类规模对面向对象度量易变性预测能力的潜在混和效应:一个元分析

卢红敏 周毓明 徐宝文

(南京大学计算机科学与技术系 南京 210023)

(南京大学软件新技术国家重点实验室 南京 210023)

摘 要 最近的研究表明,类的规模对面向对象(OO)度量的易变性预测能力存在很强的混和效应,因此需要将其作为一个混和变量来考虑,否则有可能会得到误导性的结果.然而,先前的研究仅仅分析了一个软件系统,因此不清楚这个结论是否可以推广到其他系统上.为解决此问题,文中在 102 个 Java 软件系统的基础上利用元分析技术检查类的规模对 55 个 OO 度量和易变性之间关联关系的潜在混和效应.对每一个 OO 度量,首先在单个系统上分别计算在控制规模和不控制规模的两种情况下它与易变性的关联强度.然后,利用随机效应元分析模型计算在所有系统上且分别在这两种情况下它与易变性的平均关联强度.最后,在此基础上利用统计方法检测类规模的潜在混和效应.实验结果表明类规模的混和效应是广泛存在的,因此在验证 OO 度量的易变性预测能力时确实需要将其作为一个混和变量来考虑.

关键词 类规模;易变性;混和效应;面向对象;元分析

中图法分类号 TP311 **DOI 号** 10.3724/SP.J.1016.2015.01069

The Potentially Confounding Effect of Class Size on the Ability of Object-Oriented Metrics to Predict Change-Proneness: A Meta-Analysis

LU Hong-Min ZHOU Yu-Ming XU Bao-Wen

(*Department of Computer Science and Technology, Nanjing University, Nanjing* 210023)

(*State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing* 210023)

Abstract Recent research shows that class size has a strong confounding effect on the ability of object-oriented (OO) metrics to predict change-proneness and hence suggests that it should be considered as a confounding variable. Otherwise, misleading analysis results would be obtained. However, this conclusion is drawn from only one software system and it is not clear whether it can be generalized to other systems. To attack this problem, based on 102 systems, this paper employs statistical meta-analysis techniques to examine the potentially confounding effect of class size on the associations between 55 OO metrics and change-proneness. For each metric, we first compute its degrees of association with change-proneness under controlling/not controlling for class size on individual systems. Then, we employ random-effect models to compute their average degrees of associations under these two cases over all systems. Finally, we apply statistical methods to test whether class size has a confounding effect. Our experimental results indicate that the confounding effect of class size in general exists and hence confirm that we should consider it as a confounding variable when validating OO metrics on change-proneness.

Keywords class size; change-proneness; confounding effect; object-oriented; meta-analysis

收稿日期:2013-05-18;最终修改稿收到日期:2014-11-21.本课题得到国家"九七三"重点基础研究发展规划项目基金(2014CB340702)、国家自然科学基金(91318301,61321491,61300051,61272082)和江苏省自然科学基金(BK20130014)资助.**卢红敏**,女,1975 年生,博士,主要研究方向为软件度量与软件维护.E-mail:hmlu@nju.edu.cn.**周毓明**,男,1974 年生,教授,博士生导师,主要研究领域为实证软件工程.**徐宝文**(通信作者),男,1961 年生,教授,博士生导师,主要研究领域为软件工程.E-mail:bwxu@nju.edu.cn.

1 引 言

类是面向对象系统的基本模块,有可能随着系统的演化而发生变更.在先前的研究中,研究人员试图利用面向对象(OO)度量来预测系统中容易发生变更的类[1-4].预测易变的类具有重要的实际应用价值,它不仅使得开发人员可提早采取预防性措施来降低软件维护的成本和提高软件的质量,而且也可以使得项目经理能够更有效地分配软件开发资源.对于一个给定的 OO 度量,人们通常用单变量回归方程分析它与类的易变性之间的关系.如果该度量对应的回归系数是显著的,那么就认为它与易变性之间存在关联,否则认为它们不相关[2,4].

我们最近的研究结果指出上述单变量分析方法忽略了类规模的混和效应,有可能会得出误导性的结论,其原因是 OO 度量通常与类的规模之间存在高度的相关性[5].在 Eclipse 2.0 系统上,我们利用统计方法对当前 55 个主要的 OO 度量进行了类规模的混和效应检测.我们的结果表明,对 80% 以上的 OO 度量而言,类规模的混和效应都是存在的.特别地,我们发现,在不控制类的规模时,许多 OO 度量与类的易变性呈现高度的相关性.然而,在控制类的规模后,对 30% 以上的 OO 度量而言,它们与易变性之间的关联关系或者消失、或者呈现出相反的关联关系.据此,我们建议后续的研究应该将类的规模作为混和变量来考虑,以免得到不正确的实验结论.

尽管上述研究结论具有很重要的意义,但我们只分析了一个软件系统,因而不清楚该结论是否可以推广到其他系统上.为解决此问题,本文利用元分析技术和 102 个 Java 软件系统重新检查类的规模对 OO 度量和易变性之间关联关系的潜在混和效应问题.我们对 55 个 OO 度量进行了分析,其中包括 18 个内聚性度量、20 个耦合性度量和 17 个继承相关的度量.对每一个 OO 度量,我们首先在单个软件系统上分别计算在控制规模和不控制规模的两种情况下它与易变性的关联强度.然后,我们利用元分析中的随机效应模型计算在所有系统上且分别在这两种情况下它与易变性的平均关联强度[6].最后,我们在此基础上利用统计方法检测类规模的潜在混和效应.我们的实验结果证实类规模的混和效应是广泛存在的,因此在验证 OO 度量的易变性预测能力时确实需要将其作为一个混和变量来考虑.

本文第 2 节介绍元分析技术中的随机效应模型;第 3 节描述本文所采用的研究方法,包括数据来源、独立变量、依赖变量和数据分析方法;第 4 节给出详细的实验结果;第 5 节对本文的工作进行总结并对将来的工作进行展望.

2 元分析技术

元分析是一种用来将多个“研究”的结果进行合并的统计方法[6].其中,“随机效应模型”是最常用的一种元分析模型,它同时考虑“研究内方差”和“研究间方差”.根据文献[6],单个“研究”中的结果称为“可观察效应值”,它是“真实效应值”的一个估算量.对每一个 OO 度量,本文利用 102 个 Java 系统分析类规模对它与易变性之间关联强度的混和效应.此处,每一个 Java 系统是随机效应模型中的一个“研究”,关联强度是“效应值”.在随机效应模型中,不同“研究”中的“真实效应值”被假定为是不同的,并且服从一个正态分布(下文称它的平均值为“总平均值”).因此,对任一“研究”而言,其“可观察效应值”与“总平均值”之间的偏差由两部分构成:“真实效应值”与“总平均值”间的偏差和“可观察效应值”与“真实效应值”间的偏差.换句话说,每个“研究”的“可观察效应值”的方差可划分成“研究间方差”和“研究内方差”两部分:前者刻画各“研究”间“真实效应值”的差异,后者刻画每个“研究”内部的样本误差.元分析的目标是利用统计方法综合各“研究”的“可观察效应值”,得到一个“平均效应值”,用来估算“总平均值”.最近,元分析技术在软件工程中得到了许多应用[7-10].

设 k 是元分析中“研究”的数目,μ 是这些“研究”的“真实效应值”的“总平均值”.对第 i 个“研究”($1\leq i\leq k$),假设 Y_i、ζ_i 和 ε_i 各自表示它的“可观察效应值”、“真实效应值”与“总平均值”μ 的偏差以及“可观察效应值”与“真实效应值”的偏差.那么,可得

$$Y_i=\mu+\zeta_i+\varepsilon_i \tag{1}$$

其中,$E(\varepsilon_i)=0$、$Var(\varepsilon_i)=\sigma_i^2$、$E(\zeta_i)=0$ 且 $Var(\zeta_i)=\tau^2$.此处,$E(\cdot)$ 代表一个变量的期望值,$Var(\cdot)$ 代表一个变量的方差,σ_i^2 是第 i 个“研究”的“研究内方差”,τ^2 是“研究间方差”.由此,可知 Y_i 的方差为

$$Var(Y_i)=Var(\varepsilon_i)+Var(\zeta_i)=\sigma_i^2+\tau^2 \tag{2}$$

注意,上述公式中 $Var(Y_i)$、σ_i^2 和 τ^2 是种群值,在实践中要用样本值来代替.具体而言,当用样本值 s_i^2 和 T^2 各自替换式(2)中的 σ_i^2 和 τ^2 时,可得到 $Var(Y_i)$ 的样本估算值 V_i:

$$V_i=s_i^2+T^2 \tag{3}$$

其中

$$T^2=\max\left\{0,\frac{Q-df}{C}\right\} \tag{4}$$

在式(4)中，df、Q 和 C 各自表示自由度、总方差和比例因子：

$$df=k-1 \tag{5}$$

$$Q=\sum_{i=1}^{k}W_iY_i^2-\frac{\left(\sum_{i=1}^{k}W_iY_i\right)^2}{\sum_{i=1}^{k}W_i} \tag{6}$$

$$C=\sum_{i=1}^{k}W_i-\frac{\sum_{i=1}^{k}W_i^2}{\sum_{i=1}^{k}W_i} \tag{7}$$

其中，$W_i=1/s_i^2$. 随机效应模型利用对每个“研究”的“可观察效应值”进行方差倒数加权的方式计算 k 个“研究”的“平均效应值”M 及其方差 V_M：

$$M=\frac{\sum_{i=1}^{k}W_i^*Y_i}{\sum_{i=1}^{k}W_i^*} \tag{8}$$

$$V_M=\frac{1}{\sum_{i=1}^{k}W_i^*} \tag{9}$$

在上述公式中，$W_i^*=1/V_i$ 为第 i 个“研究”的权值 $(1\leq i\leq k)$.

如前所述，随机效应模型假定“可观察效应值”中的方差由真实的方差(“研究间方差”)和样本误差(“研究内方差”)组成. 在元分析中，这种真实的方差称作异质性. 通常用 I^2 来刻画在“可观察效应值”的方差中有多大比例来源于真实的方差，也就是“研究”间异质性的程度：

$$I^2=\frac{Q-df}{Q}\times 100\% \tag{10}$$

当 I^2 小于等于 25%时，表示具有低的异质性；当 I^2 介于 25%与 50%之间时，表示具有中等的异质性；当 I^2 大于等于 75%时，表示具有高的异质性[6].

3　研究方法

本节首先给出数据来源，然后介绍独立变量和依赖变量，最后详细描述所采用的数据分析方法.

3.1　数据来源

我们从 http://sourceforge.net/上下载了 102 个开源的 Java 软件系统作为实验对象. 与文献[7]中的方法一致，我们在选择这些系统时对它们的规模和类型事先没有做任何要求，只要求其实现语言是 Java. 对每个 Java 系统，我们下载了一个稳定的主版本中的两个子版本. 根据文献[2]，一个主版本的两个子版本所涉及的变更主要由修正性变更组成，它是由模块的结构特性而不是外部因素驱动的. 因此，对每一个 Java 类，这两个子版本间的代码行变更量实际上反应了由结构特性驱动的变更程度. 为方便起见，下文中将版本号小的子版本称为老版本，将版本号大的子版本称为新版本.

http://sourceforge.net/将 Java 开源软件系统分成 20 个类别，每个类别覆盖若干主题. 前 9 个类别分别为软件开发、因特网、科学/工程、游戏/娱乐、通讯、办公/业务、系统、多媒体和数据库，它们占该网站上全部 Java 系统的 80%以上. 特别地，对每个 Java 系统，http://sourceforge.net/用一个列表给出它所属的类别. 换句话说，同一个 Java 系统可同时属于不同的类别. 为简单起见，对每一个系统，本文将其列表中列出的第一个类别作为它的类别. 表 1 给出了这 102 个 Java 系统的描述性信息，其中给出了每个类别中系统的数目、开发人员数目的分布、老版本中类的数目的分布、新版本中类的数目的分布以及同时出现在新老版本中类的数目的分布. 此处，N/A 表示“不能应用”.

表 1　102 个 Java 系统的描述性信息

类别	系统数目	开发人员数目				老版本中类的数目				新版本中类的数目				同时出现在新老版本中类的数目			
		最大值	最小值	平均值	标准差	最大值	最小值	平均值	标准差	最大值	最小值	平均值	标准差	最大值	最小值	平均值	标准差
科学/工程	9	100	8	24.6	30.1	665	79	310.0	200.2	674	104	329.2	197.0	664	74	271.1	184.9
格式和协议	1	8	8	8.0	N/A	284	284	284.0	N/A	287	287	287.0	N/A	284	284	284.0	N/A
数据库	11	35	7	15.5	10.8	2858	66	732.0	882.5	2907	77	773.8	888.5	2845	65	614.8	827.5
办公/业务	10	67	9	17.8	17.4	1119	178	649.3	336.5	1544	238	781.2	483.9	1106	91	566.1	390.9
系统	10	38	8	16.7	9.4	1118	78	489.1	376.0	1286	111	520.2	404.2	851	69	339.9	246.0
游戏/娱乐	3	13	8	10.3	2.5	243	52	163.7	99.5	395	78	222.7	160.3	195	44	139.7	83.2
软件开发	25	139	7	20.3	26.4	1423	70	425.2	354.6	1527	85	471.3	370.5	1140	38	351.0	295.6
通讯	9	37	6	18.3	10.9	2187	105	476.8	650.4	2368	104	537.2	695.4	2150	104	440.3	644.9
多媒体	6	48	9	20.5	14.5	870	118	459.3	281.5	885	162	489.3	292.8	868	110	442.2	288.7
文本编辑器	1	47	47	47.0	N/A	475	475	475.0	N/A	475	475	475.0	N/A	437	437	437.0	N/A
因特网	17	123	5	33.1	34.2	2849	65	621.4	823.2	3605	66	689.3	957.0	2381	43	585.4	749.4
总计	102	139	5	21.4	23.1	2858	52	507.0	544.9	3605	66	560.7	601.2	2845	38	438.7	503.3

在每个 Java 系统上,我们生成一个对应的数据集.因此,总共得到 102 个数据集.在每一个数据集中,每个数据点表示一个同时出现在老版本和新版本中的 Java 类,它由如下成分组成:(1)在老版本中该 Java 类上收集的一组 OO 度量值;(2)该 Java 类从老版本向新版本演化时发生的 SLOC 变更量.为收集上述数据,我们首先利用程序理解工具"Understand for Java"扫描老版本和新版本的 Java 系统,得到两个对应的 Understand 数据库 .这两个数据库各自提供了老版本和新版本 Java 系统中的实体(如方法、属性和类)和引用(如调用关系和继承关系)信息.然后,我们利用 Understand for Java 提供的 Perl 接口开发了一个 Perl 程序,访问这两个 Understand 数据库,为那些同时出现在老版本和新版本中的 Java 类收集各种度量信息.对每一个这样的 Java 类,Perl 程序访问老版本系统的 Understand 数据库计算各种 OO 度量值.为得到该 Java 类从老版本向新版本演化时发生的 SLOC 变更量,Perl 程序第 1 步访问 Understand 数据库,获得它在老版本和新版本系统中对应的源代码,第 2 步利用一个 UNIX diff 算法变体比较两个版本的源代码,以统计增加的、删除的和修改的源代码行.第 3 步,Perl 程序利用如下规则计算 SLOC 变量[4]:(1)每一被增加的或者被删除的代码行计为 1 个 SLOC 变更;(2)每一被修改的代码计为 2 个 SLOC 变更.具体而言,对一个 Java 类,如果在从老版本系统演化到新版本系统的过程中增加了 x 行代码、删除了 y 行代码并且修改了 z 行代码,那么它所对应的总 SLOC 变更量为 $x+y+2\times z$.此处,每行被修改的代码之所以被记为 2 个 SLOC 变更,是因为可以将它看成是先删除一行代码然后再增加一行代码.

3.2 依赖变量

本文的依赖变量是类级的易变性,它是一个非常重要的外部质量属性,指一个类在跨版本演化时所发生的变更程度.与前人的研究一样,我们使用一个类从老版本向新版本演化时发生的 SLOC 变更量作为易变性的代理变量.当然,也可以采用其他代理变量,例如一个类从老版本向新版本演化时发生的修改次数.然而,用 SLOC 变更量作为易变性的代理变量不仅能反映每个类的被修改频率,而且也能反应它的被修改程度[2].从这个角度而言,用 SLOC 变更量作为代理变量更合适.

3.3 独立变量

本文的独立变量由 2 个规模度量和 55 个 OO 度量组成.我们利用 NMIMP 和 SLOC 作为规模度量来调查类规模对 55 个 OO 度量的易变性预测能力的潜在混和效应.其中,NMIMP 是一个类中实现的方法数目,SLOC 是一个类中所包含的非注释行、非空白行的源代码行数.这 55 个 OO 度量由 3 个度量维组成,包括 18 个内聚性度量、20 个耦合性度量和 17 个继承相关度量(请参见附录 A).我们之所以选择这 55 个度量,主要原因在于它们是软件工程文献中被广泛分析的度量[11-14].选择这些主流的 OO 度量可以使得我们的实验分析结果更具有代表性和实际应用价值.

3.4 数据分析方法

在 3.4.1 节中,我们介绍在单个软件系统上建模混和效应模型.在 3.4.2 节中,我们详细描述如何利用元分析方法综合多个系统上的混和效应得到平均的混和效应,并给出其统计上的检测方法.

3.4.1 混和效应模型

假定 X 是独立变量(一个 OO 度量)、Y 是依赖变量(易变性的代理变量,即 SLOC 变更量)、Z 是第三方变量(类规模度量 NMIMP 或 SLOC).混和效应是指 X 和 Y 之间的关联关系被 Z 扭曲了,它包括两种情况[5,15]:X 和 Y 间存在关联的原因可部分或者全部由 Z 来解释,或者 X 和 Y 之间不存在关联的原因是没有考虑 Z 的影响.第三方变量 Z 通常称为混和变量,它可导致人们高估或者低估 X 和 Y 之间真正的关联关系.在基于线性回归的混和效应模型中,X、Y 和 Z 之间的关系可以用如下等式描述:

$$Y=\beta_0^*+\tau X+e^* \tag{11}$$

$$Y=\beta_0+\tau' X+\gamma Z+e \tag{12}$$

$$Z=\beta_0^{**}+\beta X+e^{**} \tag{13}$$

其中,β_0^*、β_0 和 β_0^{**} 为回归截距,e^*、e 和 e^{**} 为余差,τ 代表不控制 Z 时 X 和 Y 之间的关系,τ' 代表控制 Z 时 X 和 Y 之间的关系,γ 代表控制 X 时 Z 和 Y 之间的关系,β 代表 X 和 Z 之间的关系.注意,上述参数都是种群值,在实践中要替换为样本值.由上述公式,我们可得到

$$\tau=\tau'+\beta\gamma \tag{14}$$

由此可见,第三方变量 Z 导致 X 和 Y 之间关联关系的扭曲程度为 $\beta\gamma$.只要 β 和 γ 都不为 0,那么 τ 和 τ' 就不相等.在先前的文献中,τ 被称为"总效应",τ' 被称为"直接效应",$\beta\gamma$ 被称为"间接效应".如果 Z 导致高估了 X 和 Y 间的关联强度,那么它是"正的混和变量".相反,如果 Z 导致低估了 X 和 Y 之间的关联强度,那么它是"负的混和变量".在极端情况下,一个负的混和变量不仅会导致关联强度的改变,而且会导致关联方向的改变.

为便于描述混和效应的元分析过程，下面假定 X_s、Y_s 和 Z_s 各自是 X、Y 和 Z 对应的标准化变量（减去平均值再除以标准差）. 由式(14)可得

$$\tau_s=\tau'_s+\beta_s\gamma_s \tag{15}$$

其中，τ_s、τ'_s、β_s 和 γ_s 分别为 τ、τ'、β 和 γ 所对应的标准化回归系数. 对一个给定的数据集，假设 N 是其数据点数目，r_{XY}、r_{XZ} 和 r_{YZ} 分别为 X 和 Y、X 和 Z、Y 和 Z 间的样本皮尔森相关系数，那么总效应 τ_s 的样本估算值为

$$\hat{\tau}_s=r_{XY} \tag{16}$$

它的方差为

$$var(\hat{\tau}_s)=\frac{1-r_{XY}^2}{N-2} \tag{17}$$

直接效应 τ'_s 的样本估算值为

$$\hat{\tau}'_s=\frac{r_{XY}-r_{XZ}r_{ZY}}{1-r_{XZ}^2} \tag{18}$$

它的方差为

$$var(\hat{\tau}'_s)=\frac{1-R^2}{(1-r_{XZ}^2)(N-3)} \tag{19}$$

其中

$$R=\sqrt{\frac{r_{XY}^2+r_{ZY}^2-2r_{XY}r_{ZY}r_{XZ}}{1-r_{XZ}^2}} \tag{20}$$

间接效应 $\beta_s\gamma_s$ 的样本估算值为

$$\hat{\beta}_s\hat{\gamma}_s=\frac{r_{XZ}(r_{ZY}-r_{XY}r_{XZ})}{1-r_{XZ}^2} \tag{21}$$

它的方差为

$$var(\hat{\beta}_s\hat{\gamma}_s)=\boldsymbol{a}\boldsymbol{\Phi}\boldsymbol{a}' \tag{22}$$

其中

$$\boldsymbol{a}=\left[\frac{r_{XZ}^2r_{ZY}+r_{ZY}-2r_{XZ}r_{XY}}{(1-r_{XZ}^2)^2},\ \frac{-r_{XZ}^2}{1-r_{XZ}^2},\ \frac{r_{XZ}}{1-r_{XZ}^2}\right] \tag{23}$$

$$\boldsymbol{\Phi}=\begin{bmatrix} var(r_{XZ}) & cov(r_{XZ},r_{XY}) & cov(r_{XZ},r_{ZY}) \\ cov(r_{XZ},r_{XY}) & var(r_{XY}) & cov(r_{ZY},r_{XY}) \\ cov(r_{XZ},r_{ZY}) & cov(r_{ZY},r_{XY}) & var(r_{ZY}) \end{bmatrix} \tag{24}$$

对任意 2 个变量 a 和 b 而言，它们的样本皮尔森相关系数 r_{ab} 的方差为

$$var(r_{ab})=\frac{(1-r_{ab}^2)^2}{N} \tag{25}$$

对任意 3 个变量 a、b 和 c 而言，两个相关系数 r_{ab} 和 r_{ac} 的协方差为

$$cov(r_{ab},r_{ac})=\frac{0.5\times(2r_{bc}-r_{ab}r_{ac})(1-r_{ab}^2-r_{ac}^2-r_{bc}^2)+r_{bc}^3}{N} \tag{26}$$

3.4.2　混和效应的元分析过程

我们使用随机效应模型计算 102 个系统上平均的“总效应”、“间接效应”和“直接效应”，在此基础上利用统计方法检测规模度量 Z 是否对 OO 度量 X 和易变性度量 Y 之间的关联存在混和效应. 如图 1 所示，这个元分析过程由 3 步组成. 第 1 步，对 102 个数据集（每个数据集对应一个 Java 系统）进行预处理，排除掉 X 的非零数据点数目少于 6 个的数据集（每个数据点对应一个 Java 类）. 先前的研究文献也采用了相同的处理方法，原因是一个方差很小的 OO 度量其预测能力必然很有限. 因此，在进行后续的分析前，我们先排除掉这样的数据集.

第 2 步，首先在单个数据集上利用式(16)和(17)计算标准化的总效应及其方差、利用式(18)和(19)计算标准化的直接效应及其方差、利用式(21)和(22)计算标准化的间接效应及其方差. 如 3.4.1 节所述，这些公式都建立在皮尔森相关系数的基础上. 皮尔森相关系数要求数据服从正态分布，但我们得到的大多数 OO 度量数据都与正态分布相差甚远. 因此，为解决上述问题，我们不直接计算皮尔森相关系数，而采用通过斯皮尔曼相关系数逼近的方法. 具体而言，对一对变量，我们首先计算出它们的斯皮尔曼相关系数 ρ，然后利用如下公式来得到它们的皮尔森相关系数 r[5]：

$$r=2\sin\left(\frac{\pi}{6}\times\rho\right) \tag{27}$$

在得到这些效应及方差后，我们利用随机效应模型计算平均的标准化总效应 PSTE、平均的标准化直接效应 PSDE 以及平均的标准化间接效应 PSIE.

第 3 步，检查规模度量 Z 是否对 OO 度量 X 和易变性度量 Y 之间的关联存在混和效应. 如果存在混和效应，进一步识别其方向并判断是否“污染”①了 X 的验证结果：

(1) 检查是否存在混和效应. 如果平均的标准化间接效应的种群值不为 0（即 PSIE 的 p 值小于 0.05），则 Z 存在混和效应，否则不存在混和效应.

(2) 识别混和效应的方向. 如果平均的标准化直接效应的种群值为 0（即 PSDE 的 p 值大于或等于 0.05），则 Z 的混和效应方向为“正”. 否则，当平均的标准化直接效应 PSDE 和平均的标准化间接效应 PSIE 具有相同的方向时存在“正”的混和效应，

① 当下述 3 种情况之一出现时，称 X 的验证结果被“污染”了：(1) 在控制 Z 前 X 和 Y 的关联是显著的，但控制 Z 后 X 和 Y 的关联不显著；(2) 在控制 Z 前 X 和 Y 的关联不显著，但控制 Z 后 X 和 Y 的关联显著；(3) 控制和不控制 Z 时，X 和 Y 的关联都显著但方向相反.

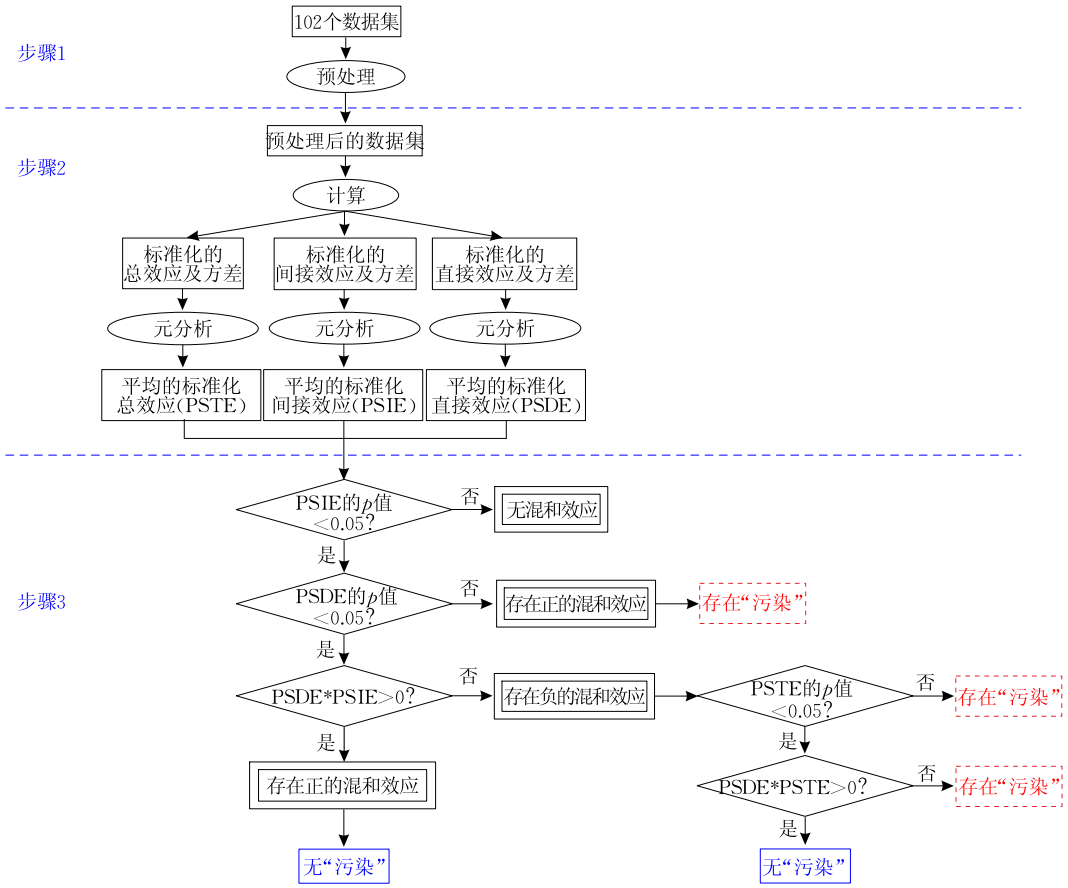

图 1 类规模对一个 OO 度量易变性预测能力的潜在混和效应的元分析过程

当它们具有相反的方向时具有“负”的混和效应.

(3)判断 X 的验证结果是否被“污染”. 当出现下述 3 种情况之一时,X 的验证结果被“污染”了:① 平均的标准化直接效应的种群值为 0;② 平均的标准化直接效应的种群值不为 0,但平均的标准化总效应的种群值为 0;③ 平均的标准化直接效应和平均的标准化总效应的种群值都不为 0,但它们的方向相反.

我们利用 Perl 语言编写了一个元分析工具,实现图 1 所描述的元分析过程. 在实际分析时,我们用一个文本文件列出 102 个数据集的名称,该工具访问该文本文件,依次读入相应的数据集,然后进行预处理和元分析工作,最后报告在每一个规模度量下的混和效应方向以及污染情况.

4 实验结果

4.1 节给出混和效应元分析的详细实验结果,4.2 节讨论并分析效度威胁.

4.1 混和效应分析

表 2 汇总了类规模的潜在混和效应的元分析结果. 对每一个 OO 度量:(1)第 2 列给出了实际参与元分析的“研究”数目;(2)第 3 列给出了平均的标准化总效应的样本值、它的 p 值以及异质性度量 I^2;(3)第 4 列和第 5 列分别给出了在规模度量 NMIMP 和 SLOC 下的平均的标准化直接效应的样本值、它的 p 值以及异质性度量 I^2;(4)第 6 列和第 7 列分别给出了在规模度量 NMIMP 和 SLOC 下的平均的标准化间接效应的样本值、它的 p 值以及异质性度量 I^2;(5)第 8 列和第 9 列分别给出了在规模度量 NMIMP 和 SLOC 下的混和效应方向. 此处,“+”表示 OO 度量验证结果没有被“污染”的正的混和效应,“⊕”表示 OO 度量验证结果被“污染”的正的混和效应,“−”表示 OO 度量验证结果没有被“污染”的负的混和效应,“⊖”表示 OO 度量验证结果被“污染”的负的混和效应,“0”表示不存在混和效应.

由表 2,我们可以观察到:

(1)内聚性度量. 当 SLOC 作为类规模度量时,Co 的平均的标准化间接效应种群值为 0. 在其他情况下,所有内聚性度量的平均的标准化间接效应的种群值都不为 0. 因此,对 Co 而言,当用 NMIMP 作

表 2　混和效应的元分析结果

面向对象度量	k	不控制规模	控制规模		混和效应测试结果		混和效应方向	
			NMIMP	SLOC	NMIMP	SLOC	NMIMP	SLOC
		$\hat{\tau}_s$(p-value)	$\hat{\tau}'_s$(p-value)	$\hat{\tau}'_s$(p-value)	$\hat{\beta}_s\hat{\gamma}_s$($p$-value)	$\hat{\beta}_s\hat{\gamma}_s$($p$-value)		
LCOM1	102	0.311(<0.01) I^2：92.533	0.084(<0.01) I^2：75.103	−0.006(0.679) I^2：81.992	0.226(<0.01) I^2：81.209	0.319(<0.01) I^2：92.153	+	⊕
LCOM2	102	0.253(<0.01) I^2：91.505	0.032(0.055) I^2：79.274	0.017(0.178) I^2：80.987	0.217(<0.01) I^2：88.107	0.235(<0.01) I^2：93.006	⊕	⊕
LCOM3	102	0.248(<0.01) I^2：91.797	0.038(0.031) I^2：84.416	0.025(0.049) I^2：82.739	0.208(<0.01) I^2：89.534	0.222(<0.01) I^2：92.032	+	+
LCOM4	102	0.098(<0.01) I^2：86.571	−0.040(<0.01) I^2：81.418	0.012(0.269) I^2：83.678	0.132(<0.01) I^2：94.296	0.081(<0.01) I^2：92.895	⊖	⊕
Co	102	0.022(0.099) I^2：85.110	0.057(<0.01) I^2：80.758	0.014(0.199) I^2：83.080	−0.023(<0.01) I^2：89.238	0.007(0.183) I^2：89.315	⊖	0
Co′	102	−0.047(<0.01) I^2：87.097	0.040(<0.01) I^2：77.221	−0.003(0.772) I^2：83.781	−0.077(<0.01) I^2：92.902	−0.039(<0.01) I^2：91.109	⊖	⊕
LCOM5	100	0.171(<0.01) I^2：85.578	0.077(<0.01) I^2：81.245	0.029(0.019) I^2：79.302	0.081(<0.01) I^2：88.330	0.136(<0.01) I^2：93.065	+	+
Coh	100	−0.254(<0.01) I^2：91.385	−0.106(<0.01) I^2：85.592	−0.036(0.009) I^2：78.967	−0.138(<0.01) I^2：86.254	−0.214(<0.01) I^2：93.705	+	+
TCC	102	−0.057(<0.01) I^2：87.195	0.041(<0.01) I^2：80.673	0.013(0.270) I^2：84.960	−0.088(<0.01) I^2：90.936	−0.059(<0.01) I^2：88.774	⊖	⊕
LCC	102	−0.028(0.070) I^2：88.072	0.037(<0.01) I^2：83.833	0.008(0.512) I^2：87.326	−0.054(<0.01) I^2：89.776	−0.028(<0.01) I^2：86.835	⊖	+
ICH	102	0.334(<0.01) I^2：92.352	0.204(<0.01) I^2：84.201	0.044(0.01) I^2：87.267	0.124(<0.01) I^2：85.724	0.291(<0.01) I^2：93.705	+	+
OCC	102	0.117(<0.01) I^2：89.803	0.004(0.774) I^2：87.212	−0.025(0.071) $I2$ 89.409	0.104(<0.01) I^2：92.093	0.135(<0.01) I^2：94.300	⊕	⊕
PCC	102	0.101(<0.01) I^2：87.937	−0.009(0.497) I^2：84.569	−0.023(0.074) I^2：87.108	0.102(<0.01) I^2：91.143	0.117(<0.01) I^2：92.480	⊕	⊕
DC_D	102	−0.054(<0.01) I^2：86.710	0.045(<0.01) I^2：78.814	0.012(0.303) I^2：83.755	−0.090(<0.01) I^2：92.081	−0.055(<0.01) I^2：90.174	⊖	⊕
DC_I	102	−0.022(0.154) I^2：88.003	0.042(<0.01) I^2：82.693	0.008(0.530) I^2：86.498	−0.053(<0.01) I^2：90.199	−0.023(<0.01) I^2：87.517	⊖	+
CAMC	102	−0.297(<0.01) I^2：92.669	−0.054(0.028) I^2：85.524	−0.020(0.207) I^2：87.031	−0.239(<0.01) I^2：88.795	−0.273(<0.01) I^2：96.089	+	⊕
NHD	101	0.234(<0.01) I^2：85.965	0.061(<0.01) I^2：49.181	0.030(<0.01) I^2：73.290	0.155(<0.01) I^2：92.933	0.195(<0.01) I^2：94.283	+	+
SNHD	101	−0.147(<0.01) I^2：83.004	−0.015(0.165) I^2：72.197	−0.012(0.206) I^2：73.562	−0.125(<0.01) I^2：91.257	−0.131(<0.01) I^2：94.121	⊕	⊕
CBO	102	0.365(<0.01) I^2：94.317	0.288(<0.01) I^2：92.225	0.179(<0.01) I^2：91.856	0.070(<0.01) I^2：86.123	0.179(<0.01) I^2：89.143	+	+
RFC	102	0.307(<0.01) I^2：93.085	0.203(<0.01) I^2：90.974	0.122(<0.01) I^2：90.930	0.098(<0.01) I^2：88.785	0.174(<0.01) I^2：90.771	+	+
MPC	102	0.350(<0.01) I^2：92.773	0.262(<0.01) I^2：89.968	0.145(<0.01) I^2：88.556	0.079(<0.01) I^2：88.017	0.198(<0.01) I^2：93.854	+	+
DAC	102	0.290(<0.01) I^2：89.387	0.159(<0.01) I^2：83.591	0.042(<0.01) I^2：84.128	0.125(<0.01) I^2：90.400	0.241(<0.01) I^2：94.055	+	+
ICP	102	0.349(<0.01) I^2：92.784	0.263(<0.01) I^2：90.226	0.147(<0.01) I^2：89.507	0.078(<0.01) I^2：87.782	0.195(<0.01) I^2：93.412	+	+
IH-ICP	97	0.063(<0.01) I^2：91.092	0.054(<0.01) I^2：88.532	0.053(<0.01) I^2：89.483	0.008(0.013) I^2：85.161	0.009(0.043) I^2：87.480	+	+
NIH-ICP	102	0.363(<0.01) I^2：93.139	0.279(<0.01) I^2：90.791	0.155(<0.01) I^2：89.482	0.077(<0.01) I^2：87.357	0.202(<0.01) I^2：93.730	+	+
ACAIC	30	0.012(0.314) I^2：68.351	−0.008(0.471) I^2：64.604	−0.014(0.150) I^2：64.554	0.014(<0.01) I^2：68.021	0.022(<0.01) I^2：79.528	+	+
ACMIC	43	0.010(0.390) I^2：69.627	−0.011(0.308) I^2：70.895	−0.009(0.372) I^2：68.421	0.013(<0.01) I^2：76.037	0.014(<0.01) I^2：78.288	+	+
DCAEC	3	<0.001(0.995) I^2：18.817	−0.002(0.895) I^2：0	0.004(0.792) I^2：0	0.002(0.794) I^2：78.120	−0.005(0.623) I^2：83.874	0	0
DCMEC	12	0.030(0.039) I^2：60.296	0.005(0.671) I^2：32.044	0.010(0.357) I^2：37.042	0.022(<0.01) I^2：77.227	0.018(<0.01) I^2：78.350	⊕	⊕

续 表

面向对象度量	k	不控制规模	控制规模		混和效应测试结果		混和效应方向	
			NMIMP	SLOC	NMIMP	SLOC	NMIMP	SLOC
		$\hat{\tau}_s$(p-value)	$\hat{\tau}'_s$(p-value)	$\hat{\tau}'_s$(p-value)	$\hat{\beta}_s\hat{\gamma}_s$($p$-value)	$\hat{\beta}_s\hat{\gamma}_s$($p$-value)		
OCAIC	102	0.290(<0.01) I^2: 89.776	0.160(<0.01) I^2: 84.664	0.042(<0.01) I^2: 84.605	0.124(<0.01) I^2: 89.808	0.242(<0.01) I^2: 93.753	+	+
OCAEC	99	0.130(<0.01) I^2: 83.067	0.040(<0.01) I^2: 81.513	0.036(<0.01) I^2: 83.066	0.080(<0.01) I^2: 91.592	0.085(<0.01) I^2: 91.282	+	+
OCMIC	102	0.325(<0.01) I^2: 91.173	0.172(<0.01) I^2: 80.689	0.026(0.084) I^2: 82.093	0.143(<0.01) I^2: 87.915	0.297(<0.01) I^2: 92.161	+	⊕
OCMEC	98	0.064(<0.01) I^2: 82.097	−0.020(0.082) I^2: 80.640	0.017(0.095) I^2: 81.742	0.073(<0.01) I^2: 89.559	0.039(<0.01) I^2: 87.627	⊕	⊕
AMMIC	97	0.065(<0.01) I^2: 90.484	0.054(<0.01) I^2: 87.863	0.052(<0.01) I^2: 88.623	0.010(<0.01) I^2: 85.197	0.012(<0.01) I^2: 87.653	+	+
DMMEC	87	0.072(<0.01) I^2: 77.271	0.009(0.326) I^2: 70.531	0.029(<0.01) I^2: 73.259	0.054(<0.01) I^2: 86.874	0.036(<0.01) I^2: 79.732	⊕	+
OMMIC	102	0.361(<0.01) I^2: 93.275	0.276(<0.01) I^2: 90.895	0.152(<0.01) I^2: 89.274	0.076(<0.01) I^2: 87.509	0.204(<0.01) I^2: 94.184	+	+
OMMEC	102	0.142(<0.01) I^2: 89.406	0.015(0.226) I^2: 83.149	0.026(0.025) I^2: 84.988	0.114(<0.01) I^2: 89.804	0.106(<0.01) I^2: 91.551	⊕	+
CBI	91	00.62(<0.01) I^2: 77.032	0.002(0.789) I^2: 67.204	0.025(<0.01) I^2: 69.826	0.052(<0.01) I^2: 86.179	0.032(<0.01) I^2: 79.598	⊕	+
DIT	102	−0.034(0.01) I^2: 86.121	0.004(0.776) I^2: 85.694	0.019(0.076) I^2: 82.168	−0.033(<0.01) I^2: 87.207	−0.046(<0.01) I^2: 91.823	⊕	⊕
AID	102	−0.034(0.01) I^2: 86.121	0.004(0.776) I^2: 85.694	0.019(0.076) I^2: 82.168	−0.033(<0.01) I^2: 87.207	−0.046(<0.01) I^2: 91.823	⊕	⊕
CLD	91	0.049(<0.01) I^2: 76.500	−0.002(0.853) I^2: 66.872	0.025(<0.01) I^2: 68.757	0.044(<0.01) I^2: 87.292	0.021(<0.01) I^2: 79.453	⊕	+
NOC	91	0.048(<0.01) I^2: 75.526	−0.002(0.849) I^2: 65.439	0.025(<0.01) I^2: 67.240	0.043(<0.01) I^2: 86.992	0.020(<0.01) I^2: 78.895	⊕	+
NOP	101	−0.037(<0.01) I^2: 85.057	−0.002(0.827) I^2: 81.336	0.011(0.282) I^2: 79.240	−0.029(<0.01) I^2: 86.559	−0.040(<0.01) I^2: 90.351	⊕	⊕
NOD	91	0.048(<0.01) I^2: 75.510	−0.002(0.823) I^2: 65.321	0.025(<0.01) I^2: 67.041	0.043(<0.01) I^2: 87.135	0.020(<0.01) I^2: 79.089	⊕	+
NOA	102	−0.034(0.01) I^2: 86.121	0.004(0.776) I^2: 85.694	0.019(0.076) I^2: 82.168	−0.033(<0.01) I^2: 87.207	−0.046(<0.01) I^2: 91.823	⊕	⊕
NMO	97	0.041(0.01) I^2: 90.729	0.013(0.340) I^2: 86.883	0.030(0.019) I^2: 89.097	0.027(<0.01) I^2: 88.029	0.015(<0.01) I^2: 88.676	⊕	+
NMI	98	0.015(0.290) I^2: 87.059	0.034(<0.01) I^2: 82.964	0.052(<0.01) I^2: 83.672	−0.014(<0.01) I^2: 87.994	−0.029(<0.01) I^2: 88.966	⊖	⊖
NMA	102	0.309(<0.01) I^2: 92.497	−0.028(0.474) I^2: 84.026	−0.043(0.018) I^2: 86.818	0.337(<0.01) I^2: 86.451	0.353(<0.01) I^2: 92.581	⊕	⊖
SIX	97	0.016(0.253) I^2: 87.137	0.002(0.878) I^2: 84.060	0.021(0.049) I^2: 83.632	0.015(<0.01) I^2: 86.219	0.001(0.802) I^2: 87.938	+	0
SPA	77	0.099(<0.01) I^2: 81.581	0.061(<0.01) I^2: 81.607	0.040(<0.01) I^2: 79.502	0.032(<0.01) I^2: 86.769	0.054(<0.01) I^2: 89.896	+	+
SPD	64	0.094(<0.01) I^2: 72.596	0.033(<0.01) I^2: 72.623	0.042(<0.01) I^2: 71.208	0.053(<0.01) I^2: 88.463	0.044(<0.01) I^2: 82.082	+	+
SP	87	0.130(<0.01) I^2: 83.266	0.067(<0.01) I^2: 82.462	0.056(<0.01) I^2: 80.728	0.053(<0.01) I^2: 90.229	0.065(<0.01) I^2: 90.341	+	+
DPA	97	0.040(0.013) I^2: 90.579	0.012(0.347) I^2: 86.901	0.030(0.018) I^2: 89.061	0.026(<0.01) I^2: 87.752	0.014(<0.01) I^2: 88.430	⊕	+
DPD	86	0.079(<0.01) I^2: 74.186	0.010(0.281) I^2: 70.809	0.031(<0.01) I^2: 70.351	0.061(<0.01) I^2: 89.896	0.042(<0.01) I^2: 84.311	⊕	+
DP	98	0.070(<0.01) I^2: 90.836	0.012(0.382) I^2: 87.373	0.041(<0.01) I^2: 88.961	0.053(<0.01) I^2: 92.348	0.031(<0.01) I^2: 90.109	⊕	+

为类规模度量时存在混和效应. 对其余的内聚性度量而言,无论用 NMIMP 还是 SLOC 作为类规模度量,总是存在混和效应. 在大多数情况下,类规模呈现出“正”的混和效应. 对一些内聚性度量而言,这种正的混和效应非常强以至于完全解释了它们与易变性之间的关联关系. 例如,在控制 NMIMP 或者 SLOC 后,LCOM2、OCC、PCC 和 SNHD 与易变性之间的关联完全消失. 此外,类规模的“负”的混和效应不仅

有可能导致关联强度的改变，而且也有可能会导致关联方向的改变. 例如，在控制 NMIMP 前，LCOM4 与易变性正相关；在控制 NMIMP 后，LCOM4 与易变性负相关. 内聚性度量的实验结果表明：① 当 NMIMP 作为类规模度量时，LCOM2、LCOM4、Co、Co′、TCC、LCC、OCC、PCC、DC_D、DC_I 和 SNHD 的验证结果被“污染”；② 当 SLOC 作为类规模度量时，LCOM1、LCOM2、LCOM4、Co′、TCC、OCC、PCC、DC_D、CAMC 和 SNHD 的验证结果被“污染”.

(2) 耦合性度量. DCAEC 的平均的标准化间接效应种群值为 0. 在其他情况下，所有耦合性度量的平均的标准化间接效应的种群值都不为 0. 因此，对 DCAEC 而言，类规模度量不存在混和效应. 对其余的耦合性度量而言，总是存在混和效应. 耦合性度量的实验结果表明：① 当 NMIMP 作为类规模度量时，DCMEC、OCMEC、DMMEC、OMMEC 和 CBI 的验证结果被“污染”；② 当 SLOC 作为类规模度量时，DCMEC、OCMIC 和 OCMEC 的验证结果被“污染”.

(3) 继承相关度量. 当 SLOC 作为类规模度量时，SIX 的平均的标准化间接效应种群值为 0. 在其他情况下，所有继承相关度量的平均的标准化间接效应的种群值都不为 0. 因此，对 SIX 而言，当用 NMIMP 作为类规模度量时存在混和效应. 对其余的继承相关度量而言，无论用 NMIMP 还是 SLOC 作为类规模度量，总是存在混和效应. 继承相关度量的实验结果表明：① 当 NMIMP 作为类规模度量时，DIT、AID、CLD、NOC、NOP、NOD、NOA、NMO、NMI、NMA、DPA、DPD 和 DP 的验证结果被“污染”；② 当 SLOC 作为类规模度量时，DIT、AID、NOP、NOA、NMI 和 NMA 的验证结果被“污染”.

表 3 汇总了类规模的潜在混和效应的元分析结果. 其中，表 3(a)给出了在各度量维上混和效应方向的分布，表 3(b)给出了在各度量维上 OO 度量验证结果“污染”程度的分布. 此处，“正”指“+”或者“⊕”，“负”指“−”或“⊖”，强“正”指“⊕”，强“负”为“⊖”. 从表 3，我们可以观察到：

(1) 在每个度量维上，类规模对超过 90%的 OO 度量都具有混和效应. 当不区分度量维时，类规模对 95%以上的 OO 度量具有混和效应.

(2) 在每个度量维上，类规模对超过 60%的 OO 度量具有“正”的混和效应. 当不区分度量维时，类规模对超过 80%的 OO 度量具有“正”的混和效应.

(3) 对内聚性度量和继承相关的度量而言，至少有 30%的 OO 度量的验证结果被“污染”. 当不区分度量维时，至少有 35%的 OO 度量的验证结果被“污染”.

(4) 对内聚性度量和继承相关的度量而言，在控制类的规模后，至少有 22%的 OO 度量与易变性的相关性消失. 当不区分度量维时，在控制类的规模后，至少有 30%的 OO 度量与易变性的相关性消失.

表 3　类规模潜在混和效应的元分析结果汇总

(a) 混和效应分布

类别	NMIMP/%		SLOC/%	
	“正”	“负”	“正”	“负”
内聚性度量	61	39	94	0
耦合性度量	95	0	95	0
继承相关度量	94	6	82	12
所有度量	84	15	91	4

(b) 污染程度分布

类别	NMIMP/%		SLOC/%	
	“正”	“负”	“正”	“负”
内聚性度量	强“正”	强“负”	强“正”	强“负”
耦合性度量	22	39	56	0
继承相关度量	25	0	15	0
所有度量	71	6	24	12

上述结果表明：(1) 对 OO 度量而言，类规模的混和效应是普遍存在的，在大多数情况下会导致人们高估它们与易变性之间的关联关系；(2) 许多 OO 度量的验证结果被类规模的混和效应“污染”了，其中最常见的“污染”是类规模的混和效应导致 OO 度量与易变性之间具有虚假的关联关系.

4.2　效度威胁

效度指实验结论的真实性程度和有效性程度，主要包括建构效度、内部效度和外部效度. 建构效度指依赖变量和独立变量在多大程度上准确地度量了它们所要代表的概念. 内部效度指自变量和因变量之间关系的确定性程度，涉及到实验结论的真实性程度. 外部效度指自变量和因变量之间关系的可推广性程度，涉及到实验结论的代表性程度.

4.2.1　建构效度的威胁

本文的依赖变量是一个类的两个子版本间的 SLOC 变更量. 在文献[2]中，Arisholm 等人手动检查了 Java 软件中同一个主版本下的两个子版本间的变更情况，发现它们是由结构性变更组成的. 据此，以两个子版本间的 SLOC 差异作为依赖变量来分析 OO 度量的易变性预测能力具有一定的合理性. 然而，在软件维护实践中，维护人员可能会由于版本发布计划、可用资源等外部条件的因素而将一些结构修正推迟进行，从而有可能会影响分析结果. 实际上，这一问题是软件度量实证研究难以避免的通用问题. 例如，在验证 OO 度量的缺陷预测能力

时，许多文献以软件发布后一个固定时间段长度（例如6个月）内报告的bug数目作为依赖变量[16]. 在实践中，时间段长度的选取必定会影响分析结果. 我们计划在后续研究中以跨度更大的子版本为实验对象，分析这一威胁对实验结果的影响.

本文的独立变量是2个规模度量和55个OO度量，先前的文献已对它们的建构效度进行了调查[11-12,17-18]. 特别地，我们利用商业工具Understand for Java提供的API收集度量数据，在一定程度上保证了度量数据收集的可靠性.

4.2.2 内部效度的威胁

本文实验结论的内部效度存在3个可能面临的威胁. 第1个威胁是依赖变量计算规则的未知影响. 在先前的分析中，每一被增加的或者被删除的源代码行被计为1个SLOC变更，每一被修改的源代码行被计为2个SLOC变更（下文称对应的依赖变量为“TS变更”）. 为检查计数规则对实验结论的影响，我们使用3个附加的计数规则生成3个依赖变量：

(1)“AS变更”. 只对被增加的源代码行计数，每一被增加的源代码行被记为1个SLOC变更；

(2)“DS变更”. 只对被删除的源代码行计数，每一被删除的源代码行被记为1个SLOC变更；

(3)“MS变更”. 只对被修改的源代码行计数，每一被修改的源代码行被记为1个SLOC变更.

图2给出了在4个不同依赖变量下102个数据集的Java类上的SLOC变更量的分布情况. 在每一个特定的数据集上，每一SLOC变更量区间（0、1～10、11～20、21～30、31～50、51～100、101～200和>200）的分布用一个百分比表示. 由于总共有102个数据集，对每一给定的SLOC变更量区间而言，总共有102个百分比，我们因此用一个箱线图描述其总体分布. 例如，对于SLOC变更区间“1～10”，102个数据集上“AS变更”分布的中位值为10.96%，第三四分位数为17.65%，第一四分位数为5.73%. 由图2的SLOC变更区间0上的分布可以看出，不管对哪一种依赖变量，超过50%以上的数据集中有65%以上的Java类在系统版本演化过程中没有发生变更.

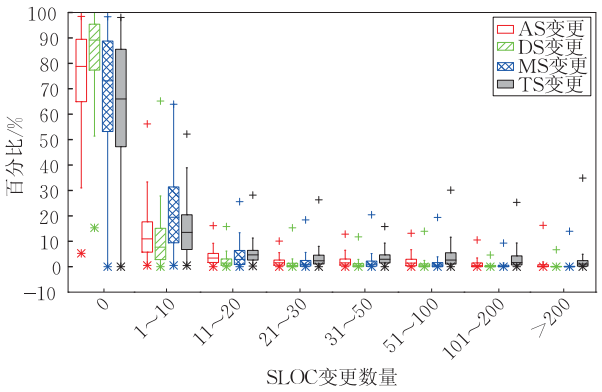

图2 不同依赖变量下的类的SLOC变更量分布

图3给出了在不同依赖变量下的实验分析结果. 从图3(a)可以看出，当NMIMP或者SLOC作为类规模度量时，不管在哪个依赖变量下，类规模的混和效应对90%以上的OO度量都存在，其中主要是“正”的混和效应. 从图3(b)可以看出，当NMIMP或者SLOC作为类规模度量时，不管在哪个依赖变

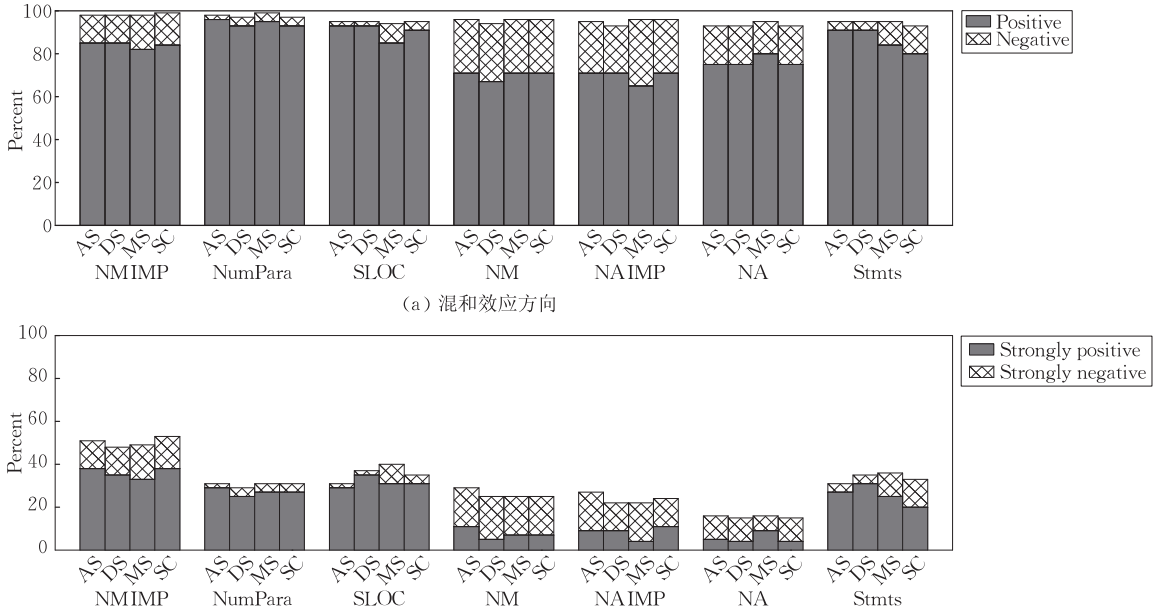

图3 不同依赖变量下的元分析实验结果

量下，30％以上的 OO 度量的验证结果都被“污染”了，其中大部分度量与易变性之间的关联都是虚假的. 总体上，上述结果表明依赖变量的计算规则对我们的实验结论没有大的影响.

第 2 个威胁是类规模度量的选择对实验结论的未知影响. 在前文中，我们只使用 NMIMP 和 SLOC 作为类规模的度量. 为调查这个威胁，我们使用如下类规模度量重新分析潜在混和效应：(1) NM. 一个类中的方法总数，包括继承的方法和非继承的方法；(2) NAIMP. 类中属性的数目(排除继承的属性)；(3) NA. 类中的属性总数，包括继承的和非继承的；(4) Stmts. 类的所有方法中语句总数，包括声明语句和可执行语句. 图 3 给出了不同的类规模度量下的实验结果，容易看出：(1) 不管对哪个依赖变量使用哪个规模度量，类规模的混和效应对 90％以上的 OO 度量都存在；(2) 对除 NA 外的其他规模度量而言，20％以上的 OO 度量的验证结果都被类规模的混和效应“污染”了. 由此可知，类规模度量的选择对我们的实验结论也没有实质性的影响.

第 3 个威胁是斯皮尔曼相关系数到皮尔森相关系数的转换对实验结论的未知影响. 在先前的分析中，我们首先将斯皮尔曼相关系数转换为皮尔森相关系数，然后在此基础上计算“总效应”、“直接效应”和“间接效应”并进行元分析. 由于在我们的数据集上，大多数 OO 度量值都不服从正态分布，因此这种处理是合适的. 尽管如此，有人可能会认为这种变换实际上是不必要的，因为皮尔森相关系数具有一定的鲁棒性，数据是否服从正态分布对其影响不大. 为消除这个威胁，我们直接使用皮尔森相关系数重新分析类规模的混和效应，发现实验结果非常类似. 因此，这种转换对我们的实验结论的影响也很小.

4.2.3　外部效度的威胁

本文实验结论的外部效度存在 3 个可能面临的威胁：第 1 个威胁是所得的实验结论可能只对特定类别的软件系统适用. 在实验中，我们在选择这 102 个系统时，事先没有对它们的规模和类型做任何要求，只要求其实现语言是 Java，这与前人的系统选择方法是一致的[7]. 如前文所述，这些软件系统涵盖了软件开发、因特网、科学/工程、游戏/娱乐、通讯、办公/业务、系统、多媒体和数据库等主要类别，在这些数据集上得出的结论不太可能只对特定类别的软件系统适用；第 2 个威胁是所得的实验结论有可能不能推广到其他系统上. 在前文的分析中，我们使用了元分析中的随机效应模型来进行分析. 根据文献[6]，随机效应模型不仅考虑了“研究”(即系统)内的样本误差，也考虑了“研究”间的差异，因此所得的结论可以推广到其他“研究”上；第 3 个威胁是所得的实验结论可能只对 Java 软件系统适用，而不能推广到其他语言实现的软件系统上. 我们只使用了 Java 软件系统的数据进行分析，其他语言实现的系统在结构特性上有可能有差别，这一威胁有待在将来的工作中进一步调查.

5　结束语

我们最近的研究指出类的规模对 OO 度量的易变性预测能力具有很强的混和效应，因此需要将其作为混和变量来考虑，这对正确地分析 OO 度量与易变性之间的关系具有重要的实际意义. 但我们先前的研究只分析了一个软件系统，其结论是否能够推广到其他系统上需要进一步调查. 为此，本文采用元分析方法和 102 个 Java 系统上的数据对类规模的混和效应进行了更细致的分析. 我们的实验结果证实和强化了先前的结论：类规模的混和效应是广泛存在的，而且在大多数情况下会导致高估 OO 度量的易变性预测能力. 因此在 OO 度量易变性预测能力研究中，人们确实需要考虑类规模这个混和变量，以便得出正确的结论. 在将来的工作中，我们计划对软件缺陷预测上下文中的规模混和效应进行元分析研究[19].

参　考　文　献

[1] Koru A, Tian J. Comparing high-change modules and modules with the highest measurement values in two large-scale open-source products. IEEE Transactions on Software Engineering, 2005, 31(8): 625-642

[2] Arisholm E, Briand L, Føyen A. Dynamic coupling measurement for object-oriented software. IEEE Transactions on Software Engineering, 2004, 30(8): 491-506

[3] Koru A, Liu H. Identifying and characterizing change-prone classes in two large-scale open-source products. Journal of Systems and Software, 2007, 80(1): 63-73

[4] Li W, Henry S. Object-oriented metrics that predict maintainability. Journal of Systems and Software, 1993, 23(2): 111-122

[5] Zhou Y, Xu B, Leung H. Examining the potentially confounding effect of class size on the associations between object-oriented metrics and change-proneness. IEEE Transactions on Software Engineering, 2009, 35(5): 607-623

[6] Borenstein M, Hedges L, Higgins J, Rothstein H. Introduction to Meta-Analysis. UK: John Wiley & Sons, Ltd., 2009

[7] Succi G, Pedrycz W, Djokic S, et al. An empirical exploration of the distributions of the Chidamber and Kemerer object-oriented metrics suite. Empirical Software Engineering, 2005, 10(1): 81-104

[8] Hannay J, Dybå T, Arisholm E, Sjøberg D. The effectiveness of pair-programming: A meta-analysis. Information and Software Technology, 2009, 51(7): 1110-1122

[9] Pickard L, Kitchenham B, Jones P. Combining empirical results in software engineering. Information and Software Technology, 1998, 40(14): 811-821

[10] Miller J. Applying meta-analytical procedures to software engineering experiments. Journal of Systems and Software, 2000, 54(1): 29-39

[11] Briand L, Daly J, Wüst J. A unified framework for cohesion measurement in object-oriented systems. Empirical Software Engineering, 1998, 3(1): 65-117

[12] Briand L, Daly J, Wüst J. A unified framework for coupling measurement in object-oriented systems. IEEE Transactions on Software Engineering, 1999, 25(1): 91-121

[13] Chidamber S, Kemerer C. A metrics suite for object-oriented design. IEEE Transactions on Software Engineering, 1994, 20(6): 476-493

[14] Counsell S, Swift S, Crampton J. The interpretation and utility of three cohesion metrics for object-oriented design. ACM Transactions on Software Engineering and Methodology, 2006, 15(2): 123-149

[15] Emam K, Benlarbi S, Goel N, Rai S. The confounding effect of class size on the validity of object-oriented metrics. IEEE Transactions on Software Engineering, 2001, 27(7): 630-650

[16] Zimmermann T, Premraj R, Zeller A. Predicting defects for eclipse//Proceedings of the 3rd International Workshop on Predictor Models in Software Engineering. Minneapolis, USA, 2007: 1-9

[17] Briand L, Wüst J, Daly J, Porter D. Exploring the relationships between design measures and software quality in object-oriented systems. Journal of Systems and Software, 2000, 51(3): 245-273

[18] Briand L, Morasca S, Basili V. Property-based software engineering measurement. IEEE Transactions on Software Engineering, 1996, 22(1): 68-86

[19] Zhou Y, Xu B, Leung H, Chen L. An in-depth study of the potentially confounding effect of class size in fault prediction. ACM Transactions on Software Engineering and Methodology, 2014, 23(1): article 10: 1-51

附录 A.

下面简介本文中所分析的 55 个 OO 度量，具体定义可参见文献[5].

(1)内聚性度量

LCOM1：类中不访问相同属性的方法对数目.

LCOM2：类中不访问相同属性的方法对数目与访问相同属性的方法对数目的差.若该差为负，则置 0.

LCOM3：无向图 G 的连通子图数.其中，G 的节点为类中的方法，边表示两个方法访问相同的属性.

LCOM4：与 LCOM3 的定义类似，除了无向图 G 上增添表示方法调用的边.

Co：$Co=2(|E|-|V|+1)/((|V|-1)(|V|-2))$.其中，$E$ 为 LCOM4 的无向图 G 的边集，V 为节点集.

Co′：Co 的一个变体.$Co'=2|E|/(|V|(|V|-1))$.

LCOM5：$LCOM5=\left(n-\frac{1}{k}\sum_{j=1}^{k}\mu(a_j)\right)\Big/(n-1)$.其中 n 和 k 分别为类中方法和属性的数目，$\mu(a_j)$为访问属性 a_j 的方法数目.

Coh：LCOM5 的一个变体，$Coh=\left(\sum_{j=1}^{k}\mu(a_j)\right)\Big/(n\cdot k)$.

TCC：紧密内聚度，指 public 方法中直接相连的方法对所占的百分比(如果两个方法访问相同的属性，则称它们“相连”).

LCC：松散内聚度，指 public 方法中直接或者间接相连的方法对所占的百分比.

ICH：基于信息流的内聚性度量.一个方法的 ICH 为它对类中其他方法的调用次数与相应参数数目的乘积.一个类的 ICH 为它的所有方法的 ICH 之和.

OCC：无向图 G 中节点可达的方法数与 $n-1$ 的比值的最大值.其中，G 中的节点为类中的方法，边表示两个直接或者间接访问相同属性，n 是节点的数目.如果 $n\leq 1$，则 OCC 的值为 0.

PCC：与 OCC 的定义相同，除了 G 是一个有向图，其中弧表示两个方法间存在“写属性—读属性”依赖关系.

DC_D：与 TCC 的定义相同，除了“相连”的含义.两个方法是“相连”的，如果存在调用关系、直接或者间接访问相同属性、或者直接或者间接调用相同的方法.

DC_I：与 LCC 的定义相同，其中“相连”的含义同“DC_D”.

CAMC：$CAMC=\frac{1}{nl}\sum_{i=1}^{n}\sum_{j=1}^{l}po_{ij}$.其中，$\boldsymbol{PO}$ 为 $n\times l$ 的参数发生矩阵，n 是类中的方法数目，l 是这些方法的不同参数类型数目.如果第 i 个方法有第 j 个参数类型，则 $po_{ij}=1$，否则 $po_{ij}=0$.

NHD：$NHD=\frac{2}{ln(n-1)}\sum_{j=1}^{n-1}\sum_{i=j+1}^{n}pa_{ij}$.其中，$\boldsymbol{PA}$ 为 $n\times n$ 的参数一致性矩阵，pa_{ij} 为第 i 个方法和第 j 个方法的参数发生向量中值相等的位数.

SNHD：尺度化(scaled)的 NHD.

(2)耦合性度量

CBO：类间的非继承耦合.

RFC：类的响应集，指该类实现的方法数目与其直接或间接调用的方法数的总和.

MPC：消息传递耦合，指类的方法中的方法调用总数.

DAC：数据抽象耦合，指类中以其他类为类型的属性数目.

ICP：基于信息流的耦合，与 ICH 定义类似，但只考虑类间的方法调用.

IH-ICP：与 ICP 相同，但只考虑基于继承的耦合.
NIH-ICP：与 ICP 相同，但只考虑非继承耦合.
ACIAIC：祖先类“类-属性”交互导入耦合.
ACMIC：祖先类“类-方法”交互导入耦合.
DCAEC：后裔类“类-属性”交互导出耦合.
DCMEC：后裔类“类-方法”交互导出耦合.
OCAIC：其他类“类-属性”交互导入耦合.
OCAEC：其他类“类-属性”交互导出耦合.
OCMIC：其他类“类-方法”交互导入耦合.
OCMEC：其他类“类-方法”交互导出耦合.
AMMIC：祖先类“方法-方法”交互导入耦合.
DMMEC：后裔类“方法-方法”交互导出耦合.
OMMIC：其他类“方法-方法”交互导入耦合.
OMMEC：其他类“方法-方法”交互导出耦合.
CBI：基于继承的耦合.
(3)继承相关度量
DIT：继承树的深度.
AID：平均的继承深度.
CLD：类到叶节点的最大深度.
NOC：儿子节点数，即子类数目.
NOP：父亲节点数，即父类数目.
NOD：后裔节点数，即后裔类数目.
NOA：祖先节点数，即祖先类数目.
NMO：覆写的方法数.
NMI：继承的方法数.
NMA：新增的方法数.
SIX：SIX＝NMO×DIT/(NMO＋NMA＋NMI).
SPA：祖先类中的静态多态.
SPD：后裔类中的静态多态.
SP：SP＝SPA＋SPD.
DPA：祖先类中的动态多态.
DPD：后裔类中的动态多态.
DP：DP＝DPA＋DPD.

LU Hong-Min, born in 1975, Ph. D. Her current research interests are software metrics and software maintenance.

ZHOU Yu-Ming, born in 1974, professor, Ph. D. supervisor. His research interest is empirical software engineering.

XU Bao-Wen, born in 1961, professor, Ph.D. supervisor. His research interest is software engineering.

Background

In software metrics community, a hot research topic is to validate the relationships between object-oriented (OO) metrics and change-proneness. In the last two decades, it is common to use univariate linear regressions to investigate the relationships between OO metrics and change-proneness. However, recent research shows that this methodology is problematic, as it does not taken into account the effect of class size, a strong confounding variable, on the validity of OO metrics. For many OO metrics, the confounding effect of class size completely accounts for their associations with change-proneness or results in a change of the direction of the association. It is hence recommended that class size should be considered as a confounding variable. Otherwise, misleading analysis results would be obtained.

However, previous research only uses one system to investigate the potentially confounding effect of class size. Therefore, it is not clear whether this conclusion can be generalized to other systems. In this paper, based on 102 Java systems, we employ random-effect meta-analysis model to investigate the relationships between OO metrics and change-proneness. The investigated OO metrics cover four metrics dimensions, including 18 cohesion metrics, 20 coupling metrics, and 17 inheritance metrics. Our experimental results indicate that the confounding effect of class size in general exists and hence confirm that we should consider it as a confounding variable when validating OO metrics on change-proneness. Since this study uses random-effect models to combine the experimental results from individual systems, the drawn conclusions can be generalized to beyond the included systems. In other words, the conclusions in this paper can be generalized to other systems.

The work in this paper is supported by the National Basic Research Program (973 Program) of China (2014CB340702), the National Natural Science Foundation of China (91318301, 61321491, 61300051, 61272082), and the National Natural Science Foundation of Jiangsu Province (BK20130014). These projects aim to develop novel software quality assurance techniques for improving the trustworthiness of software systems. This paper gives a correct method to validate OO metrics on change-proneness, which provides a foundation to use OO metrics to evaluate and improve software trustworthiness.