文章编号:1674-2974(2021)10-0131-06 DOI:10.16339/j.cnki.hdxbzkb.2021.10.015

基于鲸鱼优化算法的类图重构研究

胡志刚[1],杨娜[1],刘伟[1,2†]
(1. 中南大学 计算机学院,湖南 长沙 410083;2. 湖南中医药大学 信息科学与工程学院,湖南 长沙 410208)

摘 要:为了提高软件质量,组合使用重构技术、软件度量和元启发式搜索可以有效改进软件的结构而不影响其功能.本文提出一种基于鲸鱼优化算法的类图重构方法,并结合耦合、继承、抽象 3 个指标所构建的质量模型来指导类图重构序列寻优.在 6 个不同的开源程序上使用鲸鱼优化方法对类图进行重构,研究结果表明:基于鲸鱼优化算法的类图重构在质量增益上优于模拟退火算法和爬山算法,可有效提高重构质量.

关键词:软件质量;软件重构;鲸鱼优化;质量度量;类图
中图分类号:TP311 **文献标志码**:A

Research on Class Diagram Refactoring Based on Whale Optimization Algorithm

HU Zhigang[1], YANG Na[1], LIU Wei[1,2†]
(1. School of Computer Science and Engineering, Central South University, Changsha 410083, China;
2. College of Computer Science and Electronic Engineering, Hunan University of Chinese Medicine, Changsha 410208, China)

Abstract: In order to improve software quality, the combination of refactoring techniques, software metrics, and meta-heuristic search can effectively improve the structure of software without affecting its function. In this paper, a class diagram refactoring method based on Whale Optimization Algorithm is proposed, and the quality model constructed by index coupling, inheritance and abstraction is empolyed to guide the search for the optimal refactoring sequence. Cetacean optimization method is used to refactor the class diagram in six different open source programs. The results show that the class diagram refactor the based on cetacean optimization algorithm is superior to Simulated Annealing Algorithm and Hill Climbing Algorithm in terms of quality gain, and can effectively improve the quality refactoring.

Key words: software quality; software refactoring; whale optimization algorithm; quality measures; class diagram

软件产品经常随着需求的变更演变为完成不同的功能,这些演变可能使软件设计变得更加复杂,致使软件质量降低.重构被定义为优化现有系统的内部结构而不改变其功能,主要通过改进抽象、耦合、继承等质量属性来提高软件质量.重构最初是在面向对象软件的环境中提出的[1],软件重构已经被应用

* **收稿日期**:2020-11-04
基金项目:国家自然科学基金资助项目(61572525),National Natural Science Foundation of China(61572525)
作者简介:胡志刚(1963—),男,山西孝义人,中南大学教授,博士生导师
† 通信联系人,E-mail:weiliu@csu.edu.cn

于面向方面的软件、软件产品线等不同的环境,以及应用在代码、模型、文档等不同的层次中.

软件重构阶段可以分为代码层重构和模型层重构,模型层的类图重构通过找到最佳重构序列来提高软件质量,而寻找最佳重构是一个优化问题,可通过进化算法来实现,进化算法包括爬山算法[2]、模拟退火算法、群体智能算法和生物地理算法等. 群体智能算法主要是模拟动物的群体行为,依靠相互合作来捕获食物,通过自身学习及向他人学习来不断改变自身的搜索方向来提高捕食的效率. 群体智能优化算法的优势在于利用群体能力进行协同搜索,从而在解空间内找到最优解. 其中鲸鱼优化算法是根据鲸鱼围捕猎物的行为而提出的算法.

很多人都对软件重构进行了相关研究. Kebir 等人[3]提出了一种基于遗传算法的软件构件自动重构方法,首先检测出代码坏味道,然后从基于组件的源代码中构建源代码模型,最后用遗传算法搜索重构的最佳序列来减少源代码模型中存在的代码坏味道. Mohan 等人[4]介绍了许多优化算法用于自动化重构,实现了一个 Java 重构工具 MultiRefactor,该工具着眼于软件质量、代码优先级及重构覆盖率. Khatchadourian 等人[5]提出了一种自动重构方法,并设计了一个 Eclipse IDE 的插件,帮助开发人员以一种保持语义的方式编写高效的流代码. 该方法基于一种新的数据排序和类型状态分析,由前置条件和转换组成,用于自动确定将顺序流转换为并行流、将已经并行的流进行无序或去并行化. 与这些针对局部代码的重构相比,本文侧重于全局的类图重构及模型优化.

软件质量模型被广泛应用于软件重构以识别重构对象、引导重构策略的选择以及评估重构后软件质量的变化. 本研究重点考虑抽象、耦合和继承属性,它们在软件质量中占很大因素,对于单个质量指标可以表示软件度量的不同方面. 继承可以指导类是否是正确关联和扩展的[6],以及类的层级结构,该度量结合了实现接口和抽象类的使用. 耦合可用来指导软件系统中对象之间相互依赖的程度[7],期望尽可能的低耦合. 抽象是指导系统的稳定,系统的变化是体现在具体类上的,使用抽象可以提升设计的简单性,改善软件开发的质量[8]. 因此我们采用继承、耦合和抽象来构建质量模型.

鲸鱼优化算法搜索能力强、结构简单、参数少并且易于实现[9],所以选择鲸鱼优化算法寻找最优的类图重构序列来进行类图重构,最后通过构建质量模型来评价类图重构的质量.

1 质量模型构建

重点选取耦合、继承、抽象这三个质量属性. 表 1、表 2 分别列出了每个属性的简要描述以及针对每个度量的指标描述,表 3 中给出了每个属性的计算方法,即三个适应度函数,度量耦合采用类相互依赖的数量、不同类间的相互引用数量两者权重平均分配. 类间的相互引用表示为类中属性使用了另一个类或接口的数量、其他类将本类作为属性的数量、类中方法参数使用其他类或接口的数量、本类作为其他类或接口中方法参数的数量. 实现接口数量、每个类的被继承数量、子类和父类的数量四个度量指标共同决定继承性,这四个度量指标的权重平均分配. 抽象体现在类和接口的比例和数量上,抽象类和类的比例、包中接口数及类中接口数三者占同等比例. 权重规范化为 1,按度量标准和权重根据经验值、实验及软件因素进行相应设定. 对不同的属性进行归一化处理,通过采用加权聚合方法来对不同的度量因子进行归一化,$Q_g = \sum_{i=1}^{i=3} w_i*(q_i^t - q_i)$,其中 w_i 为权重,q_i^t 为重构后的质量属性值,q_i 为重构前的质量属性值,通过比较质量增益来看重构后的质量变化.

表 1 度量属性的简要描述

Tab.1 A brief description of each attribute

度量属性	描述
抽象	对软件系统进行扩展和构建的难易程度. 基于当前抽象类的数量和当前实现接口的数量进行评估
耦合	根据其他类对类、属性和参数的使用数量来度量类之间的依赖关系
继承	根据接口实现的数量以及子类和父类的数量来衡量项目的类结构

为了简化 UML 类图在 RAM 中的表示,以便用鲸鱼优化算法处理类图,用抽象数据结构 UML 映射复杂度更低一些. 考虑 15 种常规的类图重构重命名类、用委托关系代替继承关系、用继承关系代替委托关系、提取到子类、提取到父类、折叠继承关系、内联类、提取类、函数下移、方法提到基类中、方法重命名、字段下移、字段上移、字段重命名、封装字段. 通过一系列的转化来找到最优的重构序列,然而在一些场景中,如果多个类都继承自某一父类或者父接

口,会具有接口上的一致性. 但是如果将其中的某些类单独提取出来，可能会破坏这种一致性. 因此,为了提高类图的整体质量，有时候需要综合考虑多个因素,而不仅仅只是考虑接口的一致性,进而获得更好的解决方案.

表 2　度量指标的描述

Tab.2　Software metrics used in experiment

度量指标	描述
abstractness	包中抽象类与类的比率
numInterf	包中的接口数量
iFImpl	类中实现接口的数量
iC_Attr	一个类中使用另一个类或接口作为该类属性的数量
eC_Attr	一个类作为属性被外部使用的数量
iC_Par	使用另一个类或接口作为该类方法中的参数数量
eC_Par	一个类作为方法中的参数被外部使用的数量
Dep_In	依赖于类的元素数量
Dep_Out	一个类所依赖的元素的数量
iFImpl	一个类实现接口的数量
NOC	NOC 是类的直接子类的数目
numDesc	类的子类数量
numAnc	类的父类数量

表 3　度量属性的计算方法

Tab.3　Metric details for each fitness function

属性	度量组件和权重
抽象	0.33*abstractness+0.33*numInterf+0.33*iFImpl
耦合	−0.125*iC_Attr−0.125*eC_Attr−0.125*iC_Par−0.125*eC_Par −0.25*Dep_In−0.25*Dep_Ou
继承	0.25*iFImpl + 0.25*NOC + 0.25*numDesc + 0.25*numAnc

2　基于鲸鱼优化算法的类图重构

鲸鱼优化算法是一种元启发式优化算法，该算法模拟了座头鲸的捕猎行为，座头鲸喜欢捕食一群靠近水面的小鱼和小虾，遇到猎物后会先向下俯冲大约 12 m,然后开始在猎物周围制造螺旋形气泡,最后游向水面捕食猎物[10]. 这个过程提取出三个数学模型即围捕猎物、螺旋气泡网捕食和寻找猎物. 用鲸鱼算法解决类图优化的基本思想如图 1 所示.

根据类图可能的重构序列及质量模型作为优化的参数，鲸鱼群中的每一个个体的位置均包含一组重构序列. 利用鲸鱼寻找猎物的方式不断更新类图的重构序列,直到找到最好的位置,即找到最佳重构序列. 针对质量度量,首先采用耦合、继承和抽象这三个适应度函数来进行指导，并获得鲸鱼优化算法针对单个度量的改进程度.

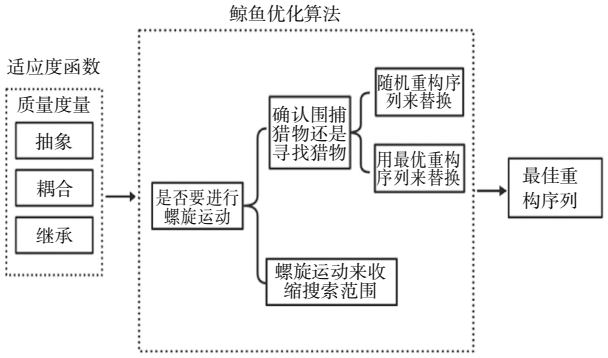


图 1　类图优化架构图

Fig.1　Class diagram optimization basic idea

2.1　提取重构序列

假设当前最优重构序列是目标序列或接近最优目标序列. 定义了最优搜索代理后,其他搜索代理将试图更新它们的位置,以寻找最优,此行为可以表示为公式(1)和公式(2):

$$\vec{D} = \left|\vec{C}\cdot\vec{X}^{*}(t) - \vec{X}(t)\right| \tag{1}$$

$$\vec{X}(t+1) = \vec{X}^{*}(t) - \vec{A}\cdot\vec{D} \tag{2}$$

式中:t 代表迭代次数，$\vec{A}$ 和 $\vec{C}$ 为系数向量,X^{*} 为目前所得的最优重构序列解，$\vec{X}$ 为位置向量. $\vec{A}$ 和 $\vec{C}$ 的公式可表示为(3)和(4):

$$\vec{A} = 2\vec{a}\cdot\vec{r} \tag{3}$$

$$\vec{C} = 2\cdot\vec{r} \tag{4}$$

$\vec{a}$ 在迭代过程中从 2 减少到 0,$\vec{r}$ 为[0,1]的随机向量.

2.2　螺旋气泡网寻找

2.2.1　收缩包围机制

如图 2 所示，求解最佳重构序列的性能也会受到 $\vec{A}$ 的影响,通过减少 $\vec{a}$ 的值来影响 $\vec{A}$ 实现收缩最佳重构序列的范围，算法的收敛性是由参数 a 来决定的,由于 a 从 2 递减为 0,在算法初期,迭代的递减速率相对较慢,从而可以有效扩大算法的搜索空间,在算法后期,迭代的递减速率较快,有利于加快算法收敛,提高其收敛性.

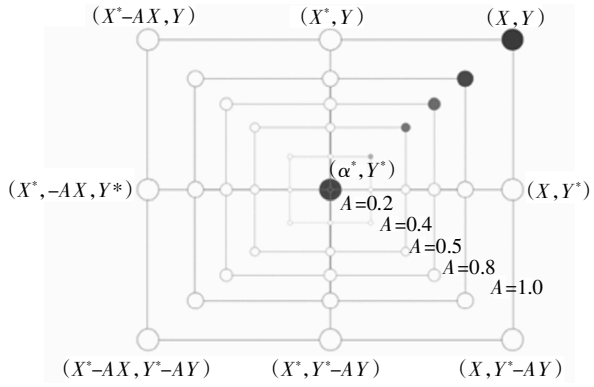
2.2.2　螺旋更新位置

如图 2 所示计算出位于(X,Y)处的鲸鱼和(X^{*},Y^{*})的猎物之间的距离,然后在鲸鱼和猎物之间建立

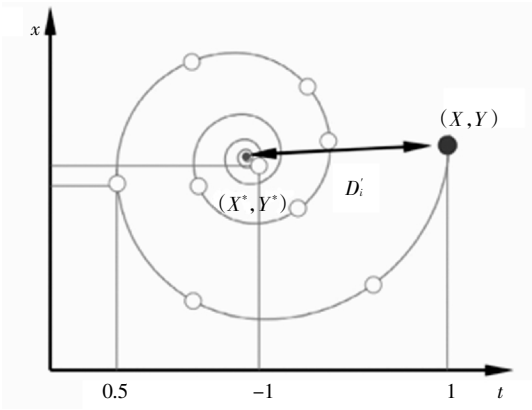
一个螺旋方程来模拟座头鲸的螺旋运动，螺旋方程如公式(5)所示：

$$\vec{X}(t+1)=\vec{D'}\cdot e^{bl}\cdot\cos(2\pi l)+\vec{X}^*(t) \tag{5}$$

式中：$\vec{D'}=\left|\vec{X}^*(t)-\vec{X}(t)\right|$，$b$ 为对数螺旋的一个常量，l 为[-1,1]中的随机数.



(a)收缩包围机制



(b)螺旋更新位置

图 2 螺旋气泡网搜索机制

Fig.2 Bubble-net search mechanism

座头鲸会在缩小包围圈的同时沿一个螺旋形路径不断的逼近猎物. 为了模拟这种同时发生的行为，假设在收缩包围机制和螺旋更新位置之间有 50%的可能性进行选择，以便在优化过程中更新重构序列，此行为表示为公式(6)：

$$\vec{X}(t+1)=\begin{cases}\vec{X}^*(t)-\vec{A}\cdot\vec{D}, & p<0.5\\ \vec{D'}\cdot e^{bl}\cdot\cos(2\pi l)+\vec{X}^*(t), & p\geqslant 0.5\end{cases} \tag{6}$$

式中：p 为[0,1]中的随机数.

2.3 搜索最优重构序列

$\vec{A}$ 的变化被用来寻找最优重构序列，座头鲸会根据彼此的位置随机搜索. 因此，使用 $\vec{A}$ 取随机值(>1,<-1）来强迫搜索代理远离参考鲸. 和开发阶段不同的是，在探索阶段随机选择搜索代理来更新搜索代理的位置，而开发阶段是选取最优搜索代理.

$\left|\vec{A}\right|$ >1 时，允许鲸鱼优化算法执行全局搜索，数学模型如公式(7)(8)所示：

$$\vec{D}=\left|\vec{C}\cdot\overrightarrow{X_{\text{rand}}}-\vec{X}\right| \tag{7}$$

$$\vec{X}(t+1)=\overrightarrow{X_{\text{rand}}}-\vec{A}\cdot\vec{D} \tag{8}$$

式中：$\overrightarrow{X_{\text{rand}}}$代表随机位置向量.

鲸鱼优化算法从一组随机解开始，在迭代过程中，搜索会更新它们的位置(要么随机代理，要么最优代理). 将参数 a 从 2 降到 0，分别用于开发和探索. $\left|\vec{A}\right|$来决定是选择一个随机搜索代理还是选择最优代理来更新搜索代理的位置. 根据 p 的值，鲸鱼优化算法可以同时满足螺旋更新位置和收缩包围机制. 算法 1 为鲸鱼优化算法(Whale Optimization Algorithm，WOA)的伪代码.

算法 1 鲸鱼优化算法 WOA

输入：初始化种群 $X_i(i=1,2,\cdots,n)$，每个搜索代理的适应度函数

输出：X^*

1. while (t < maximum number of iterations)
2. for each search agent
3. Update a, A, C, l, and p
4. if ($p<0.5$) //收缩包围机制
5. if ($|A|<1$) //选择最优搜索代理
6. 根据式(1)选择最优搜索代理
7. else if ($|A|\geqslant 1$)// 选择随机搜索代理
8. Select a random search agent (X_{rand})
9. 根据式(8)来更新搜索代理
10. else if($p\geqslant 0.5$)//螺旋更新位置
11. 根据式(5)来更新搜索代理
12. end if
13. end for
14. 计算每个搜索代理的适应度函数
15. 更新 X^*
16. $t=t+1$
17. end while

鲸鱼优化算法从理论上可以被认为是一个全局优化，因为它包含了探索和开发能力，并且只需要调整两个主要内部参数(A 和 C). 此外，基于群体的算法在随后的迭代中会保留搜索空间的信息，而基于进化的算法在新种群形成时就丢弃了所有信息；基于群体的算法与进化算法相比有较少的操作符更易实现.

3　实验环境与结果分析

3.1　数据来源

将鲸鱼优化算法用于类图的优化,将抽象、耦合和继承这三个适应度函数来进行衡量优化的结果. 实验的输入程序包括 6 个开源 Java 项目:JSON,一个用于数据交换格式的 Java 库; 词法分析器生成器 JFlex;基于 xml 的远程过程调用库 Apache-XmlRpc; Mango 及解析器生成器和基于 GUI 框架开发的图形编辑器 JHotDraw. 上述 6 个开源项目针对不同的软件结构,有关实验的 Java 项目的详细信息见表 4. 实验的总运行次数为 10*3(搜索)*3(函数)*6(基准),总共运行 540 次. 实验在一台 3.40GHz Intel Core i7-3770 处理器和 8gb RAM 的 PC 上进行.

表 4　实验的 Java 项目

Tab.4　Java programs used in experiment

项目名称	类数量	代码行数
JSON 1.1	12	2 196
JFlex 1.4.1	56	15 094
Apache-XmlRpc 3.1.1	100	6 532
Mango	78	3 470
Beaver 0.9.8	81	7 851
JHotDraw 5.3	241	27 824

3.2　实验结果分析

图 3 展示了运用鲸鱼优化算法的结果,显示了 6 个基准程序中每个适应度函数的平均质量增益（通过最终的总体度量分数减去初始分数,10 次运行的平均值).

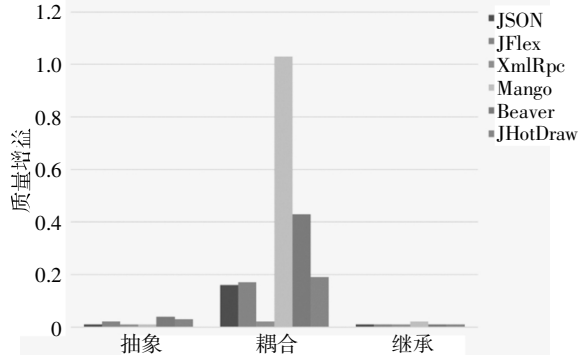


图 3　每个适应度函数的平均质量增益

Fig.3　Average quality gain of each fitness function

在三个适应度函数中，耦合是唯一一个显示出显著改善的因子. 抽象的改进很小,继承完全没有变化. 事实上,继承函数唯一有变化的情况是在模拟退火中. 对于耦合函数在改进方面更有效,在抽象和继承函数中改进很小表明用于组合这些函数的度量缺乏波动性. 在 Mango 项目中耦合函数的增益明显高于其他，这说明 Mango 项目耦合程度较高，易于改进. XmlRpc 在耦合方面改进很小，可能是因为它类之间的耦合很小.

图 4 展示了使用三种不同优化算法得出的 6 个程序的平均质量增益.

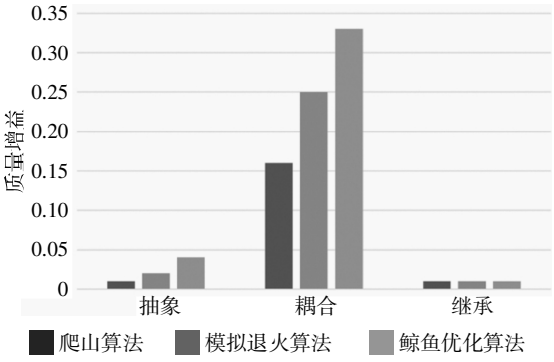


图 4　不同优化算法的总体平均质量增益

Fig.4　Average quality gain by different optimization algorithms

结果表明鲸鱼优化算法相对质量提升更优,但也表明,模拟退火算法优于爬山算法,这是因为爬山算法更容易陷入局部最优，提前找到的最优重构序列并不是全局最优，鲸鱼优化算法相比较而言不容易陷入局部最优. 同样的抽象和继承缺乏应用的重构操作，这些函数的质量增益差的原因是由于缺少可用的操作,而其他指标更不稳定,并且有更多的重构操作可用来改进他们.

为了进一步验证鲸鱼优化算法的求解性能,将鲸鱼优化算法与爬山算法和模拟退火算法这三种不同的优化算法进行了 60 次求解. 图 5 表明了鲸鱼优化算法用于类图重构的收敛性更好，因为爬山算法是选择邻近点,容易陷入局部最优,达到快速收敛,模拟退火算法的特点是随机选择，相比较而言没那么容易陷入局部最优,收敛速度相对慢一些,而鲸鱼优化收敛是取决于 a,固定住 a,来弱化算法,从而减弱其收敛性,实验结果表明,鲸鱼优化算法的全局优化能力最强,且算法相对稳定,而模拟退火算法优化能力和稳定性居中,且优于爬山算法,所以鲸鱼优化算法在收敛性和搜索空间上都要更优一些. 对函数的初始和最终度量分数进行统计分析，使用具有 95%置信水平的 Wilcoxon 符号秩检验，所获得的结

果在比较函数的每次运行时具有统计学意义.

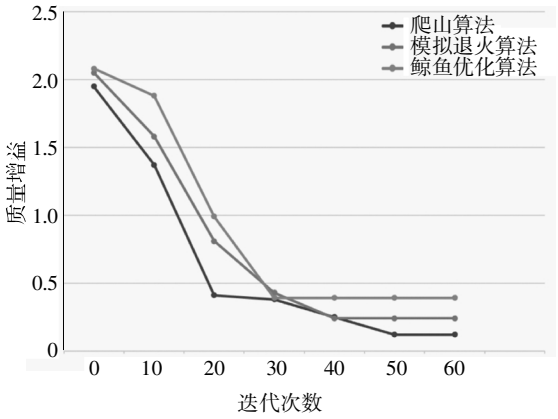


图 5 三种算法的收敛性比较

Fig.5 Convergence comparison of the three algorithms

4 总 结

近年来,有很多关于软件重构的研究,但现有的重构研究多侧重于在代码编写阶段为开发人员提供局部代码的重构及代码推荐, 很少考虑在分析设计阶段为开发设计人员提供全局性的重构. 本文采用鲸鱼优化算法来调整类成员与类之间的映射关系,得到一组最优重构序列. 同时,实验对比证明了本文提出的鲸鱼优化算法可以有效应用于软件重构进而提高软件质量. 后续工作将对稳定性、可靠性等属性做进一步研究.

参考文献

[1] BAQAIS A A B, ALSHAYEB M. Automatic software refactoring: a systematic literature review [J]. Software Quality Journal, 2020, 28(2): 459—502.

[2] 张英杰,郭会芳,付海滨,等. 面向多模态函数的自适应混沌爬山微粒群算法[J]. 湖南大学学报(自然科学版),2013,40(2):77—81.
ZHANG Y J, GUO H F, FU H B, *et al*. An adaptive chaotic hill-climbing particle swarm optimization algorithm for multimodal functions[J]. Journal of Hunan University (Natural Sciences), 2013, 40(2): 77—81. (In Chinese)

[3] KEBIR S, BORNE I, MESLATI D. A genetic algorithm-based approach for automated refactoring of component-based software[J]. Information and Software Technology, 2017, 88: 17—36.

[4] MOHAN M, GREER D. Using a many-objective approach to investigate automated refactoring[J]. Information and Software Technology, 2019, 112: 83—101.

[5] KHATCHADOURIAN R, TANG Y M, BAGHERZADEH M, *et al*. Safe automated refactoring for intelligent parallelization of Java 8 streams [C]//2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). Montreal, QC, Canada. IEEE: 2019: 619—630.

[6] SAVIC M, IVANOVIC M, RADOVANOVIC M. Analysis of high structural class coupling in object-oriented software systems[J]. Computing, 2017, 99(11): 1055—1079.

[7] BASHIR R S, LEE S P, KHAN S U R, *et al*. UML models consistency management: Guidelines for software quality manager [J]. International Journal of Information Management, 2016, 36(6): 883—899.

[8] GHANNEM A, KESSENTINI M, HAMDI M S, *et al*. Model refactoring by example: A multi-objective search based software engineering approach [J]. Journal of Software: Evolution and Process, 2018, 30(4): e1916.

[9] CHAKRABORTY A, KAR A K. Swarm intelligence: A review of algorithms[M]//Nature-Inspired Computing and Optimization. Cham: Springer International Publishing, 2017: 475—494.

[10] NASIRI J, KHIYABANI F M. A whale optimization algorithm (WOA) approach for clustering [J]. Cogent Mathematics & Statistics, 2018, 5(1): 1483565.