计算机科学 1999Vol. 26№. 5

面向对象度量综述

Introduction to Object-Oriented Metrics

袁望洪 谢 涛 陈向葵

(北京大学计算机科学技术系 北京 100871)

Abstract Focusing on software quality control has spurred the object-oriented approach to software development and increased the demand for software metrics. Object-oriented metrics are an integral part of object technology and play an important role in software development. The differences between object-oriented software development and traditional software development make traditional metrics ill-applicable in object-oriented system. Therefore, new metrics suitable for object-oriented software development are needed, and the research and application of object-oriented metrics are increasing focused on. This paper is a summarization on object-oriented metrics. It presents metrics theory and introduces related metrics on object-orientation and software reuse.

Keywords Software metrics, Object-oriented, Software complexity, Software project management

软件是信息技术的核心,因而管理人员对软件质量控制越来越重视。这种重视引起了两种效果:(1)要求新的、更好的软件开发方法和技术;(2)在软件开发过程中,进行软件度量。软件度量可以帮助管理人员控制、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。软件度量的必要性和重要性已为软件界所认同。作为90年代的领先技术,面向对象的技术已经在软件产业中得到了广泛的运用,面向对象产品也得到了迅猛的发展。面向对象度量是对象技术不可分割的一部分,其在面向对象软件开发中的应用具有以下的功用[19]:

· 定量理解系统的体系结构和详细设计,利用度量的反馈信息以构造更好的系统。

· 提供面向对象项目开发成本估算和进度预计的良好基础。

· 以系统的生产率、可维护性和可复用性等定量化评估对象技术在软件开发中的应用。

度量的研究可以追溯到70年代,其时,最为流行的软件开发方法是传统的结构化方法。早期著名的度量,如McCabe的环计数(cyclomatic number)[26]、Halstead的软件科学(software sciences)[20]和Albrecht的函数点(function point)[1]等等,也都是针对传统的开发方法。制定的一些软件质量标准也是针对传统的结构化的软件,如ISO9000-3标准[2,4]。

面向对象软件开发在软件生存周期、系统结构、复用以及项目管理等方面同传统的软件开发有着显著的区别,这些区别很大程度上影响了软件度量。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,也不足以刻划、评估、预示面向对象系统的质量[3]。因而面向对象度量的研究在学术界得到了普遍的重视[34,8],并在产业界具有良好的应用前景。

本文是一篇关于面向对象度量的综述。

一、面向对象软件开发的特点及其对度量的影响

尽管面向对象软件开发与传统的软件开发有相似之处,但是两者之间还是有着显著的区别,主要表现在以下几方面:· 递归/叠代的生存周期;· 多重抽象层次的系统结构;· 复用的支持;· 项目管理。

面向对象系统生存周期的各个阶段有着无缝的衔接和高度的叠代[9~11]。因而需要更好的生存周期模型和方法论加以支持,并且更强调复用和质量保证。从度量角度看,要求度量的生存周期的早期可用;另一方面,由于各阶段没有明显的界线,要确认

度量能用于生存周期的哪些阶段。

面向对象系统的结构充分反映了面向对象范型所支持的抽象概念，具有多重的抽象层次，包括系统、子系统、类和方法；强调的是对等实体之间的关系而不是传统控制流所决定的层次分解关系。虽然有些传统的度量准则能用于面向对象系统，如对等实体关系可以用扇入(fan-in)和扇出(fan-out)来度量，类的大小可以用属性和方法的带权和进行度量；但是，另一方面，传统度量用到面向对象系统中有很大的局限性，譬如，子类大小的度量就很成问题，尤其是涉及到多继承时。更为重要的是，有些面向对象概念，如类之间的继承结构就无法用现有的度量准则进行度量。

复用能有效地减少项目的开发成本，Grahan 指出复用策略能减少 20%的开发成本[16]。传统方法的复用主要表现在函数库的代码复用，代码复用的级别较低而且难度较大。面向对象方法对复用提供了良好的支持，降低了复用的难度并提高了复用的程度[26]。复用可用于面向对象系统生存周期的各个阶段，表现为：分析复用、设计复用、代码复用以及测试复用等等。可复用的构件有多种形态：模式、框架、类树以及类等。面向对象的复用遵循所谓的"开放/封闭原则"：构件经过测试放入构件库之后，其内部细节对外封闭，但外部接口开放，可用继承和参数化等方式进行扩展。面向对象复用引起了成本估算和进度安排的变化，软件开发不再是从头开始，而是涉及到构件的查找、理解、扩展和组装等新的开发过程。这些变化都要求有新的度量方法和准则，面向对象复用的度量还是一个有待进一步研究的课题。

面向对象系统的新特点带来了面向对象项目管理的新问题：(1)所选用的生存周期模型；(2)合适的开发组员结构；(3)复用方法的考虑(for-reuse 和 with-reuse)[25]；(4)构件库的组织与管理，就以上问题，面向对象的项目管理要求新的度量方法和准则，以保证项目进度、软件质量以及成本估算。

面向对象软件开发的上述不同及其对度量的影响，导致了对新的面向对象度量方法和准则的需求，面向对象度量的理论研究和实践应用成了软件工程研究的热点之一。

二、度量理论

对软件度量的批评之一就是度量缺乏理论基础。Kearney，Vessey 和 Weber 等人批评软件度量没有坚实的理论基础并缺少合适的性质[25,35]，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[32,36]。Zuse 和 Fenton 等人认为度量准则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准则是否适合特定的环境[12]。

Baker 和 Fenton 等人认为度量涉及以下四项活动[5,12,37]，度量理论的研究需要为这些活动提供理论基础：

(1)确认所关心的软件属性；

(2)建立软件属性的经验关系系统；

(3)通过度量准则把经验关系系统映射到形式关系系统；

(4)评估度量准则。

软件属性可分为内部属性和外部属性。内部属性，如软件大小、控制流、耦合度等，通常描述软件结构上的复杂性。内部属性一般有清晰的定义并能进行客观的度量。外部属性，如复杂性、可维护性、可读性等，涉及到人和环境等外部因素。外部属性的度量才能真正给出人们所需的可靠数据并预示软件开发行为。不幸的是，外部属性常缺乏清晰的定义，也无法进行直接的客观度量，只能通过内部属性进行推测性的度量。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。

经验关系系统捕捉了软件属性直觉(经验)上的概念。譬如，设计者通常认为一个有 20 个方法的类比另一个只有 10 个方法的类复杂。经验关系系统可以表示为[37]：

$$D=(A,R_i,O_j)\quad i=1\text{ to }n,j=1\text{ to }m$$

其中：A 是非空的软件属性集合；R_i 是 A 元素之间的经验关系，如大于、更复杂等；O_j 是 A 元素之间的二元操作。

建立经验关系系统之后，就要找到一个度量准则 M，把经验关系系统映射到形式关系系统。形式关系系统可以定义为[37]：

$$F=(C,S_i,B_j)\quad i=1\text{ to }n,j=1\text{ to }m$$

其中：C 是非空集，如实数集；S_i 是 C 元素之间的形式关系，如>、<、=等；B_i 是 C 元素之间的二元形式操作，如+、-、*等；

度量准则 M 是 D 到 F 的同构映射，满足：

$\forall a\in A,\exists c\in C$，使得 $M(a)=c$；

$\forall a,b\in A$，$R_i(a,b)\Leftrightarrow S_i(M(a),M(b))$；

$\forall a,b\in A$，$M(aO_jb)\Leftrightarrow M(a)B_jM(b)$；

为了增强软件度量的可用性，许多研究人员提出了软件度量应有的一些性质[11,23]：(1)有效性(va-

lidity)，(2)可靠性(reliability)，(3)健壮性(robustness)和(4)易于应用和计算。针对面向对象的软件生存周期和质量保证，Brito E Abreu 提出了以下七条标准[4]：(1)度量应该形式化定义，(2)非大小度量准则应与系统大小无关；(3)度量应选择合适的衡量(scale)类型；(4)度量应在生存周期的早期可用；(5)度量应具有向下可扩展性；(6)度量应易于计算；(7)度量应独立于语言。

随着软件度量纳入更严密的理论体系，有必要将度量评估纳入形式化范畴。Weyuker 提出了一组形式化评估软件度量性质的定理[16]：

性质 1：(∃ P)(∃ Q)(|P|≠|Q|)。该性质表明所有程序的度量值不会都相同。

性质 2 对于非负数 c，只有有限个程序具有度量值 c。该性质保证度量准则有足够的精度。

性质 3：(∃ P)(∃ Q)(P≠Q & |P|=|Q|)。该性质表明度量准则不能太精细，以致所有程序有不同的度量值。

性质 4：(∃ P)(∃ Q)(P≡Q &|P|≠|Q|)。该性质表明功能相同的两个程序，由于实现细节不同，其度量值不一定相同。

性质 5：(∀ P)(∀ Q)(|P|<=|P;Q|&|Q|<=|P;Q|)。该性质表明度量准则具有单调性，两个程序合并后的度量值不小于其中任一个。

性质 6：(∃ P)(∃ Q)(∃ R)(|P|=|Q|&|P;R|≠|Q;R|)；(∃ P)(∃ Q)(∃ R)(|P|=|Q|&|R;P|≠|R;Q|)。该性质表明程序 P 和 R 的交互与 Q 和 R 的交互不同，合并后的程序 P;R 和 Q;R 的度量值不同。

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|。该性质表明语句顺序影响度量值。

性质 8：如果 P 是 Q 的重命名，则|P|=|Q|。该性质表明对程序中的变量进行换名不会影响度量值。

性质 9：(∃ P)(∃ Q)(|P|+|Q|<|P;R|)。该性质表明两个程序的交互增加了复杂性，使其度量值不小于两者之和。

Weyuker 所提出的性质定理并不完美。考虑到心理复杂性(psychological complexity)，变量的命名会影响到复杂性，因而其中的性质 8 并不满足；Fenton 认为这组定理基于的程序复杂性缺乏清晰的定义[12]；Zues 批评它们与衡量原则不一致[17]；Chernıavsky 和 Smith 认为它们只给出了良好度量准则的必要条件，但不是充分条件[7]。但是，正如 Gustafson 和 Prasad 所指出的，形式化分析超越了早期定义的非形式化的性质，并提出了用于评估度量的语言，因而 Weyuker 的这组性质定理仍不失为度量评估的良好出发点[18]。

三、有关面向对象的度量

Tegarden 等人[21]试图把传统的软件度量，如代码行数、Halstead 的软件科学[20]和 Albrecht 的函数点[1]应用到面向对象系统中。但是这些传统的软件度量应用到面向对象系统中有一定的局限性，而且对某些面向对象特性根本无法度量，这些面向对象特性包括[22]：• 继承结构的高度和宽度；• 系统中类的数目；• 协作类的数目；• 系统中代码的复用比例。

上述面向对象特性需要专门的适合面向对象的度量，Chidamber 和 Kemerer 提出了六条适于面向对象设计的度量准则[5]：

(1)类方法的带权和(Weighted Methods Per Class)WMC：如果类 C 定义了 n 个方法 $M_1,\cdots,M_n$，设 $c_1,\cdots,c_n$ 是这些方法的复杂性，则 WMC $=\sum_{i=1}^{n}c_i$。WMC 揭示了开发和维护类的时间和精力。类的 WMC 越大，对子类的影响就越大，其通用性和可复用性就越差。

(2)继承树深度(Depth of Inheritance Tree)DIT：DIT 定义为类在继承树中的最大深度。类的 DIT 越大，其继承的方法就越多，因而很难预示其行为；但另一方面所继承的方法可复用性越大。越深的继承树涉及了更多的类和方法，从而设计复杂性越大。

(3)子类数目(Number of Children)NOC：NOC 定义为该类的直接子类数目。类的 NOC 越大，其可复用性越大，但其影响就越大，从而需要更多的测试。

(4)对象类的耦合度(Coupling between Object Classes)COB：COB 定义为与该类耦合的类数目。类的 COB 太大，有害于模块化设计并限制了复用，其改变对系统的其他部分影响很大，维护更加困难，为了提高模块性和封装性，应使对象类的耦合度最小。

(5)类响应数目(Response For a Class)RFC：RFC=|RS|，其中 RS 是该类的响应集合，RS={M}∪$_{all\ i}${R_i}，其中的{R_i}是方法 i 调用的方法集，{M}是类的方法集。类的的 RFC 越大，就越复杂，测

试和调试就越难。

(6)类方法缺乏内聚度(Lack of Cohesion in Methods,LCOM)。如果类C定义了n个方法M_1,…,M_n,设$\{I_i\}$是方法M_i所用的实例变量集,$P=\{(I_i,I_j)|I_i\cap I_j=\varnothing\}$,$Q=\{(I_i,I_j)|I_i\cap I_j\neq\varnothing\}$,$LCOM=|P|-|Q|$如果$|P|>|Q|$,否则为0。LCOM越大,类的封装性就越差,应分成几个子类;LCOM越小,类的内聚度就越大,封装性越好。

Chidamber和Kemerer就Weyuker所提出的性质定理[16]对上述六条准则进行了形式化评估,结果发现这六条准则均不满足性质7,性质7所涉及的顺序改变对面向对象设计来说是完全不合适的,因为类及对象的方法定义顺序是无关的[7]。Chidamber和Kemerer将这些度量准则建立在度量理论之上,结合了Bunge的实体论概念[5,6],对其中的一些准则给出了清晰的数学定义。但Graham怀疑Bunge实体论概念用到面向对象上的可靠性[17],实体论概念认为对象由其属性定义,但是在对象技术中,对象可以改变属性但不影响其内部标志。

除了上述工作之外,Lorenz、Rajaraman和Lyu等人还提出了一些适于面向对象的度量准则[27,13]。这些准则能对面向对象的类、类接口、继承以及系统级关系等方面进行有效的度量。

四、有关复用的度量

在软件开发过程中,软件复用在提高软件生产率和软件产品质量上有着巨大的潜力。软件开发者在利用软件复用方法进行开发时,需要去度量开发进度,找出最有效的复用策略,以及测度出复用的潜力和预期的收益,另外还要能够识别出现存系统中的可复用构件等。面向对象技术为软件复用提供了更强有力的支持,带来了更多的复用机会,软件复用度量也成为面向对象度量的重要方向之一。

软件复用度量基于传统度量和面向对象度量的方法,构造出适当的度量模型来评价软件开发过程中与复用相关对象和活动的特性。KarIsson等人认为软件复用度量研究涉及三方面:产品度量、过程度量和成本估计[24]:

·产品度量主要用于判定构件的可复用性和质量,包括先期度量和后期度量。先期度量基于检查表和静态度量,在构件的开发期进行;而后期度量则基于复用者的反馈信息,在构件已被复用后进行。

·过程度量用于判定复用对生产率、质量和开发时间的作用,可以在不同级别上度量,包括构件级、产品族级、项目级、机构级。

·成本估计用于估计面向复用和基于复用的开发成本。面向复用的开发成本指开发可复用构件的成本;而基于复用的开发成本指用来自构件库的一个或多个可复用构件开发系统的成本。

有关复用的度量分为两类:复用度量和可复用性度量。复用度量(Reuse Metrice)用来判定复用活动对软件开发的生产力、质量和开发时间的作用。可复用性度量(Reusability Metrics)用来判定一个构件的可复用性和质量。

复用度量的主要动机有[14]:

·监控在过去时间内已复用的数量;

·为判定复用对软件生产力和质量的作用提供基础;

·加深对开发可复用软件的理解力;

·判定特定行为对复用数量的作用。

可复用性度量的主要动机有[31,24]:

·识别出遗产系统中的领域知识和有用的产品,为构件提取提供基础;

·对构件库中的可复用构件进行客观的控制,保证构件库中保存的是高可复用性和高质量的构件;

·复用库中如果能同时包含构件的复用和质量度量的信息,可以给使用构件库中构件的复用用户提供有价值的帮助。

当前已经存在不少复用度量的模型和准则,William Frakes和Carol Terry在文[14]中总结了这些模型和准则:

·经济模型类(Economic Models)。典型的有成本/生产力模型(Cost/Productivity Models)、投资质量(Quality of Investment)和商业复用度量(Business Reuse Metrics)等。

·成熟度模型(Maturity Model)。典型的有复用成熟度模型(Reuse Maturity Model)和复用能力度模型(Reuse Capability Model)等。

·复用比率模型(Reuse Ratio Model)。典型的有复用水平(Reuse Level)、复用部分(Reuse Fraction)和面向对象度量和模型(Object-Oriented Metrics and Models)等。

·复用潜力度量和模型(Reuse Potential Metrics and Models)。典型的有Ada的复用度量(Measuring Reusability for Ada)和生存周期对象的复用预计(Reuse Prediction)等。

到目前为止,软件工程界仍然缺乏有效的可复

用性度量指标，Jeffrey S Poulin 总结了现有的几种重要的经验度量方法[9]：REBOOT 方法，NATO 方法，Chen 和 Lee 方法，Caldiera 和 Basili，Hislop 方法，Torres 和 Samadzadeh 方法和 Mayobre 方法等。

以上的可复用性度量方法大都是根据构件的内部属性来得出度量结果的，如果研究发现构件所处的上下文(context)也和其内部属性一样影响构件的可复用性，就必须在度量构件的可复用性的同时，考虑构件的内部属性，及其所处的领域属性和环境属性[1]。

五、今后的研究方向

面向对象软件的度量目前仍然只是处于探讨阶段，尚需从面向对象软件的特点来挖掘具有理论价值和实用价值的度量方法和准则。我们认为面向对象软件度量还有待于在以下几方面进行进一步研究：

·生存周期各阶段的应用：度量应是软件生存周期各阶段活动的一部分，以便更好地对软件开发进行评估、控制和安排。现有的许多度量准则只适于设计阶段和编码阶段，有必要将度量应用到生存周期的早期，因为早期的活动对整个开发过程影响最大。

·软件属性清晰严格的定义：许多软件属性，尤其是外部属性，如复杂性、可理解性、可维护性以及质量等等都缺乏严格的定义；有关的经验关系系统还有待研究，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"[13]。

·形式化评估度量准则：Weyuker 所提出的形式化定理有待进一步完善，而且这组定理是针对传统度量准则的。随着面向对象度量准则的研究和应用，有必要提出一组适合面向对象度量准则的性质，并提供形式化评估的定理。

·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。

·自动化支持：Grady 认为真正有用的度量具有自动化功能[15]，否则，就无法收集和计算度量所用的数据，因而对给定的度量准则，有必要提供相应的工具支持。

参 考 文 献

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

2 Baker A L, et al. A Philosophy for Software Measurement, J. System Software, 1990, 12, 277～281

3 Basili Victor R, et al. A Validation of Object-Oriented Design Metrics as Quality Indicators, IEEE Trans. Software Eng., 1996, 22(10): 751～761

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

5 Bunge M. Treatise on Basic Philosophy, Ontology I The Furniture of the World. Boston: Riedel, 1977

6 Bunge M. Treatise on Basic Philosophy, Ontology I - The World of System. Boston: Riedel, 1979

7 Chernlavsky J C, Smith C H. On Weyuker's Axioms for Software Complexity Measures, IEEE Trans Software Eng., 1991, 17: 636～638

8 Chidamber S R, Kemerer C F. A Metrics Suite for Object Oriented Design, IEEE Trans. Software Eng., 1994, 20(6)

9 Coad Peter, Yourdon Edward. Object-Oriented Analysis. Yourdon Press, 1990

10 Coad Peter, Yourdon Edward. Object-Oriented Design. Yourdon Press, 1991

11 Emory C W, Cooper D R. Business Research Methods, 4th ed., Richard D, Irwin, Homewood, IL, 1991

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

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

14 Frakes W. Software reuse, quality, and productivity. In: ISOE Proceedings. Juran Institute, Inc. 1992

15 Grady R B. Practical Software Metrics for Project Management and Process Improvement, Prentice Hall, Englewood Cliffs, NJ, 1992

16 Graham I. Object-Oriented Methods. Addison-Wesley, Wokingham, UK, 1991. 410

17 Graham I M. Migrating to Object Technology. Addison-Wesley, Wokingham, UK, 1995

18 Gustafson D A, Prasad B. In Denvir T, et al. Eds. Properties of Software Measures, Formal Aspects of Measurement. NY: Springer-Verlag, 1991

19 Henderson-Sellers B. Object-Oriented Metrics—Measures of Complexity, Prentice Hall PTR, 1996

20 Halstead M H. Elements of Software Science. New York: Elsevier North-Holland, 1977

21 ISO 9000-3, Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software, ISO/IEC, 1991

22 Jacobson I. et al Object-Oriented Software Engineering A Use Case Driven Approach. Addison-Wesley, Reading, MA, 1992. 524

23 Jones C Applied Software Measurement. Assuring Productivity and Quality McGraw-Hill, NY, 1991

24 Software Reuse: A Holistic Approach-Measuring the Effect of Reuse Chapter. Even-Andre Karlsson, ed. Chichester; New York. Wiley, c1995, published by Wiley & Sons, Ltd Baffins Lane, Chichester Sussex PO 191 UD, England

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

26 Lewis J A, et al. On the Relationship between the Object-Oriented Paradigm and Software Reuse: An Empirical Investigation. J. Object-Oriented Programming, 1992, 5(4) 35～41

27 Lorenz M Object-Oriented Software Development: A Practical Guide, Prentice Hall, NJ, 1993

28 McCabe T J. A Complexity Measure, IEEE Trans Software Eng., 1976, 2(4): 308～320

29 Mili Hafedh, et al. Reusing Software: Issues and Research Directions, IEEE Trans. On SE, 1995, 21(6): 528～562

30 Piper Joanne C, Barner Wanda L The RAPID Center Reusable Components (RSCs) Certification Process. U. S. Army Information Systems Software Development Center-Washing, Ft Belvoir, VA

31 Poulin Jeffey S. Measuring Software Reusability Third International Conference on Software Reuse, 1994

32 Prather R E An Axiomatic Theory of Software Complexity Measures, Comput. J., 1984, 27 340～346

33 Rajaraman C, Lyu M R. Some Coupling Measures for C++ Programs. In Procs. TOOLS USA'92, Prentice Hall, Englewood Cliffs, NJ, 1992. 225～234

34 Tegarden D P, Sheetz S D. Object-Oriented System Complexity: An Integrated Model of Structure and Perceptions. In OOPSLA'92 Workshop on Metrics for Object-Oriented Software Development Washington DC, 1992

35 Vessey I, Weber R. Research on Structured Programming. An Empiricist's Evaluation, IEEE Trans. Software Eng., 1984, 9-10: 394～407

36 Weyuker E Evaluating Software Complexity Measures, IEEE Trans. On Software Eng., 1988, 14: 1357～1365

37 Zues H. Software Complexity: Measures and Methods, Walter de Gruyter, Berlin, 1990, 605

(上接第 20 页)

包含(IsMadeOf):是与 Element 类之间的关系,标识构成软件资产的元素;

需要(Requires):是与 Asset 类之间的关系,标识相互依赖的软件资产;

另见(SeeAlso):是与 Asset 类之间的关系。标识对本软件资产的理解、使用等可能有帮助的其它软件资产;

由…制作(WasCreateBy):是与 Organization 类之间的关系。标识软件资产的制作者。

2 Element

Element 类中定义的关系包括:

由…提供(ProvidedBy):是与 Organization 类之间的关系。标识元素的提供者。

3. Library

Library 类中定义的关系包括:

联系(ContactIs):是与 Organization 类之间的关系。标识对库负责的组织。组织可能是个人。

结论:UDM 为在库间传送软件资产提供了一个通用的表示,消除了每个库为所有其它库产生映射的需要。UDM 也标识出了描述可复用软件资产的一些重要的特性,如价格、认证、语言、保证、目标环境、位置、作者、内容等。UDM 还提供了一个包括 UDM 属性、关系和属性值列表的术语集,库可以将这个术语集与各自的术语建立联系。

参 考 文 献

1 RIG Basic Interoperability Data Model (BIDM): [RPS-0001]: Reuse Library Interoperability Group, April 1993

2 RIG Uniform Data Model for Reuse Libraries (UDM): [RPS-0002]. Reuse Library Interoperability Group, January 1994

3 RIG Glossary: [SDR-0001]. Reuse Library Interoperability Group, April 1993