文章编号: 1000-5862(2015)04-0377-06

基于演化依赖的 Java 软件聚类实现技术研究

钟林辉,李俊杰,张能伟,黄小明
(江西师范大学计算机信息工程学院,江西 南昌　330022)

摘要: 传统的软件聚类方法没有考虑软件实体间存在一些无法通过静态分析手段获取的关系(即演化依赖关系),这将导致聚类后的软件可能不符合"高内聚、低耦合"特征. 为了解决上述问题,提出了将软件演化信息纳入软件聚类过程的策略,并在扩展的 Java 依赖模型的基础上,实现了一个基于模拟退火的软件聚类算法. 实验结果表明: 该方法能达到提高软件聚类准确度的目的.

关键词: 软件聚类; 演化信息; 高内聚; 低耦合
中图分类号: TP 311　　**文献标志码**: A　　**DOI**: 10.16357/j.cnki.issn1000-5862.2015.04.09

0　引言

模块化的软件设计思想是尽可能地将软件系统划分成若干相对独立的、由软件元素构成的、具有自包含功能的可复用模块. 通过这种设计,不仅能实现软件复用的目的,而且能较容易地处理和适应软件的变化. 软件聚类是软件模块化设计过程中常见的一种划分方法,在划分的过程中通常将程序抽象成(带权)有向图,然后利用相似性度量[1]、搜索、概念格[2-3]以及模式匹配[4-5]等技术实施聚类.

基于搜索的方法将软件聚类看作是求解满足指定目标函数的最优化问题. 根据目标函数个数的不同,可以区分为单目标函数的软件聚类和多目标函数的软件聚类. (i) 单目标函数的软件聚类: 如文献[6]提出的 Bunch 方法是一种基于模块质量(MQ)的层次聚类方法; 文献[7]通过将边的权重纳入模块化函数中并定义了扩展的模块化函数 Q_d,基于此实现了面向特征的软件聚类算法 SArF; C. Zahn 等在聚类过程中使用最小生成树作为聚类的目标函数[8-9]. (ii) 多目标函数的软件聚类: 文献[10]提出的软件聚类方法采用的目标函数是多元的,即划分数目和每个划分的规模需要同时满足指定条件. 文献[11]中采用多目标演化算法,在尽可能保持 Java 包原有设计决策的基础上实现 Java 程序的重构,其需要满足的目标是包内内聚的最大化、包间耦合的最小化以及包间循环依赖的最小化; 文献[12]中提出了满足独立性度量和划分规模的有向图分解方法; 文献[14]提出了满足包的结构性耦合度量(ICP)和语义相关性度量(CCBC)的聚类方法.

但是,上述这些方法因为主要考虑的是软件元素之间的静态关系(例如函数之间的调用关系),而静态关系的获取通常存在花费时间较高,信息不够完全的缺点,这导致软件划分的结果不够理想[3]. 为此,许多研究者考虑在聚类的过程中额外增加其他一些辅助信息[13~14]. 如在文献[15]中将注释、文件,类或者方法的名字、文件修改时间、作者、目录结构等信息纳入到聚类过程,而有的研究者则利用体系结构层面的有关信息例如设计模式、系统层次模型等作为软件聚类的启发信息[16-17].

为了提高划分的准确性,本文考虑将软件演化信息纳入到软件之间的依赖关系中,形成扩展的软件依赖模型,并利用模拟退火算法实现该模型上的最优化划分.

1　扩展的 Java 程序依赖模型 EDM

在 EDM Extended Dependency Model 模型中,类被抽象成为有向图中的节点,节点之间的有向边表示类间的逻辑依赖关系和演化依赖关系.

收稿日期: 2015-02-21
基金项目: 国家自然科学基金(61262015,61462040),江西省自然科学基金(20142BAB207027,20142BAB207011),江西省科技支撑项目(20142BBE50028)和江西省教育厅科学技术(GJJ13230)资助项目.
作者简介: 钟林辉(1974-),男,江西赣州人,副教授,博士,主要从事构件化软件、软件体系结构、软件自动化和软件演化的研究.

定义 1 扩展的程序依赖模型 EPDM 是一个二元组 $\langle C,E\rangle$. 其中 C 表示面向对象程序中所有的类集合; $E\subset C\times C$,表示类之间存在的逻辑或者演化依赖关系.

定义 2 逻辑依赖关系包括不同类之间的直接继承关系、聚合关系、关联关系,以及类之间方法调用关系和数据访问关系. 例如类 A 中的方法访问了类 B 中的数据,或者调用了类 B 中的方法,则类 A 和类 B 存在逻辑依赖关系.

定义 3 演化依赖关系指的是 Java 程序在演化过程中,类之间存在的蕴含依赖关系,即若类 C_1, $C_2,\cdots,C_k$在程序的若干个版本中同时发生了变化,则称这些类具有演化依赖关系. 在 EDM 模型中体现为$\{C_1,C_2,\cdots,C_k\}$ 中任意 2 个类存在一条双向边.

可以将演化依赖关系分为单变化演化依赖关系和多变化演化依赖关系. 多变化演化依赖关系是在单变化演化依赖关系的基础上,通过数据分析技术得到的. 下面首先定义单变化依赖关系,然后介绍如何用多变化演化依赖关系及其选择算法.

1) 单变化演化依赖关系.

单变化演化依赖关系 $R_i=\langle\{C_1,C_2,\cdots,C_k\},\{V_i\}\rangle$,表示 Java 程序从第 $i-1$ 个版本变化到第 i 个版本时,类 $C_1,C_2,\cdots,C_k$发生了变化.

2) 多变化演化依赖关系.

给定一个变化事务序列 $R_i,\cdots,R_j$,多变化事务的演化依赖关系定义为 $R_{i,\cdots,j}=\langle\{C_1,C_2,\cdots\cdots,C_k\},\{V_i,\cdots,V_j\}\rangle$,表示对任意的程序版本 V_k,$V_k\in\{V_i,\cdots,V_j\}$,在程序从第 V_{k-1} 版本变化到 V_k 版本时,类 $C_1,C_2,\cdots\cdots,C_k$同时发生了变化.

在以前的研究中[18-19],可以将所有的多变化演化依赖关系体系在概念格中,下面给出选择大于指定阈值的多变化演化依赖关系的算法.

算法: 多变化演化依赖关系选择算法 SelectMCD.

输入: 多变化演化依赖关系的概念格 K(在这里 K 采用有向图的邻接表表示形式),阈值 Value.

输出: 满足大于等于阈值 Valued 的多变化演化依赖关系,结果存放在集合 MCDSet 中.

伪代码:

```
/* MCDSet 存放满足条件的多变化依赖关系 */
MCD [] MCDSet;
int intvisited [];//标识节点是否被访问
void SelectMCD (LinkedGraph * k,int value,int i)
{
MCD * p;
    (i) 返回 MCD 中类的集合
    ClassSetcs = return ClassesOfMCD(k.adjlist [i]);
    (ii) 返回 MCD 中版本的集合
    VersionSet vs =
        return VersionsOfMCD(k.adjlist [i]);
    (iii) 判断类集合中元素个数及版本集合中元素个数是否满足指定的条件. 其中 M 为程序中类的个数,N 为系统演化的版本个数.
    if (sqart(cs.nums* cs.nums /M* M +
    vs.nums* vs.nums /N* N)) > = Value)
    visited [i] =1;
    (iv) 从当前节点开始,进行图的深度优先遍历,即
    p = k.adjlist [i].FirstEdge;
    while (p) {
    if (! visited [p->adjvex])
    SelectMCD(k,value,p->adjvex);
    p = p->next;
    }
}.
```

2 基于模拟退火的 Java 软件聚类算法

本节首先介绍基于模拟退火思想的 Java 软件聚类算法,然后通过一个例子说明基于扩展的 Java 依赖关系模型的软件聚类算法的有效性.

2.1 算法

模拟退火算法的基本思想是通过模拟物体的退火过程进行寻优计算,其程序框架如下:

```
Algorithm Annealing<D,Z>(d: D; tmax,tmin,
a: real; kmax: int; f: Z->real; Init: D->Z; N: Z->
Set(Z))
begin
  letx = Init(d),t = tmax,xbest = x;
  while (t > tmin) do
    let k = 0;
while (k < kmax) do
      if (|N(x)| =0) then break;
let r = Random(0,|N(x)|),x´= N(x) [r],
delta = f(x) -f(x´);
if (delta <0) \/ (exp(-delat/t) <
  Random(0,1)) then
x <-x´;
if (f(x) < f(xbest)) then xbest <-x;
  //更新最优解
k <-k +1;
t <-a* t;
```

return xbest;
end.

上述框架中的核心问题是针对具体的应用,设计出对应的目标函数 f,以及如何随机产生新的解 x'. 本文目标函数和随机产生解 x'的策略如下:

1) 目标函数 $f = MQ$.

$$MQ = \begin{cases} \frac{1}{k}\sum_{i=1}^{k} A_i - \frac{1}{\frac{k(k-1)}{2}}\sum_{i,j=1}^{k} E_{i,j}, & \text{若 } k > 1, \\ A_1, & \text{若 } k = 1. \end{cases}$$

本文选择文献 [6] 中提出的模块质量函数 MQ 作为模拟退火算法的目标函数 f,MQ 是介于 -1 到 1 之间的一个值. 一般,MQ 的值越大,其代表的聚类效果越符合“高内聚、低耦合”的软件工程原则. 其中 A_i 为第 i 个子图的内部关联度,子图 i 的内部关联度由组成子图的 N_i 个节点以及节点之间的边关联度 μ_i 决定; $E_{i,j}$ 为第 i 个子图以及第 j 个子图之间的关联度,$E_{i,j}$ 由节点个数 N_i,N_j 以及第 i 个子图和第 j 个子图的边关联度 $\varepsilon_{i,j}$ 决定.

2) 随机产生新的解 x'的策略.

软件聚类的一个划分代表的是一个解. 在给定划分数的情况下,本文通过不同划分子集之间元素的交换、移动等操作产生一个新的解. 即首先随机地选择 2 个划分子集 P_1 和 P_2,以及操作类型(交换或者移动操作),然后根据选择的操作类型进行 P_1 和 P_2 中元素的调整: 若操作类型是交换,则随机的在 P_1 和 P_2 中各选择一个元素进行交换; 若是移动操作则随机的在 P_1 或者 P_2 中选择一个元素,将其移动到 P_2 或者 P_1 中.

Java 软件聚类算法包括 annealing1 以及 annealing2,annealing1 和 annealing1 的不同之处在于第 1 个参数,annealing1 的第一参数是指定的划分子集个数,annealing2 的第 1 个参数是按照指定划分子集个数构造的第 1 个划分.

算法 annealing1.

输入: 待划分的个数 num,图 g,最高温度 $tmax$,最低温度 $tmin$,阈值 a,最大迭代次数 $kmax$.

输出: 图 g 的一个划分.

伪代码:

```
GraphPartion annealing1(int num,Graph g,float tmax,float tmin,float a,int kmax)
{
(i) 若 num 大于 g 中节点个数,则报错;
(ii) 生成一个新的类型,初始划分 init,即
GraphPartion init = new GraphPartion(g);
(iii) 产生第 1 个划分子集个数为 num 的划分,其方法是前 num - 1 个划分子集只包含一个节点,第 num 个划分子集包含剩下的所有节点. 即
int i = 0;
for(;i < = num - 2;i + +){
NodeSet k1 = new NodeSet();
k1.add(i);
init.add(k1);
}
NodeSet k1 = new NodeSet();
for(;i < = g.getNodeNum() - 1;i + +)
k1.add(i);
init.add(k1);
(iv) 调用算法 annealing2
return annealing2(init,tmax,tmin,a,kmax);
}.
```

算法: annealing2.

输入: 初始划分 initPartion,最高温度 $tmax$,最低温度 $tmin$,阈值 a,最大迭代次数 $kmax$.

输出: 图 g 的一个划分.

伪代码:

```
GraphPartion annealing(GraphPartioninitPartion,float tmax,float tmin,float a,int kmax)
{
GraphPartion x,x1,xbest;
float t:
xbest = initPartion;
t = tmax; x = xbest.Clone();
while(t > tmin){
int k = 0;
(i) 分别构造用于选择划分子集,选择操作,划分子集中元素的随机器
Random rand = new Random();
Random rand1 = new Random();
Random rand2 = new Random();
(ii) 如果迭代次数小于设定的最大迭代此次,循环产生新解,并根据目标函数及临界值进行替换.
while (k < kmax){
(iii) 随机在当前划分中选择 2 个不同的划分之集,分别存储 在 firstNodeSet 和 secondNodeSet 中,即
int num = initPartion.getSize();
int numOfFirstSet = rand.nextInt(num);
int numOfSecondSet = randomGenerateSet(rand,numOfFirstSet);
x1 = x.Clone();
```

```
firstNodeSet = x1. getNodeSet( numOfFirstSet) ;
secondNodeSet = x1. getNodeSet ( numOfSecond-
Set) ;
```

(iv) 如果随机器 rand1 产生的操作是“移动”操作,执行以下步骤: 若第 1 个划分子集元素个数大于 1,则随机获取第 1 个划分子集中的一个节点 k1,将其从 firstNodeSet 中剔除,加入到 secondNodeSet 中;

(v) 如果操作类型是“交换”,利用随机器 sand2 分别在 firstNodeSet 和 secondNodeSet 中选择 2 个节点 k1 和 k2,并将其交换. 即

```
{
    k1 = firstNodeSet. getNode( numOfNode( rand2) ) ;
    secondNodeSet. add( k1) ;
    firstNodeSet. delete( k1) ;
    k2 = secondNodeSet. getNode( numOfNode( rand2) ) ;
    firstNodeSet. add( k2) ;
    secondNodeSet. delete( k2) ;
    }
```

(vi) 按模拟退火思想进行下一次迭代

```
    float delta = x. calMQ( ) - x1. calMQ( ) ;
    if( ( delta < 0) ||
       ( Math. exp( -delta / t) <
       Math. random( ) ) ) {
    x = x1;
    if( x. calMQ( ) > xbest. calMQ( ) ) xbest = x;
        }
    k + + ;
    }
    t = a* t;
    }
    return xbest;
}.
```

2.2 举例

考虑一个支持多文法状态机的编译器,该编辑器采用多种不同程序设计语言进行编码和组装,经历了多次变化,形成了多个版本[20]. 其经过静态分析得到的依赖关系图如图 1 所示. 其中构件 LL,构件 LR_1,构件 LR_2是支持不同文法的 3 个状态机; 构件 $SemLR_1$,构件 $SemLR_2$和构件 SemLL 则是相应的语义变换处理规则; 构件 TLRTuning 负责配置和调用合适的文法状态机和相应的语义变化处理规则,例如,构件 LL 必须和 SemLL 配合使用.

最初的编译器体系结构在经过若干的版本变化后,其多变化的演化依赖关系如下:

$\langle\{LR_1, SemLR_1\}, \{v_1, v_2, v_3\}\rangle$; $\langle\{LR_2, SemLR_2\}, \{v_1, v_2, v_3, v_4\}\rangle$; $\langle\{LL, SemLL\}, \{v_2, v_3, v_5\}\rangle$.

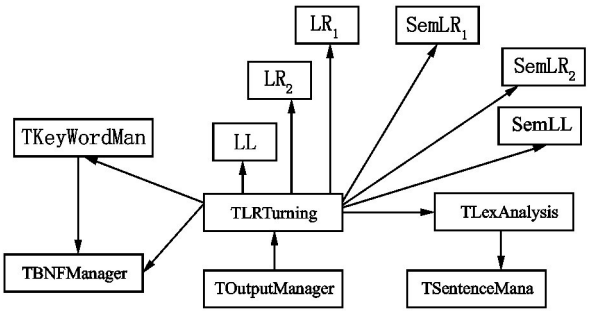

图 1 支持多种文法的编译器

表 1 给出了针对不同指定划分个数,添加演化依赖关系前后所计算出的 MQ 值. 从表 1 可以看出,当指定的划分数为 1 到 8 时,添加演化信息后经过聚类得到的 MQ 值比未添加演化依赖关系时要高,说明划分数 1 ~ 8 时的聚类效果要好.

表 1 不同划分数对应的 $MQ \times 100\%$ 值

划分数	未加演化依赖	加了演化依赖
1	8.333 300	10.416 600
2	14.999 999	16.499 999
3	17.187 500	18.750 000
4	12.585 033	20.138 888
5	9.861 112	16.300 000
6	7.999 999	17.500 000
7	6.547 619	10.119 048
8	2.343 750	4.017 800
9	3.125 000	0
10	-0.277 700	-3.333 333

另外,MOJO 是用来度量不同图之间差异性的计算公式[21-22],本身 MOJO 是通过计算 2 个图之间的编辑距离来判断图之间的差异性,MOJO 值越大,说明 2 个图之间的差异性也大. 为了更客观地衡量软件聚类的准确性和不同划分的相似性,本文定义 $R\_MOJO(i,j) = (1 - MOJO(i,j))/N$,其中 N 为 2 个图节点个数的最大值. 可以看出,R_MOJO 值越大,表示其 2 个划分更相似. 例如在表 2 中,当目标划分数为 3 时,未加入演化依赖关系前的划分同专家识别的软件划分相似性为 75%,同添加演化依赖关系后的划分与专家识别的划分相似度相同(也是 75%); 当目标划分数为 6 时,加入演化依赖关系后的划分同真实的划分相同(R_MOJO 值为 100%),而未添加演化依赖关系的划分同真实的划分 R_MOJO 值则为 75%,由此可以得出添加演化依赖关系后的软件聚类在准确度上要比未添加演化依赖关系的软件聚类要高. 从图 3 中亦可判断,当在软件聚

类过程中添加演化依赖关系,所得的到软件划分更接近真实的软件划分.

表 2　不同划分数对应的 R_MOJO × 100% 值

划分数	未加演化依赖	加了演化依赖
1	58.333 300	58.333 300
2	66.666 667	66.666 667
3	75.000 000	75.000 000
4	75.000 000	83.333 333
5	66.666 666	75.000 000
6	75.000 000	100.000 000
7	75.000 000	83.333 333
8	66.666 660	75.000 000
9	66.666 660	66.666 660
10	58.333 330	58.333 330

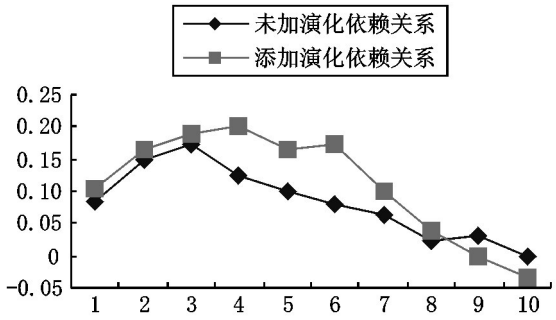

图 2　添加演化信息前后 *MQ* 对比图

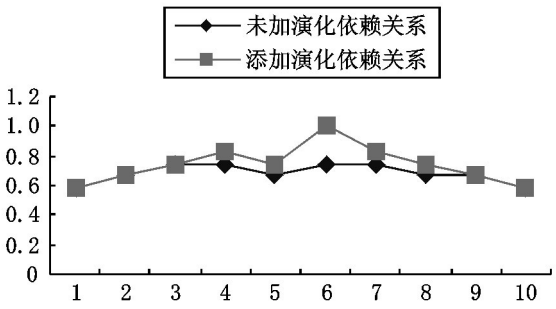

图 3　添加演化信息前后 R_MOJO 值对比图

3　结束语

软件演化依赖关系是一种重要的、软件实体间蕴含的演化信息,能起到克服静态分析方法花费时间较高,提高软件分析准确度的目的,在近几年得到了广泛的重视和研究. 许多研究者从各种角度研究软件演化信息在软件开发和维护过程中的应用,例如将怀疑有设计缺陷的类的演化依赖关系以及单个系统本身信息相结合,能检查出了“数据类(Data Class) ”和“上帝类(God Class) ”2 种具体的面向对象设计缺陷.

类似地,为了弥补传统程序依赖模型只考虑程序元素静态关系以及静态分析方法代价较高的不足,本文将软件演化依赖关系纳入程序依赖模型当中,并基于模拟退火思想实现了一个软件聚类算法,通过实验验证了软件演化依赖关系能提高软件聚类的准确性. 目前的实验还仅限于较小的软件系统,在今后的工作中将改进算法的性能并运用到较大型软件的开发和维护工作中.

4　参考文献

[1] Mitchell B S,Mancoridis S. Comparing the decompositions produced by software clustering algorithms using similarity measurements [C]. Florence: IEEE Computer Society Press,2001: 744-753.

[2] LindigC,Snelting G. Assessing modular structure of legacy code based on mathematical concept analysis [EB/OL]. [2014-12-27]. http: // citeseerx. ist. psu. edu/viewdoc/summary? doi = 10. 1. 1. 133. 9719.

[3] Siff M,Reps T. Identifying modules via concept analysis [J]. Software Engineering,1999,25(6) : 749-768.

[4] Bauer M,Trifu M. Architecture-aware adaptive clustering of OO systems [EB/OL]. [2014-11-18]. http: // citeseerx. ist. psu. edu/showciting? cid = 3677504.

[5] Sartipi K,Kontogiannis K. A graph pattern matching approach to software architecture recovery [EB/OL]. [2014-11-19]. http: // citeseerx. ist. psu. edu/viewdoc/summary? doi = 10. 1. 1. 85. 1552.

[6] Mancoridis S,Mitchell B,Chen Y,et al. Bunch: a clustering tool for the recovery and maintenance of software system structures [EB/OL]. [2014-11-18]. http: // citeseerx. ist. psu. edu/viewdoc/summary? doi = 10. 1. 1. 142. 3736.

[7] Kenichi Kobayashi,Manabu Kamimura,Koki Kato,et al. Feature-gathering dependency-based software clustering using dedication and modularity [EB/OL]. [2015-01-16]. http: // arxiv. org/abs/1306. 2096.

[8] Zahn C. Graph-theoretical methods for detecting and describing gestalt clusters [J]. IEEE Transactions on Computers,1971,20: 68-86.

[9] Bauer M,Trifu M. Architecture-aware adaptive clustering of OO systems [EB/OL]. [2015-01-13]. http: // citeseerx. ist. psu. edu/showciting? cid = 3677504.

[10] Praditwong K,Harman M,Yao X. Software module clustering as a multi-objective search problem [J]. Software Engineering,2010,37(2) : 264-282.

[11] Abdeen H,Sahraoui H,Shata O,et al. Towards automatically improving package structure while respecting original design decisions [EB/OL]. [2015-01-16]. http: // citese-

erx. ist. psu. edu/viewdoc/summary? doi = 10. 1. 1. 412. 1416.

[12] 罗景,赵伟,秦涛,等. 基于有向带权图迭代的面向对象系统分解方法 [J]. 软件学报,2004,15(9):1292-1299.

[13] Gabriele Bavota,Andrea De Lucia,Andrian Marcus,et al. Using structural and semantic measures to improve software modularization [J]. Empirical Software Engineering,2013,18(5):901-932.

[14] Pollet D,Ducasse S,Poyet L,et al. Towards a process-oriented software architecture reconstruction taxonomy [J]. Software Maintenance and Reengineering,2007(1):137-148.

[15] AndritsosP,Tzerpos V. Information-theoretic software clustering [J]. IEEE Trans Software Eng,2005,31(2):150-165.

[16] Pinzger M,Gall H. Pattern-supported architecture recovery [C]. Paris. Paris: IEEE Computer Society Press,2002:53-61.

[17] Sora I,Glodean G,Gligor M. Software architecture reconstruction: an approach based on combining graph clustering and partitioning [EB/OL]. [2015-01-15]. http://citeseerx. ist. psu. edu/viewdoc/summary? doi = 10. 1. 1. 428. 9827.

[18] 钟林辉,姚昕凡,徐静,等. 基于扩展的构件依赖关系图聚类的体系结构重构策略研究 [J]. 计算机应用研究,2010,27(8):2987-2990.

[19] 钟林辉,郑鑫,郑燚,等. 演化信息驱动的软件体系结构重构策略研究 [J]. 计算机工程与应用,2009,45(14):66-69.

[20] 钟林辉. 构件化软件开发中演化信息的获取和应用技术研究 [D]. 北京:北京大学,2007.

[21] Tzerpos V,Holt R C. MoJo: a distance metric for software clusterings [EB/OL]. [2015-01-15]. http://www. computer. org/csdl/proceedings/wcre/1999/0303/00/03030187-abs. html.

[22] Abreu F B e,Goulao M. Coupling and cohesion as modularization drivers: are we being over-persuaded? [EB/OL]. [2015-01-17]. http://www. computer. org/csdl/proceedings/csmr/2001/1028/00/10280047-abs. html

The Research on Java Software Clustering Based on Software Evolution Dependency

ZHONG Linhui,LI Junjie,ZHANG Nengwei,HUANG Xiaoming

(School of Computer Information and Engineering,JiangXi Normal University,Jiangxi 330022,China)

Abstract: Software clustering can be used to solve the software partitioning problem and realize the software modularization. However,traditional software clustering methods have not considered the potential relation between software elements,which cannot be identified by using the static analysis method. So it may lead to software not meet the "high cohesion,low coupling" feature after clustering. In order to solve the above problem,a strategy by introducing the software evolution information into the software clustering process,and propose a software clustering algorithm based on the extended Java dependence model and simulated annealing idea have been proposed. Experiments show that this method can improve the accuracy of software clustering.

Key words: software clustering; software evolution information; high cohesion; low coupling

(责任编辑:冉小晓)