代码坏味检测及重构的现状分析

姜德迅，马培军，苏小红，王甜甜

（哈尔滨工业大学 计算机科学与技术学院，哈尔滨 150001）

摘　要： 软件良好的设计质量能使维护和重用变得更加容易和方便，而如果代码中存在各种各样的坏味，那么必将导致软件整体设计质量降低。首先对坏味的定义、描述和分类进行分析，之后列举现有的坏味检测以及重构研究，对其进行分析和比较。现状分析之后指出了现存的不足之处，为新的研究指明了方向。

关键词： 研究现状；代码坏味；坏味检测；坏味重构

中图分类号： TP311　　**文献标识码：** A　　**文章编号：** 2095－2163(2014)03－0023－05

Related Work Analysis of Code Bad Smell Detection and Refactoring

JIANG Dexun, MA Peijun, SU Xiaohong, WANF Tiantian

(**School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China**)

Abstract: Good quality of software would make the maintenance and re－use easier and more convenient. When there are too many bad smells, the total quality would be reduced. In this paper, bad smell is defined, described and classified, and then the related work about bad smells detection and refactoring is listed, analyzed and contrasted. The limitation and shortage of the related work are presented to point out the way of further researches.

Key words: Related Work; Code Bad Smell; Bad Smell Detection; Bad Smell Refactoring

0　引　言

坏味(bad smell)[1]就是指代码中出现的一种“迹象”，导致代码难于理解和修改。坏味并不是代码中已经出现的错误或者缺陷，但是可能会导致错误或者缺陷的发生。所以说，坏味实际上是代码中潜在问题的警示信号。因此，当出现代码坏味时，就应当对其进行重构。重构是指对软件内部结构的一种调整，在不改变软件可观察行为的前提下，借助重构提高其可理解性，降低其修改成本。

坏味破坏了程序的设计质量，特别是面向对象程序中基于对象编程的设计质量。坏味会影响程序的可理解性。并且，坏味导致程序出现错误的可能性增大。因此，程序中出现坏味，将降低程序的整体质量，非常不利于程序的开发、修改、维护及扩展。坏味的检测分为人工检测和自动检测两种。人工检测代码的速度较慢，且主观性较高，检测准确性也较低。但是由于坏味的特征不够明确和统一，自动检测仍具有较大难度。

找到坏味并不是最终目的，本文研究的目的是为了对坏味进行修改并使之消除，从而达到提高代码质量的目的。这种消除坏味的方法即称为重构。Martin Fowler 在其著作[1]中是这样定义重构的：重构是对软件的内部结构所做的一种改变，这种改变在可观察行为不变的条件下使软件更容易理解，而且修改更廉价。这种重构指的是代码级别的修改行为。

1　坏味的定义和描述

Fowler 将代码坏味进行分类，论述了程序中可能存在的22种代码坏味，并逐类对其各自在代码中的表现进行描述，也给出了改善这种坏味现象的大体思路。在此基础上，Kerievsky[2]提出了5种新的代码坏味加以补充。Abebe[3]进一步给出了5种基于词法的代码坏味，并根据程序代码文本上出现的不良问题进行了新坏味的定义和描述。

将坏味进行分类，必将有助于对坏味的理解和使用。在提出坏味之后，Wake[4]等人对于坏味有一种简单的分类，即将坏味分为“类内坏味”和“类外坏味”两种。这种分类方式过于简单，并且分类中还包括很多不属于坏味的情况。Mantyla[5]对文献[1]中提出的22种代码坏味进行分类，将坏味分为：过度膨胀(the bloaters)、面向对象滥用(the object－orientation abusers)、妨碍修改(the change preventers)、可有可无情况(the dispensables)、封装问题(encapsulators)以及其他情况等。对坏味进行分类的意义在于，使得坏味更容易理解，坏味之间的关系更加明确，使对坏味的研究能够更加深入。

2　坏味检测

本文提到的坏味检测，是指坏味的自动检测。坏味的自动检测即是使用程序对代码中可能存在的坏味进行检测的过程。可以按照检测方法的不同将相关研究分为以下几类。

2.1　基于文本的坏味检测

基于文本的坏味检测主要是将源代码经过词法分析，并

收稿日期： 2014－03－08
基金项目： 国家自然科学基金(61073052)。
作者简介： 姜德迅(1983－)，男，黑龙江哈尔滨人，博士研究生，主要研究方向：代码坏味检测、坏味重构；
马培军(1963－)，男，山东潍坊人. 博士，教授，博士生导师，主要研究方向：软件工程、信息融合、图像处理与识别等；
苏小红(1963－)，女，黑龙江哈尔滨人. 博士，教授，博士生导师，主要研究方向：程序理解、克隆代码检测与重构、软件缺陷和代码坏味检测等。

万方数据

将得到的token串作为待检源,再通过文本比对和分析的方法进行检测的。该类研究能够检测的坏味通常也是与源代码文本相关的坏味,最常见的通过文本进行检测的坏味是重复代码[1]。其它可以通过文本进行检测的坏味还有switch惊悚现身、平行继承体系、过多的注释等。

Johnson[6]利用查找完全相同子串的方法处理冗余代码。Baker[7]提出了检测代码段中出现完全相同或近似相同现象的方法。这里,近似相同的含义是:若将一段代码中的一组变量名和常量用另外一段代码中的相应一组变量名和常量替换,则两段代码完全相同。

Adar[8]设计了一种代码克隆检测工具GUESS,并与Kim一起将其改进为SoftGUESS工具。该工具由一个克隆库和一系列小型应用程序组成,能够在系统依赖、设计信息和包结构等的文本层面上对代码克隆现象进行分析。同时,该工具可以对克隆代码现象进行多版本的可视化显示。

张鹏[9]研究两个代码是否具有相似性的课题,从相似代码的类型、从属关系等特点,建立其属性库,并根据相似评价标准来进行检测。使用最长公共子序列算法来实现程序代码之间的相似性比对,测量精度可达94%以上。

刘鑫[10]使用数据挖掘技术,实现了一个基于token串的重复代码检测模型。将重复代码检测问题转化成为一个序列模式,使用改进的CloSpan算法查找支持度至少为2的频繁子序列,对应于程序中的重复代码段,通过筛选得到重复代码对结果。该方法的时间复杂度较低,适于分析检测大规模程序,并且可以检测出经过修改的重复代码现象。

使用简单文本作比较来进行重构定位,检测速度快,误检率低,但是没有考虑代码的语义信息,所能检测的种类过少,漏检率高。

2.2 基于度量的坏味检测

基于度量的坏味检测,是指从源代码中提取或统计一些可以代表程序特征的数值,可能需要计算和分析,得到一系列度量值,并在预设阈值的作用下,进行比较和分析,最终得到结果,判断代码中是否存在相应的坏味。

Simon等人[11-12]主要使用基于内聚性的距离来量化代码中可能的缺陷,根据度量结果分析代码结构,从而识别到有缺陷的代码。其缺点在于识别代码缺陷的能力会因为度量的规则而存在差异,对于多态、异常和多线程的情况也没有加以考虑。

Tahvildari和Kontogiannis[13]完成了一个设计流再工程框架,并定义了复杂性度量、耦合性度量和内聚性度量三类,通过利用这些度量,来完成对程序设计性质方面的检测。缺点在于分析的结果需要程序员的自行判断,而且也无法进行自动定位。

Tsantalis和Chatzigeorgiou[14]使用实体距离作为度量,对程序运行中需要重构的代码坏味进行判断和定位,并自动进行了移动方法的重构操作。本文在Simon的距离理论的基础上做了一系列的改进,增加了类和实体之间、类和类之间的距离概念,使得能够对重构进行直接定位。但是这种方法计算量较大。

Reddy和Rao[15-16]通过面向依赖的复杂性度量值来检测坏味。首先利用类的调用关系来定义类之间的耦合度,通过计算一个类与其他类之间的平均耦合度来判断该类是否可能存在发散式变化坏味。这种方法思路明确,计算代价小,但是具有一定的局限性。

Sant' Anna[17]提取了代码中一些基于关系的度量数据,并用于坏味检测。通过将设计关系和代码之间映射的抽象化,来表达出程序可能存在的设计问题。这种坏味检测只能发现整个系统中坏味的存在,但无法做到明确定位。

通过提取度量值进行坏味检测的方法,其提取的特征同源程序有较高的符合度,并且具有较快的检测速度。但是对于度量的选取,是通过检测者的人为规定来完成的;得到度量值后有时需要与预设定的阈值进行比较判断才能得到检测结果。以上两点都会降低坏味检测的客观性,从而影响到检测结果的准确性。

2.3 基于聚类的坏味检测

聚类是指将物理或抽象对象的集合分成由类似的对象组成的多个类的过程。使用聚类分析的方法来检测坏味,主要是针对由于内聚性和耦合性等相关问题引发的一类坏味。

Lung等人[18]将聚类技术应用到程序的函数设计中,将内聚度低的函数分解成若干个新的函数,新函数的内聚度提高。同时也能够为函数重组提供更多的选择。文献中没有明确说明如何获取合适的实体属性,算法普适性较差。基于此,Lung等人在文献[19]中对实体属性获取方法做了改进和描述,使得聚类方法能够应用于更加普遍的源代码上。

Alkhalid等人[20]提出一种自适应K近邻算法,通过计算数据和控制属性的权重,来处理由于低内聚问题造成的相关代码坏味。该方法没有给出具体的重构方案。

上述研究采用聚类分析进行坏味检测,都需要人为设定阈值来对最终的聚类结果进行判断和分析,才能得到检测结果。阈值源于经验数据,具有较大的主观性,因此会导致坏味检测的准确率较低。

Srinivas[21]采用K近邻聚类方法来进行面向对象代码中包层级的重构。文章认为在包的层级,包内的类具有高内聚性,而包间耦合度低。通过判断各类中是否存在其他类的实例,做为内聚算法的数据。

Ratzinger和Sigmund等人[22]提出了一种根据程序开发历史记录来预测可能存在的代码坏味的方法。从程序的版本控制系统中获取程序自身的规模、编写者变动信息、修改次数、修改频率、修改习惯以及其他相关信息,作为进行聚类分析的数据来源。根据这些数据,采用决策树、逻辑树、重复增量修剪以及最近邻等聚类算法,获得待重构位置的可能预测结果。作者对聚类算法并没有进行实现,也无从对比各方法的优劣性。

Grosser等人[23]提出一种预测系统质量性质的方法。在保存有大量系统质量信息的数据库中,指定系统的质量因素(健壮性),是从其他与指定系统具有最大相关度的系统中获得。而获取最大相关度的其他系统,是通过K近邻聚类计算来完成的。

万方数据

采用聚类分析的方法进行相关代码坏味的检测,具有精确度高、检测结果准确的特点。聚类分析进行坏味检测,也要使用阈值(或其他预设值),但是阈值对检测结果的影响比基于度量的坏味检测方法要小,特别是使用一些特定聚类方法(如动态确定 K 值的 K 近邻聚类分析),阈值已经基本不会影响到检测结果。但是基于聚类的检测方法,需要通过计算来分析程序内部的关系,得到待处理的组群,之后再进行相关聚类分析,整体检测方法的时间复杂度较高,检测也相对耗时。

2.4　其他坏味检测方法

有一些研究使用不变式进行坏味检测。不变式是指在程序中固定不变出现的某种特征。基于 Banerjee 和 Kim[24] 对面向对象数据库模式的研究,Willan F. Opdyke[25] 定义了 7 个基于 C++ 的不变式,按照不同级别对程序进行概括。包括唯一超类、不同类名、不同的实体名、被继承的成员变量不能被重定义、成员变量重定义兼容、类型安全的赋值、语义等价的引用和操作等。

有一些研究使用程序的抽象语法树中间表示结构来完成代码坏味的检测以及重构工作。该类方法的特点是将源程序转化为抽象语法树,在这种树结构上寻找坏味。这些研究认为,在源代码上直接进行坏味检测,对程序元素之间的很多关系缺乏直观和明确的表示,需要构建其他中间表示如程序流图或依赖图等,这会导致程序分析和计算的费用较高,检测过程缓慢。而通过对源程序解析生成抽象语法树,之后对其进行分析,可避免这一问题。

李军超等人[26] 进行了基于抽象语法树的代码味道识别工具的分析与设计,将源代码转换成抽象语法树,对其进行分析和度量判断,可使方法过长、依恋情节等 10 种代码坏味得以识别。该研究对于坏味的判断是通过度量值和预定值完成,识别标准由人为设定,识别结果仍然受到太多的主观因素影响。

Kontogiannis 等[27] 提出一种由 5 种度量技术组成的方法来检测任意两个代码片段的相似性,其基本思想是将待检源代码转化成抽象语法树,得到 5 维向量,计算其欧氏距离,从而判断是否存在重复代码。在某些情况下,该方法会产生错误匹配。

刘建宾[28] 定义了一种叫做过程蓝图的图形化程序过程规格说明方法,可对程序源代码进行描述,具有丰富的语义,能够支持对程序的坏味进行检测。过程蓝图的本质实际上就是一种抽象语法树。在此基础上,李建忠等人[29] 利用过程蓝图进行重复代码的自动检测研究工作,通过分析抽象实现结构图的节点类型和带数据流节点的表达式,直接获得程序源码中所代表的过程控制结构和语句表达式的静态信息。

3　坏味重构的研究现状

在已有的研究中,大部分研究在给出重构方案或执行重构之后,并未对重构的效果进行评估。研究默认,在执行重构后,重构所要针对的问题(即重构动机)已经获得了解决,重构操作已经完成了预定任务,代码已经一切正常。这些研究认为,在重构定位过程中已经准确无遗漏地找到了所有的相关坏味(即需要使用重构操作实施改进的位置),并且在执行重构操作时,也已经考虑到重构操作的安全性,因此并不需要进行重构的评估。

有一些研究者对重构的行为进行了分析和评估,以此来对"重构确实保证了程序可观察行为不变"进行证明,此外对于重构的效果提供直观准确的评价,方便对同类研究工作进行对比和改进。这些研究对于重构的评估主要分为两个方面:一是重构操作是否去除或改善了代码中的不良现象,提高了程序的可维护性、易读性、可理解性等,优化了程序质量;二是重构操作是否会引起程序"可观察行为"的改变。

Kataoka 和 Imai 等人[30] 提出一种性能评估方法,来衡量程序重构的可维护性增强效果。该方法基于耦合度量来评估重构的影响。通过比较重构前后的耦合性,能够评估出可维护性增强的程度。但是方法只使用了程序的耦合性来代表可维护程度,度量获取片面,对一些不改变(或改变很少)耦合度的重构行为,无法评估其重构效果。

Tahvildari 和 Kontogiannis[31] 提出了一系列关于复杂性、耦合性和内聚性的度量,并用于程序转换框架的驱动条件中。同时,在程序转换后用这些度量值和判断准则来评估程序重构后的效果。只是文中方法得到的评估结果不够全面,并不具备最佳说服力。

Murphy - Hill 和 Black[32] 阐述了现有重构工具在解决"过长方法"方面的问题,定义了三个工具并将其组合来进行重构。结果显示在速度、准确性和用户满意度三个方面都有明显改进。但是该研究主要是针对用户使用重构工具的界面友好型和简易型进行设计和改进,对于重构的原理和方法却未做改变。

Tsantalis 和 Chatzigeorgiou[33] 根据程序实体间的距离关系来确定进行"移动方法"重构的时机,通过此重构操作来解决依恋情结代码坏味。在进行重构操作之后,本着面向对象程序在设计层面上高内聚、低耦合的思想,定义并计算类和整个系统的平均距离,并以此判断重构效果的好坏。但是这种评价准则只是一个粗略的方法,并未考虑到细节情况,而且也没有考虑到重构操作对系统质量其他方面的影响。

Bansiya 和 Davis[34] 建立了一个面向继承的评估模型 QMOOD(Quality Model for Object - Oriented Design),在灵活性、可重用性、可理解性等方面对软件的设计性能进行评估。Keeffe 和 Cinneide[35-37] 使用多种搜索算法来随机执行类继承方面的重构细粒度操作,而且使用 QMOOD 模型来评估各操作,从而达到提高程序质量的作用。但是该模型只能针对面向对象继承方面的表象进行评估,没有考虑其他方面,并且评估标准人为决定,其正确性尚需进一步的验证。

4　现有坏味检测及重构研究的不足

4.1　坏味定义和描述研究的不足

(1)在代码坏味的概念被提出之后,其对提高程序质量特别是重用效果方面的作用越来越大。随着对程序代码研究的逐级深入,越来越多的坏味被发现,原本笼统模糊的坏

万方数据

味也相应实现了细化。但是在程序质量方面,在许多程序中尚有多种坏味现象没有得到有效的定义和解决。例如程序中频繁存在类间功能过度相关,继承关系混乱的现象。

(2)在定义新坏味的研究[2-3,38]中,研究者只是提出了代码坏味的定义、描述以及危害,但是对如何检测到这种坏味,特别是使用程序进行自动检测的方法,以及对检测结果的效用和准确性方面,尚且缺少足够的分析结果。

4.2 坏味检测研究的不足

(1)许多研究将坏味检测工作规定为人工完成。所有与提高程序质量相关的研究工作,从去除、改进不良的代码现象角度来说,基本上包括两个方面:不良现象的查找,以及不良现象的处理。前者是指代码坏味的检测,后者是指针对代码坏味的重构工作。现有研究的大部分工作都偏重于后一方面,即默认坏味已经检测到,并针对坏味进行重构[39]。这些研究认为,坏味是一个相对笼统和模糊的概念,对于坏味的认知因人而异,因此很难通过程序来自动完成检测[40]。对于代码坏味,目前少有研究使用了量化方法,通过程序自动完成针对源代码的坏味检测工作,即是目前坏味自动检测研究的一个不足之处。但是,实际上通过对各类代码进行分析,除去几种少数特殊形式的坏味,目前提出的大多数坏味,都是能够通过数值化方法,表达坏味现象的存在。

(2)坏味特征提取困难。在已有的坏味自动检测研究[41-43]中,从程序中提取的坏味特征,不能完全体现代码的真实情况,导致坏味检测结果并不能完全真实地反映出程序中存在的代码不良现象造成的设计质量问题。

(3)坏味判断缺少客观量化的依据。在现有的坏味自动检测研究中,基本上是使用预设定的阈值来进行"是否是坏味"的判断。一些与代码规模相关的代码坏味,例如过长方法、过大的类、过长参数列,对其不论是采用自动检测或者是人工检测,预设值的设定更为重要。在一些使用聚类分析进行坏味检测的研究中[39,44],通过将聚类结果与预设值进行比较来判定是否存在坏味。使用预设值检测坏味,其检测结果主观性较高,不易反映出程序质量的真实情况。因此,需要弱化甚至去除坏味检测过程中所需要的预设值,提高坏味检测的客观性。

4.3 坏味重构研究的不足

在现有研究中对重构效果进行评估方面,仅考虑了重构是否消除了引发重构的动机(原有代码坏味)。实际上,重构行为除了可消去现有坏味之外,还可能会隐性地去除其他已存在坏味或者降低其他坏味存在的可能性。相应地,重构行为也可能会引入新的坏味。重构行为在这几方面的作用,共同影响了程序的质量。

5 结束语

代码坏味的检测及重构,已经受到越来越多的重视,因为其对程序的质量具有重要的影响。本文分别对坏味的定义、检测过程以及重构过程分别进行了研究现状的分析,指出了其不足之处,为进一步的研究指明了方向。

参考文献:

万方数据

[1]FOWLER M, BECK K, BRANT J, et al. Refactoring: Improving the design of existing code [M]. Addison Wesley, 1999.

[2]J. Kerievsky. Refactoring to patterns [M]. Addison - Wesley, 2004.

[3]ABEBE S L, HAIDUC S, TONELLA P, et al. Lexicon bad smells in software [C]// Working Conference on Reverse Engineering, 2009: 95 - 99.

[4]Wake W C. Refactoring Workbook [M]. Addison - Wesley, 2003.

[5]MANTYLA M. Bad smells in software: a taxonomy and an empirical study [D]. Ph. D. dissertation, Helsinki University of Technology, 2003.

[6]JOHNSON J H. Identifying redundancy in source code using fingerprints[C]//CASCON'93, 1993:171 - 183.

[7]BAKER B S. On finding duplication and near - duplication in large software systems[C]//Proceedings of The 2th Working Conference on Reverse Engineering(WCRE95). IEEE Computer Society Press, July 1995.

[8]ADAR E, KIM M. SoftGUESS: visualization and exploration of code clones in context [C]//International Conference on Software Engineering (ICSE'07), 2007: 762 - 766.

[9]张鹏. C代码相似代码识别方法的研究与实现[D]. 大连:大连理工大学,2007.

[10]刘鑫. 重复代码检测方法及其应用[D]. 哈尔滨:哈尔滨工业大学,2007.

[11]SIMON F, LOFFLER S, LEWERENTZ C. Distance based cohesion measuring [C]//European Software Measurement Conference 99, Technologist Institute Amsterdam, 1999.

[12]SIMON F, STEINBR F, LEWERENTZ C. Metrics based refactoring [C]//Proc European Conf Software Maintenance and Reengineering, 2001.

[13]TAHVILDARI L, KONTOGIANNIS K. A metric - based approach to enhance design quality through meta - pattern transformations [C]// European Conference Software Maintenance and Reengineering, 2003:183 - 192.

[14]TSANTALIS N, CHATZIGEORGIOU A. Identification of extract method refactoring opportunities [C]// Software Maintenance and Reengineering (CSMR '09), Kaiserslautern, Germany, March 2009:119 - 128.

[15]REDDY K R, RAO A A. Dependency oriented complexity metrics to detect rippling related design defects [J]. ACM SIGSOFT Software Engineering Notes, 2009, 34(4):1 - 7.

[16]REDDY K N, RAO A A. A quantitative evaluation of software quality enhancement by refactoring using dependency oriented complexity metrics[C]// International Conference of Emerging Trends in Engineering and Technology (ICETET), 2009: 1011 - 1018.

[17]Sant' ANNA C, GARCIA A, LUCENA C, Evaluating the efficacy of concern - driven metrics: a comparative study [C]//Assessment of Contemporary Modularization Techniques (ACoM'08), 2008: 25 - 30.

[18]LUNG C H, ZAMAN M. Using clustering technique to restructure programs [C]// International Conference on Software Engineering Research and Practice. Las Vegas: CSREA Press, 2004: 853 - 858.

[19]LUNG C H, XU X, ZAMAN M, et al. Program restructuring using

clustering techniques [J]. The Journal of Systems and Software, 2006,79(9):1261 - 1279.

[20] ALKHALID A, ALSHAYEB M, MAHMOUD S. Software refactoring at the function level using new Adaptive K - Nearest Neighbor algorithm [J]. Advances in Engineering Software, 2010, 41(10 - 11):1160 - 1178.

[21] SRINIVAS S S. Package level software refactoring using A - KNN clustering technique [C]// International Conference on Computing and Control Engineering (ICCCE 2012),2011, 5(3): 276 - 284.

[22] RATZINGER J, SIGMUND T, VORBURGER P, et al. Mining software evolution to predict refactoring [J]. Empirical Software Engineering and Measurement, 2007:354 - 363.

[23] GROSSER D, SAHRAOUI H A, VALTCHEV P. Analogy - based software quality prediction [C]// Quantitative Approaches In Object - Oriented Software Engineering, QAOOSE, 2003,3.

[24] BANERJEE J, KIM W. Semantics and implementation of schema evolution in object - oriented databases [C]//ACM SIGMOD Conference,1987.

[25] OPDYKE W F. Refactoring object - oriented frameworks [D]. Ph. D. thesis, University of Illinois,1992.

[26] 李军超,尹俊文,徐振阳. 基于抽象语法树的代码味道识别工具的分析与设计[J]. 株洲工学院学报. 2005,19(6):53 - 56.

[27] KONTOGIANNIS K A, DEMORI R, MERLO E, et al. Pattem matching for clone and concept detection [J]. Automated Sotrware Engineering, 1996,3(1 - 2):77 - 108.

[28] 刘建宾. 过程蓝图设计方法学[M]. 北京:科学出版社,2005.

[29] 李建忠,刘建宾. 重复代码自动检测工具的研究与设计[J]. 韩山师范学院学报,2006,6:24 - 29.

[30] KATAOKA Y, IMAI T, ANDOU H, et al. A quantitative evaluation of maintainability enhancement by refactoring [C]// International Conference of Software Maintenance, 2002: 576 - 585.

[31] TAHVILDARI L, KONTOGIANNIS K. A metric - based approach to enhance design quality through meta - pattern transformations [C]//European Conference of Software Maintenance and Reengineering,2003:183 - 192.

[32] MURPHY E, BLACK A. Breaking the barriers to successful refactoring [C]//ACM International Conference on Software Engineering. 2008,5:421 - 430.

[33] TSANTALIS N, CHATZIGEORGIOU A. Identification of move method refactoring opportunities [J]. IEEE Transactions on Software Engineering, 2009,35(3):347 - 367.

[34] BANSIYA J, DAVIS C. A hierarchical model for object - oriented design quality assessment [J]. IEEE Transactions on Software Engineering ,2002,28(1):4 - 17.

[35] O'KEEFFE M. Search - based refactoring for software maintenance [D]. PhD Thesis, School of Computer Science and Informatics, University College Dublin, October 2007.

[36] O'KEEFFE M, Ó CINNÉIDE M. Getting the most from search - based refactoring [C]//Genetic and Evolutionary Computation Conference (GECCO'07), 2007:1114 - 1120.

[37] O' KEEFFE M, Ó CINNÉIDE M. Search - based refactoring: an empirical study [J]. Journal of Software Maintenance and Evolution: Research and Practice, 2008, 20(5): 345 - 364.

[38] ABBES M, KHOMH F, GUE W G, et al. An empirical study of the impact of two antipatterns, blob and spaghetti code, on program comprehension [C]//European Conference on Software Maintenance and Reengineering (CSMR), 2011:181 - 190.

[39] YANG L M, LIU H, NIU Z D. Identifying fragments to be extracted from long methods [C]// Asia - Pacific Software Engineering Conference, 2009,12:43 - 49.

[40] MEANANEATRA P, RONGVIRIYAPANISH S. Using software metrics to select refactoring for long method bad smell [C]//International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Infromation Technology, 2011,5:492 - 495.

[41] TSANTALIS N, CHATZIGEORGIOU A. Identification of move method refactoring opportunities [J]. IEEE Transactions on Software Engineering, 2009,35(3):347 - 367.

[42] TSANTALIS N, CHATZIGEORGIOU A. Identification of extract method refactoring opportunities [C]// European Conference on Software Maintenance and Reengineering, 2009,3:119 - 128.

[43] MURPHY E, BLACK A. Breaking the barriers to successful refactoring [C]//ACM International Conference on Software Engineering,2008,5:421 - 430.

[44] RAO A A, REDDY K N. Identifying clusters of concepts in a low cohesive class for extract class refactoring using metrics supplemented agglomerative clustering technique [J]. International Journal of Computer Science Issues, 2011,8(5):185 - 194.

万方数据