并最终度量软件系统的复杂性
文献[22]方法	C&K	UML 类图中类设计缺陷的检测	应用面向对象度量技术将设计缺陷量化为度量指标，找出存在设计缺陷的类并改进相应的设计实体
本文方法	C&K	UML 类图中类设计质量	在初期确定一个类设计质量标准，用 C&K 度量方法获得各个类的设计参数，然后将所获得的参数与类设计质量标准进行对照，从而确定类设计质量的优劣

结束语 本文首先分析软件度量研究学的发展历程，着重介绍面向对象软件度量方法中最著名的 C&K 度量方法，并介绍了该方法的 6 个度量指标。由于开发初期能够获得的信息比较少，针对如何利用开发初期具有的信息(如 UML 用例图、UML 类图、UML 序列图等)来评估类设计的质量，本文提出了一种基于灰色关联分析的类设计质量评估方法，评估的依据是：在初期确定一个类设计质量标准，用 C&K 度量方法获得各个类的设计参数，然后将获得的参数与类设计质量标准进行对照，从而确定类设计质量的优劣。该方法为软件设计人员提供了参考，使得设计质量低劣的类以及将来有可能会发生缺陷的类在早期就被识别并纠正。

参 考 文 献

[1] MARTIN J. An Information Systems Manifesto[M]. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1986.

[2] HU W S, ZHAO M, WU S Y, et al. Requires analysis based on software maintainability [C] // 2014 International Conference Reliability, Maintainability and Safety(ICRMS). 2014: 354-357.

[3] HECTOR M. Olagu_assessing mainability information theory metrics and iterative software processes [D]. Huntsville: Alabama, 2006: 7-33.

[4] CHIDAMBER R, KEMERER F. A metrics suite of object-oriented design [J]. IEEE Transactions on Software Engineering, 1994, 20(6): 467-493.

[5] ABREU F B E, CARAPUCA R. Object-Oriented software engineering: measuring and controlling the development process[C] // Proceedings of the 4th International Conference on Software Quality. McLean, Va, USA, 1994: 1-8.

[6] TEGARDEN D P, SHEETZ S D, MONARCHI D E. A software complexity model of object-oriented system [J]. Decision Support Systems, 1995, 13(3/4): 241-262.

[7] ETZKORN L, BANSIYA J, DAVIS C. Design and code complexity metrics for OO classes [J]. Journal of Object Oriented Programming, 1999, 12(1): 35-40.

[8] BANSIYA J, DAVIS C G. A Hierarchical Model for Object-Oriented Design Quality Assessment[J]. IEEE Transactions on Software Engineering, 2002, 28(28): 4-17.

[9] SURESH Y, KUMAR L, RATH S K. Statistical and Machine Learning Methods for Software Fault Prediction Using CK Metric Suite: A Comparative Analysis[C] // ISRN Software Engineering. 2014: 1-15.

[10] DUBEY S K, RANA A. Assessment of Maintainability Metrics for Object-Oriented Software System[J]. ACM SIGSOFT Software Engineering Notes Page1, 2011, 36(5): 1-7.

[11] LI X K, LIU Z T, PAN B, et al. Software and Research on Measure Experiments [J]. Chinese Journal of Computers, 2000, 23(11): 1220-1225. (in Chinese)
李心科，刘宗田，潘飚，等. 一个面向对象软件度量工具的实现和度量实验研究[J]. 计算机学报，2000，23(11)：1220-1225.

[12] 刘思峰，谢乃明. 灰色系统理论及其应用(第六版)[M]. 北京：科学出版社，2016.

[13] LIU S F, CAI H, YANG Y J, et al. Advance in grey incidence analysis modelling[J]. Systems Engineering-Theory & Practice, 2013, 33(8): 2041-2046. (in Chinese)
刘思峰，蔡华，杨英杰，等. 灰色关联分析模式研究进展[J]. 系统工程理论与实践，2013，33(8)：2041-2046.

[14] LORENZ M. Object-Oriented Software Development: A Practical Guide [M]. Englewood Cliffs, N. J.: PTR Prentice Hall, 1993.

[15] ROSENBERG L, STAPKO R, GALLO A. Object-oriented Metrics for Reliability [C] // Presentation at IEEE International Symposium on Software Metrics. 1999.

[16] SHATNAWI R, LI W, SWAIN J, et al. Finding software metrics threshold values using ROC curves [C] // Journal of Software Maintenance and Evolution: Reseach and Practice. Res (Pract 2010). 2010: 1-16.

[17] D'AMBROS M, LANZN M. Reverse engineering with logical coupling[C] // IEEE Computer Society Proceedings of the 13th Working Conference on Reverse Engineering. Washington, D C, USA, 2006: 189-198.

[18] KHAN T, BARTHEL H, EBERT A, et al. Visualization and Evoluton of Software Architectures[C] // Visualization of Large and Unstructured Data Sets Workshop. 2011: 25-42.

[19] ZHANG Y, TAO J, QIAN L Q. A Metrics Suite for Class Complexity Based on UML [J]. Computer Science, 2002, 29(10): 128-132. (in Chinese)
张涌，陶隽，钱乐秋. 一种基于 UML 的类复杂性度量方法[J]. 计算机科学，2002，29(10)：128-132.

[20] FU X D, ZOU P. A Measurement Method of structural complexity for UML class diagrams[J]. Computer Applications, 2007, 27(b06): 302-307. (in Chinese)
付晓东，邹平. 一种 UML 类图结构复杂性度量方法[J]. 计算机应用，2007，27(b06)：302-307.

[21] JING F B. Study on Software Complexity Measurement Method and Tool Based on UML [D]. Chongqing: Chongqing University, 2015. (in Chinese)
景富波. 基于 UML 的软件复杂性度量方法和工具的研究[D]. 重庆：重庆大学，2015.

[22] XIE L M. A Study on Class Diagram Design Flaws Detection [D]. Shanghai: East China Normal University, 2011. (in Chinese)
谢玲梅. 类图设计缺陷的检测研究[D]. 上海：华东师范大学，2011.

(上接第 130 页)

[14] COUSOT P, COUSOT R. A galois connection calculus for abstract interpretation[C] // ACM Sigplan-Sigact Symposium on Principles of Programming Languages. ACM, 2014: 3-4.

[15] COUSOT P, COUSOT R. Abstract interpretation: past, present and future[M]. ACM, 2014.

[16] BALL T, PODELSKI A, RAJAMANI S K. Boolean and Cartesian abstraction for model checking C programs[M] // Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2001: 268-283.

[17] HENZINGER T A, JHALA R, MAJUMDAR R, et al. Lazy Abstraction[C] // POPL. 2002: 58-70.

[18] FOSCHER J, JHALA R, MAJUMDAR R. Joining dataflow with predicates[J]. AcmSigsoft Software Engineering Notes, 2005, 30(5): 227-236.

[19] LALIRE G, ARGROUND M, JEANNET B. Interproc[OL]. http://pop-art.inrialpes.fr/people/bjeannet/bjeannet-forge/interproc.

[20] MAUBORGNE L, RIVAL X. Trace Partitioning in Abstract Interpretation Based Static Analyzers [M] // Programming Languages and Systems. Springer Berlin Heidelberg, 2005: 5-20.