软件克隆检测技术研究*

梁正平[1,2]，程一群[1]，谭佳加[1]，马骁驰[1]
(1. 深圳大学 计算机与软件学院，广东 深圳 518060；2. 武汉大学 软件工程国家重点实验室，武汉 430072)

摘　要：软件克隆检测在软件维护、软件结构优化等方面具有重要价值和意义。综述了软件克隆的定义与分类，对软件克隆的检测过程进行了划分和讨论，介绍了软件克隆检测领域最为活跃的代码克隆检测技术和模型克隆检测技术。最后对软件克隆检测的研究现状和急需解决的问题进行了分析，展望了该领域未来的研究方向。

关键词：软件克隆；克隆检测；代码克隆；模型克隆
中图分类号：TP311.5　　**文献标志码**：A　　**文章编号**：1001-3695(2012)05-1623-05
doi:10.3969/j.issn.1001-3695.2012.05.005

Research on software clone detection

LIANG Zheng-ping[1,2], CHENG Yi-qun[1], TAN Jia-jia[1], MA Xiao-chi[1]
(1. *College of Computer Science & Software Engineering, Shenzhen University, Shenzhen Guangdong* 518060, *China*; 2. *State Key Laboratory of Software Engineering, Wuhan University, Wuhan* 430072, *China*)

Abstract: Software clone detection is a very important technology for software maintenance, software structure optimization, etc. The paper summarized the definition of software clone and its categories, divided and discussed the detection process for software clone. After that, it introduced two kinds of most actively research domain of software clone detection, that was code clone detection and model detection. Finally, the paper reviewed the state-of-art of current software clone detection research and identified some open research issues and pointed out possible future research directions in software clone detection area.
Key words: software clone; clone detection; code clone; model clone

0　引言

软件分析是对软件进行人工或者自动分析，以验证、确认或发现软件性质(或者规约、约束)的过程或活动[1]。克隆分析是一项重要的软件分析技术。狭义的软件克隆是指软件代码中相同或相似的代码片段[2]；广义的软件克隆还包括软件需求规约、软件模型等软件制品中存在的相同或相似部分[3,4]。

软件开发实践中，由于大量“复制—粘贴—修改”操作的存在，以及受开发人员固有思维模式和开发能力的影响等，软件克隆现象普遍存在[5]。以代码克隆为例，据相关文献统计，大型软件系统中约 20% ~50% 的代码为克隆代码[6]。软件克隆对于软件的可维护性具有十分明显的负面效应[7]，主要表现如下：a)软件克隆增加了软件制品的长度，造成软件维护工作量和费用的增加，同时软件运行时要求更多的资源；b)若对部分克隆实例进行了修改，而其他对应的克隆实例未进行修改，容易造成软件行为出现不一致，导致各种软件错误的产生。软件克隆对于软件的结构和抽象层次等同样具有明显的负面影响，不利于软件结构的优化和抽象层次的提升[7]。

不论软件克隆的产生原因和负面影响如何，软件克隆存在的根源在于软件的不同部分之间存在内在的语义联系[8]。因此，对软件克隆进行检测和标志，发掘各克隆实例之间的语义关联，特别是发掘高层的结构克隆，有利于优化软件的结构和抽象层次，提高软件质量，同时降低软件维护的难度和成本。软件克隆检测可广泛应用于程序理解、软件重构、抄袭检测、版权保护、方面挖掘、产品线开发、恶意代码分析及错误预防等领域[2,5]。

本文通过对软件克隆的定义与分类、软件克隆的检测过程、现有的代码克隆检测技术和模型克隆检测技术等较全面的介绍，系统地分析和梳理软件克隆检测技术的研究现状，并对未来的发展方向进行了讨论。

1　软件克隆的定义与分类

近年来，软件克隆的研究十分活跃，是软件工程领域最热门的研究方向之一，ICSE[3]、ICSM[9]、IEEE Trans on SE[6]、ACM Trans on SE&M[10]等顶级学术会议和期刊都有大量相关研究进展和成果的报告。软件克隆研究包括克隆定义与分类、克隆检测、克隆分析、克隆管理以及相关的实例研究与工具开发等[5]。其中，对软件克隆定义与分类的研究是对其他软件克隆问题进行研究的出发点。

对于什么是软件克隆，至今没有一个严格的形式定义[2]。较普遍接受的是早期 Baxter 等人[11]从相似性角度提出的代码

收稿日期：2011-10-24；**修回日期**：2011-12-12　　**基金项目**：国家自然科学基金资助项目(60903114)；软件工程国家重点实验室开放基金资助项目(SKLSE20080702)；深圳市科技研发资金基础研究计划资助项目(JC201005280432A)

作者简介：梁正平(1979-)，男，湖南涟源人，副教授，博士，主要研究方向为模型克隆检测、需求建模与分析等(liangzp@szu.edu.cn)；程一群(1985-)，男，硕士研究生，主要研究方向为模型克隆检测.

克隆定义:符合某一相似性定义的相同或相似代码片段。该定义强调了代码克隆中存在的相似性,但对于什么是相似并没有给出明确说明。这也是目前软件克隆领域,特别是其中的代码克隆领域研究方法和途径众多,研究成果百花齐放的根源。模型克隆的定义与代码克隆类似,通常模糊地定义为相同或相似的模型片段[4]。

克隆的分类与克隆的定义密切相关。对于代码克隆,分类方式众多,较普遍认可的是 Roy 等人[2]分为四类的提议:第一类指除空白和注释外完全相同的代码片段,也称为精确克隆(exact clone);第二类指除空白、注释以及变量名及常量值的变化外,语法结构完全相同的代码片段;第三类指除包含第二类克隆所涉及差异外,还存在少量语句增加、删除、修改的代码片段,也称为近似克隆(near-miss clone);第四类指具有相同的语义,但采用不同语法形式的代码片段。前三类克隆从文本相似的角度进行划分,第四类则从功能相似的角度划分。此外,Basit 等人[6]为发掘代码中隐藏的设计模式,以便更利于软件维护、程序理解等任务,将上述四类克隆统称为简单克隆(simple clone),而将由简单克隆组合而成的高层粗粒度克隆称为结构克隆(structural clone)。

模型克隆研究由于刚刚兴起,除 Pham 等人[4]将其粗略地分为精确克隆和相似克隆(similar clone)两类外,明确讨论其分类体系的仅有文献[12]。该文仿照上述代码克隆的分类,同时结合模型克隆的特点将其分为四类。其中,第一类指结构、内容、布局等完全相同的模型片段;第二类指除布局、注释存在与语义无关的差异外,其他完全相同的模型片段;第二类在第一类的基础上还存在常量值的替换;第三类指除第二类中存在的差异外,进一步存在少量边和/或节点的增加、删除、修改的模型片段。上述第一至三类模型克隆不存在结构的差异,模仿代码克隆的命名方式,可统称为精确克隆,第四类模型克隆则存在结构的差异,可称为近似克隆。

此外,Domann 等人[13]对需求规格说明书中存在的克隆进行了研究。他们把需求规格说明书看成是一个由单词构成的序列,其克隆的定义如下:需求规格说明书中至少重复出现两次,且长度不小于某一给定值的连续字符串。同时,将克隆组分为相关克隆组(relevant clone group)和假阳性克隆组(false positive group)两类。其中,前者中的元素既满足克隆的定义,又在语义上相似,且属需求规格说明书中与某一系统描述对应的字符序列;后者中的元素则仅满足克隆的定义,但语义不相似或不具有某一较独立的语义功能。

2 软件克隆的检测过程

不同类型软件制品的形态不同,所采用的克隆检测技术亦具有较大的差异,但它们的整体检测过程基本相同,主要包括预处理与规范化、匹配检测、格式化、后处理、结果展示与报告五个阶段,如图 1 所示。

a)预处理与规范化是软件克隆检测的初始阶段,其处理对象为原始形态的待检测软件制品。预处理对待检测软件制品进行合理的简化、组合或分解,包括删除待检测软件制品中与克隆检测无关的内容,如各种注释信息,将待检测软件制品的结构进行分解等。规范化从格式、表示形式等方面对待检测软件制品进行处理,包括按某种特定的风格对待检测软件制品的内容进行编排,将待检测软件制品转换为某种较单一、标准的中间形式,如将源代码转换为归一化的形符(token)序列[14],将模型转换为有向标记图(labeled directed graph)[15]等,预处理和规范化的目的是减少各种外围噪声对克隆检测的干扰,从而减少后阶段克隆匹配检测所需的比对次数,并降低匹配检测的难度和复杂度。此阶段的核心问题是规范化的方式和程度。规范化方式不同,后阶段所需采取匹配流程和技术不同,所能检测的克隆类型也不同。对于规范化程度,若过于规范化,容易导致检测结果中出现大量的假阳性,降低克隆检测的精确率;若规范化程度不够,则可能导致很多克隆类,特别是各种相似克隆类无法检测到,降低克隆检测的召回率。

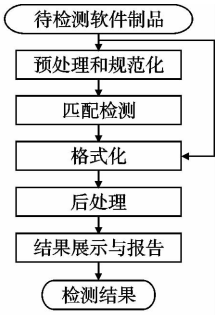

图1 软件克隆检测过程

b)匹配检测是软件克隆检测的核心阶段,包括三个方面的主要内容,即候选克隆片段的选取、候选克隆片段的两两比对、克隆类的构建。候选克隆片段的选取是决定整体克隆检测速度的关键因素之一,与待检测软件制品的形态及其规范化表示形式密切相关。对于代码克隆的检测,可基于编程语言的语法结构选取克隆片段,相对比较容易。而对于模型克隆的检测,由于模型内部的结构特征通常并不明显,如何选取克隆片段是一个需要细致研究的问题,通常引入启发式策略以规避穷举方式所带来的复杂时空开销。候选克隆片段的比对方式也是决定整体克隆检测速度的关键因素,同样与待检测软件制品的形态以及其规范化表示形式密切相关。对于形符形态的源代码规范化表示,可使用后缀树(suffix tree)[14]和频繁子序列挖掘(frequent subsequent mining)[16]算法等进行匹配检测,对于有向标记图形态的模型规范化表示,可使用子图同构(subgraph isomorphism)[15]、特征向量(characteristic vector)[4]等技术进行检测。克隆类的构建相对较为容易处理,极大团覆盖算法[17]是较常采用的克隆聚类算法,此处稍微麻烦的是不同克隆类之间存在重叠情形的处理,特别是元素粒度不同的重叠克隆类还存在元素间包含关系的情形。

c)格式化阶段的输入包括两类:原始的待检测软件制品以及在待检测软件制品规范化表示形态基础上检测出的克隆对和克隆类。本阶段首先构建待检测软件制品规范化表示到其原始形态的映射,再基于上一阶段的检测结果在原始形态的待检测软件制品中对相应的克隆对和克隆类进行定位。本阶段与软件的规范化表示形成互补。

d)后处理阶段对前面阶段的检测结果进行过滤和分类。由于预处理和规范化表示阶段对待检测软件制品进行了抽象,在此基础上获得的检测结果中可能存在某些假阳性和不相关性,需基于克隆检测的动机与目的、所检测出克隆片段的语法

结构与语义自含性等,设计有效的克隆对和克隆类过滤与分类规则,对克隆检测结果进行清洗,以提高检测结果的精确率和可用性。由于软件制品的内部结构和语义逻辑复杂,本阶段的工作通常难以依赖计算机全部自动完成。现有的很多软件克隆检测技术在本阶段采用半自动方式,即人工辅助方式对克隆检测结果进行过滤。

e)结果展示与报告是软件克隆检测的最后阶段。本阶段主要关注如何以直观、易于接受与理解的形式对检测结果进行格式化展现和报告。较常用的技术包括采用 HTML 构建克隆片段间的链接关系、采用 XML 形式对存在克隆关系的软件片段进行描述、对克隆片段进行着色处理、以饼图/柱状图/散点图等形式报告各类克隆片段的分布和数量等。

上述五个阶段是对软件克隆检测过程的宏观划分。在实际的软件克隆检测过程中,受待检测软件制品类型以及各类检测技术侧重点不同的影响,不同检测技术对检测步骤和阶段的划分可能存在差异,所采用的技术细节也可能相差很大。

3　代码克隆检测

代码克隆检测是软件克隆研究中成果最为丰富的研究课题,相关的研究工作从 1990 年代末至今持续保持活跃的状态。不同的代码克隆检测技术可从检测速度、可移植性、可检测的克隆类型、检测结果的精确率、召回率(recall)等方面进行分析。根据预处理过程中代码所变换中间表示形式的不同,代码克隆检测可分为如下五大类型:

a)基于文本(text)的检测是最早提出的克隆检测技术,该类检测除对文本布局、注释等进行规范化处理外,一般不对源代码作其他形式的变换。该类检测技术具有可移植性强、精确率高的优点,但其召回率较低,一般只能检测第一类克隆,其检测速度则取决于所采用的比较算法。为改进此类检测技术,2005 年,Marcus 等人[18] 应用信息检索领域的潜在语义索引(latent semantic indexing)算法挖掘源代码中隐含的高层概念,可检测出部分第三类克隆。此后,Roy 等人[19] 开发的 NICAD 应用灵活的过滤和规范化机制对文本检测进行改进,可有效检测全部第三类克隆。2010 年,Hummel 等人[9] 针对源代码行基于索引(index)方式提出了一种递增、分布式的快速检测第一、二类克隆的技术。

b)基于形符的检测采用词法分析方式将源代码变换为归一化的形符序列。该类技术具有召回率高、检测速度快的优点,但精确率较低,其可移植性则由源代码的词法分析器决定。2002 年,Kamiya 等人[14] 开发的 CCFinder 使用后缀树匹配技术对形符序列进行克隆检测,是此类检测方式中最为经典的技术,CCFinder 可检测第三类克隆。基于形符方式的另一经典技术是 2006 年 Li 等人[16] 提出的 CP-Miner,该技术使用频繁子序列挖掘算法对形符序列进行检测,同样具有较好的检测效果。2007 年,Livieri 等人[20] 将 CCFinder 扩展为用分布方式实现的 D-CCFinder,以满足超大型系统检测的需要。2009 年 Juergens 等人[21] 开发了一个基于形符的克隆检测工具 CloneDetective,该工具采用模块形式,可灵活支持克隆检测各阶段不同技术的组合定制,为软件克隆检测的研究提供了一个友好的开放式平台。

c)基于抽象语法树(abstract syntax tree,AST)的检测采用语法分析方式将源代码变换为抽象语法树。该类技术具有精确度高的优点,但召回率和可移植性均较差,其检测速度同样依赖于具体所采用的比较算法。早期经典的方法是 1998 年 Baxter 等人[11] 基于子树匹配(subtree matching)算法提出的 ClonerDr。2007 年,Jiang 等人[22] 开发的 DECKARD 为每棵子树赋予一个数字向量,再使用位置敏感哈希(locality sensitive hashing)函数通过聚类方式检测相似的子树,可提高克隆检测的召回率。2008 年,Falke 等人[23] 将 AST 子树线性化为节点序列,再使用后缀树算法进行检测,提高了检测的速度。此后,2009 年,Evans 等人[24] 采用结构抽象方式改进传统的 AST 子树匹配机制,能有效支持第三类克隆的检测。Chilowicz 等人[25] 利用哈希方式对 AST 子树构建索引,然后借助数据库技术进行快速匹配,该方法检测速度快、简单明了。

d)基于程序依赖图(program dependency graph,PDG)的检测采用静态程序分析方法将源代码变换为图形化的 PDG,再基于子图同构的概念对代码克隆进行检测。由于 PDG 保存了程序的语义信息,该类技术具有非常好的精确度,且能有效地检测第四类克隆,但其召回率一般,且可移植性差。同时因子图同构检测属 NP 难题,该类检测技术速度非常慢、可扩展性低,一直难以进行实际应用。为降低检测难度,2001 年,Komondoor 等人[26] 在 PDG 的子图同构检测过程中引入了程序切片(program slicing)技术。2008 年,Gabel 等人[27] 将 PDG 中的子图同构问题规约为子树同构问题,再利用 DECKARD 进行检测,以提高检测的速度和可扩展性。最近,Higo 等人[28] 报告了一种基于启发式策略的 PDG 检测技术,较好地推进了该类技术的实用性。

e)基于度量(metrics)的检测通过提取代码的某些度量值对代码克隆进行间接检测。该类技术支持第三类克隆的检测,具有检测速度快、可扩展性好、精确率和召回率适中的优点,但其与源代码的预处理方式密切相关,可移植性较差。早期 Davey 等人[29] 利用神经网络(neural network)技术探讨了基于度量方式对代码段进行克隆检测的可行性,Mayrand 等人[30] 则专门研究了基于度量方式的函数克隆检测问题。2009 年,Grant 等人[31] 将数字信号处理领域的独立分量分析(independent component analysis)技术用于代码克隆的检测,初步结果显示该方法具有较好的效果。2010 年,Shawky 等人[32] 对克隆检测过程中各度量值的有效性进行了分析。

不同类型代码克隆检测技术各具优缺点,为扬长避短,近年来,学术界掀起了组合上述各类检测技术的研究热潮。上文提到的 DECKARD 即是一种组合了抽象语法树和度量技术的检测工具,NICAD 则是一种组合了文本和抽象语法树的检测技术。2010 年,Funaro 等人[33] 探讨了组合词法分析和文本分析的克隆检测方法,Selim 等人[34] 提出了使用中间表示形式增强代码克隆检测能力的技术。与此同时,一些新的克隆检测思路和方法也不断涌现,如 Davis 等人[35] 提出的基于对应的汇编语言对代码克隆进行检测等。

4　模型克隆检测

随着模型驱动开发技术的成熟和广泛应用[36],模型克隆

检测逐渐成为软件克隆检测研究的前沿[37]。相比代码克隆检测已取得的丰硕成果,模型克隆检测方兴未艾,已有的主要成果如下:

2006 年,首次专门研究模型克隆检测的是北京大学 Liu 等人[38],他们以 UML 顺序图模型为检测对象。其核心思想是利用顺序图自身的特点,将二维的顺序图序列化,再基于后缀树算法对该一维序列进行检测,可有效检测出顺序图模型中存在的克隆。

2008 年,Deissenboeck 等人[15]对一个超过 2 万个节点的大型 MATLAB/Simulink 模型进行了克隆检测。他们采用深度优先的方式对预处理后得到的有向标记图进行搜索,为提高检测速度,对具有多个分支的节点,基于启发式剪枝技术,仅搜索其中最可能存在克隆的一个分支。该工作能检测模型中结构同构的精确克隆,并首次验证了对大型模型进行克隆检测的可行性。

2009 年,Pham 等人[4]以四个开源 MATLAB/Simulink 模型为例,基于频繁子图挖掘算法和 Exas 特征向量提出了两种分别针对精确克隆和相似克隆的检测技术,并开发了相应的检测工具 ModelCD。ModelCD 是首个能检测模型近似克隆的工具,在精确克隆检测方面比 Deissenboeck 的工作具有更高的召回率。

2010 年,Deissenboeck 吸取 ModelCD 的优点,对前期的精确克隆检测工作进行了改进,并结合四个不同规模的实例验证并分析了改进后的克隆检测效果[37]。此外,Störrle 在 2010 年报告了一个基于域模型对各类 UML 模型统一进行克隆检测的初步框架[39]。

上述面向大型模型的克隆检测技术中,Liu 的方法通过对图进行一维序列化处理,可充分借用代码克隆检测领域中已有的成熟技术,但该方法与待检测模型的特性密切相关,通用性较差。Deissenboeck 的方法基于子图同构思路仅支持精确克隆的检测。Pham 的方法虽通过引入特征向量思想支持近似克隆的检测,但一方面该方法仅迭代检测由递增方式直接生成的候选克隆,检测结果的召回率存在较大的改进空间,另一方面对检测结果的精确率未作具体研究,在检测速度、实例分析等方面也只报告了初步成果,有待进一步提高。此外,现有工作中尚没有任何模型结构克隆检测方面的研究成果。

5 软件克隆检测技术发展动态

综观软件克隆检测研究的现状和发展动态不难发现:

a)软件克隆检测技术已获得学术界的广泛关注和重视,并已取得较丰硕的研究成果,特别是有关代码克隆检测技术的研究,各具特色、形态各异的检测方法和检测工具众多。但软件克隆检测技术总体上还未进行大规模实用,因此急需将已取得的研究成果与现有各类软件开发与维护技术、工具等有机融合,提升软件克隆检测技术在实际软件开发和维护过程中的价值和作用。

b)现有的软件克隆检测技术虽然在代码克隆检测方面取得了显著的成果,但对于其他形态的软件制品,如模型、需求规格说明书等,相关的克隆检测技术尚处于萌芽状态,还存在许多急需解决和改进的地方,如模型克隆检测速度的提高、模型近似克隆检测技术的改进、模型结构克隆的检测方法、大型模型克隆检测的经验研究和实例分析、大型需求规格说明书克隆检测的经验研究和实例分析、克隆检测结果精确率和召回率的提高等。与此同时,代码克隆检测技术的研究亦并非尽善尽美,如基于语义的代码克隆检测问题、大型代码库克隆检测的速度问题等。

c)软件克隆的研究除包括克隆检测外,还包括克隆分析、克隆管理、克隆检测技术在其他领域的应用等多个方面。现有的成果主要以各类形式的克隆检测技术为中心,缺乏对其他相关问题的分析和讨论,软件克隆的整体研究广度有待扩宽。

6 结束语

软件克隆检测是一种重要的软件分析技术,对于软件维护、软件结构优化等具有重要的价值和意义,有利于提高软件产品的质量,降低软件开发的成本。作为当前软件工程领域的重要研究课题,软件克隆检测的研究已获得国际学术界和产业界众多研究人员和机构的极大关注,并有望在未来的软件开发和维护中发挥重要的实际作用。

本文从软件克隆的定义与分类、软件克隆的检测过程出发,讨论了软件克隆检测所需解决的主要问题。通过对现有代码克隆检测技术和模型克隆检测技术的介绍与分析,对软件克隆检测的研究现状进行了综述,在此基础上对软件克隆检测技术的发展趋势进行了分析和讨论。

参考文献:

[1] 梅宏, 王千祥, 张路,等. 软件分析技术进展[J]. 计算机学报, 2009, 32(9):1697-1710.

[2] ROY C K, CORDY J R, KOSCHKE R. Comparison and evaluation of clone detection techniques and tools: a qualitative approach[J]. Science of Computer Programming, 2009,74(7):470-495.

[3] JUERGENS E, DEISSENBOECK F, FEILKAS M, *et al*. Can clone detection support quality assessments of requirements specifications [C]//Proc of the 32nd ACM/IEEE International Conference on Software Engineering. New York: ACM, 2010:79-88.

[4] PHAM N H, NGUYEN H A, NGUYEN T T, *et al*. Complete and accurate clone detection in graph-based models[C]//Proc of the 31st International Conference on Software Enginee-ring. Washington DC: IEEE Computer Society, 2009: 276-286.

[5] ROY C K, CORDY J R. A survey on software clone detection research,2007-541[R]. Kingston, Ontario: Queen's University,2007.

[6] BASIT H, JARZABEK S. A data mining approach for detecting higher-level clones in software[J]. IEEE Trans on Software Engineering, 2009,35(4):497-513.

[7] JUERGENS E, DEISSENBOECK F, HUMMEL B, *et al*. Do code clones matter? [C]// Proc of the 31st International Conference on Software Engineering. Washington DC: IEEE Computer Society, 2009:485-495.

[8] JARZABEK S, XUE Yin-xing. Are clones. harmful for maintenance [C]// Proc of the 4th International Workshop on Software Clones New York: ACM, 2010: 73-74.

[9] HUMMEL B, JUERGENS E, HEINEMANN L, *et al*. Index-based code clone detection: incremental, distributed, scalable[C]// Proc of

the 26th IEEE International Conference on Software Maintenance. Washington DC:IEEE Computer Society, 2010:1-9.

[10] DUALA-EKOKO E, ROBILLARD M P. Clone region descriptors:representing and tracking duplication in source code[J]. ACM Trans on Software Engineering and Methodology,2010,20(1):3.1-3.31.

[11] BAXTER I, YAHIN A, MOURA L,*et al*. Clone detection using abstract syntax trees[C]// Proc of the 14th IEEE International Conference on Software Maintenance. Washington DC:IEEE Computer Society, 1998:368-377.

[12] GOLD N, KRINKE J, HARMAN M,*et al*. Issues in clone classification for dataflow languages[C]// Proc of the 4th International Workshop on Software Clones. New York:ACM, 2010:83-84.

[13] DOMANN C, JUERGENS E, STREIT J. The curse of copy&paste cloning in requirements specifications[C]//Proc of the 3rd International Symposium on Empirical Software Engineering and Measurement. Washington DC:IEEE Computer Society, 2009:443-446.

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

[15] DEISSENBOECK F, HUMMEL B, JUERGENS E,*et al*. Clone detection in automotive model-based development[C]//Proc of the 30th International Conference on Software Engineering. New York: ACM, 2008:603-612.

[16] LI Zen-min, LU Shan, MYAGMAR S,*et al*. CP-Miner:finding copy-paste and related bugs in large-scale software code[J]. IEEE Trans on Software Engineering,2006,32(3):176-192.

[17] BRON C, KERBOSCH J. Algorithm 457:finding all cliques of an undirected graph[J]. Communications of the ACM,1973,16(9):575-577.

[18] MARCUS A, RAJLICH V, BUCHTA J,*et al*. Static techniques for concept location in object-oriented code[C]//Proc of the 13th IEEE International Workshop on Program Comprehension. Washington DC: IEEE Computer Society, 2005:33-42.

[19] ROY C K, CORDY J R. Near-miss function clones in open source software:an empirical study[J]. Journal of Software Maintenance and Evolution: Research and Practice,2010,22(3):165-189.

[20] LIVIERI S, HIGO Y, MATSUSHITA M,*et al*. Very-large scale code clone analysis and visualization of open source programs using distributed CCFinder: D-CCFinder[C]// Proc of the 29th International Conference on Software Engineering. Washington DC:IEEE Computer Society, 2007:106-115.

[21] JUERGENS E, DEISSENBOECK F, HUMMEL B. CloneDetective:a workbench for clone detection research[C]//Proc of the 31st International Conference on Software Engineering. Washington DC: IEEE Computer Society, 2009:603-606.

[22] JIANG Ling-xiao, MISHERGHI G, SU Zhen-dong,*et al*. DECKARD: scalable and accurate tree-based detection of code clones[C]//Proc of the 29th International Conference on Software Engineering. Washington DC:IEEE Computer Society, 2007:96-105.

[23] FALKE R, FRENZEL P, KOSCHKE R. Empirical evaluation of clone detection using syntax suffix trees[J]. Empirical Software Engineering,2008,13(6): 601-643.

[24] EVANS W S, FRASER C W, MA Fai. Clone detection via structure abstraction[J]. Software Quality Journal,2009,17(4):309-330.

[25] CHILOWICZ M, DURIS E, ROUSSEL G. Syntax tree fingerprinting for source code similarity detection[C]//Proc of the 17th IEEE International Conference on Program Comprehension. Washington DC: IEEE Computer Society, 2009:243-247.

[26] KOMONDOOR R, HORWITZ S. Using slicing to identify duplication in source code[C]//Proc of the 8th International Symposium on Static Analysis. London:Springer, 2001:40-56.

[27] GABEL M, JIANG Ling-xiao, SU Zhen-dong. Scalable detection of semantic clones[C]//Proc of the 30th International Conference on Software Engineering. New York:ACM, 2008:321-330.

[28] HIGO Y, KUSUMOTO S. Code clone detection on specialized PDGs with heuristics[C]//Proc of the 15th European Conference on Software Maintenance and Reengineering. Washington DC:IEEE Computer Society, 2011:75-84.

[29] DAVEY N, BARSON P C, FIELD S D H,*et al*. The development of a software clone detector[J]. International Journal of Applied Software Technology,1995,1(3/4):219-236.

[30] MAYRAND J, LEBLANC C, MERLO E M. Experiment on the automatic detection of function clones in a software system using metrics[C]//Proc of the 12th International Conference on Software Maintenance. Washington DC:IEEE Computer Society, 1996:244-253.

[31] GRANT S, CORDY J R. Vector space analysis of software clones[C]//Proc of the 17th IEEE International Conference on Program Comprehension. Washington DC:IEEE Computer Society, 2009:233-237.

[32] SHAWKY D M, ALI A F. An approach for assessing similarity metrics used in metric-based clone detection techniques[C]//Proc of the 3rd International Conference on Computer Science and Information Technology. Washington DC:IEEE Computer Society, 2010:580-584.

[33] FUNARO M, BRAGA D, CAMPI A,*et al*. A hybrid approach (syntactic and textual) to clone detection[C]// Proc of the 4th International Workshop on Software Clones. New York:ACM, 2010:79-80.

[34] SELIM G, FOO K C, ZOU Ying. Enhancing source-based clone detection using intermediate representation[C]//Proc of the 17th Working Conference on Reverse Engineering. Washington DC:IEEE Computer Society, 2010:227-236.

[35] DAVIS I J, GODFREY M W. Clone detection by exploiting assembler[C]//Proc of the 4th International Workshop on Software Clones. New York:ACM, 2010:77-78.

[36] 蒋哲远, 蒋建国. 面向服务领域软件系统的模型驱动建模方法[J]. 计算机科学, 2008, 35(5):274-279.

[37] DEISSENBOECK F, HUMMEL B, JUERGENS E,*et al*. Model clone detection in practice[C]//Proc of the 4th International Workshop on Software Clones. New York:ACM, 2010:57-64.

[38] LIU Hui, MA Zhi-yi, ZHANG Lu,*et al*. detecting duplications in sequence diagrams based on suffix trees[C]//Proc of the 13th Asia Pacific Software Engineering Conference. Washington DC:IEEE Computer Society, 2006:269-276.

[39] STÖRRLE H. Towards clone detection in UML domain models[C]// Proc of the 4th European Conference on Software Architecture. New York:ACM,2010: 285-293.