计算机科学
COMPUTER SCIENCE

一种静态分析与知识图谱结合的Java冗余代码检测方法

刘昕炜，陶传奇

引用本文

刘昕炜，陶传奇. 一种静态分析与知识图谱结合的Java冗余代码检测方法[J]. 计算机科学, 2023, 50(3): 65-71.

LIU Xinwei, TAO Chuanqi. Method of Java Redundant Code Detection Based on Static Analysis and Knowledge Graph [J]. Computer Science, 2023, 50(3): 65-71.

相似文章推荐（请使用火狐或 IE 浏览器查看文章）

Similar articles recommended (Please use Firefox or IE to view the article)

基于表示学习的知识图谱推理研究综述

Survey of Knowledge Graph Reasoning Based on Representation Learning

计算机科学, 2023, 50(3): 94-113. https://doi.org/10.11896/jsjkx.220900136

医学知识图谱研究与应用综述

Survey of Medical Knowledge Graph Research and Application

计算机科学, 2023, 50(3): 83-93. https://doi.org/10.11896/jsjkx.220700241

细粒度语义知识图谱增强的中文OOV词嵌入学习

Fine-grained Semantic Knowledge Graph Enhanced Chinese OOV Word Embedding Learning

计算机科学, 2023, 50(3): 72-82. https://doi.org/10.11896/jsjkx.220700249

基于图神经网络的多信息优化实体对齐模型

Multi-information Optimized Entity Alignment Model Based on Graph Neural Network

计算机科学, 2023, 50(3): 34-41. https://doi.org/10.11896/jsjkx.220700242

基于关系约束的上下文感知时态知识图谱补全

Context-aware Temporal Knowledge Graph Completion Based on Relation Constraints

计算机科学, 2023, 50(3): 23-33. https://doi.org/10.11896/jsjkx.220400255

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.220700240

一种静态分析与知识图谱结合的 Java 冗余代码检测方法

刘昕炜[1]　陶传奇[1,2,3,4]
1 南京航空航天大学计算机科学与技术学院　南京 210016
2 高安全系统的软件开发与验证技术工信部重点实验室　南京 210016
3 计算机软件新技术国家重点实验室　南京 210023
4 软件新技术与产业化协同创新中心　南京 210016
(932097753@qq.com)

摘　要　冗余代码普遍存在于商业和开源软件中，它的存在可能会增加内存占用，影响代码可维护性，增加维护成本。快速类型分析算法是当前 Java 冗余代码检测中常用的静态分析方法，该算法在虚方法分析方面还存在一些不足。XTA 是一种调用图构造算法，在处理虚方法的调用方面具有较高的精度和效率。文中提出了一种基于 XTA 调用图构建算法的方法来检测 Java 代码中的冗余代码，在一个名为“RCD”(Redundant Code Detection)的工具原型中实现了这种方法，并通过构建知识图谱辅助人工审查，以提高人工审查的效率以及冗余代码检测的可信度。通过在 4 个开源 Java 应用程序上的实验对 RCD 与其他 3 个冗余代码检测工具进行了比较。实验结果表明，RCD 在检测冗余代码的准确性方面相比其他工具提高了 1%～30%，同时在检测冗余虚方法的完整性方面提升了 4%左右。

关键词：冗余代码检测；调用图构建；静态分析；知识图谱
中图法分类号　TP311

Method of Java Redundant Code Detection Based on Static Analysis and Knowledge Graph

LIU Xinwei[1] and TAO Chuanqi[1,2,3,4]
1 College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China
2 Ministry Key Laboratory for Safety-Critical Software Development and Verification, Nanjing 210016, China
3 State Key Laboratory for Novel Software Technology, Nanjing 210023, China
4 Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing 210016, China

Abstract　Redundant code is common in commercial and open source software, and its presence can increase memory footprint, affect code maintainability, and increase maintenance costs. Rapid type analysis algorithm is a common static analysis method in Java redundant code detection, but it still has some shortcomings in virtual method analysis. XTA is a call graph construction algorithm with high precision and efficiency in handling virtual method calls. A method based on XTA call graph construction algorithm is proposed to detect redundant code in Java code. This method is implemented in a prototype tool called redundant code Detection(RCD), and the knowledge graph is constructed to assist manual review to improve the efficiency of manual review and the reliability of redundant code detection. RCD is compared with three other redundant code detection tools by experiments on four open source Java applications. Experimental results show that RCD improves the accuracy of detecting redundant codes by 1%～30% compared with other tools, and improves the integrity of detecting redundant virtual methods by about 4%.

Keywords　Redundant code detection, Call graph construction, Static analysis, Knowledge graph

1　引言

冗余代码普遍存在于商业和开源软件中。Brown 等[1]在检查一个工业软件系统的代码的过程中发现，开发人员对软件系统中 30%～50%的源代码没有任何理解或记录。Boomsm 等[2]在一个用 PHP 编写的 web 应用程序上的研究发现，开发人员删除了程序中的 2740 个冗余文件，大约占文件总大小的 30%。Romano 等[3]对 Java 应用程序中的死方法进行了研究，结果表明，这些应用中的死方法占比在 5%～10%之间。最后，Yamashita 等[4]提出，专业的软件开发人员希望有一个工具支持对冗余代码进行检测。

冗余代码的存在可能会带来很多问题，它会增加源代码文件的大小，导致过度使用内存，并可能增加软件执行时间。Mäntylä 等[5]表示冗余代码阻碍了开发者对源代码的理解，使其结构不明显，而 Fard 等[6]认为冗余代码影响了软件的可维护性，因为它使源代码更难以理解。Romano 等[7]开展的

到稿日期：2022-07-24　返修日期：2022-12-14
通信作者：陶传奇(taochuanqi@nuaa.edu.cn)

一个多项调查研究表明,冗余代码的存在严重影响了开发人员对不熟悉源代码的理解,也会对不熟悉源代码的修改产生负面影响。此外,开发人员可能会浪费时间维护冗余代码[8],从而导致维护成本更高。当包含错误的冗余代码在不经意间变得可操作时,可能会影响系统的可靠性。

尽管冗余代码的存在是一个普遍现象[1-3],但它可能是有害的[5-8],而且对专业的软件开发人员来说很重要[4]。然而,冗余代码检测很少受到软件工程研究领域的关注,在用 Java 编写的软件系统中,几乎没有工具支持检测冗余代码。

当前静态分析面临的重要问题是,如何在满足程序不断变大的需求的同时保持分析效率和分析精度之间的平衡。上下文敏感分析和流敏感分析是分析精度较高的方法,但分析所需要的成本较高,而且不能很好地满足程序的扩展性,因此无法在实际中得到广泛的应用。而快速类型分析(Rapid Type Analysis,RTA)和 XTA 方法都是上下文不敏感和流不敏感的,算法较简单,实现容易,分析所需要的成本低,能够更好地在分析效率和分析精度之间取得平衡。快速类型分析(RTA)是当前冗余代码检测中常用的调用图构造算法,该算法以 main 方法为程序的起始方法,假设没有方法被调用且没有方法被实例化,那么就将可达方法的集合 R 设为空。首先将起始方法加入可达方法的集合,并从起始方法开始遍历所有的方法。如果遍历到的方法中存在某个对象的实例化信息,则将这个对象加入实例化对象的集合中,若该实例化对象与之前遍历过的方法中存在的任何一个虚方法调用的接受对象的声明类型或子类类型一致,则说明该实例化对象对应的类型在该接受对象的运行时可能类型集合中。通过迭代处理每一个包含实例化对象的方法及其虚方法调用点。尽管 RTA 算法能够检测出大部分的不可达代码(冗余代码),但其在虚方法调用的处理方面还存在一定缺陷,分析精度有待提高。

本文使用 XTA 算法在 Java 应用程序中检测冗余代码。XTA 是一种调用图构造的算法,它可以更好地处理虚拟方法的调用。本文使用 XTA 算法构建调用图,识别所有从起始方法中可达的方法,利用可达的方法来检测冗余方法。本文在一个冗余代码检测工具 RCD 中实现了这种方法,并构建了知识图谱辅助人工审查,在 4 个 Java 应用程序上进行的实验中评估了这个工具和方法的有效性。我们将该工具与 DCF, JTombstone 和 DUM-TOOL 这 3 个冗余代码检测工具进行了比较,结果表明,RCD 在检测冗余代码的准确性方面优于其他工具,同时在检测冗余虚方法的完整性方面有所提升。

本文的主要贡献有:

(1)提出了一种基于 XTA 算法的 Java 冗余代码检测方法,通过构建调用图检测 Java 项目中所有可达方法,并通过可达方法找出不可达的冗余方法。

(2)在冗余代码检测工具 RCD 中实现了所提方法,并通过实体提取、关系提取、属性提取、实体消歧以及知识推理来构建知识图谱辅助人工审查,提高人工审查的效率。

(3)将 RCD 与 3 个冗余代码检测工具在 4 个开源 Java 应用程序上进行了实验比较,实验结果表明,RCD 检测冗余代码的准确性相比其他工具提高了 1%～30%,同时在检测冗余虚方法的完整性方面提升了 4%左右。

本文第 2 节讨论了研究背景和相关工作;第 3 节介绍了这种冗余代码检测方法以及知识图谱的构建;第 4 节重点介绍了实验的设计;第 5 节对实验结果进行了分析;最后总结全文。

2 相关工作

冗余代码检测的相关工作主要分为静态分析和动态分析两个方面。

静态分析方面的工作主要是通过构建抽象语法树以及调用图来检测冗余代码,此外也有基于静态程序切片的冗余代码检测方法。Chen 等[9]提出了一个面向 C++和 C 实体的可达性分析和冗余代码检测方法集。Romano 等[3]提出了面向 Java 应用程序的冗余代码检测方法 DUM,该方法使用调用图表示方法及方法之间的调用关系,通过遍历调用图来检测冗余方法。在调用图中,从起始节点可以到达的节点被认为是活跃的,所有其他节点都被标记为冗余节点。Romano 等[10]在一个名为 DUM- tool 的 Eclipse 插件中实现了 DUM。Wang[11]使用语法树进行词法和语法分析,以检测 C 语言代码中存在的多种冗余代码。Gong 等[12]提出了一种基于控制结构的冗余代码检测模型,使用 TOKEN 序列建立复合语句结构信息表,并检测幂等操作、死代码以及冗余赋值 3 种冗余代码。Gong 等[13]设计了一种大规模代码冗余检测系统 RC-Finder,用于检测幂等、隐幂等、私有变量、条件冗余、冗余代码和冗余传参 6 种冗余代码。Sou 等[14]实现了一个用于冗余代码检测与代码缺陷提示的检测器 NRefactory。Leitao[15]通过构建抽象语法树并分析语法树中的子节点来检测 C 语言代码中的冗余代码。Alabwaini 等[16]使用程序切片技术从程序中提取冗余代码模型,利用分解切片技术提取程序的活动代码并去除冗余代码,以此提高程序代码的质量。Scanniello[17]定义了一种基于 Kaplan Meier 评估器的方法来分析冗余代码如何对 5 个正在开发中的软件系统产生影响。随后,Scanniello[18]对 13 个软件指标进行了调查,结果表明 13 个软件指标中有 5 个可以作为冗余代码的预测指标,其中 LOC(代码行)是最好的冗余代码预测指标,当类越大时,其方法是冗余代码的可能性就越大。针对当前静态分析方法存在的分析精度不够的问题,本文对现有静态分析方法进行了改进,提高了 Java 冗余代码检测的精度。

动态分析方面的工作主要是与静态分析相结合,通过追踪程序执行路径或分析动态切片来检测冗余代码。Boomsma 等[2]提出了一种动态方法来识别并删除用 PHP 编写的 web 系统中的冗余文件。这种方法在给定的时间范围内监控 web 系统的执行,收集有关 PHP 文件使用情况的数据。如果文件在一定的时间范围内没有被使用,则将其标记为冗余;Fard 等[6]提出了一种用于检测 web 系统坏味的方法 JSNOSE,该方法将静态分析和动态分析相结合来检测客户端代码中包括冗余代码在内的 13 种坏味,并通过计算语句的执行或可达性来检测冗余代码。Wang 等[19]结合静态切片和动态切片分析方法,提出了一种基于程序切片的冗余代码检测方法,并在 LLVM 基础上设计了冗余代码检测框架。Obbink 等[20]针对 Web 中代码复用带来的浏览器解析多余 JavaScript 冗余代码

而造成 Web 应用系统的整体性能下降的问题，提出了一种基于静态和动态分析的 JavaScript 冗余代码消除方法 Lacuna。尽管动态分析的精度高于静态分析，但是其要求程序可以执行，时间复杂度较高且泛用性较低，不适用于大型软件的分析。因此本文使用静态分析的方法对冗余代码进行检测。

3 基于静态分析与知识图谱的 Java 冗余代码检测方法

3.1 冗余代码检测方法

本文提出的冗余代码检测方法主要基于 XTA 算法构建调用图。XTA 算法[21] 是一种虚方法调用解决策略，它对快速类型分析算法进行了改进。该算法处于流敏感分析 0-CFA 算法和快速类型分析算法之间的一个中间状态，既能得到比快速类型分析算法更高的精度，同时算法又没有 0-CFA 方法复杂，能够很好地满足实用性和程序扩展性的要求。

XTA 算法采取增量式的方式考虑各种方法的可达性，首先将 main 方法作为起始方法，并加入可达方法的集合 R 中，然后从起始方法开始遍历所有的方法，对每个可达方法内的虚方法调用点进行分析。XTA 算法分析了所有可达方法的可用类型集合，并且基于可用类型集合在各可达方法间的类型传播，对可达方法内的每个虚方法调用点中的接受对象的运行时可能类型集合进行约减，每个可达方法的可用类型集合包括：1)在该方法中实例化的对象类型；2)通过参数传递从调用该方法的可达方法中传进来的可用类型。

对于被调用方法存在返回类型的情况，在调用该方法的可达方法的可用类型集合中加入该方法的返回值类型。当该返回值类型对调用该方法的可达方法中的某个虚方法调用点中接受对象的可能类型集合产生影响时，需要重新对该虚方法调用点进行分析，重新确定该虚方法调用点的可能的目标方法。因此，XTA 算法对虚方法调用点的处理是一个迭代的过程，当分析达到某个固定点时停止，这与快速类型分析算法有所不同。

我们用变量 R 表示可达方法的集合，用 $StaticType(e)$ 表示 e 的声明类型，用 $SubTypes(t)$ 表示类型 t 的所有子类的集合，并且用 $StaticLookup(C,m)$ 表示在类 C 中静态查找其是否有声明为 m 的方法。对项目中的每种方法 M 给定一个集合 S_M，对每个域 x 给定一个集合 S_X，用 $ParamTypes(M)$ 表示方法 M 的参数的声明类型的集合，用 $ReturnType(M)$ 表示方法的返回类型。定义方法 $SubTypes()$ 满足：

$$SubTypes(M)=\bigcup_{m\in M} SubTypes(m) \tag{1}$$

算法 1 XTA 算法

输入：待测项目的启动方法 main

输出：所有可达方法的集合 R

1. main∈R(main 表示程序中的 main 方法)
2. For each method M, each virtual call site e.m() occurring in M, and each class C ∈ SubTypes (StaticType (e)) where StaticLookup(C, m) = M′:
   $(M\in R)\wedge(C\in S_M)\Rightarrow M'\in R\wedge SubTypes(ParamTypes(M'))\cap S_M\subseteq S_{M'}\wedge SubTypes(ReturnTypes(M'))\cap S_{M'}\subseteq S_M\wedge C\in S_{M'}$
3. For each method M, and for each “new C()” occurring in M:
   $(M\in R)\Rightarrow(C\in S_M)$
4. For each method M in which a read of a field x occurs:
   $(M\in R)\Rightarrow(S_X\subseteq S_M)$
5. For each method M in which a write of a field x occurs:
   $(M\in R)\Rightarrow(SubTypes(StaticTypes(x))\cap S_M)\subseteq S_X$

步骤 2 表示：以可达方法 M 内的虚方法调用点为例，如果某个包含目标方法的类属于该可达方法的可用类型集合 S_M，则将这一类目标方法加入可达方法的集合 R 中。同时，将调用该方法的可用类型集合与被调用方法的形参声明类型及其子集的类型集合的交集，以参数传递的方式传递给被调用方法，作为其可用类型集合的子集。当被调用方法有返回值且返回类型不为空时，可以将其与被调用方法的可用类型集合的交集作为子集传递给调用方法的可用类型集合。在调用方法中的实例化对象类型也属于被调用方法的可用类型集合。

步骤 3 表示：将可达方法 M 的可用类型集合 S_M 初始化为在可达方法 M 中实例化的对象类型。

步骤 4 表示：对于可达方法 M 内的某个域 x，如果存在对域 x 的读取，那么变量 x 可能指向的对象类型集合是该可达方法的可用类型集合的子集。

步骤 5 表示：对于可达方法内的某个域 x，如果存在对域 x 的写入，那么变量 x 可能指向的对象类型集合包含域 x 的声明类型及其子类的类型集合与可达方法 M 的可用类型集合的交集。

相比其他静态类型分析算法对程序中所有可达方法给定一个实例化对象的集合，XTA 算法对每种方法都给定了一个特定的可用类型的集合。此外，XTA 算法还考虑了在可达方法之间由于参数传递和方法返回值引起的实例化类型的传播，以及可达方法内声明的域可能的指向类型集合和可达方法本身的可用类型集合之间的类型的传播。因此，XTA 算法能提高静态分析的精度。

本文使用的冗余代码检测方法主要包括以下几个阶段。

(1)识别启动方法。在这个阶段中，我们主要在软件系统中识别启动方法。启动方法又分为 main 方法和反射调用的方法。

(2)构建调用图。对于每个起始方法 s_i，使用 XTA 算法构建调用图 $G_i=(V_i,E_i)$。V_i 是节点的集合，即方法；而 E_i 是边的集合，即方法调用。在构建调用图时，XTA 算法不能很好地处理线程的 *run*()方法的隐式调用。我们使用一个单独的方法来帮助 *XTA* 处理这些隐式调用。它还可以处理 *initializers*，*finalizers* 和 *AccessController* 等隐式调用。

(3)检测所有可达方法。内部类位于正在分析的软件系统的代码库中，外部类位于软件系统的依赖项中(库或框架)。$V=V_1\cup\cdots\cup V_i\cup\cdots\cup V_n$ 是与内部类和外部类中所有的可达方法相对应的节点集。本文的检测对象是内部类中的冗余代码，因此不考虑外部类中的方法。*Internal* 是在内部类中定义的方法，我们将检测可达方法 *Reachable*，如式(2)所示：

$$Reachable=V\cap Internal \tag{2}$$

(4)识别冗余方法。在这一阶段检测所有不可达的冗余方法。内部类中定义的冗余代码集合 *Redundant* 的计算式如下：

$$Redundant = Internal \backslash Reachable \quad (3)$$

3.2 知识图谱构建

知识图谱是一种大规模语义网络，表达了实体及其之间的各种语义关系。构建知识图谱辅助人工审查不仅可以提供更全面的冗余代码相关信息，还能改进人工审查的质量并提高效率。本文的知识图谱构建主要分为实体提取、关系提取、属性提取、实体消歧以及知识推理 5 个部分。我们选择知识图谱的原因主要有以下 3 点：

(1)知识图谱可以直观、有效地表达出实体间的关系，并提供了从关系的角度分析问题的能力。

(2)冗余代码检测工具的检测结果可能存在误差。

(3)冗余代码具有特殊性，不可达的代码不一定是无用的，需要人工进行判断。

3.2.1 实体提取

实体提取指从文本数据集中自动识别命名实体，是信息提取中最基本、最关键的部分。传统的知识图谱数据大多来自百科全书站点和垂直站点的结构化数据，从各种半结构化数据中获取相关实体的属性值，以扩展属性描述[22]。实体是知识图谱构建的基础，因此，有必要尽可能选择具有代表性的信息[23]。

本文构建的知识图谱与传统的知识图谱略有不同，我们将项目中的方法作为知识图谱中的实体。本文的目标是对冗余代码进行审查，而在 XTA 构建的调用图中不包含冗余代码，因此需要提取冗余代码加入已有的调用图中。在此步骤中，根据冗余代码检测的结果识别和筛选冗余方法，并将方法名作为实体提取到知识图谱中。

3.2.2 关系提取

在获得一系列离散命名实体后，为了获得语义信息，需要从语料库中提取实体之间的关系，并通过关系链接实体，形成知识结构网络[24]。

本文中的知识图谱以 XTA 算法构建的调用关系图为基础。由于调用关系图$G_i=(V_i, E_i)$中的节点V_i是方法，边E_i是方法调用，因此知识图谱中连接实体之间的关系主要为调用关系。对于前一步中提取的冗余方法实体，自动定位其所在类的位置，提取冗余方法实体与所在类中的其他方法，将冗余方法实体以关联关系连接到所在类中的其他方法上。对于与其他冗余方法之间有调用关系的冗余方法，还将提取其调用关系作为连接实体之间的关系。

3.2.3 属性提取

本文构建的知识图谱是从提高代码审查效率的角度出发的，因此实体属性的选择也应考虑是否有助于代码审查。为了提高审查效率，我们将各个实体的一些信息作为属性提取。在这里，提取属性的实体是冗余方法和活跃方法。冗余方法的实体属性是所在类的类名、有调用关系的冗余方法数以及代码行数。由于冗余代码检测工具可能存在误判的情况，当冗余方法调用的方法数或代码行数较多时，可以通过所在类的类名快速找到冗余方法的位置。而活跃方法的实体属性只有所在类的类名，该属性可以在代码审查时快速发现与冗余方法相关联的方法实体，提高代码审查的效率。

3.2.4 实体消歧

实体消歧是一种专门用于解决同名实体歧义问题的技术。在真实的语言环境中，经常会遇到一个实体引用项对应多个命名实体对象的问题。例如，术语“李娜”可以对应作为歌手或网球运动员的李娜的实体。通过实体消歧，我们可以根据当前上下文准确地建立实体链接[25]。

本文中不同的方法实体可能有相同的方法名，而相同的方法名可能会对代码审查产生误导，影响代码审查的效率。为了防止同名的冗余代码造成混乱，将同名冗余代码所在类的信息加入实体名中，例如 A 类中的 b 方法，方法名可设置为 $A.b$。对于同一类中的同名方法，向重复方法的末尾中加入随机数。通过更改重复的实体名来消除实体歧义，在区分同名实体的同时方便了审查人员的理解。

3.2.5 知识推理

知识图中常用的语义关系包括概念之间的继承关系、整体部分关系和领域特定的语义关系[26]。根据先前建立的实体和实体之间的关系，本文基于知识图中整体和部分之间的关系进行知识推理，并挖掘其中的知识。

在本文构建的知识图谱中，每个冗余方法实体和活跃方法实体都有调用关系或关联关系。当一个冗余方法实体与多个冗余方法实体之间存在调用关系时，其中很可能存在被冗余代码检测工具误判为冗余方法的活跃方法，而依靠知识图谱可以通过所在类的类名快速找到冗余方法的所在位置，提高人工审查的效率。同理，当同一个类中有大量冗余方法或冗余方法行数较大时，也可能存在误判的情况，需要对冗余方法实体进行代码审查。

4 实验设计

4.1 研究目标

本文的实验目标为：从开源 Java 应用程序的研究人员和开发人员的角度，分析不同的冗余代码检测工具，以评估冗余代码检测工具的正确性、完整性和准确性。

根据研究目标，本文制定了以下研究问题：

RQ1 我们的冗余代码检测工具在检测冗余代码的准确性方面是否优于其他工具？

RQ2 我们的冗余代码检测工具在检测冗余虚方法的完整性方面是否有所提升？

4.2 实验对象

本文选取了 4 个开源 Java 应用程序，利用 RCD 和其他冗余代码检测工具自动化检测冗余代码。表 1 列出了 Java 应用程序的基本信息，这些应用程序在大小、应用程序域和用户接口方面各不相同，且它们的源代码可以在 web 上获得。一些实验对象在之前的研究中也使用过[3,10]。

我们选择的冗余代码检测工具主要有 DCF，DUM-Tool[10] 和 JTombstone。对这几种工具的简要描述如下。

(1)JTombstone

一个基于命令行界面的工具，用于检测 Java 软件系统中的冗余方法。JTombstone 主要分析具体的方法。

(2)DUM-Tool

一个 Eclipse 插件，用于检测 Java 应用程序中的冗余

方法，它支持检测具体的和抽象的冗余方法。

(3)DCF

一个基于命令行界面的工具，使用一种依赖于 RTA 调用图构造的算法，通过 RTA 构建的调用图中识别活跃的方法，利用可运行的方法检测冗余代码。

表 1 Java 应用程序的基本信息

Table 1 Basic information of Java application

Application	LOC	Class	Methods	redundant method/%
LaTeXDraw	65 320	252	3 130	8
aTunes	42 357	778	4 067	7
MediaPesata	1 580	31	162	5
LaTazza	1 291	18	116	40

4.3 自变量和因变量

实验中的自变量是工具，自变量的值为：RCD，DCF，DUM-Tool 和 JTombstone。

在实验中，我们对冗余代码检测的正确性、完整性和准确性进行了评估。

正确性：反映了被判定为冗余的方法确实是冗余的。

完整性：反映了检测到的冗余方法集相对于所有的冗余方法而言是完整的。

准确性：反映了正确性和完整性。

为了评估这些结构，本文使用了基于信息检索的策略：使用 precison 来评估正确性，用 recall 评估完整性。

$$precision=\frac{|TP|}{|TP|+|FP|} \tag{4}$$

$$recall=\frac{|TP|}{|TP|+|FN|} \tag{5}$$

TP(True Positives)是工具正确检测为冗余的方法，集合 FN(False Negatives)是工具无法检测到的冗余方法，而 FP (False Positives)是工具错误检测为冗余的方法。precison 和 recall 为区间[0,1]内的值，其中 1 是最佳值。precison 值越高，冗余方法检测越正确。recall 值越高，冗余方法检测就越完整。

为了评估准确性，本文采用了 precison 和 recall 的平衡值 F-measure。

$$F\text{-}measure=\frac{2*precision*recall}{precison+recall} \tag{6}$$

它代表了结果的正确性和完整性之间的权衡。F-measure 值为区间[0,1]中的值，其中 1 是最佳值。F-measure 的值越高，冗余方法检测越准确。

综上所述，实验的自变量是工具，因变量是 precison，recall 和 F-measure。

5 实验结果与分析

5.1 实验结果

表 2 列出了 Precision，Recall 和 F-measure 的平均值和标准差的值。在平均值方面，RCD 的 Precision(0.79)和 F-measure(0.83)均优于其他冗余代码检测工具，这说明 RCD 检测结果在正确性、完整性和准确性方面的平均表现是相对较好的。其中，RCD 在正确性方面的表现优于其他工具。而在完整性方面，RCD，DCF 和 DUM-Tool 的检测结果相近，Recall 分别为 0.89，0.87 和 0.88，而 JTombstone 的 Recall 为 0.94。尽管 JTombstone 的完整性明显优于 RCD，但正确性远低于其他工具。在准确性方面，RCD 在大部分情况下优于其他工具。同时可以发现，RCD 标准差的值低于其他工具：Precision 为 0.15，Recall 为 0.07，F-measure 为 0.11。这表明当实验对象发生变化时，RCD 检测到的冗余方法的正确性、完整性和准确性的变化小于其他工具。

表 2 冗余代码检测结果的平均值与标准差

Table 2 Mean and standard deviation of redundant code detection results

Statistic	Variable	RCD	DCF	DUM-Tool	JTombstone
Mean	Precision	0.79	0.78	0.72	0.45
	Recall	0.89	0.87	0.88	0.94
	F-measure	0.83	0.82	0.76	0.48
SD	Precision	0.15	0.16	0.26	0.39
	Recall	0.07	0.08	0.10	0.08
	F-measure	0.11	0.12	0.18	0.35

表 3 列出了每个实验对象的正确性、完整性和准确性的数据。其中，RCD 的性能全面优于 DCF。在大部分的情况下，RCD 的 Precision 最高。尽管 JTombstone 在 Recall 方面的表现很好，但是其 Precision 非常低。而在 F-measure 方面，RCD 在 aTunes 上的表现优于其他工具。在 LaTeXDraw 上，表现最好的方法是 DUM-Tool，但 RCD 仍优于其他工具。在 LaTazza 上，RCD(F-measure 为 0.97)略差于 DUM-Tool(F-measure 为 0.99)和 JTombstone(F-measure 为 1)。

表 3 各个冗余代码检测工具的实验结果

Table 3 Experimental results of each redundant code detection tool

Statistic	Variable	RCD	DCF	DUM-Tool	JTombstone
LaTeXDraw	Precision	0.63	0.61	0.77	0.24
	Recall	0.78	0.75	0.74	0.83
	F-measure	0.70	0.67	0.76	0.38
aTunes	Precision	0.74	0.73	0.38	0.11
	Recall	0.93	0.92	0.93	0.99
	F-measure	0.82	0.82	0.54	0.20
LaTazza	Precision	1	1	1	1
	Recall	0.95	0.93	0.98	1
	F-measure	0.97	0.96	0.99	1
MediaPesata	Precision	1	1	0.88	0.07
	Recall	0.91	0.88	0.88	1
	F-measure	0.95	0.93	0.88	0.13

表 4 列出了对冗余虚方法的检测结果。本文主要对 RCD 和 DCF 的虚方法检测能力进行了比较，DUM-Tool 和 JTombstone 因为不考虑虚方法调用，所以不参与比较。从结果可以看出，使用 XTA 算法的 RCD 工具比使用快速类型分析算法的 DCF 工具在检测冗余的虚方法方面的表现更好。

表 4 冗余虚方法检测结果

Table 4 Detection results of redundant virtual method

Application	Methods	Redundant method/%	Virtual redundant method detected by RCD	Virtual redundant method detected by DCF
LaTeXDraw	3 130	8	129	124
aTunes	4 067	7	184	177
MediaPesata	162	5	2	2
LaTazza	116	40	18	18

5.2 实验结果分析

从实验结果中可以看出，RCD 检测到的冗余方法的正确

性较高,这说明 RCD 错误检测为冗余的方法数量较少。从开发人员的角度来看,这有助于开发人员对应用程序进行重构。如果开发人员删除了被误检为冗余的方法,并且重构后的应用程序在语法上是正确的,那么应用程序的行为可能与其预期行为发生偏差,引入新的 Bug。因此,错误检测的冗余方法数量越少,改变应用程序行为的风险就越小。在本实验中的比较工具中,RCD 的正确性最高(平均 Precision 为 0.79),因此 RCD 在正确性方面的表现优于其他工具。

在检测冗余代码的完整性方面,RCD 的表现与 DCF 和 DUM-Tool 相近,而 JTombstone 的完整性优于 RCD。从开发人员的角度来看,RCD,DCF 和 DUM-Tool 检测到的真实冗余的方法数量基本相同。尽管 JTombstone 能检测到更多的真实冗余方法,但它检测的冗余方法的正确性偏低。对于需要检测冗余代码的开发人员来说,正确性的重要性大于完整性,因此 JTombstone 不是一个很好的冗余代码检测工具。

准确性代表正确性和完整性之间的平衡。RCD 的高准确性说明其相对于其他的工具拥有更高的正确性和完整性,更适合开发人员使用。

尽管 RCD 的检测精度高于其他工具,但其所需的运行时间也有所增加。例如,在 aTunes 上的执行时间如下:RCD 为 9 min,DCF 为 3 min,DUM-tool 为 58 min,JTombstone 为 1 min。在运行时间方面 RCD 可能稍逊于 DCF,但优于 DUM-tool(时间长)和 JTombstone(精度低)。

综上所述,我们可以回答本文的研究问题:RCD 检测到的冗余代码在准确性方面比其他工具提升了 1%~30%,同时在检测冗余虚方法的完整性方面提升了 4%左右。

5.3 有效性威胁

尽管我们尽可能地减少了对有效性的威胁,但有些威胁是不可避免的。

(1)内部有效性:这种有效性涉及到实验内部可能影响结果的因素。本文方法的实现可能会威胁到结果的有效性。此外,冗余代码检测工具的源代码可能包含 bug,因此影响结果。

(2)外部有效性:主要考虑的是实验结果被推广的可能性。使用的实验对象的数量(4 个)可能会影响到开源 Java 应用程序领域中结果的普遍性。此外,我们不能保证所得的结论可以推广到其他类型的应用程序或用其他编程语言编写的软件系统中。

(3)结论有效性:这种有效性与结果有关。在我们的实验中需要判断一个冗余方法是否真的冗余,该判断的准确性可能会影响我们从结果中得出正确结论。如果一些方法被错误地标记为冗余方法,这将影响对 RCD 和其他工具的评估。此外,我们用于评估正确性、完整性和准确性的因变量可能也会影响结果。

结束语 本文提出了一种基于 XTA 的方法来检测 Java 应用程序中的冗余代码。我们在一个名为 RCD 的工具中实现了这种方法,并通过构建知识图谱来辅助人工审查。在 4 个开源 Java 应用程序上与其他 3 个冗余代码检测工具(DCF,DUM-Tool 和 JTombstone)进行了实验,对这些方法和 RCD 进行了实验评估。结果表明,RCD 在正确性和准确性方面优于其他工具,同时检测到的冗余方法也有相对较高的完整性。在未来工作中,我们将进一步提高冗余代码检测工具的准确性。同时,我们还考虑对冗余代码进行评估,让开发者知道何时删除冗余代码的风险较低,以及是否通过删除冗余代码改进了源代码的可理解性。

参考文献

[1] BROWN W J,MALVEAU R C,MCCORMICK H W,et al. AntiPatterns:Refactoring Software, Architectures, and Projects in Crisis[M]. New York:John Wiley & Sons,Inc. ,1998.

[2] BOOMSMA H,GROSS H G. Dead code elimination for web systems written in PHP:Lessons learned from an industry case [C]// IEEE International Conference on Software Maintenance. IEEE,2013:511-515.

[3] ROMANO S,SCANNIELLO G,SARTIANI C,et al. A graph-based approach to detect unreachabl-e methods in Java software [C]// Acm Symposium on Applied Computing. ACM,2016:1538-1541.

[4] YAMASHITA A,MOONEN L. Do developers car-e about code smells? An exploratory survey[C]// Reverse Engineering. IEEE,2013:242-251.

[5] MÄNTYLÄ M,VANHANEN J,LASSENIUS C. A Taxonomy and an Initial Empirical Study of Bad Smells in Code[C]// International Conference on Software Maintenance. IEEE,2003:381-384.

[6] FARD A M,MESBAH A. JSNOSE:Detecting JavaScript Code Smells[C]// IEEE International Working Conference on Source Code Analysis & Manipulation. IEEE Computer Society,2013:116-125.

[7] ROMANO S,VENDOME C,SCANNIELLO G,et al. A Multi-study Investigation Into Dead Code[J]. IEEE Transactions on Software Engineering,2018,46(1):71-99.

[8] EDER S,JUNKER M,JURGENS E,et al. How much does unused code matter for maintenance? [C]// 2012 34th International Conference on Software Engineering(ICSE). Switzerland:IEEE,2012:1102-1111.

[9] CHEN K R,VÁCLAV R. Case study of feature location using dependence graph[C]// 8th International Workshop on Program Comprehension(IWPC). Limerick:IEEE,2000:241-247.

[10] ROMANO S,SCANNIELLO G. DUM-Tool[C]// 2015 IEEE International Conference on Software Maintenance and Evolution. Bremen:IEEE,2015:339-341.

[11] WANG W. Research on C redundant code and related defect detection methods[D]. Harbin: Harbin Institute of technology,2010.

[12] GONG D D,WANG T T,SU X H,et al. Redund-ant code defect detection method[J]. Journal of Harbin Institute of technology,2012,44(7):58-63.

[13] GONG D,WANG T,SU X,et al. RCfinder:redundancy detection for largescale source code[C]// 2012 second International Conference on Instrumentation,Measurement,Computer,Communication and Control. Harbin:IEEE,2012:243-248.

[14] SHOU N,ZHAO F Y. Research on redundancy detection and

defect based on nrefactory[J]. Small microcomputer system, 2015, 36(9): 1973-1976.

[15] LEITAO A M. Detection of redundant code using R2D2[J]. Software Quality Journal, 2004, 12(4): 361-382.

[16] ALABWAINI N, ALDAAJE A, JABER T. Using Program Slicing to Detect the Dead Code[C]// 2018 8th International Conference on Computer Science and Information Technology. 2018: 230-233.

[17] SCANNIELLO G. Source code survival with the Kaplan Meier [C]// IEEE International Conference on Software Maintenance. IEEE, 2011: 524-527.

[18] SCANNIELLO G. An Investigation of Object-Oriented and Code-Size Metrics as Dead Code Predictors[C]// IEEE Computer Society. 2014: 392-397.

[19] WANG X, ZHANG Y, ZHAO L, et al. Dead code detection method based on program slicing[C]// 2017 International Conference on Cyber-enabled Distributed Computing and Knowledge Discovery(CyberC). Guilin: IEEE, 2017.

[20] OBBINK N G, MALAVOLTA I, LUCA G, et al. An extensible approach for taming the challenges of JavaScript dead code elimination[C] // IEEE International Conference on Software Analysis, Evolution and Reengineering. Antwerp, Belgium: IEEE, 2018: 291-401.

[21] TIP F, PALSBERG J. Scalable propagation-based call graph construction algorithms[C]// Proceedings of the 2000 ACM Conference on Object-Oriented Programming Systems, Languages and Applications. Minnesota: ACM, 2000: 281-293.

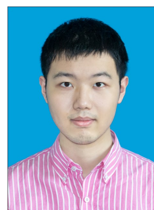
[22] LIN Z Q, XIE B, ZOU Y Z, et al. IntelligentDevelopment Environment and Software Kno-wledge Graph[J]. Journal of Computer Science and Technology, 2017, 32(2): 242-249.

[23] LIU Q, LI Y, DUAN H. Knowledge graph costruction techniques[J]. Journal of Computer Research and Development, 2016, 32(2): 242-249.

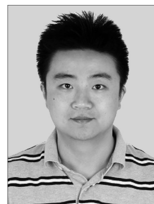
[24] ZHANG X, LIU X, LI X, et al. An approach to generate metallic materials knowledge graph based on DBpedia and Wikipedia[J]. Computer Physics Communications, 2017, 211(1): 98-112.

[25] SUN X B, WANG L, WANG J W, et al. Construct Knowledge Graph for Exploratory Bug Issue Searching[J]. Acta Electronica Sinica, 2018, 46(7): 1578-1583.

[26] LIN X, LIANG Y, GIUNCHIGLIA F, et al. Relation path embedding in knowledge graphs[J]. Neural Computing and Applications, 2018, 31(9): 5629-5639.



LIU Xinwei, born in 1995, postgraduate. His main research interests include redundant code detection and so on.



TAO Chuanqi, Ph. D, associate professor. His mian research interests include intelligent software testing, regression testing, cloud-based mobile testing as a service, and quality assurance for big data applications.

（责任编辑：喻藜）