计算机科学
COMPUTER SCIENCE

**自动化软件重构质量目标与非质量目标有效性研究**

郭亚琳，李晓晨，任志磊，江贺

引用本文

郭亚琳，李晓晨，任志磊，江贺. 自动化软件重构质量目标与非质量目标有效性研究[J]. 计算机科学, 2022, 49(11): 55-64.

GUO Ya-lin, LI Xiao-chen, REN Zhi-lei, JIANG He. Study on Effectiveness of Quality Objectives and Non-quality Objectives for Automated Software Refactoring[J]. Computer Science, 2022, 49(11): 55-64.

---

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220300058

# 自动化软件重构质量目标与非质量目标有效性研究

郭亚琳　李晓晨　任志磊　江　贺
大连理工大学软件学院　辽宁 大连　116600
(1628713274@mail.dlut.edu.cn)

**摘　要**　随着软件不断迭代发展,软件维护成本也相应增加。自动化重构可以降低软件维护成本,基于搜索的重构方法是解决该问题最典型的方法之一。其中目标的选择对搜索过程起决定性作用,质量目标与非质量目标都是开发人员在重构时通常会考虑的目标。然而,尚未有研究系统地分析在相同的评价环境下,哪些目标更有利于代码重构,特别是得到符合开发者预期的代码重构结果;并且也未分析质量目标与常用的非质量目标进行组合是否会有更好的效果。文中提出了基于搜索的多目标软件重构方法,探索了7个不同目标的组合对软件重构质量的影响。在6个规模不同的开源软件项目上进行了验证,应用多种指标对重构前后软件质量进行评估,并分析了不同优化目标组合的表现。实验结果表明,质量目标与非质量目标组合比单独使用质量目标组合对重构效果的提升更明显,其中质量目标与之前重构记录的一致性的组合对重构有较好的提升效果。

**关键词**:软件重构;多目标优化;软件质量;基于搜索的软件工程;软件维护
**中图法分类号**　TP311

## Study on Effectiveness of Quality Objectives and Non-quality Objectives for Automated Software Refactoring

GUO Ya-lin,LI Xiao-chen,REN Zhi-lei and JIANG He
School of Software,Dalian University of Technology,Dalian,Liaoning 116600,China

**Abstract**　The cost of software maintenance increases as the continuous iterative development of software. To reduce this cost, automated software refactoring is proven to be an effective solution. One of the most typical automated software refactoring approaches is the search-based refactoring. This approach refactors software according to some predefined objectives. Existing studies show that the selection of objectives plays a decisive role in the search process. Although many quality objectives and non-quality objectives are proposed,there is no research to analyze that in the same evaluation framework which objectives can guide the software refactoring to achieve results that meet developers' expectations. Meanwhile,whether the combination of quality objectives and non-quality objectives will have better results is not analyzed. To answer these questions,this paper designs a search-based multi-objective software refactoring evaluation framework. Under the framework,this paper investigates the impact of the combination of seven different objectives on software refactoring with six open-source software projects in different scales;the software quality before and after refactoring is evaluated using a variety of indicators,and the effectiveness of different objective combinations is also analyzed. Experimental results show that the combination of quality objectives and non-quality objectives can better improve the effectiveness of software refactoring than using one single type of objectives,and the combination of the 'quality' objective and the 'similarity with recorded code changes' objective is important to the improvement of software refactoring effectiveness.

**Keywords**　Software refactoring, Multi-objective optimization, Software quality, Search-based software engineering, Software maintenance

## 1　引言

随着软件项目规模的增大,系统维护成本与难度也相应增加。有研究报告[1-3]指出,软件的维护成本占软件项目总成本的80%以上,而且软件开发人员通常会花60%的时间去理解正在维护的代码。因此,重构被提出,用于改进软件的内部结构,从而降低软件的复杂性和维护成本[4]。

然而,对于大型复杂软件系统而言,采用人工的方式对其进行重构难度过高,并且很难保证重构解决方案具有通用性。因此,目前很多工作[5-7]致力于研究软件的自动重构,其中基于搜索的重构方法(Search-Based Refactoring,SBR)是解决该问题最典型的方法之一,其将基于搜索的软件工程(Search-Based Software Engineering,SBSE)[8]应用于重构问题,将软件重构问题视为一个搜索优化问题,利用基于搜索的方法,在问题解空间中,根据特定的目标进行代码重构,得到问题的最优解,即得到对应的最佳重构解决方案,以提升代码质量。

到稿日期:2022-03-07　返修日期:2022-04-30
通信作者:李晓晨(xiaochen.li@dlut.edu.cn)

已有的基于搜索的软件重构工作包括单目标优化与多目标优化两类方法。单目标优化方法只能对某一个具体目标进行优化，而多目标优化方法可以对多个不同的目标进行优化，多目标可以更好地对有冲突的目标进行权衡。

研究表明，目标的选择会极大地影响重构代码的质量。现有文献[7,9-11]中提出的目标可以分为两类：质量目标（如软件的可重用性、可理解性、有效性和灵活性等）和非质量目标（如代码异味纠正率、重构操作成本、与之前重构记录的相似性等）。在已有研究中已经提出了十余个不同的目标/目标组合。在不同的研究中，研究者通常提出自己的新目标，并采用自定义的评价方式进行各自算法的评价。

但目前尚没有研究系统地分析在相同的评价环境下，这些目标中哪些更有利于代码重构，特别是有助于得到符合开发者预期的代码重构结果。同时，大部分研究通常只单独关注质量目标或非质量目标，并没有分析现有两类目标的组合对代码重构的意义。

因此，本文首次系统地分析不同类型重构目标对生成满足开发者预期重构结果的作用，以及不同类型重构目标的多目标组合是否有更好的效果。

为此，本文设计了多目标软件重构评估框架，该框架使用非支配排序遗传算法（Non-dominated Sorting Genetic Algorithm II，NSGA-II）对质量目标与非质量目标的组合进行研究，将重构问题视为一个组合优化问题，其中搜索空间中的解表示重构解决方案。通过计算适应度值来引导重构选择策略，从而找到最优重构解决方案。结合此框架，本文选择了之前文献中常用的 7 个目标，包括 4 个质量属性目标（软件的可重用性、可理解性、有效性和灵活性），以及 3 个非质量目标（代码异味纠正率、重构操作成本及与之前重构记录的相似性），以分析质量目标在实际重构中的表现、非质量目标及其组合在实际重构中的表现，以及非质量目标及其组合对质量目标的影响 3 个研究问题，并利用重构准确率、重构召回率指标考察每种重构方式是否能够得到符合开发者预期的重构序列。

实验分析结果表明，质量属性目标中可理解性目标表现更好。在非质量组合及其目标中，单独使用与之前重构记录的一致性以及代码异味目标和与之前重构记录的一致性的组合对重构的提升效果较好。此外，大部分的非质量目标对质量目标有提升效果。

本文的主要贡献如下：

(1)建立评估框架，首次系统地分析了不同类型重构目标对生成满足开发者预期的重构结果的作用。

(2)结合评估框架进行了大量实验，探索了已有文献中常用的 7 种不同类型优化目标及其组合对重构的影响。

(3)首次探索了质量目标与非质量目标的组合对软件代码重构的作用。

本文第 2 节讨论研究背景和相关工作；第 3 节介绍多目标软件重构评估框架；第 4 节描述实验设计以及研究问题；第 5 节分析并讨论实验结果；第 6 节讨论实验方法有效性的威胁；最后总结全文。

## 2 研究背景及相关工作

本节首先介绍软件重构的背景，包括重构的定义以及步骤，然后总结软件重构的相关工作，包括现有的手动、自动重构方法，并讨论使用基于搜索方法解决软件重构存在的问题。

### 2.1 软件重构背景

为了降低系统复杂性，持续优化已有代码是维护系统生命力的最好方法，重构让代码重新焕发活力。重构被定义为在代码编写完成后，通过改变其内部结构而不改变其外部行为来改进代码的过程[4]。

重构过程包括 6 个主要步骤[12]：1)识别需要重构的代码对象；2)确定重构方法，将特定重构操作应用于需要重构的位置；3)确保应用的重构操作不会破坏软件行为；4)执行重构，对需要重构的代码对象执行选定的重构操作；5)评估重构，通过软件度量评估重构的影响；6)维护重构工件和其他软件构件间的一致性。

软件重构有以下两方面的作用：1)改进软件设计；2)提高软件效率。

目前很多 IDE 工具针对简单的重构操作为开发人员提供了半自动化支持，如类的重命名等重构操作，但对于自动化推荐重构操作序列依旧充满挑战。下面对重构方法的相关工作进行介绍。

### 2.2 软件重构相关工作

对于手动重构，Fowler 使用了手工重构的方式[13]，他提出 22 种代码异味，并分别给出了相应的重构策略。Mens 等提出通过手动重构的方式去修复不同缺陷[12]，但是手动重构很难保证解决方案对纠正代码异味具有通用性，相同类型的代码异味可能有不同的纠正方案。

由于手动重构耗时，在过去的几十年里，人们对自动化软件重构进行了大量研究，其中基于搜索的重构方法是软件重构中使用最广泛的方法之一。基于搜索的软件重构方法将基于搜索的软件工程应用于软件重构问题中，通过搜索的方法找到可能的重构解决方案。

之前的大多数工作将基于搜索的软件重构定义为单目标优化问题。Seng 等使用演化算法对类的结构进行优化，将软件重构视为一种单目标优化问题[14]。Kessentini 等提出了单目标组合优化[15]，使用遗传算法找到重构操作的最佳序列，通过尽可能减少在源代码中检测到的设计缺陷的数量来提高代码的质量。

在之后的工作中，开发人员在重构时并不只是考虑单个因素，通常会考虑多个因素，因此基于多目标优化的软件重构方法受到了开发人员的广泛关注。Praditwong 等第一次将软件重构问题视为多目标优化问题进行考虑[16]。Ouni 等使用 NSGA-II 算法，通过减少代码异味数量以及减少代码更改量等指标来提高软件质量[10]。Harman 等提出了一种基于搜索的方法，使用遗传算法来改进子系统的结构，将两个质量度量的组合定义为适应度函数[17]。Bavota 等使用交互式遗传算法进行软件重构，适应度由自动评估因素与人工评价因素组成[18]。Ouni 等使用软件不同版本的历史变更信息对相似的上下文中的代码进行重构，使用 NSGA-II 算法在最小化代码异味目标与最大化利用开发历史等目标之间找到最佳的折中方案[19]。Mkaouer 等提出了一种改进的基于 NSGA-III 遗传算法的多目标优化方法，实现了对大量质量指标进行同时

优化[5]。O'Keeffe 等在 QMOOD[20] 度量的指导下，使用基于局部搜索的算法，如爬山算法和模拟退火算法，实现软件自动重构[21]。Mohan 等使用了软件质量、代码优先级、重构覆盖率和元素近期性的度量这 4 个目标进行软件重构[22]。Fernandes[23] 等对不同的质量目标进行了重构分析，包括内聚性、耦合性等属性。Abid 等使用基于搜索的算法在安全性目标以及质量目标之间做了一个平衡[24]。Mariani 等对基于搜索的软件重构研究进行了调研[9]，结果显示研究中使用最广泛的目标包括最小化代码异味数量、最小化代码修改数量、最大化与之前重构记录的一致性[25] 以及 QMOOD 度量等目标。表 1 列出了在相关文献中常用的重构目标。

表 1 已有文献中使用的优化目标

Table 1 Objectives used in related literatures

| | Objective | References |
|---|---|---|
| Quality-objectives | QMOOD | [14,20-22,24,26-27] |
| | OO metrics | [5,17] |
| | Cohesion | [16,23] |
| | Coupling | [16,23] |
| Non-Quality-objectives | Number of-code smells | [6,10,15,19,25-26] |
| | Refactoring efforts | [6,10,26-28] |
| | Similarity with- Recorded- Code Changes | [6,19,25,28-29] |
| | User feedback | [11,18,29] |
| | Security-critical code | [24,30] |
| | Priority objective | [22] |
| | Refactoring coverage | [22] |
| | Element recentness | [22] |

在已有研究中，目前已经提出十余种不同的目标/目标组合，研究者提出的目标可以分为两类：质量目标（QMOOD、内聚性、耦合性等，其中 QMOOD 包括可重用性、可理解性、可扩展性、功能性、有效性等 6 个质量属性目标）、非质量目标（包括代码异味纠正率、重构操作成本等）。

大多数现有的多目标重构技术[9] 都是通过利用基于搜索的方法优化不同的目标，进而提出一套非主导的重构解决方案作为输出，以便软件开发人员可以从一组可能的重构解决方案中选出最佳的解决方案。研究表明，目标的选择会极大地影响重构代码的质量。

在不同的研究中，研究者通常提出自己的新目标，并采用自定义的评价方式进行各自算法的评价，但目前尚没有研究系统地分析在相同的评价环境下，这些目标中哪些更有利于代码重构，特别是有助于得到符合开发者预期的代码重构结果；同时，大部分研究通常只单独关注质量目标或非质量目标，并没有分析现有两类目标的组合对代码重构的意义。因此，本文首次系统地分析不同类型重构目标对生成满足开发者预期的重构结果的作用，以及不同类型重构目标的多目标组合是否有更好的效果。

## 3 多目标软件重构评估框架

本节首先对本文建立的评估框架进行概述，其次详细介绍了框架中软件重构方法相应的组件。

### 3.1 框架概述

本文建立的评估框架使用 NSGA-II 算法，并将重构问题视为一个组合优化问题，其中搜索空间中的解表示重构解决方案。软件重构的主要任务是找到最有效的重构序列。基于搜索的软件重构方法通过计算适应度值来引导重构选择策略，从而找到最优重构解决方案。从整体上来看，因为已经确定了需要重构的代码片段，所以软件重构被视为一个包括实施重构与评估重构的迭代过程。框架的总体结构如图 1 所示，首先，框架需要重构的软件组件（类或方法）的集合作为输入。其次，软件重构方法可以选择性地接收可能的重构操作、适应度函数相应的软件度量以及算法其他附加信息来指导方法的执行。这一步骤中对算法的各个方面进行定义，包括重构解决方案的映射、遗传操作算子以及适应度函数的设定，具体相关信息将在下一节中进行详细说明。最后，在算法中使用相应的适应度函数对解决方案进行动态评估，在搜索收敛至最优时，输出最佳的重构解决方案。

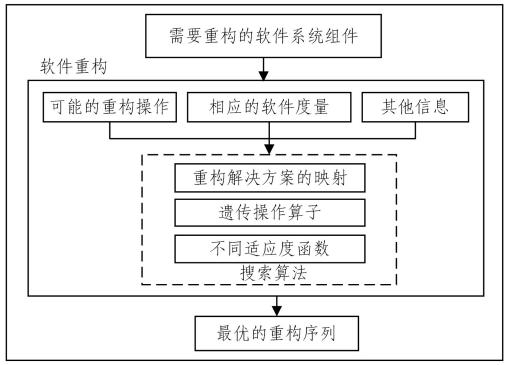



图 1 方法框架

Fig. 1 Architecture of approach

### 3.2 软件重构方法组件

本文选择 NSGA-II 算法作为本文框架中的搜索算法。NSGA-II 是目前最流行的多目标遗传算法之一，具有运行速度快、解集收敛性好的优点。

算法的第一步是随机初始化一个指定规模的种群，种群中的每个个体用特定的编码表示；之后，算法根据每个个体不同的适应度值对其进行选择、交叉以及变异等操作，对种群进行迭代进化，直到达到终止条件，并且返回终止条件时的最优解。

其中，算法将需要重构的代码片段作为输入，通过使用不同的适应度函数，引导搜索过程，当算法达到终止条件（最大迭代次数）之后，返回适应度值最高的解决方案，即最优的重构序列。

软件重构方法的关键是重构解决方案的映射、遗传操作算子的设定（包括选择、交叉以及变异等遗传算子）以及适应度函数的设定，下文将对这 3 种组件进行详细描述。

#### 3.2.1 重构解决方案的映射

在传统的遗传算法中，优化问题所生成的解被称为个体，用一个变量序列来表示。个体编码将一个问题的可行解从相应的解空间到遗传算法能处理的搜索空间进行一个转换。

在软件重构阶段，对于候选解的表示，是一组重构操作的向量，如图 2 所示，向量中每一位（又称基因）表示为一个重构操作[10]：

$$ROs=\{RO_1,RO_2,\cdots,RO_i,\cdots,RO_n\}$$

| $RO_1$ Move method | $RO_2$ Pull up field | ... | $RO_i$ Move class | ... | $RO_n$ Extract method |
|---|---|---|---|---|---|

图 2 个体的组成

Fig. 2 Individual composition

表 2 列出了本文所涉及的重构操作。本文所使用的方法涉及的 11 个重构操作[10-11]均为最常用的重构操作。重构操作(ROs)分为两种类型:高级重构(High-level Refactoring,HLR)和低级重构(Low-level Refactoring,LLR)[10]。本文所使用的 11 种重构操作均为高级重构操作。LLR 是由一个基本的重构操作(如创建类、删除方法和添加字段)组成的基本重构,HLR 是由两个或多个 LLR 构成的重构操作序列。例如,Move Method 是由 1 个 add method、1 个 delete method 以及 $n$ 个 Redirect method call 以及 $n$ 个 redirect field access 4 种 LLR 操作组成。

表 2 重构操作

Table 2 Refactoring operations

| High-level refactoring | Low-level refactoring | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | create class | delete class | add method | delete method | add field | delete field | redirect method call | redirect field access | add parameter | remove parameter | rename method | rename field | rename class |
| Move Method | | | 1 | 1 | | | $n$ | $n$ | | | | | |
| Move Field | | | | | 1 | 1 | | $n$ | | | | | |
| Pull Up Field | | | | | 1 | 1 | | $n$ | | | | $n$ | |
| Pull Up Method | | | 1 | 1 | | | $n$ | $n$ | | | $n$ | | |
| Push down Field | | | | | 1 | 1 | | $n$ | | | | $n$ | |
| Push down Method | | | 1 | 1 | | | $n$ | $n$ | | | $n$ | | |
| Inline class | | 1 | | | | | | | | | | | $n$ |
| Extract method | | | 1 | | | | $n$ | $n$ | $n$ | $n$ | $n$ | | |
| Extract class | 1 | | $n$ | $n$ | $n$ | $n$ | $n$ | $n$ | | | | | |
| Move class | 1 | 1 | | | | | $n$ | $n$ | | | | | $n$ |
| Extract interface | 1 | $n$ | $n$ | $n$ | $n$ | $n$ | $n$ | $n$ | | | | | $n$ |

### 3.2.2 遗传操作算子

NSGA-II 的操作算子包括选择、交叉和变异算子,遗传算子通过对候选解进行进化,从而使算法对搜索空间具有高效的搜索能力。

选择算子体现出优胜劣汰的原理,通过选择适应度值高的个体,去除劣质个体,从而达到优胜劣汰的效果。选择算子用来确定交叉或变异的个体,以及被选父个体将产生多少子个体。本文使用的选择策略为轮盘赌算法,在该方法中,个体的选择概率与适应度值成比例,因此个体的适应度值越大,被选中的概率越大,进而通过交叉或变异算子将其特征遗传给新个体的概率就越大。

交叉算子通过交换个体间的遗传信息来产生新的个体,从而确保基于父种群中个体后代的产生。交叉算子在已有的两个父个体的基础上进行操作,产生新的子个体。本文使用单点交叉,首先随机选择两个父种群个体,并且在某一个个体上随机选择一个交叉点,将两个父个体进行交叉重组。通过图 3 可以看出,在交叉前后,两个父个体的交叉点在第 3 个基因位,共有 3 个位置的基因信息得到交换,从而产生了新的子个体。在交叉算子的操作中,如果交换的基因信息相同,那么就不会产生新的子个体。因此,算子将引入变异算子来保证种群的多样性。

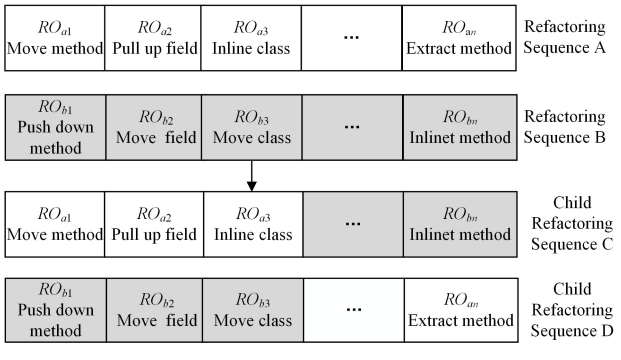



图 3 交叉算子

Fig. 3 Crossover operator

变异算子通过对个体的部分基因进行突变而得到没有被使用的遗传信息,可以防止个体在迭代生成最优解的过程中过早地收敛,从而保持种群的多样性。变异通过改变某一个重构操作来实现。首先根据设定的变异概率判断是否需要进行变异,如果存在变异的情况,则随机选择一个变异点,随机生成一个新的重构操作;然后用新生成的重构操作替换变异点处的重构操作。如图 4 所示,变异发生在第 3 个基因位,将重构操作 Inline class 变异为 Move class 操作。

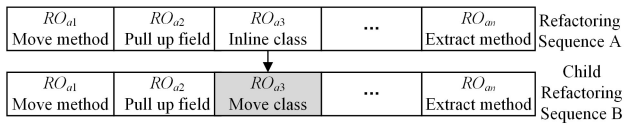



图 4 变异算子

Fig. 4 Mutation operator

本文在给定的概率下应用选择、变异和交叉算子,对新生成的种群中的个体使用适应度函数进行评估,不断重复此过程,直到满足一个停止准则,在所有新生成的种群中找到一组最优解,即最佳的重构序列。

### 3.2.3 适应度值函数

在每次迭代过程产生新个体之后,需要对每一个个体进行评估,本文通过计算适应度函数得到适应度值,从而对个体的优劣进行区分,适应度值越高意味着个体被选择的概率越大。在重构阶段,本文选择不同优化目标,通过使用不同的适应度函数分别评估不同的解决方案。

本文选择的用于设计适应度函数的 7 个目标包括:1)最大化软件可重用性;2)最大化软件可理解性;3)最大化软件灵活性;4)最大化软件有效性目标;5)最大化代码异味纠正率;6)最小化重构操作成本;7)最大化与之前重构记录的一致性。

结合表 1,对于质量属性目标,QMOOD 在之前的工作中使用频率最高且具有一定代表性,因此本文选择 QMOOD 作为质量目标。QMOOD 质量目标中使用的度量包括 OO metric、内聚性、耦合性所用到的度量,因此本文不再重复使用这

3个目标进行评估。QMOOD[20]涉及的质量属性包括软件的可重用性、可理解性、可扩展性、功能性、有效性以及灵活性这6个方面。本文不选择功能性作为优化目标，因为重构的前提就是在保留外部行为的同时改进内部结构。此外，本文也排除了可扩展性目标，因为可扩展性目标在某种程度上是一个主观的质量因素，仅使用静态度量的模型来评价可扩展性目标是远远不够的。

对于非质量目标，除本文选择的3个目标外，如表1所列，其他常见的非质量目标还有用户交互目标[11,18,29]、代码安全性[24,30]、代码优先级[22]、重构覆盖率[22]、元素近期性[22]等目标。本文不考虑上述目标的原因是：用户交互目标与开发人员本身的代码能力有很大关系，主观性过大；此外，其他目标，如代码安全性、代码优先级、重构覆盖率、元素近期性等均只适用于特定的场景和项目，具有一定局限性。

因此本文选取通用性目标作为考虑。下面是对7个优化目标适应度函数的介绍。

本文选择的4个质量属性目标包括软件可重用性、可理解性、灵活性以及有效性，定义如表3所列。

表3 质量属性目标

Table 3 Objectives of quality attributes

| Quality attribute | Calculation |
|---|---|
| Reusability | $=-0.25*DCC+0.25*CAM+0.5*CIS+0.5*DSC$ |
| Flexibility | $=0.25*DAM-0.25*DCC+0.5*MOA+0.5*NOP$ |
| Understandability | $=-0.33*ANA+0.33*DAM-0.33*DCC+0.33*CAM-0.33*NOP-0.33*NOM-0.33*DSC$ |
| Effectiveness | $=0.2*ANA+0.2*DAM+0.2*MOA+0.2*MFA+0.2*NOP$ |

表4列出了本文选取的4个质量属性目标所使用到的度量值[20]。

表4 质量属性目标所用的度量值

Table 4 Metrics for quality attributes

| Design Property | Metric | Description |
|---|---|---|
| Design size | DSC | Design size in classes |
| Complexity | NOM | Number of methods |
| Coupling | DCC | Direct class coupling |
| Polymorphism | NOP | Number of polymorphic methods |
| Hierarchies | NOH | Number of hierarchies |
| Cohesion | CAM | Cohesion among methods in class |
| Abstraction | ANA | Average number of ancestors |
| Encapsulation | DAM | Data access metric |
| Composition | MOA | Measure of aggregation |
| Inheritance | MFA | Measure of functional abstraction |
| Messaging | CIS | Class interface size |

本文选择的4个质量属性目标如下：

(1)可重用性。在软件开发时，对模块进行重用可以快速开发可靠、适应性强并且灵活的软件系统，可重用性是面向对象设计质量的一个重要目标。可重用性可以在不发生较大工作量改动的情况下重新用于解决新问题。当软件系统拥有较高内聚性、较低的耦合性以及较大的接口类与设计规模时，则认为软件的可重用性越高。可重用性适应度函数定义如下：

$$Reusability=\frac{Reusability_{after}-Reusability_{before}}{Reusability_{before}} \quad (1)$$

(2)灵活性。软件的灵活性不管是对于开发人员还是对于用户来说都是一个重要特性，随着软件的发展，开发人员会添加一些新的功能，这会导致软件的灵活性变差，而重构可以改善软件的灵活性。灵活性允许在软件设计中对不同的功能进行合并变更，反映了软件系统适应相关功能的能力。当软件具有较好的封装、多态以及较低的耦合性时，软件的灵活性越高。灵活性适应度函数定义如下：

$$Flexibility=\frac{Flexibility_{after}-Flexibility_{before}}{Flexibility_{before}} \quad (2)$$

(3)可理解性。可理解性更加关注软件产品的设计特征，研究[1]表明，软件开发人员通常会花60%的时间去理解他们正在维护的代码，因此，将可理解性目标作为一个重构中的优化目标是合理的。可理解性与多个度量有关，继承和多态的使用使软件可重用性、功能性得到了提升，但是可能会对灵活性和可理解性产生负面影响，复杂性越高，可理解性越差。可理解性适应度函数定义如下：

$$Understandability=\frac{Understandability_{after}-Understandability_{before}}{Understandability_{before}} \quad (3)$$

(4)有效性。软件有效性表示软件实现预期功能和行为的能力。研究表明，封装、多态以及抽象的使用都会提升软件的有效性。有效性适应度函数定义如下：

$$Effectiveness=\frac{Effectiveness_{after}-Effectiveness_{before}}{Effectiveness_{before}} \quad (4)$$

本文选择的3个非质量目标如下：

(1)最大化代码异味纠正目标。可维护性缺陷，也被称为代码异味。因此，出于对软件可维护性方面的考虑，开发人员应当尽早地识别并重构代码以纠正代码异味。根据相关工作[6]，如表5所列，本文主要关注以下6种代码异味类型[1,6]，包括Blob，Spaghetti Code，Functional Decomposition，Data Class，Shotgun Surgery，Feature Envy，因为它们是最常见的开发人员通过重构消除的异味。

表5 代码异味类型

Table 5 Code-smell types

| 代码异味种类 | 定义 |
|---|---|
| Blob | 赋予一个类过多职责，类中包含过多的成员变量或方法，增加类的理解难度 |
| Spaghetti Code | 有较为复杂的控制结构，对可理解性、可维护性有负面影响 |
| Functional Decomposition | 当设计一个类的目的是执行单个功能时，就会发生功能分解 |
| Data Class | 只有一些字段和用于访问这些字段的函数，单纯用作数据存储的类 |
| Shotgun Surgery | 多个类之间的耦合性过高，当外界发生变化时，需要同时修改软件中的多个类 |
| Feature Envy | 某个类的方法必须通过调用其他类中的大量数据才能完成其自身的工作，即该方法对其他类中数据的依赖远超宿主类 |

代码异味纠正率[10](Code-smell Corrected Ratio，CCR)由以下适应度函数定义：

$$CCR=\frac{\#Correctedcodesmells}{\#Totalcodesmells} \quad (5)$$

当代码重构后，代码异味被纠正越多，代码异味纠正率越高，其中 #*Corrected code smells* 表示算法纠正的代码异味的数量，即重构前后代码异味数量间的差值，# *Total code smells* 表示重构之前代码异味的总数。适应度函数返回纠正的代码异味数量与重构前总的代码异味数量之比。代码异味的检测利用相关工作中提出的检测规则进行[10,31]。已有研究证实，基于规则的检测在 8 个大规模系统上的平均准确率为 91%，召回率为 87%。

(2)最小化重构操作成本。不同的重构解决方案在修复相同的代码异味时所导致的代码更改量是不同的。在提高代码质量的目标之下，开发人员更愿意用更少的代码更改量去实现。Ouni 等[6]的工作也将最小化代码更改量作为优化的目标之一。在重构的第四个步骤，执行重构的过程中会涉及各种代码的更改。在生成重构建议时，最小化重构操作的成本，也就意味着最小化代码量的更改，这对于开发人员理解代码设计的改进有很大作用，并且可以减少开发人员在重构时的工作量[13]。

在表 2 列出的高级重构操作(HLR)中，每一个 HLR 均需要由若干的低级重构操作(LLR)实现，不同 LLR 的权重值不同，这取决于代码片段的复杂性以及更改之后造成的影响，表 6 列出了每个 LLR 的操作权重[6,10]。HLR 所带来的具体的代码更改量取决于 LLR 的种类以及权重。

表 6 低级重构操作权重

Table 6 LLR weights

| LLR | weight | LLR | weight |
|---|---|---|---|
| create class | 2 | redirect field access | 2 |
| delete class | 3 | add parameter | 1 |
| add method | 1 | remove parameter | 2 |
| delete method | 3 | rename method | 1 |
| add field | 1 | rename field | 1 |
| delete field | 3 | redirect method call | 2 |
| rename class | 1 | | |

本文通过计算代码重构所需要的代码更改量来定义最小化代码重构成本这个目标。为了方便后续多目标适应度值的计算，本文将最小化目标也转化为最大化目标，转化后的最小化操作数量成本适应度函数[10]如下：

$$Effort=1-\frac{\#Code\_changes}{\#Max\_Code\_Changes} \tag{6}$$

其中，#*Code_changes* 表示相应的代码更改量，#*Max_Code_Changes* 表示执行重构之后的最大代码更改量。一个重构操作序列由多个高级重构操作组成，代码更改量的计算如下：

$$Code\_changes=\sum_{i=1}^{n} HLR\_Code\_changes_i \tag{7}$$

其中，$HLR\_Code\_changes_i$ 表示第 $i$ 个 HLR 重构操作所带来的代码更改量，其中一个 HLR 重构操作序列由两个或多个 LLR 重构操作组成，*HLR_Code_changes* 的计算如下：

$$HLR\_Code\_changes=\sum_{i=1}^{p} W_i \tag{8}$$

其中，$W_i$ 表示第 $i$ 个 LLR 重构操作对应的权重值。

(3)最大化与之前重构记录的一致性。在提出新的重构解决方案时，在类似的环境下，与之前重构记录代码保持一致可以使开发者增强对提出的新的重构建议的信心。为了更好地指导搜索过程，当在类似的上下文中提出新的重构时，可以考虑过去应用的已记录的代码更改。这些知识可以与结构性信息和文本信息相结合，以提高重构建议的自动化程度。与之前重构记录一致性的适应度函数如下：

$$Similarityrefactoring(RO)=\sum_{j=1}^{n} e_j \tag{9}$$

其中，$n$ 表示记录重构的次数；$e_j$ 是一个重构权重，反映了建议的重构操作 RO 与记录的重构操作 RO 之间的相似性(见表 7[6,24])。权值 $e_j$ 的计算如下：如果两个重构操作有完全相同的类型(如“Move Method”和“Move Method”)，那么权重 $e_j=2$；如果重构操作比较相似，那么权重 $e_j=1$；(例如，一些复杂的重构操作，如“Extract Class”，可以由其他重构操作如“Move Method”“Move Field”“Create New Class”等组合在一起实现。)否则，权重 $e_j=0$。

表 7 不同重构操作间相似性权重

Table 7 Similarity weights between refactoring types

| | Move Method | Move Field | Pull Up Field | Pull Up Method | Push down Field | Push down Method | Inline class | Extract method | Extract class | Move class | Extract interface |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Move Method | 2 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| Move Field | 1 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| Pull Up Field | 0 | 1 | 2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Pull Up Method | 1 | 0 | 1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| Push down Field | 0 | 1 | 0 | 0 | 2 | 1 | 0 | 0 | 0 | 0 | 1 |
| Push down Method | 1 | 0 | 0 | 0 | 1 | 2 | 0 | 0 | 0 | 0 | 1 |
| Inline class | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 0 | 0 | 0 | 0 |
| Extract method | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 2 | 0 | 0 | 0 |
| Extract class | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 2 | 1 | 0 |
| Move class | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 |
| Extract interface | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 2 |

## 4 实验研究

本节将对本文所研究的问题、数据集、实验设置以及评估指标进行系统性的介绍。

### 4.1 研究问题

在本文的研究中，为了探索不同目标/组合对重构的影响，使用不同指标评估不同优化目标/组合所产生的解决方案的优劣。在实验阶段，通过分析以下 3 个研究问题对所提出的研究进行验证与分析。

RQ1：质量属性目标在实际重构中的表现如何？

RQ2：非质量目标及其组合在实际重构中的表现如何？

RQ3：非质量目标及其组合对质量目标有何种影响？

其中，RQ1 主要关注质量属性目标对实际重构效果的影响。本文选择 4 个质量属性目标作为单目标，包括软件可理解性、可重用性、有效性、灵活性。此外，参考之前的工作[22]，将 4 种质量属性目标归一化之后取平均值作为一个整体的质量目标(Q)并作为优化目标进行研究。在实验设计中，因为这 4 个质量属性目标是 4 个相关但不冲突的目标，所以我们将这 4 个不同的质量属性目标归一化取平均值之后视为一个整体质量目标(Q)进行研究，所用到的适应度函数为：

$$Quality=\frac{Q_{\text{after}}-Q_{\text{before}}}{Q_{\text{before}}} \tag{10}$$

其中，$Q_{\text{after}}$ 为重构后的质量值，$Q_{\text{before}}$ 为重构前的初始质量值。

RQ2 主要关注非质量目标及其组合对实际重构效果的影响。本文选择的 3 个非质量目标均为先前研究中使用频率较高的目标，包括代码异味纠正率、重构操作成本以及与之前重构记录的相似性 3 个目标。因为本文研究的 3 个非质量目标是互相冲突的目标，所以本文采用多目标进行实验。研究的目标及其目标组合如表 8 所列。

表 8 不同的非质量目标/组合

Table 8 Different non-quality objectives/combinations

| Objectives/Combination | Code smell | Refactoring Effort | Similarity with Recorded Code Changes |
|---|---|---|---|
| C | √ | | |
| E | | √ | |
| S | | | √ |
| C-E | √ | √ | |
| C-S | √ | | √ |
| E-S | | √ | √ |
| C-E-S | √ | √ | √ |

RQ3 主要关注非质量目标及其组合对质量目标(Q)在实际重构中的影响。Deb 等证明了 NSGA-II 方法容易受到目标数量的影响[32]，若目标数量超过 3 个，则帕累托的优势将减弱，参考文献[22]的实验设计，本文将之前的 4 个质量属性目标作为一个整体质量目标(RQ1 的实验结果表明，将 4 个质量属性目标作为整体质量目标(Q)的效果最佳)，考虑质量目标对非质量目标组合的影响，不同目标/组合如表 9 所列。

表 9 不同目标/组合

Table 9 Different objectives/combinations

| Objectives/Combination | Quality | Code smell | Refactoring Effort | Similarity with Recorded Code Changes |
|---|---|---|---|---|
| Q | √ | | | |
| Q-C | √ | √ | | |
| Q-E | √ | | √ | |
| Q-S | √ | | | √ |
| Q-C-E | √ | √ | √ | |
| Q-C-S | √ | √ | | √ |
| Q-E-S | √ | | √ | √ |

## 4.2 评测数据集

本文选择了 6 个不同规模的开源软件用于实验测试，包括 Xerces-J[1)]，JFreeChart[2)]，GanttProject[3)]，JHotDraw[4)]，Rhino[5)]和 Apache Ant[6)]等不同领域的软件，它们均为 Java 项目。Xerces-J 是由 Apache 组织推动的一项 XML 文档解析开源项目；JFreeChart 是一个开放的图表绘制类库；GanttProject 是一个使用甘特图进行项目调度和管理的工具；JHotDraw 是一个用于结构化绘图的 Java GUI 框架，主要用于支持用 Java 开发的图形编辑器；Rhino 是一个用 Java 编写的 JavaScript 解释器和编译器；Apache Ant 是一个将软件编译、测试、部署等步骤联系在一起加以自动化的一个工具。以上 6 个软件项目均在之前的研究[6,10-11]中被使用过，它们包含大量需要重构的代码，并且这些系统在各自的领域都是较为重要的软件，在工业界认可度较高。

表 10 列出了实验中所用系统的描述信息，第 1－6 列分别是系统名字、系统版本号、类的数目、代码千行数、代码异味数目和在后续版本中使用到的重构操作数目。

表 10 系统统计信息

Table 10 Programs statistics

| System | Release | Classes | Kloc | Code-smells | Refactorings |
|---|---|---|---|---|---|
| Xerces-J | v2.7.0 | 991 | 240 | 61 | 80 |
| JFreeChart | v1.0.9 | 521 | 170 | 51 | 96 |
| GanttProject | v1.11.0 | 245 | 41 | 60 | 63 |
| JHotDraw | v6.1 | 585 | 21 | 22 | 36 |
| Rhino | v1.7R1 | 305 | 42 | 79 | 50 |
| Apache Ant | v1.8.2 | 1 191 | 255 | 61 | 74 |

## 4.3 实验设置

本文主要使用 NSGA-II 算法进行实验的研究，但是遗传算法本质上具有随机性，为了减少算法随机性对实验的影响，对于本文的每个任务，重复进行 10 次独立模拟运行，取实验结果的平均值。NSGA-II 可以灵活地配置目标的数量并且在执行中考虑不同的目标。由于遗传算法中涉及参数的校准，如种群规模(Population Size，即种群中个体的数目)、最大迭代次数、交叉与变异概率以及染色体长度(即每个个体中的重构操作数目)，本文将最大染色体长度设置为 200、种群规模为 50、最大迭代次数为 100 来进行实验。本文选取了 11 种重构方法，如表 2 所列。

## 4.4 评测指标

为了回答本文所提出的研究问题，本文选择了能够度量重构与开发者预期的一致性相关的指标，使用重构操作精确率(Refactoring Precision，RP)[6,10]来与重构操作召回率(Refactoring Recall，RR)[6,10]分析其对重构的影响，指标定义如下：

$$RP=\frac{SuggestedRefactorings\cap ExpectedRefactorings}{SuggestedRefactorings} \tag{11}$$

$$RR=\frac{SuggestedRefactorings\cap ExpectedRefactorings}{ExpectedRefactorings} \tag{12}$$

通过比较方法推荐的重构和预期的重构[10,26]，从而计算指标值。其中 *Suggested Refactorings* 表示算法所推荐的

1) http://xerces.apache.org/xerces-j/

2) http://www.jfree.org/jfreechart/

3) www.ganttproject.biz

4) http://www.jhotdraw.org/

5) http://www.mozilla.org/rhino/

6) http://ant.apache.org/

重构操作序列，*Expected Refactorings* 是由软件开发团队应用到之后版本的预期重构，表 10 最后一列为预期重构数量。

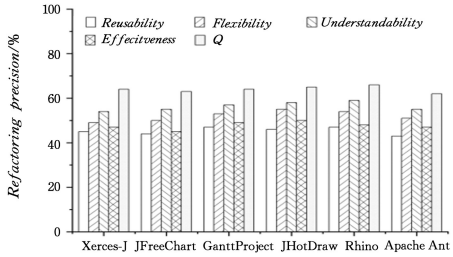
## 5 实验结果与分析

本节将分别对 3 个不同研究问题的实验结果进行讨论与分析。

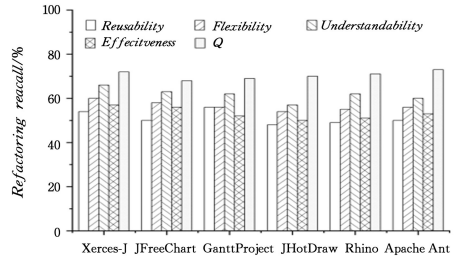
### 5.1 质量属性目标对重构结果的影响

图 5(a)给出了质量属性目标在 RP 指标下的实验结果，其中横轴表示 6 个不同的软件项目，纵轴表示不同目标的 RP 值。在 4 个质量属性目标中，可理解性目标在所有 6 个软件项目上表现最佳，在 Rhino 系统上最高可达到 59%；其次为灵活性目标，在 JHotDraw 系统上可达 55%。参考之前的工作[22]，将 4 个质量属性归一化取平均值之后，质量目标(Q)在 Rhino 系统上最高达到 66%，在 AntApache 系统上最低为 62%，在所有软件项目上的表现均优于单独使用某一种质量属性目标，说明仅仅关注软件的单一维度的质量属性对软件的改善效果是有限的。

图 5(b)给出了质量属性目标在 RR 指标下的实验结果，其中横轴表示 6 个不同的软件项目，纵轴表示不同目标的 RR 值。在 4 种质量属性目标中，表现较好的目标是可理解性目标与灵活性目标，最高都可达到 60%以上。其中质量目标(Q)表现最好，在 AntApache 系统上最高可以达到 73%，且均高于 4 种质量属性目标。




(a)Refactoring Precision(RP)




(b)Refactoring Recall(RP)

图 5 质量属性目标重构精确率(RP)与召回率(RR)

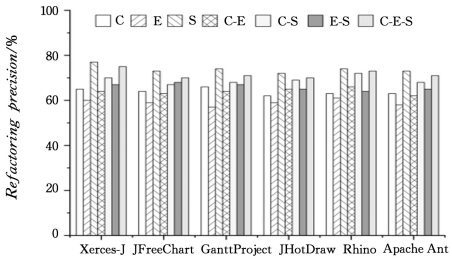
Fig. 5 RP and RR of quality attribute objectives

总体上来说，质量目标(Q)组合在两种指标上都表现较好，均高于 4 种质量属性目标。在 4 种质量属性目标中，可理解性和灵活性目标表现更佳，说明这两种目标对软件重构更为重要；其次，代码重构往往需要考量整体代码，整体质量目标(Q)的重构效果均优于单个的质量属性目标。

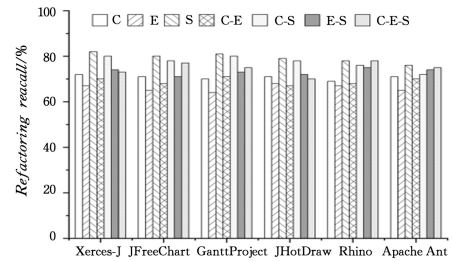
### 5.2 非质量目标对重构结果的影响

图 6(a)给出了非质量目标/组合在 RP 指标下的实验结果，其中横轴表示 6 个不同的软件项目，纵轴表示不同目标的 RP 值。可以看出，S 与 C-E-S 这两种目标/组合的效果更佳，其中与之前重构记录的一致性目标(S)在 Xerces-J 上达到了 77%，代码异味纠正目标(C)、重构操作成本目标(E)及与之前重构记录的一致性目标(S)的组合最高在 Xerces-J 系统上达到 75%。除此之外，代码异味纠正目标(C)和与之前重构记录的一致性目标(S)也表现较好，在 Xerces-J 系统上可以达到 70%。

图 6(b)给出了非质量目标在 RR 指标下的实验结果，其中横轴表示 6 个不同的软件项目，纵轴表示不同目标 RR 值。其中，S 与 C-S 这两种目标/组合的重构效果优于其他目标。与之前重构记录的一致性目标(S)在 Xerces-J 上可以达到 82%，代码异味纠正目标(C)和与之前重构记录的一致性目标(S)在 Xerces-J 系统上最高可以达到 80%，优于其他目标/组合。




(a)Refactoring Precision(RP)




(b)Refactoring Recall(RP)

图 6 非质量目标重构精确率(RP)与召回率(RR)

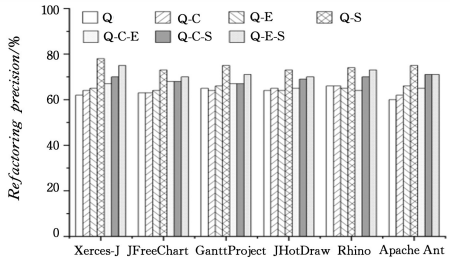
Fig. 6 RP and RR of non-quality attribute objective

总体上来说，6 种非质量目标/组合对软件重构均有提升效果。在 RP 与 RR 两个指标下，S，C-S 等重构效果优于其他目标/组合，说明与之前重构记录的一致性目标以及代码异味目标更符合开发者预期的重构，因此在之后的重构过程中可以更多地使用以上目标/组合。

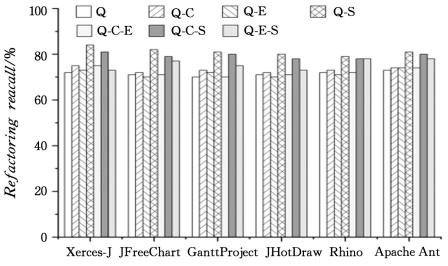
### 5.3 非质量目标对质量目标的影响

图 7(a)和图 7(b)分别给出了不同目标组合在重构精确率(RP)、重构召回率(RR)两种指标下的实验结果。其中横轴表示不同目标组合，纵轴表示两个指标相应的值。从图 7(a)可以看出，非质量目标/组合对质量目标均有提升作用，其中表现较好的目标有 Q-S 和 Q-E-S，质量目标(Q)和与之前重构记录的一致性目标(S)组合在 Xerces-J 上达到 78%，对软件重构有较大的提升，比单独使用质量目标(Q)提升了 16%。从图 7(b)可以看出，除了质量目标(Q)与重构操作成本目标(E)组合没有明显提升之外，非质量目标与质量目标均有明显提升的作用。从定量的角度可以发现，相较于单独使用 Q，Q-E 的目标组合产生了更少的代码更改量，这在实际的重构过程中被开发人员认为是可以接受的。其中提高最多的目标组合为 Q-S，最高在 Xerces-J 上达到 84%，Q-C-S 次

之，在 Xerces-J 上达到 81%。




(a) Refactoring Precision(RP)




(b) Refactoring Recall(RP)

图 7 不同目标组合的重构精确率(RP)与召回率(RR)

Fig. 7 RP and RR of different types of objectives combinations

总的来说，在 RP 与 RR 两个指标下，大部分非质量目标/组合对质量目标均有增强效果。其中表现较好的组合有 Q-S 以及 Q-C-S 组合，且均优于单独使用质量目标(Q)的效果。证明了在使用质量目标的基础上，与代码异味目标或与之前重构记录的相似性目标组合在一起时，重构效果更好。

## 6 有效性威胁

首先考虑内部有效性。内部有效性主要与有可能影响到实验正确性的内部因素有关。NSGA-II 算法的随机性意味着每次运行实验都有可能获得不同的结果。本文对于每种实验任务均运行 10 次，取结果的平均值，可以在一定程度上降低这种威胁。如何确定遗传算法中的更佳参数组合一直是遗传算法中研究的重要问题，在本文中，我们使用了最常用的参数设置。

外部有效性关注结果和结论的概括程度是否具有普遍性。主要包括两个方面：1)遗传算法是一种鲁棒性高的随机优化方法，实验研究的结果在一定程度上具有普遍性；2)本文研究所用到的 6 个软件项目均在之前的研究被使用过，并且涵盖了从中型到大型的软件规模，数据集具有一定的代表性。本文所选取的 11 种重构方法以及 6 种代码异味类型均在之前的研究中得到证实，具有一定的代表性。

结论有效性主要关注使用的评估指标是否合理。RR，RP 等指标用来衡量软件重构序列的优劣，已在之前的研究中被证实是有效的，具有一定的客观性，可以从不同的方面评价符合开发者预期的重构效果。

**结束语** 软件重构的研究是软件工程领域的研究热点之一。本文对基于搜索方法的不同目标组合的软件重构进行了研究。本文首次系统地分析了不同类型重构目标对生成满足开发者预期的重构结果的作用，以及不同类型重构目标的多目标组合是否有更好的效果。建立了一种基于 NSGA-II 算法的多目标软件重构评估框架，研究了 7 个不同优化目标对实际重构效果的影响。本文在 6 个不同规模的开源软件项目中使用与开发者预期的一致性的指标去评估重构的效果。实验结果表明，在软件重构过程中，单独使用质量单目标的效果欠佳，非质量多目标/组合与质量目标(Q)的效果明显优于质量属性目标。目标的选择对搜索过程起决定作用，不同优化目标所产生的重构解决方案区别较大。后续的工作主要包括：首先，将该研究扩展至其他编程语言，如 C 和 C++等；其次，计划在更多开源系统上进行进一步的研究。

## 参 考 文 献

[1] BROWN W H, MALVEAU R C, MCCOR-MICK H W S, et al. AntiPatterns: refactoring software, architectures, and projects in crisis [M]. John Wiley & Sons, Inc., 1998.

[2] ERLIKH L. Leveraging legacy system dollars for e-business[J]. IT Professional, 2000, 2(3): 17-23.

[3] BELL D. Software Engineering: A Programming Approach[M]. Hoboken: Addison Wesley, 2000.

[4] OPDYKE W, JOHNSON R. Refactoring: An Aid in Designing Application Frameworks and Evolving Object-Oriented Systems [C]// Proceedings of the Symposium on Object Oriented Programming Emphasizing Practical Applications (Sooppa'90). 1990: 145-161.

[5] MKAOUER M W, KESSENTINI M, BECH-IKH S, et al. High dimensional search-based software engineering: finding tradeoffs among 15 objectives for automating software refactoring using NSGA-III[C]// Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation. 2014: 1263-1270.

[6] OUNI A, KESSENTINI M, SAHRAOUI H, et al. Multi-criteria code refactoring using search-based software engineering: An industrial case study [J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2016, 25(3): 1-53.

[7] MOHAN M, GREER D. A survey of search-based refactoring for software maintenance[J]. Journal of Software Engineering Research and Development, 2018, 6(1): 1-52.

[8] WU N, SONG F M, LI X D. Quantum search-based software engineering: An exploratory study[J]. SCIENTIA SINICA Informationis, 2015, 45(5): 623-633.

[9] MARIANI T, VERGILIO S R. A systematic review on search-based refactoring [J]. Information and Software Technology, 2017, 83: 14-34.

[10] OUNI A, KESSENTINI M, SAHRAOUI H, et al. Maintainability defects detection and correction: A multi-objective approach [J]. Automated Software Engineering, 2013, 20(1): 47-79.

[11] ALIZADEH V, KESSENTINI M, MKAOU-ER M W, et al. An interactive and dynamic search-based approach to software refactoring recommenda-tions[J]. IEEE Transactions on Software Engineering, 2018, 46(9): 932-961.

[12] MENS T, TOURWE T. A survey of software refactoring[J]. IEEE Transactions on Software Engineering, 2004, 30(2): 126-139.

[13] FOWLER M. Refactoring: improving the design of existing code

[M]. Hoboken: Addison-Wesley Professional, 2018.
[14] SENG O, STAMMEL J, BURKHART D. Search-based determination of refactorings for improving the class structure of object-oriented systems[C]// Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation. 2006: 1909-1916.
[15] KESSENTINI M, KESSENTINI W, SAHRA-OUI H, et al. Design defects detection and correction by example[C]// 2011 IEEE 19th International Conference on Program Comprehension. IEEE, 2011: 81-90.
[16] PRADITWONG K, HARMAN M, YAO X. Software module clustering as a multi-objective search problem[J]. IEEE Transactions on Software Engineering, 2010, 37(2): 264-282.
[17] HARMAN M, HIERONS R M, PROCTOR M. A New Representation and Crossover Operator for Search-based Optimization of Software Modularization [C]// GECCO. 2002: 1351-1358.
[18] BAVOTA G, CARNEVALE F, DE LUCIA A, et al. Putting the developer in-the-loop: an interactive GA for software remodularization[C]// International Symposium on Search Based Software Engineering. Berlin: Springer, 2012: 75-89.
[19] OUNI A, KESSENTINI M, SAHRAOUI H, et al. The use of development history in software refactoring using a multi-objective evolutionary algorithm[C]// Proceedings of the 15th annual conference on Genetic and evolutionary computation. 2013: 1461-1468.
[20] BANSIYA J, DAVIS C G. A hierarchical model for object-oriented design quality assessment [J]. IEEE Transactions on Software Engineering, 2002, 28(1): 4-17.
[21] O'KEEFFE M, CINNÉIDE M O. Search-based refactoring for software maintenance[J]. Journal of Systems and Software, 2008, 81(4): 502-516.
[22] MOHAN M, GREER D. Using a Many-Objective Approach to Investigate Automated Refactoring [J]. Information & Software Technology, 2019, 112: 83-101.
[23] FERNANDES E, CHÁVEZ A, GARCIA A, et al. Refactoring effect on internal quality attributes: What haven't they told you yet? [J]. Information and Software Technology, 2020, 126: 106347.
[24] ABID C, KESSENTINI M, ALIZADEH V, et al. How Does Refactoring Impact Security When Improving Quality? A Security-Aware Refactoring Approach [J]. IEEE Transactions on Software Engineering, 2022, 48(3): 864-878.
[25] OUNI A, KESSENTINI M, SAHRAOUI H, et al. Improving multi-objective code-smells correction using development history [J]. Journal of Systems and Software, 2015, 105: 18-39.
[26] MKAOUER M W, KESSENTINI M, BECHIKH S, et al. Recommendation system for software refactoring using innovization and interactive dynamic optimization[C]// Proceedings of the 29th ACM / IEEE International Conference on Automated Software Engineering. 2014: 331-336.
[27] JENSEN A C, CHENG B H C. On the use of genetic programming for automated refactoring and the introduction of design patterns[C]// Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation. 2010: 1341-1348.
[28] MKAOUER W, KESSENTINI M, SHAOUT A, et al. Many-objective software remodularization using NSGA-III [J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2015, 24(3): 1-45.
[29] GHANNEM A, BOUSSAIDI G E, KESSENTINI M. Model refactoring using interactive genetic algorithm[C]// International Symposium on Search Based Software Engineering. Berlin: Springer, 2013: 96-110.
[30] ABID C, ALIZADEH V, KESSENTINI M, et al. Prioritizing refactorings for security-critical code[J]. Automated Software Engineering, 2021, 28(2): 1-28.
[31] KESSENTINI M, KESSENTINI W, SAHRA-OUI H, et al. Design defects detection and correction by example[C]// 2011 IEEE 19th International Conference on Program Comprehension. IEEE, 2011: 81-90.
[32] DEB K, SAXENA D. Searching for Pareto-optimal solutions through dimensionality reduction for certain large-dimensional multi-objective optimization problems[C]// Proceedings of the World Congress on Computational Intelligence (WCCI-2006). 2006: 3352-3360.

**GUO Ya-lin**, born in 1996, postgraduate, is a student member of China Computer Federation. Her main research interests include multi-objective optimization and software refactoring.

**LI Xiao-chen**, born in 1989, Ph.D, associate professor, is a member of China Computer Federation. His main research interests include evolutionary computation, intelligent software engineering, open-source software engineering, and data analysis in software engineering.

(责任编辑:何杨)