**哈尔滨工业大学**

**博士学位论文中期检查报告**

**题 目：****基于演化的克隆代码分析与维护方法研究**

**院 (系) 计算机科学与技术**

**学 科 计算机应用技术**

**导 师 苏小红 教授**

**研 究 生 张凡龙**

**学 号 12B903017**

**开题报告日期 2016年9月27日**

**研究生院制**

**二〇一二年三月**

说 明

博士学位论文工作科研成果报告，报告形式可选择以下2种之一。

1 、提交1篇在本领域顶级期刊发表的能反映学位论文工作已取得的科研成果的论文。

2 、提交1篇反映学位论文工作已取得的科研成果的报告，报告内容按引言、主体和结论三部分撰写。

引言部分：描述研究背景和意义、研究基础、研究范围和目标、研究思路和总体方案等。

主体部分：逐一论述各项研究内容的研究方案、研究方法、研究过程、研究结果等信息，提供必要的图、表、实验及观察数据等信息。

结论部分：阐述主要研究发现，可包括研究成果的作用、影响、应用前景，和研究中的问题、经验和建议等。）

目 录

[1 课题来源及研究的目的和意义 1](#_Toc462653297)

[1.1 课题来源 1](#_Toc462653298)

[1.2 课题研究背景 1](#_Toc462653299)

[1.3 研究的目的及意义 1](#_Toc462653300)

[2 国内外在该方向的研究现状及分析 3](#_Toc462653301)

[2.1 克隆代码检测 3](#_Toc462653302)

[2.2 克隆代码分析 3](#_Toc462653303)

[2.2.1 克隆定义与表示 3](#_Toc462653304)

[2.2.2克隆演化 4](#_Toc462653305)

[2.2.3克隆评价 5](#_Toc462653306)

[2.2.4克隆可视化 5](#_Toc462653307)

[2.3 克隆维护 5](#_Toc462653308)

[2.3.1克隆重构 6](#_Toc462653309)

[2.3.2克隆维护管理方法 6](#_Toc462653310)

[3 解决的关键问题(研究基础和研究范围) 8](#_Toc462653311)

[4 课题研究内容、目标、思路及总体方案 10](#_Toc462653312)

[4.1 研究内容 10](#_Toc462653313)

[4.2 研究目标 10](#_Toc462653314)

[4.3 研究思路 11](#_Toc462653315)

[4.4 研究方案 12](#_Toc462653316)

[5 目前已完成的研究工作及结果 14](#_Toc462653317)

[5.1 基于聚类分析的克隆代码演化特征提取 14](#_Toc462653318)

[5.1.1 引言 14](#_Toc462653319)

[5.1.2 方法 14](#_Toc462653320)

[5.1.3 实验 15](#_Toc462653321)

[5.1.4 结论 18](#_Toc462653322)

[5.2 基于复制粘贴的克隆代码一致性维护需求预测 19](#_Toc462653323)

[5.2.1 引言 19](#_Toc462653324)

[5.2.2 方法 19](#_Toc462653325)

[5.2.3 实验设置 20](#_Toc462653326)

[5.2.4 贝叶斯方法评估（实验结果和分析1） 21](#_Toc462653327)

[5.2.5 多分类器实验（实验结果和分析2） 24](#_Toc462653328)

[5.2.6 结论 25](#_Toc462653329)

[5.3 基于贝叶斯网络的克隆代码变化预测 25](#_Toc462653330)

[5.3.1 引言 25](#_Toc462653331)

[5.3.1 方法 26](#_Toc462653332)

[5.3.3 实验设置 27](#_Toc462653333)

[5.3.4 实验结果与分析 27](#_Toc462653334)

[5.3.5 结论 29](#_Toc462653335)

[6 预期创新点 30](#_Toc462653336)

[7 后期拟完成的研究工作及进度安排标 31](#_Toc462653337)

[8 结论 32](#_Toc462653338)

[9 发表论文情况 33](#_Toc462653339)

1 课题来源及研究的目的和意义

1.1 课题来源

本课题来源于国家自然科学基金面上项目“无定型克隆代码的检测与重构方法”（批准号：61173021）。

1.2 课题研究背景

随着计算机软件广泛应用于经济、军事、商业等各个领域中，软件质量问题日益得到人们的广泛重视。保证软件高质量，并提高软件可维护性、可理解性已经成为系统开发和维护工作的一个不可或缺的重要方面。然而，随着需求和应用环境的不断变化，现实世界的软件系统也随着时间在不断演化，软件规模越来越大、逻辑越来越复杂，软件的质量、可维护性和可理解性会逐渐下降。

软件复用是软件开发人员常用的一种系统开发手段，软件复用提高了系统开发的效率，但同时也造成软件中大量的克隆代码。有研究表明，软件工程实践中可能产生多种类型的克隆代码，克隆代码在大型软件系统中约占代码总量的7-23%。随着时间的推移和系统功能的不断添加，软件代码规模越来越大，程序结构越来越复杂，程序中的克隆代码也会越来越多，在系统复用已有代码的同时难免会在系统中引入缺陷，这就增加了系统维护的成本和难度，降低了软件的可理解性。有研究表明，当前影响软件质量、可维护性和可理解性的一个主要因素是软件中存在的大量的克隆代码。因此，如何理解和分析系统中既有的克隆代码，并且有效的维护管理系统中的克隆代码是一个值得研究的问题。对克隆代码的分析和维护研究，可以有效降低克隆代码对软件质量的影响，已成为目前软件工程领域亟待解决的一个问题。

1.3 研究的目的及意义

克隆代码的存在会对软件系统造成不可避免的影响，针对大型软件系统对软件可靠性要求较高的实际应用背景和需求。研究人员开展了大量且较为深入的克隆代码的研究，已经取得了较多的成果。当前对克隆代码的研究可分为三个主要的研究活动，即克隆代码检测、克隆代码分析和克隆代码维护。在当前对克隆的研究活动中，克隆代码的检测是最主要的研究活动，也是软件开发和维护领域中的一项重要任务。到目前为止，已经研究和开发出许多克隆代码检测的方法和工具，可以一定程度上识别系统中存在的多种类型的克隆代码。但由于克隆代码大量存在并且情况复杂，克隆检测结果并不是完全令人满意的。当前方法对于近似克隆代码检测效果并不好，同时无法完整的检测系统四型克隆。更令人值得注意的是，克隆代码检测的研究无法消除克隆代码对软件的不利影响，仅可以帮助无法直接帮助提高软件的质量以及可维护性。因此，克隆分析和克隆维护的研究可以更好地弥补这一不足之处。

因此，本课题通过对克隆代码的分析和维护研究，帮助维护人员理解和维护软件系统中的克隆代码，可以提高软件质量、软件可理解性和可维护性。

克隆分析是当前对克隆代码研究的一项重要活动，是对既有克隆代码的分析活动，可以帮助程序人员理解克隆代码。经研究发现克隆代码是会随着软件演化而演化，在其演化过程中克隆代码会对软件系统产生影响，同时也表现出了不同的特征。因此，克隆代码分析可以对克隆代码的维护起到一定的帮助作用。但是当前对克隆分析研究也存在一定的不足之处。如何更深入的分析克隆代码，从而揭示克隆代码所隐含的信息，提取并提炼克隆代码的演化特征？因此，本课题通过克隆代码的演化分析，获取了克隆代码的演化特征，对于维护人员理解克隆代码，理解克隆代码的演化过程以及变化情况都具有积极的意义。

于此同时，克隆维护研究也是克隆研究的一项重要活动。克隆重构是消除克隆代码的重要途径，已有一些法和工具可以帮助实现克隆消除。但重构也不能完全解决克隆代码问题，满足重构条件的克隆代码占极小一部分。同时，对系统中存在的大量克隆代码，有相当数量的会频繁的发生变化。这些变化的克隆代码可能需要一致性的维护，而遗忘这种维护也会导致缺陷的引入。如何识别和判断需要一致性的克隆代码以及如何自动地支持该一致性的维护也是值得研究的问题。因此，本课题通过预测克隆代码的一致性维护需求，识别此类需要维护的克隆代码；并提供自动地维护克隆代码一致性的方法，帮助维护人员维护克隆代码的一致性。

基于以上分析，本项目针对大型软件系统对软件可靠性要求较高的实际应用背景和需求，研究基于演化的克隆代码分析与维护方案，实现对演化克隆的分析和维护。本课题将结合克隆检测和克隆演化的研究成果，重点研究克隆代码演化特征分析，揭示软件系统的克隆代码隐含的信息；并对克隆代码的一致性维护需求进行预测，分别预测在复制粘贴之时和代码变化之时克隆代码的一致性维护需求；针对克隆代码的一致性变化问题，重点研究克隆代码的一致性维护方法，从而实现对克隆代码的分析和维护。本课题对于提高软件质量，使软件更易于理解和维护，不仅具有重要的科学理论意义，还具有重要的实际应用价值。

2 国内外在该方向的研究现状及分析

克隆代码领域一直是软件工程领域的研究热点问题，对克隆代码的研究可以帮助提高软件质量提高软件的可维护性和可理解性。克隆代码也称为重复代码，是软件中彼此相似的代码片段，是软件中最司空见惯、最刺鼻的代码坏味[1]。克隆代码一般是由复制粘贴活动产生的，并有多种原因会导致克隆（开发策略、语言限制和意外克隆等）。软件中存在大量的克隆代码，达到7-21%[2]。

2.1 克隆代码检测

克隆检测是克隆代码领域研究最多内容之一，是指从软件系统中识别克隆代码的过程。目前研究克隆代码检测方法的单位比较多，国际上主要有日本、美国、英国、加拿大等，国内研究克隆代码检测的单位主要有南京大学[3]、大连理工大学[4]和汕头大学[5]等。

目前已经出现了很多检测重复代码的工具，可以分为基于文本、Token、AST、PDG、度量值的方法，尤其是检测语法相似的克隆代码[6-9]，研究表明一个工程也许包含多余20%的语法相似的克隆代码片段。文献[10]通过比较程序的抽象内存状态检测语义相似的克隆代码[11]。文献[12]中同样也涉及到了结构语义相似的克隆代码的检测方法。功能等价克隆代码研究涉及到程序等价[13-15]的问题，而程序等价一般是一个典型的不可判定的问题。基于操作语义上的等价[16,17]在很久以前就已经被提出。而文献[18,19]则是在输入输出行为定义的基础上进行功能等价的重复代码的研究。

克隆检测只能监测系统中的克隆代码，无法解决克隆代码所带来的问题，也无法充分利用克隆代码的优点。本文主要研究重点在克隆分析和克隆维护上，因此简单分析了克隆检测情况。

2.2 克隆代码分析

克隆分析是对克隆分析出克隆代码所隐含的信息，得出或发现一些有助于克隆管理的结论。目前，克隆分析主要研究内容有克隆定义与分类、克隆表示、克隆演化研究、克隆评价分析、克隆代码可视化等主要内容。在当前的研究热点是克隆演化和克隆的评价分析，而未来可能的研究热点是克隆可视化。

2.2.1 克隆定义与表示

克隆代码是指软件系统中彼此相似的代码片段，但目前对克隆代码仍无明确的定义，Roy等人[2]基于文本或功能相似把重复代码划分为如下四类，其中前三类是文本相似的重复代码，第四类是功能等价的重复代码。

类型1：完全相同的代码片段，其中不包括空格、格式和注释。

类型2：语法相同的代码片段，其中不包括标识符、常量、类型。

类型3：拷贝粘贴后修改的代码片段，如改变、增加或删除语句的代码片段。

类型4：执行相同计算，但是用不同语法实现的两个或多个代码片段。

其中，由于第4种重复代码（特别是功能等价但是程序文本差别很大，无法通过静态分析检测到的重复代码）的检测涉及到复杂的程序语义分析，目前还缺少有效的检测方法。

目前克隆是以直接使用代码片段，但是这种表示难以进一步分析克隆代码的，因此研究者提出使用克隆表示描述克隆信息。文献[20]认为克隆需要统一的表示形式，概念模型上分析了克隆的可能情况。文献[21]提出使用克隆区域描述符，并用跟踪克隆随着软件演化而发生的变化。RCF是目前最合适的克隆代码表示形式，给出了相关接口可以进行再次开发。Robert Tairas[23]使用两个或多个克隆代码的属性集合来本地化克隆表示，并可以分析和确定克隆的属性。克隆组织是对克隆代码的统一的组织形式，但目前仍缺少适用的研究成果。在一些工具中，克隆以克隆对的形式表现，体现了克隆代码的克隆关系。Kim使用克隆组和克隆家系[24]来组织克隆代码，克隆组是某一版本软件中相似克隆代码的集合，克隆家系从软件演化的角度刻画了克隆代码的演化情况，是目前最为合适的表示形式。

2.2.2克隆演化

在连续版本的软件中，克隆代码不是孤立存在的，而是伴随着软件的演化而演化的。故此研究者提出了若干种分析克隆代码演化的方法，并通过分析克隆代码的演化过程，总结归纳了几种演化模式，演化模式对帮助维护人员理解克隆代码和维护软件质量有重要意义。

克隆演化过程研究是对克隆代码在演化过程的跟踪分析。Duala提出了使用克隆区域描述符描述克隆代码，并使用它跟踪软件中的克隆代码并分析克隆的演化情况[21,25]。Kim[24,26]首先提出了克隆家系的概念，并使用克隆家系跟踪分析克隆代码的演化过程。文献[27]可以识别克隆代码之间的克隆关系以分析克隆演化过程。文献[28]根据提出了一种克隆映射方法，并提出了额克隆味道的概念分析克隆演化过程。文献[29]可以提取精确和近似克隆的克隆家系情况，并可以识别克隆模式以分析克隆演化过程。

克隆演化模式是克隆代码在演化过程中所表现出来的一些演化特征，通过分析克隆模式可以帮助理解克隆代码。其中，不一致变化是研究者们最感兴趣的克隆模式之一，不一致变化会引起相关缺陷[30-34]。T. Bakota所提出的克隆味道其实是克隆模式的另一种定义，文中分析了若干种克隆味道对软件系统的影响 [35]。另外也有研究者分析了克隆代码迁移[36]、变化次数[37]和延后传播[38]等。研究者在对克隆代码的研究过程中，发现克隆代码要比非克隆代码更稳定，寿命也更为长久，因此出现了对克隆稳定性的相关研究。Krinke Jens[39,40]和Nils[41-43]认为克隆代码是稳定的，不会对维护造成太大的影响，而Roy[44,45]等人认为克隆不一定是稳定在，这与克隆的类型有直接关系。

2.2.3克隆评价

克隆评价是通过对已有克隆代码的分析，对克隆代码有害性、克隆缺陷、克隆的维护代价等方面进行分析，帮助理解克隆代码。传统观点认为克隆代码是有害的[46-50]，认为克隆代码会引发软件缺陷，但近期研究发现也有观点认为克隆代码不一定是有害的[51,53]。因此，克隆代码评价是一个热门的研究问题。

克隆代码会导致软件缺陷，这也是克隆代码危害的最主要体现，人们研究了克隆代码如何导致缺陷。研究人员对克隆代码分析帮助找到与克隆相关的缺陷[54-56]，还利用克隆代码进行错误定位的研究[57,58]。同时，克隆代码会产生一些额外的维护代价[59,60]，这对系统也是不利的，同时也有观点认为克隆代码不会导致额外的缺陷[61,62]，甚至会减少维护代价是系统的可维护性大大提高[63]。也有研究者研究克隆代码的有益性，认为克隆代码是有益的，可以作为复用的一种形式，发现克隆代码具有良好的健壮性，可以提高软件质量[64-67]。

2.2.4克隆可视化

克隆克隆可视化是指将克隆代码信息以直观的方式展示给维护人员，是对克隆代码如何可视化是亟待解决的问题。Stan等人对克隆代码进行检索以满足不同的克隆分析需求，同时将结果可视化显示[68,69]。R. Tairas等人也对克隆可视化做了大量工作，使用可视化和检索技术分析克隆代码 [70, 71]。文献[72,73]提出了一个可视化框架，并在此基础上可视化克隆研究了克隆对耦合和内聚的影响。文献[74-76]使用散点图技术可视化克隆代码是应用最多的一种形式，并使用D-CCFinder一个分布式的大规模分析克隆并揭示经常使用的克隆模式。而Doppel-Code[77]是可视化克隆代码用于分析克隆对真个系统的影响。SoftGUESS[78]实现了一定程度上的克隆代码查询功能，在几个系统特征的基础上查询克隆代码。

2.3 克隆维护

克隆维护是主动地维护克隆代码，是主动地解决克隆代码已经引发的问题或者可能引发的问题，克隆维护操作主要包括克隆重构和克隆管理。克隆重构旨在消除克隆代码，从而消除克隆代码对系统的影响。但随着研究发现重构不都是有效的，克隆维护已从单纯重构开始向克隆代码管理进行转变，并已广泛引起工业界和学术界的重视。克隆管理试图从管理的角度维护克隆代码，对克隆代码不再是简单的消除，而是更加深刻的理解和维护。

2.3.1克隆重构

重构是对既有软件设计的改善甚至是重新设计，是在不改变软件外部行为的条件下优化和改变程序的外部行为[1]。通过定义克隆变化模式来查找可重构的克隆候选，并通过关联规则挖掘对克隆代码进行可重构排序[79]；使用GA算法自动调度适合重构的克隆代码进行重构，从而最大化的改进软件质量[80]使用一个重构代价模型，提出一种极限编程方法对克隆代码进行重构调度[81]。描述了一个工具可以支持grow-and-prune模型在演化软件中识别合并（重构）的相似的函数克隆[82]；Schulze Sandro对克隆代码进行分类，以识别适于重构的克隆代码[83]，并使用相似度计算的方法改进了对重构候选的识别[84]。Balazinska, Magdalena使用对克隆进行重构分析寻找重构机会，帮助系统重构[85-87]。

Yoshida, Norihiro提出了基于度量值的重构方法，可以快速的识别可重构的克隆代码并将其重构，取得了较好的结果[88-92]。同时，研究人员提出了若干种方法重构克隆代码[93-99]。研究者们在对重构后的软件系统进行了分析，并得出了一些相关的结论。通过对重构行后的分析[100]，理解维护人员选择什么样的克隆进行重构并使用什么技术进行重构，从而试图帮助克隆管理更好的进行重构。Choi, Eunjong通过观测重构行为，发现了几种重构模式，并且在系统上进行了验证[101-102]。Tairas, Robert提出子克隆的概念，提出重构部分克隆而不是整体体克隆试图达到一种更好的效果[103,104]。Zibran, Minhaz[105,106]通过对克隆重构的研究发现了一些结论用于帮助继续重构克隆克隆。

2.3.2克隆维护管理方法

但是克隆重构不一定有效，人们开始转向管理的克隆代码管理方法研究，首先有人提出了克隆代码管理的概念模型，并使用克隆变化管理维护克隆，通过跟踪粘贴板活动控制克隆代码，同时也开发了许多克隆代码管理工具。

克隆管理的概念在1997年就已经提出来[107]，但一直没有受到重视。近几年，越来越多的研究人员和开发人员已经开始关注和进行克隆代码的管理和维护，从几种不同的角度分析了克隆管理的研究[108-110]。Rainer Koschke从面向管理的角度将克隆管理分为三条主线：预防性克隆管理、补偿性克隆管理和改正性克隆管理[108]。预防性克隆管理是避免产生新的克隆代码的活动；补偿性克隆管理也称为问题挖掘，是发现并解决克隆代码产生的缺陷和影响；改正性克隆管理对克隆可能产生的问题积极主动去消除克隆，从而尽量避免克隆引发不利影响。综述文献[109]较为系统的分析了克隆避免的主要实现方法、特点和不足之处。在面向工业界研讨会分析了克隆代码管理的各个方面，将克隆管理分为克隆检测、克隆表示、克隆整合、克隆重构、克隆管理等[110]，指出克隆管理是重要的研究方向，其余研究应向克隆管理提供服务。C. K. Roy从各个方面对克隆管理较为详尽阐述，分析了克隆研究领域的发展趋势 [111] 。因此，工业界和学术界都认为克隆管理的将是最新的研究问题，克隆管理的目的是识别当前克隆和组织当前的克隆，消除克隆产生的不利影响，有效利用克隆，并且避免和消除克隆代码。

克隆代码会随着系统的维护也不断地发生变化，在变化的过程中可能会引入缺陷，因此Y. Yamanaka等人对克隆代码在系统的变化进行研究，提出了克隆变化管理的概念，使用变化通知机制帮助维护人员维护克隆代码[112-114]。文献[115]对克隆代码按照一定的标准进行排序，并在此基础上对克隆代码管理给出一定的参考价值。Nguyen对演化中的克隆代码提供了管理功能，是当前对克隆代码管理和克隆一致性管理的克隆管理中较好的方法，但该方法不能处理其它的克隆演化模式，没有提供克隆重构的支持[116-118]。复旦大学通过使用一种克隆代码事件的方法来对克隆代码进行管理[119]，可以监视并管理克隆代码的产生和维护过程。Tairas Robert将克隆代码检测、分析和重构统一起来，来对克隆代码进行维护操作[120]。研究人员通过跟踪粘贴活动来管理和控制克隆代码，并开发了许多插件来支持这种管理和活动。如CLONEBOARD[121]，CnP[122]，CPC[123]，CReN[124]，CSeR[125]。

3 解决的关键问题(研究基础和研究范围)

**研究问题：**本课题通过结合软件演化、机器学习、程序分析方法，如何提取克隆演化中所隐含的信息，即克隆演化特征？ 针对变化的克隆代码，如何对其进行一致性需求维护预测，识别出需要关注的克隆代码及其变化？对需要一致性维护的发生变化的克隆代码，如何提供一致性维护支持方法，自动地维护克隆代码的一致性？

**子问题1：**克隆代码随着系统的演化而演化，在克隆代码及其演化过程中必然隐含了一定的规律和信息，如何揭示隐藏在克隆代码及其演化过程中的克隆代码特征，帮助软件开发和维护人员充分理解和分析克隆代码？

在软件中存在大量克隆代码，我们使用当前的克隆检测工具可以检测系统中大量克隆代码。尽管我们可以检测克隆代码，但是这些克隆代码是否隐藏了一些信息，如何获取这些信息是值得研究的一个问题。同时，克隆代码并不是静态的，是随着软件系统演化而演化。目前对于克隆代码的演化过程和演化模式也有比较深入的研究，但是对克隆演化的研究都集中在克隆演化过程和克隆模式上，演化过程和演化模式中是否存在特定的结论，我们称其为克隆演化特征。克隆代码演化特征是在克隆代码及其演化过程中，所包含或所隐藏的有助于理解和维护克隆代码的一些指导性结论。因此，如何结合克隆演化过程和克隆演化模式，揭示软件中的克隆演化特征，并用于指导克隆代码维护是一个值得研究的问题。

**子问题2：**如何在复制和粘贴之时快速有效的识别出系统中的可能需要一致性维护的克隆代码，并在复制粘贴之时避免此类克隆代码的产生或者复用不需要维护的克隆代码，从而帮助降低软件的维护代价？

系统中存在大量的克隆代码，并且克隆也随系统演化而演化。在这些克隆代码中，存在相当数量的经常发生变化的克隆代码，尤其是发生一致性变化的克隆代码。克隆代码的一致性变化会对软件产生深远的影响，如会引发额外的维护代价，于此同时遗忘这种变化会引入额外的维护代价。因此，如何在克隆代码产生之时即复制粘贴时，预测此克隆代码在未来演化中是否会发生一致性变化是值得研究的问题。即在复制粘贴时预测克隆代码的一致性维护需求仍是一个值得研究问题。在复制粘贴之时预测该克隆代码的一致性维护需求，可以用于克隆代码的预防和避免中。如果其需要一致性维护需求，则不允许该克隆代码的产生，从而可以减少克隆代码导致的维护代价，也可以避免一致性缺陷。同时，在复制粘贴之时预测该克隆代码的一致性维护需求，还可以用于帮助快速开发软件系统，保证软件质量。如果其不需要一致性维护需求，则程序开发人员可以放心大胆的复用该克隆代码，可以提高软件开发速度，同时因为不需要一致性维护也可以提高代码质量。

**子问题3：**如何在克隆代码发生变化之时，快速有效的预测同组克隆代码是否同时需要发生一致性变化，并在发生变化时提示维护人员对克隆代码进行一致性维护，从而避免引入与之相关的一致性缺陷？

系统中存在大量的克隆代码，并且克隆也随系统演化而演化。在这些克隆代码中，存在相当数量的经常发生变化的克隆代码，尤其是发生一致性变化的克隆代码。克隆代码的一致性变化会对软件产生深远的影响，如会引发额外的维护代价，于此同时遗忘这种变化会引入额外的维护代价。因此，如何在克隆代码发生变化之时，预测同组的克隆代码在未来演化中是否会发生一致性变化是值得研究的问题。即在克隆代码变化之时预测同组克隆代码的一致性维护需求是一个值得研究问题。当克隆代码发生变化时，往往同时需要确认同组的克隆代码中是否需要同样的变化，即一致性变化。确认这种变化需求会增加软件的维护代价，维护人员需要花费大量的时间对其进行分析。同时，对于需要一致性维护的克隆代码，遗忘这种一致性维护会直接导致一致性缺陷的产生，将大大降低软件质量，进一步增加软件的维护代价。

**子问题4：**在克隆代码发生变化后，对于需要一致性变化的克隆代码，如何有效地帮助维护人员维护克隆代码的一致性，自动维护发生变化的克隆代码，从而提高软件的可维护性并降低引入一致性缺陷的可能性？

系统中存在大量的克隆代码，并且克隆也随系统演化而演化。在这些克隆代码中，大部分的克隆代码是静止不变的。但同时存在相当数量的经常发生变化的克隆代码，尤其是发生一致性变化的克隆代码。克隆代码的一致性变化会引发额外的维护代价，维护人员需要对同组的克隆代码进行反复确认所发生的变化，同时还要主动且人工的对其进行一致性的维护。这一过程会花费维护人员的大量时间，与此同时人工的确认和维护还容易引发更多的缺陷，将会进一步的导致维护代价的增大。因此，如何在克隆代码发生变化之时，对需要一致性的克隆代码，对其进行自动地一致性变化维护是值得研究的问题。即在克隆代码变化之时对同组克隆代码进行一致性维护是一个值得研究问题。自动地对发生变化的克隆代码进行一致性维护，可以大大地降低软件的维护代价，并同时可以减少一致性缺陷的引入。同时，该方法可以切实的与软件开发环境相结合，在开发过程中实现对克隆代码的维护。

4 课题研究内容、目标、思路及总体方案

4.1 研究内容

克隆代码随着软件系统的演化同时演化，在其演化过程中所表现出的特征称之为克隆演化特征。克隆演化特征对理解和维护克隆代码有极为重要的意义，本文结合机器学习方法，提取相应的度量值对克隆代码进行演化特征分析。在演化过程中，克隆代码的一致性维护问题是影响软件质量的一个重要因素，因此进而研究克隆代码的一致性维护需求。在克隆代码产生和变化之时，分别对其进行一致性维护需求预测，从而避免引入和克隆代码相关的额外维护代价。克隆代码的一致性变化深深影响着软件质量，对克隆代码的一致性变化提供支持，实现克隆代码的一致性维护，可以帮助维护克隆代码，避免引入由于变化导致的克隆一致性缺陷。论文通过对克隆代码的演化特征分析，发现克隆代码一致性问题是影响软件质量原因。通过在复制粘贴和代码变化之时预测克隆代码的一致性维护需求，帮助避免克隆所产生的额外维护代价。通过对克隆代码的一致性维护实现对克隆代码变化的自动维护，提高系统的可维护性和软件质量。

**(1)** **研究基于聚类分析的克隆代码演化特征提取方法**

a) 研究基于演化的克隆静态和演化度量提取与优选；

b) 研究基于聚类的克隆演化特征提取。

**(2)** **研究****基于复制粘贴的克隆代码一致性维护需求预测方法**

a) 研究基于复制粘贴的克隆代码度量提取与优选；

b) 研究基于机器学习的克隆代码一致性维护需求预测。

**(3) 研究****基于贝叶斯网络的克隆代码一致性变化预测方法**

a) 研究基于演化的克隆静态和动态度量提取与优选；

b) 研究基于贝叶斯网络的克隆代码一致性变化预测；

**(4) 研究****基于代码静态分析的克隆代码一致性维护方法**

a) 研究基于代码静态分析的代码差异性分析。

b) 研究基于程序分析的克隆代码一致性维护。

4.2 研究目标

针面向越来越多的领域软件越来越复杂，在演化过程中产生的克隆代码越来越多的背景，本项目针对克隆代码会导致软件系统难于维护、软件质量下降等问题，将克隆分析和克隆维护相结合提出基于演化的克隆代码分析和维护方法，以达到可以提取克隆代码演化特征、预测克隆代码和一致性维护需求，自动维护克隆代码的一致性变化，从而增强软件的可维护性和可理解性、提高软件质量。具体目标如下：

（1）能够实现克隆代码的演化特征提取，分析克隆在演化过程中的隐含信息，从而帮助维护人员理解和维护克隆代码。

（2）能够实现对克隆代码的一致性维护需求预测，分别在复制粘贴之时和代码变化之时对其进行预测，从而帮助避免潜在的克隆一致性缺陷、提高系统的可维护性。

（3）能够实现对克隆代码的一致性变化维护，在克隆代码发生变化时实现对克隆代码的自动维护，避免一致性缺陷的引入、降低克隆代码的维护代价、提高软件质量。

4.3 研究思路

博士课题研究基于演化的克隆代码分析与维护方法，综合考虑软件演化、机器学习和程序分析方法实现对克隆代码的分析和维护。研究思路如图3-1所示：



**图4-1 基于演化的克隆代码分析和维护方法研究框架**

克隆代码随着软件系统的演化同时演化，在其演化过程中所表现出的特征称之为克隆演化特征。克隆演化特征对理解和维护克隆代码有极为重要的意义，本文结合机器学习方法，首先，提取相应的度量值对克隆代码进行演化特征分析。在演化过程中，克隆代码的一致性维护问题是影响软件质量的一个重要因素，因此进而研究克隆代码的一致性维护需求。然后，在克隆代码产生和变化之时，分别对其进行一致性维护需求预测，从而避免引入和克隆代码相关的额外维护代价。最后，克隆代码的一致性变化深深影响着软件质量，对克隆代码的一致性变化提供支持，实现克隆代码的一致性维护，可以帮助维护克隆代码，避免引入由于变化导致的克隆一致性缺陷。

论文通过对克隆代码的演化特征分析，发现克隆代码一致性问题是影响软件质量原因。通过在复制粘贴和代码变化之时预测克隆代码的一致性维护需求，帮助避免克隆所产生的额外维护代价。通过对克隆代码的一致性维护实现对克隆代码变化的自动维护，提高系统的可维护性和软件质量。

4.4 研究方案

既定开题报告中有四个研究内容，分别为克隆演化特征研究、克隆评价研究、克隆可视化研究和克隆维护研究。随着研究的深入，本课题认为克隆评价研究较为主观、且不够深入。因而将其修改为对克隆代码的一致性维护需求预测的两个研究内容：即基于复制粘贴的克隆一致性维护需求预测和基于克隆变化的克隆一致性维护需求预测。同时，将克隆代码维护研究更改为克隆代码一致性维护研究，具体解决克隆代码的一致不一致变化问题。最后，既定开题报告中的克隆可视化研究不再进行深入研究。

因此，现学位论文按照当前的四个子内容进行研究，研究方案如下：

**(1)基于聚类分析的克隆代码演化特征提取（已完成）。**

(a)通过构建克隆家系，分析和提取克隆代码及其演化的相关度量，使用聚类方法分析得出了克隆代码的演化特征。

(b)研究发现，克隆代码在其演化的过程中是较为稳定的，大部分的克隆代码是保持不变的。同时，也有相当数量的克隆代码发生了变化，尤其是一致性变化和不一致变化，这可能会对克隆代码的维护产生影响。

**(2)基于复制粘贴的克隆代码一致性维护需求预测（已完成）。**

(a)对复制粘贴产生的克隆代码进行一致性维护预测；预测在其未来的演化过程中是否会导致额外的维护代价，从而提前避免引入克隆代码的额外维护代价。

(b)通过对复制粘贴操作所导致的克隆代码进行特征提取，提取表示该操作的粘贴代码和复制代码。通过构建克隆家系分析其一致性维护需求，并使用贝叶斯网络对其进行一致性预测。实验表明，所采用的贝叶斯网络可以有效的解决复制粘贴导致的克隆代码的一致性维护需求预测。

**(3)克隆代码一致性变化预测（已完成）。**

(a)系统中存在的克隆代码会发生变化，当发生变化时，对其同组的克隆代码进行一致性维护需求预测；预测同组克隆代码是否需要同时变化，从而避免引入克隆一致性缺陷。

(b)通过对发生变化克隆变化实例进行特征提取，提取表示该变化实例的三组属性特征。通过构建克隆家系收集并分析其一致性维护需求，使用贝叶斯网络对其进行一致性预测。实验表明，所采用的贝叶斯网络可以较好的解决克隆代码变化所引发的一致性维护需求预测。

**(4)克隆代码一致性变化维护（未完成）。**

(a)对系统中需要一致性维护的克隆代码变化，提供一致性维护的支持，使同克隆组内的克隆代码同时发生一致性的维护，从而避免克隆代码的一致性缺陷问题。

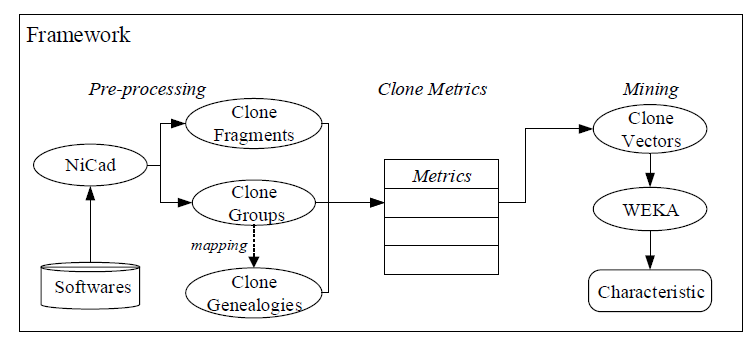
(b)拟通过静态程序分析方法（抽象语法树和程序依赖图），提供对发生变化的克隆代码的差异性分析、变化分析和变化维护，从而达到对克隆代码的一致性维护支持的目的。

5 目前已完成的研究工作及结果

5.1 基于聚类分析的克隆代码演化特征提取

5.1.1 引言

软件中存在大量的克隆代码，随着软件的演化，克隆代码也会随之发生变化，称之为克隆演化。在大量克隆代码和克隆演化中，必然存在着一些隐含的信息，称之为克隆演化特征。克隆演化特征可以帮助维护人员理解和维护克隆代码。为了分析并获得克隆演化特征，本文从三个方面对克隆代码及其演化过程进行分析：克隆片段、克隆组和克隆家系，通过提取不同的度量值表示上述实体，并通过聚类分析提取克隆代码的演化特征。在两个开源系统上进行实验表明，在演化过程中大部分的克隆代码是较为稳定的，有一部分克隆代码甚至是极端稳定而从未发生变化。于此同时，也存在一定数量的发生变化的克隆代码，其中一致性变化的克隆代码要多于不一致变化的克隆代码。

5.1.2 方法

**图5-1克隆演化特征提取框架**

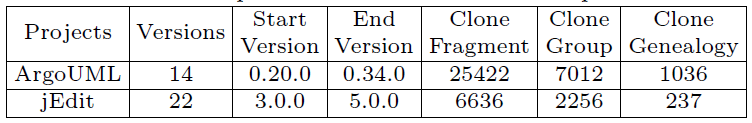
图5-1是克隆演化特征提取框架。克隆演化过程提取可以划分为三个阶段，分别是预处理、度量提取和演化特征分析。预处理阶段使用NiCad检测系统中的克隆代码，并通过克隆片段映射版本间的克隆代码并构建克隆家系。特征提取阶段通过分别提取克隆片段、克隆组和克隆家系的特征，并将其生成为向量。在演化特征分析阶段，使用聚类方法分别对所提取的向量进行聚类分析，识别并提取克隆演化特征。

本文从三个角度对克隆代码进行分析，分别是克隆片段、克隆组和克隆家系，并分别使用不同的度量值表示克隆代码。克隆片段是克隆代码本身的表示，使用 Clone Life、Ischanged和 Change Times三维度量表示克隆代码；克隆组是以组的形式表示克隆代码，使用Group Life、 Static Same、 Add、 Subtract、 Consistent Change、 Inconsistent Change、Split表示克隆代码。克隆家系是克隆组在整个演化过程的表示，使用Genealogy Life、 Static Number、Same Number、Add Number、Subtract Number、Consistent Number、Inconsistent Number、Split Number表示克隆代码。

最后使用WEKA中的X-means聚类方法对克隆代码进行聚类分析，挖掘克隆代码演化特征。

5.1.3 实验

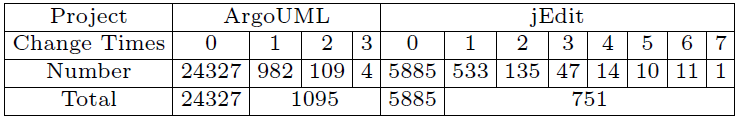
表5-1是列出了本文所使用的实验系统信息，分别为ArgoUML和jEdit。

**表5-1 实验系统信息**

（1）Clone Fragment Experiment

首先对克隆片段进行实验分析，实验分为两个部分，分别是静态统计分析和聚类分析。

静态统计分析结果如表5-2所示。由表中可以看出，大部分的克隆片段没有发生变化（24327 in ArgoUML, and 5885 in jEdit）。只有非常小比例的克隆代码发生了变化（changed(1095 in ArgoUML, and 751 in jEdit）。同时，只有很少部分的克隆代码会发生多次变化。因此，克隆代码在其演化过程中是非常稳定的，只有少部分的克隆代码会发生变化。

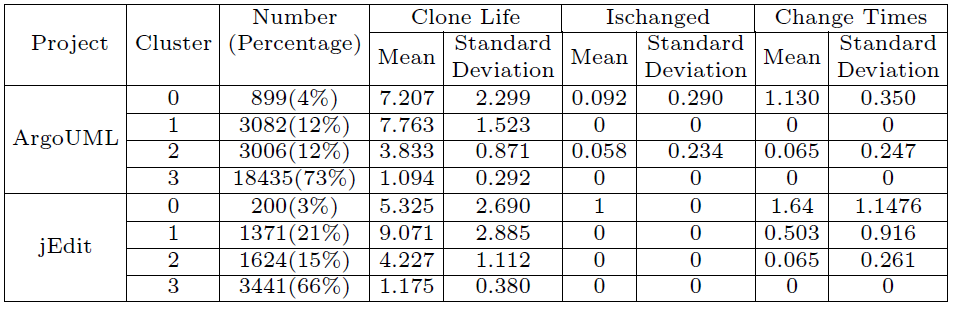
**表5-2 克隆片段静态统计分析结果**

聚类分析结果如表5-3所示。表中Cluster 0是最少的类，表示上一版本发生变化的克隆片段。可以看出只有非常少的克隆代码发生了变化（与静态分析结果相吻合）。观察“isChange”列可以发现，ArgoUML的Cluster1&3和jEdit的Clusters 1, 2&3中克隆代码并没有发生变化。这也意味着大部分的克隆代码是稳定的。两个系统的Cluster3是未发生变化的克隆代码，同时其寿命非常短。意味着克隆代码在刚进入系统中时是标为稳定的，我们需要关注那些在系统中存在一定时间的克隆代码。

因此，在克隆演化中，只有少部分的克隆代码是发生变化的，这些变化也往往发生在其在系统中存在了几个版本之后。

（2）Clone Group Experiment

然后，对克隆组进行实验分析。同样，将实验分成静态统计分析和聚类分析两个部分。

**表5-3 克隆片段聚类分析结果**

静态统计分析如表5-4所示。在表5-4中，使用“Present”和“Absent”分别表示该克隆组是否具有某一特征。同时，我们定义“Static”" 和“Same”为稳定的克隆模式，而其它特征为动态的克隆模式。在两个系统中，大部分的克隆组（72%-85%）拥有稳定的克隆模式， 而只有少部分的克隆组（少于5%）拥有动态的克隆模式。同时，我们也可以看到在系统中存在较多的一致性和不一致变化模式。这应引起程序维护人员的注意，应为此种变化容易导致克隆缺陷。在jEdit中，一致性变化远远多于不一致性变化，在ArgoUML中，一致性变化也多于不一致变化。因此，一致性变化似乎比不一致变化更容易发生。

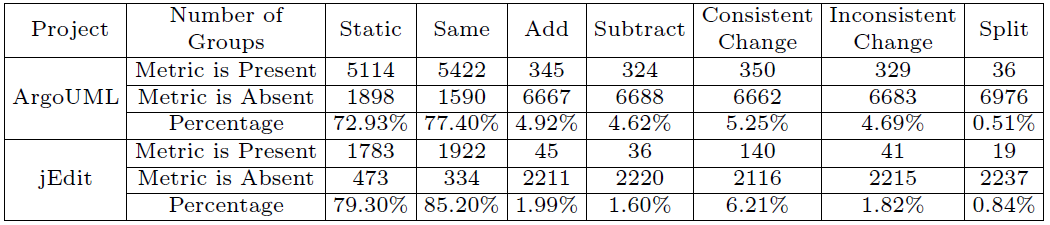
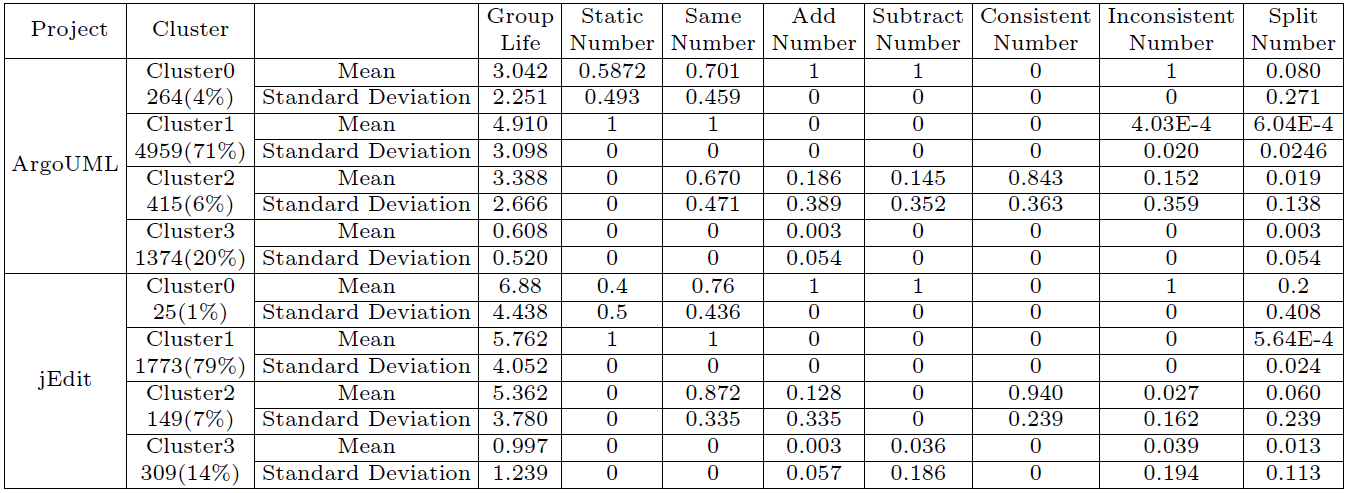
**表5-4 克隆组静态统计分析结果**

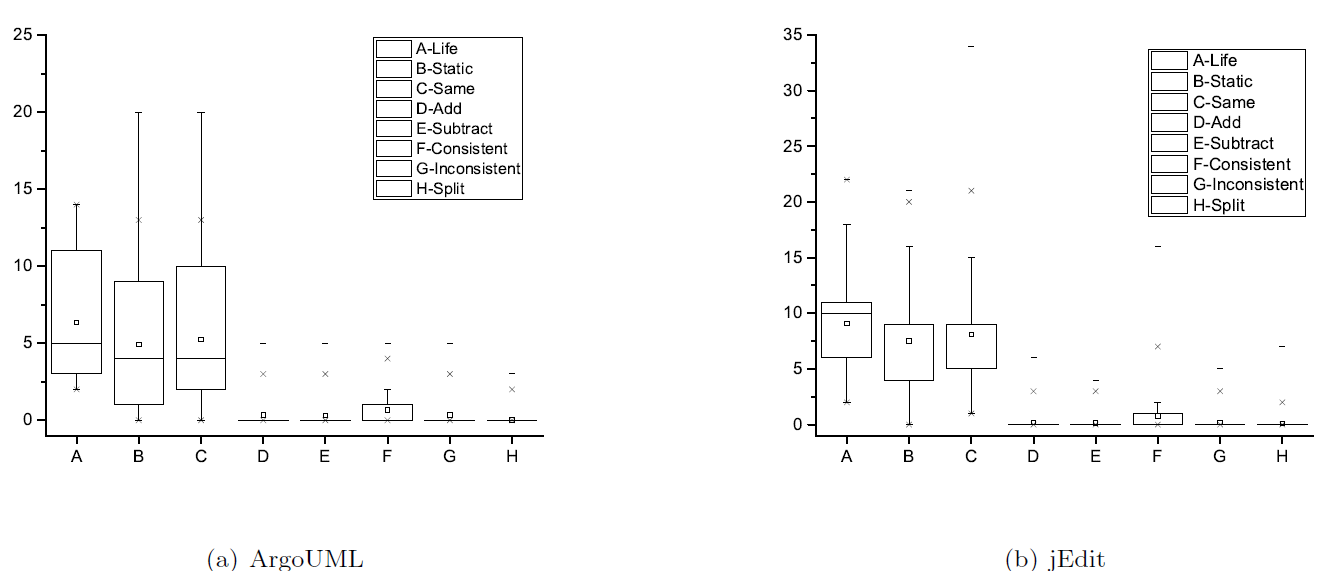
表5-5是克隆组的聚类分析结果。Cluster1是数量最大的克隆组类（71% in ArgoUML, 79% in jEdit），这些克隆组是比较稳定的克隆组（含有大量的稳定克隆模式，不含有动态克隆模式），同时，上述克隆组寿命也大于其他组。大部分的不一致变化模式发生在Cluster0中，Cluster0仅占有了很小的比例（4% in ArgoUML, only 1% in jEdit）。因此，不一致变化并不是频繁的发生。同时，一致性变化模式仅仅发生在Cluster2中，同样是寿命较长的克隆组。因此，动态的克隆模式更容易发生在较长寿命的克隆组中，但是数量较少。如果我们同时检查Cluster 0&2，可以看出含有一致性变化的克隆组数量（Cluster2）要大于含有不一致性变化的克隆组数（Cluster0）。这意味着一致性变化比不一致性变化更容易发生。因此我我们建议程序开发人员应该考虑偏向使得克隆变化的一致性，从而避免导致缺陷的产生。

**表5-5 克隆组聚类分析结果**

因此，克隆组在一般意义上说是较为稳定的。动态的克隆模式更容易发上在克隆组存在系统中一段时间之后，但是也仅仅有少量的动态克隆模式存在。当程序开发人员改变克隆代码时，应该更为细致的考虑是否将该变化传递到其它的克隆片段上以满足一致性变化模式。

（3）Clone Genealogy Experiment

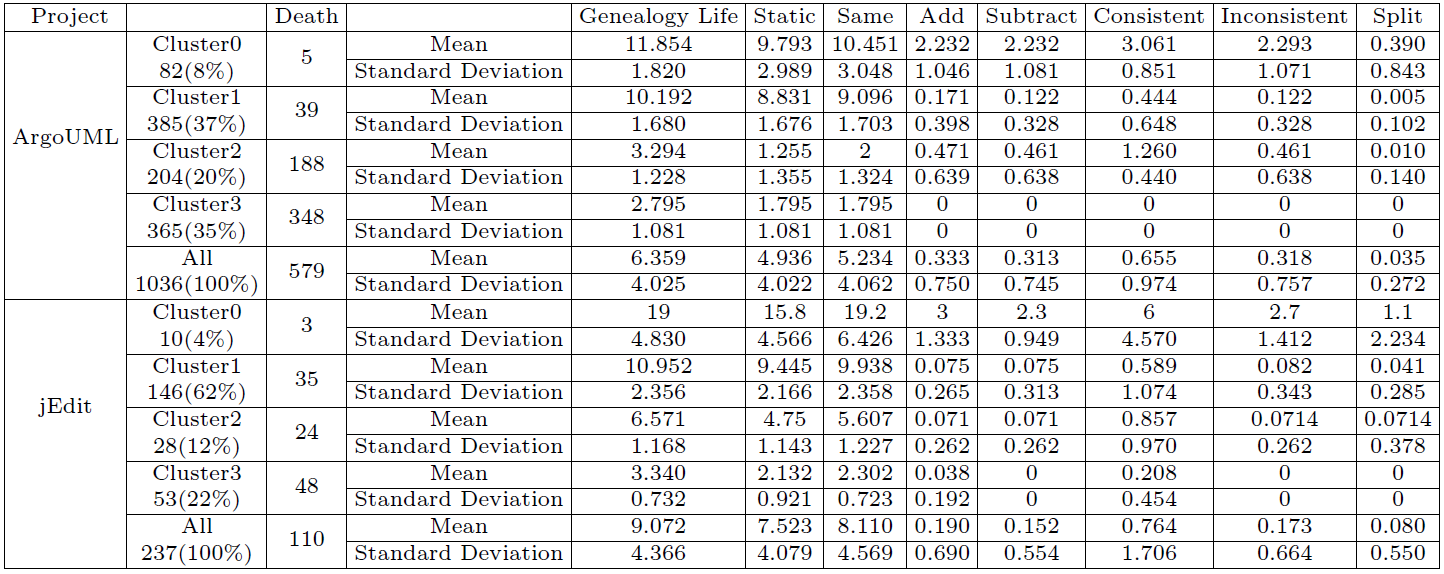
最后，本文还对克隆家系进行实验分析。同样，也将实验划分为静态统计分析和聚类分析。

图5-2是克隆家系静态分析统计结果。从图中可以看出，克隆家系在系统中存在的时间比较长（ArgoUML中有五个版本jEdit中有10个版本），只有很少一部分的克隆家系具有极端长和极端短的寿命。同时，我们也可以看出，态的克隆模式要远远大于动态的克隆家系。尤其是，一致性变化也大于不一致变化的克隆模式。

**图5-2 克隆家系静态分析统计结果**

表5-6是克隆家系聚类分析结果。在表中，我们定义了一个新的度量“Death”，表示该克隆家系在某一聚类中的死亡的数量。根据这个度量，我们可以确定，大部分的克隆家系仍然存活在系统中，因此我们的数据是相对完整的。从表中看出，克隆寿命和克隆静态模式具有强烈的正相关关系。同时，我们定义克隆家系为稳定的克隆家系，该克隆家系具有大量的静态克隆模式，反之为动态的。从图中可以看出，大部分的克隆家系是静态克隆家系。与之相反，只有少量的克隆家系是动态的（Cluster 0）。同时，动态克隆家系尤其是一致和不一致变化往往会发生在寿命较长的克隆家系中。这建议程序维护人员需要各格外的注意寿命较长的克隆家系，原因在于一致和不一致变化更容易引发缺陷。对于Cluster 3，克隆家系的寿命十分短，并且大部分的克隆家系已经死亡。因此，寿命较短的克隆家系中的克隆代码往往更加稳定。程序开打人员需要格外注意哪些寿命较长的克隆家系。

综上所述，克隆家系在演化过程中是较为稳定的。寿命较短的克隆家系尤为稳定。动态变化模式往往发生在寿命较长的克隆家系中。其中，一致性变化比一不一致性变化发生的更为频繁。因此我们建议程序维护人员格外注意长寿面的克隆家系，并且考虑当代码发生变化时需要将此变化统一为一致性变化。

**表5-6 克隆家系聚类分析结果**

5.1.4 结论

我们提出了一个分析提取克隆代码的演化特征的方法，该方法结合代码度量提取、机器学习方法从三个角度对克隆代码进行了分析。所使用的机器学习方法为X-mean聚类方法，在克隆片段、克隆组和克隆家系三个维度上进行了详细的实验分析。通过对两个实验系统的实证研究，研究发现克隆代码在其演化过程中是较为稳定的，并不会经历太多的变化。因此，我们建议程序开发人员需要格外的关注那些寿命较长的克隆代码。同时由于一致性变化比不一致性变化的更容易发生，程序开发人员也应该更需要考虑克隆代码的一致性变化。

5.2 基于复制粘贴的克隆代码一致性维护需求预测

5.2.1 引言

程序开发人员通过复制和粘贴活动产生克隆代码。所产生的克隆代码会随着系统演化，在演化过程中往往需要保证克隆代码的一致性变化，这样就会导致额外的维护代。如果不能保证一致性变化，则有可能引入克隆代码相关的缺陷，这将进一步增大维护的代价。本文将由复制粘贴产生的克隆代码在其演化过程中是否需要发生一致性变化，称为一致性维护需求。因此，在复制粘贴时预测其是否需要一致性维护，可以有效降低克隆所导致的维护代价，帮助提高软件质量。鉴于此，本文定义了由复制粘贴操作产生的克隆代码的一致性变化和一致性维护需求，并在软件开发过程中预测克隆代码的一致性维护需求.通过建立克隆家系收集和分析系统中所有的复制粘贴实例，并提取两组度量值表示复制粘贴实例。在此基础上，使用机器学习方法构建和训练基于复制粘贴操作实时监测和分析的克隆代码一致性维护需求预测模型。在四个开源软件系统上对本文方法进行评估的结果表明，本文方法以较高的准确率和召回率预测克隆代码的一致性维护需求。同时，这个预测过程与软件开发过程相结合，以插件的形式嵌入到集成开发环境中，实现对复制粘贴操作的实时监测和分析，以帮助程序员在软件开发阶段避免需要一致性维护的复制粘贴操作，从而降低克隆代码的维护代价。

5.2.2 方法

为了识别复制粘贴操作，本文首先使用克隆检测工具NiCad从系统中获取克隆代码。然后，使用克隆家系描述克隆代码演化过程。在克隆家系中，第一次出现的克隆代码是由复制粘贴操作导致的克隆代码。通过分析克隆组在克隆家系的演化情况(识别一致性变化模式)，可分析得出该克隆组是否需要一致性维护需求.基于复制粘贴活动的一致性维护预测方法如图5-3所示，分为两个阶段：训练阶段和预测阶段。训练阶段通过收集和分析软件中的克隆代码，构建一致性维护需求预测器。预测阶段使用已训练好的预测器，对新产生的克隆进行一致性维护需求预测。通过训练好的预测器预测一致性维护需求。

为使用机器学习方法，提取两组属性抽象化复制粘贴实例，将之用于模型训练与预测中，分别表示被复制代码的属性和复制粘贴实例的上下文属性。代码属性是被复制代码的属性，可以表示代码信息词法语义信息。代码属性从代码角度描述了被复制代码，包括：克隆粒度、Halstead度量属性、结构属性、参数访问数量、总函数调用、本地函数调用、库函数调用、其它调用。上下文属性是被粘贴代码与被复制代码的关系属性，是两者之间的上下文环境，包括：代码相似度、局部克隆标识、文件名相似度、Masked文件名相似度、方法名相似度、总参数名相似度、最大参数名字相似度、总参数类型相似度、块信息标识。

**图5-3 基于复制粘贴实时监测和分析的克隆一致性维护需求预测框架**

本文方法有两种使用模式：即保守模式和激进模式。保守模式面向不需要一一致性维护的复制粘贴实例。在保守模式下，程序员比较谨慎的执行复制和粘贴操作，仅仅允许不需要一致性维护需求的实例的发生，并且会尽量阻止需要一致性维护需求的实例，从而尽量避免引入额外的维护代价。激进模式面向需要一致性维护需求的复制粘贴实例。在激进模式下，会避免那些需要一致性维护需求的实例，同时尽量允许不需要一致性维护需求的实例产生,从而节约开发时间。

5.2.3 实验设置

为评估本文方法,选择四个开源软件进行评估，其基本信息如表5-7所示。从表中可看出，软件系统中存在大量的复制粘贴实例，数量从633到3366.在所有的软件系统中，大多数复制粘贴实例不需要一致性维护(比例从59.8%到88.47%)。但同时也可看到，软件系统中也存在较多需要一致性维护的复制粘贴实例(数量从73个到1353个)。

**表5-7 开源实验系统信息**

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 不需要一致性维护 | 需要一致性维护 | 总数 |
| ArgoUML | 2574(77.07%) | 766(22.93%) | 3340 |
| JEdit | 560(88.47%) | 73(11.53%) | 633 |
| JFreeChart | 2013(59.80%) | 1353(40.20%) | 3366 |
| Tuxguitar | 1016(71.10%) | 413(28.90%) | 1429 |

本文从两个角度对本文方法进行了评估，贝叶斯网络方法评估和分类器评估。贝叶斯网络实验中，预测方法采用贝叶斯网络，并从三个角度对本文方法进行全面评估，可划分为：全属性实验、属性组实验和交叉验证实验。在分类器评估中，本文实验了几种不同的分类器，包括贝叶斯网络、朴素贝叶斯、决策树和决策树森林分类器。

在保守模式下，对不需要一致性维护需求的复制粘贴实例进行评估。使用三个度量评估该使用模式：推荐率(Recommendation Rate,RR)、准确率(Precision,P)和召回率(Recall，R)。推荐率是方法所允许的复制粘贴实例占全部实例的比例，表明所推荐复制粘贴实例。准确率是在所推荐的复制粘贴实例中，正确推荐的复制粘贴实例的比例，表示预测器的准确性。召回率是所有推荐的复制粘贴实例占真实的不需要一致性维护的复制粘贴实例的比例，表示了预测器的召回性能。

在激进模式下，对需要一致性维护需求的复制粘贴实例进行评估。同样使用三个度量进行评估：警告率(Warning Rate，WR)，准确率(Precision，P)和召回率(Recall，R)。 警告率是方法所警告的复制粘贴实例占全部实例的比例，表明所避免的复制粘贴实例。准确率是在所警告的复制粘贴实例中，正确警告的复制粘贴实例的比例，表示预测器的准确性。召回率是所有警告的复制粘贴实例占真实的需要一致性维护的复制粘贴实例的比例，表示了预测器的召回性能。

5.2.4 贝叶斯方法评估（实验结果和分析1）

本节使用贝叶斯网络作为预测器进行评估。贝叶斯网络预测时，计算得出复制粘贴实例的一个概率值，表示该实例需要一致性维护需求的概率(0~1)。在本节实验中，使用K2搜索算法建立网络结构，并设置贝叶斯网络最大父节点个数为3。同时全属性实验和属性组实验在数据集上使用十倍交叉验证。

（1）全属性实验

全属性实验是使用全部属性在四个实验系统上进行评估。实验评估结果如表5-8所示。由表中可以看出，本文方法在两种模式下均取得了较好效果。在保守模式下，四个系统在不同的阈值的准确度高介于86.6~97.22%之间，同时召回率介于76.97~96.39%。在激进模式下，除JEdit外，其余系统在不同阈值下取得了可以接受的效果，在不同的阈值的准确度介于75.79~94.94%之间，同时召回率介于57.87~80.86%。JEdit系统在激进模式下，预测效果不够理想，其准确度和召回率在50%左右，依然提高了预测的精度。因此，本文方法使用贝叶斯网络作为分类器可以达到一个较好的预测结果。同时，本文建议需要对预测模型进行完全的训练，以达到较好的预测效果。

**表5-8 两种模式实验评估结果**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 保守模式 | | | | 激进模式 | | | |
| 系统 | T | RR(%) | P(%) | R(%) | T(%) | WR(%) | P(%) | R(%) |
|  | 0.01 | 74.61 | 95.91 | 92.85 | 0.9 | 18.95 | 94.94 | 78.46 |
|  | 0.05 | 76.83 | 95.21 | 94.91 | 0.8 | 19.64 | 92.84 | 79.50 |
| ArgoUML | 0.1 | 77.63 | 94.83 | 95.53 | 0.7 | 20.03 | 91.93 | 80.29 |
|  | 0.15 | 78.26 | 94.68 | 96.15 | 0.6 | 20.24 | 91.27 | 80.55 |
|  | 0.2 | 78.47 | 94.66 | 96.39 | 0.5 | 20.54 | 90.23 | 80.81 |
|  | 0.01 | 73.93 | 97.22 | 81.25 | 0.9 | 10.27 | 52.31 | 46.58 |
|  | 0.05 | 78.67 | 95.98 | 85.36 | 0.8 | 12.16 | 48.05 | 50.68 |
| JEdit | 0.1 | 80.73 | 95.11 | 86.79 | 0.7 | 12.80 | 46.91 | 52.05 |
|  | 0.15 | 81.67 | 94.97 | 87.68 | 0.6 | 13.59 | 46.51 | 54.79 |
|  | 0.2 | 82.15 | 94.62 | 87.86 | 0.5 | 14.06 | 47.19 | 57.53 |
|  | 0.01 | 54.69 | 92.40 | 84.50 | 0.9 | 34.25 | 91.67 | 78.12 |
|  | 0.05 | 57.22 | 91.23 | 87.28 | 0.8 | 35.03 | 90.75 | 79.08 |
| JFreeChart | 0.1 | 59.24 | 89.72 | 88.87 | 0.7 | 35.56 | 90.31 | 79.90 |
|  | 0.15 | 60.22 | 89.15 | 89.77 | 0.6 | 36.19 | 89.49 | 80.56 |
|  | 0.2 | 61.14 | 88.87 | 90.86 | 0.5 | 36.57 | 88.87 | 80.86 |
|  | 0.01 | 61.51 | 88.96 | 76.97 | 0.9 | 20.08 | 83.28 | 57.87 |
|  | 0.05 | 66.83 | 87.96 | 82.68 | 0.8 | 21.48 | 81.76 | 60.77 |
| Tuxguitar | 0.1 | 69.28 | 87.47 | 85.24 | 0.7 | 22.74 | 79.38 | 62.47 |
|  | 0.15 | 70.82 | 86.96 | 86.61 | 0.6 | 23.30 | 78.38 | 63.20 |
|  | 0.2 | 72.08 | 86.60 | 87.80 | 0.5 | 24.28 | 75.79 | 63.68 |

**表5-9 保守模式属性组实验评估结果**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 代码属性 | | | 上下文属性 | | |
| 系统 | T | RR(%) | P(%) | R(%) | RR(%) | P(%) | R(%) |
|  | 0.01 | 70.75 | 95.64 | 87.80 | 47.31 | 98.10 | 60.22 |
|  | 0.05 | 73.53 | 95.07 | 90.71 | 61.74 | 96.99 | 77.70 |
| ArgoUML | 0.1 | 74.82 | 94.60 | 91.84 | 69.22 | 96.24 | 86.44 |
|  | 0.15 | 75.93 | 94.32 | 92.93 | 71.53 | 95.90 | 89.01 |
|  | 0.2 | 76.59 | 94.02 | 93.43 | 73.59 | 95.52 | 91.22 |
|  | 0.01 | 72.20 | 96.94 | 79.11 | 32.95 | 99.10 | 54.60 |
|  | 0.05 | 76.30 | 96.48 | 83.21 | 42.19 | 97.25 | 68.60 |
| JEdit | 0.1 | 78.67 | 95.78 | 85.18 | 47.74 | 95.02 | 75.86 |
|  | 0.15 | 80.09 | 95.66 | 86.61 | 52.55 | 93.95 | 82.56 |
|  | 0.2 | 81.36 | 94.76 | 87.14 | 54.10 | 93.36 | 84.45 |
|  | 0.01 | 39.69 | 90.94 | 60.36 | 32.95 | 99.10 | 54.60 |
|  | 0.05 | 43.94 | 87.96 | 64.63 | 42.19 | 97.25 | 68.60 |
| JFreeChart | 0.1 | 46.70 | 86.32 | 67.41 | 47.74 | 95.02 | 75.86 |
|  | 0.15 | 48.07 | 86.16 | 69.25 | 52.55 | 93.95 | 82.56 |
|  | 0.2 | 48.66 | 85.71 | 69.75 | 54.10 | 93.36 | 84.45 |
|  | 0.01 | 56.75 | 89.15 | 71.16 | 29.60 | 93.14 | 38.78 |
|  | 0.05 | 64.52 | 86.98 | 78.94 | 43.88 | 92.34 | 56.99 |
| Tuxguitar | 0.1 | 67.88 | 86.80 | 82.87 | 51.99 | 92.46 | 67.62 |
|  | 0.15 | 69.56 | 86.62 | 84.74 | 56.54 | 91.58 | 72.83 |
|  | 0.2 | 71.24 | 86.05 | 86.22 | 59.69 | 91.56 | 76.87 |

（2）属性组实验

为预测复制粘贴操作的一致性维护需求，本文提取两组度量表示复制粘贴实例。在该实验中，本文每次使用一组度量去预测复制粘贴实例，观察其对实验效果的影响。表5-9和表5-10分别是保守和激进模式下的预测结果。

表5-9为保守模式下的实验评估结果。代码属性实验中，可以看到实验结果依然较好，因此可说明本文提取的代码属性有积极意义。但和全属性实验对比发现，全部的召回率都下降了，准确率除jEdit中少数几个之外也全部下降，因此可以说明上下文属性。在仅有上下文属性的实验结果中，系统出JEdit外，预测结果的准确率提高了，但系统的召回率却大大的降低了。因此在保守模式下，上下文属性对系统的系统的准确率影响较大，代码属性对系统的召回率影响较大，同时都起到了积极的作用。

表5-10为激进模式下的实验结果。代码属性的实验结果明显不如全属性组实验，但仍在可接受的范围之内，说明代码属性有积极的意义，同时上下文属性对激进模式的影响也较大。上下文属性的实验发现，仅仅使用上下文属性在激进模式下，部分系统的实验效果要优于全属性组实验，而部分系统的结果不如全属性组实验。因此，上下文属性和代码属性在激进模式下都具有积极地意义，而上下文属性的影响更大，部分系统仅使用上下文属性更好。因此本文建议维护人员可以根据项目需求或者自身需求适当的选择属性，从而达到不同的要求目标。

**表5-10 激进模式属性组实验评估结果**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 代码属性 | | | 上下文属性 | | |
| 系统 | T | RR(%) | P(%) | R(%) | RR(%) | P(%) | R(%) |
|  | 0.9 | 17.96 | 92.67 | 72.58 | 15.78 | 98.10 | 67.49 |
|  | 0.8 | 18.89 | 89.54 | 73.76 | 16.98 | 95.77 | 70.89 |
| ArgoUML | 0.7 | 19.67 | 87.52 | 75.07 | 17.51 | 94.02 | 71.80 |
|  | 0.6 | 20.21 | 85.93 | 75.72 | 18.50 | 91.59 | 73.89 |
|  | 0.5 | 20.78 | 84.44 | 76.50 | 19.64 | 87.80 | 75.20 |
|  | 0.9 | 9.48 | 55.00 | 45.21 | 31.91 | 96.65 | 76.72 |
|  | 0.8 | 11.37 | 52.78 | 52.05 | 33.87 | 94.47 | 79.60 |
| JEdit | 0.7 | 13.27 | 48.81 | 56.16 | 34.94 | 93.54 | 81.30 |
|  | 0.6 | 14.22 | 45.56 | 56.16 | 35.89 | 92.30 | 82.41 |
|  | 0.5 | 14.85 | 44.68 | 57.53 | 37.20 | 90.97 | 84.18 |
|  | 0.9 | 27.04 | 88.90 | 59.79 | 31.91 | 96.65 | 76.72 |
|  | 0.8 | 28.19 | 87.57 | 61.42 | 33.87 | 94.47 | 79.60 |
| JFreeChart | 0.7 | 28.85 | 86.61 | 62.16 | 34.94 | 93.54 | 81.30 |
|  | 0.6 | 29.92 | 84.61 | 62.97 | 35.89 | 92.30 | 82.41 |
|  | 0.5 | 30.57 | 83.58 | 63.56 | 37.20 | 90.97 | 84.18 |
|  | 0.9 | 17.91 | 82.03 | 50.85 | 11.06 | 93.04 | 35.59 |
|  | 0.8 | 20.22 | 78.55 | 54.96 | 15.40 | 87.27 | 46.49 |
| Tuxguitar | 0.7 | 21.48 | 76.55 | 56.90 | 20.50 | 83.62 | 59.32 |
|  | 0.6 | 23.02 | 74.16 | 59.08 | 23.30 | 80.18 | 64.65 |
|  | 0.5 | 24.14 | 71.88 | 60.05 | 26.94 | 74.81 | 69.73 |

因此，代码属性和上下文属性对预测都起到了积极的作用，并且所发生的作用不一致。在针对不同的系统进行预测时，维护人员可选择不同的属性进行预测。本文未对每一个属性进行分析，仅对两组属性进行了不同的分析。

（3）交叉验证实验

在系统开发的初始阶段，系统内没有足够的数据去训练预测模型。因此，为解决此问题，本文进行了系统交叉实验。使用已有系统的数据训练一致性预测模型，并将其预测本系统的一致性维护需求。实验结果如表5-11所示。

保守模式下，四个系统的准确率和召回率依然达到了较高的水平，准确率在60.01~91.20%之间，召回率56.06~91.43%之间。与全属性组对比发现四个系统的预测效果都有了大幅的下降。 因此，本文建议优先选用系统自身的数据进行一致性为需求预测。在自身系统数据不足以训练的情况下，可以对其进行保守模式的系统交叉验证。在激进模式下，四个系统的预测效果都太差，准确率在15.36~61.01%之间，而召回率则更低，仅在10%徘徊。因此，在激进模式下本文不建议使用系统交叉的方式进行预测。与全属性组对比发现，四个系统的预测效果下降的极为严重(对比于保守模式)， 因此不适合于使用系统交叉的方式进行预测。

**表5-11 两种模式交叉实验评估结果**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | 保守模式 | | | 激进模式 | | | |
| 系统 | T | RR(%) | P(%) | R(%) | T | WR(%) | P(%) | R(%) |
|  | 0.01 | 59.16 | 73.03 | 56.06 | 0.90 | 8.38 | 15.36 | 5.61 |
|  | 0.05 | 67.57 | 75.59 | 66.28 | 0.80 | 10.30 | 15.70 | 7.05 |
| ArgoUML | 0.10 | 71.95 | 76.28 | 71.21 | 0.70 | 12.75 | 20.19 | 11.23 |
|  | 0.15 | 74.10 | 76.40 | 73.47 | 0.60 | 14.52 | 18.76 | 11.88 |
|  | 0.20 | 75.81 | 76.18 | 74.94 | 0.50 | 16.38 | 20.66 | 14.75 |
|  | 0.01 | 78.99 | 91.20 | 81.43 | 0.90 | 2.37 | 33.33 | 6.85 |
|  | 0.05 | 85.62 | 90.41 | 87.50 | 0.80 | 3.95 | 24.00 | 8.22 |
| JEdit | 0.10 | 87.52 | 90.25 | 89.29 | 0.70 | 5.53 | 20.00 | 9.59 |
|  | 0.15 | 89.10 | 89.72 | 90.36 | 0.60 | 6.64 | 19.05 | 10.96 |
|  | 0.20 | 90.21 | 89.67 | 91.43 | 0.50 | 6.95 | 20.45 | 12.33 |
|  | 0.01 | 74.48 | 60.07 | 74.81 | 0.90 | 2.70 | 50.55 | 3.40 |
|  | 0.05 | 82.92 | 60.01 | 83.21 | 0.80 | 4.72 | 61.01 | 7.17 |
| JFreeChart | 0.10 | 86.51 | 60.65 | 87.73 | 0.70 | 6.36 | 54.67 | 8.65 |
|  | 0.15 | 88.38 | 61.01 | 90.16 | 0.60 | 6.95 | 53.42 | 9.24 |
|  | 0.20 | 88.92 | 60.84 | 90.46 | 0.50 | 7.46 | 52.59 | 9.76 |
|  | 0.01 | 59.55 | 75.44 | 63.19 | 0.90 | 6.02 | 53.49 | 11.14 |
|  | 0.05 | 70.40 | 74.16 | 73.43 | 0.80 | 7.98 | 55.26 | 15.25 |
| Tuxguitar | 0.10 | 74.74 | 74.91 | 78.74 | 0.70 | 8.89 | 52.76 | 16.22 |
|  | 0.15 | 78.38 | 74.29 | 81.89 | 0.60 | 10.15 | 49.66 | 17.43 |
|  | 0.20 | 80.76 | 74.00 | 84.06 | 0.50 | 11.69 | 44.31 | 17.92 |

综上所述，在进行一致性维护需求预测时，优先选用自身的数据进行模型训练。当自身数据太少不足以训练模型时，应该尽量使用大量的数据训练模型从而使模型训练完备。同时，不建议使用系统交叉的方式对激进模式下的一致性维护进行预测。

5.2.5 多分类器实验（实验结果和分析2）

为验证本文方法的一般适用性，本文在多个分类方法进行了预测实验。本文选取的四个分类方法分别为：贝叶斯网络、朴素贝叶斯网络、决策树和随机森林。多分类器实验可为进一步验证本文提出的方法，分别计算了在两种模式下的准确率和召回率。分类方法均采用WEKA中实现的方法，并使用其默认参数配置实验。该结果仅表明本文方法的适用性，对某种分类方法还可以进一步的进行分析研究，从而确定针对该问题的最优配置。

**表5-12 多分类器实验评估结果**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | 贝叶斯网络 | | 朴素贝叶斯 | | 决策树 | | 随机森林 | |
| 系统 | 模式 | Precision | Recall | Precision | Recall | Precision | Recall | Precision | Recall |
| ArgoUML | 保守 | 94.5 | 97.4 | 91.7 | 94 | 90.7 | 98.5 | 95.5 | 99.1 |
| 激进 | 90.2 | 80.8 | 78 | 71.4 | 93 | 66.1 | 96.6 | 84.5 |
| 平均 | 93.5 | 93.6 | 88.6 | 88.8 | 91.2 | 91.1 | 95.8 | 95.7 |
| JEdit | 保守 | 94.3 | 47.2 | 94.1 | 85 | 88.5 | 1 | 91.9 | 98.9 |
| 激进 | 91.6 | 57.5 | 33.9 | 58.9 | 0 | 0 | 80 | 32.9 |
| 平均 | 88.9 | 87.7 | 87.1 | 82 | 78.3 | 88.5 | 90.5 | 91.3 |
| JFreeChart | 保守 | 87.9 | 93.2 | 86.3 | 92.7 | 85.7 | 95.8 | 89.2 | 94.7 |
| 激进 | 88.9 | 80.9 | 87.8 | 78 | 92.4 | 76.3 | 91.3 | 83 |
| 平均 | 88.3 | 88.2 | 86.9 | 86.8 | 88.4 | 87.9 | 90.1 | 90 |
| Tuxguitar | 保守 | 86.1 | 91.7 | 85.3 | 85.8 | 89.2 | 94.4 | 88 | 96.3 |
| 激进 | 75.8 | 63.7 | 64.6 | 63.7 | 83.9 | 71.9 | 88.1 | 67.8 |
| 平均 | 83.1 | 83.6 | 79.3 | 79.4 | 87.7 | 87.9 | 88 | 88 |

实验结果如表5-12所示，表中的“平均”是指两种模式下的共同的预测结果。从表中可以看出，四种分类器在四个系统上的平均实验结果都比较好，达到了较好的预测结果，准确率和召回率均在78.3~95.8%之间。在保守模式和激进模式中，除JEdit系统之外，剩余三个系统都达到较好的预测效果。JEdit系统，由于样本数据太少，模型训练依然不够完全，因此，实验结果不够理想。综上，本文的方法可适用于本文方法可以适用于一般的分类和预测方法。维护人员可以根据自己的需求选取相应的预测方法，并需要进一步的实验验证和调整该预测方法的参数，从而获取最佳的效果。

5.2.6 结论

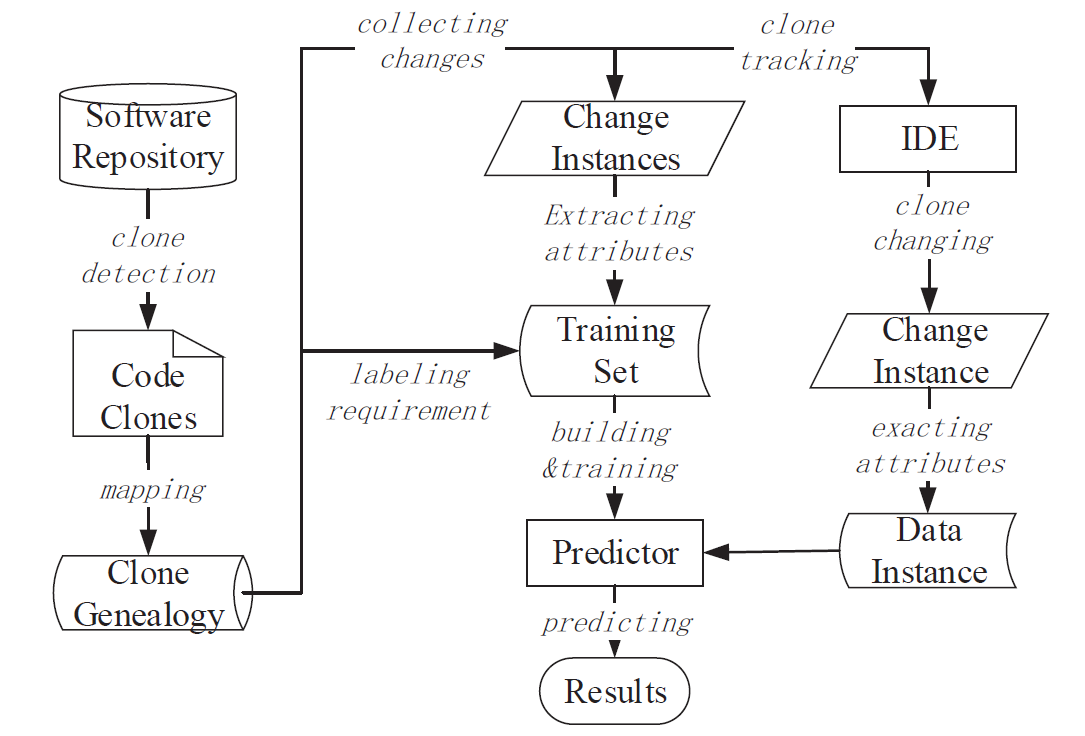
为了避免由复制粘贴操作而产生的额外的维护代价，本文对提出一种基于复制粘贴实时监测和分析的克隆代码一致性维护需求预测方法。该方法首先通过构建克隆家系收集系统中的复制粘贴实例，并分析其演化过程，从不同的角度使用代码属性和上下文属性这两组属性特征表示复制和粘贴实例，使用WEKA提供的方法训练预测模型，然后设计实现了一个Eclipse插件CCP集成到软件开发环境中,实时监测程序员在软件开发过程中的复制粘贴操作，避免有害的复制粘贴操作的产生，利用训练好的预测模型对该复制粘贴操作进行一致性维护需求的预测。在四个不同规模的开源系统上进行的方法评估实验结果表明，本文方法可以以较高的准确率和召回率预测克隆代码的一致性维护需求。

5.3 基于贝叶斯网络的克隆代码变化预测

5.3.1 引言

克隆代码是软件开发过程中必可避免的一种软件现象，克隆代码的存在也影响着软件的质量。而克隆组和克隆家系可以帮助程序开发人员理解克隆的演化过程。但由于克隆代码之间的相似性，对某一克隆代码的变化可能需要传递到同组的其它克隆片段中，称之为克隆一致性变化。倘若没有对其进行一致性变化，会引入克隆相关缺陷。因此，本文提出一种方法在克隆代码发生变化之时，预测其同组克隆代码的一致性维护需求，即是否同时需要一致性变化。本文通过构建克隆家系收集系统的一致性变化和不一致变化实例，并抽取三组度量值表示变化实例，包括：代码属性、上下文属性和演化属性。本文的预测器是基于WEKA中的贝叶斯网络实现。本文在三个开源系统上进行了实证研究，研究发现本文方法可以以较高的准确率和召回率预测克隆代码的一致性维护需求。本文方法可以应用到软件开发过程中，帮助避免软件的一致性缺陷，从而提高软件质量。

5.3.1 方法

本文方法可以划分为两个步骤：预测器构建和预测阶段。在构建阶段，本文首先收集系统中所有变化实例（一致性变化和不一致性变化），并使用其训练贝叶斯网络的预测器。在预测阶段，使用训练好的贝叶斯网络预测克隆代码的一致性维护需求。图4-4是本文方法的框架。

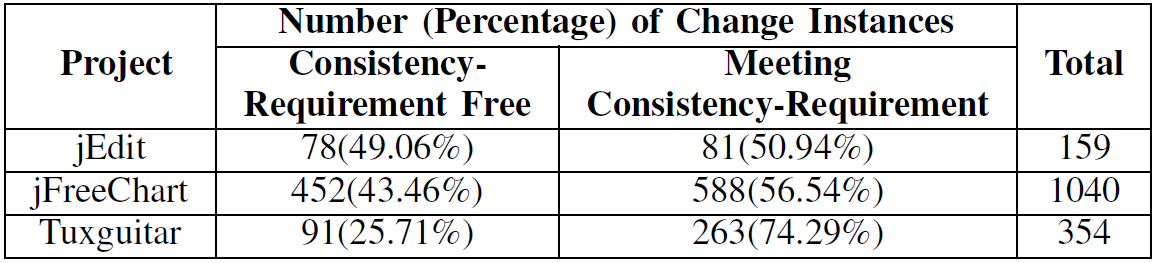
**图5-4 基于贝叶斯网络的克隆代码一致性维护需求预测**

首先是收集克隆变化实例。使用NiCad检测系统中的克隆代码，并映射版本间的克隆代码构建克隆家系。然后，通过比较两个版本间的克隆代码，识别其是否发生了变化以及其演化模式。然后，分别提取三组度量值表示变化实例，分别为代码属性、上下文属性和历史属性。代码属性包括克隆组大小、Halstead度量、语法信息度量。上下文属性包括代码相似度、文件名相似度、方法名相似度、参数名相似度、参数类型相似度、最大参数名和类型相似度。历史属性包括寿命、历史模式、当前模式、历史变化统计。

在收集克隆变化实例并提取相应的度量值之后，本文获得了实验训练所需要的训练集。并使用WEKA中提供的贝叶斯网络对其进行训练和实验。

5.3.3 实验设置

本文所采用的实验系统如表5-13所示，采用三个开源系统对本文方法进行实证研究，分别是：jEdit、jFreeChart、Tuxguitar。从表中可以看出，在每一个系统中都存在大量的变化实例，从159-1040个。表中第二列和第三列分别给出了不需要和需要一致性维护的克隆代码变化实例的统计情况。可以看出变化实例中满足一致性维护需求的占全部变化实例的51%-74%。

**表5-13 实验系统统计信息**

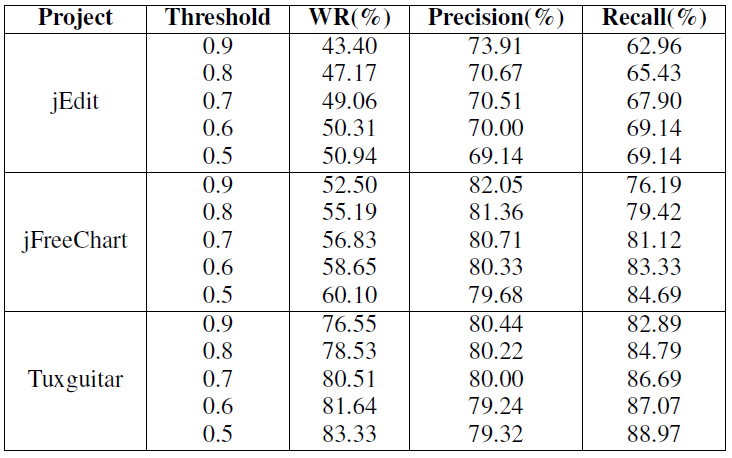
我们使用WEKA提供的模型构建本文的预测模型，在模型构建阶段使用K2算法学习网络结构并使用SimpleEstimator对网络结构的参数进行评估计算。贝叶斯网络模型会输出一个预测值，该值表示了该变化实例需要一致性维护的可能性大小。

为了评估本文方法， 我们分别进行了三组实验，分别为全属性实验、属性组实验和交叉验证实验。属性组实验分析不同属性组的重要程度。交叉验证实验使用训练好的模型去预测新系统，探索将本文方法应用到新系统的可能性。。并使用三个度量值评估方法的有效性：警告率(Warning Rate，WR)，准确率(Precision，P)和召回率(Recall，R)。 警告率是方法所警告的复制粘贴实例占全部实例的比例，表明所避免的复制粘贴实例。准确率是在所警告的复制粘贴实例中，正确警告的复制粘贴实例的比例，表示预测器的准确性。召回率是所有警告的复制粘贴实例占真实的需要一致性维护的复制粘贴实例的比例，表示了预测器的召回性能。

5.3.4 实验结果与分析

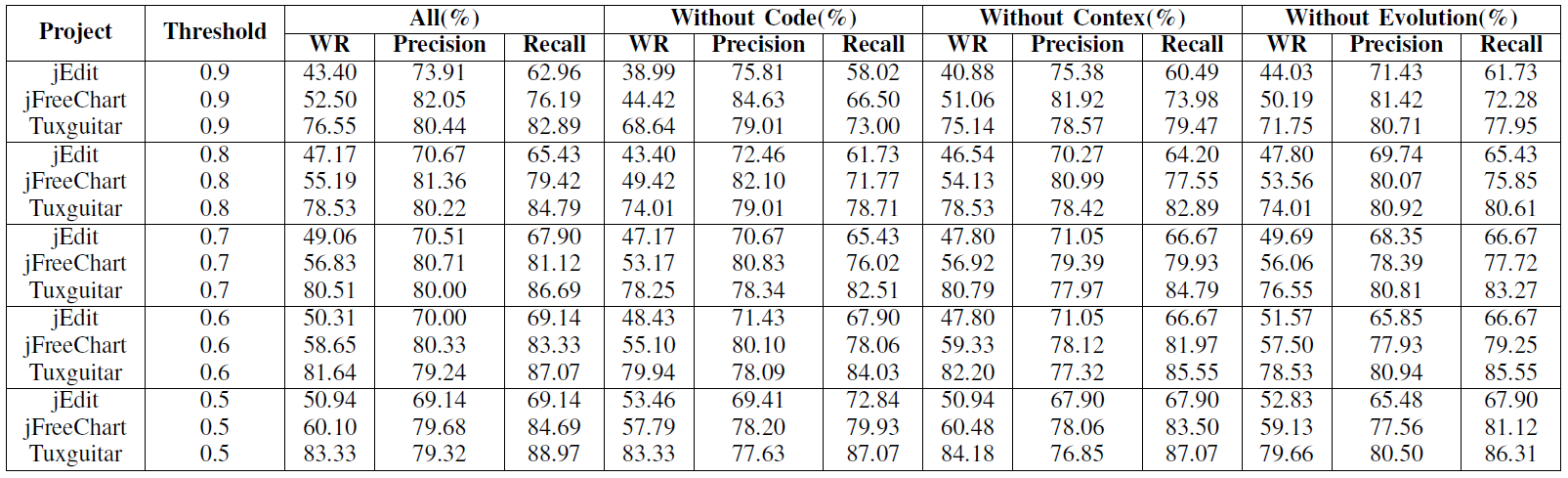
（1） 全属性实验

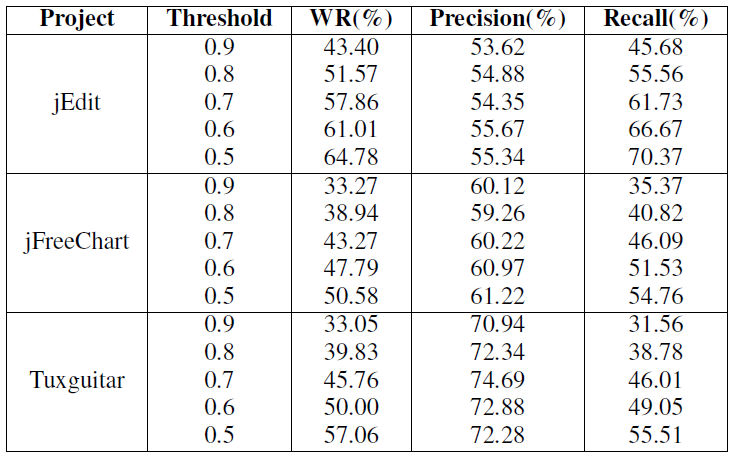
首先，本文进行全属性实验，对所有系统使用所有的度量值进行一致性维护需求评估。表5-14是全属性实验结果。由表中可以看出，我们的预测模型在jFreeChart and Tuxguitar上有较高的准确率和召回率，同时在jEdit的准确率和召回率也是可以接受的。对于所有的系统，我们的放大在警告率是有很好的表现，几乎约等于满足一致性维护需求的变化实例的比例。观察阈值对其的影响发现，召回率更容易受到阈值的影响。这表明本文方法有较好的稳定性，但是还值得进一步研究如何提高其召回率。

**表5-14 全属性实验结果**

（2）属性组实验

在该实验中，本文每次使用删除一组度量去预测其一致性维护需求，观察其对实验效果的影响。表5-15是预测结果。从表中可以看出，每一组度量值都会对其产生正面且积极的影响，但每一个属性组所产生的影响是不同的。程序开发人员可以根据自己的需求按需选择相应的度量值。

**表4-15 属性组实验结果**

**表4-16 交叉验证实验结果**

（3）交叉验证实验

在系统开发的初始阶段，系统内没有足够的数据去训练预测模型。因此，为解决此问题，本文进行了系统交叉实验。使用已有系统的数据训练一致性预测模型，并将其预测本系统的一致性维护需求。实验结果如表5-16所示。从表中可以看出，所有的预测结果均遭受了剧烈的下降，已导致该模型不能很好对其预测。即模型交叉验证可以得出，我们方法依赖于某一个系统自身的数据，而不具有一般性。我们建议程序维护人员使用单独的系统进行预测。

5.3.5 结论

软件中存在的克隆代码会对软件产生一定的影响，尤其是当克隆代码发生变化的时候。因此，本文提出了一种在克隆代码发生变化之时对其进行一致性维护需求预测的方法。本文通过构建克隆家系并分析其整个演化过程，用于识别并提取系统中全部的变化实例。对每一个变化实例，我们提取了三组度量值表示该变化实例：代码度量、上下文度量和历史度量。并使用WEKA贝叶斯网络训练我们的预测器。实验结果表明本文方法可以以较高的准确率和召回率预测克隆代码的一致性变化维护需求。同时，本文的方法还可以方便的嵌入到软件开发环境中，帮助程序开发人员预测克隆代码的一致性变化，从而避免克隆一致性缺陷，提高软件质量。

6 预期创新点

学位论文包括以下预期创新点：

（1）结合机器学习方法（聚类方法）和克隆演化分析对克隆代码、克隆群和克隆家系进行聚类分析，揭示了克隆代码的演化特征，对于帮助理解和维护克隆代码具有重要意义。

（2）使用机器学习方法对克隆代码进行一致性维护需求预测，分别在复制粘贴和代码变化时对其进行了预测，有效地防止了克隆缺陷的引入，也降低了对克隆代码的维护代价。

（3）结合程序静态分析（抽象语法树和依赖图）对克隆变化进行差异性分析， 并支持对克隆变化的一致性维护功能，降低了维护人员的维护代价，提高软件质量。

7 后期拟完成的研究工作及进度安排标

2016年10月-2017年3月：研究基于程序分析的克隆代码一致性维护方法，并撰写相关学术论文；

2017年4月-2017年10月：撰写博士毕业论文，准备论文答辩。

8 结论

本课题对系统中存在的克隆代码进行分析和维护研究，充分考量了克隆代码伴随着软件演化而演化的过程，并结合软件度量提取、机器学习方法，对克隆代码进行了深入的分析，主要结论如下：

（1）所提出的克隆代码的演化特征的方法，结合代码度量提取和聚类分析从三个角度对克隆代码进行了分析：克隆片段、克隆组和克隆家系。研究发现克隆代码在其演化过程中是较为稳定的，并不会经历太多的变化。但同时也存在一定数量的克隆变化，尤其是一致性和不一致性变化。因此，本文建议程序开发人员需要格外的关注那些寿命较长的克隆代码。同时由于一致性变化比不一致性变化的更容易发生，程序开发人员也应该更需要考虑克隆代码的一致性变化。

（2）所提出的基于复制粘贴析的克隆代码一致性维护需求预测方法，可以避免由复制粘贴操作而产生的额外的维护代价。方法通过构建克隆家系收集系统中的复制粘贴实例，从不同的角度使用代码属性和上下文属性这两组属性特征表示复制和粘贴实例，并使用机器学习方法训练预测模型，对该复制粘贴操作进行一致性维护需求的预测。在四个不同规模的开源系统上进行实验，实验结果表明本文方法可以以较高的准确率和召回率预测克隆代码的一致性维护需求。

（3）所提出的在克隆代码一致性变化预测方法，可以在克隆发生变化之时对其进行一致性维护需求预测。通过构建克隆家系并分析其整个演化过程提取系统中全部的变化实例，并提取三组度量值表示该变化实例：代码度量、上下文度量和历史度量。实验结果表明本文方法可以以较高的准确率和召回率预测克隆代码的一致性变化维护需求。同时，本文的方法还可以方便的嵌入到软件开发环境中，帮助程序开发人员预测克隆代码的一致性变化，从而避免克隆一致性缺陷、提高软件质量。

在已经完成的研究中，本文分析了克隆演化特征，得出了克隆偏稳定的结论，同时也存在大量的一致性不一致变化。由于该变化可能会引入新的克隆缺陷并大大提高了克隆代码的维护代价，因此对克隆代码的一致性变化进行了预测，分别在复制粘贴和代码变化时进行了预测。预测结果可以用于真正的软件工程实践中，并可以帮助维护人员理解和维护克隆代码。但是，对克隆代码的一致性维护仍然需要由程序人员进行，同样会使得软件的维护代价提高。因此，本文下一个研究内容将定位到克隆代码的一致性维护中，以实现对克隆代码的自动高效的一致性维护，从而降低克隆的维护代价，提高软件质量。

9 发表论文情况

1. Zhang Fanlong, Su Xiaohong, Zhao Wen, Wang Tiantian. An empirical study of code clone clustering based on clone evolution[J]. Journal of Harbin Institute of Technology. （第一作者，录用待发表，EI）
2. Zhang Fanlong, Khoo Siau-cheng, Su Xiaohong, Machine-Learning Aided Analysis of Clone Evolution[J], Chinese Journal of Electronics. (第一作者，录用待发表，SCI)
3. Zhang Fanlong, Khoo Siau-cheng, Su Xiaohon. Predicting Consistent Clone Change. [C]// Proceedings of 27th International Symposium on Software Reliability Engineering, October 23-26, 2016, Ottawa, Canada. （第一作者，录用待发表，CCF推荐B类会议）
4. 苏小红，张凡龙. 克隆管理的关键科学问题及其研究进展[J].计算机学报.（第二作者导师为第一作者，大修，EI一级学报）
5. Su Xiaohong, Zhang Fanlong, Li Xia, et al. Functionally Equivalent C Code Clone Refactoring by Combining Static Analysis with Dynamic Testing[C]//Proceedings of International Conference on Soft Computing Techniques and Engineering Application. Springer India, 2014: 247-256.（第二作者，导师为第一作者，已发表，EI）
6. Yuan Yue，Zhang Fanlong，Wang Tiantian，SuXiaohong. CloneAy: A Tool for Code Clone Analyzation. [C]// Proceedings of the 3rd International Conference on Information Science and Control Engineering, July 8-10, 2016, Beijing, China. （第二作者，录用待发表，EI）
7. 张凡龙, 苏小红, 李智超, 等. 基于支持向量机的克隆代码有害性评价方法[J]. 智能计算机与应用，2016, 6(4).（第一作者，已发表，核心）
8. 张凡龙, 袁悦, 苏小红. 基于复制粘贴监测的克隆一致性维护预测.计算机研究与发展.（第一作者，已完成拟投稿，EI一级学报）
9. Zhang Fanlong, Khoo Siau-cheng, Su Xiaohong. Predicting Consistency-Requirement for Clone Change. [J] Journal of Systems and Software. （第一作者，已完成拟投稿，SCI）
10. 张凡龙，何蔷，苏小红.克隆代码可视化方法研究[J].工大学报.（第一作者，已完成拟投稿，EI）。10 主要参考文献

[1] [美]Joshua Kerievsky著，杨光，刘基诚译，重构与模式（Refactoring to Ptterns），人民邮电出版社，2010年12月第2版.

[2] Roy, C. K., Cordy, J. R., & Koschke, R. (2009). Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. Science of Computer Programming, 74(7), 470–495. doi:10.1016/j.scico.2009.02.007

[3] 张耀衡.软件维护中基于多重程序模型的克隆代码检测技术研究[D].南京大学硕士学位论文.2003.

[4] 刘恒.程序中克隆代码的自动检测[D].大连理工大学硕士学位论文.2003.

[5] 李建忠.克隆代码检测技术的研究[D].汕头大学硕士学位论文.2005.

[6] B. S. Baker. On ﬁnding duplication and near-duplication in large software systems[C]. In Working Conference on Reverse Engineering (WCRE). 1995:86–95.

[7] I. D. Baxter, A. Yahin, L. Moura, M. Sant’Anna, and L. Bier. Clone detection using abstract syntax trees[C]. In ICSM. 1998:368–377.

[8] L. Jiang, G. Misherghi, Z. Su, and S. Glondu. Deckard: Scalable and accurate tree-based detection of code clones[C]. In ICSE. 2007:96–105.

[9] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: a multilinguistic token-based code clone detection system for large scale source code[C]. TSE. 2002, 28(7):654–670.

[10] H. Kim, Y. Jung, S. Kim, and K. Yi. MeCC: memory comparison-based clone detector[C]. In: 33rd International Conference on Software Engineering, ACM. 2011:301-310.

[11] M. Gabel, L. Jiang, Z. Su. Scalable detection of semantic clones[C]. In: Proceedings of the 30th International Conference on Software Engineering, ICSE 2008. 2008:321–330.

[12] 王甜甜. 结构语义相似的程序识别方法研究[D]. 哈尔滨工业大学博士论文. 2009.

[13] G. Cousineau and P. Enjalbert. Program equivalence and provability [J]. In MFCS: Symposium on Mathematical Foundations of Computer Science. 1979:237-245.

[14] Limin Jia, Jianzhou Zhao, Vilhelm Sjöberg, and Stephanie Weirich. Dependent Types and Program Equivalence[C]. POPL '10 Proceedings of the 37th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. 2010:275-286.

[15] Sudipta Kundu, Zachary Tatlock, Sorin Lerner. Proving Optimizations Correct using Parameterized Program Equivalence[C]. PLDI '09 Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation. 2009:327-337.

[16] A. M. Pitts. Operational semantics and program equivalence [J]. In Applied Semantics, Advanced Lectures, volume 2395 of LNCS, Tutorial, Springer-Verlag.2002:378–412.

[17] J.-C. Raoult, J. Vuillemin. Operational and semantic equivalence between recursive programs [J]. Journal of the ACM. Oct. 1980, 27(4):772–796.

[18] M. Bertran, F.-X. Babot, A. Climent. An input/output semantics for distributed program equivalence reasoning[C]. Electr. Notes Theor. Comput. Sci. 2005, 137(1):25–46.

[19] R. L. Crole and A. D. Gordon. A sound metalogical semantics for input/output effects [J]. In Computer Science Logic (CSL): 8th workshop, volume 933 of LNCS. 1995:339-353.

[20] J. Harder, “The limits of clone model standardization,” 2013 7th Int. Work. Softw. Clones, pp. 10–11, May 2013.

[21] E. Duala-Ekoko and M. P. Robillard, “Clone Region Descriptors: Representing and Tracking Duplication in Source Code,” ACM Trans. Softw. Eng. Methodol. vol. 20, no. 1, pp. 1–31, Jun. 2010.

[22] Harder, J., & Göde, N. (2011). Efficiently handling clone data. In Proceeding of the 5th international workshop on Software clones - IWSC ’11 (p. 81). New York, New York, USA: ACM Press. http://doi.org/10.1145/1985404.1985427

[23] R. Tairas, F. Jacob, and J. Gray, “Representing clones in a localized manner,” Proceeding 5th Int. Work. Softw. Clones - IWSC ’11, p. 54, 2011.

[24] M. Kim, V. Sazawal, D. Notkin, and G. Murphy, “An empirical study of code clone genealogies,” ACM SIGSOFT Softw. …, pp. 187–196, 2005.

[25] E. Duala-Ekoko and M. P. Robillard, “Tracking Code Clones in Evolving Software,” 29th Int. Conf. Softw. Eng., pp. 158–167, May 2007.

[26] M. Kim and D. Notkin, “Using a clone genealogy extractor for understanding and supporting evolution of code clones,” Proc. 2005 Int. Work. Min. Softw. Repos. - MSR ’05, pp. 1–5, 2005.

[27] J. Krinke, N. Gold, Y. Jia, and D. Binkley, “Distinguishing copies from originals in software clones,” in Proceedings of the 4th International Workshop on Software Clones - IWSC ’10, 2010, pp. 41–48.

[28] T. Bakota, “Tracking the evolution of code clones,” SOFSEM 2011 Theory Pract. Comput. …, pp. 86–98, 2011.

[29] R. Saha, C. Roy, and K. Schneider, “An automatic framework for extracting and classifying near-miss clone genealogies,” … Maint. (ICSM), 2011 …, pp. 293–302, 2011.

[30] N. Bettenburg, W. Shang, W. M. Ibrahim, B. Adams, Y. Zou, and A. E. Hassan, “An empirical study on inconsistent changes to code clones at the release level,” Sci. Comput. Program. vol. 77, no. 6, pp. 760–776, Jun. 2012.

[31] K. Inoue, Y. Higo, N. Yoshida, E. Choi, S. Kusumoto, K. Kim, W. Park, and E. Lee, “Experience of finding inconsistently-changed bugs in code clones of mobile software,” in 2012 6th International Workshop on Software Clones (IWSC), 2012, pp. 94–95.

[32] J. Krinke, “A Study of Consistent and Inconsistent Changes to Code Clones,” 14th Work. Conf. Reverse Eng. (WCRE 2007), pp. 170–178, Oct. 2007.

[33] L. Yin, L. Zhang, M. Hou, and D. Liu, “A Novel Approach for Predicting the Probability of Inconsistent Changes to Code Clones Based LDA,” in Proceedings of the 2014 International Conference on Computer, Communications and Information Technology, 2014, no. 2011, pp. 118–122.

[34] N. Bettenburg, W. Shang, W. Ibrahim, B. Adams, Y. Zou, and A. E. Hassan, “An Empirical Study on Inconsistent Changes to Code Clones at Release Level,” 2009 16th Work. Conf. Reverse Eng., pp. 85–94, 2009.

[35] T. Bakota, R. Ferenc, and T. Gyimothy, “Clone Smells in Software Evolution,” Softw. Maintenance, 2007. …, pp. 24–33, 2007.

[36] S. Xie, F. Khomh, Y. Zou, and I. Keivanloo, “An empirical study on the fault-proneness of clone migration in clone genealogies,” in 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), 2014, pp. 94–103.

[37] N. Gode and R. Koschke, “Frequency and risks of changes to clones,” Softw. Eng. (ICSE), 2011 33rd …, 2011.

[38] L. Barbour, F. Khomh, and Y. Zou, “Late propagation in software clones,” 2011 27th IEEE Int. Conf. Softw. Maint. pp. 273–282, Sep. 2011.

[39] J. Krinke, “Is Cloned Code More Stable than Non-cloned Code?” 2008 Eighth IEEE Int. Work. Conf. Source Code Anal. Manip. pp. 57–66, Sep. 2008.

[40] J. Krinke, “Is cloned code older than non-cloned code?” Proceeding 5th Int. Work. Softw. Clones - IWSC ’11, p. 28, 2011.

[41] J. Harder and N. Göde, “Cloned code: stable code,” J. Softw. Evol. Process, 2012.

[42] N. Gode and J. Harder, “Clone Stability,” in 2011 15th European Conference on Software Maintenance and Reengineering, 2011, pp. 65–74.

[43] N. Göde and J. Harder, “Oops! . . . I changed it again,” in Proceeding of the 5th international workshop on Software clones - IWSC ’11, 2011, p. 14.

[44] M. Mondal, C. K. Roy, M. S. Rahman, R. K. Saha, J. Krinke, and K. A. Schneider, “Comparative stability of cloned and non-cloned code,” in Proceedings of the 27th Annual ACM Symposium on Applied Computing - SAC ’12, 2012, no. 2, p. 1227.

[45] C. K. Roy, “A Change-Type Based Empirical Study on the Stability of Cloned Code.”

[46] F. Gauthier, T. Lavoie, and E. Merlo, “Uncovering access control weaknesses and flaws with security-discordant software clones,” in Proceedings of the 29th Annual Computer Security Applications Conference on - ACSAC ’13, 2013, pp. 209–218.

[47] X. Wang, Y. Dang, L. Zhang, D. Zhang, E. Lan, and H. Mei, “Can I clone this piece of code here?,” Proc. 27th IEEE/ACM Int. Conf. Autom. Softw. Eng. - ASE 2012, p. 170, 2012.

[48] W. Hordijk, M. Ponisio, and R. Wieringa, “Harmfulness of code duplication-a structured review of the evidence,” Conf. or Work. Pap. (Full Pap. Talk), vol. 609, no. 638, pp. 1–10, 2009.

[49] Y. Higo, K. Sawa, and S. Kusumoto, “Problematic Code Clones Identification Using Multiple Detection Results,” in 2009 16th Asia-Pacific Software Engineering Conference, 2009, pp. 365–372.

[50] A. Lozano, M. Wermelinger, and B. Nuseibeh, “Evaluating the Harmfulness of Cloning: A Change Based Experiment,” Fourth Int. Work. Min. Softw. Repos. (MSR’07ICSE Work. 2007), pp. 18–18, May 2007.

[51] C. Kapser and M. Godfrey, “‘Cloning Considered Harmful’ Considered Harmful,” 2006 13th Work. Conf. Reverse Eng., pp. 19–28, 2006.

[52] C. Kapser and M. Godfrey, “‘Cloning considered harmful’ considered harmful: patterns of cloning in software,” Empir. Softw. Eng., 2008.

[53] F. Rahman, C. Bird, and P. Devanbu, “Clones: what is that smell?” Empir. Softw. Eng., pp. 72–81, May 2012.

[54] K. Jalbert and J. S. Bradbury, “Using clone detection to identify bugs in concurrent software,” 2010 IEEE Int. Conf. Softw. Maint. pp. 1–5, Sep. 2010.

[55] N. Yoshida, T. Hattori, and K. Inoue, “Finding similar defects using synonymous identifier retrieval,” in Proceedings of the 4th International Workshop on Software Clones - IWSC ’10, 2010, pp. 49–56.

[56] D. Lo, L. Jiang, and A. Budi, “Active refinement of clone anomaly reports,” 2012 34th Int. Conf. Softw. Eng., pp. 397–407, Jun. 2012.

[57] D. Chatterji and B. Massengill, “Measuring the efficacy of code clone information: An empirical study,” Proc. …, 2010.

[58] D. Chatterji, J. C. Carver, B. Massengil, J. Oslin, and N. a. Kraft, “Measuring the Efficacy of Code Clone Information in a Bug Localization Task: An Empirical Study,” in 2011 International Symposium on Empirical Software Engineering and Measurement, 2011, pp. 20–29.

[59] A. Lozano and M. Wermelinger, “Assessing the effect of clones on changeability,” 2008 IEEE Int. Conf. Softw. Maint. pp. 227–236, Sep. 2008.

[60] J. Harder and R. Tiarks, “A controlled experiment on software clones,” in 2012 20th IEEE International Conference on Program Comprehension (ICPC), 2012, pp. 219–228.

[61] YunLin, Xing, Z., Peng, X., Liu, Y., Sun, J., Zhao, W., & Dong, J. (2014). Clonepedia: Summarizing Code Clones by Common Syntactic Context for Software Maintenance. In ICSME 2014.

[62]Thummalapenta, S., Cerulo, L., Aversano, L., & Di Penta, M. (2009). An empirical study on the maintenance of source code clones. Empirical Software Engineering, 15(1), 1–34.

[63]Monden, A., Nakae, D., Kamiya, T., Sato, S., & Matsumoto, K. (2002). Software quality analysis by code clones in industrial legacy software. In Proceedings Eighth IEEE Symposium on Software Metrics (pp. 87–94). IEEE Comput. Soc.

[64]Basit, H. a., Rajapakse, D. C., & Jarzabek, S. (2005). Beyond templates: a study of clones in the STL and some general implications. Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005. 451–459.

[65]Bousse, E., Combemale, B., & Baudry, B. (2014). Scalable Armies of Model Clones through Data Sharing, 2014.

[66]CAI, D., & Kim, M. (2011). An Empirical Study of Long-Lived Code Clones, 432–446. http://doi.org/10.1007/978-3-642-19811-3\_30

[67]Moriwaki, T., Yamanaka, Y., Igaki, H., & Yoshida, N. (2014). Towards an Analysis of Who Creates Clone and who reuses it, 63.

[68] Z. Xing, Y. Xue, and S. Jarzabek, “CloneDifferentiator: Analyzing clones by differentiation,” in 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), 2011, pp. 576–579.

[69] H. A. Basit, S. Jarzabek, D. Anh, and M. Low, “Query-based filtering and graphical view generation for clone analysis,” in 2008 IEEE International Conference on Software Maintenance, 2008, pp. 376–385.

[70] R. Tairas, F. Jacob, and J. Gray, “Representing clones in a localized manner,” Proceeding 5th Int. Work. Softw. Clones - IWSC ’11, p. 54, 2011.

[71] R. Tairas and J. Gray, “An information retrieval process to aid in the analysis of code clones,” Empir. Softw. Eng., vol. 14, no. 1, pp. 33–56, Sep. 2008.

[72] Z. M. Jiang, A. E. Hassan, and R. C. Holt, “Visualizing Clone Cohesion and Coupling,” 2006 13th Asia Pacific Softw. Eng. Conf., pp. 467–476, 2006.

[73] Z. M. Jiang and A. E. Hassan, “A Framework for Studying Clones In Large Software Systems,” Seventh IEEE Int. Work. Conf. Source Code Anal. Manip. (SCAM 2007), pp. 203–212, Sep. 2007.

[74] R. K. Saha, C. K. Roy, and K. A. Schneider, “Visualizing the evolution of code clones,” in Proceeding of the 5th international workshop on Software clones - IWSC ’11, 2011, p. 71.

[75] S. Livieri, Y. Higo, M. Matushita, and K. Inoue, “Very-Large Scale Code Clone Analysis and Visualization of Open Source Programs Using Distributed CCFinder: D-CCFinder,” 29th Int. Conf. Softw. Eng., pp. 106–115, May 2007.

[76] J. R. Cordy, “Exploring Large-Scale System Similarity Using Incremental Clone Detection and Live Scatterplots,” 2011 IEEE 19th Int. Conf. Progr. Compr. pp. 151–160, Jun. 2011.

[77] C. Forbes, I. Keivanloo, and J. Rilling, “Doppel-Code: A Clone Visualization Tool for Prioritizing Global and Local Clone Impacts,” 2012 IEEE 36th Annu. Comput. Softw. Appl. Conf., pp. 366–367, Jul. 2012.

[78] E. Adar and M. Kim, “SoftGUESS: Visualization and Exploration of Code Clones in Context,” 29th Int. Conf. Softw. Eng., pp. 762–766, May 2007.

[79] M. Mandal, C. K. Roy, and K. A. Schneider, “Automatic ranking of clones for refactoring through mining association rules,” in 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), 2014, pp. 114–123.

[80] S. Lee, G. Bae, H. S. Chae, D. Bae, and Y. R. Kwon, “Automated scheduling for clone-based refactoring using a competent GA,” Softw. Pract. Exp., vol. 41, no. 5, pp. 521–550, Apr. 2011.

[81] M. F. Zibran and C. K. Roy, “A Constraint Programming Approach to Conflict-Aware Optimal Scheduling of Prioritized Code Clone Refactoring,” 2011 IEEE 11th Int. Work. Conf. Source Code Anal. Manip. pp. 105–114, Sep. 2011.

[82] T. Mende, R. Koschke, and F. Beckwermert, “An evaluation of code similarity identification for the grow-and-prune model,” J. Softw. Maint. Evol. Res. Pract., vol. 21, no. 2, pp. 143–169, Mar. 2009.

[83] S. Schulze and M. Kuhlemann, “Advanced Analysis for Code Clone Removal ∗,” Work. Software-Reengineering, 2009.

[84] S. Schulze, M. Kuhlemann, and M. Rosenmüller, “Towards a refactoring guideline using code clone classification,” in Proceedings of the 2nd Workshop on Refactoring Tools - WRT ’08, 2008, pp. 1–4.

[85] M. Balazinska, E. Merlo, M. Dagenais, B. Lague, and K. Kontogiannis, “Advanced clone-analysis to support object-oriented system refactoring,” in Proceedings Seventh Working Conference on Reverse Engineering, 2000, pp. 98–107.

[86] M. Balazinska, E. Merlo, M. Dagenais, B. Lague, and K. Kontogiannis, “Partial redesign of Java software systems based on clone analysis,” in Sixth Working Conference on Reverse Engineering (Cat. No.PR00303), 1999, pp. 326–336.

[87] M. Balazinska, E. Merlo, M. Dagenais, B. Lague, and K. Kontogiannis, “Measuring clone based reengineering opportunities,” in Proceedings Sixth International Software Metrics Symposium (Cat. No.PR00403), 1999, pp. 292–303.

[88] E. Choi, N. Yoshida, T. Ishio, K. Inoue, and T. Sano, “Extracting code clones for refactoring using combinations of clone metrics,” Proceeding 5th Int. Work. Softw. Clones - IWSC ’11, p. 7, 2011.

[89] E. Choi, N. Yoshida, and T. Ishio, “Finding Code Clones for Refactoring with Clone Metrics: A Case Study of Open Source Software,” Proc. Inst. Electron. Infoormation Commun. Eng., pp. 53–57, 2011.

[90] Y. Higo, S. Kusumoto, and K. Inoue, “A metric-based approach to identifying refactoring opportunities for merging code clones in a Java software system,” J. Softw. Maint. Evol. Res. Pract., vol. 20, no. 6, pp. 435–461, Nov. 2008.

[91] Y. Higo, T. Kamiya, S. Kusumoto, K. Inoue, and T. Agency, “ARIES: REFACTORING SUPPORT ENVIRONMENT BASED ON CODE,” in ASTED Conf. on Software Engineering and Applications. 2004, pp. 222–229.

[92] A. Goto, N. Yoshida, M. Ioka, E. Choi, and K. Inoue, “How to extract differences from similar programs? A cohesion metric approach,” in 2013 7th International Workshop on Software Clones (IWSC), 2013, pp. 23–29.

[93] G. Krishnan and N. Tsantalis, “Unification and refactoring of clones,” … Maintenance, Reengineering …, 2014.

[94] Barbosa, F. S., & Aguiar, A. (2013). Removing Code Duplication with Roles. 2013 IEEE 12th International Conference on Intelligent Software Methodologies, Tools and Techniques (SoMeT), 37–42.

[95] Goto, A., Yoshida, N., Ioka, M., Choi, E., & Inoue, K. (2013). How to extract differences from similar programs? A cohesion metric approach. In 2013 7th International Workshop on Software Clones (IWSC) (pp. 23–29).

[96] Higo, Y., Kamiya, T., Kusumoto, S., Inoue, K., & Agency, T. (2004). ARIES: REFACTORING SUPPORT ENVIRONMENT BASED ON CODE. In ASTED Conf. on Software Engineering and Applications. (pp. 222–229.).

[97] Higo, Y., Kusumoto, S., & Inoue, K. (2008). 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, 20(6), 435–461.

[98] Koni-N’Sapu, G. G. (2001). A Scenario Based Approach for Refactoring Duplicated Code in Object Oriented Systems.

[99] Volanschi, N. (2012). Safe clone-based refactoring through stereotype identification and iso-generation. In 2012 6th International Workshop on Software Clones (IWSC) (pp. 50–56).

[100] S. Bazrafshan, “No clones, no trouble?” in 2013 7th International Workshop on Software Clones (IWSC), 2013, pp. 37–38.

[101] E. Choi, N. Yoshida, and K. Inoue, “What kind of and how clones are refactored?” in Proceedings of the Fifth Workshop on Refactoring Tools - WRT ’12, 2012, pp. 1–7.

[102] E. Choi, N. Yoshida, and K. Inoue, “An Investigation into the Characteristics of Merged,” Choi E, Yoshida N, Inoue K. An Investig. Into Charact. Merged Code Clones Dur. Softw. Evol. No. 5, pp. 1244–1253, 2014.

[103] R. Tairas and J. Gray, “Sub-clone refactoring in open source software artifacts,” Proc. 2010 ACM Symp. Appl. Comput. - SAC ’10, p. 2373, 2010.

[104] R. Tairas and J. Gray, “Sub-clones: Considering the Part Rather than the Whole,” … Conf. Softw. Eng. Res. …, pp. 284–290, 2010.

[105] M. F. Zibran, R. K. Saha, C. K. Roy, and K. A. Schneider, “Genealogical insights into the facts and fictions of clone removal,” ACM SIGAPP Appl. Comput. Rev., vol. 13, no. 4, pp. 30–42, Dec. 2013.

[106] M. F. Zibran, R. K. Saha, C. K. Roy, and K. A. Schneider, “Evaluating the conventional wisdom in clone removal,” in Proceedings of the 28th Annual ACM Symposium on Applied Computing - SAC ’13, 2013, p. 1123.

[107] Lague, B., Proulx, D., Mayrand, J., Merlo, E. M., & Hudepohl, J. (1997). Assessing the benefits of incorporating function clone detection in a development process. In Proceedings International Conference on Software Maintenance (pp. 314–321). IEEE Comput.

[108] Koschke, R. (2008). Frontiers of software clone management. In 2008 Frontiers of Software Maintenance (pp. 119–128). IEEE.

[109]Ali, A., & Sulaiman, S. (2014). A Systematic Literature Review of Code Clone Prevention Approaches. International Journal of Software Engineering …, 1–6. Retrieved from

[110]Koschke, R., Baxter, I., Conradt, M., & Cordy, J. (2012). Software Clone Management towards Industrial Application (Dagstuhl Seminar 12071). Dagstuhl Reports, 2(2), 21–57.

[111]Roy, C. C. K.., Zibran, M. M. F., & Koschke, R. (2014). The vision of software clone management: Past, present, and future (Keynote paper). 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE), 18–33.

[112] Y. Yamanaka, E. Choi, N. Yoshida, K. Inoue, and T. Sano, “Applying clone change notification system into an industrial development process,” in 2013 21st International Conference on Program Comprehension (ICPC), 2013, pp. 199–206.

[113] Y. Yamanaka, E. Choi, N. Yoshida, K. Inoue, and T. Sano, “Industrial application of clone change management system,” 2012 6th Int. Work. Softw. Clones, pp. 67–71, Jun. 2012.

[114] Y. Higo, Y. Ueda, S. Kusumoto, and K. Inoue, “Simultaneous Modification Support based on Code Clone Analysis,” in 14th Asia-Pacific Software Engineering Conference (APSEC’07), 2007, pp. 262–269.

[115] R. D. Venkatasubramanyam, S. Gupta, and H. K. Singh, “Prioritizing code clone detection results for clone management,” in 2013 7th International Workshop on Software Clones (IWSC), 2013, pp. 30–36.

[116] T. T. Nguyen, H. a. Nguyen, N. H. Pham, J. M. Al-Kofahi, and T. N. Nguyen, “Cleman: Comprehensive Clone Group Evolution Management,” 2008 23rd IEEE/ACM Int. Conf. Autom. Softw. Eng., pp. 451–454, Sep. 2008.

[117] H. A. Nguyen, T. T. Nguyen, N. H. Pham, J. Al-Kofahi, and T. N. Nguyen, “Clone Management for Evolving Software,” IEEE Trans. Softw. Eng., vol. 38, no. 5, pp. 1008–1026, Sep. 2012.

[118] T. T. Nguyen, H. A. Nguyen, N. H. Pham, J. M. Al-Kofahi, and T. N. Nguyen, “Clone-Aware Configuration Management,” 2009 IEEE/ACM Int. Conf. Autom. Softw. Eng., pp. 123–134, Nov. 2009.

[119] G. Zhang, X. Peng, Z. Xing, and W. Zhao, “Towards contextual and on-demand code clone management by continuous monitoring,” in 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2013, pp. 497–507.

[120] R. Tairas and J. Gray, “Increasing clone maintenance support by unifying clone detection and refactoring activities,” Inf. Softw. Technol., vol. 54, no. 12, pp. 1297–1307, Dec. 2012.

[121] M. de Wit, A. Zaidman, and A. van Deursen, “Managing code clones using dynamic change tracking and resolution,” 2009 IEEE Int. Conf. Softw. Maint. pp. 169–178, Sep. 2009.

[122] D. Hou, P. Jablonski, and F. Jacob, “CnP: Towards an environment for the proactive management of copy-and-paste programming,” in 2009 IEEE 17th International Conference on Program Comprehension, 2009, pp. 238–242.

[123] V. Weckerle and F. Universit, “CPC: an eclipse framework for automated clone life cycle tracking and update anomaly detection,” Master’s thesis, Freie Univ. Berlin, Ger. 2008. 2008.

[124] P. Jablonski and D. Hou, “CReN: A Tool for Tracking Copy-and-Paste Code Clones and Renaming Identifiers Consistently in the IDE,” in Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange - eclipse ’07, 2007, pp. 16–20.

[125] F. Jacob, D. Hou, and P. Jablonski, “Actively comparing clones inside the code editor,” Proc. 4th Int. Work. Softw. Clones - IWSC ’10, pp. 9–16, 2010.