计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.200800067

复杂软件系统的重构技术:现状、问题与展望

孟繁祎　王　莹　于　海　朱志良
东北大学软件学院　沈阳 110169
(mengfanyineu@163.com)

摘　要　软件重构是改善软件质量的一种重要手段,它在不改变软件外部行为特性的情况下,通过调整软件内部结构来提高软件的可理解性、可维护性和可扩展性。然而,随着开源软件的迅猛发展,软件的规模和复杂程度日益增加,现有的重构技术在应对规模庞大且复杂的软件系统时,重构效果并不如意。因此,提高重构技术的可扩展性一直是软件工程领域研究的热点。从技术负债角度出发,探究重构时机,思考重构技术对软件质量的深入影响,明确重构技术旨在寻找重构代码的自动化方法,从而降低维护成本,提高代码质量。文中对工程实例进行分析并对文献进行梳理,调研了自2010年至今国内外96篇相关领域的文献,尝试以复杂系统的视角对这些研究工作进行归纳、比较,提炼总结软件重构领域的研究方向与技术方法,探讨重构技术研究中的特点与难点,思考重构技术研究中存在的问题及未来的研究方向,对软件重构技术的研究趋势进行了展望。

关键词:软件重构;技术负债;代码异味;自动化工具;软件质量
中图法分类号　TP311

Refactoring of Complex Software Systems Research:Present,Problem and Prospect

MENG Fan-yi,WANG Ying,YU Hai and ZHU Zhi-liang
Software College,Northeastern University,Shenyang 110169,China

Abstract　Software refactoring is the process of improving the design of existing code by changing its internal structure without affecting its external behavior,with the main aim of improving the quality of software products. Therefore,there is a belief that refactoring improves quality factors such as understandability,maintainability,and extensibility. With the rapid development of open source software,the size and complexity of software are continuously increasing. The refactoring result is less than satisfactory based on large-scale and complex software systems. Therefore,improving the scalability of refactoring technology has always been a hot topic in the software engineering field. From the perspective of technical debt,this paper explores refactoring opportunities and consides the impact of refactoring technology on software quality. The refactoring technology should provide an automated refactoring approach to reduce maintenance costs and improve code quality. Based on the analysis of engineering examples and literature review,this paper investigates 96 domestic and foreign literature in related fields since 2010. It first compares these researches from the perspective of complex systems and summarizes the research direction and technical methods in the field of software refactoring. Then,it explores the characteristics and difficulties and considers the problems and shortcomings in the research of refactoring technology. Finally,the research trend of software refactoring technology is discussed.

Keywords　Software refactoring,Technical debt,Code smell,Automated tool,Software quality

1　引言

软件重构是改善软件可理解性、可维护性和可扩展性的关键软件技术,它是软件工程领域的研究热点及重要实践。软件重构不仅改变了软件过分依赖前期设计的局面,力求得到恰如其分的软件结构,同时还保持了设计的简单性以及灵活性[1],因此得到了学术界和工业界的广泛关注。目前已有一系列有关重构的研究,研究内容主要集中在不良程序结构的识别和检测、程序理解方法、基本重构算法和复合重构算法,以及自动化软件重构工具。如图1所示,从整体来看,软件重构是一个"识别-实施-评估"的迭代过程:首先针对可靠性受到严重威胁的软件系统,识别出具有设计缺陷的不良代码或历经版本演化而导致架构混乱的代码结构;然后针对选中的不良代码,为其找到合适的重构策略;待重构操作完成后,评估软件的重构效果并重复这一过程直至不良代码被消除,从而提高软件代码本身的质量,对系统功能元素进行验

到稿日期:2020-08-11　返修日期:2020-09-14　本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家自然科学基金(61977014,61902056,61603082);中央高校基本科研业务费专项资金(N2017011,N2017016)
This work was supported by the National Natural Science Foundation of China(61977014,61902056,61603082) and Fundamental Research Funds for the Central Universities of Ministry of Education of China(N2017011,N2017016).
通信作者:于海(yuhai@mail.neu.edu.cn)

证，保证重构操作前后系统行为的一致性；最终，根据需求为系统扩展新的功能。

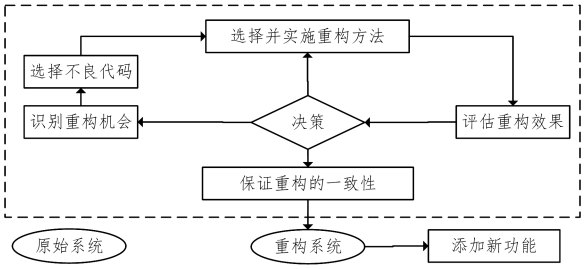


图 1 软件重构过程

Fig. 1 Software refactoring process

随着软件向网络化、生态化以及复杂化发展，其创新模式、知识管理、软件开发技术等都发生了巨大变化。在软件的开发和维护过程中，为了引入新功能、适应新应用场景以及移除软件缺陷，软件系统不断进行演化，其复杂性也不断增加。与此同时，在版本演化的过程中，过短的开发周期往往会使得开发人员减少对高质量的软件设计、良好的开发习惯以及测试覆盖率等方面的关注，从而导致技术负债的出现[2]。技术负债的概念最早由 Cunningham 于 1992 年提出，是一种反映技术折中的隐喻说法，即在尽可能短的时间内为了暂时的利益而采取合适的但技术欠佳的决策[3]。技术负债被认为是软件项目风险管理中最具挑战性的一项研究，管理复杂软件系统中的技术负债尤为困难。现如今开源软件系统逐渐趋于复杂，由此引发了大量技术负债，例如需求、架构、设计、代码、测试、构建、文档、基础设施、版本控制、缺陷等技术负债。此外，重构与技术负债有着密切的联系，它是有效消除债务的技术策略之一[4-6]。经过重构的代码往往比未经重构的代码具有更高的质量，因此重构对于保证软件质量有着重要的实际意义和应用价值。但是，并不是所有的技术负债都需要被重构，而是需要考虑债务本身是否达到合适的重构时机，否则会适得其反，从而降低软件的质量[7]。因此，在对软件系统进行有效维护前识别出合适的重构时机尤为关键。现有关于标识重构时机的常用方法主要包括面向软件质量的度量方法、面向先决条件的度量方法及面向集群的度量方法[7-9]。

为了深入了解复杂软件系统重构技术的发展动态，本文调研了自 2010 年至今的国内外 96 篇相关领域的文献，以近 10 年的文献分析来展示软件重构问题的研究脉络，通过对相关技术的综述，期望为未来研究寻求可能的科研契机与突破方向。本文的主要内容包括：

(1)以“识别-实施-评估”的重构过程和复杂系统的视角对已有的研究工作进行归纳、比较，提炼总结软件重构领域的研究方向与技术方法。

(2)分析软件重构技术的研究内容，探讨当前工作面临的问题与挑战。

(3)根据目前研究中存在的问题，对未来的研究方向和发展趋势进行展望，提出未来可能的科研重点与突破角度。

2 研究现状

为了把握软件重构技术的研究现状及发展趋势，本节通过工程实例分析和文献梳理，对研究内容进行收集整理，再根据现有工作的研究内容进行分类，并逐一展开讨论。

2.1 代码异味的识别和检测

在进行重构操作之前，如何识别检测出需要被重构的不良代码或混乱的代码结构是进行后续重构的先决条件。代码异味是程序设计中存在的不良设计模式或设计缺陷，也是对可读性、可理解性以及易变性等软件质量造成消极影响的不良代码结构[10]。研究人员通过自身对代码异味的理解，以及对其表现特征的分析和把握，提出了各种各样的检测方法。目前，具有丰富领域经验知识的专家依靠手工代码分析小型软件系统中的代码异味，但随着软件系统的规模不断扩大，逻辑变得愈加复杂，检测的难度也越来越大，手工代码分析由于缺乏针对性、效率低、不具备细致的检测能力，因此并不适用于复杂且规模庞大的软件系统。学术界和工业界提出了一些自动化检测代码异味的方法，这些方法也正在不断融入新的计算技术和数学模型，在软件重构研究工作中发挥着重要作用。自动化检测代码异味的方法根据检测技术的不同可以分为以下几种：基于度量的代码异味检测、基于规则的代码异味检测、基于搜索的代码异味检测以及基于可视化的代码异味检测。表 1 直观地列出了上述检测方法的优势与局限[11]。

表 1 代码异味检测方法的汇总

Table 1 Summary of code smell detection approaches

检测方法	优势	局限
基于度量的代码异味检测	可以直接提取源码的特征进行检测，具有较快的检测速度	主观性很强，结果的准确性取决于是否正确选择阈值，但阈值的选择通常依赖开发人员的经验，因此可靠性不高
基于规则的代码异味检测	将代码异味的症状转化为规则进行检测，每一种规则是人为可扩展及可解释的，并且为对应的异味特别制定的	没有相同的解释来定义标准规则，检测结果的精确度较低
基于搜索的代码异味检测	主要应用机器学习算法检测异味，可以分析大量的软件系统	主要依赖数据集和训练集的质量，在处理未知和变化的代码异味时具有一定的局限性
基于可视化的代码异味检测	通过可视化数据的趋势和模式来识别代码异味，同时可以系统地查看大规模复杂软件系统中代码异味存在的范围	研究人员的判断能力会受可视化类型的影响，容易出现错误的判断

2.1.1 基于度量的代码异味检测

基于度量的代码异味检测指首先度量表示源代码特征的数值，如属性数量、代码行数、参数数量、方法数量及类数量等数据，然后结合指标的阈值来判定是否为代码异味。这方面代表性的工作如下。Marinescu 团队提出了 inCode 插件，此插件根据系统所存在的问题，设置相应检测代码异味指标的阈值，实现了对 4 种代码异味的检测，同时可以持续评估系统的软件质量，帮助开发人员进行重构决策[12]。Veerappa 等将度量耦合性和内聚性的质量属性作为检测代码异味的依据[13]。Fard 等提出了 13 种关于 JavaScript 语言的代码异味，结合面向对象语言的一些指标来检测代码异味，并创建了第一个关于动态语言的代码异味检测工具 JSNose[14]。Chen 等使用 3 种不同的过滤策略来指定度量阈值，提出了基于

Python语言的代码异味检测技术[15]。Jiang等针对Large Class这一代码异味,提出了一种基于类长度的分布模型和基于内聚性度量的异味检测方法[16]。

2.1.2 基于规则的代码异味检测

基于规则的代码异味检测指,结合规则以及逻辑表达式,分析代码异味的表现症状,将其转化为检测规则,最终选择合适的阈值进行检测。Carvalho等提出了针对Android应用平台的20种代码异味类型,将其分为了与组件相关的异味以及与资源相关的异味两种类型,并针对每种异味类型提出了检测规则,设计了一套检测工具[17]。Moha等提出了DECOR工具,通过抽象异味概念来获取关键词汇表,使用领域特定语言并以规则卡片的形式执行规范,最后根据规则卡片创建模型,自动生成检测算法[18]。

2.1.3 基于搜索的代码异味检测

基于搜索的代码异味检测的核心是应用不同的算法和规则直接在源代码元素中检测异味,主要使用的算法包括机器学习[19]和深度学习[20]等。这方面具有代表性的工作是Palomba团队所提出的方法,即通过抽取特定的代码结构信息,并分析其演化情况,诊断出其中可能存在的代码异味。例如,通过挖掘版本演化过程中经常同时修改的方法集合,来识别Shotgun Surgery这一代码异味[21-22]。由于基于度量的代码异味检测与基于规则的代码异味检测均需要设定阈值,因此基于机器学习的代码异味检测研究呈明显上升趋势[23]。Fontana团队首先收集了大量不同类型的复杂软件系统,使用一组检测工具来识别代码异味以记录结果,然后人工评估结果报告中代码异味的候选对象,并为它们分配不同程度的权重,人工标记的作用是训练监督分类器,并将其性能(精确度、召回率等)与其他工具进行比较[24]。该团队在之后的工作中不仅认为代码异味的严重程度是检测结果准确性的一个重要因素,还认为根据代码异味的严重程度可以给出重构顺序,从而使重构的效果达到最大化,因此提出了基于机器学习的代码异味严重程度分类方法,同时通过采用二分类模型取得了较好的检测结果[25]。Nucci等分析了前人提出的方法,发现Fontana团队[23]在研究中之所以能够取得较好的性能,因为使用了特定的数据集,而不是机器学习算法在代码异味检测方面的实际能力在起作用[26]。

2.1.4 基于可视化的代码异味检测

基于可视化的代码异味检测指结合人类专业能力与自动化检测过程而形成的一个人机交互的半自动化检测方法。该检测方法可以使用图形来处理复杂且规模庞大的软件系统,帮助开发人员确定需要改进的代码点,从而减少代码异味。在这方面的工作中,Steinbeck提出了一种可视化方法,通过结合树状图、热图以及柱状图来帮助开发人员查看任意代码异味的分散情况以及演化情况[27]。Mumtaz等利用平行坐标图和径向坐标可视化的形式来呈现多变量软件质量度量指标,这些指标可以用于评判代码异味;同时开发了一个交互式的视觉分析系统,支持对异味的自动检测以及对系统其他方面异常模式的分析[28]。

2.2 软件重构

随着开源软件持续性演化,软件设计缺陷逐渐浮现,软件规模与复杂程度日益增加。为了应对如今规模庞大且复杂的软件系统,并保证其良好的软件质量,软件重构技术层出不穷。本文依据研究领域的热点主题对典型的重构技术进行归纳、分类,表2列举了从不同角度划分的重构方法,主要包括基于搜索的重构方法、基于UML模型的重构方法、基于代码异味的重构方法、基于测试驱动的重构方法以及自动化重构工具。

表2 软件重构方法汇总

Table 2 Summary of software refactoring

研究角度	重构方法
研究方法及实现技术角度	基于搜索的重构方法[29]
研究场景角度	基于测试驱动的重构方法[30]
研究与工程应用角度	自动化重构工具[11]
研究对象及研究内容角度	基于UML模型的重构方法[31]
	基于代码异味的重构方法[11]

2.2.1 基于搜索的重构方法

软件重构的主要任务与目标是找到一个最有效的重构序列,但这一直被认为是软件工程领域的难点[32]。开发人员对软件进行维护时,通常需要进行多次重构。由于每执行一次重构就可能会触发另一个重构的前置条件,因此只能执行所有重构点的子集。又因为不同子集的重构对软件质量的影响存在一定的差异,所以构建一个合理的重构序列是保证软件质量的关键。

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)应用于重构任务,利用软件度量作为目标来指导搜索过程,从而得到一个最优的重构序列。选择合理的问题描述模型,将重构问题转换为基于搜索的目标优化的解,用度量对目标搜索算子实施搜索优化,通过搜索在问题解空间中求得各种大规模异构性问题的优化解,即得到对应原问题的最佳方案[33]。现有的重构研究主要应用的搜索算法如表3所列。

表3 搜索算法

Table 3 Search algorithms

算法	定义
遗传算法	借鉴生物界自然选择和自然遗传机制的高度并行、随机性,自适应搜索最优解的算法[34]
遗传编程算法	利用达尔文生物进化思想设计的一种优化算法[35]
非支配排序遗传算法	基于遗传算法的多目标优化算法[35-36]
爬山算法	一种局部择优的方法,采用启发式方法,对深度优先搜索进行改进,利用反馈信息生成解的决策[37]
模拟退火算法	基于蒙特卡罗迭代求解策略的一种随机寻优算法[38]
粒子群优化算法	一种利用群体在解空间中找寻最优粒子进行搜索的计算智能方法[39]

重构序列的组合所形成的解空间十分庞大,因此从庞大的解空间中寻求一个最优的重构序列是十分困难的,而这正是搜索算法的优势所在。如图2所示,首先,针对需要被改进的软件系统,将此类系统转换成软件组件(类或方法)的集合或者抽象表示为图形的形式并作为算法的输入;然后,使用重构建议、软件质量度量指标和其他信息作为算法的指导过程,并且将所有的重构操作映射为向量作为解决方案的表现形式;最后,利用适应度函数(Fitness Function)动态评估解的优

劣，在搜索收敛至最优或局部最优时，即可得到一组最优的重构序列。

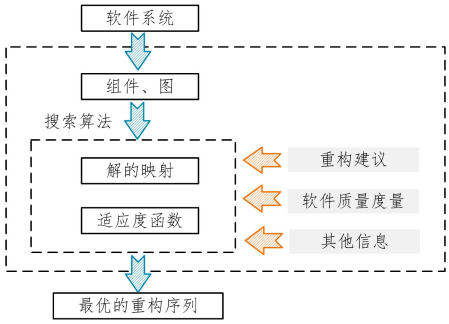


图 2 基于搜索的重构方法

Fig. 2 Search-based refactoring approaches

在这方面的工作中，Amal 等提出了一种基于神经网络的适应度函数，通过遗传算法手动评估重构的解决方案以进行少量迭代，然后利用人工神经网络进行训练以评估剩余迭代的重构解决方案[40]。在某些重构问题中，任何两个或多个目标之间可能存在冗余。Dea 等提出了一种新的软件重构方法，即 PCA-NSGA-II 多目标重构方法。该方法是在 PCA-NSGA-II 演化多目标算法的基础上改进而来的，通过去除冗余来保留冲突目标，从而避免维数灾难[41]。为了解决相互冲突的优化目标，Mkaouer 等提出了一种新的重构问题表示方法，即将每个需要改进的质量属性视为一个独立的优化目标，利用 NSGA-III 算法，使用 8 个不同的目标来评估重构解决方案[42]。

Mohan 等提出了一种基于多目标遗传算法的自动化重构工具 MultiRefactor。该工具将 4 个独立的软件度量作为优化目标进行重构操作，并分别测试了基于多目标和基于单目标的重构方法对软件质量的影响[43]。Boukharata 等提出了基于多目标搜索的优化方法 WSIRem，以帮助和服务开发人员改进服务接口的模块化[44]。一些研究工作还将图作为算法的输入，如控制流图[45-46]、调用图[47]以及类图[48]等。

2.2.2 基于 UML 模型的重构方法

模型驱动的软件工程(Model-Driven Software Engineering，MDSE)指在不同抽象层次上使用模型来开发、维护和演化软件系统的技术。这些模型通常使用通信和系统文档的草图来作为系统详细规范[49]。随着建模技术的不断发展，其中面向对象建模最为活跃的统一建模语言(UML)在软件重构工作中得到了广泛的应用。UML 是对软件系统进行说明、可视化和编制文档的一种语言，基于 UML 模型的重构是一种特殊的模型转换，能够在保持模型内部质量特征的同时改进模型的结构，并且将软件重构的重点从源代码转移到设计模型的方法上。

Reimann 等利用 Eclipse 模型框架(Eclipse Modeling Framework，EMF)，提出了一种基于角色模型的通用重构方法，并使用 UML，Web Ontology Language (OWL)等多种建模语言和重构方法进行评估[50]。其中，UML 建模语言将结构完整性和正确性作为模型的约束，为了保证模型重构的准确性，在重构操作完成后，需要检查模型自身是否仍然满足约束条件，若不满足条件，则将拒绝执行重构。Steimann 通过将约束检查替换为约束求解，解释了如何将约束的作用从允许或拒绝尝试性重构转换为计算重构可执行所需的额外模型的变化程度，同时提出了从规则到重构所需的约束映射条件[51]。Arcelli 等针对 EPSILON 平台中一组著名的性能反模式，实现了对 UML 模型的检测规则和重构操作，并利用 EPSILON 语言检查属性实现了对模型的应用重构[52]。Einarsson 等将模型到模型的转换应用到底层模型以及相关的图表，提出了一种在 UML 工具中重构 UML 模型及其图的方法，同时实现了一个基于 Eclipse 的原型插件[53]。

为了满足软件演进的优化性能要求，Arcelli 等提出了一种工具 PADRE，它可以检测 UML 模型中的性能反模式，并对其重构以删除之前检测到的反模式[54]。Lu 等提出了基于搜索的对象约束语言重构方法(Search-Based OCL constraint Refactoring Approach，SBORA)，其利用 4 种语义保持性重构算子及 3 种对象约束语言的质量度量指标，来重构 UML 元模型中指定的约束集[55]。

2.2.3 基于代码异味的重构方法

代码异味一直是软件重构领域的热点问题。代码异味可能发生在软件演化过程中的任何阶段，部分代码异味具有一定的隐蔽性，难以检测，而软件系统自身的复杂性也导致了代码异味的多样性。同时，由于代码异味所处软件环境复杂，增加了基于代码异味的重构难度，因此吸引了一大批来自企业和学术界的研究者对代码异味的重构技术开展研究。

Bavota 团队提出了“Methodbook”算法来移除 Feature Envy 和 Inappropriate Intimacy 的代码异味，结合语义和结构相似性度量指标共同评估函数之间的“友谊”，寻找与每个函数关系最密切的类，将其作为搬移重构操作的目标类[56]。该团队在其他工作中，利用最大流最小割算法来分解 God class，并将此算法与图论相结合，将与语义紧密相关的函数和属性提炼出来形成新类[57]。Mumtaz 等调查了 TextFileAnalysis，Jtar，Cobertura，JgraphX，GanttProject 这 5 个软件系统的代码异味，针对其中 3 种不同类型的代码异味，重构系统中与软件安全度量指标相关的代码异味以提高软件系统自身的安全性[58]。Politowski 等调查了 Blob 和 Spaghetti Code 这两种代码异味对程序可理解性的影响。他们通过收集来自 3 所大学的 133 位不同参与者的 372 个调查数据发现，这两种代码异味同时出现会比单一出现在程序中对程序的可理解性影响更大，因此他们认为这两种代码异味具有关联性，应该一起被重构[59]。

Wang 等提出了基于多维度软件复杂网络模型的自动化重构技术，将软件系统映射成多依赖关系类型网络和方法级加权依赖网络，借助重构预处理操作以及加权聚类算法，根据“高内聚、低耦合”原则对系统的模块重新进行划分。同时针对非继承体系和继承体系内部由内聚和耦合性引起的代码异味，提出了 3 种不同类型的重构建议[60]。Bu 提出了一种基于深度神经网络的上帝类检测方法，通过帮助开发人员缩小人工检测的范围，来更快地锁定上帝类代码异味的重构时机[61]。Liu 等设计了一种代码异味的检测工具，通过检测结果提醒开发人员对软件系统进行重构，从而达到提高软件质量、降低开发成本的目的[62-63]。

Oliveira 等提出了一种推荐搬移方法和搬移字段的重构技术,用于移除由于软件演化性而导致的细粒度软件组件中协同变更的代码异味,同时保证在重构操作完成后不会引入新的静态依赖关系[64]。Sas 等认为软件依赖关系网络组件之间的方法调用和数据流动会随着版本的推进愈发复杂,进而使软件架构逐渐偏离最初的设计。因此,该团队调查了来自不同应用领域、具有不同规模的 14 个开源项目的共计 524 个版本的软件架构,分析了其中 3 种架构异味的演化性以及表现特征。通过调查发现,不同类型架构异味的增长情况、在软件依赖网络中随时间变化受影响情况以及对系统的影响时间都存在一定的差异。针对上述特征,他们给出了一些重构架构异味的优先顺序的建议[65]。Palomba 等调查了来自 60 个 Android 应用程序中的 9 种特有的代码异味,并分析了代码异味对移动应用程序能耗的影响程度。调查发现,具有 Internal Setter,Leaking Thread,Member Ignoring Method,Slow Loop 这 4 种代码异味类型的组件的消耗量是具有其他异味类型的组件的 87 倍。该团队认为重构代码异味是提高能源效率的关键因素[66]。

2.2.4 基于测试驱动的重构方法

基于测试驱动的重构方法指在重构源代码之前先调整测试代码,然后修改源代码以适应调整后的测试代码,从而保证正确地实施连续小步骤的重构,使代码更加精炼,以此获得瑕疵率极低的软件系统。

Xuan 等提出了一种新的名为 B 重构的测试驱动重构技术。B 重构主要是将一个测试用例分割成小的测试片段,而这些片段会覆盖控制流中一个更简单的部分,从而为动态分析提供更好的支持[67]。Gao 等通过建立重构与测试用例失败之间的映射关系来指导自动化修复测试用例、分析测试用例失败的原因以及重构对软件接口的影响,并提出了一种能够分析重构对回归测试用例影响的方法[68]。Chu 等举例说明了测试用例重构在基于模式的软件开发中的重要性,提出了一种四阶段的方法来指导设计模式的测试用例重构。其在初始阶段为功能和非功能需求指定基于角色的模式结构,以帮助构建测试用例[69]。

2.2.5 自动化重构工具

软件重构是一项耗时且复杂的代码调整过程,要求开发人员对代码整体进行感知,并针对“何处、何时以及如何”进行重构的问题做出复杂的决策。自动化重构工具作为降低重构成本、提高重构效率以及代码质量的重要手段,用于实现对复杂系统的快速检测及处理,可以为开发人员提供决策的辅助支持[70]。

针对面向对象语言的自动化重构工具,学术界一直有广泛的研究。这方面的工作有 Khatchadourian 等提出的 MIGRATE skeleton Implementation to Interface (MSITI)自动化重构工具,用于转换遗留 Java 代码以使用新的默认构造函数,分析复杂的类型层级结构,解决多个实现继承问题,协调类和接口方法之间的差异。该工具可以实现高效、完全自动化及基于类型约束的重构操作,同时也可以作为 Eclipse 插件。它的优点是重构后的代码在语义上与原始代码等价,并且简洁易懂,呈现了较高的模块化程度[71]。开发人员如果进行不当的异步编程会导致软件系统内存泄漏、结果丢失以及能源浪费等问题。为了解决上述问题,Lin 团队提出了一个自动化重构工具 ASYNCDROID[72-73],该工具可以使 Android 开发人员将错误使用的异步构造转换为正确的构造。同时该工具还可以分析被广泛使用的 611 个 Android 应用程序,将其作为语料库,探究 Android 应用程序的异步环境,了解开发人员如何改造异步应用程序以及改造时遇到的困难。表 4 列出了一些主流的自动化重构工具,描述了其重构的粒度及可重构的语言。

表 4 自动化重构工具

Table 4 Automated refactoring tools

工具	粒度	语言
MSITI[71]	Class/Method	Java
ASYNCDROID[72-73]	Class/Method	Java (Android)
BeneFactor[74]	Class/Method	Java
WitchDoctor[75]	Class/Method	Java
LambdaFicator[76]	Class/Method	Java
FaultBuster[77]	Class/Method	Java
Historef[78]	Class/Method	Java
Jdeodorant[79]	Class/Method	Java
TOAD[80]	Class/Method	Pharo
RefBot[81]	Class/Method	Java
RefDiff 2.0[82]	Class/Method, File/Class/Method, File/Method	Java, JavasScript, C
Ref-Finder[83]	Class/Method	Java
R3[84]	Packages/Class	Java
GenReferee[85]	Class	Java
SOMOMOTO[86]	Packages/Class	Java
DINAR[87]	Class/Method	Java
CMMiner[88]	Class/Method	Java
Refactory[89]	Class/Method	Python

2.3 重构对软件质量的影响

在识别、实施过程之后,对软件质量属性进行分析与度量,其主要目的在于探究系统结构特性与质量属性之间的关系,评估重构效果,发现存在的缺陷问题,从而判断是否需要对系统重新进行重构或者优化。软件质量属性分为软件内部质量属性和软件外部质量属性两种。软件内部质量属性包括内聚性、耦合性及代码尺寸等可以用代码本身来度量的属性;外部质量属性包括可维护性、易错性及可重用性等属性[90]。为了度量外部质量属性,软件工程师需要考虑软件环境以及该环境和软件组件之间的交互。在某些情况下,度量外部质量属性所需的信息可能并不可用。因此,研究人员在以往实证研究的基础上,提出了利用内部质量属性来评估外部质量属性的公式和模型,以探讨内部质量属性与外部质量属性之间的联系[91]。

Kannangara 等讨论了重构能否提高软件质量的问题。针对可分析性、可变性、时间行为及资源利用率等外部质量属性,选取了 10 种重构技术进行分析,发现“以多态取代条件式”这一重构技术改善软件质量的效果最好,而“引入空对象”的重构技术改善软件质量的效果最差[92]。Alshayeb 等也探究了重构对软件质量的影响,发现在软件质量属性中没有一致的改善趋势[93]。Elish 等根据重构方法对软件质量属性的影响程度进行分类,发现重构对软件质量不仅具有良性影响,还伴随着负面影响[94]。

由于软件系统除了在不断修复错误以外，也在不断添加新的功能，因此其架构缺陷问题亦日益增多，当问题累积到系统难以维护时，架构会逐渐被腐蚀，对软件质量造成严重影响。针对上述问题，Feng 等通过跟踪处理问题而修改的文件之间的交互作用来监控软件架构的演进，可以更早、更精确地检测到软件衰退的迹象，以提醒开发人员重构，从而保证软件质量[95]。Cai 等构建了一个体系架构模型 DRSpace，该模型可以从不用角度反映任意复杂软件系统的多个方面、特征、模式等，为分析软件质量提供了新的途径。Cai 等还提出了一种新的检测方法 ArchRoot，用来识别系统中具有缺陷倾向的文件，以提醒开发人员应该注意以及重构存在问题较多的文件[96]。

3 问题与挑战

综上所述，软件重构是软件工程的重要研究领域之一，在软件维护活动中占据关键地位。通过对软件重构领域国内外研究现状的分析，本文认为软件重构工作当前主要面临如下挑战：

(1)软件重构过程中的识别阶段，主要围绕代码异味的检测来展开分析。随着开源软件的不断发展，自身逻辑逐渐复杂化，代码质量参差不齐，复杂软件系统中的代码异味种类也越来越多。虽然原本笼统模糊的异味越来越细化，并被归类总结，但是仍有一些异味症状没有得到明确的定义及描述，例如，频繁依赖于第三方软件造成过度耦合的现象。因此，亟需提出新的异味定义来总结这些未被定义和描述的不良症状，并通过对应的异味检测和重构手段进行处理，以达到提高软件质量的目的。

(2)基于度量和基于规则的代码异味检测方法都需要设定阈值来进行检测。由于阈值选择具有主观性，因此在面对阈值选择对检测结果的影响这一问题时，需要考虑人为设定的合理性与完备性。但由于主观性的设定具有一定的不可解释性，因此亟需提出新的技术方法来避免主观性地确定阈值来检测代码异味，以便为后续重构操作提供精确的决策与支持。

(3)现有基于搜索的代码异味检测方法多以机器学习算法为基础，方法较为简单，主要依赖数据集和训练集的质量，无法支持复杂软件系统的检测。该方法只利用了部分代码异味的统计特征，在检测精确度方面仍有很大的提升空间。此外，基于机器学习的检测方法相比基于度量和规则的方法，可以检测代码异味的类型较少，因此未来应以检测多种代码异味类型为目标，利用机器学习技术作为基本手段，构建面向复杂软件系统的精准检测模型。

(4)在当今快速开发迭代的环境中，软件系统更新极为频繁。这种频繁的系统更新要求基于可视化的代码异味检测具备快速在线更新和适应的能力，以保证异味检测实时性。然而，现有基于可视化的检测方法基本采用离线模式呈现代码异味，无法适应持续性的系统演化。与此同时，由于检测结果会受可视化类型的影响，容易出现错误的判断，因此对自适应扩展及在线动态快速检测方法亟待展开深入的研究。

(5)目前，大多数基于搜索的重构方法将重构问题视为单目标优化问题，但基于单目标优化的方法仅仅为度量一个质量属性进行优化求解。然而，软件质量并非只由一个属性决定，而是与多种质量属性相关。因此，良好的重构方案应该对多个与重构问题相关的不同质量属性进行优化。针对此问题，一些研究提出了多目标优化方法，但是属性之间存在的冲突问题一直是学术界需要攻克的难点。

(6)基于模型重构的一个重要目标是，在不改变软件模型行为的前提下提高软件模型的质量。尽管软件质量保障技术已经逐渐成熟，但是对 UML 模型和模型驱动开发的适用性仍然处于初级阶段。因此，基于模型的重构除了需要适用于 UML 模型的质量度量之外，还需要一个在这些度量和外部质量属性之间建立关联的框架，通过探究模型度量和设计模式对模型重构技术的影响，来更好地对软件进行维护。

(7)目前，针对基于代码异味的重构技术，相关研究大多从代码异味所具有的属性特征的角度展开分析，且只将单一版本作为研究对象，并未放眼于软件项目多个版本所提供的演化规律，同时也未从系统和组成元素维度观测软件组件特征以及代码异味的变化趋势。因此，针对代码异味问题采取何种自动化重构技术才能获取改善软件质量的最优效果，还需要进一步的研究。

(8)重构是测试驱动开发的核心，在进行重构前后都应该采用单元测试。如果单元测试在逻辑上依赖代码本身，那么重构后进行测试的方法可能不适用于现有代码。因此，如何编写在逻辑上不依赖代码本身的单元测试还需要进一步的研究。

(9)现有自动化重构工具大多数是针对 Java 开发语言展开研究的，而且只针对单一语言作为研究对象，只有极少数工作扩展到多个开发语言，且不全面，因此研究者还需要进一步研究多语言自动化重构工具。

(10)由于软件的复杂性，很难以人工观察的方式发现系统的质量问题，因此开发软件质量分析的自动化工具辅助开发人员的软件系统评估工作，并将其应用到后续的自动化软件重构活动中，使得质量度量数据得到充分利用以提高软件质量，是学术界和工业界面临的一项艰巨任务。

4 未来研究展望

在大量分析和调研已有研究后，本节对未来的研究方向进行了展望。

(1)在开源软件社区飞速发展的今天，软件系统的规模和复杂程度不断提高，不仅企业因此面临着成本、质量和交付时间等多方面的压力，软件系统中的设计缺陷也越来越多。软件组件之间通过彼此的依赖关系构成了一个复杂的软件网络，其间的数据传递和依赖关系随着软件的演化也愈发复杂，这给传统的用于分析代码异味的技术如检测技术、静态分析技术等带来了新的挑战。因此，如何在不断追求检测效率的同时，寻找到最佳的检测效果，始终是一个值得研究的课题。

(2)应用场景的差异性是否导致软件重构结果的变化。软件重构在识别阶段多数关注代码异味与系统易发生变更或是发生错误倾向的关系，在实施阶段多数关注软件质量的改善程度。软件重构研究的本质是改善软件质量问题的初衷，

应用场景差异与重构结果变化方式间的关系是软件重构未来重要的研究方向。

(3)从软件重构与其他研究领域结合的角度来看,面对复杂的软件系统,通过利用数据分析、深度学习技术及信息检索方法对版本历史信息进行挖掘,对源代码信息及维护信息进行关联,找到需要重构的不良代码结构,为重构研究开辟了一条新的道路。

(4)开源时代的混源软件系统具有复杂异质异构、多源异步演化等特性,这也使得众多软件之间的依赖性较强。在软件维护过程中需要对整个软件体系之间的依赖关系有深入的认识,如何保障这种复杂软件系统的质量及提出具有针对性的重构技术,仍是一个开放的课题。

(5)复杂软件系统由于逻辑结构复杂化、功能智能化,导致其可靠性和安全性面临巨大的威胁。如何找到影响系统可靠性和安全性的原因,进而重构需要改进的代码点是最有效的重构思路。在不良代码分析方面,如何利用新的分析方法对复杂系统进行分析,获取关于不良代码的多维度信息,以提高检测分析的能力,是当前软件重构最具挑战性的问题之一。

(6)软件演化涉及软件的各方面,异常复杂,如何在软件演化的过程中关注其演化的效果及演化的质量,及时发现问题从而进行重构,避免问题积累使软件维护成本增加,一直是学术界及工业界关注的话题。因此,如何从演化的角度提出一种自动化的重构方法,是未来需要探讨的课题。

总体来说,软件重构技术在理论研究、自动化工具开发和应用推广等方面都有着重大进展,尤其是开源时代的到来,为重构技术带来了广阔的应用群体和快速的技术进步。但是,软件重构的不均衡发展也为该领域带来了大量有待解决的问题和难题。

结束语 开源软件规模和复杂程度的爆炸式增长,给软件重构工作带来了巨大的挑战。本文系统地阐述了软件重构研究的进展。首先,分别从基于度量的代码异味检测、基于规则的代码异味检测、基于搜索的代码异味检测以及基于可视化的代码异味检测等方面介绍了如何有效地检测不良代码结构,以提高复杂软件系统的设计缺陷诊断能力;然后,通过对已有重构技术展开分析,发现了一些现有技术难以解决的问题;接着探讨了重构操作对软件质量的影响,也为未来软件质量的保障提供了支撑;最后,指出现有研究工作中存在的问题和不足,并对未来的发展进行了展望。

参 考 文 献

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

[2] CODABUX Z,WILLIAMS B. Managing technical debt:An industrial case study[C]// International Workshop on Managing Technical Debt (MTD 13). New York:IEEE,2013:8-15.

[3] CUNNINGHAM W. The WyCash portfolio management system [J]. ACM Sigplan Oops Messenger,1992,4(2):29-30.

[4] BEHUTIYE W N,RODRÍGUEZ P,OIVO M,et al. Analyzing the concept of technical debt in the context of agile software development:A systematic literature review[J]. Information & Software Technology,2017,82:139-158.

[5] BESKER T,MARTINI A,BOSCH J. Managing architectural technical debt:A unied model and systematic literature review [J]. Journal of Systems & Software,2018,135:1-16.

[6] LI Z,AVGERIOU P,LIANG P. A systematic mapping study on technical debt and its management[J]. Journal of Systems and Software,2015,101(3):193-220.

[7] DALLAL J A,ABDIN A. Empirical Evaluation of the Impact of Object-Oriented Code Refactoring on Quality Attributes:A Systematic Literature Review[J]. IEEE Transactions on Software Engineering,2018,44(1):44-69.

[8] MOHAN M,GREER D. A survey of search-based refactoring for software maintenance[J]. Journal of Software Engineering Research & Development,2018,6(1):3.

[9] DALLAL J A. Identifying refactoring opportunities in object-oriented code:A systematic literature review[J]. Information & Software Technology,2015,58:231-249.

[10] TUFANO M,PALOMBA F,BAVOTA G,et al. When and Why Your Code Starts to Smell Bad (and Whether the Smells Go Away)[J]. IEEE Transactions on Software Engineering,2017,43(11):1063-1088.

[11] LACERDA G,PETRILLO F,PIMENTA M,et al. Code Smells and Refactoring:A Tertiary Systematic Review of Challenges and Observations[J]. Journal of Systems and Software,2020,167:110610.

[12] MARINESCU R,GANEA G,VEREBI I. InCode:Continuous Quality Assessment and Improvement[C]// European Conference on Software Maintenance & Reengineering (CSMR 10). LOS ALAMITOS:IEEE Computer Society,2010:274-275.

[13] VEERAPPA V,HARRISON R. An Empirical Validation of Coupling Metrics Using Automated Refactoring[C]// ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 13). Los Alamitos:IEEE Computer Society,2013:271-274.

[14] FARD A M,MESBAH A. JSNOSE:Detecting JavaScript Code Smells[C]// International Working Conference on Source CodAnalysis and Manipulation (SCAM 13). Los Alamitos:IEEE Computer Society,2013:116-125.

[15] CHEN Z F,CHEN L,MA W W Y,et al. Understanding metric-based detectable smells in Python software:A comparative study [J]. Information & Software Technology,2018,94:14-29.

[16] JIANG D X,MA P J,SU X H,et al. Detection and refactoring of bad smell caused by large scale[J]. International Journal of Software Engineering & Applications,2013,4(5):1-13.

[17] CARVALHO S G,ANICHE M,VERÍSSIMO J,et al. An Empirical Catalog of Code Smells for the Presentation Layer of Android Apps [J]. Empirical Software Engineering,2019,24(6):3546-3586.

[18] MOHA N,GUEHENEUC Y G,DUCHIEN L,et al. DECOR:A Method for the Specification and Detection of Code and Design Smells[J]. IEEE Transactions on Software Engineering,2010,36(1):20-36.

[19] BARBEZ A,KHOMH F,GUÉHÉNEUC Y G. A Machine-learning Based Ensemble Method For Anti-patterns Detection

[J]. Journal of Systems and Software, 2020, 161: 110486.

[20] BARBEZ A, KHOMH F, GUÉHÉNEUC Y G. Deep Learning Anti-patterns from Code Metrics History[C]// IEEE International Conference on Software Maintenance and Evolution (ICSME 19). New York: IEEE, 2019: 114-124.

[21] PALOMBA F, BAVOTA G, PENTA M D, et al. Detecting bad smells in source code using change history information[C]// IEEE/ACM International Conference on Automated Software Engineering (ASE 13). New York: IEEE, 2013, 268-278.

[22] PALOMBA F, BAVOTA G, PENTA M D, et al. Mining Version Histories for Detecting Code Smells[J]. IEEE Transactions on Software Engineering, 2015, 41(5): 462-489.

[23] FONTANA F A, MÄNTYLÄ M V, ZANONI M, et al. Comparing and experimenting machine learning techniques for code smell detection [J]. Empirical Software Engineering, 2016, 21(3): 1143-1191.

[24] FONTANA F A, ZANONI M, MARINO A, et al. Code Smell Detection: Towards a Machine Learning-Based Approach[C]// IEEE International Conference on Software Maintenance (ICSM 13). New York: IEEE, 2013: 396-399.

[25] FONTANA F A, ZANONI M. Code smell severity classification using machine learning techniques[J]. Empirical Software Engineering, 2017, 128: 43-58.

[26] NUCCI D D, PALOMBA F, TAMBURRI D A, et al. Detecting code smells using machine learning techniques: Are we there yet? [C]// IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER 18). New York: IEEE, 2018: 612-621.

[27] STEINBECK M. An arc-based approach for visualization of code smells[C]// IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER 17). New York: IEEE, 2017: 397-401.

[28] MUMTAZ H, BECK F, WEISKOPF D. Detecting Bad Smells in Software Systems with Linked Multivariate Visualizations [C]// IEEE Working Conference on Software Visualization (VISSOFT 18). Los Alamitos: IEEE Computer Society, 2018: 12-20.

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

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

[31] MISBHAUDDIN M, ALSHAYEB M. UML model refactoring: a systematic literature review[J]. Empirical Software Engineering, 2015, 20(1): 206-251.

[32] SENG O, STAMMEL J, BURKHART D. Search-based determination of refactorings for improving the class structure of object-oriented systems[C]// Genetic and Evolutionary Computation Conference (GECCO 06). New York: ASSOC Computing Machinery, 2006: 1909-1916.

[33] 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.

[34] KEBIR S, BORNE I, MESLATI D. A Genetic Algorithm-Based Approach for Automated Refactoring of Component-Based Software[J]. Information & Software Technology, 2018, 88: 17-36.

[35] 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.

[36] ALKHAZI B, ABID C, KESSENTINI M, et al. On the Value of Quality Attributes for Refactoring ATL Model Transformations: A Multi-Objective Approach[J]. Information & Software Technology, 2020, 120: 106243.

[37] PRADITWONG K, HARMAN M, YAO X. Software Module Clustering as a Multi-Objective Search Problem [J]. IEEE Transactions on Software Engineering, 2011, 37(2): 264-282.

[38] KESSENTINI M, DEA T J, OUNI A. A context-based refactoring recommendation approach using simulated annealing: two industrial case studies[C]// Genetic and Evolutionary Computation Conference (GECCO 17). 2017: 1303-1310.

[39] KESSENTINI M, MANSOOR U, WIMMER M, et al. Search-based detection of model level changes[J]. Empirical Software Engineering, 2016, 22(2): 1-46.

[40] AMAL B, KESSENTINI M, BECHIKH S, et al. On the use of machine learning and search-based software engineering for ill-defined fitness function: A case study on software refactoring [C]// International Symposium on Search-Based Software Engineering (SSBSE 14). 2014: 31-45.

[41] DEA T J. Improving the performance of many-objective software refactoring technique using dimensionality reduction[C]// International Symposium on Search-Based Software Engineering (SSBSE 16). 2016: 298-303.

[42] MKAOUER M W, KESSENTINI M, BECHIKH S, et al. On the use of many quality attributes for software refactoring: a many-objective search-based software engineering approach[J]. Empirical Software Engineering, 2016, 21(6): 2503-2545.

[43] MOHAN M, GREER D. Using a Many-Objective Approach to Investigate Automated Refactoring[J]. Information & Software Technology, 2019, 112: 83-101.

[44] BOUKHARATA S, OUNI A, KESSENTINI M, et al. Improving web service interfaces modularity using multi-objective optimization[J]. Automated Software Engineering, 2019, 26(2): 275-312.

[45] GRIFFITH I, WAHL S, IZURIETA C. Evolution of legacy system comprehensibility through automated refactoring [C]// International Workshop on Machine Learning Technologies in Software Engineering (MALETS 11). New York: ACM, 2011: 35-42.

[46] GRIFFITH I, WAHL S, IZURIETA C. TrueRefactor: An automated refactoring tool to improve legacy system and application comprehensibility[C]// International Conference on Computer Applications in Industry and Engineering(CAINE 11). Cary: International Society for Computers and Their Applications, 2011: 316-321.

[47] OUNI A, KESSENTINI M, SAHRAOUI H. Search-Based Refactoring Using Recorded Code Changes[C]// European Con-

ference on Software Maintenance & Reengineering (CSMR 13). New York:IEEE,2013:221-230.

[48] JENSEN A C,CHENG B H C. On the Use of Genetic Programming for Automated Refactoring and the Introduction of Design Patterns[C]// Genetic and Evolutionary Computation Conference (GECCO 10). New York:ACM,2010:1341-1348.

[49] BRAMBILLA M,CABOT J,WIMMER M. Model-Driven Software Engineering in Practice[M]. Morgan & Claypool,2012.

[50] REIMANN J,SEIFERT M,AβMANN U. Role-Based Generic Model Refactoring[C]// International Conference on Model Driven Engineering Languages & Systems (MODELS 10). Berlin:Springer Verlag,2010:78-92.

[51] STEIMANN F. From well-formedness to meaning preservation: model refactoring for almost free[J]. Software and systems modeling,2015,14(1):307-320.

[52] ARCELLI D,CORTELLESSA V,POMPEO D. Performance-driven software model refactoring[J]. Information & Software Technology,2018,95:366-397.

[53] EINARSSON H T,NEUKIRCHEN H. An approach and tool for synchronous refactoring of UML diagrams and models using model-to-model transformations[C]// Proceedings of the Fifth Workshop on Refactoring Tools(WRT 12). New York:ACM,2012:1-8.

[54] ARCELLI D,CORTELLESSA V,POMPEO D D. Automating Performance Antipattern Detection and Software Refactoring in UML Models[C]// IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER 19). New York:IEEE,2019:639-643.

[55] LU H,WANG S,YUE T,et al. Automated Refactoring of OCL Constraints with Search[J]. IEEE Transactions on Software Engineering,2019,45(2):148-170.

[56] BAVOTA G,OLIVETO R,GETHERS M,et al. Methodbook: Recommending move method refactorings via relational topic models[J]. IEEE Transactions on Software Engineering,2014,40(7):671-694.

[57] BAVOTA G,DE LUCIA A,OLIVETO R. Identifying extract class refactoring opportunities using structural and semantic cohesion measures[J]. Journal of Systems and Software,2011,84(3):397-414.

[58] MUMTAZ H,ALSHAYEB M,MAHMOOD S,et al. An empirical study to improve software security through the application of code refactoring[J]. Information & Software Technology,2018,96(4):112-125.

[59] POLITOWSKI C,KHOMH F,ROMANO S,et al. A large scale empirical study of the impact of Spaghetti Code and Blob anti-patterns on program comprehension[J]. Information & Software Technology,2020,122:106278.

[60] WANG Y,YU H,ZHU Z L,et al. Automatic Software Refactoring via Weighted Clustering in Method-Level Networks[J]. IEEE Transactions on Software Engineering,2018,44(3):202-236.

[61] BU Y F,LIU H,LI G J. God Class Detection Approach Based on Deep Learning[J]. Journal ofSoftware,2019,30(5):1359-1374.

[62] LIU H,MA Z,SHAO W,et al. Schedule of Bad Smell Detection and Resolution:A New Way to Save Effort[J]. IEEE Transactions on Software Engineering,2012,38(1):220-235.

[63] LIU H,LIU Q,NIU Z,et al. Dynamic and Automatic Feedback-Based Threshold Adaptation for Code Smell Detection[J]. IEEE Transactions on Software Engineering,2016,42(6):544-558.

[64] OLIVEIRA M C D,FREITAS D,BONIFÁCIO R,et al. Finding needles in a haystack:Leveraging co-change dependencies to recommend refactorings[J]. Journal of Systems and Software,2019,158:110420.

[65] SAS D,AVGERIOU P,FONTANA F A. Investigating instability architectural smells evolution: an exploratory case study [C]// IEEE International Conference on Software Maintenance and Evolution (ICSME 19). New York:IEEE,2019:557-567.

[66] PALOMBA F,NUCCI D D,PANICHELLA A,et al. On the impact of code smells on the energy consumption of mobile applications[J]. Information & Software Technology,2019,105:43-55.

[67] XUAN J F,CORNU B,MARTINEZ M,et al. B-Refactoring: Automatic test code refactoring to improve dynamic analysis [J]. Information & Software Technology,2016,76:65-80.

[68] GAO Y,LIU H,FAN X Z,et al. Analyzing Refactorings' Impact on Regression Test Cases[C]// International Computer Software and Applications Conference (COMPSAC 15). New York:IEEE,2015:222-231.

[69] CHU P H,HSUEH N L,CHEN H H,et al. A test case refactoring approach for pattern-based software development[J]. Software Quality Journal,2012,20(1):43-75.

[70] CHOWDHURY I,ZULKERNINE M. Using complexity,coupling,and cohesion metrics as early indicators of vulnerabilities [J]. Journal of Systems Architecture,2011,57(3):294-313.

[71] KHATCHADOURIAN R,MASUHARA H. Defaultification refactoring:A tool for automatically converting Java methods to default[C]// IEEE/ACM International Conference on Automated Software Engineering(ASE 17). New York:IEEE,2017:984-989.

[72] LIN Y,DIG D. Refactorings for Android Asynchronous Programming[C]// IEEE/ACM International Conference on Automated Software Engineering (ASE 16). New York:IEEE,2016:836-841.

[73] LIN Y,OKUR S,DIG D. Study and Refactoring of Android Asynchronous Programming[C]// IEEE/ACM International Conference on Automated Software Engineering (ASE 15). New York:IEEE,2015:224-235.

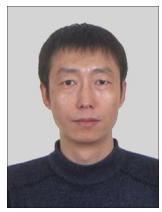
[74] GE X,DUBOSE Q L,MURPHY-HILL E. Reconciling manual and automatic refactoring [C]// International Conference on Software Engineering (ICSE 12). New York:IEEE,2012:211-221.

[75] FOSTER S R,GRISWOLD W G,LERNER S. WitchDoctor: IDE support for real-time auto-completion of refactorings[C]// International Conference on Software Engineering (ICSE 12). New York:IEEE,2012:222-232.

[76] GYORI A,FRANKLIN L,DIG D,et al. Crossing the Gap from Imperative to Functional Programming Through Refactoring

[C]// ACM SIGSOFT Symposium on the Foundation of Software Engineering/European Software Engineering Conference (ESEC/FSE 13). New York: ACM, 2013: 543-553.
[77] SZOKE G, NAGY C, FULOP L J, et al. FaultBuster: An automatic code smell refactoring toolset[C]// IEEE International Working Conference on Source Code Analysis & Manipulation (SCAM 15). New York: IEEE, 2015: 253-258.
[78] HAYASHI S, HOSHINO D, MATSUDA J, et al. Historef: A Tool for Edit History Refactoring[C]// IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER 15). New York: IEEE, 2015: 469-473.
[79] MAZINANIAN D, TSANTALIS N, STEIN R, et al. JDeodorant: Clone Refactoring[C]// International Conference on Software Engineering (ICSE 16). New York: IEEE, 2016: 613-616.
[80] ANTEZANA A S. TOAD: a tool for recommending auto-refactoring alternatives[C]// International Conference on Software Engineering: Companion (ICSE-Companion 19). New York: IEEE, 2019: 174-176.
[81] ALIZADEH V, OUALI M A, KESSENTINI M, et al. RefBot: Intelligent Software Refactoring Bot[C]// IEEE/ACM International Conference on Automated Software Engineering (ASE 19). New York: IEEE, 2019: 823-834.
[82] SILVA D, SILVA J P, SANTOS G J D S, et al. RefDiff 2.0: A Multi-language Refactoring Detection Tool[J]. IEEE Transactions on Software Engineering. DOI: 10.1109/TSE.2020.2968072.
[83] KIM M, GEE M, LOH A, et al. Ref-Finder: A refactoring reconstruction tool based on logic query templates[C]// International Symposium on Foundations of Software Engineering (FSE 10). New York: ACM, 2010: 71-372.
[84] BAVOTA G, GETHERS M, OLIVETO R, et al. Improving software modularization via automated analysis of latent topics and dependencies[J]. Acm Transactions on Software Engineering & Methodology, 2014, 23(1): 1-33.
[85] LIU H, NIU Z, MA Z, et al. Identification of generalization refactoring opportunities[J]. Automated Software Engineering, 2012, 20(1): 81-110.
[86] ZANETTI M S, TESSONE C J, SCHOLTES I, et al. Automated software remodularization based on move refactoring: a complex systems approach[C]// International Conference on Modularity (MODULARITY 14). New York: ACM, 2014: 73-84.
[87] MKAOUER M W, KESSENTINI M, BECHIKH S, et al. Recommendation system for software refactoring using innovization and interactive dynamic optimization[C]// ACM/IEEE international conference on Automated software engineering(ASE 14). New York: IEEE, 2014: 331-336.
[88] KRASNIQI R, CLELAND-HUANG J. Enhancing Source Code Refactoring Detection with Explanations from Commit Messages[C]// International Conference on Software Analysis, Evolution and Reengineering (SANER 20). New York: IEEE, 2020: 512-516.
[89] HU Y, AHMED U Z, MECHTAEV S, et al. Re-Factoring Based Program Repair Applied to Programming Assignments[C]// International Conference on Automated Software Engineering (ASE 19). New York: IEEE, 2019: 388-398.
[90] FENTON N E, PFLEEGER S L. Software Metrics: A Rigorous & Practical Approach[M]. Boston: PWS Publishing Company, 1997.
[91] JABANGWE R, BÖRSTLER J, ŠMITE D, et al. Empirical evidence on the link between: object-oriented measures and external quality attributes: A systematic literature review[J]. Empirical Software Engineering. 2015, 20(3): 640-693.
[92] KANNANGARA S H, WIJAYANAYAKE W M J I. Impact of refactoring on external code quality improvement: An empirical evaluation[C]// International Conference on Advances in ICT for Emerging Regions (ICTer 13). New York: IEEE, 2013: 60-67.
[93] ALSHAYEB M. The Impact of Refactoring to Patterns on Software Quality Attributes[J]. Arabian Journal for ence & Engineering, 2011, 36(7): 1241-1251.
[94] ELISH K O, ALSHAYEB M. A Classification of Refactoring Methods Based on Software Quality Attributes[J]. Arabian Journal for ence & Engineering, 2011, 36(7): 1253-1267.
[95] FENG Q, CAI Y F, KAZMAN R, et al. Active Hotspot: An Issue-Oriented Model to Monitor Software Evolution and Degradation[C]// IEEE/ACM International Conference on Automated Software Engineering (ASE 19). New York: IEEE, 2019: 986-997.
[96] CAI Y, XIAO L, KAZMAN R, et al. Design Rule Spaces: A New Model for Representing and Analyzing Software Architecture [J]. IEEE Transactions on Software Engineering, 2019, 45(7): 657-682.

MENG Fan-yi, born in 1994, Ph.D. Her main research interests include software refactoring, intelligent software development and mining repositories.



YU Hai, born in 1971, Ph.D, associate professor. His main research interests include complex networks, chaotic encryption, software testing, software refactoring and software architecture.