基于关联度和特征约束的软件设计模式识别方法

古 辉 张炜星 金 鹏 顾杰杰
(浙江工业大学计算机科学与技术学院 杭州 310023)

摘 要 在程序理解和软件逆向工程研究中,找到准确和快速地描述软件的设计模式和待识别源代码的方法,对于构建合理的设计模式识别框架和高效的识别算法是至关重要的。运用无向图的邻接表和连通分量的原理,提出类与类之间关联度的概念,由待识别源代码构建一个关联类集合,旨在减小设计模式识别算法的搜索空间;根据设计模式的特征,提出基于关联度和特征约束的设计模式识别算法。对Junit、JHotDraw和JreFactory 3个开源应用程序进行的设计模式识别表明,该算法能够准确高效地完成对源代码设计模式的识别。

关键词 程序理解,设计模式识别,关联度,特征约束
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.2.037

Method of Software Design Patterns Identification Based on Correlation and Feature Constraints

GU Hui ZHANG Wei-xing JIN Peng GU Jie-jie
(College of Computer Science and Technology, Zhejiang University of Technology, Hangzhou 310023, China)

Abstract In program comprehension and software reverse engineering research, it is crucial to find a method that can describe software design patterns and source code to be identified accurately and quickly for constructing a reasonable design pattern identification framework and efficient recognition algorithms. With the principle of adjacency list and connected component of undigraph, we presented the concept of correlation between the class in source code, constructed association class collection from source code to be identified so as to reduce the search space of the algorithm of design pattern recognition. According to the features of design patterns, algorithms of design pattern identification based on correlation and feature constraints were proposed. The algorithm was applied to three open sources including applications-Junit, JHotDraw and JreFactory. These results demonstrate the algorithms can accurately and efficiently complete the recognition of design patterns in source code.

Keywords Program comprehension, Design patterns identification, Correlation, Feature constraints

1 引言

现代软件工程由于要适应日益变化的用户需求,越来越重视软件设计的复用性和灵活性,因此对设计模式的使用变得尤为频繁。设计模式往往是一些特定问题的特定解决方案或设计经验,可以帮助开发人员更加简单方便地复用成功的设计和体系结构[1]。同时,作为整个软件体系结构,设计模式的微结构可以从结构上对软件程序进行分析和理解。因此,从源代码中识别出该程序所运用的设计模式具有两个方面的重要意义:(1)根据特定的设计模式解决某些特定问题,从程序功能上帮助维护人员理解该模块在整个软件系统中的作用;(2)根据特定设计模式所体现的结构类图,从软件结构上帮助维护人员进行程序理解,有助于对整个遗留系统软件体系结构的恢复。因此,研究人员在对遗留系统,特别是对面向对象程序系统进行程序理解和逆向工程研究时,从设计模式识别的角度分析源代码结构设计信息是一种有效的途径。

近几年来,国内外研究人员对程序理解中设计模式识别的研究做了大量的工作,其研究内容也越来越深入。现有的相关研究主要包括两个方面:

(1)对设计模式准确描述方法和规则的研究

对设计模式的识别首先要确定设计模式的定义,给出能准确描述任何设计模式的方法,才能在待识别的目标源代码中匹配出符合该设计模式特征的候选微结构。Erich Gramma等从模式名称和分类、意图、别名、动机、适用性、结构、参与者、效果、实现等方面对设计模式进行了完整的描述[1]。该描述虽然是公认的设计模式定义标准,但从程序理解中设计模式识别的角度上看其仍然无法准确识别定位特定的设计模式。为此,文献[2]将已有的设计模式表示成Prolog规则,将待抽取的源代码的设计信息翻译成Prolog事实,而将设计模式识别的过程表达成Prolog查询过程,从而实现设计模式的搜索与识别。文献[3]提出一种由可复用特征类型组成的可变模式定义,用来描述特定的设计模式和源代码信息,然后通过适合特定特征的搜索技术检测特征类型来识别设计模式。文献[4]提出一种基于XML的设计模式标记语言(DPML),

到稿日期:2014-03-14 返修日期:2014-06-24

古 辉(1956-),男,教授,主要研究方向为软件理解与文档自动生成技术、嵌入式应用技术等,E-mail:gh@zjut.edu.cn;张炜星(1988-),男,硕士生,主要研究方向为程序理解技术与软件逆向工程;金 鹏(1990-),男,硕士生,主要研究方向为程序理解与文档自动生成技术;顾杰杰(1991-),男,硕士生,主要研究方向为程序理解与文档自动生成技术。

用来表示设计模式特征。文献[5]在 DPML 的基础上增加由半动态和动态分析得到的特征(DPMLd)来定义和表示设计模式,以提高设计模式识别的准确性,相对于文献[4]有所改进。本文提出了一种基于 XML Schema 技术定义 XML 文档结构的设计模式定义语言——Design Pattern Definition Language based on XML Schema(DPDLXS)[6],DPDLXS 语言将设计模式的信息表示分为 3 个部分,包括描述设计模式中参与角色的类信息〈Roles〉、描述类之间关系的〈Relations〉和描述自定义类型的〈TypeRep〉。通过这 3 个元素节点及其子元素和属性信息可以唯一描述和识别特定的设计模式。

(2)对设计模式识别框架和识别算法的研究

有了能够准确表述设计模式特征和唯一识别特定设计模式的定义方法之后,合理的设计模式识别框架和高效的识别算法是完成设计模式识别的关键。然而,针对程序理解和软件维护的复杂性以及庞大的代码量等特点,不难分析出当前设计模式识别的困难在于以下两个方面:(1)庞大的代码量决定了设计模式识别的搜索空间巨大,导致识别效率不高;(2)代码间复杂的关系决定了设计模式特征匹配的复杂性。因此,如何减小搜索空间以及设计合理的识别框架和算法是相关研究人员的主要工作目标。

文献[7]提出了一种基于多级归约策略的结构型设计模式抽取方法,即将设计模式和源代码都表示成一种称为抽象对象语言(AOL)的中间表示方式,创建 AOL 设计模式知识库;然后通过解析 AOL 知识库得到抽象语法树(AST),从而抽取到设计模式类度量信息;最后采用一种多级规约策略来减小设计模式搜索识别的复杂性,从而识别出候选结构型设计模式。然而,这种设计模式的中间表示方式只是简单地描述每个类角色的属性和方法的数目,难以准确表述特定设计模式的特征信息。文献[8]在抽取结构特征的同时结合数字特征来增强对完全的和不完全的设计模式事件的识别。该方法主要通过数字特征来减小搜索空间,增强结构识别的性能和准确率。对类构建规则和度量等数字特征,从而可以根据特定的数字特征去除搜索空间中不匹配该数字特征的类,提高模式识别效率。

文献[4]提出的模式挖掘算法首先解析 C++源代码并构建抽象语法图(ASG),其次利用自己提出的一系列设计模式挖掘算法将描述设计模式的 DPML 模式表示信息与抽象语法图进行匹配,从而达到识别设计模式的目的。文献[9]通过分析特定设计模式中每个类之间的关系,建立对应的属性模型并且以关系演算语言将设计模式表示为逻辑语言表达式,通过计算表达式中逻辑语句的可满足性来检查程序中是否运用了该设计模式。文献[10]在静态分析源代码信息的同时,强调动态分析 UML 类图格式信息以进行设计模式的检测。在进行动态分析时,该方法利用代码插桩技术以及 UML 序列图来获取运行时动态信息。

综上所述,对设计模式进行识别主要包括两方面的工作:(1)给出准确描述特定设计模式特征的定义方法,该方法可以唯一识别特定设计模式,同时可以描述待识别的源代码信息;(2)抽取源代码信息,设计合理的设计模式识别框架和高效率的识别算法。因此,本文利用 DPDLXS 语言描述设计模式信息和待抽取的源代码信息;提出源代码间关联度的概念,构建基于关联度的关联类集合,旨在减小设计模式的搜索空间;根据设计模式的特征约束和构建的关联类集合,提出基于关联度和特征约束的设计模式识别算法;最后,对构建的关联类集合进行设计模式识别,分析识别结果。

2 源代码信息抽取

设计模式识别是对程序源代码信息进行分析,抽取出其中所运用的设计模式,是软件逆向工程的一部分。因此,设计模式识别的对象一般为面向对象程序系统的源代码信息。本文以 Java 源代码为设计模式识别目标,结合 Java 语言的特点进行源代码信息的抽取。

2.1 源代码信息抽取

根据 DPDLXS 语言的定义规则,将设计模式的特征信息描述为类角色 Roles、类之间关系 Relations 和自定义类型 TypeRep 3 个部分。源代码信息的抽取也遵循 DPDLXS 语言的定义规则,解析为类角色 Roles 和类之间关系 Relations。其中,一个类角色 Roles 又分为类属性 Attribute 和类操作 Operation 子元素以及各自的属性。根据 Java 语言的特点和设计模式描述语言 DPDLXS 的定义规则,给出了源代码信息抽取流程,如图 1 所示。

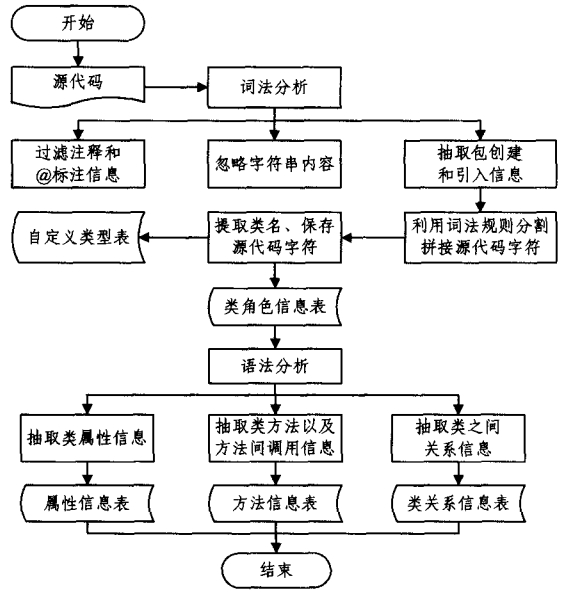


图 1 源代码信息抽取流程

2.2 类无向图和连通分量

对源代码进行设计模式的识别往往都是以一个工程或一个面向对象程序系统为一个整体进行分析,其中包含了数量庞大的类文件。然而,一个设计模式所参与的类角色只有 3—5 个左右,为了匹配一个设计模式的特征而去遍历整个工程的类文件显然是不可取的。

根据设计模式的结构特征,构成一个设计模式的每个角色类至少与其他角色类存在一种关系,包括一般化关系(Generation)、关联关系(Association)、聚合关系(Aggregation)、合成关系(Composition)和依赖关系(Dependency)。从图论的角度上来说,如果把每一个类角色看成是一个顶点(Vertex),类角色之间的任何一种关系看成是两个顶点之间的一条边(Edge),整个设计模式类图就可以看成是一个无向图,而且是一个任意两个顶点都连通的连通图(Connected Graph)[11]。同理,待识别的整个工程的源代码类图就可以看成是一个庞大的无向图,如图 2 所示;无向图中的每一个极大连通子图都是一个连通分量(Connected Component)[11],如

图 3 所示。因此，匹配的某一设计模式特征的待识别类候选集合一定是该工程无向图中一个顶点数目大于等于设计模式角色类数目的连通分量。

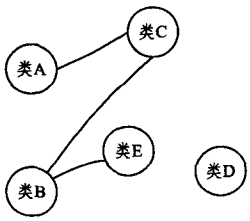


图 2 无向图

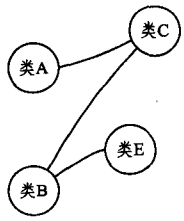
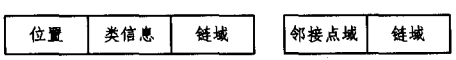


图 3 无向图的一个连通分量

2.3 关联度

通过引用图论中无向图和连通分量的概念，将设计模式的识别从源代码中匹配特定设计模式特征转化成在无向图中查找满足顶点数目的连通分量。通过过滤不满足该设计模式角色类数目的连通分量可以减小候选集合，从而减小搜索空间，提高识别效率。然而，在信息抽取的初级阶段并不知道待识别工程源代码无向图的连通分量。为了方便遍历无向图从而得到该无向图的连通分量，本文将待识别源代码的无向图表示成一种链式存储结构——邻接表[11]。本文根据从源代码信息中抽取出的类之间关系信息来构建无向图，得到无向图的顶点集合 V 和代表类之间关系的边集合 VR。将集合 V 中的每一个顶点都表示成一个头结点，如图 4(a)所示；根据集合 VR 中顶点之间的关系构建该头结点包含的表结点，如图 4(b)所示，从而得到图 2 中无向图对应的邻接表，如图 5 所示。



(a) 头结点　　(b) 表结点

图 4 无向图头结点和表结点

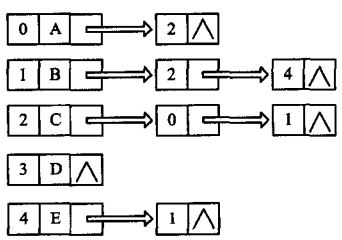


图 5 图 2 无向图的邻接表

运用邻接表这种链式存储结构，从第一个头结点开始遍历可以得到无向图的所有连通分量。无向图连通分量的具体深度遍历算法如下：

Step1 遍历头结点，设置该头结点的 visited 为 true；若该头结点的链域为空而且是第一个头结点，则结束；若不是第一个头结点则返回上一层表结点，跳到 Step4；若链域不为空则继续下一步。

Step2 遍历链域所指向的表结点，根据表结点的邻接点域找到下一个头结点。

Step3 若该头结点的 visited 为 true，则返回上一层表结点，跳到 Step4；否则跳到 Step1。

Step4 遍历表结点的链域，若链域不为空则跳到 Step2；若链域为空，则返回上一层表结点并判断是否存在上一层，若存在则跳到 Step4；否则结束。

该算法一次遍历结束后，所有遍历到的顶点集合就是一个连通分量。然后寻找下一个 visited 为 false 的头结点作为第一个结点继续遍历，直到遍历完所有头结点。利用该算法对图 5 所示邻接表进行遍历可以得到两个连通分量，其顶点集合分别为：$S1\{A,C,B,E\}$ 和 $S2\{D\}$。

每一次遍历都可以得到一个连通分量，每个连通分量都是待识别源代码类图中至少与其他类存在一个关系的所有类集合。该集合中的类彼此之间存在着可能构成某种设计模式的类关系。因此，本文提出用关联度(Correlation)的概念来衡量和唯一标识待识别源代码之间的这种联系。每一个关联度值标识了一个连通分量，连通分量中的各个类拥有相同的关联度值，其集合称为关联类集合。

3 设计模式识别

在完成源代码信息抽取(包括类信息、类的属性和操作，特别是根据类之间的关系构建了关联类集合)之后，需要根据具体设计模式特征来检测和识别源代码中运用的设计模式。

3.1 设计模式识别流程

为了减小设计模式识别的搜索空间，本文在源代码信息抽取阶段将待识别源代码类组合成一个个关联类集合。根据设计模式角色类之间的关系特征可以得出：只有同一个关联类集合中存在该设计模式的所有角色类，并且关联类之间符合对应角色类之间的所有关系特征，才能判定关联类集合运用该设计模式。因此，首先可以根据设计模式角色类数目过滤类数目不足的关联类集合；其次可以根据设计模式所蕴含的角色类之间的关系特征过滤不满足类之间关系类型和关系数目的集合；最后根据设计模式的具体特征约束遍历满足条件的候选关联类集合，得出最终的识别结果。具体的识别流程如图 6 所示。

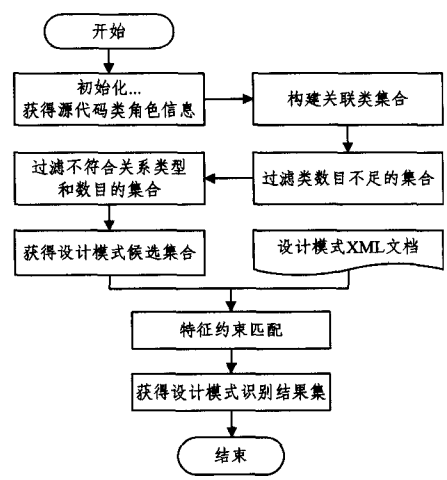


图 6 设计模式识别流程

3.2 基于关联度和特征约束的设计模式识别算法

根据本文提出的源代码类角色信息间关联度的概念以及设计模式识别的流程，充分利用关联类集合和设计模式特征约束来减小设计模式识别的搜索空间，可以得出基于关联度和特征约束的设计模式识别算法 DETECT_DESIGNPATTERNS，具体描述如算法 1 所示。

算法 1 基于关联度和特征约束的设计模式识别算法 DETECT_DESIGNPATTERNS

Algorithm DETECT_DESIGNPATTERNS(SC,RC,DR)

Input:SC[0,…,n],RC[0,…,m],DR[0,…,j]

```
Output:DP[0,…,k]
Begin
1. foreach RC∈SC
2.   if RC. size ≥ DR. size then
3.     if isMeetRelation(RC,DR) then
4.     foreach class RC[i]∈RC
5.     if match the feature of DR then
6.       add DR to DP
7.     end if
8.     end if
9.   end if
End
function isMeetRelation(RC,DR):bool
1. if Num(Generation,RC)≥Num(Generation,DR) then
2.   if Num(Association,RC)≥Num(Association,DR) then
3.     if Num(Dependency,RC)≥Num(Dependency,DR) then
4.       return true
5.     end if
6.   end if
7. end if
8. return false
endfunc
SC:set of all class in source code
RC:set of relevant class
DR:set of design patterns roles
DP:set of result about detect design patterns
Num(R,C):the number of relationKind R in class set C
```

3.3 Proxy 模式特征约束

本文以 Proxy(代理)模式为例,对满足代理模式关联度的所有关联类集合进行特征约束匹配。

3.3.1 代理模式特征描述

根据文献[1]对设计模式的概括以及代理模式类图(如图 7 所示)的描述,可以得出代理模式的特征如下:

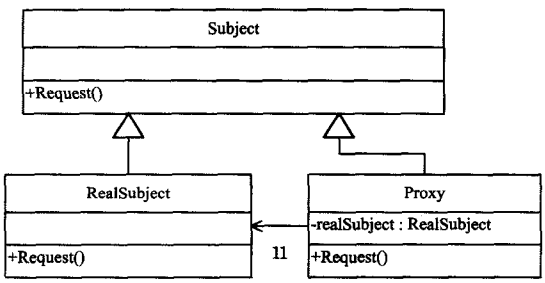


图 7 代理模式类图[12]

(1)一个代表抽象主题的 Subject 抽象类有抽象方法 Request;

(2)一个代表真实主题的 RealSubject 类和 Proxy 类都继承自 Subject 类;

(3)RealSubject 类、Proxy 类和 Subject 类有共同的方法 Request;

(4)Proxy 类有 RealSubject 类型的数据成员,即 Proxy 类和 RealSubject 类存在关联关系;

(5)Proxy 类的 Request 方法实现 Subject 类的 Request 方法,并且该 Request 方法调用 RealSubject 类的 Request 方法。

3.3.2 代理模式特征约束算法

在得到代理模式的特征描述之后,就可以实现基于关联度和特征约束的设计模式识别算法的最后部分——判断是否满足特征约束。特征约束识别是设计模式识别过程中非常关键并且最复杂的部分,主要完成算法 1 中第 5 行的特征匹配。通过合理调整代理模式特征约束的顺序,在遍历关联类集合的时候可以尽可能地提前过滤不满足的关联类集合,从而提高识别的效率。经过调整后的代理模式特征约束算法如下:

Step1 查找一个抽象类;

Step2 查找该抽象类的派生类集合,并且派生类数目至少存在两个;

Step3 遍历每个派生类,查找与派生类有关联(Association)关系的类,即在该派生类中充当数据成员的类;

Step4 对存在关联关系的类进行筛选,该类必须也是之前抽象类的派生类;

Step5 判断两个派生类之间的方法调用情况(一个派生类方法要调用另一个派生类方法,并且方法都重写抽象类的方法):

a)遍历抽象类的所有抽象方法;

b)判断两个派生类是否重写该抽象方法;

c)查找派生类的重写方法中调用的所有方法集合;

d)遍历方法调用集合,判断是否存在另一个派生类的重写方法,若存在,则满足代理模式的特征约束;否则不存在代理模式,结束本算法。

4 结果和分析

根据所提出的源代码信息抽取流程以及基于关联度和特征约束的设计模式识别算法,本文对 Junit、JHotDraw 和 Jrefactory 3 个开源应用程序进行信息抽取和部分设计模式的识别。

4.1 工厂方法模式

基于提出的源代码信息抽取流程(见图 1),本文从源代码文件数目、类、属性、方法和类之间的关系数目以及源代码行数等方面对 3 个开源应用程序源代码进行信息抽取。其抽取结果如表 1 所列。

表 1 源代码信息抽取结果

Code info	Junit	JHotDraw	JreFactory
Files	115	485	1173
LOC	12937	49823	154183
Classes	115	485	1173
Attributes	123	822	2881
Operations	643	4838	9774
Generation	2	86	1090
Association	1	43	412
Dependency	3	138	980

从表 1 的抽取结果可以看出,这 3 个开源应用程序在文件数目以及代码复杂性等方面各不相同。类之间存在的关系越多,体现了源代码结构复杂度越高,同时也意味着可能运用了更多的设计模式。

4.2 设计模式识别结果

针对表 1 的源代码信息抽取结果,我们利用基于关联度和特征约束的设计模式识别算法,结合待识别设计模式的具体特征约束,依据本文的设计模式识别流程得出 Junit、JHotDraw 和 JreFactory 的部分设计模式识别结果,如表 2 所列。

(下转第 203 页)

2012,22(7):1-14
[7] 学者网[EB/OL]. http://www.scholat.com
[8] Shi X X, Li Y, Yu P S. Collective prediction with latent graphs [C]// Proceedings of the 20th ACM Internation conference on Information and knowledge management. ACM, 2011: 1127-1136
[9] Aggarwal C. Social network data analytics[M]. Springer press, Berlin, German, 2011
[10] Rabelo J, Prudencio R B C, Barros F. Collective classification for sentiment analysis in social networks[C]// Proceedings of the 24th International Conference on Tools with Artificial Intelligence. IEEE, 2012, 1: 958-963
[11] Laorden C, Sanz B, Santos I. Collective classification for spam filtering[C]// Proceedings of the 4th International Conference on Computational Intelligence in Security for Information Systems. Springer. 2011: 1-8
[12] Guan J H, Liu H, Xiong W, et al. Effectively predicting protein functions by collective classification-An extended abstract[C]// Proceedings of the 2012 IEEE International Conference on Bioinformatics and Biomedicine Workshops. IEEE, 2012: 634-639
[13] McDowell L, Gupta K M, Aha D W. Cautious collective classification[J]. Journal of Machine Learning Research, 2009, 10(12): 2777-2836
[14] Neville J, Jensen D. Iterative classification in relational data [C]// Proceeding of the AAAI 2000 Workshop on Statistical Relational Learning of the National Conference on Artificial Intelligence. 2000: 42-49
[15] Kazienko P, Kajdanowicz T. Label-dependent node classification in the network[J]. Neurocomputing, 2012, 75(1): 199-209
[16] Macskassy S A, Provost F J. A simple relational classifier[C]// Proceedings of the 2nd Workshop on Multi-Relational Data Mining. 2003: 64-76
[17] Kibriya A M, Frank E, Pfahringer B, et al. Multinomial naive bayes for text categorization revisited[C]// Proceedings of 17th Australian Joint Conference on Artificial Intelligence. Springer, 2005: 488-499
[18] Su J, Shirab J S, Matwin S. Large scale text classification using semisupervised multinomial naive bayes[C]// Proceedings of the 28th International Conference on Machine Learning. 2011: 97-104
[19] de Campos L M, Fernández-Luna J M, Huete J F, et al. Link-based text classification using bayesian networks[J]. Lecture Notes in Computer Science, 2010, 6203: 397-406
[20] Kong X, Shi X, Philip S Y. Multi-label collective classification [C]// Proceedings of 2011 SIAM International Conference on Data Mining. SDM, 2011: 618-629

(上接第 176 页)

表 2 设计模式识别结果

DesignPatterns	Junit	JHotDraw	JreFactory
FactoryMethod	—	1	2
Proxy	—	—	—
Strategy	1	9	17
AdapterClass	—	—	4
AdapterObject	—	10	34
Observer	—	6	7
Total	1	26	64
Time(ms)	9774	48874	111856

通过对部分设计模式识别结果分析得出，该设计模式识别方法还存在一定的不足：(1)对设计模式识别的准确率有待提高。待识别源代码类结构和设计模式之间的特征匹配如果不够充分就会增加识别结果的错误肯定率；反之，则会增加识别结果的错误否定率。(2)对设计模式异构体或变体的识别考虑不足。每一个设计模式虽然都有一个标准和公认的结构特征，但往往有多种不同的特征表现形式，例如观察者(Observer)模式。这种不同的表现形式可以称为异构体或变体，这对设计模式的识别提出了更高的要求。因此，今后需要对设计模式异构体之间的特征做更细致的分析，从而得到更准确高效的设计模式识别方法。

结束语 本文利用 DPDLXS 语言描述设计模式信息和待抽取的源代码信息，提出源代码间关联度的概念，构建基于关联度的关联类集合，旨在减小设计模式的搜索空间，一定程度上提高识别效率；根据设计模式的特征约束和构建的关联类集合，提出基于关联度和特征约束的设计模式识别算法；最后，将该设计模式识别算法应用于 Junit、JHotDraw 和 JreFactory 3 个开源应用程序的设计模式识别，结果表明该设计模式识别算法具有较高的召回率和效率。

参 考 文 献

[1] Gamma E, Helm R, Johnson R, et al. Design Patterns-Elements of Reusable Object-Oriented Software[M]. New Jersey: Addison-Wesley, 1995
[2] Krämer C, Lutz Prechelt. Design recovery by automated search for structural design patterns in object-oriented software[C]// Proceedings of the Third Working Conference on Reverse Engineering. 1996: 208-215
[3] Rasool G, Mader P. Flexible Design Pattern Detection Based on Feature Types[C]// 26th IEEE/ACM International Conference on Automated Software Engineering (ASE). 2011: 243-252
[4] Balanyi Z, Ferenc R. Mining Design Patterns from C++ Source Code[C]// Proc. Internationl Conf. on Software Maintenance (ICSM'03). 2003: 305-314
[5] Dobis M, Majtas L. Mining Design Patterns from Existing Projects Using Static and Run-Time Analysis[J]. Springer Software Engineering Techniques, 2011, 4980: 62-75
[6] 古辉，张炜星. 基于 XML Schema 技术的设计模式定义方法[J]. 计算机科学，2014，41(1)：254-257
[7] Antoniol G, Fiutem R, Cristoforetti L. Design pattern recovery in object-oriented software[C]// Proceedings of 6th IEEE International Workshop on Program Comprehension (IWPC 1998). 1998: 153-160
[8] Gueheneuc Y G, Guyomarc'h J Y, Sahraoui H. Improving design-pattern identification: a new approach and an exploratory study[J]. Software Quality Journal, 2010, 18(1): 145-174
[9] 苗康，余啸，赵吉，等. 基于关系演算的 Java 模式识别[J]. 计算机应用研究，2010，27(9)：3425-3430
[10] Li F, Li Q S, Su Y, et al. Detection of design patterns by combining static and dynamic analyses[J]. Journal of Shanghai University(English Edition), 2007, 11(2): 156-162
[11] 严蔚敏，吴伟民. 数据结构(c 语言版)[M]. 北京：清华大学出版社，1997
[12] 阎宏. Java 与模式[M]. 北京：电子工业出版社，2002