计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.190500148

代码相似性检测方法与工具综述

张 丹 罗 平
清华大学软件学院 北京 100084
信息系统安全教育部重点实验室(清华大学) 北京 100084
(zd18@mails.tsinghua.edu.cn)

摘 要 在代码开源的潮流下,代码克隆在提高代码质量和降低开发成本的同时,一定程度地影响了软件系统的稳定性、健壮性与可维护性。代码相似性检测在计算机与信息安全发展方面具有重要的意义。为应对代码克隆带来的各种危害,目前学术界和工业界提出了很多代码相似性检测的方法,这些方法按照源代码信息处理程度可分为基于文本、词法、语法、语义和度量值5类;并开发了相应的检测工具,这些工具实现了很好的检测效果,但在大数据时代背景下也面临着数据规模不断扩大带来的一系列挑战。文中综述了代码相似性检测的方法,对5类检测方法做了详细比较;结合传统方法与机器学习技术,归类了不同检测方法对应的检测工具;按照不同评价标准评估了检测工具的检测效果,总结了每种检测方法的首选检测工具,并对未来代码相似性检测的研究方向做出了展望。

关键词:代码克隆;克隆检测;克隆评估
中图法分类号 TP311

Survey of Code Similarity Detection Methods and Tools

ZHANG Dan and LUO Ping
School of Software,Tsinghua University,Beijing 100084,China
Key Laboratory of Information System Security(Tsinghua University),Ministry of Education,Beijing 100084,China

Abstract Source code opening has become a new trend in the information technology field. While code cloning improves code quality and reduces software development cost to some extent,it also affects the stability,robustness and maintainability of a software system. Therefore,code similarity detection plays an important role in the development of computer and information security. To overcome the various hazards brought by code cloning,many code similarity detection methods and corresponding tools have been developed by academic and industrial circles. According to the manner of processing source code,these detection methods could be roughly divided into five categories:text analysis based,lexical analysis based,grammar analysis based,semantics analysis based and metrics based. These detection tools can provide good detection performance in many application scenarios,but are also facing a series of challenges brought by ever-increasing data in this big data era. This paper firstly introduced code cloning problem and made a detailed comparison between code similarity detection methods divided into five categories. Then,it classified and organized currently available code similarity detection tools. Finally,it comprehensively evaluated the detection performance of detection tools based on various evaluation criteria. Furthermore,the future research direction of code similarity detection was prospected.

Keywords Code clone,Clone detection,Clone evaluation

1 代码克隆

1.1 代码克隆的背景和概念

随着互联网的不断发展和软件开源精神的发扬,代码开源成为了一种潮流。软件开发人员在开发过程中可以快速地在开源代码库(如GitHub、SourceForge、码云、开源中国等)中搜索到相关项目的源代码,通过复制、粘贴、修改等操作复用已有代码,大大加快了当前项目的开发工作。

复制源代码后,软件系统中某一文件的代码与其他文件中的代码相同或相似,这种现象被称为代码克隆[1]。当且仅当两个代码段是相似序列时,它们之间存在克隆关系,这一对代码段被称作克隆对,由克隆对组成的集合称为克隆类[2-4]。

到稿日期:2019-05-27 返修日期:2019-10-01 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:国家重点研发计划项目(2018YFF0215901)
This work was supported by National Key R&D Program of China (2018YFF0215901).
通信作者:罗平(luop@mail.tsinghua.edu.cn)

1.2 代码克隆的分类

代码克隆可以按照克隆程度分为以下 5 类[5]。

1)完全克隆:完全复用整个代码文件;

2)完整克隆:在完全克隆的基础上,完全复用除空格、注释外的代码;

3)重命名克隆:在完整克隆的基础上,对代码片段的变量名、常量名、类型名等做修改,保持代码结构,继续复用代码片段;

4)增删改克隆:在重命名克隆的基础上,对代码中的程序语句做增加、修改、删除等操作,基本保持代码文本内容,继续复用代码片段;

5)自实现克隆:与源代码相比,程序功能相同,但是实现方式不同。

1.3 代码克隆的优缺点

代码克隆具有如下优点:

1)代码质量高。在知识分享、代码开源的潮流下,越来越多的开发人员参与到开源软件的开发中,不断地优化开源软件,及时修复软件缺陷,使得开源软件的质量和稳定性日益得到提高。

2)开发成本低。在软件开发过程中,需求分析、系统与详细设计、编码与测试等阶段都需要大量的人力、物力及财力。进行代码克隆后,可以提高开发效率,大大降低开发成本。

已有研究指出代码克隆会给开发者和软件带来危害[6-7]。具体来说,代码克隆会带来以下问题:

1)开发项目的额外成本升高。在开源项目中,专业性强、注释不完整等原因,使得开发人员需要花费更多的时间去理解代码,从而增加了额外的开发成本。另外,开源项目代码量大,在代码克隆时复用较多的关联代码会增大开发结束后的总代码量,从而导致编译时间增长,机器内存需求增大。

2)开发软件易存在漏洞风险。在开源项目中,软件代码虽然具有很高的质量和稳定性,但也存在还未被发现的潜在漏洞,而这些漏洞会增加系统风险,降低系统的安全性。

3)可能侵犯开源软件的著作权。在使用开源项目时,需要遵循开源许可协议(GPL,BSD,Apache License 等),如使用 GitHub 代码时需要注意版权问题。在代码克隆时,如果违规使用开源代码,则可能会侵犯软件著作权并需要承担相应的后果。

2 代码相似性检测方法

由于在代码克隆过程中存在很多风险,因此进行代码相似性检测是非常必要的。

2.1 固定粒度和自由粒度

在对代码段进行相似性检测之前,首先需要将代码切分成一定大小的基本代码单元(检测粒度)。检测粒度分为固定粒度和自由粒度[8]。对于固定粒度,如果粒度太大,则检测漏检率会很高;如果粒度太小,则检测工作量会较大。因此,为实现更好的检测效果,在实际应用中基本采用自由粒度。

2.2 代码相似性检测流程

目前,代码相似性检测的基本流程是相同的,如图 1 所示。Whale[9]首次提出检测过程的两个阶段:转换代码格式和确定相似度。首先,对源代码做一些预处理,以减少检测工作量。其次,利用不同的检测方法对代码做中间表现形式的转换,将其转换成 Token 序列、抽象语法树(Abstract Syntax Tree,AST)、程序依赖图(Program Dependence Graph,PDG)等结构。然后,在转换后的新结构上,使用不同的匹配算法实现相似性检测,这一过程可以得到克隆对。最后,整理检测结果,将其还原到源文件中,可视化展示克隆信息。

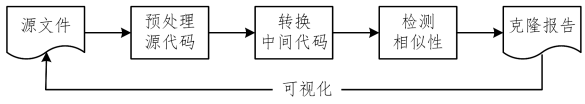

图 1 代码相似性检测的基本流程

Fig. 1 Basic process of code similarity detection

2.3 代码相似性检测方法的分类

目前,代码相似性检测方法可以大致划分为以下 5 类。

1)基于文本的检测方法

基于文本来实现检测是最早的检测代码相似性的技术。首先,预处理代码段,如除去空格、注释等;接着,将代码段转换成字符,如果两个代码段的字符相同,则两段代码相同。基于文本的检测方法只能实现代码克隆类别 1 和类别 2 的检测,难以实现更复杂的克隆类别 3—类别 5 的检测。此类方法的优点是算法实现过程简单,几乎可以用来检测所有编程语言的源代码;缺点是不能识别程序的语法、语义等信息,检测准确率较低。

2)基于词法的检测方法

基于词法的检测方法,也称为基于 Token 的检测方法[1,10-11]。首先,将代码段解析成一个字符串序列(Token 序列);接着,检测不同代码段中的 Token 序列,如果存在相同的 Token 子序列,说明存在代码克隆现象。常见的检测算法有最长公共子序列(Longest Common Subsequence,LCS)、后缀树匹配、Karp-Rabin 指纹算法、语义索引技术等。基于词法的检测方法会使用一些轻量级工具统一处理标识符,可以实现对代码克隆类别 3 的检测。此类方法的优点是能使用轻量级工具,可扩展到对多种编程语言的代码和纯文本的检测,同时相对于基于语法、语义等的复杂检测算法具有更低的时空复杂度;其缺点与基于文本的检测方法类似,即不能识别程序的语法、语义等逻辑信息,因此检测准确率较低。

3)基于语法的检测方法

基于语法的检测方法,也称为基于树的检测方法[12-14]。首先,通过对代码进行词法和语法分析,来构建源程序的一棵抽象语法树;接着,比较相同或相似的子树,进而确定是否进行了代码克隆操作。基于语法的检测方法通常和基于词法的检测方法相结合,通过词法和语法的分析,实现对代码克隆类别 3 的检测。此类方法的优点是相对于基于文本或基于 Token 的检测方法,可以识别程序的语法信息,提高检测准确率;其缺点是构造 AST 以及匹配语法子树两种算法的代价都很大,随着程序规模的扩大,最终检测方法的时间复杂度和空间复杂度都会非常高。

4)基于语义的检测方法

基于语义的检测方法,也称为基于图的检测方法[15-20]。

首先，通过对代码的语法结构、上下文环境等进行分析，来构建源程序的程序依赖图；接着，通过匹配算法和程序切片得到相同或相似的子图同构的 PDG，进而确定是否进行了克隆操作。通过语义分析、程序中函数调用分析等手段，基于语义的检测方法可以实现对代码克隆类别 4 和类别 5 的检测。此类方法的优点是可以识别程序的语义逻辑信息，提高检测准确率；其缺点是构造 PDG 和子图同构的 PDG 的代价较大，且随着程序规模的扩大，检测方法的时间复杂度和空间复杂度也在不断提高。

5）基于度量值的检测方法

基于度量值的检测方法[21-28]只针对对代码进行固定粒度检测的情况。首先，将代码切分成固定粒度的比较代码单元；接着，从比较代码单元中抽取度量值，进而确定是否进行了代码克隆操作。度量值包括代码变量、参数、返回值等。基于度量值的检测方法的优点是检测准确率高，便于代码重构；其缺点是局限于固定粒度的检测，如果粒度太大，则漏检率会很高。

本文对上述 5 类检测方法的代码中间表现形式、匹配算法及优缺点做了比较，如表 1 所列。

表 1　不同代码相似性检测方法的比较

Table 1　Comparison of different code similarity detection methods

检测分类	中间表现形式	匹配算法	优点	缺点
基于文本	字符	字符匹配	算法实现简单，几乎可以检测所有编程语言的源代码	不能识别程序的语法、语义等信息，检测准确率较低
基于词法	Token 序列[1,10-11]	LCS、后缀树[24]、语义索引、Karp-Rabin 指纹算法	使用轻量级工具，可扩展到对多种编程语言的代码和纯文本的检测，同时相对于复杂算法具有更低的时空复杂度	不能识别程序的语法、语义等逻辑信息，检测准确率较低
基于语法	抽象语法树[12-14]	子树匹配[12]	可识别程序的语法信息，检测准确率较高	构造 AST 的代价较大，子树匹配算法的复杂度较高
基于语义	程序依赖图[15-20]	子图匹配[22]、程序切片[17,29]	可识别程序的语义逻辑信息，检测准确率较高	构造 PDG 和子图同构的 PDG 的代价较大；随着程序规模的扩大，时间复杂度和空间复杂度也提高
基于度量值	程序属性[30-33]	直接比较[30]、欧氏距离[31]、度量值比较[30,32]	检测准确率高，便于代码重构	局限于固定粒度的检测，如果粒度太大，则漏检率会很高

3　代码相似性检测工具

3.1　代码相似性检测工具的介绍

表 2 列出了 2.3 节中 5 类代码相似性检测方法对应的工具及其核心思想。

随着机器学习（Machine Learning，ML）和深度学习（Deep Learning，DL）的迅速发展，基于这些技术实现的代码相似性检测方法不断出现[35]。White 等[36]提出了基于深度学习技术的检测方法，其中用于表示源代码中的术语和片段的所有内容都是从存储库中挖掘出来的。代码分析支持一个依赖于深度学习的框架，用于自动链接在词汇级别挖掘的模式与在语法级别挖掘的模式。其从软件维护者的角度评估了基于学习的新型代码克隆检测方法的可行性，对 8 个真实 Java 系统中的 398 个文件对和 480 个方法对进行了采样和手动评估，在文件级和方法级样本上该方法的正确预测率都达到了 93%；并比较了此方法与传统面向结构的技术，发现基于深度学习的方法检测到了知名工具 Deckard 未检测到或次优报告的代码克隆问题。Sheneamer 等[37]使用 AST 的新特征来检测 PDG 的句法克隆和语义克隆，具体是将一对方法块表示为向量，并使用提取到的特征来学习模型，以便使用大量算法来检测语法和语义克隆。最终发现，对于使用的所有分类器，使用这种新特征可以表现出更高的性能。其中，Rotation Forest，Random Forest 和 Xgboost 分类器表现出了很好的效果。随着语法和语义特征的增加，每个分类器的性能都会大大提高。平均而言，当添加语法特征时，分类器的性能提高 10.8%，而继续在所有分类算法中添加语义特征时，分类器的性能提高 19.2%，这表明添加除传统特征之外的更复杂的特征在代码克隆检测中非常有效。

表 2　代码相似性检测方法对应的工具及其核心思想

Table 2　Tools and core ideas of code similarity detection methods

检测分类	介绍	检测工具
基于文本	每行代码做哈希，进行文本比较	Johnson
	每行代码做哈希，使用点阵图可视化比较	Duploc
	使用合适的指纹检测相似文件	Sif
	使用散点图组合较小的孤立复制片段	DuDe
	倒排索引的数据结构和具有 n 邻居距离概念的索引	SDD
	标识符和注释的潜在语义索引	Marcus
	输出语法，然后与阈值进行文本比较	Basic NICAD
	具有灵活的代码规范化和过滤的语法输出，然后与阈值进行文本比较	Full NICAD
	转换为中间格式的原子指令和编辑距离	Nasehi
	算法与灵活选项的文本比较	Simian
基于词法	每行 Token 生成后缀树	Dup
	Token 标准化，基于后缀树搜索	CCFinder(X)
	用于非常大的系统的 CCFinder 的分布式实现	D-CCFinder
	使用 CCFinder 的无间隙克隆，使用间隙和克隆散点图以交互式和可视方式查找缺口克隆	GeX
	灵活的 Token 化和后缀数组比较	RTF
	频繁 Token 序列的数据挖掘	CP-Miner
	代码得到 Token，建立索引查找	ConQAT
	具有后缀数组的 IDE 中的实时 Token 比较	SHINOBI
	带有 Token 频率表的 Karp-Rabin 字符串匹配算法	CPD
	与 Visual Studio 集成的规范化 Token 比较	Clone Detective
	与后缀树的标准化 Token 比较	Clones
	克隆适用于一次检测多个版本的克隆	iClones
	Winnowing 算法基于滑动窗口提取指纹特征[34]	棱镜七彩

(续表)

检测分类	介绍	检测工具
基于语法	哈希语法树和树比较	CloneDR
	语法模式和模式匹配的推导	Asta
	哈希语法树和树比较	cdiff
	语法树的序列化和后缀树检测	cpdetector
	语法树的度量标准和度量向量与哈希的比较	Deckard
	AST 节点的后缀树比较	Tairas
	使用频繁项集数据挖掘技术的 AST 的 XML 表示	CloneDetection
	AST 的 XML 表示和反统一/代码抽象	CloneDigger
	具有 levenshtein 距离的 CodeDOM 图的 Token 序列	C2D2
	AST 节点的 Token 序列和无损数据压缩算法	Juillerat
	从 ANTLR 获得的子树比较	SimScan
	与 cpdetector 一样,但在解析树的节点上工作	clast
	与 CloneDr 一样具有 AST 的不同中间表示	ccdiml
	AST 到 FAMIX,然后进行树匹配	Coogle
基于语义	对 PDG 中类似子图的近似搜索	Duplix,GPLAG
	使用切片在 PDG 中搜索类似的子图	Komondoor
	将 PDG 子图映射到结构化语法并重用 Deckard	Gabel
	预处理并且哈希之后得到数字字符串,构造后缀树	Ccdiml
	方法块为向量,增加 AST 新特征到 PDG 中	ML
基于度量值	神经网络聚类特征向量	Davey
	比较函数/开始结束块的指标	—
	比较网站的指标	—

3.2 代码相似性检测工具的检测评价

在代码相似性检测模型的基础上,不同检测工具对不同的检测规模、编程语言等具有不同的检测效果。针对不同检测方法,为满足不同的检测需求并达到相应的检测效果,本文调研了多种检测工具,并从精确度、召回率、克隆类型等多个角度对检测工具和检测效果进行了评价。

3.2.1 精确度与召回率

精确度(Precision)是指代码相似性检测算法所检测到的候选代码与克隆真实代码的比例[35]:

$$Precision=TP/(TP+FN) \tag{1}$$

召回率(Recall)是指所有被检测到的代码克隆数量占整体代码克隆数量的比例[35]:

$$Recall=TP/(TP+FP) \tag{2}$$

式(1)和式(2)中,TP(True Positive)是被模型预测为正的正样本数量;TN(True Negative)是被模型预测为负的负样本数量;FP(False Positive)是被模型预测为正的负样本数量;FN(False Negative)是被模型预测为负的正样本数量。

Ragkhitwetsagul 等[38]针对第 2.3 节中提到的 5 种克隆类别,在不同参数的设置下,按照准确率、召回率、F-score 等多种评价标准对克隆检测、剽窃检测、压缩等检测类型涉及到的 30 种工具做了详细的比较与排名,其中克隆检测工具主要包括 ccfx,Deckard,iClones,NICAD 和 simian 等;接着,选取前 10 名工具继续进行比较与排名,并展示了这些工具在特定数据集上设置不同数量时所得的精度,结果证明 simian 和 ccfx 是基于文本和基于 Token 检测的首选工具。

3.2.2 克隆类型

克隆类型[35]是针对上述 5 类检测方法,评价一个代码相似性检测工具的重要指标,这一指标决定了代码相似性检测工具的应用场景。

Roy 等[39]从用法、交互、语言、克隆、技术、调参、基本转换/标准化、代码显示、程序分析、评估等多个角度对 38 种检测工具做了不同级别的解释,并对每一类检测方法的可用工具做了相应的评估;同时根据第 2.3 节中提到的 5 种克隆类别的具体情况,对这些检测工具做了整体的评估,其中克隆效果检测评级标准如表 3 所列。表 3 中,L 表示每种技术或工具以不同精度和召回率检测每个(子)场景的能力,L 后的数字越小,检测能力就越强。

表 3 克隆效果检测评级标准

Table 3 Detection performance rating

符号	含义	描述
L1	非常好	以高精度和可靠性检测克隆;具有针对不同类型克隆的可调参数;具有用于检测克隆的不同粒度;能够以合理的时间和空间检测(子)场景的克隆
L2	好	检测(子)场景的克隆,但可能返回少量误报;可能会错过一些克隆;不符合 L1 的一个或多个标准
L3	中等	检测(子)场景的克隆,但可能返回许多误报;不符合 L1 的多个标准
L4	低	检测到大量误报(低精度);也可能错过许多类似的克隆(低召回率);不符合 L1 的许多标准
L5	可能可以	技术/工具的基本技术可能能够检测所讨论的(子)场景的克隆;可能会产生大量误报(精度非常低);可能会遗漏一些克隆(非常低的召回率)
L6	可能不可以	可能是不可能地检测所讨论的(子)场景的克隆;我们认为没有实证研究或任何证据表明该主题工具有能力检测所讨论的(子)场景的克隆
L7	不能	不可能检测到克隆的克隆(子)方案有问题;没有经验研究或任何证据表明该工具能够检测克隆(子)方案的问题

表 4 列出了对不同检测工具的检测效果的评估结果。

表 4 不同工具的检测效果评级

Table 4 Detection performance rating results of different tools

检测分类	检测工具	1)完全	2)删除空格注释	3)		4)			5)实现不同
				修改名称	修改类型	增加语句	删除语句	修改语句	
基于文本	Johnson	L2	L1	L7	L7	L7	L7	L7	L7
	Duploc	L7	L1	L7	L7	L3	L7	L2	L7
	Sif	L3	L3	L7	L5	L7	L7	L5	L7
	DuDe	L5	L1	L7	L7	L3	L3	L2	L7
	SDD	L3	L2	L7	L7	L4	L4	L2	L7
	Marcus	L3	L7	L7	L3	L7	L7	L3	L7
	Basic NICAD	L1	L1	L5	L5	L2	L2	L2	L7
	Full NICAD	L1	L1	L1	L2	L2	L2	L2	L7
	Nasehi	L3	L3	L3	L3	L4	L4	L3	L3
	Simian	L6	L1	L3	L3	L7	L7	L7	L7
基于词法	Dup	L7	L1	L1	L2	L7	L7	L7	L7
	CCFinder(X)	L2	L2	L2	L2	L7	L7	L5	L7
	GeX/Gemini	L2	L2	L2	L2	L3	L3	L3	L7
	RTF	L1	L1	L2	L2	L7	L7	L7	L7
	CP-Miner	L2	L2	L2	L2	L3	L3	L2	L7
	SHINOBI	L2	L2	L2	L2	L7	L7	L5	L7
	CPD	L1	L6	L3	L3	L7	L7	L7	L7
	Clone Detective	L2	L2	L2	L2	L7	L7	L7	L7
	Clones/iClones	L1	L1	L1	L2	L7	L7	L7	L7
	棱镜七彩	L1	L1	L2	L2	L3	L3	L3	L5
基于语法	CloneDR	L1	L1	L1	L1	L3	L3	L	L7
	Asta	L1	L1	L3	L3	L7	L7	L3	L7
	cpdetector	L1	L1	L2	L2	L7	L7	L7	L7
	Deckard	L2	L2	L2	L2	L3	L3	L3	L7
	Tairas	L2	L2	L6	L3	L7	L7	L7	L7
	CloneDetection	L2	L1	L2	L2	L7	L7	L7	L
	CloneDigger	L1	L2	L2	L2	L6	L7	L2	L7
	C2D2	L3	L3	L3	L3	L6	L6	L3	L7
	Juillerat	L1	L1	L7	L7	L7	L7	L7	L7
	SimScan	L1	L1	L2	L2	L5	L5	L5	L7
	Ccdiml	L1	L1	L1	L1	L3	L1	L3	L7

（续表）

检测分类	检测工具	1）完全	2）删除空格注释	3）		4）			5）实现不同
				修改名称	修改类型	增加语句	删除语句	修改语句	
基于语义	Dupli	L2	L2	L2	L2	L3	L3	L3	L7
	Komondoor	L2	L2	L2	L2	L5	L5	L4	L7
	GPLAG	L2	L2	L2	L2	L3	L3	L4	L3
	Gabel	L2	L2	L2	L2	L3	L3	L2	L7
基于度量值	Kontogiannis	L2	L2	L2	L2	L4	L4	L4	L4
	Mayrand	L2	L7	L2	L2	L4	L4	L3	L7
	Dagenais	L3	L3	L3	L3	L5	L5	L3	L6
	Merlo	L2	L2	L2	L2	L4	L4	L3	L7
	Davey	L2	L2	L2	L2	L5	L5	L3	L6
	Patenaude	L2	L2	L2	L2	L4	L4	L4	L7
	Antoniol	L2	L2	L2	L2	L4	L4	L3	L7

举例说明该表的使用及作用：当用户想要检测包含不同语言的许多系统的克隆情况时，目的是检测出尽可能多的克隆类型，同时计算复杂性应该是合理的（不一定是理想的）。基于 Token 的工具在大多数情况下依赖于语言（至少需要词法分析器），但计算效率高。寻找基于 Token 的检测工具时，在表 4 中可以看到 CP-Miner 涵盖了大多数克隆类型/子场景，但结合工具本身的特点发现，如果需要适应不同语言，则需要成熟的解析器，因此不选择 CP-Miner。接着，结合工具属性值得到基于文本的 Basic NICAD 检测工具、基于 Token 的 Cordy 工具、基于树的 Deckard 工具，但是 Cordy 仅适用于 HTML，将其排除。从表 4 中看到，Deckard 比 Basic NICAD 涵盖了更多的克隆类型。

3.3　总结

综合 3.2 节中不同评价指标对检测工具进行的多角度评估，simian 被证明是基于文本检测的首选工具，ccfx 和棱镜七彩是基于 Token 检测的首选工具。对于基于语法和语义的检测方法，在加入更复杂的语义特征后，可以结合现有的深度学习技术提升其性能。基于树的 Deckard 和基于图的 Gabel 与 Duplix 工具可以涵盖较多的克隆类型。

为改善代码相似性的检测效果，Toomim 等[40]指出了从软件开发初期就要进行代码相似性检测的必要性，这样便于整个软件系统的开发与维护；Walenstein 等[41]提出应该结合实际应用与检测目标，融合各种检测方法的优点，以降低代码检测的时空复杂度。

4　代码相似性检测的发展趋势

代码相似性检测在 20 年的研究历程中受到了学术界与工业界的广泛关注和重视，取得了很多进展，但其中还存在许多需要进一步探索的新研究方向。

1）对软件代码克隆相关问题的整体研究。代码克隆包括克隆分析、克隆管理、克隆检测、克隆可视等多个方面[42]。然而，现有的研究成果主要是针对克隆检测做详细的分析，缺乏对其他方面的研究与探索，因此代码克隆的整体研究将有待拓展。

2）有效应对大数据时代数据规模的挑战[42-43]。关于代码相似性检测的研究，虽然已经涌现出各具特色的检测方法和形态各异的检测工具，但在代码开源和代码大数据的时代，众多的检测方法并不能有效应对大规模代码检测带来的挑战。因此，未来的代码相似性检测研究应重点关注如何提升检测方法对大规模代码的处理能力，有机融合现有检测技术与各类软件开发与维护技术，最终提升代码相似性检测技术在实际应用中的价值。

3）对 AI 生成的代码进行有效相似性检测。随着人工智能的迅速发展，软件开发前沿理论研究包括群智能软件[44]和基于 AI 技术自动生成的软件[45]。由这些理论指导并开发出的软件系统也需要做代码相似性检测，因此未来代码相似性检测技术需要继续拓展检测目标，以适应新的发展需求。

结束语　代码开源已经成为了信息领域的潮流。代码克隆虽然一定程度上降低了软件开发成本，但也对软件的健壮性、可维护性等带来了影响。为应对代码克隆带来的各种危害，目前学术界和工业界提出了很多代码相似性检测的方法并开发了相应的工具。为便于用户更好地理解代码克隆并进行代码克隆检测，本文基于当前研究进展，首先介绍了代码克隆的基本概念，总结了 5 类代码相似性检测的方法；然后整理归类了目前已有的各种代码相似性检测的工具，并从多方面对它们的性能进行了评估；最后展望了代码相似性检测问题的未来研究方向。

参 考 文 献

[1] BAKER B S. A Program for Identifying Duplicated Code[C]// Proceedings of Computing Science and Statistics: 24th Symposium on the Interface. 1992.

[2] KIM M, SAZAWAL V, NOTKIN D, et al. An empirical study of code clone genealogies [J]. ACM SIGSOFT Software Engineering Notes. ACM, 2005, 30(5): 187-196.

[3] KAMIYA T, KUSUMOTO S, INOUE K. CCFinder: A multilinguistic token based code clone detection system for large scale source code [J]. IEEE Transactions on Software Engineering, 2002, 28(7): 654-670.

[4] CAO Y Z, JIN M Z, LIU C. Overview on code clones detection [J]. Computer Engineering: Science, 2006, 28 (A2): 9-13.

[5] BELON S, KOSCHKE R, ANTONIOL G, et al. Comparison and evaluation of clone detection tools [J]. IEEE Transactions on Software Engineering, 2007, 33(9): 577-591.

[6] FOWLER M. Refactoring: Improving the Design of Existing Code[C]// Xp Universe and First Agile Universe Conference on Extreme Programming and Agile Methods-Xp/agile Universe. Springer-Verlag, 2002: 256.

[7] MANN Z A. Three public enemies: cut, copy, and paste [J]. Computer, 2006, 39(7): 31-35.

[8] RIEGER M. Effective clone detection without language barriers [D]. Bern, Switzerland: University of Bern, 2005.

[9] WHALE G. Plague: plagiarism detection using program structure[R]. 1988.

[10] DUCASSE, STÉPHANE, RIEGER M, et al. A Language Independent Approach for Detecting Duplicated Code[C]// IEEE International Conference on Software Maintenance. IEEE, 1999.

[11] GITCHELL D, TRAN N. Sim: a utility for detecting similarity in computer programs [C]// Thirtieth Sigcse Technical Symposium on Computer Science Education. ACM, 1999.

[12] KIM Y C, CHO Y Y, MOON J B. A Plagiarism Detection System Using A Syntax-Tree [C]// International Conference on Computational Intelligence. DBLP, 2004.

[13] JIANG L, MISHERGHI G, SU Z, et al. DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones[C]// Inter-

national Conference on Software Engineering. IEEE Computer Society, 2007.
[14] BAXTER I D, YAHIN A, MOURA L, et al. Clone detection using abstract syntax trees [C]// Conference on Reverse Engineering. IEEE, 2006.
[15] FERRANTE J, OTTENSTEIN K J, WARREN J D. The program dependence graph and its use in optimization [J]. Acm Trasactions on Programming Languages & Systems, 1987, 9(3): 125-132.
[16] LIU C, CHEN C, HAN J, et al. GPLAG: detection of software plagiarism by program dependence graph analysis [C]// Acm Sigkdd International Conference on Knowledge Discovery & Data Mining. ACM, 2006.
[17] KOMONDOOR R, HORWITZ S. Using Slicing to Identify Duplication in Source Code[C]// International Symposium on Static Analysis. Springer-Verlag, 2001.
[18] KRINKE J. Identifying Similar Code with Program Dependence Graphs[C]// Conference on Reverse Engineering. IEEE, 2001.
[19] PHAM N H, NGUYEN H A, NGUYEN T T, et al. Complete and accurate clone detection in graph-based models[C]// International Conference on Software Engineering. IEEE, 2009.
[20] SHENEAMER A, ROY S, KALITA J. A detection framework for semantic code clones and obfuscated code [J]. Expert Systems with Applications, 2018, 97(1): 405-420.
[21] ARWIN C, TAHAGHOGHI S M M. Plagiarism detection across programming languages[C]// Computer Science, Twenty-nineth Australasian Computer Science Conference. DBLP, 2006.
[22] ENGELS S, LAKSHMANAN V, CRAIG M. Plagiarism detection using feature-based neural networks [C]// Sigcse Technical Symposium on Computer Science Education. ACM, 2007.
[23] SINGHE S. Neural networks and disputed authorship: new challenges[C]// International Conference on Artificial Neural Networks. IET, 1995: 24-28.
[24] ELENBOGEN B S, SELIYA N. Detecting outsourced student programming assignments[M]. Consortium for Computing Sciences in Colleges, 2008.
[25] CIESIELSKI V, WU N, TAHAGHOGHI S. Evolving Similarity Functions for Code Plagiarism Detection [C]// Conference on Genetic & Evolutionary Computation. ACM, 2008.
[26] CHEN X, FRANCIA B, LI M, et al. Shared Information and Program Plagiarism Detection [J]. IEEE Transactions on Information Theory, 2004, 50(7): 1545-1551.
[27] ZHANG L, ZHUANG Y T, YUAN Z M. A Program Plagiarism Detection Model Based on Information Distance and Clustering [C]// International Conference on Intelligent Pervasive Computing. IEEE Computer Society, 2007.
[28] XIONG H, YAN H H, HUANG Y G, et al. Code Similarity Detection Approach Based on Back-Proagation Neural Network [J]. Computer Science, 2010, 37(3): 159-164.
[29] WEISER M. Program slicing [J]. TSE, 1984, 10 (4): 352-357.
[30] MERLO E, ANTONIOL G, PENTA M D, et al. Linear complexity object-oriented similarity for clone detection and software evolution analyses [C]// IEEE International Conference on Software Maintenance. IEEE Computer Society, 2004.
[31] KONTOGIANNIS K. Evaluation experiments on the detection of programming patterns using software metrics[C]// Conference on Reverse Engineering. IEEE, 1997.
[32] MAYRAND J, LEBLANC C, MERLO E M. Experiment on the automatic detection of function clones in a software system using metrics [C]// International Conference on Software Maintenance. IEEE, 1996.
[33] GUO Y, CHEN F H, ZHOU M H. Code Clone Detection Method for Large-Scale Source Code [J]. Journal of Frontiers of Computer Science and Technology, 2014, 8(4): 417-426.
[34] SCHLEIMER S, WILKERSON D S, AIKEN A. Winnowing: local algorithms for document fingerprinting [C]// Proc Acm Sigmod Conference. 2003.
[35] CHEN Q Y, LI S P, YAN M, et al. Code clone detection: A literature review[J]. Journal of Software, 2019, 30(4): 962-980.
[36] WHITE M, TUFANO M, VENDOME C, et al. Deep learning code fragments for code clone detection [C]// IEEE/ACM International Conference on Automated Software Engineering. IEEE, 2016.
[37] SHENEAMER A, KALITA J. Semantic Clone Detection Using Machine Learning [C]// IEEE International Conference on Machine Learning & Applications. IEEE, 2017.
[38] RAGKHITWETSAGUL C, KRINKE J, CLARK D. A comparison of code similarity analysers [J]. Empirical Software Engineering, 2017(9): 1-56.
[39] ROY C K, CORDY J R, KOSCHKE R. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach [J]. Science of Computer Programming, 2009, 74(7): 470-495.
[40] TOOMIM M, BEGEL A, GRAHAM S L. Managing Duplicated Code with Linked Editing[C]// Proc IEEE Symposium on Visual Languages & Human-centric Computing. 2004.
[41] WALENSTEIN A, JYOTI N, LI J, et al. Problems creating task-relevant clone detection reference data[C]// Conference on Reverse Engineering. IEEE, 2003.
[42] SAJNANI H, SAINI V, SVAJLENKO J, et al. SourcererCC: scaling code clone detection to big-code[C]// International Conference on Software Engineering. IEEE, 2016.
[43] SAJNANI H. Large-Scale Code Clone Detection [D]. Irvine: University of California, 2016.
[44] BECKER K, SCHLICH J G. AI Programmer: Autonomously Creating So ware Programs Using Genetic Algorithms[J/OL]. http://arxiv.org/abs/1709.05703.
[45] DANIEL A, ABOLA A, NOROUZI M, et al. Neural Program Synthesis with Priority eue Training[J/OL]. http://arxiv.org/abs/1801.03526.

ZHANG Dan, master. Her main research interests include information security and software analysis.

LUO Ping, born in 1959, Ph.D, professor. His main research interests include information security and code detection.