# Java类和包的易替换性度量与影响因素分析

刘辉辉,李必信,廖 力,王家慧
(东南大学计算机科学与工程学院,江苏南京 211189)

**摘 要:** 按照ISO25010标准中代码易替换性定性描述,人们很难从被替换软件产品的代码出发,定量地刻画其易替换性.为了自动化地度量代码的易替换性,本文充分考虑Java类/包的耦合关系和本身的复杂度,定义了一个类/包的易替换性度量公式.然后,在100个开源项目上进行实验,结果表明:(1)不同构造型的类的易替换性差异较大,其差异性与类承担的交互职责的多少有关;(2)包的易替换性与包中类个数没有显著的线性相关性;(3)与按层次划分的包相比,按功能特性划分的包具有更高的易替换性.从代码易替换性角度来看,在设计类和包时,本文的经验研究为开发者提供了有益的建议.

**关键词:** 类的易替换性;包的易替换性;耦合;类的构造型;按功能特性划分包;按层次划分包
**中图分类号:** TP311 **文献标识码:** A **文章编号:** 0372-2112 (2017)09-2149-07
**电子学报URL:** http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2017.09.014

# Replaceability Measurement and Impact Factor Analysis for Java Classes and Packages

LIU Hui-hui, LI Bi-xin, LIAO Li, WANG Jia-hui
(*School of Computer Science and Engineering, Southeast University, Nanjing, Jiangsu* 211189, *China*)

**Abstract:** Based on qualitative description of code replaceability in ISO25010 standard, it is difficult to quantitatively measure code replaceability. In order to make this process automatic, we define a replaceability measurement formulation by adequately considering the coupling relationship and intrinsic complexity in classes/packages. Then, an experiment is performed on 100 popular open source projects, and results show that (1) there are significant differences among different classes in terms of replaceability and these differences largely depend on the degree of communication between different classes and (2) there is no significant linear relationship between package replaceability and the total number of classes located in it and (3) the replaceability value of package designed by feature is more than the value of package designed by layer. From perspective of code replaceability, our empirical study also provides some suggestion for developers when they design a class or package.

**Key words:** class replaceability; package replaceability; coupling; class stereotype; package organized by feature; package organized by layer

## 1 引言

按照ISO25010对软件质量属性的分类,软件的易替换性(replaceability)是可移植性的二级子属性,它是指在使用环境和目的相同情况下,用某个软件产品替换另外一个软件产品的难易程度.在面向对象系统中,软件产品可以是web服务、子系统、构件、包、类或函数.通常,对演化系统实施在线升级时,需要对替换的产品进行易替换性评估.当前,人们对软件易替换性的研究主要集中在构件层次[1,4~9].Pradel和Gross[2]针对子类实例替换父类实例的不安全问题进行了分析.Chaki等人[4~6]基于自动机描述构件的行为和子类型替换原则,研究了演化系统中构件的易替换性问题,提出了一个涵盖包含和兼容性判定的构件易替换性分析框架,其不足在于替换后系统的行为不可减少.Belguidoum和Dagnat[7]基于构件内部的依赖、构件之间的依赖以及上下文环境的描述,提出了一种构件替换的形式化定义和兼容规则,以验证构件替换的正确性和安全性.

收稿日期:2016-08-11;修回日期:2016-12-05;责任编辑:覃怀银
基金项目:国家自然科学基金(No.61572126,No.61402103);华为技术有限公司项目(No.YBN2016020009)

从以上的研究现状来看,当前的方法过多依赖被替换构件和备选构件的个体特性,这种方法很难对系统中构件的全体实施批量式的易替换性评估. 其次,尚未发现学者关注 Java 中类和包的易替换性度量和影响因素研究. 针对以上问题,本文将围绕下面三个问题研究 Java 软件系统中类和包的易替换性:

**RQ1**:对于不同构造型的类,它们的易替换性差异是否很大? 一个类的构造型(stereotype)是指这个类的一种标签或类型(具体的解释见 2.3 节).

**RQ2**:包中类的个数是否影响其易替换性?

**RQ3**:包的不同组织方式是否影响其易替换性? 我们将研究这两种组织方式,即,按功能特性划分(by feature)和按逻辑层次划分(by layer)对包的易替换性的影响.

本文的主要贡献如下:

(1)结合 Java 类和包的复杂性和耦合性,提出了一种类和包的易替换性度量公式;

(2)在 Java 开源软件中,不同构造型类的易替换性差异较大,其差异性与类承担的交互职责的多少有关;

(3)包的易替换性与包中类个数没有显著的线性相关性;

(4)按功能特性划分的包具有更高的易替换性.

## 2 易替换性度量与评估过程

Java 类和包的易替换性度量与评估过程如图 1 所示,主要包括:(1)代码解析;(2)易替换性度量;(3)类的构造型生成;(4)易替换性评估.

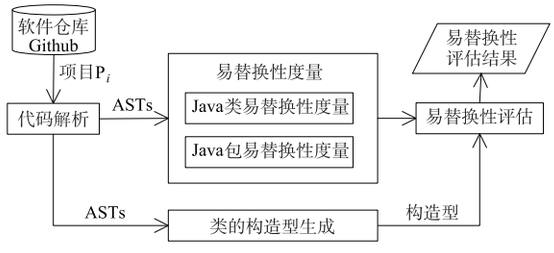



图1 Java类和包的易替换性度量与评估过程

### 2.1 代码解析

在代码解析模块中,我们采用 Eclipse 插件 JDT 对源代码进行解析. JDT 的输入是项目中所有 Java 源文件,输出是每个 Java 源文件对应的抽象语法树 AST(Abstract Syntex Tree,AST),一棵 AST 在 JDT 中也被称为一个编译单元(Compilation Unit). 为了方便用户遍历 AST 上的每个结点的信息,JDT 采用访问者模式提供了高达 84 种 visit 函数,借助这些函数,我们将提取类间各种耦合信息.

### 2.2 易替换性度量

在介绍易替换性度量方法之前,首先介绍一些相关概念以及计算公式.

**定义 1 类的传出耦合** 类的传出耦合是指该类依赖的其它类个数;

**定义 2 类的传入耦合** 类的传入耦合是指依赖该类的其它类个数.

对于定义 1 和定义 2 中的耦合关系,结合 Briand[10] 等人的工作,本文将 Java 中的依赖关系分为 7 种,如表 1 所示,这些类型的具体描述可参见表 1 的第二列.

**表 1** Java 程序中类间主要依赖(耦合)类型

| 分类 | 描述(类 $A$ 依赖类 $B$) |
|---|---|
| 继承耦合(Inheritance,IH) | 类 $A$ 继承类 $B$ |
| 方法调用耦合(Method Invocation,MI) | 类 $A$ 中的方法 $m1$ 调用了类 $B$ 中的方法 $m2$ |
| 方法参数耦合 Method Parameter,MP) | 类 $A$ 中方法 $m$ 的参数类型为类 $B$ |
| 方法返回类型耦合(Method Return Type,MRT) | 类 $A$ 中方法 $m$ 的返回类型为类 $B$ |
| 公有属性引用耦合(Public Attribute Reference,PAR) | 类 $A$ 中的公有属性 $a$ 被类 $B$ 使用 |
| 属性类型耦合(Attribute Type,AT) | 类 $A$ 中的属性 $a$ 的类型为类 $B$ |
| 局部变量类型耦合(Local Variable Type,LVT) | 类 $A$ 中方法 $m$ 的局部变量 $v$ 的类型为类 $B$ |

**定义 3 包的传出耦合** 包 $P$ 的传出耦合是指包 $P$ 依赖的其它包的数量.

**定义 4 包的传入耦合** 包 $P$ 的传入耦合是指依赖包 $P$ 的其它包的数量.

在定义 3 和定义 4 中,包的传出和传入耦合是通过包中类的依赖传达的. 换句话说,假设包 $P_1$ 中类的集合记为 $\{a_1,a_2,\cdots,a_m\}$,包 $P_2$ 中类的集合记为 $\{b_1,b_2,\cdots,b_n\}$,则包 $P_1$ 依赖包 $P_2$ 的含义是指,至少存在类 $a\in C(P_1)$ 和类 $b\in C(P_2)$,且通过表 1 中的一种或多种耦合类型,使得类 $a$ 依赖类 $b$.

定义 5 类/包的易替换性 类/包的易替换性是指替换类/包的难易程度. 类/包是否容易被替换主要取决于两点:(1)该类/包本身的复杂度;(2)类/包与软件中其它代码实体的耦合程度.

本文给出的类/包的易替换性度量公式如下:

$$R(x)=\left(1-\frac{n}{m}\right)\frac{\mathrm{Ce}(x)}{\mathrm{Ce}(x)+\mathrm{Ca}(x)}$$

其中,$x$ 表示类或包,$R(x)$ 表示 $x$ 的易替换性,$\mathrm{Ce}(x)$ 表示 $x$ 的传出耦合,$\mathrm{Ca}(x)$ 表示 $x$ 的传入耦合. 当 $x$ 表示类时,$n$ 表示类中高复杂方法的个数,$m$ 表示类中方法的总数;当 $x$ 表示包时,$n$ 表示包中高复杂类的个数,$m$ 表示包中类的总数. 本文中高复杂方法是指圈复杂度大于 10 的方法,高复杂类是指类的加权方法 WMC(Weighted Methods per Class)度量值大于 50 的类. 这里

阈值 10 参照了静态代码检查工具 PMD(https://pmd.github.io/pmd)中的默认值和学术界的研究成果[18],而阈值 50 是通过在 github 代码托管库上随机下载了 2000 个流行度较高的 Java 开源项目算出来的,具体的计算公式为 WMC = lq + 1.5(uq - lq),其中 lq 和 uq 分别为些项目中所有类的圈复杂度的下四分位数和上四分位数[19].

从定义 5 可以看出:Java 中类(包)的易替换性与其复杂性成反比;类(包)的传出耦合越大,它越依赖于其它类(包),当该类(包)发生变更时,对其它类/包的影响也就越小,因此,类(包)的传出耦合越大,其易替换性越大;一个类(包)的传入耦合越大,该类(包)被依赖的程度也就越高,说明此类(包)承担的职责太多,因此,它发生更改或被替换的代价也就越大. 从上述公式中也不难看出,类(包)的易替换性度量值 $R(x)$ 的取值介于 0 到 1 之间,$R(x)$ 值越接近 1,说明该类(包)越容易被替换;反之,$R(x)$ 的值越接近 0,说明类(包)越难被替换.

结合 2.1 节的代码解析过程,下面给出算法 1 以详细地说明从源码到易替换性度量的整个过程.

**算法 1　Java 类与包的易替换性度量**

```
输入:项目 P 的源代码
输出:项目 P 中所有类与包的易替换性度量值
01. sourceFiles = extractSourceFiles(P);
02. astSet = generateASTs(sourceFiles,jarFiles);
03. classNodeSet = extractClassNodes(astSet);
04. FOR each classNode in classNodeSet DO
05.    pkgName = extractPackageName(classNode);
06.    clsName = extractClassName(classNode);
07.    superClasses = extractSuperClasses(classNode);
08.    update(classCouplings,superClasses,IH);
09.    FOR each subnode x in classNode DO
10.       IF isMethodInvocation(x) THEN
11.          update(classCouplings,locatedClass(x),MI);
12.          parTypes = extractMethodParameterTypes(x);
13.          FOR each parType in parTypes DO
14.             IF isUserDefinedType(parType) THEN
15.                update(classCouplings,parType,MP);
16.             END IF
17.          END DO
18.          retType = extractMethodReturnType(x);
19.          IF isUserDefinedType(retType) THEN
20.             update(classCouplings,retType,MRT);
21.          END IF
22.       END IF
23.       attrRef = extractPublicAttributeReference(x);
24.       IF isUserDefinedType(attrRef) THEN
25.          update(classCouplings,attrRef,PAR);
26.       END IF
27.       attrType = extractAttributeType(x);
28.       IF isUserDefinedType(attrType) THEN
29.          update(classCouplings,attrType,AT);
30.       END IF
31.       varType = extractLocalVariableType(x);
32.       IF isUserDefinedType(varType) THEN
33.          update(classCouplings,varType,LVT);
34.       END IF
35.    END FOR
36. END FOR
37. FOR each coupleNode in classCouplings DO
38.    update(packageCouplings,coupleNode);
39. END FOR
40. classRepList = computeClassRep(classCouplings);
41. pkgRepList = computePackageRep(packageCouplings);
42. RETURN classRepList∪pkgRepList;
```

算法 1 中,首先将项目 $P$ 中每个源文件转化成相应的 AST,进而遍历每棵 AST,以提取 $P$ 中所有类结点(Line1 ~ Line3);然后,遍历 classNodeSet 集合中的每个类结点(及其所有子结点),并按照表 1 中的 7 种耦合类型,分别提取每个类的传入耦合类和传出耦合类,其中:

(1)位于 Line8 ~ Line33 处的 update()方法用于更新集合 classCouplings 中类的传入耦合类和传出耦合类. classCouplings 集合中元素的结构为四元组〈classCaSet,pkgName,className,classCeSet〉,其中 classCaSet 是 className 类的传入耦合类集合,它记录了每个传入耦合类的包名,类名,耦合类型. 传出耦合类集合 classCeSet 的元素结构与 classCaSet 类似.

(2)位于 Line38 处的 update()方法用于聚合 classCouplings 中类的耦合信息,以提取包的传入耦合和传出耦合. packageCouplings 集合中元素的结构为三元组〈pkgCaSet, pkgName, pkgCeSet〉, 其中 pkgCaSet 是包 pkgName 的传入耦合包集合,其记录了每个传入耦合包的信息,具体包括:包名和包中每个类的相关信息(如,类名,耦合类型等). 传出耦合包集合 pkgCeSet 的结构与 pkgCaSet 类似.

最后,利用 classCouplings 和 packageCouplings 中的信息,按照定义 5 中的公式,计算每个类和包的易替换性度量值(Line40 ~ Line41). 值得注意的是,本算法只考虑了用户自定义的类,而不考虑 JDK 及第三方 jar 包中类. 此外,为方便阅读,算法 1 中并没有描述每个方法和类的圈复杂度计算过程,在实验中,我们参照了 PMD 度量工具中的计算方法.

在算法 1 中,假设 $|classNodeSet| = N$,即项目 $P$ 中共有 $N$ 个类;每个类结点中的子结点个数不超过 $M$,以及 $P$ 中包的总数为 $K$,则计算每个类的传入传出耦合的复杂度为 $O(NM)$;将类的耦合聚合成包的耦合复杂度

为 $O(K)$,以及计算类的易替换性和包的易替换性的时间复杂度分别为 $O(N)$ 和 $O(K)$,因此算法 1 的总共复杂度为 $O(NM+K+N)$.

### 2.3 类的构造型生成

类的构造型(stereotypes)是类的一种标签或类型,是类的角色或职责的抽象描述. Dragan 等人[11,12]对类和方法的构造型分类体系做了详尽的描述,其中,将类的构造型分为 11 种,为了便于下文分析,简要介绍如下:

(1) Entity 是一种封装数据和行为的类,也是数据模型和业务逻辑的管理者,例如,在观察者模式中,观察目标类就是典型的 Entity 类型.

(2) Data provider 是一种封装属性的类,它的方法主要由访问器(accessor)构成.

(3) Commander 是一种封装行为的类,它的方法主要由修改器(mutator)构成.

(4) Boundardy 是一种交互类,其大部分方法的构造型属于协作(collaborational)类型,少部分属于控制器(controller)类型和工厂(factory)类型.

(5) Factory 是一种创建对象类,其大部分方法的构造型由工厂(factory)方法构成.

(6) Controller 是一种处理外部数据的类,其方法主要包括控制器(cotroller)方法和工厂方法.

(7) Pure Controller 是 Controller 构造型的特殊形式,该类的方法只包括控制器方法和工厂方法.

(8) Large Class[14]是一种包含过多职责的类,它违背了类的"单一职责"设计原则.

(9) Lazy Class[14]是一种平凡类,它承担的职责"过少",这种类往往是新创建的,其预期的功能尚未完全实现.

(10) Degenerate Class 是一种属性和行为退化的类,该类中空方法的数量所占的比重超过 50%.

(11) Data Class[14]是一种行为退化的类,只拥有一些属性以及用于读写这些属性的 setter 和 getter 方法. 这种类仅仅是数据容器,在绝大数多情况下,被其它类过份细琐地操控.

以上这些构造型可以采用 Moreno 等人[13]开发的工具 JStereoCode 对源代码中的类进行自动化地标注.

### 2.4 易替换性评估

在类的易替换性评估方面,首先计算项目中所有类的易替换性度量值的分布. 具体来说,我们将类的易替换性度量值范围[0,1]均分为 10 个数据点,然后统计在每个数据点的易替换性所占百分比,这些数据客观地反映了系统中类的易替换性分布状况. 此外,在评估 2.3 节中的 11 种类的易替换性时,通过计算每种构造型类的易替换性均值的方式,判断它们的易替换性分布是否均匀.

在包的易替换性评估方面,我们将利用统计学中假设检验的方法判断包的易替换性与包中类的个数是否相关,由于我们并不知道包的易替换性和包中类的数量是否服从正态分布,为此我们选择了 Spearman 和 Kendall[15,16]两种相关性检验方法. 值得注意的是,在统计包中类的个数时,仅涵盖包中顶层类和成员类,不考虑匿名类和局部类. 其次,在评估按功能特性和逻辑层次划分包对其易替换性影响程度时,我们采用人工校验的方式判断一个包的划分方式. 通常,按功能特性划分包的名子可以明确反映某个领域概念,这些概念可以通过阅读项目相关资料获得,按层次划分包的名字后缀往往出现 view, action, model, dao, ui 等关键字.

## 3 实验研究

### 3.1 实验对象

本文选取了 100 个 Java 开源软件,它们的源代码可以在 github 上获得. 这些项目在 github 上的流行度较高、规模中等,且广泛地被学术界引用. 在进行实验之前,首先清除代码中无效源文件,比如,去除没有包名的源文件;其次,重复源文件只保留一份. 为了降低评估代价,我们开发了一款 Java 构件易替换性评估工具 JCREvaluator (Java Component Replaceability Evaluator),这里的构件是指 Java 程序中类或包.

### 3.2 度量公式有效性验证

软件易替换性属于软件质量的外部属性,在学术界和工业界还没有统一的易替换性度量公式,为了验证本文提出的度量公式是否可以有效地度量包/类的易替换程度,我们采用问卷调查的形式进行验证. 我们邀请了 25 位软件工程专业的在校硕士研究生和 25 位企业界的程序员参与评估工作. 他们大多具有 3 年以上的 Java 软件编程经验. 为了减少评估偏倚,所选的类/包的实例信息是随机的,每个实例包括:源代码,传入/传出耦合的类型和易替换性度量值. 问卷中设置的问题为:"基于 Java 中类/包的耦合关系和本身的复杂程度,你认为该度量公式是否可以用于度量类/包的易替换性? 如果认为不可以,请说出理由;如果可以,请给出其准确程度:1(非常准确),2(准确),3(一般),4(准确性较低)". 调查的结果如表 2 所示,数据表明:大部分受访者认为该度量公式可以用于评估构件的易替换性(35/50 = 70%),持有异议者认为该公式过于简单,还应该考虑构件之间的耦合强度. 此外,还有人认为易替换性与开发文档是否完善、以及项目的语言也有关系. 这些反馈结果对度量公式的改进具有一定的参考价值.

表 2　问卷调查结果

| | 非常准确 | 准确 | 一般 | 准确性较低 |
|---|---|---|---|---|
| 在校研究生 | 2 | 18 | 4 | 1 |
| 企业开发人员 | 3 | 17 | 3 | 2 |

### 3.3　实验结果分析

针对 RQ1-RQ3 的研究问题,本节对 100 个开源项目上的实验结果进行分析.

**RQ1**:对于不同构造型的类,它们的易替换性差异是否很大?

不同构造型的类履行不同的职责,代表着设计者不同的意图. 图 2 是 100 个开源项目中类的易替换性度量值分布,不难看出,60% 的类易替换性介于 0.5 和 0.9 之间,仅有 20% 的类易替换性介于 0 和 0.5 之间. 图 3 是 100 个开源项目中所有构造型类的易替换性分布,其中每种构造型的易替换性度量值(图中黑色长方体)代表该构造型所有实例的易替换性平均值.

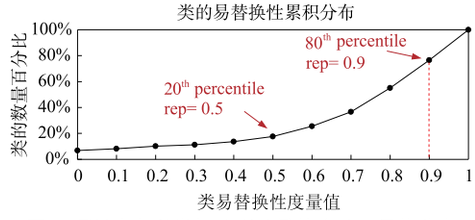



图2　项目中各种类构造型的易替换性度量值分布

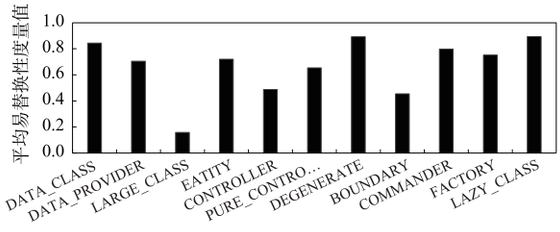



图3　项目中各种类构造型的平均易替换性度量值

为了说明问题,本文选取了 ArgoUML0.12、log4j-1.2.16、junit4.12、checkstyle6.3、guava14.0 五个项目进行人工分析. 从图 3 可以看出,不同构造型的类,其易替换性差异较大. Large Class,Controller 以及 Boundary 类型的易替换性较低(小于 0.5),而 Data Class,Data Provider, Entity, Pure Controller, Degenerate Class, Commander,Factory 以及 Lazy Class 替换性较高(约介于 0.6 和 0.9 之间). 前三个构造型的类均属于交互类,它们在系统中往往承担较多的职责,需要与别的类交互才能完成,因此,这些类的传入/传入耦合通常较高. 实验中,Large Class 的实例数量非常小,仅在 ArgoUML0.12 中发现一个 Configuration 类,其位于 org.argouml.application.api 包中. 如图 4 所示,Configuration 类总共含有 30 方法(其中包括 1 个构造方法,10 个 getter 方法,4 个 setter 方法,6 个重载方法,9 个其它方法).

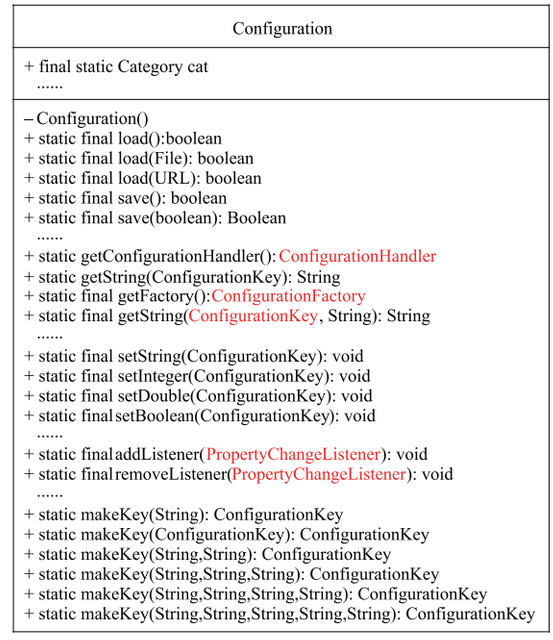



图4　ArgoUML0.12 中 Configuration 类的类图

经过分析,Configuration 类承担了太多的职责,具体包括资源的加载/保存,配置属性的设置/访问,配置属性变更的监听,以及构件配置主键的创建等. 此外,该类有 4 个传出耦合类(图 4 中红色标识的类),16 个传入耦合类,因此,Configuration 类的易替换性度量值仅有 0.2(即,4/(4 + 16) = 0.2,Configuration 类中没有出现高复杂方法).

在其余的 8 种类构造型中,构造型 Lazy Class、Degenerate Class 和 Data Class 均是退化的类,这三种构造型的类履行的职责很简单,因此它们的易替换性均值都超过了 0.8;而构造型 Entity、Data provider 和 Commander 主要由“访问器”和“修改器”构成,也可能包含一些其它类型的方法,因此其易替换性要弱于退化类,介于 0.7 和 0.8 之间;构造型 Factory 主要由工厂方法构成,由于该类主要负责类对象的创建,与其它类的交互相对较少,因此其易替换性也弱于退化类,但接近于 0.8;构造型 Pure Controller 是 Controller 和 Factory 的特例,其易替换性的度量值介于 Controller 和 Factory 的易替换性之间. 通过以上的分析,可以得出:对于不同构造型的类,其易替换性差异较大,差异的程度主要取决于该类是否承担交互职责.

**RQ2**:包中类的个数是否影响其易替换性指标值?

为了回答 RQ2,我们采用假设检验的方法判断包的易替换性与包中类的数目是否存在单调线性关系,其原假设和备择假设如下:

H0:包的易替换性与包中类的个数无关;

H1:包的易替换性与包中类的个数存在单调线性关系;

Spearman 和 Kendall 的检验结果如表 3 所示,可以看出两种检验方法的 p-value 相对较高,因此,我们无法拒绝原假设 H0,即,包的易替换性与包的个数不存在线性关系. 这一结论与我们的直觉相反. 从理论上来说,聚合在一个包中的类应该充分交互以实现系统的一个子功能,从而使得该包具有较高的内聚性. 表 4 反映的是 ArgoUML0. 12、junit4. 12 和 checkstyle6. 3 三个项目中位列前 5 的包(类的数量按从大到小顺序排序),其中 Ca 表示传入包的个数,Ce 表示传出包的个数,Rep 表示包的易替换性. 从表 4 可以得出,只有项目 checkstyle 中 com. puppycrawl. tools. checkstyle 包的易替换性较小(Rep = 0. 04),通过查看源代码,我们发现该包位于 src/main/java 目录中,这个目录一共有 21 个包,其中 com. puppycrawl. tools. checkstyle 包是其余 20 个包的顶层目录,它的功能涵盖了配置的加载、实用工具集、错误记录器、抽象语法树中节点的检测、包名的加载、对象创建等,这些功能的实现只用到了一个外部包 com. puppycrawl. tools. checkstyle. api,而其大多数功能却被其余的 20 个包使用. 此外,如此庞杂的包中还包含 4 个过于复杂类,因此,该包设计并不合理(140 个类,功能过于分散),违背了单一职责设计原则,最终导致了该包的易替换性度量值仅为 0. 04.

**表 3 包的易替换性与类数目的线性关系假设检验结果**

| | Spearman | Kendall |
|---|---|---|
| p-value | 0. 62 | 0. 87 |
| rho/tau | -0. 08 | -0. 06 |
| accept/reject | accept | accept |

**表 4 类数目排名前五的包及其易替换性**

| Package Name | #Cs | #CCs | Ca | Ce | Rep |
|---|---|---|---|---|---|
| org. junit. tests. experimental. rules | 158 | 1 | 0 | 9 | 0. 99 |
| org. argouml. uml. ui | 143 | 6 | 15 | 31 | 0. 65 |
| com. puppycrawl. tools. checkstyle | 140 | 4 | 25 | 1 | 0. 04 |
| com. puppycrawl. tools. checkstyle. coding | 114 | 2 | 0 | 1 | 0. 98 |
| org. argouml. uml. cognitive. critics | 99 | 0 | 3 | 15 | 0. 83 |

注:表 4 中#Cs 表示类的个数,#CCs 表示复杂类的个数

**RQ3**:包的不同组织方式是否影响其易替换性?

采用 2. 4 节的评估方法,我们分别调查了包的两种组织方式对包的易替换性影响. 实验结果如图 5 所示,从图中可以得出:按功能划分与按层次划分相比,前者可以明显地提高包的易替换性,即 75% 功能包的替换性介于 0. 5 和 0. 9 之间,而 75% 的层次包的替换性介于 0. 1 和 0. 6 之间. 两种不同的划分方式各有利弊,争论较大. 然而,从易替换性指标来看,按功能分类方式划分包,可以将包中类(及类的属性)的可见性设置为"包可见",从而降低了这些类暴露于包之外的风险. 同时,在访问同一个包中类的属性时,可以避免创建大量的"修改器"函数. 此外,从功能的易扩展性来看,按功能划分的包使得新功能的扩展更容易,即,局限于一个包内修改代码,满足开闭设计原则. 因此,从提高包的易替换性和可扩展性角度,在软件开发实践中,应该提倡按功能划分包,而非按逻辑层次划分包.

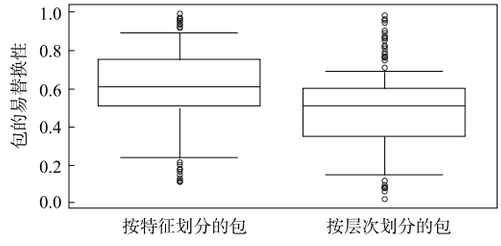



图5 包的不同划分形式对其易替换性的影响

## 4 总结

本文从耦合性和复杂度两个维度刻画了 Java 类/包的易替换性,有别于当前主流的研究方法,我们仅考虑被替换类/包的代码信息. 该方法好处是可以对项目中所有类和包进行批量式地度量和评估. 我们从代码托管库网站 github 上下载了 100 个开源项目进行实验研究,结果表明:(1)对于不同构造型的类,其易替换性的度量值差异较大,Large Class,Controller 以及 Boundary 类型的易替换性较低(小于 0. 5),而 Data Class,Data Provider,Entity,Pure Controller,Degenerate Class,Commander,Factory 以及 Lazy Class 替换性较高(介于 0. 6 和 0. 9 之间),差异的程度主要取决于该类是否承担交互职责;(2)在 Java 系统中,包的易替换性与包中类的数量不存在显著的线性关系;(3)在 Java 系统中,按功能划分的包与按层次划分的包相比,具有较高的易替换性,即 75% 功能包的替换性介于 0. 5 和 0. 9 之间,而 75% 的层次包的替换性介于 0. 1 和 0. 6 之间. 下一步的工作将完善易替换性公式和分析易替换性在多个版本中的演化规律.

### 参考文献

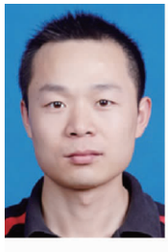
[1] Zuberek W M. Checking compatibility and substitutability of software components[A]. Models and Methodology of System Dependability[C]. Oficyna:Wydawnicza Politechniki Wroclawskiej,2010. 175 - 186.

[2] Pradel M,Gross T R. Automatic testing of sequential and concurrent substitutability[A]. International Conference on Software Engineering[C]. USA:IEEE,2013. 282 - 291.

[3] 张敬周,任洪敏,宗宇伟,钱乐秋,朱三元. 基于行为自动

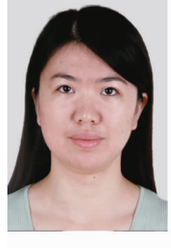
机的构件可替换性分析与验证[J]. 软件学报,2010,21(11):2768 - 2781.
Zhang Jin-zhou, Ren Hong-min, Zong Yu-wei, Qian Leqiu, Zhu San-yuan. Component substituability analysis and verification based on behavior automata[J]. Journal of Software, 2010, 21(11): 2768 - 2781. (in Chinese)
[4] Chaki S, Sharyina N, Sinha N. Verification of evolving software[A]. The 3rd Workshop on Specificaiton and Verificaiton of Component-Based Systems[C]. USA: ACM, 2004. 55 - 61.
[5] Chaki S, Clarke E, Sharygina N, Sinha N. Dynamic component substitutability analysis[A]. Proceedings of the 2005 International Conference on Formal Methods[C]. Berlin: Springer, 2005. 512 - 528.
[6] Sagar C, Edmund C, Natasha S, Nishant S. Verification of evolving software via component substitutability analysis[J]. Formal Methods in System Design, 2008, 32(3): 235 - 266.
[7] Belguidoum M, Dagna F. Formalization of component substitutability[J]. Electronic Notes on Theoretical Computer Science, 2008, 215(5): 75 - 92.
[8] Brada P, Valenta L. Practical verification of component substitutability using subtype relation[A]. The 32nd Euromicro Conference on Software Engineering and Advanced Applications[C]. USA: IEEE Computer Society, 2006. 38 - 45.
[9] Flores A, Polo M. Testing-based process for component substitutability[J]. Software Testing, Verification and Reliability, 2012, 22(8): 529 - 561.
[10] Briand L C, Daly J W, Wüst J. A unified framework for coupling measurement in object-oriented systems[J]. IEEE Transactions on Software Enggineering. 1999, 25(1): 91 - 121.
[11] Dragan N, Collard M L, Maletic J I. Reverse engineering method stereotypes[A]. 22nd IEEE International Conference on Software Maintenance[C]. USA: IEEE, 2006. 24 - 34.
[12] Dragan N, Collard M L, Maletic J I. Automatic identification of class stereotypes[A]. 26th IEEE International Conference on Software Maintenance[C]. USA: IEEE, 2010. 1 - 10.
[13] Moreno L, Marcus A. JStereoCode: Automatically identifying method and class stereotypes in java code[A]. 27th IEEE/ACM International Conference on Automated Software Engineering[C]. Germany: IEEE, 2012. 358 - 361.
[14] Fowler M, Beck K, Brant J, Opdyke W, Roberts D. Refactoring: Improving the Design of Existing Code[M]. USA: Addison Wesley Professional, 1999.
[15] Kendall M G, Gibbons J D. Rank Correlation Methods[M]. London: Edward Arnold, 1948.
[16] Spearman C E. The proof and measurement of association between two things[J]. American Journal of Psychology, 1904, 15(3): 72 - 101.
[17] Vasa R, Schneider J G. Evolution of cyclomatic complexity in object-oriented software[A]. Proceedings of 7th ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering[C]. Germany: IEEE, 2003. 1 - 5.
[18] Lanza M, Marinescu R. Object-Oriented Metrics in Practice[M]. Germany: Springer, 2006.

## 作者简介

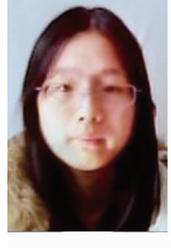


**刘辉辉**　男,1983 年出生于江苏省宿迁市. 现为东南大学计算机科学与工程学院博士研究生. 主要研究方向包括代码变更检测,代码坏味的演化与影响分析.
E-mail: lhhshuxue@126.com

**李必信**　男,1969 年出生于安徽省庐江县,博士/博士后,现任东南大学计算机科学与工程学院教授,博士生导师. 研究方向为软件建模、分析、测试与验证,演化系统的软件质量保证等.
E-mail: bx.li@seu.edu.cn



**廖　力**　女,1976 年生于陕西省宝鸡市. 现为东南大学计算机科学与工程学院讲师. 主要研究方向为软件工程,软件维护与演化.
E-mail: lliao@seu.edu.cn



**王家慧**　女,1994 年生于江苏省苏州市. 现为东南大学计算机科学与工程学院硕士研究生. 研究方向为架构坏味检测与分析.
E-mail: JhWang_SEU@163.com