# 面向管理的克隆代码研究综述

苏小红 张凡龙

(哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

**摘 要** 软件复用作为一种常见的软件开发手段,会导致大量克隆代码的产生,这无疑增加了软件维护的代价.对克隆代码的维护需求引发了一系列关于克隆代码的研究,如克隆检测、克隆分析、克隆维护等.但是,上述克隆研究无法解决克隆代码维护困难的问题.为了避免克隆代码维护困难,提高软件的可维护性,克隆代码管理势在必行.然而,目前的克隆管理与克隆检测、克隆分析、克隆维护等过程彼此之间是相互独立的,也没有与软件开发过程相结合,无法有效解决克隆代码维护困难的问题.首先,该文分析了克隆代码研究领域的热点和趋势,以及克隆检测、分析和维护的研究进展.其次,该文对克隆管理的研究现状进行了分析,重点对克隆代码研究内容之间的关系以及现有的克隆管理存在的不足和难点问题进行了分析,指出只有将克隆检测、分析和维护等过程与软件开发过程有机地结合为一个整体,才能有效降低克隆维护的代价,但这势必增加了克隆管理的难度.为此,在未来的研究展望中,该文给出了一个面向软件开发过程的克隆管理方法,将克隆检测、克隆分析和克隆维护等与软件开发过程紧密结合,以实现边开发、边维护和边管理克隆代码.最后,该文分析了克隆代码研究领域未来的研究方向和发展趋势.克隆管理为克隆代码研究注入了新的活力,现已引起学术界和工业界的广泛关注,对于提高软件的可维护性、可理解性以及软件质量都具有重要意义.

**关键词** 克隆代码;克隆管理;克隆检测;克隆分析;克隆维护

**中图法分类号** TP311 **DOI 号** 10.11897/SP.J.1016.2018.00628

# A Survey for Management-Oriented Code Clone Research

SU Xiao-Hong ZHANG Fan-Long

(*School of Computer Science and Technology, Harbin Institute of Technology, Harbin* 150001)

**Abstract** Software reuse is a common technology in software development which would introduce amount of code clones into the software. The existence of code clones increases the cost of software maintenance. The request for code clone maintenance given birth to a series of researches, such as the detection, analysis, maintenance of code clones. Concretely, clone detection aims to help developer collect all the code clones from software repository, clone analysis aims to help developer understand the presence of code clones as well as the impact to the software, and clone maintenance aims to help developers solve the problems caused by code clones. Though the researches have been launched much, the key problems of clone maintenance, i. e. the complexity and high cost of clone maintenance have not been solved effectively. To reduce the cost of software maintenance and improve the maintainability of the software, the management of code clones is imperative. However, the current studies on code clones are independent of each other. And unfortunately, the current clone management methods are not associated with the development process of the software, accordingly, the complexity of clone maintenance cannot be handled well. In

收稿日期:2016-07-13;在线出版日期:2017-07-28.本课题得到"十三五"国家重点研发计划课题(2017YFC0702204)、国家自然科学基金(61173021,61672191)资助.**苏小红**,女,1966 年生,博士,教授,中国计算机学会(CCF)高级会员,主要研究领域为智能软件工程、程序分析、软件错误定位、克隆代码检测与分析和软件缺陷预测.E-mail:sxh@hit.edu.cn.**张凡龙**(通信作者),男,1987 年生,博士研究生,主要研究方向为软件工程、程序分析、克隆代码分析和维护、软件仓库挖掘等.E-mail:hitzhangfanlong@gmail.com.

this paper, we make a survey of the state-of-art achievements in the code clone researches. Firstly, we analyze the hotspots and trends of clone research, and find that clone detection and analysis were the hotspot research in the past, and the clone maintenance and clone management will become the hotspot research in the future. After that, we investigate all the clone researches from the different perspectives including clone definition, clone representation, clone detection, clone analysis and clone maintenance. Then, we analyze the current research status of clone management, and focus on the relationship among these researches. We also point out the shortage and the difficulties of current clone management methods from the point of view on clone management. We believe that clone management should combine with clone detection, analysis, maintenance as well as the software development process so as to effectively reduce the cost for maintaining clones at the software development phase. Yet, it will also undoubtedly increase the difficulty for clone management. To this end, in section of the future research prospects, we propose an approach of clone management that orienting to software development process, which aims to maintain and manage code clones along with software development process through combining clone detection, clone analysis, clone maintenance. In our approach, we believe that the clone representation and clone evaluation will be the key point in clone management. Finally, we discuss the directions and trends of future research. Clone management has brought new vitality to code clone research, which has aroused widely attention from both academia and industry, and it is significant in improving the maintainability, comprehensibility and quality of the software.

**Keywords**　code clones; clone management; clone detection; clone analysis; clone maintenance

## 1　引　言

克隆代码(简称克隆)是软件中彼此相似的代码片段[1].导致克隆代码产生的原因是多种多样的,最常见的形式是通过复制粘贴操作复用已有的代码.研究结果表明系统中通常存在着大量的克隆代码,其比例大约在7%～23%左右,有的甚至高达59%[2].长期存在于系统中的克隆代码,不仅影响着系统的可维护性,也影响着系统的可理解性.因此,对克隆代码的维护需求引发了一系列关于克隆代码的研究,包括克隆检测、克隆分析、克隆维护和克隆管理等.

克隆检测是克隆领域研究最早,也是研究最为充分的一个研究内容.许多克隆检测方法和工具被相继提出,如NiCad[3]、CCFinder[4]等.按照所使用的检测技术,可划分为基于Text、基于Token、基于Tree、基于Graph和基于Metric的检测方法,不同的方法适合检测不同类型的克隆代码.但是,目前尚没有一种检测方法能够检测出系统中所有的克隆代码,尤其是语法相似的和语义相似的克隆代码.更重要的是从系统中检测出克隆代码并不是克隆研究的最终目的,还需要对克隆代码进行分析研究,例如克隆代码产生的原因、对系统的不利影响等,可以辅助开发人员理解和维护克隆代码.因此,克隆检测仅仅是克隆代码研究的初级阶段.

克隆代码分析的目的是辅助开发人员理解克隆代码,是目前克隆代码研究中较为活跃的研究内容.例如,Kim等人[5]提出克隆家系的概念描述、分析克隆代码的演化过程及其变化规律.这引发了对克隆演化的研究热潮.克隆演化分析是从软件演化的角度来揭示克隆代码的变化规律及其基本特征.在此之前,有研究认为克隆代码是一种代码坏味[6],容易引发软件缺陷,从而使软件难以理解和维护[7-8].但对克隆演化研究使人们逐渐意识到克隆代码并非都是有害的.有研究结果表明克隆代码的稳定性较好,复用此类克隆代码不仅不会降低软件质量,还可以提高软件开发的效率[9-11].事实上,克隆有害性研究一直被研究人员所关注,直到现在也没有确切的证据表明克隆代码是完全有害的[12-14].虽然克隆有害性评价分析可以辅助开发人员分析克隆代码对系统产生的不利影响,但无法直接消除克隆代码及其

对系统产生的不利影响;而且很多评价方法属于实证研究,带有一定的主观性和片面性,缺少客观的理论依据.此外,为了辅助开发人员更好地理解和维护克隆代码,不仅需要全面客观的克隆评价方法,还需要在克隆可视化分析、可重构性分析、可复用性分析、缺陷分析等方面进行研究.

克隆代码的可重构性分析和可复用性分析的目标是帮助维护克隆代码(克隆维护),例如使用重构方法消除系统中有害的不适合重构的克隆代码[15-16]、复用已有的有益的克隆代码等[17-18].但由于重构条件十分苛刻,大部分克隆不适合重构,重构也无法解决克隆代码问题.同时,克隆代码有害与有益的标准难以确定,也给克隆可复用性分析带来困难,使得克隆复用研究进展也较为缓慢.为彻底解决克隆代码维护困难的问题,近几年研究人员开始将研究重点转移到克隆管理上[19-20].克隆管理试图从管理的角度将相关的克隆研究统一为一个整体,并与软件开发过程相结合,在开发过程中高效地实现对克隆代码的组织、维护和管理.克隆管理不仅有助于消除有害克隆代码及其对系统的不利影响,提高软件的可维护性和可理解性;还有助于复用有益的克隆代码,提高软件开发效率.克隆管理作为一个崭新的研究内容,已引起学术界和工业界的广泛关注.但目前克隆管理研究还处于起步阶段,缺少有效且完善的克隆管理方法.

本文通过调研近年来克隆代码领域的研究情况,认识到克隆管理是未来克隆代码领域的研究热点与重点.在调研克隆检测、分析和维护的研究现状的基础上,按照不同阶段描述克隆研究内容及其之间的关系,指出了现有研究的不足与难点问题.认为将克隆检测、分析和维护等过程与克隆管理、软件开发过程有机结合为一个整体是极为必要的,可以实现边开发、边维护、边管理克隆代码的目标.为了实现此目标,本文提出一个面向软件开发过程的克隆代码管理方法,并对未来的研究方向和发展趋势进行了展望,给出为实现面向软件开发过程的克隆管理方法的亟待解决的几个问题.

本文第 2 节分析克隆代码研究领域的研究趋势;第 3 节、第 4 节分别介绍克隆代码定义和分类,以及目前研究中主流的克隆代码表示和组织形式;第 5 节~第 8 节分别分析克隆检测、克隆分析、克隆维护和克隆管理的研究现状;第 9 节描述目前克隆研究内容及其关系,以及克隆领域的研究不足与难点问题;第 10 节对未来的研究方向和发展趋势进行展望,并给出一个面向软件开发过程的克隆管理方法;第 11 节为本文结论.

## 2 研究热点与趋势

### 2.1 研究热点分析

克隆代码是软件工程领域的一个热点问题,迄今为止已有超过 20 年的研究时间.在此期间,研究人员取得了大量的研究成果.为方便研究人员查阅相关研究,阿拉巴马大学维护了一个克隆代码领域的文献库①.该文献库收录包括学术论文、技术报告以及学位论文在内的 353 篇文献.按照研究的内容和目标,将克隆代码的研究划分为如下 4 个研究方向:

(1) 克隆检测方向.该方向下列出了与克隆代码检测相关的研究内容,旨在从系统中检测克隆代码.

(2) 克隆分析方向.该方向下列出了与克隆代码分析相关的研究内容,旨在帮助程序开发人员分析和理解系统中存在的克隆代码.

(3) 克隆维护与管理方向.该方向下列出了与克隆代码维护与管理相关的研究内容,旨在通过各种技术手段维护和管理系统中的克隆代码.

(4) 综述和工具评估方向.该方向下列出了克隆代码综述和克隆检测工具评估相关的研究内容.

Roy 等人按照上述分类对文献库中文献进行了分析,统计了每个研究方向的文献分布和研究进展[20].但是该文献库仅收录了 2013 年以前的文献.为分析和反映克隆代码研究的最新研究热点与发展趋势,本文在该文献库的基础上做了进一步的扩展.通过检索和收集近几年的克隆代码文献,并按文献库分类方式进行了重新统计和分析.本文所统计和收集的克隆代码文献共分为三类:软件工程领域的重要国际会议论文、重要国际期刊论文以及相关学位论文.其中,重要国际会议论文为 432 篇,期刊论文 113 篇,学位论文 30 篇,共计 575 篇②.

克隆代码研究领域的文献统计分析结果如图 1 所示.从图 1 可以看出,目前对克隆检测与分析的研究较为充分.克隆检测和分析是克隆领域的研究热

① http://students.cis.uab.edu/tairasr/clones/literature/阿拉巴马大学关于克隆代码研究的文献库.该文献库更新至 2013 年,现已停止维护.

② 本文统计的所有文献可分为两个部分:一是来源于阿拉巴马大学的文献库,二是笔者在进行研究期间所阅读整理和收集的文献.在收集的过程中,已经对文献进行了初步筛选,仅选取了领域内权威期刊、会议以及学位论文.

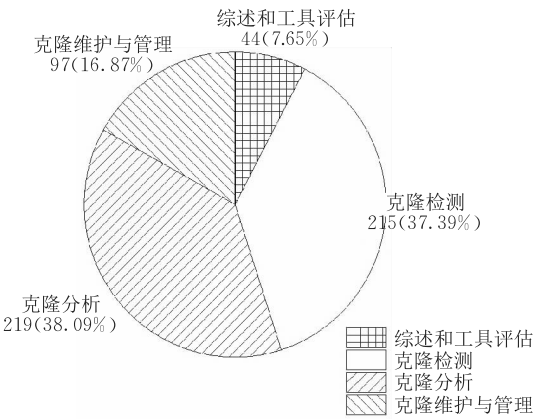



图 1　克隆代码研究领域文献分布

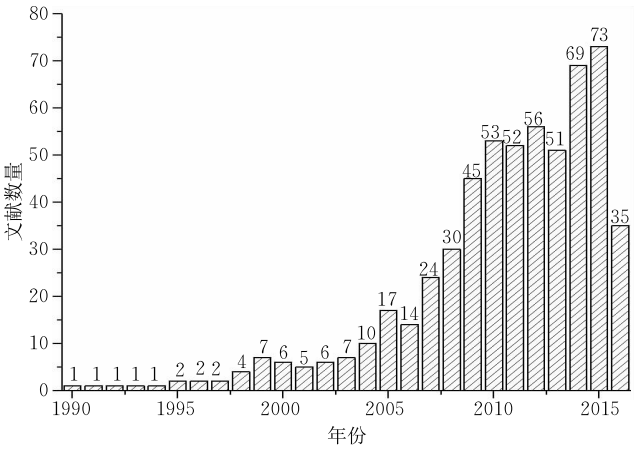
点，分别占比 37.39%和 38.09%. 而克隆维护与管理的研究占比较少(仅占 16.87%)，说明目前对该方向的研究方兴未艾.

### 2.2　研究趋势分析

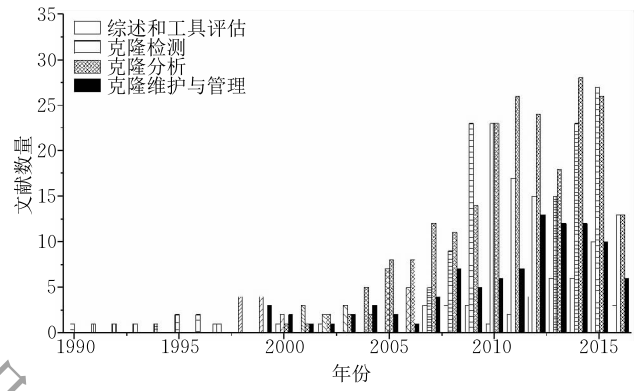
为便于分析克隆代码研究领域的发展趋势，对克隆代码研究领域的文献，按照文献的发表年份和研究方向进行了进一步统计. 统计分析结果如图 2 所示，其中图 2(a)是该领域整体上每年发表文献数量的统计，图 2(b)是在不同研究方向上每年发表文献数量的统计.

从图 2(a)可以看出，克隆代码研究可以划分为两个阶段：1990 年～2003 年和 2004 年～2016 年. 在 2003 年以前，克隆代码研究领域的文献数量较少，这一阶段是克隆代码研究的孕育期. 而在 2004 年以后的近 10 年内，克隆代码研究领域的文献数量快速增长，并且保持在一个较高的水平上. 这一阶段展现出了蓬勃的发展趋势，表明此阶段是克隆代码研究的发展期.

与图 2(a)类似，从图 2(b)在不同研究方向上每年发表的文献数量来看，其研究趋势也以 2003 年为界限划分为两个阶段. 在 2003 年以前，克隆代码研究主要集中在克隆检测方向，处于孕育期. 在 2004 年以后，各个研究方向的文献数量都有了快速增长，进入了发展期. 其中，尤以克隆检测和分析方向的发展最为迅速，发表了大量研究成果，并依然有继续增长的趋势. 这说明克隆检测和分析是目前研究最为充分和最为活跃的研究方向. 相比之下，克隆维护和管理仅在近几年才引起人们的关注. 虽然克隆维护和管理的研究近几年才刚开始起步，文献数量占比相对较少，但已呈现出逐年增长和后来者居上的趋势. 这说明克隆维护和管理正在逐渐成为克隆代码研究领域的新热点.




(a) 克隆领域每年发表的论文数量




(b) 各个克隆活动每年发表的文献数量

图 2　克隆代码领域研究趋势①

为后续讨论方便，本文先介绍克隆代码的定义和分类，然后将克隆代码的研究内容划分为克隆代码的表示和组织、克隆检测、克隆分析、克隆维护以及克隆管理，并对这些研究进行详细分析. 克隆表示和组织研究如何描述系统中的克隆代码，以及如何组织系统中的克隆代码. 克隆检测研究如何从软件系统中检测克隆代码. 克隆分析研究克隆代码演化及其特征，可以辅助开发人员理解克隆代码. 克隆维护研究如何有效地维护系统中的克隆代码，例如重构消除有害的克隆代码、复用有益的克隆代码等. 克隆管理试图从管理的视角解决克隆代码的相关问题，研究如何结合克隆检测、分析和维护等过程对克隆进行统一的管理，以降低软件维护的代价，提高软件开发的质量.

## 3　克隆定义与分类

克隆代码领域中依然缺少统一的克隆代码定义

① 由于 2016 年的文献未被完全收录到数据库中，笔者所收集到的 2016 年文献数据不完全. 在图 2 中，2015 年～2016 年的文献增长趋势未能体现出来.

《计算机学报》

和分类方法.按照克隆代码的语法、语义相似度进行定义和分类,是使用最为广泛的一种形式.可将克隆代码划分为如下四种类型[21]:

Type 1. 除空格、格式和注释外,是完全相同的代码片段(简称 1 型克隆).

Type 2. 除标识符、常量、类型外,是语法结构相同的代码片段(简称 2 型克隆).

Type 3. 拷贝粘贴后修改的代码片段,如改变、增加或删除少量语句的代码,是语法结构相似的代码片段(简称 3 型克隆).

Type 4. 执行相同的功能,但使用不同的语法结构实现的代码片段,是语义相似的代码(简称 4 型克隆).

Type 1 克隆也称为精确克隆,Type 2 和 Type 3 克隆也称为近似克隆.其中,前三类克隆代码是属于语法相似的克隆代码,Type 4 克隆是语义相似的克隆代码.

除此之外,研究人员也研究了其它类型的克隆代码,如函数克隆[22]、模型克隆[23]和结构克隆[24-25]等.函数克隆是指克隆代码为函数的特殊克隆,函数克隆也可以分类为 Type 1～Type 4 克隆.同时,在模型驱动的软件开发过程中,产生了大量的模型文件.研究人员将模型文件中相似的模型图(例如 Simulink 模型图)称为模型克隆.而区别于代码级别的克隆代码,结构克隆是粒度较大的克隆代码形式.一个结构克隆是一个包含若干个代码克隆的文件.通过分析所包含的代码克隆之间的克隆关系,可以获得结构克隆之间的克隆关系.

## 4 克隆表示与组织

克隆表示是采用适当形式来表示系统中的克隆代码.克隆组织是采用一定数据结构来组织系统中的克隆代码.克隆表示和组织往往体现在克隆代码的检测结果中,即克隆检测工具的报告形式.

### 4.1 克隆表示

克隆代码信息包括克隆代码的基本信息和扩展信息.基本信息是克隆代码的最基本信息,例如克隆代码在文件中的位置信息,通常采用"文件名+文件起始行号"的形式来表示.扩展信息是指克隆代码的其它相关支持信息,如上下文信息、演化信息等.

目前的很多克隆检测工具,如 NiCad[3]、CCFinder[4]、iClone[26]、ModelCD[27]和 Decard[28]等,都使用克隆检测报告向分析人员提供克隆代码的检测结果.在克隆检测报告中,使用特定的克隆表示方法来报告克隆代码.由于不同的检测工具采用不同的检测技术来检测克隆代码,因此不同的检测工具的克隆检测结果往往是不同的.由于不同的检测工具缺乏统一的克隆表示形式,要实现各种克隆工具的检测结果信息的共享和融合,是非常困难的.

为了实现各个检测工具的克隆检测结果的共享,克隆检测工具 iClone 使用 Rich Clone Format (RCF)表示克隆代码[29].RCF① 定义了多种函数接口,以方便用户和其它检测工具检索克隆代码.RCF 不仅可以表示克隆代码的基本位置信息,还允许用户对其进行扩展,表示克隆代码的扩展信息.但是,RCF 没有指出可以表示哪些具体的扩展信息以及以何种形式来表示这些信息.因此,RCF 支持的克隆代码信息是有限的,无法描述克隆代码的上下文信息和演化信息等其它扩展信息,无助于跟踪和分析系统中的克隆代码.

为了表示克隆代码的上下文信息,Duala-Ekoko 等人[30]提出了另一种克隆表示形式 Clone Region Description(CRD).与其它表示形式不同的是,CRD 是一种与克隆代码位置无关的表示方法.CRD 不保存克隆代码的位置信息,仅使用克隆代码本身的语法信息和克隆代码所在文件的结构信息表示克隆代码.使用 CRD 可以详细地描述克隆代码的上下文信息,尤其是语法信息.因此,具有上下文信息的 CRD 有助于辅助开发人员跟踪演化中的克隆代码.但是,CRD 只描述了克隆代码的上下文信息,缺乏与克隆代码位置信息和演化过程的描述.既不利于对克隆代码的快速检索,也不利于对克隆代码的演化分析.

各种检测工具的克隆检测结果缺乏统一的表示形式,且现有表示方法仅包含有限的克隆信息,使得现有的克隆表示方法不能满足对克隆代码分析和维护的需求.为此,Harder 分析了统一克隆表示的挑战和限制,并给出了一个克隆表示的概念模型,但未给出具体的克隆表示方法[31].同时,Kapser 等人也认为需要对克隆代码进行统一的表示,并提出了 Unified Clone Model(UCM)模型[32].UCM 模型是一种较为完备的克隆表示方法,不仅可以表示克隆代码的基本位置信息,还可以表示克隆代码的上下文信息、演化信息和度量值信息等,具有较强的理论和实际意义.由于目前克隆检测工具不能直接生成

① http://www.softwareclones.org/rcf.php,RCF 格式文档.RCF 是克隆检测工具 iClone 的默认结构形式,其支持多版本克隆代码的演化信息,RCF 开放 API 给其它用户使用.

《计算机学报》

UCM 格式的克隆检测结果，所以 UCM 提供了相应的 XML Schema 定义. 根据此定义可将其它工具的克隆检测结果转换为 UCM 格式，例如 Harder 开发了一种转换工具 ucm2rcf 以支持将 RCF 格式转换为 UCM 格式①. 但美中不足的是，UCM 使用的过于庞大的结构信息，使其在满足完备性的同时也损失了简洁性和实用性.

综上，一种好的克隆表示方法应该在完备性和实用性之间进行有效折中，既要脱离源代码文本，包含支持克隆代码检索的克隆代码定位信息，还应具有一定的可扩展性，包含支持克隆代码分析和维护的上下文和演化信息等扩展信息.

### 4.2　克隆组织

克隆组织方法是指克隆代码的组织形式，用于结构化系统中的克隆代码，不仅需要组织系统中某一版本内的克隆代码，还需组织该系统在演化过程中的版本间的克隆代码. 目前常用的克隆组织形式为克隆组(或克隆对)和克隆家系. 克隆组用于组织版本内的克隆代码，克隆家系用于组织版本间的克隆代码.

大多数的克隆检测工具仅支持检测单版本系统的克隆代码，因此克隆组和克隆对是大多数检测工具所采用的版本内克隆代码的组织形式. 克隆对仅包含两个克隆代码片段，表示两个克隆片段之间的克隆关系；克隆组是两个或两个以上的克隆代码片段的集合，表示组内所有克隆代码片段之间的克隆关系.

克隆代码不是仅存在于系统的一个版本中，而是会随着软件版本的更新而发生演化. 为有效描述软件系统版本间的克隆代码及其演化过程，Kim 等人提出了克隆家系模型. 克隆家系将多版本克隆代码组织成若干个克隆家系，每个克隆家系分别描述一个克隆组的演化过程，并使用克隆演化模式来描述在相邻两个版本演化时的克隆组变化情况.

图 3 是一个克隆家系的示意图. 图中给出了一个克隆组在五个软件版本中的演化过程($V_i$ 到 $V_{i+4}$). 从 $V_i$ 到 $V_{i+1}$，克隆组中有新的克隆代码片段产生，克隆演化模式为“增加”. 从 $V_{i+1}$ 到 $V_{i+2}$，克隆组首先分裂为两个克隆组(演化模式为“分裂”)；分裂后的一个克隆组中的克隆代码同时发生了相同的变化(演化模式为“一致性变化”). 从 $V_{i+2}$ 到 $V_{i+3}$，上方和下方的克隆组内的克隆代码数量减少(演化模式为“减少”)；下方的克隆组中的克隆代码发生了不一致的变化(演化模式为“不一致性变化”). 从 $V_{i+3}$ 到 $V_{i+4}$ 的克隆组演化模式则分别为“增加”和“减少”.

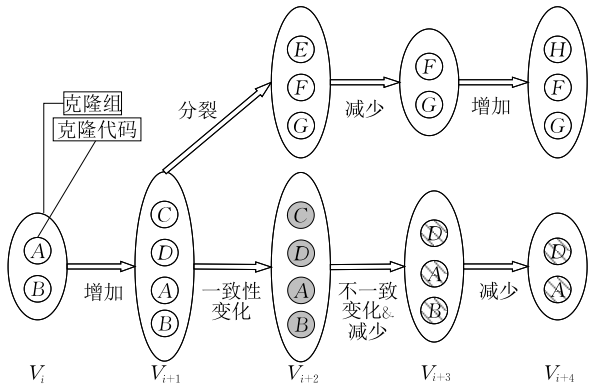



图 3　克隆家系示意图[5]

将版本内的克隆代码组织为克隆组，并使用克隆家系描述克隆代码随软件版本更新的演化过程，已经是目前被广泛认可和采用的一种克隆代码组织形式.

## 5　克隆检测

克隆检测是指从软件中检测并报告克隆代码的位置的研究. 相比于克隆代码领域的其它研究而言，克隆检测技术相对较为成熟.

### 5.1　克隆检测方法

一般而言，克隆代码检测过程分为三个步骤：代码的中间表示、相似性匹配和报告检测结果. “代码的中间表示”是使用不同的方法对源代码进行抽象表示或转换，将其表示为抽象语法树等一些中间表示形式. “相似性匹配”是对代码的中间表示形式进行相似性匹配与检测，寻找相似的代码片段. “报告检测结果”是将彼此相似的克隆代码以某种克隆组织方式存储检测结果，并向分析人员提供克隆检测报告. 迄今为止，研究人员已提出许多种克隆检测方法，并开发了相应的检测工具. 根据所使用的技术不同，将克隆检测方法划分为基于文本(Text)的方法、基于 Token 的方法、基于 Tree(如 Abstract Syntax Tree，AST)的方法、基于 Graph(如 Program Dependency Graph，PDG)的方法和基于度量值(Metric)的方法等.

基于 Text 的克隆检测方法是通过直接比较源

① http://www.softwareclones.org/ucm/index.php/Main_Page，UCM 格式文档. UCM 旨在建立一个统一模型用于组织不同克隆检测工具的检测结果，也可使得检测工具进行信息共享.

《计算机学报》

代码文本,使用字符串匹配等算法来检测克隆代码.因其并没有对源程序进行词法分析,大部分仅可以较好地支持 Type 1 克隆的检测,对其余类型克隆的支持较弱.目前使用较多的基于 Text 的克隆检测工具主要有 duploc[33-34]、Simian①、DuDe[35]、SDD[36]、NiCad[3]等.

基于 Token 的克隆检测方法是通过对源代码进行词法分析生成源代码的 Token 序列,然后通过寻找 Token 序列中相似的子序列来检测克隆代码.因其对源代码进行了词法分析,所以可以较好地支持 Type 2 克隆的检测.但由于缺乏语法和语义分析,其无法较好地支持 Type 3 和 Type 4 克隆的检测.目前使用较多的基于 Token 的克隆检测工具主要有 Dup[37]、CCFinder[4]、CPMiner[38]、iClone[26]等.

基于 Tree 的克隆检测方法是将源代码表示为某种树的形式(如抽象语法树、代码解析树等),然后通过树匹配算法从中寻找相似的子树来检测克隆代码.因其对源代码进行了语法分析,所以提高了克隆代码检测的准确率,可以较好地支持 Type 3 克隆的检测.但是由于树匹配算法的时间复杂度较高,这类算法的检测速度会低于前两种方法.目前使用较多的基于 Tree 的克隆检测工具有 CloneDr[39]、SimScan②、Deckard[28]、CloneDigger[40]等.

前三种克隆检测方法中所使用的中间表示形式较为容易实现,可使用程序静态分析方法获得源代码的中间表示.事实上大部分的克隆检测方法和检测工具都属于前三种方法,并可以检测系统中的大部分克隆代码.

基于 Graph 的克隆检测方法的主要思路是,将源代码转化成某种程序图形式(如程序依赖图),然后通过寻找同构的子图来检测克隆代码[41-42].因程序依赖图表示了程序的语义信息,所以该方法可以支持语义相似的 Type 4 克隆代码的检测.但由于程序依赖图生成算法和图匹配算法的时间和空间复杂度极高,因而导致这类检测算法的时空开销过大,使其无法应用于大规模程序的克隆代码检测.目前基于 Graph 的克隆检测工具有 Duplix[41]等.

基于 Metric 的克隆检测方法是先将源代码转换为某种中间表示,然后在其基础上提取度量值并抽象为一个特征向量,然后通过计算特征向量的相似度来检测克隆代码[43-44].该方法的主要优点是检测速度快.但因基于 Metric 的方法高度依赖于 Metric 的提取,在对源码提取度量值的过程中会损失源码的部分语义信息,因此检测效果不够理想,使其应用受限.

此外,部分克隆检测方法会结合两种或两种以上检测技术来检测克隆代码.例如,Deckard[28]在生成抽象语法树的基础上,提取结构特征向量表示源代码,然后使用聚类的方法寻找相似的结构向量,进而检测系统中的克隆代码.

### 5.2 克隆检测方法和工具的评估

不同的克隆检测方法和检测工具有其各自的特点,适合于检测不同类型的克隆代码,对同一类型的克隆代码的检测效果也不尽相同.如何选择合适的克隆检测方法和工具检测系统中的克隆代码是困扰开发人员的一个问题.因此,研究人员对主流的克隆检测方法及其检测工具进行评估,以期为用户选择克隆检测方法和工具提供一些建议.Bellon 等人最初对 6 个克隆检测工具进行了评估,分别对比了查准率、查全率以及时间和空间等性能[45].然后,Rattan 等人使用一种标准的文献综述方法,详细分析了克隆代码检测方面的 213 篇文献,对不同的检测方法进行评估分析,并给出了未来的研究方向[46].此外,Svajlenko 等人重点分析了检测工具的使用技术和适用环境,对克隆检测工具的检测效果进行了详细的对比,对帮助用户选择和使用克隆检测工具有重要的指导意义[47].

本文也对目前较为主流的克隆检测工具和方法所支持的克隆类型和检测效果进行了评估,评估结果如表 1 所示.其中,检测效果采用"较好"、"一般"和"较差"三种级别来评估:"较好"是指可以较好地支持该类型克隆代码;"一般"是指可以支持检测该类型克隆代码,但效果不佳;"较差"是指可以检测少部分的该类型克隆代码;对未支持的克隆类型未列出.

从表 1 可以看出,基于 Text 的检测工具不支持 Type 4 克隆的检测.对 Type 1 克隆的检测效果最好,对 Type 3 克隆的检测效果一般.而对 Type 2 克隆支持较弱,仅有两个工具可以支持.原因是 Type 2 克隆是标识符重命名的克隆代码,基于 Text 的方法不能很好地处理标识符重命名问题.基于 Token 的检测工具同样不支持 Type 4 克隆的检测,但是可以较好地检测 Type 1 和 Type 2 克隆.可以支持 Type 2 克隆检测的原因是将源程序转换成 Token

① Tool Simian〈http://www.harukizaemon.com/simian/index.html〉(accessed December 2015)
② Tool SimScan〈http://www.blue-edge.bg/download.html〉(accessed December 2015)

计算机学报

序列时进行了词法分析，因此可以解决标识符重命名的问题.但仅有一个工具可以检测 Type 3 克隆代码，并且检测效果一般.基于 Tree 的检测工具，同样不支持 Type 4 克隆的检测，但是对 Type 1 克隆的检测效果较好，并且几乎都支持 Type 2 和 Type 3 克隆的检测.由于采用的匹配算法不同，对近似克隆(Type 2 和 Type 3 克隆)的检测效果不尽相同，有些可以较好地支持 Type 2 克隆的检测，有些则较好地支持 Type 3 克隆的检测.基于 PDG 的检测方法不仅可以较好地检测 Type 1 和 Type 2 克隆，还可以以不同的程度支持 Type 3 和 Type 4 克隆的检测.但因其复杂度相对较高，使其并没有太多的检测工具可以利用.基于 Metric 的检测方法目前仅有一些检测方法被提出[43-44]，也缺少相应的检测工具.

**表 1　主流克隆检测工具与方法的评估**

| 类型 | 方法 | 描述 | 支持的类型 | 效果 |
|---|---|---|---|---|
| Text | Duploc[33] | 哈希每一行的字符串，并使用点图可视化其差异 | 1、3 | 较好 1，一般 3 |
| | Simian | 灵活的文本比较代码(如忽略标识符) | 1、2 | 较好 1，一般 2 |
| | DuDe[35] | 使用 scatter plot 检测克隆代码 | 1、3 | 较好 1，一般 3 |
| | SDD[36] | 使用结构化数据和索引技术检测克隆代码 | 1、3 | 较好 1，一般 3 |
| | NiCad[3] | 使用 txl 标准化代码，并灵活的检测克隆代码 | 1、2、3 | 较好 1、2、3 |
| Token | Dup[37] | 使用后缀树 Token 化每行代码 | 1、2 | 较好 1、2 |
| | CCFinder[4] | Token 化代码，并使用后缀树搜索 | 1、2 | 较好 1、2 |
| | CP-Miner[38] | 数据挖掘方法寻找频繁 Token 序列 | 1、2、3 | 较好 1、2，一般 3 |
| | iClone[26] | 使用后缀树标准 Token 序列的比较 | 1、2 | 较好 1，2 |
| Tree | CloneDr[39] | 哈希语法树，树比较算法 | 1、2、3 | 较好 1、3，一般 2 |
| | SimScan | 从 ANTLR 中进行子树比较 | 1、2 | 较好 1、2 |
| | Deckard[28] | 从语法树提取度量，并 hashing 度量值 | 1、2、3 | 较好 1、2，一般 3 |
| | CloneDigger[40] | XML 表示抽象语法树，并从进行子树比较 | 1、2、3 | 较好 1、3，一般 2 |
| PDG | Duplix[41] | 搜索相似子图 | 1、2、3、4 | 较好 1、2，一般 3，较差 4 |
| | Gabel[42] | 映射 PDG 的子图 | 1、2、3、4 | 较好 1、2、3，一般 4 |
| Metric | Kontogiannis[43] | 比较函数或代码块的度量值 | 1、2、3、4 | 较好 1、2，较差 3、4 |
| | Mayrand[44] | 比较函数或代码块的度量值 | 1、2、3、4 | 较好 1、2，较差 3、4 |

可见，目前克隆检测方法对 Type 1 和 Type 2 克隆的检测支持最好，检测效果最好，检测工具也最多；检测 Type 3 的克隆代码相对于前两种有一定的难度，因此效果一般，但也有少量工具支持；对 Type 4 克隆的检测难度最大，因此效果最差，工具最少.此外，基于 Text、基于 Token、基于 Tree 的克隆检测方法实现较为容易，是目前较为主流的方法，相应的检测工具也较多.基于 PDG 的克隆检测方法受到图匹配算法复杂性的影响使得这类方法的实现难度较大，因此实用工具不多.而基于 Metric 的克隆检测因为检测效果不够理想，所以研究较少，也没有切实可用的检测工具.

目前绝大多数的克隆检测工具仅支持单版本的克隆代码检测，无法同时检测多个版本中的克隆代码.同时，目前也仅有少数的克隆检测工具(如 iClone)采用增量式的克隆代码检测方法，即在旧版本克隆检测的基础上对新版本进行克隆代码检测，可以节约克隆检测时间，提高检测效率.此外，目前绝大部分的克隆检测工具也没有集成到软件开发环境中，开发人员无法在开发过程中实时地检测新产生的克隆代码.

因此，研究如何提高 Type3 和 Type 4 克隆的检测效果，以及如何在软件开发过程中增量式地检测多版本的克隆代码，这既是一个难点问题，也是未来的一个研究重点问题.

## 6　克隆分析

克隆分析是指使用各种技术手段分析系统中的克隆代码，并挖掘其隐含的特征，旨在帮助软件开发人员更好地理解克隆代码.目前的克隆分析研究主要集中在克隆演化分析、克隆评价分析和克隆可视化分析等方面.

### 6.1　克隆演化分析

克隆代码存在于软件系统的多个版本中，并随着软件系统进行演化.克隆代码演化分析就是通过分析克隆代码的演化过程，提取克隆代码的演化特征，从而识别克隆代码的演化规律，辅助人们更好地理解和维护克隆代码.克隆演化研究包括克隆演化过程分析和演化特征分析两个方面.克隆演化过程分析即模型化克隆代码的演化过程.克隆演化特征分析是分析克隆代码在演化过程中表现出来的演化特征或演化模式及其对软件质量的影响.

克隆演化过程分析最早是 2001 年由 Antoniol

等人[48]提出的,使用时间序列描述克隆代码的演化模型,但并未引起人们的重视. 2005 年,Kim 等人[5]提出了克隆家系模型用于描述克隆代码的演化过程,被认为是迄今为止最好的演化模型,并已成为克隆演化分析的事实标准. 其后,人们对克隆家系进行了大量的研究. Roy 等人[49]使用函数映射帮助构建克隆家系,并开发了 gCad 克隆家系提取器,大大提升了构建克隆家系的时间效率. Bakota[50]通过映射不同版本的克隆代码分析其演化过程,并提出克隆坏味(Clone Smell)帮助分析克隆代码对系统的影响. 2009 年,Harder 等人[51]对现有的演化模型进行分析,指出通过分析克隆演化特征可以帮助程序开发和维护人员理解和维护克隆代码.

究竟哪些演化特征能真实准确地反映克隆代码的规律,是克隆演化分析的难点问题. 因此目前对克隆演化分析的研究主要集中在研究确定提取克隆代码的哪些特征作为演化特征以及如何提取这些演化特征.

目前常用的克隆演化特征主要包括克隆寿命、克隆稳定性与一致性变化.

克隆寿命是指克隆代码在系统中的存在时间,即生存期. Kim 等人[5]研究发现克隆代码要比非克隆代码更加稳定,同时寿命也更长;进一步对长寿命的克隆代码进行研究后,发现对克隆代码的修改会使得克隆代码的寿命变短[52]. Krinke[53]通过对比克隆和非克隆代码,也发现克隆代码比非克隆代码的寿命更长,并通过对精确克隆和近似克隆的演化分析发现,其在演化过程中所表现出来的共同特点是:尽管克隆代码比率会随着时间而逐渐降低,但克隆代码的存在时间往往都会超过一年[54]. 因此,克隆代码会长时间地存在于系统中,在其生存期间克隆代码往往会发生变化,其变化规律与具体的软件系统相关[55].

相对于克隆寿命而言,克隆稳定性关注的是在克隆代码的生存期发生变化的问题. 被研究者普遍认可的观点是寿命较长的克隆代码是稳定的[9-11],因此不会对系统造成不利的影响,也不会增加系统的维护成本. 但是在克隆代码是否比非克隆代码更稳定这个问题上还存在一定的分歧. Göde 等人[56]研究发现大部分克隆是稳定的,不会发生变化;而 Rahman 等人的研究却发现克隆代码比非克隆代码更容易发生变化,是不稳定的[57]. Mondal 等人[58-59]给出了更为细致的分析结果,即 Type 1、Type 2 克隆是不稳定的,Type 3 克隆是稳定的;并且发现克隆代码比非克隆代码的变化更分散,Type 3 克隆比 Type 1 和 Type 2 克隆的变化更分散. 由此可见,在克隆代码的稳定性特征方面尚未达成共识,仍需要进一步研究.

克隆代码的变化包括一致性变化和不一致性变化. 开发人员遗忘一致性变化将会引发相关的软件缺陷(如标识符重命名缺陷),因此一致/不一致性变化也是克隆演化分析研究中需要关注的特征. Göde 等人[56]研究发现发生一致性变化的克隆代码占克隆代码的比例很小,Krinke[60]进一步发现发生一致性变化和不一致性变化的克隆代码比例大约各占一半,并且大部分发生不一致性变化的克隆代码在后续的演化过程中不会继续发生变化. Mondal 等人[61]研究发现发生一致性变化的克隆代码可能会导致延迟传播现象. 延迟传播是指某一个克隆片段的变化没有立即传播到其所在的克隆组中,而在间隔一定数量的版本后传播,继续发生一致性变化. 研究表明延迟传播在 Type 3 的克隆中出现得更为频繁,软件开发人员应该重点关注 Type 3 克隆代码的变化,以避免引入克隆代码相关的软件缺陷.

表 2 列出了目前对克隆演化特征的研究情况. 由表 2 可以看出,研究者较为关注的克隆演化特征是克隆寿命、克隆稳定性与一致性变化. 值得注意的是上述三个特征并不是相互独立的,克隆寿命会受到克隆稳定性和克隆变化的影响,同时克隆稳定性与克隆变化之间存在对立关系. 对克隆演化特征分析的研究大多属于实证研究,往往会依赖于具体被用于实验分析的软件系统,这就可能导致不同的研

**表 2 克隆演化特征分析**

| 文献 | 克隆模式、特征 | 结论 |
|---|---|---|
| [5] | 克隆寿命/一致性变化 | 观察并分析克隆家系的寿命与一致性变化规律 |
| [52] | 克隆寿命 | 克隆寿命与克隆修改次数、新增和减少有关 |
| [53] | 克隆寿命 | 克隆代码的寿命比非克隆的寿命更长 |
| [54-55] | 克隆比率/寿命/变化规律 | 克隆比率会随时间降低,会存在超过一年,存在期间变化规律往往和具体系统相关 |
| [9] | 克隆稳定性 | 克隆代码比非克隆代码更稳定 |
| [10-11] | 克隆稳定性 | 克隆代码十分的稳定,支持文献[52]观点 |
| [56] | 克隆变化/一致性变化 | 大部分克隆不会发生变化,而一致性变化会更少 |
| [57] | 克隆稳定性 | 克隆会比非克隆更容易发生变化 |
| [58-59] | 克隆稳定性/变化分布 | Type 1、Type 2 是不稳定的,Type 3 是稳定的;发现克隆代码变化比非克隆更加分散,同时 Type 3 克隆比 Type 1 和 Type 2 更加分散 |
| [60] | 一致性变化 | 一半的变化是不一致变化,且发生不一致性变化后,在后续的时间内会保持这种变化(即一致性变化) |
| [61] | 延后传播 | Type 3 更容易发生延后传播,并且也更容易导致缺陷 |

《计算机学报》

究得出不同的结论，例如对克隆稳定性的研究就出现了截然相反的观点.尽管如此，克隆演化特征分析依然可以给开发人员提供有价值的建议.一个普遍的共识是在维护和管理克隆代码的过程中更应关注那些克隆寿命较短、稳定性较差、发生一致性变化的克隆代码.

综上，克隆寿命、克隆稳定性和一致性变化之间究竟存在着怎样的关系，它们对软件质量究竟会产生怎样的影响，还有哪些影响软件质量的克隆代码特征，还有待进一步深入的研究.

### 6.2　克隆评价分析

克隆评价分析主要是分析克隆代码对系统产生的影响问题，以便辅助开发人员更好地理解和维护克隆代码.克隆代码是否有害一直是克隆评价关注的一个热点，也是争论的一个焦点问题.

在研究的初始阶段，人们大多倾向于克隆代码是有害的观点，因而研究的侧重点是克隆代码是否会引发软件缺陷(即克隆代码相关的缺陷分析)，以及克隆代码是否会增大系统的维护代价(即克隆代码的维护代价分析).然而，随着对克隆代码研究的深入，发现虽然克隆代码有时会对软件质量产生一些负面影响，但并不一定都是有害的，从而引发了对克隆代码有害性的讨论(即克隆代码的有害性分析).因此，克隆评价主要包括克隆相关的缺陷分析、克隆维护代价分析、克隆有害性分析等方面.

在早期，有人认为克隆代码是一种最刺鼻的代码坏味，是因为它有可能会引发相关缺陷.因此，人们研究的关注点主要集中在克隆代码的缺陷分析上.Juergens 等人研究发现克隆代码的不一致性变化会引发相应的软件缺陷，从而降低了软件质量[7].Gauthier 等人通过对克隆代码进行安全性分析，在开源软件 Joomla 和 Moodle 中发现了几个潜在的影响软件质量的安全漏洞和缺陷[62].但也有另外一些证据表明克隆代码的不一致性变化并不一定会引发缺陷，因此不会对软件质量产生显著的影响.Bettenburg 等人对不一致性变化是否引发缺陷的研究发现，仅有极少数的不一致性变化会引发缺陷[63].文献[64]的研究则表明 Type 3 克隆代码中大约有 17%的代码含有缺陷，并且 Type 3 克隆不容易发生不一致性变化.文献[65]通过对缺陷密度的分析发现，面向对象程序中的克隆类含有更少的缺陷，并且 Type 3 克隆含有的缺陷最少.因此有人将克隆代码研究应用到缺陷检测中，但没有直接证据证明克隆代码与缺陷密切相关[66-67].因此，克隆缺陷分析并不能直接证明克隆代码是有害的.

除了克隆相关的缺陷分析外，研究人员从维护代价的角度分析了克隆代码对软件产生的影响，即克隆代码维护代价分析.Harder 等人[68]通过实验发现克隆代码的存在并不会增加缺陷修复的时间，但是缺陷未被修复则可能导致维护代价的增加.Lozano 等人[69]通过研究克隆代码的可变性，也发现克隆代码的存在会增加软件维护的代价.Juergens 等人[70]不仅认为克隆代码会增加维护代价，还提出了一个计算模型来计算克隆维护代价.而 Monden 等人[71]的研究则发现包含少量克隆代码的软件模块比不含克隆代码的软件模块更可靠，但含有大量克隆代码的软件模块则正相反，实验表明克隆代码与软件可靠性和维护代价有一定的关联，但这种关联关系并不十分明确.以上维护代价分析的研究表明克隆代码的存在有可能会增加软件的维护代价，但是如何定量地计算克隆代码引起的维护代价仍然是一个尚未解决的问题.

虽然缺陷分析和维护代价分析可帮助评价克隆代码对软件产生的影响，但却不能作为评价克隆代码是否有害的直观证据.正因如此，近些年来，研究人员也展开了克隆代码有害性分析的研究.Kapser 等人通过对比 11 种克隆模式在软件开发和维护过程中的优缺点，并在开源软件 Apache 和 Gnumeric 上进行实证研究，发现在 Apache 中大约有 71%的克隆代码被分类为有益克隆，对系统维护具有积极的影响，是一种合理的存在方式[12,72].因此，人们应当正视在开发过程中克隆代码的长期存在.同时，Kapser 等人[73]通过对 Apache Web Server 中的克隆代码进行分析，研究发现其中一个子系统中聚集了大量的克隆代码；该子系统的克隆代码增加了系统的功能性，对软件开发过程是有益的.Selim 等人[14]使用风险模型判定克隆代码是否有害，研究结果发现克隆代码并不比非克隆代码具有更高的有害风险.Wang 等人[74]使用贝叶斯网络对克隆代码的有害性进行预测，该方法可以辅助开发人员在克隆代码复制和粘贴时决定是否引入该克隆代码.Higo 等人则提出一种提取有问题的克隆代码(有害克隆)的方法，该方法先对程序进行标准化，然后利用检测工具检测克隆代码，最后对检测到的克隆代码进行过滤、合并等操作来提取有问题的克隆代码，并通过在 Linux 内核 2.6.6 上的实验结果表明只有少数克隆代码是有问题的[75].Hordijk 等人[76]提出了一个结构化的证据模型分析克隆代码的有害性，研究结果表明只有少部分的证据证明克隆代码是有害的，并且仍需更多的研究去支撑这一结论.从上述对克隆

代码有害性分析的结果不难得出结论,克隆代码并非都是有害的,但是究竟什么样的克隆代码是有害的,还是一个有待深入研究的问题.

表 3 对克隆评价分析研究进行了分类统计,使用"积极"、"中立"和"消极"三种评价标准对现有的克隆评价分析研究进行了总结."积极"是指克隆代码的存在对系统有积极的影响;"中立"是指克隆代码不会对系统产生影响;"消极"是指克隆代码会对系统产生消极的影响.从表 3 可以看出,大部分研究对克隆代码持积极和中立的态度,仅有少数持消极态度.这从另外一个侧面表明目前对克隆代码是否有害还存在一定的分歧,主要原因是对克隆代码缺少统一的评价标准以及深入的特征挖掘与分析,当然这也是克隆评价分析中的最具挑战性的问题.

**表 3 克隆评价分析**

| | 评价 | 文献 | 结论 |
|---|---|---|---|
| 缺陷分析 | 消极 | [7] | 研究发现不一致变化在克隆中发生较为频繁,同时也会导致相应的缺陷 |
| | 消极 | [62] | 通过对不安全的克隆代码分析,发现了几个安全漏洞和缺陷 |
| | 积极 | [63] | 克隆代码的不一致变化并不会引入缺陷,不会对软件的质量产生影响 |
| | 积极 | [65] | 克隆代码含有更少的缺陷,Type 3 所含有的缺陷最少 |
| | 中立 | [66] | 克隆代码研究应用到缺陷检测中,但没有直接证据证明与缺陷相关 |
| | 中立 | [67] | 通过实验发现并不会增加缺陷修复时间,但是缺陷未被修复,可能会产生维护代价的增加 |
| 维护代价分析 | 中立 | [64] | Type 3 克隆中有 17% 含有缺陷,并且 Type 3 克隆更容易发生一致性变化 |
| | 消极 | [68] | 通过对克隆代码的可变性研究发现克隆代码确实会增加其方法可变化的维护代价,但没有确切的证明会说明克隆会增加系统维护代价 |
| | 消极 | [70] | 认为克隆会增加维护代价,并且提出了一个代价计算模型计算这次代价 |
| | 中立 | [71] | Monden 却发现包含克隆的模块比不含克隆的模块更可靠,同时含有大量克隆的模块却相反实验表明了克隆与系统可靠性和维护代价的关系,但是这种关系仍然不够明朗 |
| 有害性分析 | 积极 | [14] | 克隆代码并不比非克隆代码具有更高的有害风险 |
| | 积极 | [12,72] | 发现最多有 71% 的克隆对系统的可维护性具有积极的影响 |
| | 积极 | [13] | 克隆并不是真正的代码坏味,克隆是无害的 |
| | 中立 | [74] | 预测克隆有害性,其中有害的较少,无害的较多 |
| | 中立 | [75] | 使用分类模型抽取有问题的克隆代码,只有少部分是有问题的克隆 |
| | 中立 | [76] | 现有研究不能支撑克隆有害的观点,仍需进一步的研究 |

### 6.3 克隆可视化

克隆检测工具通常使用文本文件来保存克隆代码的检测结果.然而,使用文本形式保存克隆检测结果,既不形象直观,也不利于深入分析克隆代码的分布情况和关联关系.为了解决这一问题,研究人员对克隆代码进行了可视化研究.克隆可视化是使用可视化技术将克隆代码在软件中的分布情况以及克隆关系以图形的方式展现出来,以辅助开发人员更加方便直观地理解、分析和维护克隆代码.

目前使用较多的可视化技术主要有捆绑图、散点图和树图.捆绑图从宏观的角度展示克隆代码所在的文件之间的克隆关系以及克隆关系的强弱,一般以克隆文件为粒度,采用同心圆环、扇形和曲线相结合的形式可视化克隆代码[77-79].例如可视化工具 ClonEvol 使用分层捆绑图分析软件的演化情况,以辅助开发人员寻找自己感兴趣的克隆代码[77].Hauptmann 等人[78]则使用捆绑图分析软件各模块之间的耦合关系.散点图从微观的角度使用矩阵的形式对软件中克隆代码的分布情况进行可视化[80-82].Cordy[80]使用散点图分析软件之间的相似性,Higo 等人[81-82]则使用散点图分析软件中的克隆模式.树图以二维矩形图的形式对克隆代码在软件中的分布情况和层次结构进行可视化,以树图形式可视化克隆代码的工具有 VisCad[83]、SoftGUESS[84].

克隆可视化的作用不仅仅是可视化克隆代码,更重要的作用是通过克隆代码的可视化辅助软件开发人员分析克隆代码对软件产生的影响.例如,可视化工具 Doppel-Code 可以分析克隆代码对系统全局和局部产生的影响,并可以将不同克隆代码组对系统的影响进行排序[85].Jiang 等人[86-87]提出一个克隆可视化框架,辅助开发人员分析系统内各模块间的耦合和内聚情况.Zhang 等人[88]通过对结构克隆进行过滤和可视化分析,帮助维护人员寻找对开发人员有用的克隆代码,以便对克隆代码进行复用.

不同的可视化技术从不同的角度对克隆代码的分布情况或克隆关系进行可视化.但是,每种技术都各有其侧重点和局限性,而且可视化仅仅是辅助分析的一种手段,不是分析的目的,最终还是要靠人来分析这些可视化的结果.因此,如何解读克隆代码的可视化结果,辅助开发和维护人员分析或维护克隆代码是克隆可视化分析研究的一个难点问题.

## 7　克隆维护

克隆维护是解决克隆代码问题的最直接途径，指的是主动地解决克隆代码已经或者可能引发的问题. 克隆维护研究与软件开发过程结合得较为紧密，维护的方法主要包括克隆重构、克隆规避、克隆复用等.

### 7.1　克隆重构

重构作为一种常见的软件维护手段，是指在不改变软件外部行为的条件下改善软件的既有设计[6]. 最开始人们认为克隆代码是有害的，因此将重构方法应用于克隆维护中，通过重构手段消除系统中的克隆代码.

由于重构所需的条件较为苛刻，重构的代价也较大. 因此，在重构前对克隆代码进行可重构性分析和重构排序显得尤为重要. 可重构性分析旨在识别适于重构的克隆代码候选集合，可以提高克隆代码的重构效率. Lin 等人[89]通过对克隆代码进行差异性分析，帮助开发人员决定是否进行重构操作和如何执行重构操作. Mende 等人[90]实现了一个工具支持可重构性分析，在软件中识别可以被重构的函数克隆. Schulze 等人[91]提出了一个用于识别可重构的克隆代码的克隆分类方法，通过计算克隆代码的可重构指数识别可重构的克隆代码. Choi 等人[92]提出了基于度量值的可重构性分析方法，可以快速地识别可重构的克隆代码. 识别可重构克隆后，为了节省重构时间，还可以对克隆代码进行重构排序或者重构调度. 如 Mandal 先通过分析克隆演化模式来确定克隆候选，然后使用关联规则挖掘进行可重构排序[93-94]；Lee 等人[95]采用遗传算法对重构进行调度，以确定重构顺序帮助改进软件质量；Zibran 等人[96]提出一种极限编程方法对克隆代码进行重构调度. Radhika 等人[97]考虑重构代价对所有的克隆代码进行重构排序；但是，其未考虑克隆代码是否适合重构的问题，也应该结合可重构性分析指导克隆重构. 此外，Liu 等人[98]还提出了一个重构调度方法帮助优化调度过程. 尽管该方法不是针对克隆重构的，但也可以考虑将其应用于克隆代码的重构调度上.

克隆重构最主要的目的是消除系统中的克隆代码，因此人们提出了几种克隆代码重构的方法. Higo 等人[15]开发了一个名为 ARIES 工具，可以根据克隆代码的结构信息识别可重构的克隆代码，然后通过计算不同的度量值来确定使用哪一种目前已有的重构方法移除克隆代码. Krishnan 等人[16]通过分析克隆代码的程序依赖图，确定给定克隆代码能否被安全地重构，然后通过检测和参数化克隆代码的差异点对克隆代码进行重构，也取得了较好的效果. Barbosa 等人[99]使用四种规则重构克隆代码，并在开源软件 JhotDraw 上进行了实验，结果表明基于规则的重构方法也可以有效地移除软件中的克隆代码.

在重构完成后，对重构后克隆代码进行分析可以得出一些结论，从而进一步指导克隆重构. Göde[100]通过对维护人员使用的重构方法和被重构的克隆代码进行实证研究，发现在不同的软件系统中都存在克隆重构的行为，并且克隆重构不是经常性地发生而是有选择性地发生. Zibran 等人[101]通过克隆重构研究，回答了所提出的关于克隆重构的七个研究问题，并且研究发现克隆规模对克隆重构没有重要的影响，同时在软件早期的版本中重构会较为频繁地发生，发生重构的克隆在重构前往往是较为稳定的克隆代码. Choi 等人[102]通过对重构行为进行观测，识别出几种较为频繁的克隆重构模式，并且分析了每种重构模式的特征，这些都有助于进一步指导克隆代码的重构. Tairas 等人[103]通过对克隆代码的重构性分析，提出了子克隆的概念，并研究发现子克隆的重构行为更容易发生，因此建议在对克隆代码的维护过程中应重点考虑子克隆的重构.

目前，虽然一些重构方法可以以插件的形式集成到 eclipse 中①，支持在软件开发过程对一般的代码进行重构. 但是由于一般的重构方法不一定适合克隆代码的重构，因此也可以考虑将适合于克隆重构的方法集成到软件开发环境中. 此外，重构是在克隆代码出现后用以消除克隆代码的一种被动的软件维护方法. 但是，并非所有的克隆代码都适合重构，重构并不是解决克隆代码维护难题的最有效途径，更有效的方法是在软件开发过程中主动地规避、维护和管理克隆代码，这是维护系统中克隆代码的一个难点问题.

### 7.2　克隆规避

克隆规避也称克隆预防，是在克隆产生时通过某种策略或手段对其干预，以避免克隆代码的产生. 克隆规避是一种预防性的克隆维护方法，通过阻止克隆代码产生来降低克隆维护的代价.

① https://www.eclipse.org/articles/Article-LTK/ltk.html，一个支持重构功能的工具包. 以插件的形式集成到 eclipse 中支持代码重构操作.

《计算机学报》

软件开发人员的复制粘贴活动是导致克隆代码产生的最主要原因.因此在复制粘贴时对新产生的克隆代码进行分析,帮助软件开发人员决定是否规避克隆代码,是目前最常见的克隆规避方法.Ali 等人[104-105]曾提出一个克隆规避的概念模型,对克隆规避研究具有一定的指导意义,但并未给出具体实现方法.Wang 等人[74,106]基于贝叶斯网络在复制粘贴时预测克隆代码的有害性,通过判断其是否会导致一致性变化决定是否该规避新产生的克隆代码.Ravikanth 等人[107]通过分析复制粘贴操作的前置和后置条件,来确定是否允许对被复制的克隆代码实施粘贴操作,从而实现克隆规避.相对于其它的克隆研究而言,目前对克隆规避技术的研究还相对较少,克隆规避技术也不够完善.

由于受软件复用的影响和编程语言的限制,开发人员的大部分复制和粘贴操作(克隆代码)是无法避免的.克隆规避可以作为一种辅助的手段通过限制开发人员在软件开发过程中的复制粘贴操作帮助减少克隆代码的产生,但却无法规避全部的克隆代码.在实际的应用中,还需要将克隆规避和克隆管理相结合,对可以规避的克隆代码采用规避技术减少克隆代码的产生,对无法规避或者无需规避的克隆代码进行跟踪管理,主动地维护克隆代码.

### 7.3 克隆复用

在软件开发过程中,复用既有代码是一种常见的软件开发手段.近些年来,随着对克隆代码评价分析研究的深入,人们已经意识到复用健壮性好的克隆代码不仅有助于缩短软件开发周期,还有利于提高软件的健壮性和可维护性.因此,对克隆复用的研究在近些年来也引起了人们的关注,成为克隆代码领域中的一个新的研究方向.不过,相对于克隆检测和分析而言,目前对克隆复用的研究相对较少.

Krutz 等人[108]提出克隆数据库概念,将已有克隆代码组织为克隆数据库,并结合代码检索技术实现对克隆代码的复用.Ishihara 等人[109]提出的方法也允许开发人员通过代码搜索技术搜索可以复用的克隆代码.Ohta 等人[110]通过对比软件开发过程中由复制粘贴操作产生的克隆代码和系统中已存在的克隆代码,并将该克隆代码划分为较差、较好、最好三种情况,辅助开发人员决定是否能够复用该克隆代码.但是,该方法对克隆代码的可复用性评价仅依赖于开发人员的复制粘贴操作,并且仅在复制粘贴操作发生后给出可复用性建议,无法在复制粘贴操作发生前为开发人员提供可复用的克隆代码信息.因此,在软件开发过程中直接向开发人员提供一个已经通过某种评价方式确定是否可复用的克隆代码库,是一种更为主动的克隆代码复用形式.例如,Yang 等人[111]使用机器学习模型对克隆代码进行分类,分类标准可以由开发人员动态地标记,通过标记可复用的克隆代码将该方法应用于克隆复用,辅助开发人员搜索可以复用的克隆代码.Ohtani 等人[112]从代码建议的视角辅助开发人员复用已有的代码,从不同粒度(关键字级别、方法级别和混合方法)对可复用的代码提供搜索支持,实验结果表明混合方法可以更精确地提供代码是否可复用的建议.Kintab 等人[113]实现了一个克隆代码的专家推荐系统,可以在线向软件开发人员提供复用克隆代码的相关咨询,软件开发人员可以根据专家的建议决定如何复用和维护既有的克隆代码.

软件复用在提高软件开发效率的同时,势必也会导致产生大量的克隆代码,这就使得克隆复用研究面临着一些新的问题.例如,如何识别有益的克隆代码作为可复用克隆代码候选,并且如何实时地跟踪、维护和管理一个可复用的克隆代码库.因此,在保证软件开发质量的前提下,主动地复用既有代码,以提高软件开发效率,既是一个新问题,也是一个难点问题.

现有的克隆代码复用研究大都是针对单一项目的.如何实现对跨项目的克隆代码复用是克隆复用研究面临的另一个新问题.目前,在互联网环境下,大量的开源社区里也普遍存在代码复用的情况,由此也产生了跨项目克隆代码以及对跨项目代码复用的应用需求.但是目前对这种跨项目的克隆代码复用的研究依然较少,仅仅出现了少量的一些跨项目克隆代码研究.Ishihara 等人[114]对跨项目的软件进行函数克隆检测,试图建立一个公用函数库用于代码复用.Cheng 等人[115]研究了跨项目的克隆代码检测技术.Zhang 等人[116]使用代码搜索技术进行相似代码匹配.Tairas 等人[117]将信息检索技术与克隆代码分析相结合,以便快速有效地搜索可复用的克隆代码.事实上,以上这些技术都可以用于跨项目的克隆代码搜索和复用中,这可能是未来的一个研究内容.

## 8 克隆管理

克隆代码的重构是在克隆代码产生之后消除克隆代码及其对系统产生的影响,这种维护方式属于

一种较为被动的维护方式，其维护代价较高. 同时，克隆代码往往会随时间推移、需求变更而在软件系统中发生变化，这势必进一步增大克隆代码维护的代价. 如果可以将对克隆代码的维护过程提前到软件开发阶段，在软件开发过程中跟踪和管理克隆代码及其变化(例如主动地跟踪新产生的克隆代码)，则势必会降低后期维护的代价. 若要实现这种主动的维护方式，则需要从克隆代码管理的角度出发去解决克隆代码的维护难题，即通过在软件开发过程中实现边开发、边管理克隆代码，以主动的方式维护克隆代码，从而消除克隆代码对软件系统的不利影响.

### 8.1　克隆管理的提出

克隆代码管理(简称克隆管理)的概念最早是1997 年由 Koschke 提出的[118]，但当时并未引起人们的重视. 随着克隆代码规模的逐渐增大，人们发现现有手段无法有效解决克隆代码维护难的问题，才又开始把目光重新转向对克隆管理的研究.

目前克隆管理被认为是解决维护克隆代码难题最有效的方式. 早在 2012 年面向工业界的克隆管理会议(Software Clone Management Towards Industrial Application)中，就有专家学者指出克隆管理是未来的重要研究方向，如何将克隆管理与软件开发过程相结合并使之适用于工业界是目前克隆研究领域急需解决的问题[19]. 目前，人们对克隆管理的研究仍处于起步和理论研究阶段，罕有从管理的角度维护克隆代码的研究报道.

Koschke[118]将克隆管理划分为预防性、补偿性和改正性三种类型. 预防性克隆管理的目标是避免克隆代码，关注于避免新的克隆代码产生，而不是对已有克隆代码的管理. 补偿性克隆管理的目标是发现克隆代码所引发的问题，并对其对系统造成的不利影响进行补偿. 改正性克隆管理的目标是以主动的方式消除可能对系统产生不利影响的克隆代码. 然而，Koschke 对克隆管理的这种划分方法的理论意义大于实际意义. 目前克隆管理研究主要集中于补偿性克隆管理，而改正性克隆管理尚未具体应用到实际的克隆管理中.

Roy 等人在文献[20]中也对克隆管理研究进行了详尽的阐述，同样也指出了目前对克隆管理的研究依然较少，尚缺少有效的克隆管理方法，尤其是对 Type 3 和 Type 4 克隆代码的研究还不够充分，未来应针对这两种类型的克隆代码进行深入研究. 尽管 Roy 等人对克隆管理的分析涵盖了克隆代码领域的大多数研究内容，但只是以工作流的形式组织这些研究内容，并未考虑这些研究内容之间的联系，导致彼此之间信息相互割裂，同时也没有结合软件开发过程分析对克隆管理的要求，更没有给出具体的克隆管理方法.

### 8.2　目前的克隆管理研究

克隆代码管理是指从管理的角度管理和维护系统中的克隆代码，克隆管理的主要目的是降低维护克隆代码代价. 目前，克隆管理的研究内容主要包括克隆跟踪、克隆变化管理以及克隆管理工具. 其中，克隆跟踪是研究跟踪克隆代码产生和变化的方法. 克隆变化管理是研究对克隆代码的变化进行管理的方法. 克隆管理工具是现有可以实现管理克隆代码的工具.

克隆管理需要解决的首要问题是如何实时地跟踪系统中的克隆代码，包括克隆代码的产生以及克隆代码的变化. 由于复制粘贴操作是导致克隆产生的主要原因，因此对克隆代码产生的跟踪可以通过监测程序员的复制粘贴操作来实现. 例如，许多克隆跟踪工具 CLONEBOARD[119]、CnP[120]、CPC[121]、CReN[122]、CSeR[123]等可以在软件集成开发环境中跟踪克隆代码的产生. 另一方面，在软件开发过程中克隆代码可能随时发生变化，因此要实现对克隆代码的管理，不仅要跟踪克隆代码的产生，还要跟踪克隆代码的变化. Duala-Ekoko 等人使用克隆区域描述符生成克隆代码的上下文信息，然后根据这些上下文信息可以实时跟踪克隆代码的变化[30]. 该方法仅通过实验验证可以跟踪克隆代码的变化，但是并没有实现对克隆代码的管理，也没有集成到软件开发环境中对克隆代码的边开发、边管理.

为实现边开发、边管理克隆代码，在跟踪克隆代码的基础上，还需对克隆代码的变化进行维护和管理. Yamanaka 等人[124]将克隆变化与事件通知机制相结合，将每一个克隆变化都看成一个事件，在克隆发生变化时提醒开发人员及时地对克隆变化进行维护. 克隆代码的变化往往会引发一致性变化，因此 Cheng 等人[125]提出了一个基于 Token 的克隆代码一致性维护方法，当克隆组内的一个代码发生变化时，能够同时修改组内的其他克隆代码，保证克隆代码的一致性. 但是，该方法的前提是已知克隆变化需要一致性维护，却没有给出克隆变化是否需要一致性维护的判定方法. 因此，Zhang 等人[126]在克隆代码发生变化时预测克隆代码一致性维护需求. 该方法通过提取克隆代码的演化信息及其相关特征，在克隆代码发生变化时辅助开发人员确定克隆变化是

否需要一致性维护. Mondal 等人[127]通过对克隆组内的克隆代码排序,从而预测需要一致性维护的克隆代码. 此外,Murakami 等人[128]在克隆代码未发生变化时,预测克隆代码的下一次变化. 该方法通过分析历史版本中克隆代码的变化情况,提取相关的特征进行变化预测,以便在软件开发过程中提醒开发人员对有可能发生变化的克隆代码进行维护. 可见,克隆变化管理的关键不仅需要实时跟踪克隆代码的变化,更重要的是对发生变化的克隆代码进行及时的维护.

前面的研究只是针对不同的侧重点实现了克隆管理的某一个方面,却没有提供可以实际应用的克隆管理工具. 鉴于此,Zhang 等人[129]基于事件通知机制实现了一个克隆管理工具,不仅可以监测克隆代码的产生,还可以监测克隆代码的维护过程. Nguyen 等人[130]实现了一种可用于管理克隆代码的 eclipse 插件 JSync. 该插件支持在软件开发过程中对克隆代码进行克隆关系管理和一致性维护管理. 克隆关系管理是指在软件开发过程中检测系统中的克隆代码,并对具有克隆关系的代码进行管理. 一致性维护管理是指识别变化的克隆代码以及变化的克隆代码是否会导致不一致性缺陷,以便开发人员对克隆代码进行一致性维护. 而另一个较早提出的 eclipse 插件 CeDAR[131]虽然没有强调是一种克隆管理工具,但事实上也具有一定的克隆管理功能. CeDAR 将克隆检测和克隆重构融为一体,对现有克隆检测工具的克隆检测结果进行可重构性分析,寻找潜在的可重构的克隆代码,然后在软件开发过程中实现对克隆代码的重构.

与此同时,对克隆代码的管理也需要考虑其它的克隆研究,需在其它研究的基础上实现对克隆代码的管理. 因此,如何将克隆代码的检测、分析和维护与克隆管理结合为一个统一整体,并结合软件开发过程,以实现对克隆代码的边开发、边维护和边管理是克隆管理亟需解决的一个难点问题.

## 9 目前研究的不足与难点分析

目前,人们对克隆代码的研究主要是从两个角度进行的. 第一个角度是从克隆代码类型出发,即研究不同类型的克隆代码. 其中,对近似克隆的研究较多,而对语义相似的克隆研究较少,同时也存在少量的对结构克隆、模型克隆的研究. 另一个研究角度是从研究的目的和应用的角度,即克隆检测、克隆分析、克隆维护和管理等具体的研究内容和目标. 其中,对克隆检测和分析的研究较多,也较为深入;而对克隆维护的研究还不够深入,对克隆管理的研究则是刚刚起步.

为了便于分析目前研究的不足与难点,本文给出了现有的克隆代码的研究内容及其关系,如图 4 所示. 现有对克隆代码的研究可以划分为克隆检测、克隆分析、克隆维护和克隆管理四个阶段.

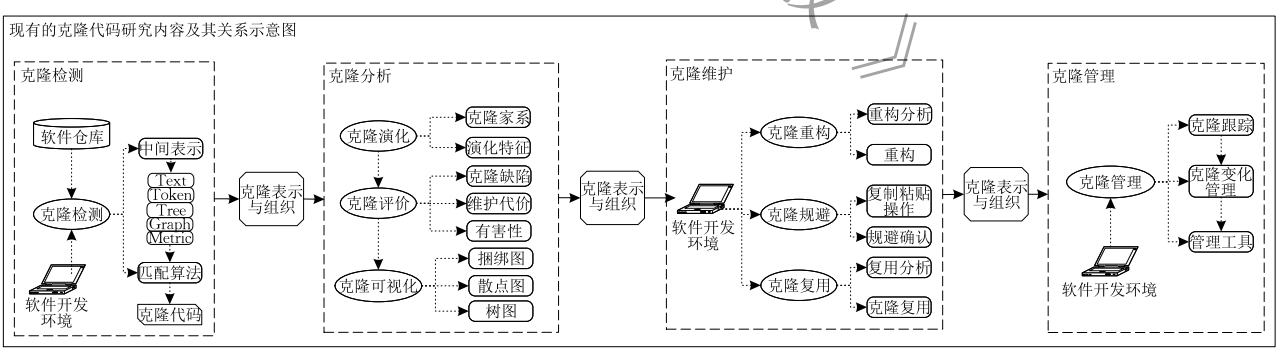



图 4 现有的克隆代码研究内容及其关系示意图

克隆检测是第一阶段的研究内容,是所有其它克隆研究的基础,其主要任务是从软件系统中检测出克隆代码. 目前的克隆检测方法可以有效地检测 Type 1、Type 2 和 Type 3 的克隆代码,但是对 Type 4 克隆代码的检测方法研究较少,也缺少对 Type 4 克隆的检测工具. 同时,现有的克隆检测关注于如何检测克隆代码并报告其出现的位置,并没有将克隆检测结合到后续的克隆分析、维护和管理过程中. 更重要的是,大多数的克隆检测方法也没有和软件开发环境相结合,仅有极少数的工具实现了这一功能,如 CloneDetective[132]、SimScan 和 SHINOBI[133]①.

① ConQAT 是集成的软件工具包,基于这个软件工具包实现了克隆检测工具 CloneDetective,可与软件开发过程相结合在 IDE 中检测克隆代码. SimScan 是作为插件集成到 eclipse、IDEA 和 Jbuilder 等软件开发环境中的克隆检测工具. SHINOBI 是集成到 Visual Studio 2005 中的克隆检测插件,但需要使用 CCFinderX 作为预处理器.

克隆分析处于中间阶段，旨在辅助开发人员理解和维护克隆代码，是克隆研究的重点内容. 其中，克隆分析的目的是辅助开发人员更好地分析和理解克隆代码. 目前的克隆评价方法虽然有助于分析人员分析得出一些有价值的结论，但仍然带有很强的主观性，无法客观全面地评价克隆代码. 与克隆检测一样，克隆评价结果也没有结合到后续的克隆维护和管理过程中，导致评价结果不能被充分利用，实际应用于维护和管理克隆代码中. 例如利用克隆代码的有害性评价结果指导克隆重构，利用克隆代码的可复用性分析结果指导克隆复用等.

克隆维护是处理克隆代码所引发的相关问题的直接途径，其主要任务是通过采取措施维护系统中的克隆代码以及解决克隆代码已经导致的相关问题. 克隆重构作为消除克隆的手段，但由于其重构条件苛刻，仅能消除小部分的克隆代码，并且克隆重构属于一种被动的维护方式，维护代价偏高. 克隆规避是一种较为主动的维护方式，可以通过监测复制粘贴活动从源头上避免克隆代码. 但是，由于多种原因无法规避全部的克隆代码，也无法解决克隆代码的维护难题. 克隆复用作为一种常用的可以帮助提高软件开发效率的手段，依然缺少成熟有效的复用模型和方法. 同时，克隆重构和克隆复用往往需要克隆分析结果的支持，如克隆评价结果指导克隆重构和克隆复用，然而遗憾的是目前的克隆维护并没有与克隆分析相结合.

克隆管理处于克隆研究的最后一个阶段，是解决克隆代码问题的最终途径. 尽管克隆管理已经引起了人们的关注，也有了一些研究成果，如通过跟踪克隆代码及其发生的变化对克隆代码进行管理. 但是，目前克隆管理方法仅具备一些初步的克隆管理功能，既没有与其它研究内容充分结合，也没有完整且有效的克隆管理方法. 更为重要的是，目前克隆管理与软件开发过程结合得也不够紧密，未能实现边开发、边维护、边管理克隆代码.

除此之外，从图 4 中可以看出克隆表示与组织是链接克隆检测、分析、维护和管理四个阶段的关键环节. 但是，目前研究中依然缺乏统一的克隆表示形式，导致各个研究阶段无法进行充分的信息交互. 因而进一步引发了另一个问题：各个研究阶段之间的联系不够紧密，甚至相互独立. 与此同时，目前克隆研究内容也大多与软件开发过程相脱离，难以有效地实现边开发、边维护、边管理克隆代码. 例如克隆检测不支持软件开发环境；克隆分析中则罕有与开发过程相结合的研究；仅有少量克隆维护和管理方法考虑了与软件开发过程相结合.

综上所述，目前对克隆代码的研究主要存在以下两个难点：

(1) 克隆检测、分析、维护和管理之间没有紧密地联系为一个整体，导致各个阶段内有关克隆代码的信息成为信息孤岛，不能共享和交互，不能为主动地维护和管理克隆代码提供服务和支撑.

(2) 克隆检测、分析、维护和管理与软件开发过程是相互脱节的，不支持在软件开发过程中主动地对克隆代码进行维护和管理，不利于开发人员边开发、边维护、边管理克隆代码.

## 10　未来研究展望

### 10.1　一个面向软件开发过程的克隆管理方法

如前所述，有效的克隆管理要求将克隆检测、克隆分析和克隆维护有机地结合在一起，并且与软件开发过程相结合，以便实现边开发、边维护和边管理克隆代码. 具体目标为：在软件开发过程中实现克隆检测，并实时地跟踪克隆代码的变化、准确客观地评价克隆代码，在此基础上对克隆代码主动地维护和管理.

上述目标就给未来的克隆管理提出了新的挑战. 为了实现这一目标，本文从软件开发过程的角度出发，提出了一个面向软件开发过程的克隆管理方法，如图 5 所示. 该方法的核心和关键是在整个的克隆管理过程中建立、实时更新和维护一个克隆仓库. 因此，需要一个统一的克隆表示和组织方法. 克隆仓库则是表示和组织方法的一个具体实现形式. 克隆仓库有着极为重要的意义，在整个克隆管理过程中起着桥梁和纽带的作用：借助于克隆仓库实现统一的克隆表示与组织方法，不仅可以统一表示克隆检测工具的检测结果，还有助于实现各个不同阶段中克隆代码信息的共享和交互；还可以在软件开发过程中统一地更新、维护和管理克隆仓库，有助于实现对克隆代码的边开发、边维护和边管理.

在图 5 所示的面向软件开发过程的克隆管理中，首先执行克隆代码检测和规避. 克隆检测又可分为初始克隆检测和增量式克隆检测. 初始克隆检测是指第一次对软件系统执行克隆检测，并报告检测结果. 增量式克隆检测是指在初始检测的基础上，通过分析和跟踪代码的变化对克隆代码进行再次检测和更新，所有被检测到的克隆代码都统一存储在克

计算机学报

隆仓库中.

其次,在克隆检测的基础上执行克隆代码分析,包括克隆演化特征分析和克隆评价分析.克隆评价分析的主要目的是区分有益的克隆和有害的克隆.克隆分析是在软件开发过程中对克隆代码进行维护的基础,分析的结果也将保存在克隆仓库中.

最后,在软件开发过程中实时地进行克隆维护和管理,主要包括克隆变化维护、克隆重构和复用等.克隆变化维护是指对软件中已存在的克隆代码进行实时跟踪,分析克隆代码的变化,并进行克隆代码的一致性维护和克隆更新操作.一致性维护是维护克隆代码的一致性,从而避免相关缺陷的产生.克隆更新操作是对变化的克隆代码实时更新并保存到克隆仓库中.与此同时,还要在克隆评价的基础上对克隆代码进行复用维护和重构维护.一方面,对评价为有益的克隆代码允许对其通过复制粘贴操作进行复用,以缩短开发周期.另一方面,结合克隆跟踪管理,对于即将产生的评价为有害的克隆代码禁止复制粘贴操作,以避免有害克隆的产生,对于已经产生的评价为有害的克隆代码,则通过可重构性分析通过重构将其消除.

在整个克隆管理的过程中,都需要克隆代码可视化的支持(克隆可视化存在于整个管理过程中,因此图 5 中未给出可视化模块).克隆可视化包含代码可视化、克隆分析可视化和克隆维护可视化.代码可视化是将克隆代码直观展示给软件开发人员.克隆分析可视化是将克隆分析结果展示给软件开发人员,以辅助其更好地理解克隆代码.维护可视化是以友好的方式在软件开发环境中维护克隆代码,软件开发人员可以全程参与并决定对克隆代码的维护操作.

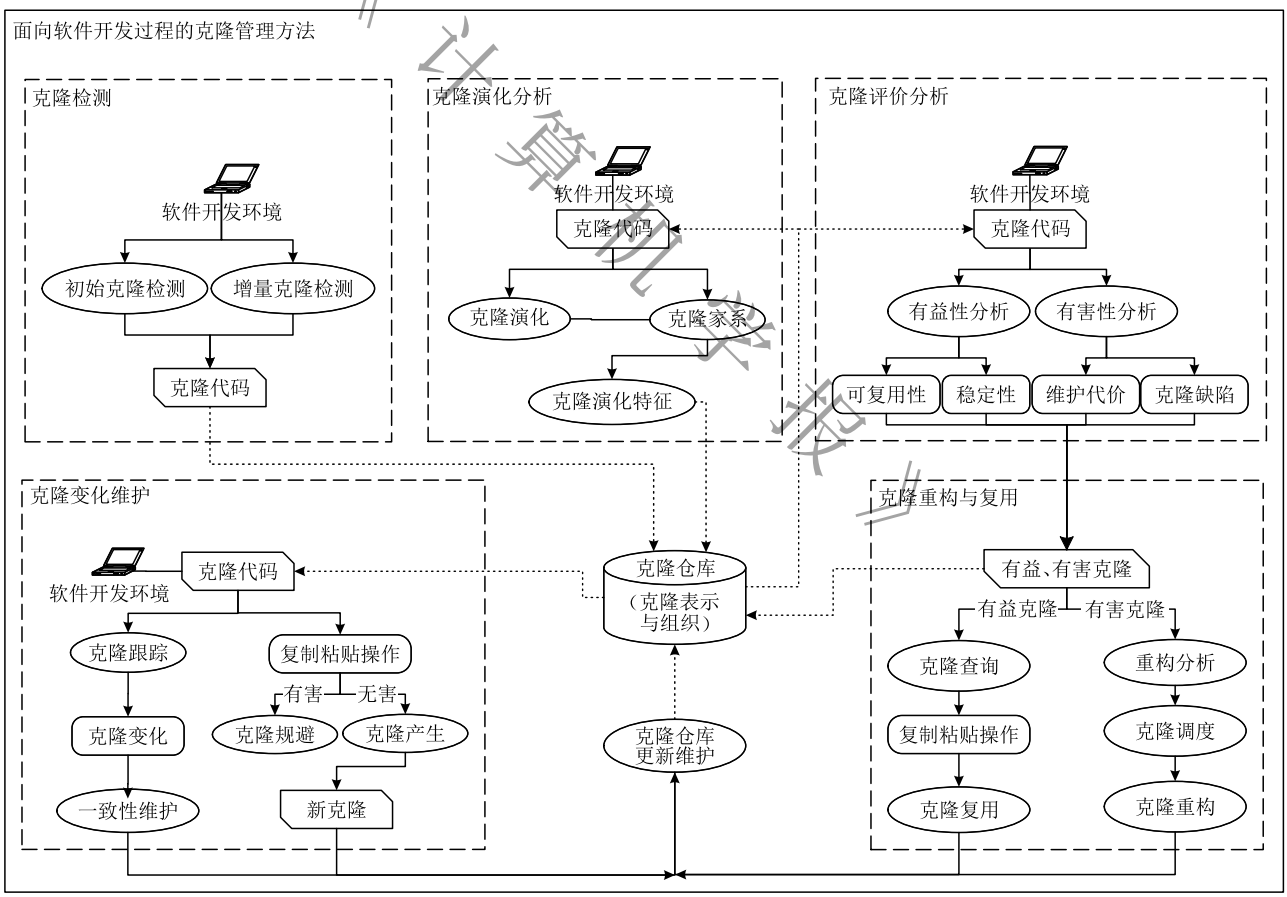



图 5 面向软件开发过程的克隆管理方法

为了在软件开发过程中实现克隆代码的管理,可将上述功能作为插件的组成部分,并集成到软件开发环境中(如 eclipse 中的插件).图 5 所给出的面向软件开发过程的克隆管理方法是一种实现边开发、边维护、边管理克隆代码的理论模型.如何实现这个理论模型还有待今后进一步的深入研究.

### 10.2 未来研究方向与发展趋势

面向软件开发过程的克隆管理对未来克隆研究提出了新的挑战,未来的研究方向与发展趋势主要集中在以下几个方面:

(1)研究统一的克隆表示与组织方法

统一的并具有可扩展性的克隆表示和组织方法是建立、更新和维护克隆仓库的前提，也是实现面向软件开发过程的克隆管理的核心和基础.统一的并具有可扩展性的克隆表示和组织方法，既要能够表示克隆代码的检测信息，又要能够表示克隆代码的分析和评价信息；既要包括当前版本的克隆代码信息，又要包括克隆代码的历史演化信息.在软件开发和克隆管理过程中实现克隆代码信息的共享和交互，从而为克隆维护和管理提供服务和支撑.

(2)研究在软件开发过程中支持实时跟踪和主动维护克隆代码的克隆维护方法

克隆代码会随着系统的演化而发生变化，因此为了实现主动地维护克隆代码，就需要研究如何在软件开发环境中实时地跟踪克隆代码及其变化.这将是主动地维护和管理克隆代码的前提，一方面可以有效避免因克隆变化而引发的缺陷，从源头上解决克隆代码的相关问题；另一方面还可以在克隆代码产生的同时就对其进行维护，是实现主动的克隆维护和管理的有效途径.

(3)研究克隆代码的客观评价方法

克隆评价的结果是否客观准确，直接影响着克隆维护和管理的手段，决定着哪些克隆需要重构，哪些克隆可以复用，因此克隆代码的客观评价是克隆维护和管理的前提和基础.在协同软件开发过程中起着至关重要的作用，不仅有利于降低软件维护成本，还有利于降低软件开发成本.因此，研究克隆代码的客观评价方法，就需要准确分析克隆代码的有害性或有益性.而准确分析克隆代码的有害性或有益性，也需要大量的历史版本信息和代码特征统计数据.因此，可以借鉴其它领域中的方法用于克隆代码的客观评价，如数据挖掘和机器学习方法等.

(4)研究面向软件开发过程的克隆管理方法

目前已有的克隆管理方法无法实现对克隆代码进行边开发、边维护和边管理，为了达到这一目的，就需要研究面向软件开发过程的克隆管理方法.面向软件开发过程的克隆管理方法有两个基本要求：一是与软件开发过程相结合；二是将其与克隆检测、分析和维护等结合，这对克隆管理方法提出了极高的要求.为了实现在软件开发过程中维护和管理克隆代码，可以从软件集成开发环境入手(如 eclipse).在 eclipse 中提供具备克隆管理功能的插件，使之具有良好的人机交互功能，方便软件开发人员在软件开发的过程中跟踪、分析评价和维护克隆代码.

(5)研究跨项目的克隆检测、分析和管理方法

随着开源社区的不断壮大，不同软件项目之间也经常会使用同一函数库，这将导致跨项目克隆代码的产生.但跨项目的克隆检测、分析和管理方法依然罕有报道，未来可能是克隆代码领域的一个研究趋势.跨项目克隆研究中一个可能的热点内容是跨项目的克隆代码复用研究.例如通过检测生成跨项目的克隆代码，并结合克隆代码评价挑选出稳定可靠的克隆代码，来生成高质量的克隆代码复用库.

另外对于缺少相关历史数据的项目而言，跨项目克隆代码研究也可以帮助针对单项目的克隆管理.例如某一软件项目在开发的初始阶段，因缺少历史数据对克隆代码的评价分析带来困难，可以通过跨项目的克隆评价研究帮助解决此问题.但是，如何利用跨项目的数据解决此问题目前尚未见诸报道.不过在软件缺陷预测领域，跨项目缺陷预测是一个主要的研究内容，研究者利用跨项目的数据对某一项目进行软件缺陷预测，也取得了较好的预测效果[134].因此，可以借鉴跨项目缺陷预测研究中的思想，进行跨项目的克隆代码检测、分析和管理的研究.

## 11　结　论

克隆代码在软件系统中大量存在，增加了软件维护的代价，因此对克隆代码的管理显得尤为重要.在软件开发的过程中对克隆代码进行维护和管理，实现边开发、边维护和边管理克隆代码，为降低软件维护的代价、提高软件质量提供了一种有效的解决途径.

本文通过深入调研软件工程方向的期刊和会议，从管理的角度详细分析了克隆代码的研究内容和研究现状，包括克隆检测、分析、维护和管理.在分析现有研究所存在问题的基础上，提出了一种面向软件开发过程的克隆管理方法，以克隆仓库为核心将克隆管理与克隆检测、分析和维护紧密结合为一个整体，并将克隆管理与软件开发过程相结合，以达到边开发、边维护、边管理克隆代码的目的.最后，论文对克隆管理的未来研究方向和发展趋势进行了展望.

## 参　考　文　献

[1] Baxter I D, Yahin A, Moura L, et al. Clone detection using abstract syntax trees//Proceedings of the International Conference on Software Maintenance. Bethesda, USA, 1998:

368-377

[2] Koschke R. Survey of research on software clones//Proceedings of the Dagstuhl Seminar at Schloss Dagstuhl—Leibniz-Zentrum für Informatik. Dagstuhl, Germany, 2007: #06301

[3] Roy C K, Cordy J R. NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization//Proceedings of the 16th IEEE International Conference on Program Comprehension. Amsterdam, Netherlands, 2008: 172-181

[4] Kamiya T, Kusumoto S, Inoue K. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. IEEE Transactions on Software Engineering, 2002, 28(7): 654-670

[5] Kim M, Sazawal V, Notkin D, et al. An empirical study of code clone genealogies//Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering. Lisbon, Portugal, 2005: 187-196

[6] Kerievsky J. Refactoring to Patterns. Boston, USA: Addison-Wesley, 2005

[7] Juergens E, Deissenboeck F, Hummel B, et al. Do code clones matter?//Proceedings of the 31st International Conference on Software Engineering. Vancouver, Canada, 2009: 485-495

[8] Lague B, Proulx D, Mayrand J, et al. Assessing the benefits of incorporating function clone detection in a development process//Proceedings of the International Conference on Software Maintenance. Bari, Italy, 1997: 314-321

[9] Krinke J. Is cloned code more stable than non-cloned code?//Proceedings of the 8th International Working Conference on Source Code Analysis and Manipulation. Beijing, China, 2008: 57-66

[10] Göde N, Harder J. Clone stability//Proceedings of the 15th European Conference on Software Maintenance and Reengineering. Oldenburg, Germany, 2011: 65-74

[11] Harder J, Göde N. Cloned code: Stable code. Journal of Software: Evolution and Process, 2013, 25(10): 1063-1088

[12] Kapser C, Godfrey M W. "Cloning considered harmful" considered harmful//Proceedings of the 13th Working Conference on Reverse Engineering. Benevento, Italy, 2006: 19-28

[13] Rahman F, Bird C, Devanbu P. Clones: What is that smell? Empirical Software Engineering, 2012, 17(4-5): 503-530

[14] Selim G M K, Barbour L, Shang W, et al. Studying the impact of clones on software defects//Proceedings of the 17th Working Conference on Reverse Engineering. Beverly, USA, 2010: 13-21

[15] Higo Y, Kusumoto S, Inoue K. A metric-based approach to identifying refactoring opportunities for merging code clones in a Java software system. Journal of Software Maintenance and Evolution: Research and Practice, 2008, 20(6): 435-461

[16] Krishnan G P, Tsantalis N. Unification and refactoring of clones//Proceedings of the Software Evolution Week—IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering. Antwerp, Belgium, 2014: 104-113

[17] Krutz D E, Mirakhorl M. Architectural clones//Proceedings of the 31st Annual ACM Symposium on Applied Computing. Pisa, Italy, 2016: 1480-1485

[18] Rosendaal F R, Reitsma P H. Copy and paste redeemed. Journal of Thrombosis and Hemostasis, 2013, 11(12): 2083-2094

[19] Koschke R, Baxter I D, Conradt M, et al. Software clone management towards industrial application//Proceedings of the Dagstuhl Reports at Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik. Dagstuhl, Germany, 2012: 21-57

[20] Roy C K, Zibran M F, Koschke R. The vision of software clone management: Past, present, and future (keynote paper)//Proceedings of the Software Evolution Week—IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering. Antwerp, Belgium, 2014: 18-33

[21] Roy C K, Cordy J R, Koschke R. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. Science of Computer Programming, 2009, 74(7): 470-495

[22] Roy C K, Cordy J R. An empirical study of function clones in open source software//Proceedings of the 15th Working Conference on Reverse Engineering. Antwerp, Belgium, 2008: 81-90

[23] Alalfi M H, Cordy J R, Dean T R, et al. Models are code too: Near-miss clone detection for Simulink models//Proceedings of the 28th IEEE International Conference on Software Maintenance. Trento, Italy, 2012: 295-304

[24] Basit H A, Jarzabek S. A data mining approach for detecting higher-level clones in software. IEEE Transactions on Software Engineering, 2009, 35(4): 497-514

[25] Basit H A, Jarzabek S. Detecting higher-level similarity patterns in programs//Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering. Lisbon, Portugal, 2005: 156-165

[26] Göde N, Koschke R. Incremental clone detection//Proceedings of the 13th European Conference on Software Maintenance and Reengineering. Kaiserslautern, Germany, 2009: 219-228

[27] Pham N H, Nguyen H A, Nguyen T T, et al. Complete and accurate clone detection in graph-based models//Proceedings of the 31st International Conference on Software Engineering. Vancouver, Canada, 2009: 276-286

[28] Jiang L, Misherghi G, Su Z, et al. Deckard: Scalable and accurate tree-based detection of code clones//Proceedings of the 29th International Conference on Software Engineering. Washington, USA, 2007: 96-105

[29] Harder J, Göde N. Efficiently handling clone data: RCF and cyclone//Proceedings of the 5th International Workshop on Software Clones. Hawaii, USA, 2011: 81-82

[30] Duala-Ekoko E, Robillard M P. Clone region descriptors: Representing and tracking duplication in source code. ACM Transactions on Software Engineering and Methodology, 2010, 20(1): 3

[31] Harder J. The limits of clone model standardization// Proceedings of the 7th International Workshop on Software Clones. San Francisco, USA, 2013: 10-11

[32] Kapser C J, Harder J, Baxter I. A common conceptual model for clone detection results//Proceedings of the 6th International Workshop on Software Clones. Zurich, Switzerland, 2012: 72-73

[33] Ducasse S, Rieger M, Demeyer S. A language independent approach for detecting duplicated code//Proceedings of the IEEE International Conference on Software Maintenance. Oxford, England, 1999: 109-118

[34] Ducasse S, Nierstrasz O, Rieger M. On the effectiveness of clone detection by string matching. Journal of Software: Evolution and Process, 2006, 18(1): 37-58

[35] Wettel R, Marinescu R. Archeology of code duplication: Recovering duplication chains from small duplication fragments //Proceedings of the 7th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing. Timisoara, Romania, 2005: 157-166

[36] Lee S, Jeong I. SDD: High performance code clone detection system for large scale source code//Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications. San Diego, USA, 2005: 140-141

[37] Baker B S. On finding duplication and near-duplication in large software systems//Proceedings of the 2nd Working Conference on Reverse Engineering. Toronto, Canada, 1995: 86-95

[38] Li Z, Lu S, Myagmar S, et al. CP-Miner: Finding copy-paste and related bugs in large-scale software code. IEEE Transactions on Software Engineering, 2006, 32(3): 176-192

[39] Baxter I D, Quigley A, Bier L, et al. CloneDR: Clone detection and removal//Proceedings of the 1st International Workshop on Soft Computing Applied to Software Engineering. Limerick, Ireland, 1999: 111-117

[40] Bulychev P, Minea M. Duplicate code detection using anti-unification// Proceedings of the Spring/Summer Young Researchers' Colloquium on Software Engineering. St. Petersburg, Russia, 2008: 51-54

[41] Krinke J. Identifying similar code with program dependence graphs//Proceedings of the 8th Working Conference on Reverse Engineering. Stuttgart, Germany, 2001: 301-309

[42] Gabel M, Jiang L, Su Z. Scalable detection of semantic clones//Proceedings of the 30th International Conference on Software Engineering. Leipzig, Germany, 2008: 321-330

[43] Kontogiannis K A, DeMori R, Merlo E, et al. Pattern matching for clone and concept detection. Automated Software Engineering, 1996, 3(1): 77-108

[44] Mayrand J, Leblanc C, Merlo E M. Experiment on the automatic detection of function clones in a software system using metrics//Proceedings of the International Conference on Software Maintenance. Monterey, USA, 1996: 244-253

[45] Bellon S, Koschke R, Antoniol G, et al. Comparison and evaluation of clone detection tools. IEEE Transactions on Software Engineering, 2007, 33(9): 577-591

[46] Rattan D, Bhatia R, Singh M. Software clone detection: A systematic review. Information and Software Technology, 2013, 55(7): 1165-1199

[47] Svajlenko J, Roy C K. Evaluating modern clone detection tools//Proceedings of the International Conference on Software Maintenance and Evolution. Victoria, Canada, 2014: 321-330

[48] Antoniol G, Casazza G, Di Penta M, et al. Modeling clones evolution through time series//Proceedings of the International Conference on Software Maintenance. Washington, USA, 2001: 273-280

[49] Saha R K, Roy C K, Schneider K. An automatic framework for extracting and classifying near-miss clone genealogies// Proceedings of the International Conference on Software Maintenance. Williamsburg, USA, 2011: 293-302

[50] Bakota T. Tracking the evolution of code clones//Proceedings of the 37th Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, 2011: 86-98

[51] Harder J, Göde N. Modeling clone evolution//Proceedings of the 5th International Workshop on Software Clones. Hawaii, USA, 2009: 17-21

[52] Cai D, Kim M. An empirical study of long-lived code clones//Proceedings of the 14th International Conference on Fundamental Approaches to Software Engineering. Saarbrücken, Germany, 2011: 432-446

[53] Krinke J. Is cloned code older than non-cloned code?// Proceedings of the 5th International Workshop on Software Clones. Hawaii, USA, 2011: 28-33

[54] Bazrafshan S. Evolution of near-miss clones//Proceedings of the 12th International Working Conference on Source Code Analysis and Manipulation. Trento, Italy, 2012: 74-83

[55] Göde N. Evolution of Type-1 clones//Proceedings of the 9th International Working Conference on Source Code Analysis and Manipulation. Edmonton, Canada, 2009: 77-86

[56] Göde N, Koschke R. Frequency and risks of changes to clones//Proceedings of the 33rd International Conference on Software Engineering. Hawaii, USA, 2011: 311-320

[57] Rahman M S, Roy C K. A change-type based empirical study on the stability of cloned code//Proceedings of the 14th International Working Conference on Source Code Analysis and Manipulation. Victoria, Canada, 2014: 31-40

[58] Mondal M, Roy C K, Rahman M S, et al. Comparative stability of cloned and non-cloned code: An empirical study// Proceedings of the 27th Annual ACM Symposium on Applied Computing. Trento, Italy, 2012: 1227-1234

[59] Mondal M, Roy C K, Schneider K A. Dispersion of changes in cloned and non-cloned code//Proceedings of the 6th International Workshop on Software Clones. Zurich, Switzerland, 2012: 29-35

《计算机学报》

[60] Krinke J. A study of consistent and inconsistent changes to code clones//Proceedings of the 14th Working Conference on Reverse Engineering. Vancouver, Canada, 2007: 170-178

[61] Mondal M, Roy C K, Schneider K A. A comparative study on the intensity and harmfulness of late propagation in near-miss code clones. Software Quality Journal, 2016, 24(4): 883-915

[62] Gauthier F, Lavoie T, Merlo E. Uncovering access control weaknesses and flaws with security-discordant software clones//Proceedings of the 29th Annual Computer Security Applications Conference. New Orleans, USA, 2013: 209-218

[63] Bettenburg N, Shang W, Ibrahim W, et al. An empirical study on inconsistent changes to code clones at release level//Proceedings of the 16th Working Conference on Reverse Engineering. Lille, France, 2009: 85-94

[64] Wagner S, Abdulkhaleq A, Kaya K, et al. On the relationship of inconsistent software clones and faults: An empirical study//Proceedings of the 23rd International Conference on Software Analysis, Evolution, and Reengineering. Suita, Japan, 2016: 79-89

[65] Elish M O, Al-Ghamdi Y. Fault density analysis of object-oriented classes in presence of code clones//Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering. Nanjing, China, 2015: 10

[66] Lo D, Jiang L, Budi A. Active refinement of clone anomaly reports//Proceedings of the 34th International Conference on Software Engineering. Zurich, Switzerland, 2012: 397-407

[67] Kamei Y, Sato H, Monden A, et al. An empirical study of fault prediction with code clone metrics//Proceedings of the Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement. Nara, Japan, 2011: 55-61

[68] Harder J, Tiarks R. A controlled experiment on software clones//Proceedings of the 20th International Conference on Program Comprehension. Passau, Germany, 2012: 219-228

[69] Lozano A, Wermelinger M. Assessing the effect of clones on changeability//Proceedings of the International Conference on Software Maintenance. Beijing, China, 2008: 227-236

[70] Juergens E, Deissenboeck F. How much is a clone//Proceedings of the 4th International Workshop on Software Quality and Maintainability. Madrid, Spain, 2010: 56-69

[71] Monden A, Nakae D, Kamiya T, et al. Software quality analysis by code clones in industrial legacy software//Proceedings of the 8th Symposium on Software Metrics. Ottawa, Canada, 2002: 87-94

[72] Kapser C J, Godfrey M W. "Cloning considered harmful" considered harmful: Patterns of cloning in software. Empirical Software Engineering, 2008, 13(6): 645-692

[73] Kapser C J, Godfrey M W. Supporting the analysis of clones in software systems. Journal of Software Maintenance and Evolution: Research and Practice, 2006, 18(2): 61-82

[74] Wang X, Dang Y, Zhang L, et al. Can I clone this piece of code here?//Proceedings of the 27th International Conference on Automated Software Engineering. Essen, Germany, 2012: 170-179

[75] Higo Y, Sawa K, Kusumoto S. Problematic code clones identification using multiple detection results//Proceedings of the 16th Asia-Pacific Software Engineering Conference. Batu Ferringhi, Malaysia, 2009: 365-372

[76] Hordijk W, Ponisio M, Wieringa R. Harmfulness of code duplication—A structured review of the evidence//Proceedings of the 13th International Conference on Evaluation and Assessment in Software Engineering. Durham, UK, 2009: 1-10

[77] Hanjalic A. ClonEvol: Visualizing software evolution with code clones//Proceedings of the 1st Working Conference on Software Visualization. Eindhoven, Netherlands, 2013: 1-4

[78] Hauptmann B, Bauer V, Junker M. Using edge bundle views for clone visualization//Proceedings of the 6th International Workshop on Software Clones. Zurich, Switzerland, 2012: 86-87

[79] Voinea L, Telea A C. Visual clone analysis with solidSDD//Proceedings of the 2nd Working Conference on Software Visualization Software Visualization. British Columbia, Canada, 2014: 79-82

[80] Cordy J R. Exploring large-scale system similarity using incremental clone detection and live scatterplots//Proceedings of the 19th International Conference on Program Comprehension. Kingston, Canada, 2011: 151-160

[81] Higo Y, Kamiya T, Kusumoto S, et al. Method and implementation for investigating code clones in a software system. Information and Software Technology, 2007, 49(9): 985-998

[82] Livieri S, Higo Y, Matushita M, et al. Very-large scale code clone analysis and visualization of open source programs using distributed CCFinder: D-CCFinder//Proceedings of the 29th International Conference on Software Engineering. Minneapolis, USA, 2007: 106-115

[83] Asaduzzaman M, Roy C K, Schneider K A. VisCad: Flexible code clone analysis support for NiCad//Proceedings of the 5th International Workshop on Software Clones. Hawaii, USA, 2011: 77-78

[84] Adar E, Kim M. SoftGUESS: Visualization and exploration of code clones in context//Proceedings of the 29th International Conference on Software Engineering. Minneapolis, USA, 2007: 762-766

[85] Forbes C, Keivanloo I, Rilling J. Doppel-Code: A clone visualization tool for prioritizing global and local clone impacts//Proceedings of the 36th Annual Computer Software and Applications Conference. Izmir, Turkey, 2012: 366-367

[86] Jiang Z M, Hassan A E. A framework for studying clones in large software systems//Proceedings of the 7th IEEE International Working Conference on Source Code Analysis and Manipulation. Paris, France, 2007: 203-212

[87] Jiang Z M, Hassan A E, Holt R C. Visualizing clone cohesion and coupling//Proceedings of the 13th Asia-Pacific Software Engineering Conference. Kanpur, India, 2006: 467-476

[88] Zhang Y, Basit H A, Jarzabek S, et al. Query-based filtering and graphical view generation for clone analysis//Proceedings of the International Conference on Software Maintenance. Beijing, China, 2008: 376-385

[89] Lin Y, Xing Z, Xue Y, et al. Detecting differences across multiple instances of code clones//Proceedings of the 36th International Conference on Software Engineering. Hyderabad, India, 2014: 164-174

[90] Mende T, Koschke R, Beckwermert F. An evaluation of code similarity identification for the grow-and-prune model. Journal of Software Maintenance and Evolution: Research and Practice, 2009, 21(2): 143-169

[91] Schulze S, Kuhlemann M, Rosenmüller M. Towards a refactoring guideline using code clone classification//Proceedings of the 2nd Workshop on Refactoring Tools. Nashville, USA, 2008: 1-4

[92] Choi E, Yoshida N, Ishio T, et al. Extracting code clones for refactoring using combinations of clone metrics//Proceedings of the 5th International Workshop on Software Clones. Nashville, USA, 2011: 7-13

[93] Mandal M, Roy C K, Schneider K. Automatic ranking of clones for refactoring through mining association rules//Proceedings of the Software Evolution Week—IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering. Antwerp, Belgium, 2014: 114-123

[94] Agrawal R, Imieliński T, Swami A. Mining association rules between sets of items in large databases. ACM SIGMOD Record, 1993, 22(2): 207-216

[95] Lee S, Bae G, Chae H S, et al. Automated scheduling for clone-based refactoring using a competent GA. Software: Practice and Experience, 2011, 41(5): 521-550

[96] Zibran M F, Roy C K. A constraint programming approach to conflict-aware optimal scheduling of prioritized code clone refactoring//Proceedings of the 11th IEEE International Working Conference on Source Code Analysis and Manipulation. Williamsburg, USA, 2011: 105-114

[97] Venkatasubramanyam R D, Gupta S, Singh H K. Prioritizing code clone detection results for clone management//Proceedings of the 7th International Workshop on Software Clones. San Francisco, USA, 2013: 30-36

[98] Liu H, Ma Z, Shao W, et al. Schedule of bad smell detection and resolution: A new way to save effort. IEEE Transactions on Software Engineering, 2012, 38(1): 220-235

[99] Barbosa F S, Aguiar A. Removing code duplication with roles//Proceedings of the 12th International Conference on Intelligent Software Methodologies, Tools and Techniques. Budapest, Hungary, 2013: 37-42

[100] Göde N. Clone removal: Fact or fiction?//Proceedings of the 4th International Workshop on Software Clones. Cape Town, South Africa, 2010: 33-40

[101] Zibran M F, Saha R K, Roy C K, et al. Evaluating the conventional wisdom in clone removal: A genealogy-based empirical study//Proceedings of the 28th Annual ACM Symposium on Applied Computing. Coimbra, Portugal, 2013: 1123-1130

[102] Choi E, Yoshida N, Inoue K. An investigation into the characteristics of merged code clones during software evolution. IEICE Transactions on Information and Systems, 2014, 97(5): 1244-1253

[103] Tairas R, Gray J. Sub-clones: Considering the part rather than the whole//Proceedings of the 8th International Conference on Software Engineering Research and Practice. Las Vegas, USA, 2010: 284-290

[104] Ali A F M, Sulaiman S. A systematic literature review of code clone prevention approaches. International Journal of Software Engineering and Technology, 2014, 1(1): 1-6

[105] Ali A F M, Sulaiman S. Enhancing generic pipeline model in preventing code clone during software development//Proceedings of the Software Engineering Postgraduates Workshop. Melaka, Malaysia, 2013: 56-61

[106] Wang X, Dang Y, Zhang L, et al. Predicting consistency-maintenance requirement of code clones at copy-and-paste time. IEEE Transactions on Software Engineering, 2014, 40(8): 773-794

[107] Venkatasubramanyam R D, Singh H K, Ravikanth K. A method for proactive moderation of code clones in IDEs//Proceedings of the 6th International Workshop on Software Clones. Zurich, Switzerland, 2012: 62-66

[108] Krutz D E, Le W. A code clone oracle//Proceedings of the 11th Working Conference on Mining Software Repositories. Hyderabad, India, 2014: 388-391

[109] Ishihara T, Hotta K, Higo Y, et al. Reusing reused code//Proceedings of the 20th Working Conference on Reverse Engineering. Koblenz, Germany, 2013: 457-461

[110] Ohta T, Murakami H, Igaki H, et al. Source code reuse evaluation by using real/potential copy and paste//Proceedings of the 9th International Workshop on Software Clones. Montreal, Canada, 2015: 33-39

[111] Yang J, Hotta K, Higo Y, et al. Classification model for code clones based on machine learning. Empirical Software Engineering, 2015, 20(4): 1095-1125

[112] Ohtani A, Higo Y, Ishihara T, et al. On the level of code suggestion for reuse//Proceedings of the 9th International Workshop on Software Clones. Montreal, Canada, 2015: 26-32

[113] Kintab G A, Roy C, McCalla G. Recommending software experts using code similarity and social heuristics//Proceedings of the 24th Annual International Conference on Computer Science and Software Engineering. Markham, Canada, 2014: 4-18

《计算机学报》

[114] Ishihara T, Hotta K, Higo Y, et al. Inter-project functional clone detection toward building libraries—An empirical study on 13,000 projects//Proceedings of the 19th Working Conference on Reverse Engineering. Kingston, Canada, 2012: 387-391

[115] Cheng X, Jiang L, Zhong H, et al. On the feasibility of detecting cross-platform code clones via identifier similarity //Proceedings of the 5th International Workshop on Software Mining. Singapore, 2016: 39-42

[116] Zhang H, Shi B, Zhang L. Automatic checking of license compliance//Proceedings of the International Conference on Software Maintenance. Timioara, Romania, 2010: 1-3

[117] Tairas R, Gray J. An information retrieval process to aid in the analysis of code clones. Empirical Software Engineering, 2009, 14(1): 33-56

[118] Koschke R. Frontiers of software clone management// Proceedings of the Frontiers of Software Maintenance. Beijing, China, 2008: 119-128

[119] De Wit M, Zaidman A, Van Deursen A. Managing code clones using dynamic change tracking and resolution//Proceedings of the International Conference on Software Maintenance. Edmonton, Canada, 2009: 169-178

[120] Hou D, Jablonski P, Jacob F. CnP: Towards an environment for the proactive management of copy-and-paste programming //Proceedings of the 17th International Conference on Program Comprehension. Vancouver, Canada, 2009: 238-242

[121] Weckerle V. CPC: An Eclipse Framework for Automated Clone Life Cycle Tracking and Update Anomaly Detection [M. S. dissertation]. Free University of Berlin, Berlin, Germany, 2008

[122] Jablonski P, Hou D. CReN: A tool for tracking copy-and-paste code clones and renaming identifiers consistently in the IDE//Proceedings of the OOPSLA Workshop on Eclipse Technology eXchange. Montreal, Canada, 2007: 16-20

[123] Jacob F, Hou D, Jablonski P. Actively comparing clones inside the code editor//Proceedings of the 4th International Workshop on Software Clones. Cape Town, South Africa, 2010: 9-16

[124] Yamanaka Y, Choi E, Yoshida N, et al. Applying clone change notification system into an industrial development process//Proceedings of the 21st International Conference on Program Comprehension. San Francisco, USA, 2013: 199-206

[125] Cheng X, Zhong H, Chen Y, et al. Rule-directed code clone synchronization//Proceedings of the 24th IEEE International Conference on Program Comprehension. Austin, USA, 2016: 1-10

[126] Zhang F, Khoo S C, Su X. Predicting consistent clone change//Proceedings of the 27th International Symposium on Software Reliability Engineering. Ottawa, Canada, 2016: 353-364

[127] Mondal M, Roy C K, Schneider K A. Prediction and ranking of co-change candidates for clones//Proceedings of the 11th Working Conference on Mining Software Repositories. Hyderabad, India, 2014: 32-41

[128] Murakami H, Hotta K, Higo Y, et al. Predicting next changes at the fine-grained level//Proceedings of the 21st Asia-Pacific Software Engineering Conference. Jeju, South Korea, 2014: 119-126

[129] Zhang G, Peng X, Xing Z, et al. Towards contextual and on-demand code clone management by continuous monitoring //Proceedings of the 28th International Conference on Automated Software Engineering. Silicon Valley, USA, 2013: 497-507

[130] Nguyen H A, Nguyen T, Pham N H, et al. Clone management for evolving software. IEEE Transactions on Software Engineering, 2012, 38(5): 1008-1026

[131] Tairas R, Gray J. Increasing clone maintenance support by unifying clone detection and refactoring activities. Information and Software Technology, 2012, 54(12): 1297-1307

[132] Juergens E, Deissenboeck F, Hummel B. Clone detective—A workbench for clone detection research//Proceedings of the 31st International Conference on Software Engineering. Vancouver, Canada, 2009: 603-606

[133] Kawaguchi S, Yamashina T, Uwano H, et al. SHINOBI: A tool for automatic code clone detection in the IDE// Proceedings of the 16th Working Conference on Reverse Engineering. Lille, France, 2009: 313-314

[134] Zhang F, Zheng Q, Zou Y, et al. Cross-project defect prediction using a connectivity-based unsupervised classifier// Proceedings of the 38th International Conference on Software Engineering. Austin, USA, 2016: 309-320

**SU Xiao-Hong**, born in 1966, Ph.D., professor. Her research interests include program analysis and its application, software fault localization, clone detection and analysis, software defect prediction.

**ZHANG Fan-Long**, born in 1987, Ph. D. candidate. His research interests include software engineering, program analysis, code clone analysis and maintenance, mining software repositories, and etc.

《计算机学报》

**Background**

In recent years, clone research community has increasing turning their attention to clone management. In general, clone management can be divided into three parts: clone detection, clone analysis and clone maintenance. Functionally, clone detection discovers clones in software; clone analysis assists developers in understanding clones, and clone maintenance aims to solve issues arisen from the presence of clones, as well as to make full use of their presence.

The most popular activity on clones is clone detection. In past decades, researchers propose several methods to detect clones, and rolled out a number of effective clone detection tools, such as NiCad, CCFinder, etc. According to the technology deployed, these clone detection methods can be classified into different bases: Text, Token, AST, PDG and Metrics. There are also lots of work on clone tools evaluation and clone survey. These works give us good directions on the clone research, and a good comprehensive set hundreds of articles.

There are lots of researchers keeping their eyes on clone analysis, such as clone classification, clone evolution, and clone evaluation, clone visualization. As clone analysis improves the understanding of clone development and clone impact on software, it has been an important research activity since a decade ago. Kim proposed clone genealogy to describe clone evaluation, and also used clone patterns to help us understand clones. The researchers have a passion on clone changes, which leads the research of clone evaluation. The clone harmfulness is still argument in clone community.

Unfortunately, clone analysis alone does not significantly mitigate software maintenance cost. Therefore, clone management issues become increasingly popular in this research community. The most popular clone management activity is clone refactoring. Researchers propose several methods to identify refactoring opportunities and to refactor clones. At the same time, several clone management tools have been built. Jysc, a novel clone management tool, provides two main functions. CloneTracker, an eclipse plug-in, provides support for tracking code clones in evolving software. CeDAR also an eclipse plug-in, can forward clone detection results to the refactoring engine in eclipse.

This research is partly supported by "the 13th Five-Year" National Science and Technology Major Project of China (Grant No. 2017YFC0702204), the National Natural Science Foundation of China under Grant Nos. 61173021 and 61672191. The group has been working for several years on software bugs detecting, software faults understanding and locating, clone codes and their related defects detection, software refactoring, software testing, and so on. A number of research papers have been published in respectable international journals and conferences, such as *Journal of Systems and Software*, ACM SIGSOFT Symposium on the Foundation of Software, etc.

《计算机学报》