博士学位论文

基于机器学习的克隆代码分析与一致性维护方法研究

RESEARCH ON CODE CLONE ANALYSIS AND CONSISTENCY MAINTENANCE BASED ON MACHINE LEARNING

张凡龙

哈尔滨工业大学

2017 年 4 月

国内图书分类号：TP311.5　　学校代码：10213
国际图书分类号：004.41　　密级：公开

工学博士学位论文

基于机器学习的克隆代码分析与一致性维护方法研究

博士研究生：张凡龙

导　　师：苏小红教授

申请学位：工学博士

学　　科：计算机科学与技术

所在单位：计算机科学与技术学院

答辩日期：2017 年 4 月

授予学位单位：哈尔滨工业大学

Classified Index: TP311.5
U.D.C: 004.41

Dissertation for the Doctoral Degree in Engineering

RESEARCH ON CODE CLONE ANALYSIS AND CONSISTENCY MAINTENANCE BASED ON MACHINE LEARNING

Candidate: Zhang Fanlong
Supervisor: Prof. Su Xiaohong
Academic Degree Applied for: Doctor of Engineering
Specialty: Computer Science and Technology
Affiliation: School of Computer Science and Technology
Date of Defence: April, 2017
Degree-Conferring-Institution: Harbin Institute of Technology

摘 要

研究表明软件系统中存在大量的克隆代码，即根据某种相似性定义彼此相似的代码片段。传统观点将克隆代码视为一种“代码坏味”，认为其存在可能会影响软件的质量、可理解性和可维护性。研究人员因而提出和开发了多种不同的克隆代码检测方法与工具，可以高效且快速的检测系统中的克隆代码。然而，在克隆代码随着系统进行演化的过程中，克隆检测无法解决克隆演化以及克隆变化对系统产生的影响问题，所以克隆分析和维护研究变得尤为重要。克隆分析可以帮助程序开发人员理解系统中存在的克隆代码，提高软件的可理解性；克隆维护则可以帮助解决克隆代码所引发的问题，切实提高软件质量和可维护性。因此，克隆代码分析和维护研究对于帮助提高软件系统的质量、增强软件的可理解性和可维护性，具有重要的科学理论意义和实际应用价值。

在克隆代码的演化过程中，克隆代码及其演化过程往往会表现出一些特征，本文称之为克隆代码演化特征。其中，引发研究人员强烈关注的是克隆代码在演化过程中的变化，即一致性变化和不一致性变化。克隆变化可能会导致与之相关的克隆代码一致性缺陷以及额外的维护代价，从而降低软件质量和可维护性。因此，本文在克隆代码演化的基础上结合机器学习方法研究克隆代码分析和一致性维护方法，通过提取分析克隆代码的演化特征分析，帮助程序开发人员理解克隆代码，通过预测克隆代码的一致性维护需求解决克隆代码的一致性维护问题，帮助程序开发人员维护克隆代码。

针对演化中的克隆代码难于理解和分析的问题，研究并提出了基于聚类的克隆代码演化特征分析方法，使用克隆演化特征帮助程序开发人员程序分析和理解克隆代码。首先，使用克隆检测工具检测系统中的克隆代码，并构建系统所有克隆代码的克隆家系用于描述克隆代码的演化过程。然后，从三个不同维度提取相应的度量值描述克隆代码及其演化过程，即克隆片段、克隆组和克隆家系。最后，使用聚类分析方法聚类克隆代码并挖掘克隆代码演化特征，帮助开发人员理解克隆代码及其演化过程。研究结果发现大部分的克隆代码在演化过程中是稳定的，但也存在相当数量的克隆代码发生变化，其中发生一致性变化的克隆代码比发生不一致性变化的克隆代码数量多一些。

针对创建的克隆代码在其演化中的一致性变化往往会导致额外的维护代价问题，研究并提出了基于贝叶斯网络的克隆代码创建时一致性维护需求预测方法。本文将复制粘贴创建的克隆代码称为克隆创建实例，将其在未来演化过程中所发生

的一致性变化称为克隆创建时一致性维护需求。首先，通过检测系统的克隆代码并构建克隆家系收集系统中的克隆创建实例。然后，提取两组不同的度量值表示克隆创建实例，即代码属性和上下文属性。最后，使用贝叶斯网络训练预测模型，并预测克隆代码创建时一致性维护需求。实验结果表明本文预测方法可以高效地预测克隆代码的一致性维护需求，从而可以帮助程序开发人员降低克隆代码的一致性维护代价。

针对克隆代码变化时遗忘克隆代码一致性变化会导致克隆一致性缺陷问题，研究并提出了基于贝叶斯网络的克隆代码变化时一致性维护需求预测方法。本文将克隆代码发生变化的克隆代码称为克隆变化实例，并将其未来演化过程中所发生的一致性变化称为克隆变化时一致性维护需求。首先，通过检测系统的克隆代码并构建系统克隆家系收集系统中的克隆变化实例。然后，提取三组不同的度量值用于表示克隆变化实例，即代码属性、上下文属性和演化属性。最后，使用贝叶斯网络训练预测模型，并预测克隆代码变化时一致性维护需求。实验结果表明本文预测方法可以有效地预测克隆代码的一致性维护需求，可以帮助程序开发人员避免克隆代码一致性缺陷。

针对克隆代码一致性维护需求预测能否扩展至其它机器学习方法和如何结合软件开发过程进行预测问题，研究并提出了基于不同机器学习方法的克隆代码一致性维护需求实证研究方法。本文将克隆代码创建和变化实例统称为克隆实例，并将其一致性维护需求统称为克隆代码一致性维护需求。首先，检测系统的克隆代码并构建克隆家系收集克隆实例，并使用不同的属性组表示克隆实例。然后，将克隆代码的一致性维护预测扩展到其它五种不同的机器学习方法中。最后，将克隆代码一致性维护需求预测与软件开发过程相结合，实现了一个 eclipse 插件可以实现边开发边预测克隆代码的一致性维护需求。实验结果表明本文预测方法可以适用于不同的机器学习方法中，且支持向量机方法更适合于克隆一致性预测。可以将本文方法嵌入到软件开发环境中，使得克隆一致性需求预测与软件开发过程相结合，帮助程序开发人员避免克隆代码一致性缺陷和降低克隆维护代价。

综上所述，本文提出的基于机器学习的克隆代码分析与一致性维护方法，为解决克隆代码分析和理解、克隆代码一致性维护、避免克隆代码的相关缺陷、降低克隆代码的维护代价、提高软件质量和可维护性等问题提供了一种新思路和新方法。

关键词：克隆代码；克隆分析与维护；克隆演化特征；一致性需求预测；一致性变化；机器学习

Abstract

Studies have shown that there are a large number of code clones in softwares, that is, code fragments that are similar to each other according to some similarity measure. The traditional viewpoint regards code clones as a famous “bad smell”, believing that presence of code clones will affect the quality, comprehensibility, and maintainability of the software. Researchers have proposed and developed a variety of different clone detection methods and tools that can efficiently and quickly detect code clones from softwares. However, clone detection can not solve the problem of clone evolution and the effect of clone changes on the software when the code clones evolving as the software’s evolution, so that research on clone analysis and maintenance becomes particularly important. Clone analysis can help developers understand code clones’ presence in the software and improve the comprehensibility of the software. Clone maintenance can help developers solve the problems caused by code clones and improve the quality of software and maintainability. Therefore, research on code clone analysis and maintenance has great scientific theoretical significance and practical applicative value for improving the quality of the software and enhancing the comprehensibility and maintainability of the software.

During the process of code clone evolution, code clones and their evolution often conceal some clone characteristics, we called them as “clone evolutionary characteristics”. Among them, researchers are strongly attracted by the changes to code clones in their evolution— been termed as “consistent changes and inconsistent changes”. Clone changes may give rise to clone consistency-defects and additional maintenance costs associated with the changes, thereby reducing software quality and maintainability. Therefore, this thesis studies on the code clone analysis and consistency maintenance based on machine learning methods and code clones’ evolution. Analyzing and extracting the clone evolutionary characteristics of code clones can help the developers understand the code clones. Predicting the consistency-requirement of code clones can solve the issues of code clone consistency maintenance, which also can help developers to maintain code clones.

Aiming to address the difficulty of analyzing and understanding the evolving code clones, an approach for extracting code clone evolutionary characteristics based on clustering method is proposed in this thesis, which can help the developers to analyze and understand the code clones. We firstly detect all code clones with detection tool from

software's repository, then build all the clone genealogies from software to describe the evolution of the code clones. After that, the corresponding attribute sets were extracted from three different perspectives of the code clones and their evolution: clone fragment, clone group and clonal genealogy. Finally, the clustering method is employed for excavating and analyzing the clone evolutionary characteristics form all the code clones and their evolution to help developers understand the code clones. The experimental results show that most of the code clones are stable during the evolution, but there are also a significant number of code clones that occur the changes. What's more, the number of clone codes that have consistent change more than the code clones that have inconsistent change.

Aiming to the problem of additional maintenance cost caused by the clone consistent change in future evolution of a clone creating operation, an approach for predicting clone creating consistency-requirement based on the Bayesian network is proposed in this thesis. We call the clone creation of copy-and-pasted operation as "clone creating instance", and call the consistent change that occurs in the future evolution of such creating instance as "clone creating consistency-requirement". Firstly, we collect all the clone creating instances through detecting all the code clones and building all clone genealogies from software's repository. And then, two different attribute sets are extracted to represent the clone creating instance from two perspectives with the code attribute set and the context attribute set. Finally, the Bayesian network model are trained with the collection of clone creating instances, and applied to predict clone creating consistency-requirement for clone creating instances. The experimental results show that our approach can effectively predict the consistency-requirement for creating instances, which can help the developers to reduce the consistency maintenance cost of the code clone at the creating time.

Aiming to the problem of consistency-defect caused by failure of clone consistent change in future evolution of a clone changing operation, an approach for predicting clone changing consistency-requirement based on Bayesian network is proposed. In this thesis, we call the change operation to clone code as "clone changing instance", and call the consistent change that occurs in the future evolution of such changing instance as "clone changing consistency- requirement". First, through detecting all the code clones and building all the clone genealogies, all the clone changing instances can be collected from software's repository. After that, three different attribute sets are extracted to represent the clone changing instance with three perspectives of the code attribute set, the context attribute set, and the evolutionary attribute set. Lastly, the Bayesian network

model is trained with clone changing instances, and is supplied to predict clone changing consistency-requirement. The experimental results show that the proposed method can reasonable predict the consistency-requirement for changing instances, which can help the developers avoid clone consistency-defects at the clone changing time.

Aiming to the problem of clone consistency-requirement with combination of software development and with consideration of the other machine learning methods, an empirical study on prediction of clone consistency-requirement based on five different machine learning methods is constructed in this thesis. We unify the clone creating instances and changing instances as "clone instances", and unify clone creating and changing consistency-requirement as "clone consistency-requirement". Firstly, we collect all clone instances by detecting code clones and building clone genealogist from software repository, and represent all instance with different attribute sets. And then, we extend and employ five different machine learning methods to the prediction of clone consistency-requirement. Finally, our prediction of clone consistency-requirement can be the combined with the software development, that is supplied by an eclipse plug-in to predict clone consistency-requirement at the software development time. The experimental results show that the proposed method can be applied to different machine learning methods, and the support vector machine method has the nicer prediction effectiveness. Our approach can be embedded into the software development environment, that is combining the prediction of clone consistency-requirement and the software development process, which can help the develops to avoid code clone consistency-defects and clone consistency-maintenance costs.

In summary, this thesis presents a machine-based code clone analysis and consistency maintenance approach in the view of clone evolution with software. It provide new ideas and methods for addressing the analysis and understand of code clone, the maintenance of code clone consistency, the avoidance of code clone consistency-defects, the reduction of code clone consistency maintenance costs, and the improvement of software quality and maintainability.

Keywords: Code clones; clone analysis and maintenance; clone evolutionary characteristics; consistency maintenance requirements; consistent change; machine learning

目 录

Contents

第 1 章 绪论

1.1 课题背景及研究目的和意义

随着计算机软件广泛应用于经济、军事、商业等各个领域中，软件质量问题日益引起了人们的广泛重视。保证软件高质量，并提高软件可理解性和可维护性已经成为软件系统开发和维护工作的一个不可或缺的重要方面。然而，随着应用需求和应用环境的不断变化，现实世界中的软件系统会随着时间而不断演化，导致了软件规模越来越大、逻辑越来越复杂，致使软件的质量、可理解性和可维护性也会随着时间逐渐的下降。其中一个重要的原因是软件系统中存在的大量的克隆代码。在软件开发和维护的过程中，代码复用作为一种常见的软件开发手段可以大大地提高软件开发效率，但是也往往会向软件系统中引入大量的克隆代码。有研究表明软件工程实践中会产生多种类型的克隆代码 [1]，克隆代码在大型软件系统中会占到代码总量的 7-23%[2][3][4]，甚至在有的软件系统中占到 59%[5]。

随着时间的推移和软件系统功能的不断添加，软件系统的规模越来越大，其中软件系统中的克隆代码也会越来越多，越来越多的克隆代码会使得软件变得越来越难以理解，降低了软件的可理解性。于此同时，软件中的克隆代码还会随着系统的演化并可能会发生变化，这也对软件的维护工作增加了新的挑战，降低了软件的可维护性。更进一步，克隆代码的变化也更容易导致软件缺陷，会降低软件的质量。因此，Fowler 将克隆代码视为一种代码坏味（Bad Smell）[6]，认为克隆代码的存在会对软件系统造成不可避免的影响，从而引发了研究人员对克隆代码缺陷 [7][8][9]、克隆代码有害性 [10][11][12] 等相关研究。克隆代码是影响软件质量、可理解性和可维护性的一个重要因素，如何分析和理解系统中既有的克隆代码，并对其进行有效的维护是一个值得研究的问题。目前对克隆代码的分析和维护研究已成为软件工程领域中的一个研究热点问题，同时也是亟待解决的一个问题。

为了解决克隆代码问题，研究人员对其展开了大量且深入的研究，也取得了较多的研究成果。当前对克隆代码的研究可以划分为三个主要的研究内容，即克隆代码检测研究、克隆代码分析研究和克隆代码维护研究。其中，克隆代码检测研究是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。截止到目前为止，研究人员已经提出和开发了许多克隆代码检测的方法和工具 [13][14][15]，可以高效且快速的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，克隆检测结果不是完全令人满意的。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，如克隆代码产生的原因及其对系统产生的不利影响等，从而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，对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，更无法直接帮助提高软件的质量以及可维护性。令人值得注意的是，对克隆代码的分析和维护研究可以更好地弥补这一不足。

因此，本文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。克隆代码分析的主要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最为活跃的一个分支。经研究发现克隆代码是会随着软件演化而演化 [16][17]，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 [18][19][20]。如何深入的分析克隆代码及其演化情况，从而揭示克隆代码所隐含的信息是一个值得研究的问题。通过对克隆代码的演化分析并获取克隆代码的演化特征，对于维护人员理解克隆代码及其演化过程具有积极的意义，并进一步提高软件的可理解性。于此同时，克隆维护研究也是克隆研究的另一项重要活动，旨在帮助维护系统中的克隆代码，帮助解决克隆代码所引发的各种问题。在克隆代码的演化过程中，往往会有相当数量代码被程序开发人员修改而发生变化 [21][22] ，本文称之为一致性变化或者不一致变化。发生一致性变化的克隆代码会对软件系统产生较大的影响：首先确认克隆代码是否需要一致性变化和执行克隆代码的一致性变化会增加软件维护代价；而遗忘克隆代码的一致性变化也会导致克隆缺陷的产生 [7][9]，将会进一步导致软件维护代价的增大。因此，如何避免和预测需要一致性维护的克隆代码是另一个值得研究的问题。通过预测克隆代码的一致性维护需求，可以有效的避免一致性缺陷，也可以降低克隆代码所导致的代价，并进一步帮助提高软件质量和可维护性。

基于以上分析，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，本文研究基于机器学习的克隆代码分析与一致性维护方法，帮助软件开发人员理解和维护软件系统中的克隆代码。具体来说，本文首先结合聚类分析和克隆演化的研究成果，研究并提取了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。然后针对克隆代码一致性变化所导致的缺陷以及额外的维护代价问题，基于贝叶斯网络分别在克隆代码创建之时和克隆代码变化之时预测克隆代码的一致性维护需求帮助维护克隆代码，可以有效地降低克隆代码维护代价和避免克隆

一致性缺陷。最后结合软件开发过程，将克隆一致性需求维护预测扩展到其它的机器学习方法上，并帮助软件开发人员在软件开发时选择合适和预测模型和预测时间。因此，本文方法对于提高软件质量，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。

1.2 国内外研究现状及其分析

研究结果表明系统中通常存在着大量的克隆代码，比其例大约在为 7-23% 左右，有的甚至高达 59%。克隆代码（简称克隆）是根据某种相似性的定义彼此相似的代码片段 [1]。目前，使用最为广泛的一种克隆代码分类方法是按照克隆代码的语法和语义对克隆代码的相似程度进行定义和分类 [23]，将克隆代码划分为如下四种类型：

• Type-1 克隆代码：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 克隆代码：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 克隆代码：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 克隆代码：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）。

其中，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，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。

1.2.1 研究热点与趋势

克隆代码研究一直以来都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，研究人员发表了大量的学术论文，取得了较多的研究成果。为方便研究人员查阅相关研究，截止到 2013 年阿拉巴马大学一直维护着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，并按照研究内容和目标将克隆代码研究划分为以下四个研究方向：

• 克隆检测方向：该方向目录下列出了与克隆代码检测相关的研究内容，旨在

[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，目前已停止维护。

使用不同的方法从系统中检测出克隆代码。

• 克隆分析方向：该方向目录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系统中存在的克隆代码，从而理解克隆代码。

• 克隆维护与管理方向：该方向目录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。

• 综述和工具评估方向：该方向目录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。

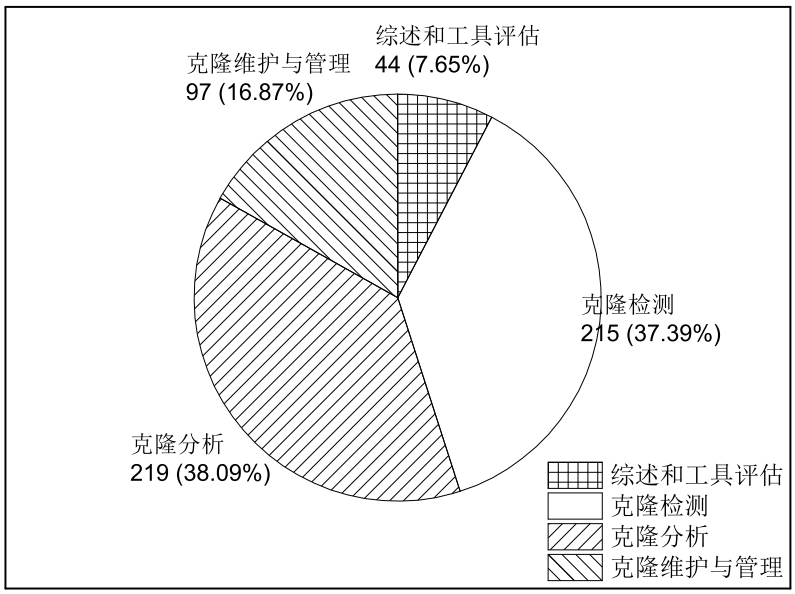


图 1-1 克隆代码研究领域文献分布

Fig. 1-1 Literature distribution of code clone research

根据此文献库中所给出的论文，Roy 按照其分类进行了统计分析，分析了每个研究方向的论文分布情况和研究进展 [24]。但是该文献库仅收录了 2013 年以前的文献，为了分析和反映克隆代码研究的最新进展、研究热点和发展趋势，本文在该文献库的基础上做了进一步的扩展，检索和收集了近几年的克隆代码相关文献，并按该文献库分类方式进行了重新统计和分析。本文统计和收集的文献共分为三类，即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] 。

[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。

克隆代码研究领域的文献统计分析结果如图 1-1 所示。从图中可以看出，目前研究中对克隆检测与分析的研究较为充分，是克隆研究领域的研究热点问题，分别占比 37.39% 和 38.09%；而克隆维护与管理的研究论文占比较少（占 16.87%），说明目前对该方向的研究还方兴未艾。克隆综述和工具评估方面的研究占比更少，仅为 7.65%，说明目前较为实用的工具还不够充分。

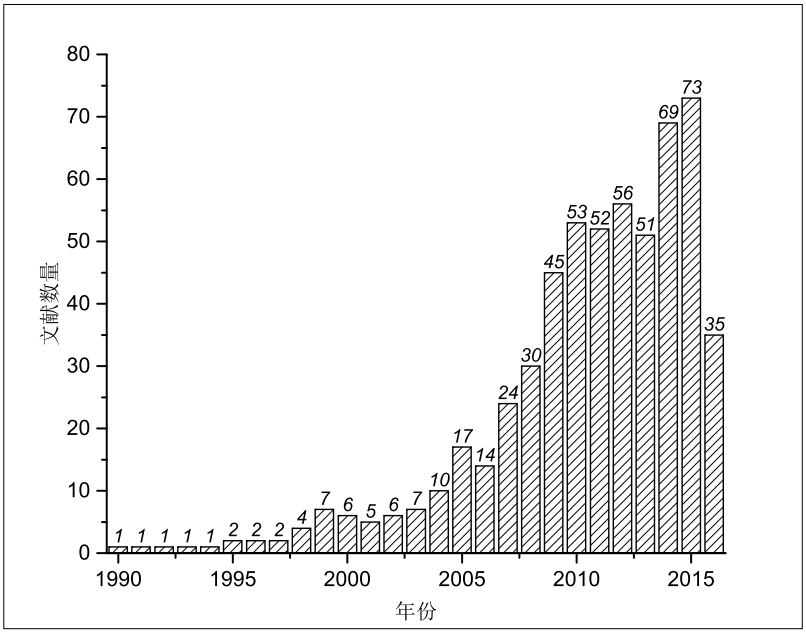


图 1-2 克隆领域每年发表的论文数量

Fig.1-2 Annual number of published papers of code clone research

FigAnnual Number of Published Papers of Code Clone Research

为便于分析克隆代码研究领域的发展趋势，本文又对克隆代码研究领域的文献按照发表年份和研究方向进行了统计。统计分析结果如图 1-2 和 1-3 所示，其中图 1-2 是领域整体上每年发表文献数量的统计情况，图 1-3 是在不同研究方向上每年发表文献数量的统计情况。

从图 1-2 可以看出，克隆研究可以划分为两个阶段：1990-2003 年和 2004-2016 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，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 1-2 类似，从图 1-3 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

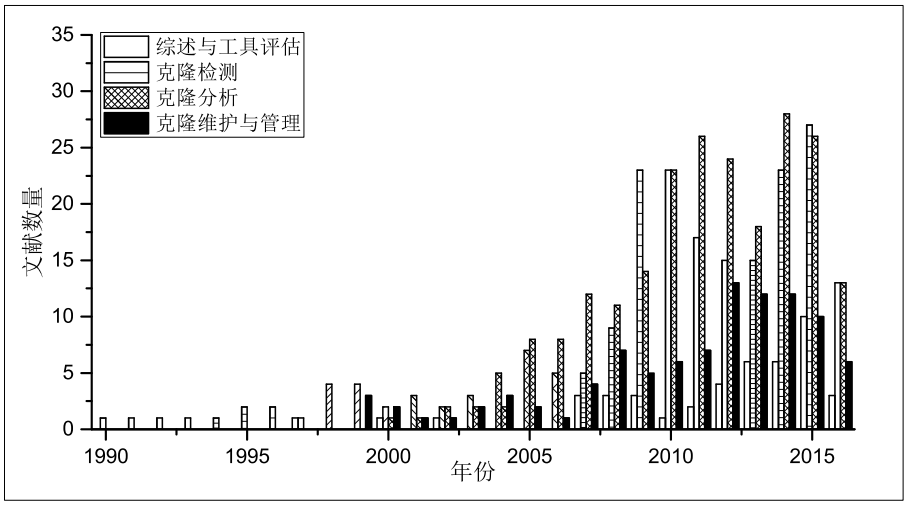


图 1-3 各个克隆研究活动每年发表的文献数量

Fig.1-3 Annual number of published papers of each clone research activity

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，说明克隆检测和分析事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，对克隆维护和管理的研究虽然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，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。

1.2.2 克隆代码检测

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。相对于克隆代码研究的其他技术而言，克隆检测技术相对成熟。

1.2.2.1 克隆检测方法

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,可以将克隆检测划分为基于文本 (Text) 的方法、基于 Token 的方法、基于树 (如 Abstract Syntax Tree，AST) 的方法、基于程序依赖图（Program Dependency Graph, PDG）的方法和基于度量值（Metric）的方法。

基于 Text 的克隆检测方法是通过直接比较源代码文本，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。目前使用较多的基于文本的克隆检测工具主要有 duploc[5]、Simian[？]、DuDe[25]、SDD[26]、NiCad[13] 等。

基于 Token 的克隆检测方法是通过对源代码进行词法分析，获得源代码的 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，所以可以较好地检测 Type-2 克隆代码的检测。但由于缺乏必要的语法和语义分析，使其无法较好地支持 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[2]、CCFinder[14]、CP-Miner[27]、iClone[28] 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），然后通过使用子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，尤其是可以较好地支持 Type-3 克隆的检测。但是由于子树匹配算法的时间复杂度高于前两种方法，因此这类算法的检测速度低于前两种方法。目前使用较多的基于 Tree 的克隆检测工具有 CloneDr[29]、SimScan[？]、Deckard[15]、CloneDigger[30] 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或者编译器前端获得源代码的中间表示。大部分的克隆检测方法和检测工具属于前三种方法中的一种，可以检测系统中的大部分克隆代码。

基于 PDG 的克隆检测方法的主要思路是，将源代码转化成程序依赖图 (包括数据依赖图和控制依赖图)，然后通过寻找同构的子图来检测克隆代码。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 PDG 的克隆检测工具主要有 Duplix[31] 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。该方法的主要优点是检测速度快。但因基于度量值的方法高度依赖于度量值的提取，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，还有人使用结合两种或两种以上检测技术的混合方法来检测克隆代码。例如，Deckard 在生成抽象语法树的基础上提取结构特征向量表示代码，然后使用聚类的方法寻找克隆代码 [15]。CloneMiner 先将源代码表示为 Token 形式，然后在

此基础上通过使用频繁模式挖掘算法寻找相似模式来检测克隆代码 [32]。

1.2.2.2 克隆检测方法及工具评估

不同的克隆检测方法和检测工具各有其优缺点，分别适合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。如何针对具体的应用，选择合适的克隆检测方法和工具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了评估研究，以期为用户选择方法和工具提供指导性的建议 [33][34][35][36]。本文也对目前较为主流的克隆检测工具和方法支持的克隆类型和检测效果等进行了评估，评估结果如表 1-1 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。

表 1-1 主流克隆检测方法与工具评估

Table 1-1 Evaluation for popular clone detection methods and tools

类型	工具或方法名	支持的克隆类型	检测效果
Text	Duploc[5]	1、3	较好 1，一般 3
	Simian[?]	1、2	较好 1，一般 2
	DuDe[25]	1、3	较好 1，一般 3
	SDD[26]	1、3	较好 1，一般 3
	NiCad[13]	1、2、3	较好 1、2、3
Token	Dup[2]	1、2	较好 1、2
	CCFinder[14]	1、2	较好 1、2
	CP-Miner[27]	1、2、3	较好 1、2，一般 3
	iClone[28]	1、2	较好 1，2
Tree	CloneDr[29]	1、2、3	较好 1、3，一般 2
	SimScan[?]	1、2	较好 1、2
	Deckard[15]	1、2、3	较好 1、2，一般 3
	CloneDigger[30]	1、2、3	较好 1、3，一般 2
PDG	Duplix[31]	1、2、3、4	较好 1、2，一般 3，较差 4
	Gabel[37]	1、2、3、4	较好 1、2、3，一般 4
Metric	Kontogiannis[3]	1、2、3、4	较好 1、2，较差 3、4
	Mayrand[38]	1、2、3、4	较好 1、2，较差 3、4

从表 **??** 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。支持 Type-2 克隆检测的原因是在将源程序转换成 Token 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 Type-3 即近似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 [3][38]，缺少相应的检测工具。

1.2.3 克隆代码分析

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和维护克隆代码。目前的克隆分析研究主要集中在克隆演化分析、克隆评价分析上。

1.2.3.1 克隆代码演化

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，识别克隆代码的演化规律，从而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。演化过程分析即模型化克隆代码的演化过程。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 等人提出的，使用时间序列描述克隆代码的演化模型 [39]，但并未引起人们的重视。2005 年，Kim 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 [16]。Roy 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 [17]。Bakota 通过映射不同版本的克隆来分析克隆代码的演化过程，并使用克隆坏味（Clone Smell）帮助分析克隆代码对系统的影响 [40]。Harder 对现有的演化模型进行分析 [41]，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？这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。

表 1-2 列出了目前对克隆演化特征的研究情况。由表中可以看出，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。上述三个特征并不是相互独立的，克隆寿命会受到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。对克隆演化分析的研究大多属于实证研究，往往会较多地依赖于具体被用于实验分析的软件系统，这就导致了不同的研究可能得出不同的结论。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。

表 1-2 克隆演化特征分析

Table1-2 The analysis of clone evolutionary characteristic

文献	克隆模式、特征	结论
[16]	克隆寿命/一致性变化	观察并分析克隆家系寿命与一致性变化规律
[42]	克隆寿命	克隆寿命与克隆修改次数、新增和减少有关
[43]	克隆寿命	克隆代码的寿命比非克隆的寿命更长
[44][45]	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
[46]	克隆稳定性	克隆代码比非克隆代码更稳定
[47][48]	克隆稳定性	克隆代码十分的稳定
[18]	克隆变化/一致性变化	大部分克隆不会变化，一致性变化会更少
[20]	克隆稳定性	克隆会比非克隆更容易发生变化
[49][19]	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
[21]	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
[50]	延后传播	Type-3 更容易发生延后传播并导致缺陷

1.2.3.2 克隆代码评价

克隆评价分析主要是分析克隆代码对系统产生的影响，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。因此，克隆评价研究主要是围绕着克隆代码是否有害而

展开的，只是在不同阶段，人们对克隆代码的态度有所不同，研究的关注点也因此有所不同而已。

在研究的初始阶段，人们大多倾向于认为克隆代码都是有害的，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，人们研究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。

表 1-3 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方法进行了总结。“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。从表中可以看出，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。

1.2.4 克隆代码维护

克隆维护是解决克隆代码问题的直接途径，主动地解决克隆代码可能或已经引发的问题。克隆维护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、克隆复用和克隆管理。

1.2.4.1 克隆代码重构

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变软件的既有设计 [62]。重构应用于克隆维护中，指的是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，同时重构的代价也较大，还需考虑重构安全的问题。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，以提高克隆重构的效率。Lin 等人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 [63]。Mende 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 [64]；Schulze 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 [65]；Choi 等人

表 1-3 克隆评价分析

Table 1-3 The analysis of clone evaluation

	文献	评价	结论
克隆缺陷分析	[7]	消极	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
	[8]	消极	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
	[51]	积极	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
	[52]	积极	克隆代码含有更少的缺陷，Type 3 所含有的缺陷最少
	[53]	中立	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
	[54]	中立	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
克隆维护代价分析	[9]	中立	Type 3 克隆中有 17% 含有缺陷，并且 Type3 克隆更容易发生一致性变化
	[55]	消极	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
	[56]	消极	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
	[57]	中立	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
克隆有害性分析	[11]	积极	克隆代码并不比非克隆代码具有更高的有害风险
	[58][10]	积极	发现最多有 71% 的克隆对系统的可维护性具有积极的影响
	[59]	积极	克隆并不是真正的代码坏味，克隆是无害的
	[12]	中立	预测克隆有害性，其中有害的较少，无害的较多
	[60]	中立	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
	[61]	中立	现有研究不能支撑克隆有害的观点，仍需进一步的研究

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 [66]。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、[67]；Lee 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 [68]；Zibran 提出一种极限编程方法对克隆代码进行重构调度 [69]。另外，Liu 等人提出了一个重构调度方法帮助优化调度过程 [70]。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。此外，Radhika 等人仅考虑重构代价对所有的克隆代码进行重构排序，未考虑克隆代码是否适合重构的问题 [71]，应该结合可重构性分析指导克隆重构。克隆重构最主要的目的是消除系统中的克隆代码。Higo 等人开发了一个工具 ARIES，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 [72]。Krishnan 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的重构，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，取得了较好的效果 [73]。Barbosa 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可以有效地移除软件中的克隆代码 [74]。

在重构完成后，对重构后代码进行分析，还可以得出一些结论以进一步指导克隆重构。Göde 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 [75]。Zibran 等人通过克隆重构研究，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 [76]。Choi 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 [77]。Tairas 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 [78]。

1.2.4.2 克隆代码规避

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进行干预，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。

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，是目前常见的克隆规避方法。Ali 曾提出一个克隆规避的概念模型 [79]，尽管对克隆规避研究具有一定的指导意义，但并未给出该模型的具体实现方法。Wang 等人基于贝叶斯网络在复制粘贴时预测克隆代码的一致性变化，通过判断一致性变化决定是否规避该新产生的克隆代码 [12][80]。Ravikanth 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 [81]。

1.2.4.3 克隆代码复用

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。

Krutz 等人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 [82]。Ishihara 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 [83]。Ohta 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 [84]。但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，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。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[85] 等人使用机器学习模型对克隆代码进行分类，分类标准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 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 [86]。Kintab 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 [87]。

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，由此产生了跨项目的克隆代码和对跨项目代码复用的应用需求。但目前对这种跨项目的克隆代码复用的研究依然较少。Ishihara 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 [88]。Cheng 等人研究了跨项目的克隆代码检测技术 [89]。Tairas 等人将信息检索技术与

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 [90]。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。

1.2.4.4 克隆代码管理

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 [91]，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时，才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克隆代码维护难题的最有效的方式。早在 2012 年的面向工业界的克隆管理会议[3]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 [92]。

克隆代码管理的主要目的是降低克隆代码维护的代价。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。克隆管理工具是研究开发有效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及其变化。由于复制粘贴操作是导致克隆产生的主要原因，因此对克隆跟踪的研究主要是通过监测程序员的复制粘贴操作来实现的。例如，许多克隆跟踪工具 CLONEBOARD[93]、CnP[94]、CPC[95]、CReN[96]、CSeR[97] 等都是在软件集成开发环境中跟踪克隆代码的产生，但并非都能跟踪克隆代码的变化。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 使用克隆区域描述符描述生成克隆代码的上下文信息，然后根据这些上下文信息实时跟踪克隆代码的变化 [98][99]。该方法仅通过实验验证可以跟踪克隆代码的变化，但是并没有实现可用的插件集成到集成开发环境中，以实现对克隆代码的边开发、边管理。

跟踪克隆代码的目的是为了对克隆代码进行维护和管理。为了实现边开发、边管理克隆代码，还要对克隆代码的变化进行维护和管理。Yamanaka 等人 [100] 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。Cheng 等人 [101] 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。该方法的前提是已知克隆变化需要一致性维护，但没有给出克隆变化是否需要一致性维护的判定方法。因此，

[3]Software Clone Management Towards Industrial Application

Zhang 等人 [102] 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 [103]。此外，Murakami 等人在克隆代码未发生变化时，预测克隆代码的下一次变化 [104]。该方法通过分析历史版本中的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提出的克隆管理方法，却没有提供可以实际应用的克隆管理工具。鉴于此，Zhang 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 [105]。Nguyen 实现了一种可用于管理克隆代码的 eclipse 插件 JSync[106]。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与一致性维护管理。这里的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。这里的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[107]。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。该插件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。

1.2.5 当前研究中存在的问题分析

克隆代码研究已经成为了软件工程领域的一个热点问题，目前的克隆研究已经取得了相当丰富的研究成果，如克隆检测、克隆分析和克隆维护研究等，但目前的研究中仍然存在一些问题。

（1）克隆检测研究可以帮助程序开发人员快速的获得系统中的克隆代码，是目前研究的较为深入且广泛的一个研究活动。已经开发和提出了相当规模的克隆检测方法和工具，使用其可以快速高效地检测系统中存在的克隆代码。但是，克隆代码检测仅仅是克隆研究的初始阶段，即不能帮助软件开发人员理解克隆代码，又不能解决克隆代码对软件产生的不利影响。因此，克隆代码分析和维护研究将是解决克隆代码问题的关键内容。

（2）克隆代码分析可以辅助开发人员理解克隆代码，也是目前克隆代码研究中

最为活跃的一个分支。在克隆分析的研究中，克隆代码演化研究是最为重要的研究内容，克隆评价研究往往也是在克隆代码演化的基础上进行，尤其是体现在演化过程的克隆代码变化对软件的影响上。但是，克隆演化分析研究中仍然不能令人满意。首先最重要的是，在目前的克隆代码的演化研究中，研究人员的研究往往集中到某一个具体的克隆代码的演化特征上，缺少从宏观上的对克隆代码演化特征的分析。例如，研究人员分别研究了克隆代码的克隆寿命、稳定性等某一个具体的问题。其次，克隆演化研究也往往带有较强的主观性，研究人员往往是通过分析既有的系统验证固有的结论，甚至出现了完全不同的研究结论。例如在对克隆代码稳定性的研究中，有研究者认为克隆代码是稳定的，也有研究者认为非克隆代码是稳定的。因此，如何从大量的克隆代码及其演化过程中全面且客观地识别克隆代码的演化特征是一个值得研究的问题。克隆代码的演化特征对于帮助程序开发人员理解克隆代码及其演化过程具有积极的意义，可以进一步提高软件的可理解性。

（3）克隆维护研究旨帮助程序开发人员解决或者避免克隆代码对系统产生的不利影响的问题。在克隆维护研究中研究人员进行了大量的研究，包括对克隆代码的重构、复用和管理等等。然而，目前克隆维护研也不能令人满意。上述克隆维护研究没有聚焦到导致克隆代码难以维护的根本原因上。克隆代码难于维护的根本原因在于克隆代码在其演化过程的中的变化问题，尤其是克隆代码的一致性变化和不一致变化，及其由此所引发的克隆一致性缺陷和额外的维护代价问题。如何避免、解决克隆代码的一致性变化问题是克隆维护研究中的一个值得研究的问题。通过预测克隆代码的一致性维护需求，可以有效的避免一致性缺陷，也可以降低克隆代码所导致的代价，并进一步帮助提高软件质量和可维护性。

1.3 研究内容与论文结构

1.3.1 研究内容

针对软件中所存在的大量的克隆代码、以及克隆在其演化过程容易被程序开发人员修改而导致克隆代码的一致性维护问题，本文结合程序分析、克隆演化和度量提取，研究基于机器学习的克隆代码演化分析与一致性维护方法，可以有效的帮助程序开发人员理解克隆代码，并可以降低克隆代码的维护代价和避免克隆一致性缺陷，以达到提高软件质量、可理解性和可维护性的目的。本文的主要研究内容可以简述如下：

（1）针对演化中的克隆代码难于理解和分析的问题，研究基于聚类的克隆代码

演化特征分析方法，使用克隆演化特征帮助程序开发人员程序分析和理解克隆代码。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，获取相应的克隆代码演化特征，帮助理解和维护克隆代码。

（2）针对创建的克隆代码在其演化过程中的一致性变化往往会导致额外的维护代价问题，研究基于贝叶斯网络的克隆代码创建时一致性维护需求预测方法。将复制粘贴创建的克隆代码称为克隆创建实例，将其在未来演化过程中所发生的一致性变化称为克隆创建时一致性维护需求。通过提取代码属性和上下文属性表示克隆创建实例，并使用贝叶斯网络预测克隆代码创建时的一致性维护需求，可以帮助程序开发人员降低克隆代码的一致性维护代价。

（3）针对克隆代码变化时遗忘克隆代码一致性变化会导致克隆一致性缺陷问题，研究基于贝叶斯网络的克隆代码变化时一致性维护需求预测方法。将克隆代码发生变化的克隆代码称为克隆变化实例，并将其未来演化过程中所发生的一致性变化称为克隆变化时一致性维护需求。通过提取代码属性、上下文属性和演化属性用于表示克隆变化实例，并使用贝叶斯网络预测克隆代码变化时一致性维护需求，可以帮助程序开发人员避免克隆代码一致性缺陷。

（4）针对克隆代码一致性维护需求预测如何结合软件开发过程和其它机器学习方法的问题，研究基于不同机器学习方法的克隆代码一致性维护需求实证研究方法。将克隆代码创建和变化实例统称为克隆实例，并将其一致性维护需求统称为克隆代码一致性维护需求。将克隆代码的一致性维护预测扩展到其它五种不同的机器学习方法中，并与软件开发过程相结合实现边开发边预测克隆代码的一致性维护需求，帮助程序开发人员在软件开发过程中选择合适的机器学习模型和预测时间，从而实现边开发、边分析、边维护克隆代码，帮助提高软件的质量和可维护性。

1.3.2 论文结构

基于机器学习的克隆代码分析与一致性维护方法，综合考虑程序分析方法、克隆代码演化、机器学习方法和属性提取方法实现对克隆代码的分析和维护。为了进一步说明本文的研究内容以及各个部分之间的关系，论文的主要研究内容及其关系示意图如图 1-4 所示。本文主要研究四个内容，首先研究克隆代码的演化特征分析方法，获取克隆代码演化特征用于指导后续的克隆代码一致性维护研究。然后，

针对克隆代码的一致性维护问题，分别在克隆代码创建时和变化时基于贝叶斯网络预测克隆代码的一致性维护需求，从而降低额外的克隆维护代价和避免克隆一致性缺陷价。最后结合软件开过程，将克隆代码一致性维护需求预测问题扩展到其它的机器学习方法中，帮助程序开发人员有效地维护克隆代码一致性，从而帮助提高软件质量和可维护性。

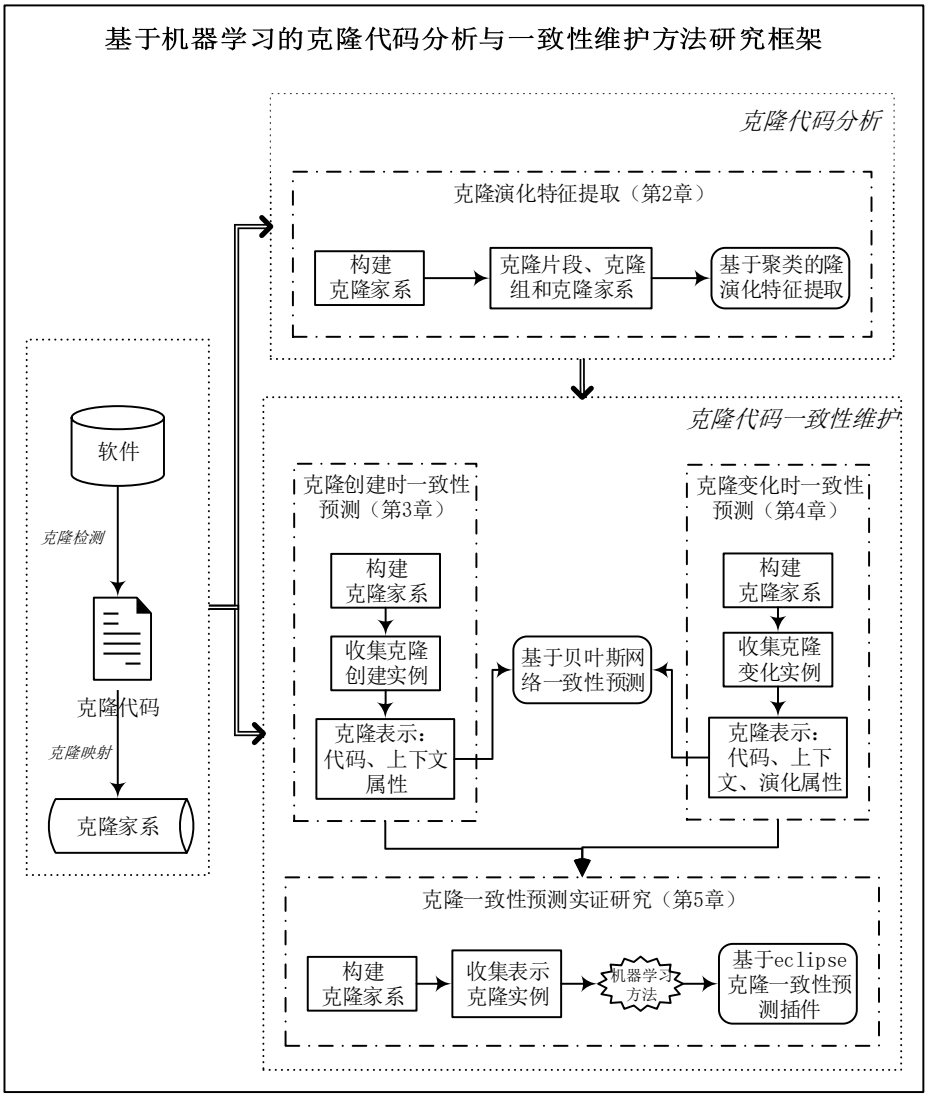


图 1-4 论文主要研究内容及其关系示意图

Fig.1-4 Main research contents and their relationship in the thesis

各章节研究内容安排如下：

第 2 章研究基于聚类的克隆代码演化特征分析方法，使用克隆演化特征帮助

程序开发人员程序分析和理解克隆代码。首先，使用克隆检测工具检测系统中的克隆代码，并构建系统所有克隆代码的克隆家系用于描述克隆代码的演化过程。然后，从克隆片段、克隆组和克隆家系三个不同角度描述克隆代码及其演化过程并提取相应的度量值。最后，使用聚类分析方法聚类克隆代码并挖掘克隆代码演化特征，帮助开发人员理解克隆代码及其演化过程。

第 3 章研究基于贝叶斯网络的克隆代码创建时一致性维护需求预测方法，定义了克隆代码创建实例及其一致性维护需求。首先，通过检测系统的克隆代码并构建克隆家系收集系统中的克隆创建实例。然后，提取代码属性和上下文属性两组不同的度量值表示克隆创建实例的被复制和被粘贴的克隆代码。最后，使用贝叶斯网络预测克隆代码创建时一致性维护需求。

第 4 章研究基于贝叶斯网络的克隆代码变化时一致性维护需求预测方法，定义了克隆代码变化实例及其一致性维护需求。首先，通过检测系统的克隆代码并构建系统克隆家系收集系统中的克隆变化实例。然后，提取代码属性、上下文属性和演化属性三组不同的度量值用于表示克隆变化实例。最后，使用贝叶斯网络训练预测克隆代码变化时一致性维护需求。

第 5 章进行克隆代码一致性维护需求实证研究，统一了克隆代码创建和变化实例为克隆实例及其相应的克隆代码一致性维护需求。首先，检测系统的克隆代码并构建克隆家系收集所有的克隆代码实例，并使用不同的属性组表示克隆实例（与第 3 章和第 4 章相同）。然后，将克隆代码的一致性维护预测扩展到其它五种不同的机器学习方法中。最后，将克隆代码一致性维护需求预测与软件开发过程相结合，实现了一个 eclipse 插件可以实现边开发边预测克隆代码的一致性维护需求。

第 2 章 基于聚类的克隆代码演化特征分析方法

2.1 引言

复用软件中的既有代码已成为了一种常见的软件开发手段，可以帮助程序开发人员节约开发时间和提高开发效率。然而，复用既有代码会向软件系统中引入大量的克隆代码，即彼此相似的代码片段。在软件随着时间进行演化的过程中，软件系统中的克隆代码不是静止不变的，也会随着软件系统进行演化，研究人员将克隆代码的这一现象称为克隆代码演化。在大量克隆代码及其演化过程中，必然存在着一些隐含的信息可以揭示克隆演化规律，本文将之称为克隆代码演化特征。克隆演化特征不仅可以帮助软件开发人员理解系统中存在的克隆代码，还可以向软件开发人员提供一些如何维护克隆代码的建议。但遗憾的是，当前研究中对克隆演化特征的研究不够充分，缺乏客观且全面的克隆演化特征研究。因此，如何分析并获取克隆代码演化特征是一个值得研究的问题。

为了帮助程序开发人员获得克隆代码演化特征，本章结合机器学习中的聚类分析方法，在提取克隆代码相应度量值表示不同克隆实体的基础上揭示克隆代码演化特征，可以帮助理解和维护克隆代码。首先，为了描述克隆代码的演化过程，本文检测软件系统所有版本的克隆代码，并在此基础上通过映射相邻版本的克隆代码构建软件系统的克隆家系，并扩展和定义了 7 种不同的克隆演化模式描述克隆代码在演化中的变化情况。然后，从三个不同的角度描述克隆代码及其演化过程，即使用克隆片段、克隆组和克隆家系三种克隆实体。为了使用机器学习方法分析克隆代码及其演化过程，提取了不同度量值分别表示不同的克隆实体。最后，根据所提取的度量值生成相应的聚类向量，并使用聚类分析挖掘和分析克隆代码的演化特征。在开源系统 *ArgoUML* 和 *jEdit* 上进行了实证研究。实验结果表明演化过程中的大部分克隆代码是稳定的，但也存在一定数量的发生变化的克隆代码；在发生变化的克隆代码中，发生一致性变化的克隆代码要多于不一致变化的克隆代码。

2.2 克隆演化及其演化特征

本节介绍与克隆代码演化相关的术语与模型，并给出克隆代码演化特征的定义。

2.2.1 克隆代码演化

软件工程实践会产生大量的克隆代码，软件中存在的大量克隆代码不仅使得系统变得更加臃肿，也使得系统越来越难以理解。为收集和检测系统中存在的克隆代码，在过去的 20 年中，研究人员提出了许多种克隆代码检测方法，并开发了大量的克隆检测工具，例如 NiCad[13]、CCFinder[14] 等（有兴趣的读者可以参考本文绪论克隆检测部分 1.2.2）。目前大多数的克隆检测工具仅可以检测单版本系统中克隆代码，即从系统某一版本的源代码中检测得到克隆代码后，向程序开发人员报告检测结果。克隆检测结果往往以克隆片段和克隆组的形式进行组织，克隆片段是具体克隆代码，克隆组是彼此相似的克隆组的集合。克隆片段和克隆组可以描述如下：

定义 2.1 (克隆片段) 克隆片段 (*Clone Fragment, CF*) 是一段代码片段，包括若干行连续的代码。克隆片段与一些其它的代码片段彼此相似，并将它们称之为彼此相似的克隆片段。

定义 2.2 (克隆组) 克隆组 (*Clone Group, CG*) 是彼此相似的克隆片段的集合，包含若干个彼此相似的克隆片段。一个克隆组揭示了克隆组内的克隆片段之间的克隆关系。

如绪论所述，克隆代码在系统中不是静止不变的，会随着软件系统的演化同时进行演化。克隆演化过程最早是 2001 年由 Antoniol 等人提出，使用时间序列描述克隆代码的演化模型 [39]，但并未引起人们的重视。2005 年，Kim 提出了克隆家系模型用于描述克隆代码的演化过程，是迄今为止最好的演化模型 [16]。本文也使用 Kim 提出的演化模型描述克隆代码演化过程。克隆代码在演化过程中可能会发生变化，为了描述这种克隆组的变化情况，同时定义克隆演化模式描述克隆代码在演化过程中的变化情况。克隆家系及其演化模式可以描述如下：

定义 2.3 (克隆家系) 克隆家系 (*Clone Genealogy, CGE*) 是一个有向无环图，描述了一个克隆组（CG）随系统进行演化的情况。CGE 图中的节点表示系统某一个版本 (V_i) 中的该克隆组（CG_i）。CGE 图中的边表示该克隆组 (CG) 在相邻的两个版本中 (V_{i-1}, V_i) 的演化关系 (CG_{i-1}, CG_i)，即 CG 由上一版本 V_{i-1} 的 CG_{i-1} 演化至下一版本 V_{i-1} 中的 CG_i。

定义 2.4 (克隆演化模式) 克隆演化模式 (*Clone Evolution Pattern, CEP*) 是某一克隆组在演化中的相邻两个版本的演化情况，用于描述该克隆组的变化情况，根据不同的变化具有不同的演化模式。

为了更为形象的描述克隆家系及其演化模式，本文给出一个克隆家系的示意图，如图 2-1所示。从图中可以看出，克隆家系（CGE）是克隆组（CG）演化的有向无环图，图的节点表示克隆组，图中边表示克隆组的演化关系和演化模式。图 2-1给出一个克隆组在五个版本 (V_i, V_{i+4}) 中的演化过程。

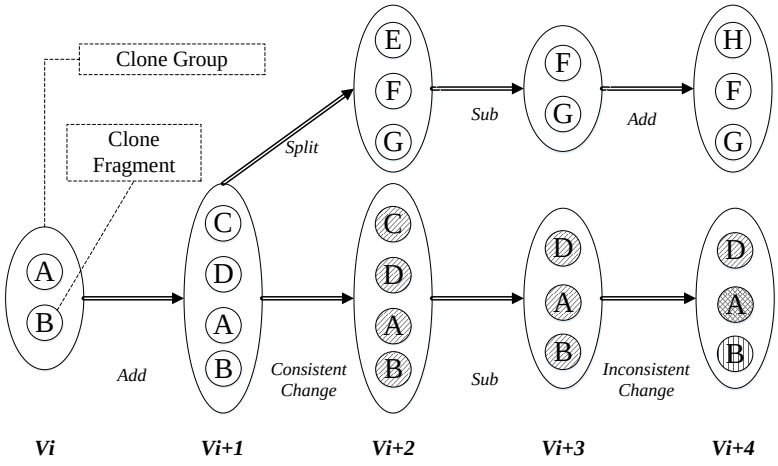


图 2-1 克隆家系示意图

Fig.2-1 An Example for Clone Genealogy

如图2-1中所示，在克隆组的演化过程中，两个相邻版本间的克隆代码可能会被程序开发人员修改而发生变化，因而也会导致克隆组也会发生变化。在克隆组演化过程中，研究人员使用克隆演化模式描述相邻版本之间的克隆组的变化情况。本文给出 7 种不同的克隆演化模式，如下所示：

分裂模式（Split Pattern）：分裂模式表示连续的在两个连续版本的演化中克隆组内克隆片段发生剧烈变化，导致该克隆组分裂成为两个不同的克隆组。

• 静态模式（Static Pattern）：静态模式表示在两个连续版本的演化中该克隆组是静止的，未发生任何变化，即克隆组内的克隆片段数量和内容均未发生变化。

• 相同模式（Same Pattern）：相同模式表示在两个连续版本的演化中该克隆组内克隆片段数量无变化，但克隆片段本身可能发生变化。

• 增加模式（Add Pattern）：增加模式表示在两个连续版本的演化中该克隆组内的克隆片段数量增加。

• 减少模式（Subtract Pattern）：减少模式表示在两个连续版本的演化中该克隆组内的克隆片段数量减少。

• 一致性变化模式（Consistent Change Pattern）：一致性变化模式表示在两个连续版本的演化中该克隆组内的克隆片段发生了一致地变化，并且发生变化的克隆片段仍然存在同一克隆组内。

• 不一致性变化模式（Inconsistent Change Pattern）：不一致性变化模式表示在两个连续版本的演化中该克隆组内的克隆片段发生了不一致地变化，并且发生变化的克隆片段仍然存在同一克隆组内。

• 分裂模式（Split Pattern）：分裂模式表示连续的在两个连续版本的演化中克隆组内克隆片段发生剧烈变化，导致该克隆组分裂成为两个不同的克隆组。

回归到图 2-1中，可以看出在五个版本内该克隆组的演化模式分布情况。从版本V_i 到V_i+1，克隆组新增加了两个克隆片段，因此与之其关联的克隆模式是 Add。从 V_i+1 到 V_i+2 中，克隆组首先分裂为两组，因此其演化模式为 Split，同时下方的克隆组发生了一致地变化，演化模式为 Consistent Change。从版本V_i+2 到V_i+3 中，演化模式一个是 Subtract，另一个是 Inconsistent Change。从版本V_i+2 到V_i+3，克隆模式分别是 Add 和 Subtract。

2.2.2 克隆演化特征

克隆家系的提出引发了人们对于克隆代码演化以及演化模式的研究热潮，进而也导致了大量的分析克隆演化特征的研究。克隆演化特征可描述如下：

定义 2.5 (克隆演化特征) 克隆演化特征指的是克隆代码在演化过程中表现出来的演化特征以及对软件所产生的影响。克隆演化特征不仅可以帮助软件开发人员理解系统中存在的克隆代码，还可以向软件开发人员提供一些如何维护克隆代码的建议。

究竟哪些演化特征能真实准确地反映克隆代码的规律？这是克隆演化特征分析的难点问题。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（具体如表 1-2所示）。

克隆寿命是指克隆代码在系统中的存在时间，即生存期。Kim 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 [16]；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 [42]。Krinke 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 [43]。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 [44]。因此，克隆代码会长时间的存在于系统中，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 [45]。

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内是发生变

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 [46][47][48]，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。例如 Gode 研究发现大部分克隆是稳定的，不会发生变化 [18]。而 Rahman 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 [20]。Mondal 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 [49][19]。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，如标识符重命名不一致性缺陷等，因此一致性变化也是克隆演化分析研究中需要关注的特征。Gode 的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 [18]。Krinke 的研究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 [21]。Mondal 等人的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 的克隆中出现的更为频繁，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 [50]。

克隆演化特征之间并不是相互独立的，例如克隆寿命会受到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。但是目前的研究中并没有揭示这些关系。更为重要的是, 在目前的克隆代码的演化研究中，研究人员的研究往往集中到某一个具体的克隆代码的演化特征上，缺少从宏观上的对克隆代码演化特征的分析。其次，克隆演化研究也往往带有较强的主观性，研究人员往往是通过分析既有的系统验证固有的结论，甚至出现了完全不同的研究结论。因此，如何从大量的克隆代码及其演化过程中全面且客观地识别克隆代码的演化特征是一个值得研究的问题。克隆代码的演化特征对于帮助程序开发人员理解克隆代码及其演化过程具有积极的意义，可以进一步提高软件的可理解性。

2.3 基于聚类的克隆代码演化特征分析框架

为分析克隆代码及其演化过程，本文基于机器学习方法提出了一种探索和分析克隆代码演化特征的方法。克隆代码作为具体的代码片段，直接分析其演化过

程和特征极为困难。因此，本文将克隆代码及其演化过程抽象成为一些特征向量，并借助机器学习中的聚类方法挖掘克隆代码及其演化过程之间的信息。为了更为具体的抽表示克隆代码及其演化过程，本文用三种不同的克隆实体描述克隆代码及其演化过程，即克隆片段、克隆组和克隆家系实体。在基础上提取相应的度量值表示克隆实体的有意义和有价值信息。最后使用机器学习中的聚类方法分析克隆代码，进而挖掘克隆代码的演化特征。

本文所提出的基于聚类的克隆代码演化特征分析框架如图 2-2所示。从图中可以看出，本文方法可以划分为三个阶段，分别是预处理阶段、克隆表示阶段和演化特征挖掘阶段。在预处理阶段，首先检测系统所有版本中的克隆代码，并通过映射连续软件版本之间的克隆代码片以及克隆组来构建系统所有的克隆家系。使用克隆家系可以细致的描述克隆代码的演化过程，同时也可以快速有效的识别克隆代码的演化模式。在克隆表示阶段阶段，使用三种不同的克隆实体表示克隆代码及其演化过程，并分别提取与之相应的度量值描述不同的克隆实体（克隆片段、克隆组和克隆家系）。所提取的度量值包含了有价值的与克隆代码演化和变化情况的信息。最后，在演化特征挖掘阶段，使用机器学习方法中的聚类方法来聚类克隆实体向量，并根据聚类结果挖掘克隆代码的克隆演化特征。

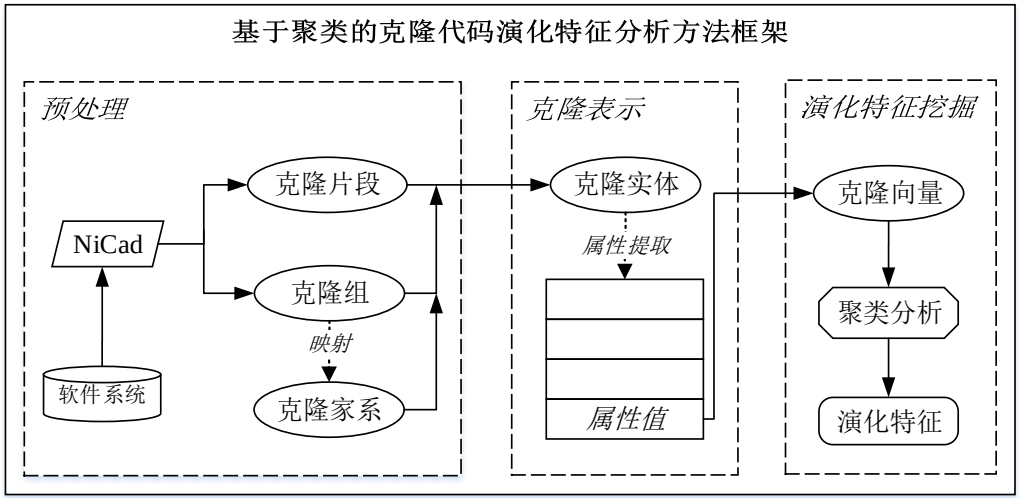


图 2-2 基于聚类的克隆演化特征提取方法框架

Fig.2-2 The framework for clone characteristic analysis based on clustering

值得注意的是，本文方法将克隆代码及其演化过程当做一种数据，然后借助机器学习领域中的聚类分析方法挖掘隐含的信息。在使用聚类分析的时候，由于克隆代码是具体的代码片段，无法直接对其聚类。因此，使用克隆聚类向量表示相应的克隆实体，而克隆聚类向量则是根据克隆实体相应的度量值生成，克隆实体的度量值用于表示克隆代码及其演化过程。本文从三种不同克隆实体表示克隆代码，

即克隆片段、克隆组和克隆家系。克隆片段实体是微观角度，从克隆代码自身角度出发，将从克隆代码片段本身是否被修改的角度分析克隆代码的演化特征，所提取的度量值重点关注克隆代码是否被修改。克隆组实体是代码区域角度，从克隆组的角度分析克隆代码演化模式与演化时间的关系，将揭示克隆组在随着软件演化时所表现出来的特征。克隆家系实体是宏观角度，描述了一个系统中所有的克隆代码（即全部克隆家系）的演化过程以及演化特征。

本文所所采用的聚类分析方法为 *X-Means*[108] 聚类，其原因在于：克隆实体所应聚类的数量具有不确定性，难以确定具体的聚类数量。如果采用人为给定的方式给出聚类的数量，则可能会引入不客观因素，对影响克隆代码演化特征的分析工作。因此，本文采用 *X-Means* 聚类方法，不需要人为的指定聚类数量， X-Means 方法会自动地给出最佳的聚类数量。

2.4 预处理阶段

在预处理阶段，将使用克隆检测工具所检测的连续版本软件中的所有克隆代码，并通过映射相邻版本的克隆代码构建系统的克隆家系。首先从开源库中下载连续版本软件的所有源代码，然后使用克隆检测工具 NiCad 检测克隆代码，最后通过映射克隆代码构建克隆家系和识别克隆演化模式。

2.4.1 克隆检测与结果描述

使用克隆检测工具 NiCad 分别检测系统所有版本中的克隆代码，并使用克隆区域描述符（Clone Region Description, CRD）描述克隆代码位置信息。

为了检测系统中的克隆代码，本文使用 NiCad[13] 检测系统中的克隆代码。NiCad 是基于文本的克隆检测工具，在工具中集成了程序转换、代码规范化和语法分析技术 [109][110]，能够以较高的准确度和召回率系统中的 Type-1、Type-2 和 Type-3 克隆代码[1]。NiCad 可以从两个不同粒度检测系统中的克隆代码：函数粒度和块粒度。因为块粒度具有更为通用的检测效果（函数克隆也是是块克隆代码），本文使用块粒度检测系克隆代码。根据 NiCad 的默认配置，其将克隆代码之间的相似度阈值设置为 70%，将相似度高于此阈值的代码片段报告为克隆代码。

NiCad 将检测到的克隆代码保存于 XML 文件中，并使用 “Filename + Start/End Line No” 标记克隆克隆代码，并将彼此相似的克隆片段保存在同一克隆组内。由于 NiCad 仅仅使用代码行表示克隆代码，不仅无法描述克隆代码的语法和语义信

[1]NiCad 可以从此网站获取：http://www.txl.ca/nicaddownload.html。

息，也不利于映射不同版本之间的克隆代码。因此，为映射相邻版本之间的克隆代码，本文使用改进的克隆区域描述符表示克隆代码，并在此基础上实现克隆代码的映射和克隆家系的构建。克隆区域描述符最早由 Duala-Ekoko 等人提出，不仅可以反映出克隆代码本身的信息，还可以用于跟踪演化过程中的克隆代码 [99]。为了进一步的帮助映射相邻版本之间的克隆代码，慈蒙等人改进了克隆区域描述符，并提出了基于克隆区域描述符的克隆群映射算法 [111][112]。为了映射两个相邻版本中的克隆代码和克隆组，新添加相对位置覆盖率和文本相似度两个额外的表示单元。使用相对位置覆盖率可以帮助定位源代码中的克隆片段，并计算版本 i 和版本 $i+1$ 中克隆片段位置之间的重叠率。文本相似度可以用于比较映射的代码片段的相似性度。具体方法可以参加慈蒙等人的论文 [112]。

2.4.2 克隆家系构建与克隆模式识别

为了映射连续版本的克隆代码，使用基于克隆区域描述符的映射算法，生成所有相邻版本的映射结果，并根据映射结果构建克隆家系和识别克隆演化模式。

构建克隆家系的关键步骤在于映射所有相邻版本间的克隆片段，而相邻版本间的克隆群映射关系则可以根据克隆群内的克隆片段的映射关系进行确定。所采用的映射算法称为基于 CRD 的克隆代码映射算法 [111][112]。假设给定一个软件系统的两个相邻版本V_i 和V_{i+1}，并分别检测两个版本之间的克隆代码，并使用 CRD 描述所检测到的克隆代码。为了映射两个版本间的克隆代码，将 V_i 中的每个克隆片段与下一版本V_{i+1} 中的每个克隆片段进行比较，寻找与之映射的克隆片段，并生成克隆片段的映射结果。于此同时，根据克隆片段的映射结果，也可以实现相邻版本中所有克隆组的映射。更进一步，将所有相邻版本之间的克隆代码进行映射，并根据映射结果生成系统中的所有的克隆家系。

克隆演化模式可以用于描述克隆组在演化过程中的变化情况，对于揭示克隆演化特征具有重要的意义。克隆演化模式识别，可以通过对比映射的两个相邻版本之间的克隆组进行。假设克隆组 CG 是存在于相邻的两个软件版本(V_i, V_{i+1}) 中，且克隆组 CG 的映射关系可以使用(CG_i, CG_{i+1}) 描述，其中CG_i 表示前一版本中的克隆组，CG_{i+1} 表示后一版本中的克隆组。通过观察从CG_i 到CG_{i+1} 的克隆组变化情况，便可以识别该克隆组CG 的克隆演化模式。

2.5 克隆表示阶段

为挖掘克隆代码的演化特征，本文将从三个不同角度分析克隆代码及其演化情况，即克隆片段、克隆组和克隆家系，并将之称为克隆代码实体（克隆实体）。正如在机器学习中通常所做的那样，本文还提取不同的度量值表示不同的克隆实体。对克隆片段实体，本文重点关注克隆代码的变化情况。对克隆家系实体，本文重点关注克隆演化模式的情况。对于克隆家系实体，本文重点关注系统所有的克隆演化模式的情况。

2.5.1 克隆片段度量

克隆片段度量给出了克隆代码本身的一些特征。在克隆片段的生命期间，克隆片段可能被程序人员开发人员修改（特别是在软件维护期间），即克隆片段在演化过程中可能会发生变化，甚至可能发生不止一次的变化。

因此，本文将克隆代码变化次数（截止到系统当前版本 V_i）和是否发生了变化（从上一版本演化到此版本时）视为克隆片段的度量，即在某一软件版本 V_i 中的克隆代码片段CF_i 度量如下所示：

• 克隆寿命（Clone Life）：截止到当前版本 V_i，克隆片段 CF 所经历的所有的版本数量称之为克隆寿命。

• 是否发生变化（Ischanged）：从上一版本 V_{i-1} 演化到此版本 V_i 时，克隆片段 CF 是否发生了变化，若发生变化则取值为 1，否则取值为 0。

• 变化次数（Change Times）：截止到当前版本 V_i，克隆片段 CF 在其演化过程中所发生的变化次数。

2.5.2 克隆组度量

克隆组提供了克隆代码的一些区域性特征。本文所使用的克隆演化模式均是针对克隆组而言，因此也需要从克隆组的角度分析克隆代码的演化情况。对于某一个克隆组，最先提取的度量是克隆寿命。克隆寿命不但揭示了其在系统中存在的时间长短，也与克隆演化模式息息相关。在克隆组进行演化的过程中，克隆模式更是从演化的角度揭示了克隆组的变化情况。因此，本文将版本 V_i 中克隆组CG_i 的寿命和克隆演化模式提取为度量值，如下所示：

• 克隆寿命（Group Life）：截止到当前版本 V_i，克隆组 CG 所经历的所有的版本数量称之为克隆寿命。

• 静态（Static）：克隆组 CG 从上一版本 V_{i_1} 演化到 V_i 时，是否发生了静态模式（Static Pattern）。

• 相同（Same）：克隆组 CG 从上一版本 V_{i_1} 演化到 V_i 时，是否发生了相同模式（Same Pattern）。

• 增加（Add）：克隆组 CG 从上一版本 V_{i_1} 演化到 V_i 时，是否发生了增加模式（Add Pattern）。

• 减少（Subtract）：克隆组 CG 从上一版本 V_{i_1} 演化到 V_i 时，是否发生了减少模式（Subtract Pattern）。

• 一致性变化（Consistent Change）：克隆组 CG 从上一版本 V_{i_1} 演化到 V_i 时，是否发生了一致性变化模式（Consistent Change Pattern）。

• 不一致变化（Inconsistent Change）：克隆组 CG 从上一版本 V_{i_1} 演化到 V_i 时，是否发生了不一致变化模式（Inconsistent Change Pattern）。

• 分裂（Split）：克隆组 CG 从上一版本 V_{i_1} 演化到 V_i 时，是否发生了分裂模式（Split Pattern）。

2.5.3 克隆家系度量

克隆家系提供了一个软件中克隆代码演化的全局视角，从而可以帮助捕获软件系统整个生命期间的克隆代码的演化特征。如前文所述，一个克隆组CG 在所有的版本中的演化过程即是一个克隆家系CGE，因此一个软件系统中将会有很多个克隆家系。对于每一个克隆家系CGE，本文提取克隆寿命和克隆演化模式数量作为克隆家系CGE 的度量。克隆寿命是克隆家系的一个极为重要的指标，描述了克隆代码在系统中存在的时间。同时，克隆家系还描述了一个克隆组在整个生命期间的全部演化过程（所经历的克隆演化模式数量）。演化模式数量揭示了克隆组的整个变化历史。因此某一克隆家系CGE 的属性如下所示：

• 克隆家系寿命（Genealogy Life）：克隆家系CGE 在其整个演化过程中所经历的软件版本的数量。

• 静态模式数量（Static Number）：克隆家系CGE 在其整个演化过程中所经历的静态模式的数量。

• 相同模式数量（Same Number）：克隆家系CGE 在其整个演化过程中所经历的相同模式的数量。

• 增加模式数量（Add Number）：克隆家系CGE 在其整个演化过程中所经历的增加模式的数量。

• 减少模式数量（Subtract Number）：克隆家系CGE在其整个演化过程中所经历的减少模式的数量。

• 一致性变化模式数量（Consistent Number）：克隆家系CGE在其整个演化过程中所经历的一致性变化模式的数量。

• 不一致变化模式数量（Inconsistent Number）：克隆家系CGE在其整个演化过程中所经历的不一致变化模式的数量。

• 分裂模式数量（Split Number）：克隆家系CGE在其整个演化过程中所经历的分裂模式的数量。

2.6 演化特征挖掘阶段

为了从克隆代码及其演化过程中挖掘演化特征，本文将使用使用 WEKA（“Waikato Environment for Knowledge Analysis”[113]）中提供的聚类方法来分析克隆代码，将从克隆片段、克隆组和克隆家系三个不同的角度进行分聚类析。本文将这个挖掘任务分成两个子任务：第一，对获得的所有克隆实体进行统计分析，并且根据统计分析结果获取克隆代码的演化特征。第二，使用 WEKA 对克隆实体进行聚类分析，根据聚类结果获取克隆代码的演化特征。

2.6.1 克隆实体聚类

使用 WEKA 聚类克隆实体也可以划分为三个部分：首先，为每一个克隆实体成一个“克隆聚类向量”，包含每个克隆实体的所有度量值。然后，使用克隆聚类向量生成克隆聚类空间，表示不同的克隆实体。最后，使用 WEKA 聚类这些克隆向量，并根据聚类结果提取克隆代码演化特征。

首先，为每一个克隆实体生成克隆聚类向量。对于每个克隆片段、克隆组和克隆家系实体，其聚类向量是一个 m 维向量：即$Vector$=（v_1，v_2，...，v_m），其中 v_i 表示该克隆实体的一个特定度量值。以某一克隆组CG为例，为该克隆组CG生成一个 8 维向量$Vector$=（v_1，v_2，...，v_8），其中 v_i（对于所有 i，$i \leq 8$）是此克隆组CG对应的度量值。

然后，生成所有克隆代码实体的聚类向量空间。为了聚类所有的克隆实体，本文将生成系统的全部克隆聚类向量。所有克隆实体的聚类向量可以组成聚类向量空间 X。X 可以由X=（x_1，x_2，...，x_n）表示，其中 n 是克隆实体的数量。本文从克隆片段、克隆组和克隆家系三个角度考聚类分析克隆代码，因此也有三个不同的克隆向量空间：X_{CF}、X_{CG} 和 X_{CGE}，其中 X_{CF} 表示所有的克隆片段聚类空

间，X_{CG} 表示所有的克隆组聚类空间，X_{CGE} 表示所有的克隆家系聚类空间。

最后，使用 WEKA 聚类克隆特征向量。WEKA 是一个用于数据挖掘的流行机器学习工具，WEKA 中实现了许多方法来分析数据，例如聚类，分类，关联规则等等（本文所使用的聚类方法由 WEKA 实现）[2]。克隆代码的向量空间 X_{CF}、X_{CG} 和 X_{CGE} 分别表示了系统中全部的克隆片段、克隆组和克隆家系。本文使用聚类方法聚类这三个不同的向量空间，并分析得到克隆演化特征。

2.6.2 X-means 聚类

本文选择的聚类方法是 X-means 聚类 [108]，X-means 聚类是 K-means 聚类 [114] 的改进方法。后者可以将 X 向量空间聚类为事前指定的 K 个 Cluster 中，并使得相似的向量分配到同一个 Cluster 中。但是，K-means 聚类必须制定所需要聚类的 Cluster 的数量。然而对于克隆代码聚类而言，难以确定所需的 Cluster 的数目，因此本文选择使用 X-means 聚类。X-means 聚类并不需要具体的指定 Cluster 数量，其会自动搜索最佳的 Cluster 数量，因此更为适合于克隆代码演化特征的聚类分析。

X-means 聚类是一种高效的聚类算法，可以通过自动搜索并确定聚类数量 [108]。给定一个克隆向量空间 X =（x_1，x_2，...，x_n），其中每个 x_i 是 d 维实向量）。X-means 聚类将向量空间 X 中的 n 个向量划分为 k 个 Clusters，即 C =（C_1，C_2，...，C_k），C_i 表示一个 Cluster。使用 X-means 聚类只需要指定 K 的范围即可，聚类算法会自动的搜索最佳的 Cluster 数量 K。

2.7 实验结果与分析

2.7.1 实验设置

在本节中，首先简要描述了所使用的开源实验系统，并描述了从三个不同克隆视角挖掘克隆演化特征的实验步骤。

选择两个开源软件作为本章的实验系统：分别为 ArgoUML 和 jEdit，两个开源系统都至少经历了 10 个版本的演化。表 2-1描述了这两个实验系统的基本信息。从表中第 2-4 列可以看出，ArgoUML 经历了 14 个版本的演化（起始和结束版本分别为 0.20.0 和 0.34.0）， jEdit 经历了 22 个版本的演化（起始和结束版本分别为 3.0.0 和 5.0.0）。表中第 5-6 列则给出了实验系统的克隆实体数量（克隆聚类空间大

[2]WEKA 是新西兰怀卡托大学用 Java 开发的数据挖掘软件工具，其几乎可以运行在所有操作系统平台上。其网站是：http://www.cs.waikato.ac.nz/ml/weka/

小）。其中，“Clone Fragment”列出了所聚类的克隆片段的数量，“Clone Group”和“Clone Genealogy”分别是所聚类的克隆组和克隆家系数量。

表 2-1 两个开源软件实验系统信息

Table2-1 The information of two open sources experimental projects

Projects	Versions	Start Version	End Version	Clone Fragment	Clone Group	Clone Genealogy
ArgoUML	14	0.20.0	0.34.0	25422	7012	1036
jEdit	22	3.0.0	5.0.0	6636	2256	237

本章实验可以分为三个部分，从克隆片段、克隆组和克隆家系三个不同的视角挖掘和分析克隆代码的演化特征，即克隆片段实验、克隆组实验和克隆家系实验。在克隆片段实验中，重点分析克隆片段在演化过程中的变化情况。在克隆组实验中，分析克隆组在演化过程中的克隆演化模式情况，并重点分析了克隆代码的一致性和不一致变化模式。在克隆家系的实验中，从全局的角度分析了系统中全部克隆代码的演化规律，重点讨论了克隆代码的稳定性以及一致性变化问题。

在每个实验中，本文均使用两种方法分析和挖掘克隆代码的演化特征：统计分析和聚类分析。第一种统计分析每种克隆实体的度量值，进而发现一些基本的演化特征。第二种是聚类分析方法，使用 X-means 分别聚类每一种克隆实体，并根据聚类结果深入挖掘克隆代码的演化特征。

2.7.2 克隆片段实验

克隆片段是指软件中存在的真实的代码片段，在其生命周期内，克隆片段可能会被开发人员修改。克隆片段实验中考虑克隆片段的三个度量值，分别是 Clone Life、Ischanged、Change Times，上述度量值将帮助了解克隆片段在其演化过程中的真实变化情况。

首先，对克隆片段变化次数进行了统计分析，分析结果如表 2-2 和表 2-3 所示。从表中可以看出，大多数克隆片段在演化过程中并不会发生变化（未发生变化的克隆片段数量在 ArgoUML 中为 24327 个，在 jEdit 中为 5885）。同时只有仅仅一小部分克隆片段在演化过程中发生了变化（ArgoUML 为 1095 个，jEdit 为 751 个）。值得注意的是，在发生变化的克隆片段中有极少数的克隆片段被改变了不止一次，其数量随着改变次数的增大而减少。因此可以得出结论：克隆片段在其生命期间是十分稳定的（大多数克隆片段从未发生变化），但依然存在一定数量的克隆代码片

段发生了变化；且在发生变化的克隆片段中，克隆变化不会频繁地发生，仅有极少量的克隆代码会频繁的发生变化。

表 2-2 ArgoUML 中克隆片段的变化情况统计

Table2-2 The statistic of clone fragment change for ArgoUML

Change Times	0	1	2	3
Number	24327	982	109	4
Total	24327	1095		

表 2-3 jEdit 中克隆片段的变化情况统计

Table2-3 The statistic of clone fragment change for jEdit

Change Times	0	1	2	3	4	5	6	7
Number	5885	533	135	47	14	10	11	1
Total	5885	751						

随后，使用 X-means 方法对克隆片段进行聚类分析，实验结果如表 2-4 和 2-5 所示。X-means 聚类将 ArgoUML 和 jEdit 的克隆代码片段分成 4 个 Cluster：即 Cluster0–Cluster4。本文根据聚类结果分别统计了四个 Cluster 的度量值信息，即平均值（Mean）、标准差（Standard Deviation，SD）和中位数（Median）。从表中可以看出，在 ArgoUML 和 jEdit 两个开源软件中，Cluster0 的克隆片段中数量最少，该 Cluster 中的克隆代码在其演化的过程中发生过变化（Change Times 约等于 1），本文将这些克隆片段称为 changed 克隆片段。因此，在所有的克隆代码片段中仅有少量的克隆代码片段发生过变化，同时相对于未发生变化的克隆代码（Cluster3，ischanged 和 Change Times 均为 0），发生变化的时刻往往是其存在于系统中一段时间后（对比 Clone Life。除此之外，根据 “isChanged” 列可以看出，在系统 ArgoUML 中的 Cluster1、Cluster3 和系统 jEdit 中的 Cluster1、Cluster2 和 Cluster3 中，所有的克隆片段都没有发生变化，它们的数量占到克隆片段数量的多数。这意味着大多数的克隆片段在其演化的过程中是稳定的，不会发生变化。最后，系统 ArgoUML 和 jEdit 中的 Cluster3 是完全没有发生变化的克隆片段，根据其寿命发现它们在软件中存在的时间极短。这表明刚刚出现在系统中的克隆是极其稳定的（在短时间内不会发生变化）。因此，我程序开发人员应该更多地关注那些已经存在系统中一段时间（存在几个版本）的克隆代码片段，因为它们更容易发生变化。

综上所述，只有少数的克隆代码片段在软件演化过程中会发生变化，同时这些克隆片段所经历的变化通常发生在它们在系统中存在一段时间之后。

表 2-4 ArgoUML 中克隆片段的聚类结果

Table2-4 Clustering results of clone fragment for ArgoUML

Cluster	Number (Percentage)	Clone Life			Ischanged			Change Times		
		Mean	SD	Median	Mean	SD	Median	Mean	SD	Median
Cluster 0	899(4%)	7.207	2.299	7	0.092	0.290	0	1.130	0.350	1
Cluster 1	3082(12%)	7.763	1.523	8	0	0	0	0	0	0
Cluster 2	3006(12%)	3.833	0.871	4	0.058	0.234	0	0.065	0.247	0
Cluster 3	18435(73%)	1.094	0.292	1	0	0	0	0	0	0

表 2-5 jEdit 中克隆片段的聚类结果

Table2-5 Clustering results of clone fragment for jEdit

Cluster	Number (Percentage)	Clone Life			Ischanged			Change Times		
		Mean	SD	Median	Mean	SD	Median	Mean	SD	Median
Cluster 0	200(3%)	5.325	2.690	5	1	0	1	1.64	1.148	1
Cluster 1	1371(21%)	9.071	2.885	8	0	0	0	0.503	0.916	0
Cluster 2	1624(15%)	4.227	1.112	4	0	0	0	0.065	0.261	0
Cluster 3	3441(66%)	1.175	0.3780	1	0	0	0	0	0	0

2.7.3 克隆组实验

克隆片段实验仅提供了克隆片段本身的变化情况，而通过对克隆组进行实验分析则可以提供了克隆代码以克隆组为单位的克隆变化情况。在克隆组实验中，通过统计和聚类克隆组在演化过程中的克隆演化模式，从而揭示克隆代码的演化特征。

本文首先统计了克隆组的“克隆演化模式”（Clone Pattern）数量，结果如表 2-6 和 2-7 所示。表中使用“Present”和“Absent”来标识克隆组是否具有某种具体克隆演化模式[3]。同时，本文也非正式地将克隆演化模式中“Static”模式和“Same”模式称为“稳定的克隆演化模式”（Stable Clone Pattern），其它的克隆模式称为“动态的克隆模式”（Dynamic Clone Pattern）。从表中可看出，在两个实验系统中，大多数的克隆组（比例约 72% - 85%）具有有稳定的克隆模式，只有一小部分克隆组（其比例小于 7%）具有动态的克隆模式（即具有 Add，Sustract，Split，Consistent/Inconsistent Change）。值得注意的是，在 ArgoUML 和 jEdit 中有数百个

[3]克隆组的演化模式之间不是相互独立的，一个克隆组可以具有多个演化模式，例如可以同时具有“Static”和“Same”。

克隆组具有 Consistent/Inconsistent Change 模式。这应该引起程序开发人员的注意，因为这种变化模式– 特别是 Consistent Change 模式– 会导致额外的维护代价甚至会引发相关的克隆缺陷。从表中可以看出，在系统 jEdit 和 ArgoUML 中，发生一致变化（Consistent Change）模式的克隆组要多于不一致变化（Inconsistent Change）模式的克隆组（ArgoUM 具有两种模式克隆组数量分别为：350 和 329，jEdit 为 140 和 41）。因此可以得出结论：相比于不一致变化模式，软件系统中的克隆组更容易发生一致性的变化。

表 2-6 ArgoUML 中克隆组的演化模式统计

Table2-6 The statistic of clone group evolution pattern for ArgoUML

Number of Groups	Static	Same	Add	Subtract	Consistent Change	Inconsistent Change	Split
Present	5114	5422	345	324	350	329	36
Absent	1898	1590	6667	6688	6662	6683	6976
Percentage	72.93%	77.40%	4.92%	4.62%	5.25%	4.69%	0.51%

表 2-7 jEdit 中克隆组的演化模式统计

Table2-7 The statistic of clone group evolution pattern for jEdit

Number of Groups	Static	Same	Add	Subtract	Consistent Change	Inconsistent Change	Split
Present	1783	1922	45	36	140	41	19
Absent	473	334	2211	2220	2116	2215	2237
Percentage	79.3%	85.20%	1.99%	1.60%	6.21%	1.82%	0.84%

表 2-8 和 2-9 给出了克隆组聚类分析的结果。与克隆片段聚类实验相似，聚类结果也分成了四类。本文统计了每一类的克隆组的演化模式，并使用“Mean”表示平均值、“SD”表示标准差、“Median”表示中位数。从表中可以看出，Cluster1 的克隆组的数量最多（在 ArgoUML 中比例为 71%，在 jEdit 中占 79%）。Cluster1 的克隆组比较稳定（克隆组具有稳定的克隆模式，并且没有动态的克隆模式），同时具有相对较长的寿命。因此，大多数的克隆组是非常稳定的，也具有相对较长的寿命（在两个软件中大约 5 个版本）。同时，从表中还可以看出大多数的不一致变化模式（Inconsistent Change Pattern）出现在 Cluster0 中，并且仅仅占用很小的比例（ArgoUML 中的 4%，在 jEdit 中只有 1%）。因此可以得出结论不一致变化模式在克隆组中不会频繁发生。另外，一致性变化模式（Consistent Change Pattern）仅发

生在 Cluster2 中，其存在的克隆组的寿命相对较长，但相比于 Cluster0 会短一些。值得注意的是，Cluster0 和 Cluster2 都是动态克隆组，因为这些克隆组都具有动态的克隆演化模式。因此得出结论，动态的克隆演化模式往往会发生在较为长寿的克隆组中，但是它们的数量非常小。

从 Cluster0 和 Cluster2 中克隆组的绝对数量上看，具有一致性变化模式的克隆组（Cluster2）的数量大于具有不一致变化模式的克隆组数量（Cluster0）。这意味着一致性变化模式相比于不一致性变化模式更容易发生。因此本文建议：开发人员在修改克隆组中某一克隆片段时需要考虑同时修改组内其它的克隆片段，即考虑克隆代码的一致性变化问题。

同时从 Cluster3 中可以看出，有相当一部分的克隆组具有极短的寿命（刚刚出现在系统中），因此其并没有相关的克隆演化模式。这也意味着在克隆组刚刚创建的初始版本中并不不需要考虑克隆组的变化情况以及对系统的影响问题。

表 2-8 ArgoUML 中克隆组的聚类结果

Table2-8 Clustering results of clone group for ArgoUML

Cluster	Metric	Group Life	Static Number	Same Number	Add Number	Subtract Number	Consistent Number	Inconsistent Number	Split Number
Cluster0	Mean	3.042	0.587	0.701	1	1	0	1	0.080
264	SD	2.251	0.493	0.459	0	0	0	0	0.271
(4%)	Median	2	1	1	1	1	0	1	0
Cluster1	Mean	4.909	1	1	0	0	0	4.03E-4	6.05E-4
4959	SD	3.098	0	0	0	0	0	0.020	0.025
(71%)	Median	5	1	1	0	0	0	0	0
Cluster2	Mean	3.389	0	0.670	0.186	0.145	0.843	0.152	0.019
415	SD	2.666	0	0.471	0.389	0.352	0.364	0.360	0.138
(6%)	Median	2	0	1	0	0	1	0	0
Cluster3	Mean	0.608	0	0	0.003	0	0	0	0.003
1374	SD	0.520	0	0	0.054	0	0	0	0.054
(20%)	Median	1	0	0	0	0	0	0	0

综上所述，克隆组在其演化过程中通常是非常稳定的。当克隆组存在于系统的一段时间之后，动态的克隆演化模式可会发生在一小部分的克隆组中。同时，当开发人员修改某一克隆片段时，本文建议需要考虑克隆组内其它克隆片段是否需要一致性修改，即确定克隆组变化的一致性。

表 2-9 jEdit 中克隆组的聚类结果

Table2-9 Clustering results of clone group for jEdit

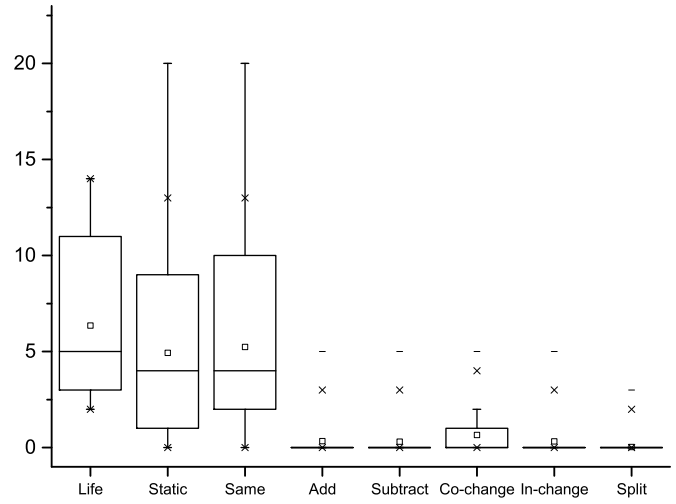
Cluster	Metric	Group Life	Static Number	Same Number	Add Number	Subtract Number	Consistent Number	Inconsistent Number	Split Number
Cluster0	Mean	6.88	0.4	0.76	1	1	0	1	0.2
25	SD	4.438	0.5	0.436	0	0	0	0	0.408
(1%)	Median	7	0	1	1	1	0	1	0
Cluster1	Mean	5.762	1	1	0	0	0	0	5.64E-4
1773	SD	4.05197	0	0	0	0	0	0	0.024
(79%)	Median	5	1	1	0	0	0	0	0
Cluster2	Mean	5.362	0	0.872	0.128	0	0.94	0.027	0.060
149	SD	3.780	0	0.335	0.335	0	0.239	0.162	0.239
(7%)	Median	4	0	1	0	0	1	0	0
Cluster3	Mean	0.997	0	0	0.003	0.036	0	0.039	0.013
309	SD	1.239	0	0	0.057	0.186	0	0.194	0.113
(14%)	Median	1	0	0	0	0	0	0	0

2.7.4 克隆家系实验

克隆家系可以提供系统中克隆代码的全局视角，因此本文也对克隆家系进行了实验。在克隆家系实验中，本文依然首先统计了克隆家系在其整个生命周期中的克隆演化模式数量，然后使用 X-means 对克隆家系进行聚类分析，从而挖掘和分析克隆代码的演化特征。

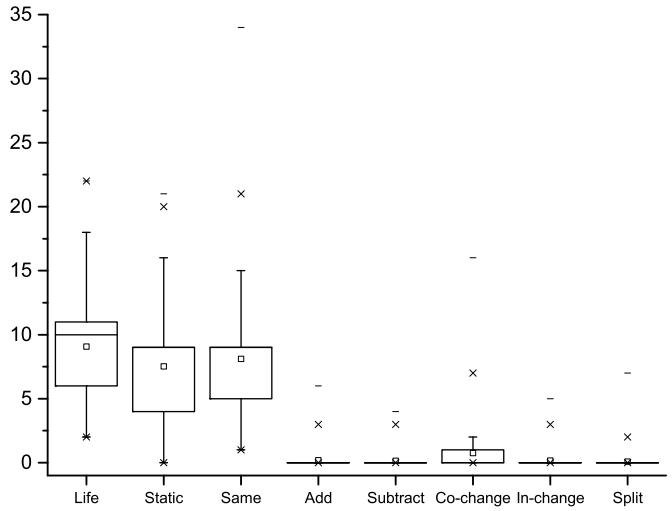
本文首先统计了克隆家系的所有度量值（包括克隆寿命和克隆演化模式数量），统计结果如图 2-3 所示。结果使用箱式图展示，图中横坐标表示克隆家系的演化情况，即本文所选用的克隆家系的度量值，纵坐标表示克隆家系度量值的数值范围。

从图中的“Life”可以看出，克隆家系在系统中会存在相当长的一段时间（Clone Life 的平均值，ArgoUML 中的克隆家系在全部 14 个版本中存在约 5 个版本，jEdit 的克隆家系在 22 版本中存在约 10 个版本）。同时，只有一小部分克隆家系存在极短的时间（少于 3 个版本）或极长的时间（多于 10 个版本）。另外，还可以看到静态的演化模式（Static 和 Same）的数量要远远高于动态的演化模式的数量。在动态的演化模式中，一致性变化（图中 Co-change）和不一致变化（图中 In-Change）模式的数量非常少，这意味着克隆家系在克隆演化的整个生命期间是非常稳定的。具体地，一致性变化模式的数量也多于超过不一致性变化模式的数量。这也意味着



a) ArgoUML 中克隆家系的演化模式统计

a) The statistics of clone genealogy evolution for ArgoUML



b) jEdit 中克隆家系的演化模式统计

b) The statistics of clone genealogy evolution for jEdit

图 2-3 克隆家系的演化模式统计

Fig. 2-3 The statistics of clone genealogy evolution

在演化过程中克隆代码更容易发生一致性变化，因此当修改克隆代码时，应该考虑是克隆代码的一致性问题。

表 2-10 ArgoUML 中克隆家系的聚类结果

Table2-10 Clustering results of clone genealogy for ArgoUML

Cluster	Death	Metric	Life	Static	Same	Add	Subtract	Consistent	Inconsistent	Split
Cluster0		Mean	11.854	9.795	10.451	2.232	2.232	3.061	2.293	0.390
82	5	SD	1.820	2.989	3.048	1.046	1.081	0.851	1.071	0.843
(8%)		Median	11	10	10	2	2	3	2	0
Cluster1		Mean	10.192	8.831	9.096	0.171	0.122	0.444	0.122	0.005
385	39	SD	1.680	1.676	1.703	0.398	0.328	0.648	0.328	0.102
(37%)		Median	11	9	10	0	0	0	0	0
Cluster2		Mean	3.294	1.255	2	0.471	0.461	1.260	0.461	0.010
204	188	SD	1.228	1.355	1.324	0.639	0.6389	0.440	0.638	0.140
(20%)		Median	3	1	2	0	0	1	0	0
Cluster3		Mean	2.795	1.795	1.795	0	0	0	0	0
365	348	SD	1.081	1.081	1.081	0	0	0	0	0
(35%)		Median	3	2	2	0	0	0	0	0
All		Mean	6.359	4.936	5.234	0.333	0.313	0.655	0.318	0.035
1036	579	SD	4.025	4.022	4.062	0.750	0.745	0.974	0.757	0.272
(100%)		Median	5	4	4	0	0	0	0	0

本文同样使用了 X-means 聚类系统中克隆家系演化情况（克隆寿命和克隆演化模式数量），进而挖掘更多的克隆演化特征，结果如表 2-10和 2-11 所示。本文还定义了一个“Death”的特殊变量，并用其标识实验所收集的克隆家系是否已经。表中“Death”列表示已经死亡的克隆家系的数量，从表中的“Cluster”列和“Death”列中可以发现，本文所收集的克隆家系是完整的，即大部分的克隆家系仍在演化中并没有死亡。同时从表中可以看出，X-means 将所有克隆家系可以聚类成四个 Cluster。

为了分析克隆家系的演化情况，本文非正式给出“稳定的”和“动态的”的克隆家系。如果一个克隆家系中具有大量的稳定的克隆演化模式，即克隆家系中存在大量的“静态”和“相同”演化模式，则称克隆家系是“稳定的”克隆家系（Stable Clone Genealogy）。相反地称一个克隆家系是“动态的”克隆家系（Dynamic Clone Genealogy），即该克隆家系具有相对较多的“add”，“subtract”，“split”和“Consistent/Inconsistent Change”克隆演化模式。

表 2-11 jEdit 中克隆家系的聚类结果

Table2-11 Clustering results of clone genealogy for jEdit

Cluster	Death	Metric	Life	Static	Same	Add	Subtract	Consistent	Inconsistent	Split
Cluster0		Mean	19	15.8	19.2	3	2.3	6	2.7	1.1
10	3	SD	4.830	4.566	6.426	1.333	0.949	4.570	1.418	2.234
(4%)		Median	22	17	19	3	2	4.5	2.5	0
Cluster1		Mean	10.952	9.445	9.938	0.075	0.075	0.589	0.082	0.041
146	35	SD	2.356	2.166	2.358	0.265	0.313	1.074	0.343	0.285
(62%)		Median	10	9	9	0	0	0	0	0
Cluster2		Mean	6.571	4.75	5.607	0.071	0.071	0.857	0.071	0.071
28	24	SD	1.168	1.143	1.227	0.262	0.262	0.970	0.262	0.378
(12%)		Median	7	5	6	0	0	1	0	0
Cluster3		Mean	3.340	2.132	2.302	0.038	0	0.208	0	0
53	48	SD	0.732	0.921	0.723	0.192	0	0.454	0	0
(22%)		Median	3	2	2	0	0	0	0	0
All		Mean	9.072	7.523	8.110	0.190	0.152	0.764	0.173	0.080
237	110	SD	4.366	4.079	4.569	0.690	0.554	1.706	0.664	0.550
(100%)		Median	10	9	9	0	0	0	0	0

从图 2-3和表 2-10 和 2-11 中可以很明显的看出，大多数的克隆家系是“稳定的”克隆家系（Cluster1，2，3），并且克隆家系寿命和“稳定的”克隆模式之间存在较强的正相关关系。另一方面，“动态的”克隆家系（Cluster0）数量十分稀少，并且具有较长的寿命且依然存在于系统中。这表明“动态的”克隆演化模式–特别是一致性变化和不一致性变化– 通常发生在寿命较长的克隆家系中。因此，软件开发人员应该对动态的克隆家系采取一些必要的措施，因为不一致性的变化可能导致软件缺陷。

对于 Cluster3 中的克隆家系（表 2-10和 2-11），其寿命较短，并且它们中的大多数已经死亡。但是，这些克隆家系却非常稳定，不存在不一致性变化演化模式。因此可以得出结论：寿命较短的克隆家系比寿命较长的克隆家系更为稳定。这也提醒软件开发人员，不需要关注那些新创建的克隆家系（寿命较短的克隆家系），但随着时间的推移，程序开发人员更应该关注那些依然存在系统中的寿命较长的克隆家系。

综上所述，克隆家系在整个演化过程中大多是稳定的，而较短寿命的克隆家系相比寿命较长的克隆家系更为稳定。同时，动态的演化模式通常发生在较长寿命的克隆家系中（即使其数量比较稀少），其中一致性变化模式比不一致性变化模式更为频繁。因此，本文建议开发人员应该更加注意寿命较长的克隆家系，并且当克隆代码发生变化时需要考虑同组克隆代码的一致性变化。

2.8 本章结论

在本章中，为挖掘克隆代码及其演化过程的克隆代码演化特征，提出了一个基于聚类的克隆代码演化特征分析方法。具体来说，本章从克隆片段、克隆组和克隆家系三个不同的角度提取相应的度量值表示克隆代码及其演化过程，并使用聚类分析（X-means）的方法挖掘和分析克隆代码演化特征。（1）本章提出一个框架来分析多版本软件的演化情况，并使用聚类方法挖掘克隆代码演化特征，从而帮助程序开发人员理解和维护克隆代码，可帮助提高系统的可理解性。（2）本文从三个不同角度分析和表示克隆代码及其演化情况，提取提取不同的度量值表示克隆片段、克隆组和克隆家系。克隆片段角度重点关注克隆的实际变化情况，克隆组角度关注克隆组的演化模式情况，克隆家系则提供了注系统中全部克隆代码的全局视角。（3）在 ArgoUML 和 jEdit 两个软件系统上的实验结果表明：克隆代码通常在其演化过程中非常稳定，不会频繁的发生变化；同时，在克隆演化过程中，仍然存在相当数量的发生变化的克隆代码，需要引起开发人员的关注；最后，由于发生变

化的克隆中，一致性变化的数量多于不一致性变化，因此建议开发人员应在修改克隆代码片段时需要考虑克隆代码的一致性问题。本文相信这些演化特性可以帮助开发人员更好地理解克隆，还可以提供一些指导来维护和管理软件开发中的克隆。

第 3 章 基于贝叶斯网络的克隆创建时一致性维护需求预测方法

3.1 引言

由于日益增长的软件开发的需求，开发人员在软件开发过程中通过复制粘贴既有代码（称为克隆创建），向系统中引入新的克隆代码。新产生的克隆代码会随着系统演化，其演化过程中可能会发生一致性变化。然而，克隆代码的一致性变化不仅会导致系统额外的维护代价，遗忘该一致性变化还会引入与之相关的克隆一致性缺陷，从而将进一步增大系统的维护代价。本文将由复制粘贴操作创建的克隆代码在其演化过程中所发生的一致性变化，称为克隆代码创建时的一致性维护需求。在克隆代码创建时预测克隆代码的一致性维护需求，可以帮助降低系统的维护代价，从而帮助提高软件质量和可维护性。

鉴于此，在定义克隆代码创建时的一致性变化和一致性维护需求的基础上，本章使用机器学习中的贝叶斯网络方法在克隆代码创建时预测其一致性维护需求。为训练克隆代码创建时一致性预测模型，首先通过构建软件系统的克隆家系来收集系统中所有的克隆创建实例（复制粘贴导致的克隆代码）。然后，提取了代码属性和上下文属性两组属性值表示克隆代码创建实例。最后，使用贝叶斯网络方法训练机器学习模型，并预测克隆代码创建时的一致性维护需求。在四个开源软件系统上对本章方法进行了评估，实验结果表明本章方法以较高的准确率和召回率高效地预测克隆代码创建时的一致性维护需求。本章所提出的预测方法可以帮助程序开发员在克隆代码创建时预测克隆代码的一致性，降低克隆代码导致的额外的维护代价，从而提高软件的质量和可维护性。

3.2 克隆代码创建时一致性维护需求

3.2.1 相关研究

在软件开发过程中，通过复制和粘贴操作复用既有代码已经成为一种常见的软件开发手段 [23]。复制粘贴活动可以减少软件开发时间和提高软件开发效率，但同时也会向软件系统中创建新的克隆代码（克隆创建）。克隆代码会长时间的存在于软件系统中，并同时随着软件系统进行演化。克隆代码随系统进行演化的过程

可以使用克隆家系进行描述 [16]（克隆演化见本文 **??** 节）。经研究发现在克隆代码的演化过程中，克隆组内的某一克隆代码片段可能会被程序开发人员修改，从而引发克隆代码的变化。同时，由于同组的克隆代码彼此相似，这种变化也可能会传播到同组的其它克隆代码片段中，从而引起克隆组一致性变化 [17]。

为了帮助分析克隆代码的变化以及演化情况，本文第二章基于机器学习中的聚类方法挖掘克隆代码的演化特征。研究表明在克隆代码的演化过程中，存在相当数量的克隆代码会发生变化。这些发生变化的克隆代码也会引发克隆组的一致性变化模式。更为重要的是，演化中的克隆代码的一致性变化往往多于不一致性变化，因此在第二章中本文建议程序开发人员需要警惕克隆变化，并且当发生变化时需要考虑克隆代码的一致性问题。（具体可参考本文 2.7 节）

为确保演化中的克隆代码的一致性，程序开发人员需要对发生变化的克隆进行一致性维护，从而导致额外的维护代价。而遗忘克隆的一致性变化，会导致克隆代码的不一致缺陷，从而进一步增加软件的维护代价 [22][51]。本文将在演化过程中由于克隆一致性变化所导致的维护代价称为一致性维护代价。为了避免克隆代码的一致性维护代价，程序开发人员可以在复制粘贴时避免可能发生一致性变化的克隆代码，或者仅允许不会发生一致性变化的克隆代码的产生。换言之，在复制粘贴时预测克隆代码在演化过程中是否会发生一致性变化，可帮助避免一致性维护代价。

因此，Wang 等人基于贝叶斯网络在复制粘贴时对克隆代码进行一致性维护需求预测，帮助避免需要一致性维护的克隆代码 [12][80]。该方法中提取了历史属性、代码属性和上下文属性三组属性表示复制粘贴操作，并使用贝叶斯网络预测克隆代码一致性维护需求。但是，该方法存在以下不足之处：

- 首先，所使用的历史属性与复制粘贴操作的关联性较弱，仅表示其所在文件的历史变化情况。同时历史属性的提取也增加了方法本身的困难程度 (历史属性提取需要分析软件全部版本源代码)；
- 其次，所提取的代码属性和上下文属性不够充分，并不能完全的表示复制粘贴操作所导致的克隆代码。例如，代码属性中仅考虑了克隆代码与系统其它模块的调用和访问关系，并没有详细的考察克隆代码自身的一些属性特征。
- 最后，方法中没有清晰的给出克隆变化以及一致性维护需求的定义。

鉴于此，本章对复制粘贴产生的克隆代码的一致性维护需求，进行了进一步的深入研究。本章在克隆代码演化的基础上结合克隆代码一致性维护，首先给出了一种克隆代码一致性变化以及一致性维护需求的定义，可更为准确地帮助预测克隆的一致性维护需求。然后，本文在 Wang 等人研究的基础上，改进了克隆代码一致

性维护需求预测所使用的度量值。舍弃了与复制粘贴操作关联性不强的历史属性，并进一步扩展了代码属性和上下文属性，可以更为详细和细致的表示由复制粘贴操作导致的克隆代码。最后，使用贝叶斯网络在克隆代码创建时预测其一致性维护需求。本章方法可以帮助程序开发人员降低克隆代码的一致性维护代价，避免克隆一致性缺陷。

3.2.2 克隆创建时一致性维护需求定义

在克隆代码的演化过程中，克隆片段可能会被开发人员修改，从而导致克隆代码的一致性变化，这种变化可能会导致额外的维护代价。为了描述这种克隆代码的修改，本章给出克隆代码变化时一致性变化定义，如下所示：

定义 3.1 (创建时一致性变化（Changing Consistent Change）) 给定两个克隆代码片段 CF_1 和 CF_2，且它们被分别地修改为 CF_1' 和 CF_2'。如果克隆代码 CF_1 和 CF_2 的变化满足以下条件，称此变化为克隆创建时的一致性变化（Creating Consistent Change），

$$\textit{textSim}(CF_i, CF_i') < 1 \; \forall i \in \{1, 2\}$$

注意 $\textit{textSim}(CF_i, CF_i') = 1 - UPI(CF_i, CF_i')$，UPI 是两个代码片段之间不同的代码行数占总代码行的比例。假如给定两个代码片段 CF_1 和 CF_2，其 $UPI(CF_1, CF_2) = 0.3$ 表示两者的差异程度为 30%。同时，其相似度可根据 UPI 计算：$SimText(CF_1, CF_2) = 1 - UPI = 0.7$，表示两者的相似度为 70%。本文在检测克隆代码时，设置克隆代码的相似度阈值为 0.7，即相似度大于阈值的代码片段为克隆代码。该定义仅要求克隆代码片段 CF_1 和 CF_2 被同时修改。

创建时一致性变化，仅要求克隆代码片段被同时的修改。原因在于：在克隆代码创建时，目标是避免新创建的克隆代码在其未来演化过程中的一致性变化，及其所导致额外的维护代价。因此，只要两个克隆片段在其演化过程同时变化，即认为会导致额外维护代价。

克隆创建（复制粘贴操作）所导致的克隆组，会随着系统的演化而演化。在此过程中，组内克隆片段的变化也导致了克隆组的变化，并使用克隆演化模式描述克隆组的变化，称其为一致性变化模式或不一致变化模式。克隆组一致性变化模式定义如下所示：

定义 3.2 (创建时一致性变化模式（Creating Consistent Change Pattern）) 在软件版

本 $j+1$ 中存在一个克隆组 CG'，假设克隆组内至少存在两个克隆代码片段 CF_1' 和 CF_2' 可以与映射到上一版本 j 的克隆组 CG 中，且 CG 中与之对应的克隆代码片段的 (CF_1, CF_2) 被修改为 (CF_1', CF_2')。如果克隆片段之间的变化（(CF_1, CF_2) 变化至 (CF_1', CF_2') 满足克隆片段的“一致性变化（Consistent Change）”，则称克隆组 CG' 具有一致性变化模式（Consistent Change Pattern）。

本章定义的克隆创建时的一致性变化模式，与其它论文中的定义不同。其原因在于：本章目的在于预测克隆代码的一致性所引发的维护代价。为了便于读者更容易理解克隆创建的克隆组的一致性维护需求，本章现给出克隆创建实例的定义，如下所示：

定义 3.3 (克隆创建实例（Clone Creating Instance）) 克隆创建实例：软件版本 j 中的一个克隆组 CG 是克隆创建实例，如果该克隆组 CG 是其克隆家系 CGE 的根节点。

给定一个克隆创建实例所产生的克隆组，在其演化过程中，可能会发生一致性变化模式。克隆组的一致性变化会引发一致性维护代价，从而降低软件质量。因此，在克隆创建时，预测其在演化过程是否会发生一致性变化，可以避免与克隆创建导致的克隆代码的一致性维护代价。本文将创建时克隆代码导致的一致性变化，称为克隆一致性维护需求。克隆创建时的一致性维护需求定义如下：

定义 3.4 (创建时一致性维护需求（Creating Consistency-Requirement）) 给定版本 j 中一个克隆创建实例，CG 满足克隆一致性维护需求（Consistency-Requirement），如果在版本 k 中存在一个克隆实例 CG'（$k > j$）满足以下条件：(1) 在 CG' 中至少存在两个克隆片段在其克隆家系 CGE 中可以映射到克隆实例 CG 中，(2) CG' 具有“一致性变化模式”（Consistent Change Pattern）。反之，假如克隆创建实例 CG 不满足克隆一致性维护需求条件，称该克隆实例不需要一致性维护（Consistency-Requirement Free，或者 Consistency-Free）。

最终，可以将本章的的研究问题表述如下：给定一个克隆创建实例 CG，即复制粘贴活动导致的克隆代码，预测创建实例 CG 是否满足克隆创建时的一致性维护需求。

更进一步，根据上述定义克隆创建实例只有两种状态：满足和不满足一致性维护需求。本章克隆创建时的克隆一致性需求预测问题可转换为一个典型的分类问题，因此使用机器学习模型解决此分类问题，具体方法见下文。

3.3 克隆代码创建时一致性维护需求预测框架

为解决本章所提出的克隆一致性需求维护预测问题，本文首先给出了一个方法框架。基于贝叶斯网络的克隆创建时一致性维护需求预测框架如图 3-1所示。方法可以划分为三个阶段，克隆创建实例收集阶段、克隆创建实例表示阶段和一致性维护需求预测阶段。收集阶段旨在收集系统中全部的克隆创建实例，可将其用于使用机器学习方法中来训练预测模型。由于实际的克隆创建实例无法直接应用于机器学习方法中，因此在表示步骤中将提取相应的属性值表示克隆创建实例。接下来，在预测步骤，使用属性化的克隆创建实例构建和训练机器学习模型，并使用其预测克隆创建实例的克隆一致性维护需求。

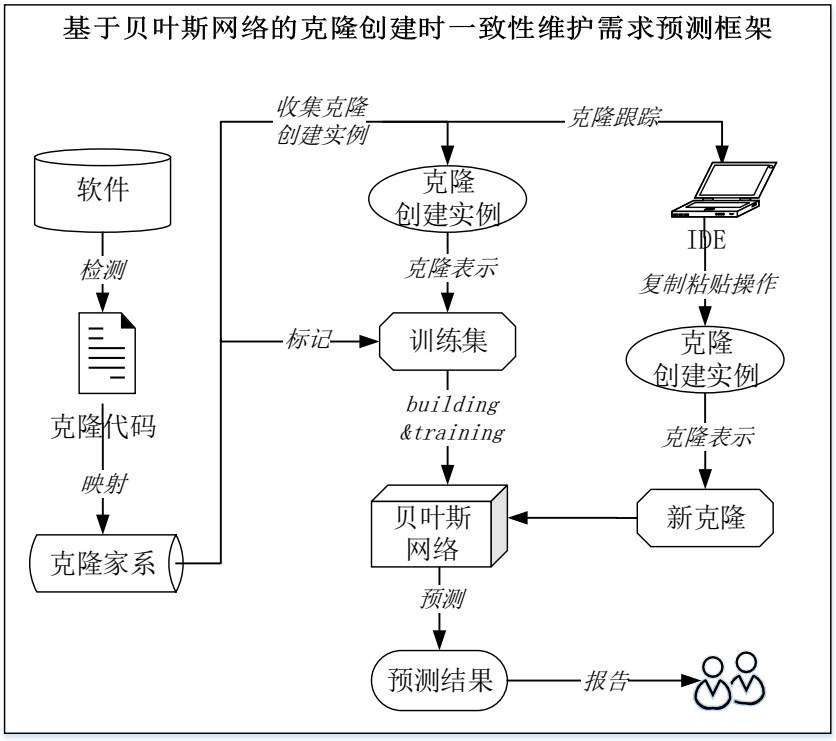


图 3-1 基于贝叶斯网络的克隆创建时一致性维护需求预测框架

Fig.3-1 The framework for clone creating consistency prediction based on Bayesian network

具体来说，在收集阶段中，通过构建系统的克隆家系从软件中收集所有的克隆创建实例。使用 NiCad 来检测软件版本中的所有克隆，并通过在相邻版本的克隆组之间进行映射来构建克隆家系，用于识别克隆创建实例。在表示阶段中，通过提取属性值表示克隆创建实例，提取了代码属性、上下文属性两组属性分别表示被复制和粘贴的克隆代码。在预测阶段中，使用收集到的克隆创建实例训练贝叶斯

网络，并在克隆创建时预测克隆一致性维护需求。在使用已构建好的模型进行预测时，可将该模型嵌入到软件开发环境中。软件开发环境需要实时监测复制粘贴操作（克隆创建实例），然后提取克隆创建实例的度量值。最后，使用模型预测其一致性维护需求，根据预测结果提醒程序开发人员采取进一步的操作。

克隆创建实例有两种不同的预测结果，即满足一致性维护需求和不满足维护需求。对于满足一致性维护需求实例来说，其在将来的演化中可能会引发一致性变化，程序开发人员可以根据情况选择拒绝此变化实例。对于不满足一致性维护需求实例来说，其在将来的演化中不会引发一致性变化，程序开发人员可以根据情况选择接受此变化实例。

本章基于贝叶斯网络预测克隆创建实例的一致性。使用克隆创建实例的属性表示贝叶斯网络中的节点，并用于构造贝叶斯网络的结构。使用收集到的克隆创建实例学习贝叶斯网络的参数，从而完成模型的训练，细节可参考本文后续章节。

3.4 克隆创建实例收集

收集克隆创建实例的目的在于生成克隆一致性预测的训练集，并将其用于训练机器学习模型。通过构建系统的克隆家系并识别其中的克隆演化模式，可以从软件中收集所有的克隆创建实例。首先使用 NiCad 来检测软件版本中的所有克隆，然后通过在相邻版本的克隆组之间进行映射来构建克隆家系，最后识别克隆一致性演化模式识别系统中的克隆创建实例。

根据定义 3.3 ，本文假定克隆家系 CGE 中第一次出现的克隆代码是由复制粘贴操作导致的，即是克隆创建实例。因此，通过检测系统的克隆代码并构建克隆家系，可以完成对克隆创建实例的收集。

（1）构建克隆家系。首先，下载系统所有版本的源代码，并使用 NiCad 的默认配置检测检测每一版本的中 Type1-3 的克隆代码。然后，通过映射所有相邻版本的克隆代码，构建系统中全部克隆家系。为完成版本间的映射，为每个克隆片段生成一个克隆区域描述符 CRD[99]，使用基于 CRD 的克隆映射算法映射两个连续版本之间的所有克隆片段和克隆组 [111][112]。根据克隆映射结果，构建系统的克隆家系。

（2）收集克隆创建实例并标识一致性维护需求。克隆家系是克隆演化的有向无环图，图中根节点是由复制粘贴操导致的克隆创建实例。因此，根据定义 3.3 通过遍历克隆家系的根节点，可收集系统中所有的克隆创建实例。在收集克隆创建实例的时候，同时根据定义 3.1 和 3.2 识别克隆一致性变化模式，从而确定克隆创建

实例在其未来演化过程发生的一致性变化。根据定义 3.4，如果克隆创建实例在其演化过程中发生了一致性变化模式（ 3.2 ），则该实例满足一致性维护需求，否则不满足维护需求。

（3）确认复制和粘贴代码。在收集克隆变化实例后，还需确认该实例中的被复制和被粘贴代码。由于被复制代码会较早地存在软件中，可将创建实例中的克隆代码向上一软件版本中进行映射。假定被复制代码会应存在于上一版本中，被粘贴代码不存在上版本中。根据映射结果，可能会存在两种情况：（a）其中一个克隆代码可以映射，另一个未映射。认为映射代码为被复制代码，未映射为被粘贴代码。（b）两者均没有映射。此情况下随机选取一个为被复制代码，另一个未未被粘贴代码。原因是两者互为克隆代码，彼此之间相似，故随机选取一个为被复制代码并不影响预测结果。注意不存在两者均可映射的情况，应在上一版本检测为克隆代码。[1]

3.5 克隆创建实例表示

本章使用贝叶斯网络预测克隆代码创建时的一致性维护需求，并使用软件中既有的克隆创建实例训练贝叶斯网络模型。但是，实际的克隆创建实例无法直接应用于贝叶斯网络中。因此，本文将提取相应的属性值表示克隆创建实例。将分别提取代码属性、上下文属性两组属性代表克隆创建实例中被复制和粘贴的克隆代码。

（1）代码属性

代码属性表示了被复制的克隆代码，从代码自身角度提取被复制克隆代码的特征。代码属性描述克隆代码的词法、语法、函数调用等信息。代码属性主要包括克隆代码粒度、Halstead 属性、结构属性、调用属性等，具体的代码属性如下所示：

- 克隆粒度：被复制克隆代码的规模，即所包含的代码行数。
- Halstead 属性：被复制克隆代码的代码复杂度，有四个基本的属性值，分别为操作符种类、操作数种类、操作符总量和操作数总量。
- 结构属性：被复制克隆代码的结构特征，是语句的统计信息，`if_then`, `if\_else`, `switch`, `while`, `do`, `for`, `this\_or\_super` 等。
- 参数访问数量：被复制克隆代码中所有函数的参数访问数量统计。
- 总函数调用次数：被复制克隆代码中所有函数调用的次数统计。
- 本地函数调用次数：被复制的克隆代码中，调用函数与被复制克隆片段在相同类的调用次数统计。

[1]幸运的是，大多数克隆组只有两个克隆片段，所以这个决定不会影响我们的克隆预测。对于具有多个克隆片段的克隆组，则随机选取两个作为克隆创建实例。

• 库函数调用次数：被复制的克隆代码中，库函数的调用次数统计，包括 java 库函数的调用、eclipse 库函数的调用以及第三方包函数的调用。

• 其它调用次数：被复制的克隆代码中，既不是库函数调用、也不是本地函数调用的其它调用次数统计，如同项目内其它包函数调用或同包内其它类中的函数调用。

（2）上下文属性

上下文属性是被复制和被粘贴代码之间的关系属性，描述了两者之间的克隆关系。上下文属性包括代码相似度、克隆分布、被复制和被粘贴代码之间的一些相似度等等。具体的上下文属性如下所示：

• 代码相似度：被复制与被粘贴的克隆代码之间的相似度，计算方法和 NiCad 相同，即 UPI[13]。

• 局部克隆标识：被复制及粘贴的克隆代码片段是否在同一个文件中。

• 文件名相似度：被复制和被粘贴克隆代码所在文件的名相似度。假定文件名分别为 M_1 和 M_2，则文件名相似度为 $Sim(M_1, M_2)$，采用李氏距离 [115] 计算 (剩余度量中相似度采用相同方法计算)。

• 文件名相似度标识：当克隆是局部克隆时，其文件名相似度为 1，为非局部克隆时为 0。该属性决定文件名相似度是否起效。

• 方法名相似度：被复制和被粘贴克隆代码所在方法的方法名字相似度。

• 总参数名相似度：假定被复制和被粘贴克隆代码所在方法为 M 和 N，计算其参数名相似度之和。假设 M 和 N 分别包含 m 和 n 个参数，即 $(P_1, P_2, ..., P_m)$ 和 $(Q_1, Q_2, ..., Q_n)$，则总参数名相似度为 $Sum(Sim(P_i, Q_j))$。

• 最大参数名相似度：假定被复制和被粘贴克隆代码所在方法为 M 和 N，其最大参数名相似度。假设 M 和 N 分别包含 m 和 n 个参数，即 $(P_1, P_2, ..., P_m)$ 和 $(Q_1, Q_2, ..., Q_n)$，最大参数名相似度为 $Max(Sim(P_i, Q_j))$。

• 总参数类型相似度：被复制和被粘贴代码克隆所在方法分别为 M 和 N，其参数类型相似度之和。假设 M 和 N 分别包含 m 和 n 个参数，其参数类型分别为 $(P_1, P_2, ..., P_m)$ 和 $(Q_1, Q_2, ..., Q_n)$，总参数类型相似度 $Sum(Sim(P_i, Q_j))$。

• 块信息标识：被复制和被粘贴克隆代码的上下文信息是否相同，相同为 1，反之为 0。

3.6 克隆创建时一致性需求预测

本章将克隆代码创建时的一致性维护需求问题，转化成了克隆创建实例的分类问题，即给定一个克隆创建实例，判别其是否满足克隆创建时的一致性维护需求。本文使用贝叶斯网络方法作为机器学习模型，并使用其预测克隆一致性维护需求。因此，本节先简单介绍贝叶斯网络方法。随后，使用属性化的克隆创建实例构建和训练贝叶斯网络模型，并使用训练好贝叶斯网络在克隆代码创建时预测其一致性维护需求。

3.6.1 贝叶斯网络方法

贝叶斯网络是（Bayesian network）是一种概率图型，可以使用已经观察到的事件来预测将来可能发生的事件 [116]。贝叶斯网络可以表示成一个有向无环图模型，图中的每一个节点表示一个随机事件，图中的边则表示随机事件发生的条件概率。因此，贝叶斯网络中的全部节点可以视为一组随机变量$X_1, X_2, ..., X_n$，贝斯网络的边所有边则可以使用随机变量的条件概率表描述（Conditional Probability Distributions, CPD）。

一般而言，贝叶斯网络的节点可以是随机变量，可以是可观察到的变量、属性、未知参数等。连接两个贝叶斯网络节点的边则代表两个随机变量之间是非条件独立的，使用事件的条件概率表示。如果两个节点间没有连接，就称其随机变量彼此间为条件独立。条件概率表（CPT）可以描述贝叶斯网络的节点和边的因果关系。

对克隆代码一致性维护需求预测而言，贝叶斯网络可用来表示克隆创建实例的属性值及其一致性维护需求间的概率关系。克隆创建实例的属性值是贝叶斯网络中的随机事件。给定一个具体的克隆创建实例，可以使用贝叶斯网络计算该实例满足一致性维护需求的概率。

3.6.2 训练与预测

接下来，使用收集到的克隆创建实例训练贝叶斯网络，并在克隆创建时预测克隆一致性维护需求。

本章没有对贝叶斯网络方法进行改进和研究，贝叶斯网络模型的构建和训练通过调用现有机器学习工具包 WEKA 完成。WEKA（Waikato Environment for Knowledge Analysis）全称是怀卡托智能分析环境，它是一个 Java 语言编写的，支持数据挖掘任务的工作平台。WEKA 集成了大量能承担数据挖掘任务的机器学习算法，

包括数据预处理，分类，聚类，关联规则，特征选择以及可视化功能。

对于每个软件系统，首先，通过收集克隆创建实例并提取相应的属性，用于构建模型训练所需的数据集。然后，调用 WEKA 中的贝叶斯网络实现构建和训练克隆一致性预测模型。

根据定义 3.4，克隆创建实例有两种不同的状态：需要一致性维护和不需要一致性维护。因此在进行一致性维护需求预测时，克隆创建实例也具有两种不同的预测结果：

• 不需要一致性维护：若克隆创建实例的预测结果为“不需要”，软件开发人员可以自由的执行克隆创建操作（复制和粘贴），从而节约开发时间提高开发效率。因为，该克隆创建实例，在未来演化的过程中不会引发一致性变化，也不会导致额外的维护代价。

• 需要一致性维护：若克隆创建实例的预测结果为“需要”，软件开发人员需要谨慎的执行克隆创建操作（复制和粘贴）。因为，该克隆创建实例，在未来演化的过程中可能会引发一致性变化，从而向系统中引入额外的维护代价。

在使用已训练好的模型进行预测时，可以与软件开发过程相结合，将该模型嵌入到软件开发环境中，帮助程序开发人员实现边开发边预测克隆创建实例的一致性维护需求。首先，在软件开发环境中需监测程序员的复制和粘贴操作，识别由此产生的克隆创建实例。然后，根据上文描述的代码和上下文属性，提取相应的特征表示该克隆创建实例。最后，使用训练好的预测器预测该克隆创建实例的一致性维护需求，根据预测结果提醒程序开发人员采取进一步的操作。

3.7 实验结果与分析

本节给出本章的实验结果与分析，首先简单介绍了实验所使用的实验系统和评估方法，然后详细给出每个实验的结果与分析。

3.7.1 实验系统与实验设置

为评估本章方法，本章选取了四个开源软件进行实验。四个实验系统的克隆创建实例统计情况如表 **??** 所示。具体来说，第 3 列和第 4 列分别给出了不需要一致性维护和需要一致性维护的克隆创建实例的数量和比例。不需要一致性维护的克隆实例，在其未来的演化中不会导致一致性变化和额外的维护代价。需要一致性维护的克隆实例，在其演化过程中可能导致一致性变化，从而增加系统维护代价。

从表 3-1 中可以得出两个发现。第一，软件系统中存在大量的克隆创建实例，

数量从 633 到 3366，其中项目 jEdit 是含有最少的克隆变化实例。这说明复制粘贴既有代码，已经成为了程序开发人员的一种常用开发手段。第二，软件系统中大部分的克隆创建实例在其演化过程中不满足一致性维护要求 (比例从 59.8% 到 88.47%)。这表明作为一种常用开发手段的复制粘贴操作并不会在演化中引入一致的变化，这意味着开发人员可以正常的使用这种技术。同时，还可以看出克隆代码在演化过程是比较稳定的，不易发生变化。这与本文第二章得出的克隆演化特征相一致。但是值得注意是：软件系统中依然存在相当数量的需要一致性维护的克隆创建实例，数量从 73 个到 1353 个。这也警告程序开发人员，即便可以使用复制粘贴复用已有代码，但也要注意克隆代码的一致性维护问题，不可以肆无忌惮的使用此技术。

表 3-1 实验系统的克隆创建实例信息统计

Table3-1 The statistics for clone creating instances in four projects

实验系统	克隆创建实例的数量（比例）		总数
	不需要一致性维护	需要一致性维护	
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

因为有两种类型的克隆创建实例，即不需要一致性维护实例和需要一致性维护实例。分别对上述两种克隆创建实例进行预测，可将实验划分为“一致性维护自由”实验（不需要一致性维护）和“一致性维护需求”实验（需要一致性维护）。同时，本章使用贝叶斯网络模型预测克隆创建实例的一致性，预测时贝叶斯网络会计算克隆创建实例的概率值，表示该实例需要一致性维护需求的概率 (在 0 1 之间，数值接近于 1 表示需要一致性维护，接近于 0 表示不需要一致性维护)。

针对不同的克隆创建实例，设置了不同的阈值进行实验分析。对于不需要一致性维护的实例，其预测值较小（接近于 0），选取阈值为 0.01、0.05、0.10、0.15 和 0.20，当预测值小于等于给定阈值时认定该实例不需要一致性维护。对于需要一致性维护的实例，其贝叶斯网络预测值较大（接近于 1），选取阈值为 0.5、1.6、0.7、0.8 和 0.9，当预测值大于等于该阈值认定该实例需要一致性维护。

为了全面评估本章提出的方法，上述两个实验中每一个都可以从三个角度进行实验分析，即又可以划分为全属性实验、属性组实验和交叉验证实验三个部分。

• 全属性实验：使用本章提取的所有属性对实验系统进行分析，评估本文方法的预测能力。

• 属性组实验：分别使用两组不同的属性组对实验系统进行分析，评估所提取的属性组对预测能力的影响。

• 交叉验证实验：使用其它系统数据作为训练集训练模型，并使用该模型在新系统上进行实验分析，从而探索模型在应用到新系统的预测能力。

本文使用使用开源软件工具 WEKA 训练和预测贝叶斯网络。在构建贝叶斯网络时，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，使用 SimpleEstimator 来估计贝叶斯网络的条件概率表。在实验时将克隆创建实例的数据集划分为训练集和测试集，使用训练集训练贝叶斯网络生成预测模型，然后使用测试集评估贝叶斯网络的预测能力。在全属性实验和属性组实验中，对每一个实验系统在数据集上使用十倍交叉验证（10 Cross-Validity[]）评估本文方法。在交叉验证实验中，使用不同系统的克隆创建实例分别作为训练集和测试集。

3.7.2 一致性维护自由实验

本实验对不需要一致性维护的克隆创建实例进行预测评估。将关注不需要一致性维护的克隆实例，从而让程序开发人员重点更加放心的执行复制和粘贴操作，从而快速安全的开发软件。此实验使用三个度量评估预测效果，分别为：推荐率、准确率和召回率。

• 推荐率 (Recommendation Rate)：指的是所推荐的不需要一致性维护的克隆创建实例比例，即预测为不需要一致性维护的克隆创建实例与系统中全部实例的比值。

• 精确率 (Precision)：指所预测的不需要一致性维护复制粘贴实例的准确率，即在预测为不需要一致性维护的复制粘贴实例中，正确预测的实例与所预测实例的比值。

• 召回率 (Recall)：指所推荐的不需要一致性维护的复制粘贴实例的查全率，即预测为不需要一致性维护的复制粘贴实例，与系统中的不需要一致性维护的实例的比值。

3.7.2.1 全属性实验

全属性实验使用全部属性在四个实验系统上进行评估，实验结果如表 3-2 所示。由表中可以看出，本文方法在预测不需要一致性维护的克隆创建实例时，在四个系统上均取得了较好效果。根据预测结果，四个系统的准确率介于 86.6–97.22%

之间，同时召回率也达到较高值，介于 76.97–96.39% 之间。由表 3-2 和表 3-1 中可以看出，本文的推荐率达到了一个合理的水平，和系统中不需要一致性维护的克隆实例比例相差不大。同时，阈值对预测结果的影响不大，准确度会随着阈值的降低而缓慢增大。四个系统在不同的阈值的准确度均较高。因此，本文方法在使用全属性在贝叶斯网络作为分类器时，可以达到一个较好的预测效果。

表 3-2 四个实验系统的全属性组实验效果

Table3-2 The effectiveness for all attribute on four projects

系统	阈值	推荐率 (%)	准确率 (%)	召回率 (%)
ArgoUML	0.01	74.61	95.91	92.85
	0.05	76.83	95.21	94.91
	0.1	77.63	94.83	95.53
	0.15	78.26	94.68	96.15
	0.2	78.47	94.66	96.39
jEdit	0.01	73.93	97.22	81.25
	0.05	78.67	95.98	85.36
	0.1	80.73	95.11	86.79
	0.15	81.67	94.97	87.68
	0.2	82.15	94.62	87.86
jFreeChart	0.01	54.69	92.40	84.50
	0.05	57.22	91.23	87.28
	0.1	59.24	89.72	88.87
	0.15	60.22	89.15	89.77
	0.2	61.14	88.87	90.86
Tuxguitar	0.01	61.51	88.96	76.97
	0.05	66.83	87.96	82.68
	0.1	69.28	87.47	85.24
	0.15	70.82	86.96	86.61
	0.2	72.08	86.60	87.80

3.7.2.2 属性组实验

本章提取了两组度量表示克隆创建实例，分别为代码属性和上下文属性。为确定每一组属性对预测效果的作用，本节对每一属性组进行了评估，即属性组实验。在本实验中每次仅使用一组度量去预测实例的一致性维护需求，并观察其对预测结果的影响。实验结果如表 3-3 所示，其中左侧为代码属性，右侧为上下文属性。

由表 3-3 和表 3-2 中可以看出，代码属性实验的预测效果依然较好，但和全属

性实验对比发现代码属性实验的召回率下降，准确率除 jEdit 中少数几个之外也全部下降，因此本文提取的代码属性对预测作用具有积极意义。在上下文属性实验中，预测结果除 jEdit 系统外准确率提高，但是系统的召回率却大大降低。因此，下上下文属性对系统的准确率影响较大，而代码属性对系统的召回率影响较大，同时两者对于预测都起到了积极的作用。

因此，本文建议在进行预测时，保留所有的属性作为最终的属性，因为某些属性可能对其它尚未验证的系统具有积极的意义。

表 3-3 在四个实验系统上属性组实验结果

Table3-3 The effectiveness for attribute set on four projects

实验系统	阈值	代码属性 (%)			上下文属性 (%)		
		推荐率	精确率	召回率	推荐率	精确率	召回率
ArgoUML	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
	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
jEdit	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
	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
jFreeChart	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
	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
Tuxguitar	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
	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

3.7.2.3 项目交叉实验

在系统开发的初始阶段，系统内可能没有足够的克隆变化实例，从而导致模型训练不完全，进一步使得预测结果不够理想。为解决此问题，本文在不同的系

统上进行了交叉验证实验，即使用已有系统的克隆变化实例训练一致性预测模型，并用于预测其它系统的一致性维护需求。在交叉验证实验中，使用中三个系统的复制粘贴实例作为训练集，然后使用另外一个系统作为测试集测试模型的有效性。在四个系统上的交叉验证实验结果如表 3-4 所示。

从表 3-4 中可以看出，四个系统的准确率和召回率依然达到了较高的水平，其中准确率在 60.01%–91.20% 之间，召回率 56.06%–91.43% 之间。t 同时，通过对比发现 jEdit 的预测效果最好 (准确率较高)，而 jFreeChart 预测效果最差。分析原因可能是由于 JEdit 系统的训练集最大，模型训练最充分，而 jFreeChart 则与之相反。将实验结果与全属性实验（表 3-2 ）对比发现，四个系统的预测效果都大幅下降。因此，预测模型的预测能力会依赖于自身系统数据的训练。鉴于此，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；在自身系统数据较少，不足以较好的训练模型的情况下，可以使用其它系统的数据对模型进行训练。

3.7.3 一致性维护需求实验

在本节中，对需要一致性维护的克隆创建实例进行评估。将关注需要一致性维护的克隆实例，由于其在演化过程中可能会引发一致性变化，因此需要警告程序开发人员谨慎的执行复制和粘贴操作，避免额外的维护代价。实验同样使用三个度量对方法进行评估：警告率、准确率和召回率：

• 警告率 (Warning Rate)：指所警告的需要一致性维护的克隆创建实例，即预测为需要一致性维护的实例与系统中全部实例的比值。这些克隆实例可能会引发一致性变化和额外维护代价。

• 准确率 (Precision)：指警告为需要一致性维护的克隆创建实例的准确率，即在所预测的需要一致性维护的实例中，正确预测的实例与全部警告实例的比值。

• 召回率 (Recall)：指所警告的需要一致性维护的克隆创建实例的召回率，即预测为需要一致性维护的实例与系统中的需要一致性维护实例的比值。

3.7.3.1 全属性实验

全属性实验同样使用全部属性在四个实验系统上进行评估，实验结果如表 3-5所示。由表中可以看出，除 jEdit 外其余系统在不同阈值下取得了可以不错的效果：准确率介于 75.79%–94.94% 之间，召回率介于 57.87% 80.86% 之间。jEdit 的预测效果不够理想，其准确度和召回率仅在 50% 左右。分析其原因可能为 jEdit 中训练数据过少导致模型不完全，数据集仅有 73 复制粘贴实例，仍需要进一步的研究确定。尽管如此，jEdit 的预测结果的准确度依然高于其系统自身的一致性维护需

表 3-4 在四个实验系统下项目交叉实验结果

Table3-4 The effectiveness for cross-project on four projects

测试系统	阈值	推荐率 (%)	精确率 (%)	召回率 (%)
ArgoUML	0.01	59.16	73.03	56.06
	0.05	67.57	75.59	66.28
	0.10	71.95	76.28	71.21
	0.15	74.10	76.40	73.47
	0.20	75.81	76.18	74.94
jEdit	0.01	78.99	91.20	81.43
	0.05	85.62	90.41	87.50
	0.10	87.52	90.25	89.29
	0.15	89.10	89.72	90.36
	0.20	90.21	89.67	91.43
jFreeChart	0.01	74.48	60.07	74.81
	0.05	82.92	60.01	83.21
	0.10	86.51	60.65	87.73
	0.15	88.38	61.01	90.16
	0.20	88.92	60.84	90.46
Tuxguitar	0.01	59.55	75.44	63.19
	0.05	70.40	74.16	73.43
	0.10	74.74	74.91	78.74
	0.15	78.38	74.29	81.89
	0.20	80.76	74.00	84.06

求的比例（11.53%）。因此，在 jEdit 系统上依然提高了预测的精度，也是有效的。最后，对于四个实验系统，本文所构建的模型均具有有十分合理的警告率，警告率十分接近于满足一致性维护需求的克隆变化实例的比例（如表 3-1 中所示）。

虽然阈值的变化可以影响预测的准确率和召回率，但影响并不是十分的剧烈，其中除 jEdit 对精确率的影响要大于对召回率的影响。尽管如此，在不同的阈值下，本文构建模型的准确率依然达到了较高的水平。因此开发人员可以非常自信地依赖于本文模型的预测结果。然而，本文方法的召回率没有达到准确两率的效果，但仍需进一步增强预测模型的召回能力，这需要进一步的深入研究。

表 3-5 四个实验系统全属性的实验效果

Table3-5 The effectiveness for all attribute sets on four projects

实验系统	阈值	警告率 (%)	精确率 (%)	召回率 (%)
ArgoUML	0.9	18.95	94.94	78.46
	0.8	19.64	92.84	79.50
	0.7	20.03	91.93	80.29
	0.6	20.24	91.27	80.55
	0.5	20.54	90.23	80.81
jEdit	0.9	10.27	52.31	46.58
	0.8	12.16	48.05	50.68
	0.7	12.80	46.91	52.05
	0.6	13.59	46.51	54.79
	0.5	14.06	47.19	57.53
jFreeChart	0.9	34.25	91.67	78.12
	0.8	35.03	90.75	79.08
	0.7	35.56	90.31	79.90
	0.6	36.19	89.49	80.56
	0.5	36.57	88.87	80.86
Tuxguitar	0.9	20.08	83.28	57.87
	0.8	21.48	81.76	60.77
	0.7	22.74	79.38	62.47
	0.6	23.30	78.38	63.20
	0.5	24.28	75.79	63.68

3.7.3.2 属性组实验

类似的，对于一致性维护需求实例，表 3-6 给出了属性组实验结果。由表中可以看出，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，

说明代码属性对预测有积极的影响。上下文属性实验中，当仅仅使用上下文属性时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，上下文属性对不同的系统所起到的作用并非一样的。上下文属性和代码属性都具有积极地意义，而上下文属性的影响更大。

表 3-6 在四个实验系统上属性组实验效果

Table3-6 The effectiveness for attribute set on four projects

系统	阈值	代码属性 (%)			上下文属性 (%)		
		警告率	精确率	召回率	警告率	精确率	召回率
ArgoUML	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
	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
jEdit	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
	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
jFreeChart	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
	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
Tuxguitar	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
	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

3.7.3.3 项目交叉实验

与一致性自由实验类似，本节在四个系统对需要一致性维护的克隆创建实例进行了交叉验证实验，实验结果如 3-7所示。从表中可以看出，与全属性实验对比，四个系统的预测效果都极具下降，准确率在 15.36%–61.01% 之间，召回率则更低，仅在 10% 徘徊。与全属性实验（表 3-5 ）对比发现，四个系统的预测效果下降的都十分极为严重。分析原因是克隆创建中大部分的数据是不需要一致性维护需求，

而需要一致性维护的实例数量太少，从而预测模型训练不够完善。另一个可能的原因是本文的预测结果也更依赖于具体的系统，不太适合于使用系统交叉的方式进行预测。如果需要进行此类实验，建议提取和选择全新的度量值。在未来工作中，可以继续探讨此类问题。

表 3-7 四个实验系统上项目交叉实验效果

Table3-7 The effectiveness for cross-project on four projects

测试系统	阈值	警告率 (%)	精确率 (%)	召回率 (%)
ArgoUML	0.90	8.38	15.36	5.61
	0.80	10.30	15.70	7.05
	0.70	12.75	20.19	11.23
	0.60	14.52	18.76	11.88
	0.50	16.38	20.66	14.75
jEdit	0.90	2.37	33.33	6.85
	0.80	3.95	24.00	8.22
	0.70	5.53	20.00	9.59
	0.60	6.64	19.05	10.96
	0.50	6.95	20.45	12.33
jFreeChart	0.90	2.70	50.55	3.40
	0.80	4.72	61.01	7.17
	0.70	6.36	54.67	8.65
	0.60	6.95	53.42	9.24
	0.50	7.46	52.59	9.76
Tuxguitar	0.90	6.02	53.49	11.14
	0.80	7.98	55.26	15.25
	0.70	8.89	52.76	16.22
	0.60	10.15	49.66	17.43
	0.50	11.69	44.31	17.92

3.7.4 讨论

本节从不同的角度评估了所建立的贝叶斯网络模型的预测能力，同时对不需要一致性维护和需要一致性维护的克隆创建实例进行了一致性预测。实验结果表明，本章所构建的模型法在在一致性维护需求和一致性维护自由的实验上均具有高效地预测能力。同时，本文所提取的代码属性和上下文属性对克隆创建实例的一致性维护需求的预测起到了积极的作用。但在不同系统中的两种使用模式下，所

产生的影响程度不一致。使用全属性在贝叶斯网络作为分类器时，可以达到一个较好的预测效果，因此建议对预测模型进行完全训练。

最后，跨项目预测实验中，结果表明本章所提出的预测模型的有效性会依赖于系统自身的某些特征。这意味着使用本章所提取的属性构建一个“通用”预测模型时，可能较为困难的预测某一特定系统的一致性维护需求。因此，本文建议优先选用自身的数据进行模型训练，并对自身系统进行预测。当自身系统的数据太少而不足以训练模型时，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，对需要一致性维护的克隆创建实例，本文不建使用系统交叉的方式进行一致性维护进行预测，可以通过其进行克隆创建的避免。

3.8 本章结论

程序开发人员的复制粘贴操作会向系统中引入克隆代码（称为克隆创建实例），而在其演化过程中，克隆创建实例可能会发生一致性变化而导致额外的维护代价。为帮助程序开发人员避免此类维护代价，本章提出了一个克隆代码创建时一致性维护需求预测方法，在克隆代码创建时预测新创建的克隆代码是否会会引发克隆代码的一致性变化，从而帮助程序开发人员决定是否执行该克隆代码创建操作。通过构建软件系统的克隆家系收集克隆创建实例，并使用其构建和训练克隆一致性预测的贝叶斯网络模型。分别使用不同的属性组表示克隆创建实例，即代码属性表示被复制克隆代码和上下文属性表示被粘贴的克隆代码。对四个开源软件项目进行一致性维护需求和自由实验，从而验证所构建模型的预测能力。实验结果表明本章方法可以以较高的准确率和召回率高效的预测克隆代码的一致性维护需求。此外，所提取的两组属性组在预测中均起到了积极的作用，但是对准确率和召回率可能有不同的影响。同时，交叉验证实验尽管预测效果不如全属性实验，但是在针对克隆创建实例不足的新系统时，仍然可以作为一个不错的选择。

第 4 章 基于贝叶斯网络的克隆变化时一致性维护预测方法

4.1 引言

由于日益增长的软件开发的需求，开发人员在软件开发过程中经常复用既有代码，从而向系统中引入了大量的克隆代码。克隆代码会随着软件系统演化，在其演化过程中，克隆组内的某一克隆代码片段可能会被软件开发员修改而发生变化。由于克隆组内的克隆片段彼此相似，克隆片段变化可能会导致其所在克隆组未来演化过程中的一致性变化。遗忘这种克隆代码的一致性变化，会引入与之相关的克隆一致性缺陷，从而增大系统的维护代价。本文将由克隆代码变化所导致的其所在克隆组未来演化过程中发生的一致性变化，称为克隆代码变化时的一致性维护需求。在克隆代码变化时时预测克隆代码的一致性维护需求，可以帮助避免克隆代码的一致性缺陷，从而帮助提高软件质量和可维护性。

鉴于此，本章与第三章相似，在定义克隆代码变化时的一致性变化和一致性维护需求的基础上，使用机器学习中的贝叶斯网络方法在克隆代码变化时预测其一致性维护需求。为训练克隆代码变化时一致性预测模型，首先通过构建软件系统的克隆家系来收集系统中所有的克隆变化实例（发生变化的克隆代码）。然后，提取了代码属性、上下文属性和演化属性三组属性值表示克隆代码变化实例。最后，使用贝叶斯网络方法训练机器学习模型，并预测克隆代码变化时的一致性维护需求。在四个开源软件系统上对本章方法进行了评估，实验结果表明本章方法以合理的准确率和召回率有效地预测克隆代码变化时的一致性维护需求。本章所提出的预测方法同样可以帮助开发人员在克隆代码变化时预测克隆代码的一致性，避免克隆变化导致的一致性缺陷，从而提高软件质量和可维护性。

4.2 克隆代码变化时一致性维护需求

4.2.1 一致性变化例子

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，但是也会向软件系统中引入大量的克隆代码。有研究表明软件中的克隆代码可以占到系统代码规模的 7%-23%[23]。软件中存在的克隆代码不是静止不变的，会随着软件系统进

行演化，其演化过程可使用克隆家系进行描述 [16]（克隆演化见本文 **??** 节）。本文第二章通过分析克隆演化特征发现，在克隆演化过程中，克隆代码的一致性变化往往多于不一致性变化，因此开发人员需要警惕发生变化的克隆代码，并且当发生变化时需要考虑克隆代码的一致性问题。（具体可参考本文 2.7 节）

演化中的克隆变化使得克隆代码变得越来越难以维护，而克隆代码的一致性问题会导致额外的维护代价。当克隆代码发生变化时，程序开发人员需要验证克隆代码的一致性，从而导致维护代价的增加；如果开发人员忘记验证克隆代码的一致性，则可能会向系统中引入新的克隆一致性缺陷在 [51] [7]。本文第三章研究了克隆代码创建时的一致性维护需求问题，在克隆代码产生时预测克隆代码一致性。在一定程度上可以降低软件的维护代价。但是，值得注意的是，并不是所有的克隆代码都可以被避免。同时，上述方法也无法处理系统中已经存在的克隆代码的一致性变化问题。因此，本章将讨论克隆代码变化时其一致性维护需求问题。

在详细论述本章方法之前，本文先给出一个克隆代码的一致性变化的具体例子，如图 4-1 所示。该一致性变化例子来自于软件系统 jEdit 中。在图 4-1 a)中给出了 3.1.0 版本中的两个克隆代码片段，它们之间是彼此相似的，可称之为克隆代码。在随着系统演化至 3.2.0 版本时，上述两个克隆代码发生了一致性变化。具体地，3.1.0 版本中的第 9-12 行和第 15 行代码在下一版本 3.2.0 中被开发人员修改。第 9-12 行，开发人员将 `selectionEnd` 更改为 0，并删除了一对花括号。在第 15 行，开发人员删除了声明语句。如图 4-1 b) 所示，这两个被修改的代码片段仍然是彼此相似的克隆代码（仍存在于版本 3.2.0 中的同一克隆组中）。假设程序人员没有对上述两个克隆代码进行一致性维护，这必然会导致克隆代码的一致性缺陷，从而降低软件质量。

鉴于此，本章对发生变化的克隆代码进 c 行一致性预测，在克隆代码变化时，预测克隆代码的一致性维护需求。本章在克隆代码演化的基础上结合克隆代码一致性维护，首先给出了一种克隆代码变化时的一致性变化和一致性维护需求的定义，可以帮助预测克隆变化的一致性维护需求。然后，在第二章研究的基础上重新设计了代码属性、上下文属性和演化属性三组属性表示克隆代码的变化实例。最后，使用贝叶斯网络在克隆代码变化时预测其一致性维护需求。本章方法可以帮助程序开发人员避免克隆一致性缺陷，从而降低克隆代码的一致性维护代价。

```
1   public void delete()
2   {
3       if(!buffer.isEditable())
4       {
5           getToolkit().beep();
6           return;
7       }
8
9       if(selectionStart != selectionEnd)
10      {
11          setSelectedText("");
12      }
13      else
14      {
15          int caret = getCaretPosition();
16          if(caret == buffer.getLength())
17          {
18              getToolkit().beep();
19              return;
20          }
21          try
22          {
23              buffer.remove(caret,1);
24          }
25          catch(BadLocationException bl)
26          {
27              Log.log(Log.ERROR,this,bl);
28          }
29      }
30  }
```

```
1   public void backspace()
2   {
3       if(!buffer.isEditable())
4       {
5           getToolkit().beep();
6           return;
7       }
8
9       if(selectionStart != selectionEnd)
10      {
11          setSelectedText("");
12      }
13      else
14      {
15          int caret = getCaretPosition();
16          if(caret == 0)
17          {
18              getToolkit().beep();
19              return;
20          }
21          try
22          {
23              buffer.remove(caret - 1,1);
24          }
25          catch(BadLocationException bl)
26          {
27              Log.log(Log.ERROR,this,bl);
28          }
29      }
30  }
```

a) 版本 3.1.0 中同一一个克隆组中的两个克隆片段

a) Two clone fragments in one clone group in version 3.1.0

```
1   public void delete()
2   {
3       if(!buffer.isEditable())
4       {
5           getToolkit().beep();
6           return;
7       }
8
9       if(selection.size() != 0)
10      {
11          setSelectedText(null);
12      }
13      else
14      {
15          int caret = getCaretPosition();
16          if(caret == buffer.getLength())
17          {
18              getToolkit().beep();
19              return;
20          }
21          try
22          {
23              buffer.remove(caret,1);
24          }
25          catch(BadLocationException bl)
26          {
27              Log.log(Log.ERROR,this,bl);
28          }
29      }
30  }
```

```
1   public void backspace()
2   {
3       if(!buffer.isEditable())
4       {
5           getToolkit().beep();
6           return;
7       }
8
9       if(selection.size() != 0)
10      {
11          setSelectedText("");
12      }
13      else
14      {
15          int caret = getCaretPosition();
16          if(caret == 0)
17          {
18              getToolkit().beep();
19              return;
20          }
21          try
22          {
23              buffer.remove(caret - 1,1);
24          }
25          catch(BadLocationException bl)
26          {
27              Log.log(Log.ERROR,this,bl);
28          }
29      }
30  }
```

b) 版本 3.2.0 中克隆组中的两个克隆片段

b) Two clone fragments in clone group in version 3.2.0

图 4-1 jEdit 中的一致性变化的例子

Fig.4-1 An example of consistent change from jEdit

4.2.2 克隆变化时一致性维护需求定义

在克隆代码的演化过程中，克隆片段可能会被开发人员修改，从而导致克隆代码的一致性变化，这种克隆代码变化可能会导致克隆一致性缺陷。为了描述克隆代码的这种修改，本章给出克隆代码变化时一致性变化定义，如下所示：

定义 4.1 (变化时一致性变化（Changing Consistent Change）) 给定存在两个克隆代码片段 CF_1 和 CF_2，且它们被分别地修改为 CF_1' 和 CF_2'。如果对于一个非常小的阈值 τ，克隆代码 CF_1 和 CF_2 的变化满足以下条件，称此变化为克隆变化时的一致性变化（Changing Consistent Change），

$$\textit{textSim}(CF_i, CF_i') < 1 \qquad \forall i \in \{1,2\} \ (1)$$

$$|\ \textit{textSim}(CF_1, CF_1') \ - \ \textit{textSim}(CF_2, CF_2')\ | \ < \ \tau \ (2)$$

注意 $\textit{textSim}(c, c') = 1$ - UPI(c, c')，UPI 是两个代码片段之间不同的代码行数占总代码行的比例，计算方式同第二章中的计算方式相同。第一个约束条件要求 CF_1 和 CF_2 同时修改；第二个约束条要求 CF_1 和 CF_2 发生满足阈值 τ 条件下一致性的变化，即对代码片段进行了非常相似且一致的变化。本章中 τ 取值为 0.003，要求代码片段之间变化完全一致。

变化时一致性变化，不仅要求克隆代码片段被同时修改，还要求一致的变化。原因在于：在克隆代码变化时，目的是避免克隆变化可能导致的未来演化中的一致性变化，及其因此所引发的克隆一致性缺陷。因此，不仅要求两个克隆代码片段同时变化，还需要发生相似的变化，否则将会引入克隆一致性缺陷。

在克隆演化过程中，克隆片段的变化必然也会导致克隆组的变化，而克隆组的变化可以使用克隆演化模式进行描述，即一致性变化模式。本文给出演化过程中克隆组的一致性变化模式定义，如下所示：

定义 4.2 (变化时一致性变化模式（Changing Consistent Change Pattern）) 在软件版本 $j+1$ 中存在一个克隆组 CG'，假设克隆组内至少存在两个克隆代码片段 CF_1' 和 CF_2' 可以与映射到上一版本 j 的克隆组 CG 中，且 CG 中与之对应的克隆代码片段的 (CF_1, CF_2) 被修改为 (CF_1', CF_2')。如果克隆片段之间的变化（(CF_1, CF_2) 变化至 (CF_1', CF_2') 满足克隆片段的“一致性变化（Consistent Change）”，则称克隆组 CG' 具有一致性变化模式（Consistent Change Pattern）。

变化时的一致性变化模式，与其它论文中使用的定义不同，其原因在于：本章目的在于预测克隆代码的一致性所导致的克隆一致性缺陷。为了便于读者更容

易理解克隆变化时的一致性维护需求，本章现给出克隆变化实例的定义，如下所示：

定义 4.3 (克隆变化实例（Clone Changing Instance）) 克隆变化实例：软件版本 j 中的一个克隆组 CG 是克隆变化实例，如果克隆组 CG 中至少两个克隆代码片段发生变化，且版本 $j+1$ 中至少存在一个克隆组 CG' 与之对应（在同一克隆家系 CGE 中）。

给定一个克隆变化实例，在其未来的演化过程中，可能会导致克隆组的一致性变化模式。克隆组的一致性变化可能会导致一致性缺陷，从而降低软件质量。因此，在克隆代码变化时，预测其未来演化过程中是否会发生一致性变化，可以避免由克隆变化实例导致的一致性缺陷。本文将克隆变化所导致的一致性变化，称为克隆一致性维护需求。克隆变化时的一致性维护需求定义如下：

定义 4.4 (变化时一致性维护需求（Changing Consistency-Requirement）) 给定版本 j 中一个克隆变化实例，CG 满足克隆一致性维护需求（Consistency-Requirement），如果在版本 k 中存在一个克隆实例 CG'（$k > j$）满足以下条件：(1) 在 CG' 中至少存在两个克隆片段在其克隆家系 CGE 中可以映射到克隆实例 CG 中，(2) CG' 具有“一致性变化模式”（Consistent Change Pattern）。反之，假如克隆变化实例 CG 不满足克隆一致性维护需求条件，称该克隆实例不需要一致性维护（consistency-requirement free，或者 consistency-free）。

最终，可以将本章的的研究问题表述如下：给定一个克隆变化实例，即克隆组中 CG 的克隆代码片段 CF 被修改，确定 CG 是否满足克隆变化时一致性维护需求。

更进一步，根据上述定义克隆变化实例只有两种状态：满足和不满足一致性维护需求。我们将变化实例的一致性要求的预测转化为分类问题。本章克隆变化时的克隆一致性需求预测问题可转换为一个典型的分类问题，因此使用机器学习模型解决此分类问题，具体方法见下文。

4.3 克隆变化时一致性维护需求预测框架

为解决本章所提出的克隆一致性需求维护预测问题，本文首先给出了一个方法框架。基于贝叶斯网络的克隆变化时一致性维护需求预测框架架如图 3-1所示。方法可以划分为三个阶段，克隆变化实例收集阶段、克隆变化实例表示阶段和一致性维护需求预测阶段。收集阶段旨在收集系统中全部的克隆变化实例，可将其

用于使用机器学习方法中来训练预测模型。由于实际的克隆变化实例无法直接应用于机器学习方法中，因此在表示步骤中将提取相应的属性值表示克隆变化实例。接下来，在预测步骤，使用属性化的克隆变化实例构建和训练机器学习模型，并使用其预测克隆变化实例的克隆一致性维护需求。

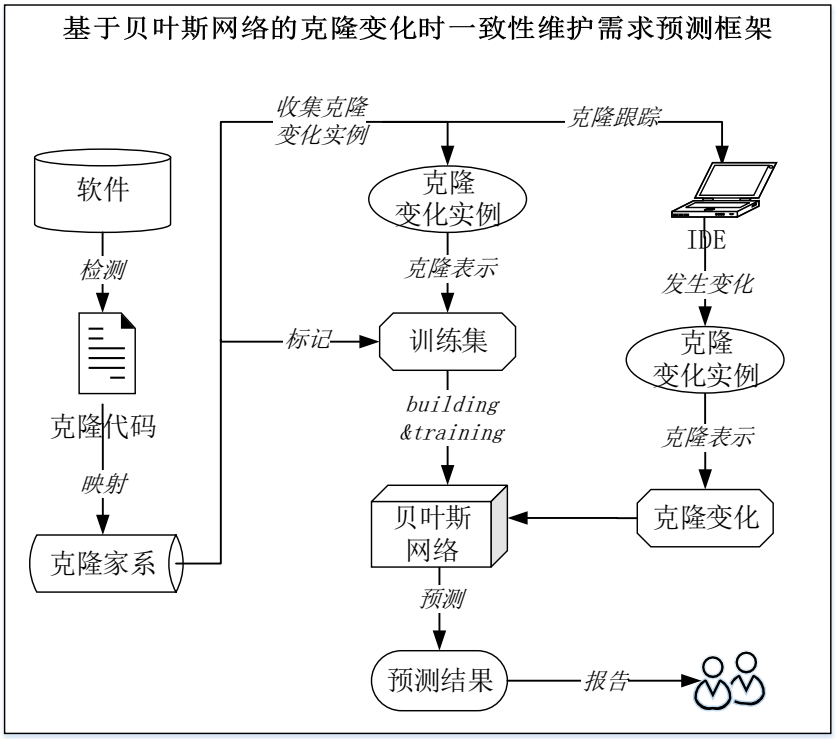


图 4-2 基于贝叶斯网络的克隆变化时一致性维护需求预测框架

Fig.4-2 The framework for clone changing consistency prediction based on Bayesian network

具体来说，在收集阶段中，通过构建系统的克隆家系从软件中收集所有的克隆变化实例。使用 NiCad 来检测软件版本中的所有克隆，并通过在相邻版本的克隆组之间进行映射来构建克隆家系，用于识别克隆变化实例。在表示阶段中，通过提取属性值表示克隆变化实例，提取了代码属性、上下文属性和历史属性三组属性组，同时，还提取了关于克隆变化的相关信息。在预测阶段中，使用收集到的克隆变化实例训练贝叶斯网络，并在克隆变化时预测克隆一致性维护需求。在使用已构建好的模型进行预测时，可将该模型嵌入到软件开发环境中。软件开发环境需要实时跟踪克隆代码的变化（克隆变化实例），然后提取克隆变化实例的度量值。最后，使用模型预测其一致性维护需求，根据预测结果提醒程序开发人员采取进一步的维护操作。

克隆变化实例有两种不同的预测结果，即满足一致性维护需求和不满足维护

需求。对于满足一致性维护需求实例来说，其在将来的演化中可能会引发一致性变化，程序开发人员需要验证克隆组内克隆代码的一致性问题。对于不满足一致性维护需求实例来说，其在将来的演化中不会引发一致性变化，程序开发人员可以放心的对该克隆代码进行修改，不必考虑克隆一致性问题。

本章基于贝叶斯网络预测克隆变化实例的一致性。使用克隆变化实例属性表示贝叶斯网络中的节点，并用于构造贝叶斯网络的结构。使用收集到的克隆变化实例学习贝叶斯网络的参数，从而完成模型的训练，细节可参考本文后续章节。

4.4 克隆变化实例收集

收集克隆变化的的目的在于生成克隆一致性预测的训练集，并将其用于训练机器学习模型。通过构建系统的克隆家系并识别其中的克隆演化模式，可以从软件中收集所有的克隆变化实例。首先使用 NiCad 来检测软件版本中的所有克隆，然后通过在相邻版本的克隆组之间进行映射来构建克隆家系，最后识别克隆一致性演化模式识别系统中的克隆变化实例。

根据定义 4.3，本文假定克隆家系 CGE 中发生变化的克隆组为克隆变化实例。因此通过构建克隆家系并识别克隆家系中发生变化的克隆组，可以收集克隆创建实例。

（1）构建克隆家系。首先，下载系统所有版本的源代码，并使用 NiCad 的默认配置检测检测每一版本的中 Type1-3 的克隆代码。然后，通过映射所有相邻版本的克隆代码，构建系统中全部克隆家系。为完成版本间的映射，为每个克隆片段生成一个克隆区域描述符 CRD[99]，使用基于 CRD 的克隆映射算法映射两个连续版本之间的所有克隆片段和克隆组 [111][112]。根据克隆映射结果，构建系统的克隆家系。

（2）识别克隆演化模式和收集克隆变化实例。首先，识别克隆家系中的克隆演化模式。构建克隆家系后，通过对比相邻版本的克隆代码，可以识别克隆家系的克隆演化模式，尤其是一致性变化模式（参考定义 2.4 和4.2 ）。所识别的克隆演化模式有三个作用：（a）可以用于表示克隆变化实例，本文使用克隆演化模式作为表示克隆变化实例的部分演化属性。因此，克隆演化模式可以用于克隆变化实例表示中。（b）克隆演化模式可以帮助收集克隆变化实例。根据定义 4.1 可以识别系统中发生一致性变化的克隆代码和克隆组，从而确定克隆家系中的克隆变化实例。（c）克隆演化模式可以帮助确认克隆一致性维护需求。根据定义 4.4，一致性维护需求，可以通过遍历克隆家系 CGE 是否发生了一致性变化模式（定义 4.2 ）进行确定。

然后，收集克隆变化实例。在识别克隆家系演化模式，尤其是一致性变化模式后（ 4.2 ），根据定义 4.3通过识别克隆家系中的变化克隆组，便可以收集系统中的克隆变化实例。

（3）标识克隆变化实例的一致性维护需求。在收集所有的克隆变化实例后，还需确认相关实例的一致性需求。根据定义 4.1 和 4.4 ，通过遍历克隆变化实例所在的克隆家系 CGE 的演化情况，确定其一致性维护需求。如果克隆变化实例在其演化过程中发生了一致性变化模式 (4.2)，则该实例满足一致性维护需求，否则不满足维护需求。

4.5 克隆变化实例表示

本章使用贝叶斯网络预测克隆代码变化时的一致性维护需求，并使用软件中既有的克隆变化实例训练贝叶斯网络模型。但是，实际的克隆变化实例无法直接应用于贝叶斯网络中。因此，本文将提取相应的属性值表示克隆变化实例。将分别提取代码属性、上下文属性和演化属性代表克隆变化实例。其中，克隆变化实例的代码属性和上下文属性与本文第三章中克隆创建实例的属性相似，区别在于克隆变化实例将从克隆组的角度提取相应的属性。同时，本章还从演化的角度提取了演化属性表示克隆变化实例。

（1）代码属性

代码属性从代码自身的角度描述了克隆变化实例中克隆代码特征。代码属性描述了克隆代码的词法、语法和函数调用等信息。代码属性主要包括克隆代码粒度、Halstead 属性、结构属性、调用属性等。与第三章中代码属性不同的是，克隆变化实例从克隆组的角度提取相应属性（注意一个克隆变化实体表示一个发生变化的克隆组）。具体的代码属性如下所示：

- 克隆粒度：在克隆变化实例中，所有克隆片段的数量。
- 代码行平均：在克隆变化实例中，所有克隆片段的平均代码行。
- Halstead 属性平均：在克隆变化实例中，所有克隆片段的 Halstead 属性平均，有四个基本的度量值，分别为平均操作符种类、平均操作数种类、平均操作符总量和平均操作数总量。
- 结构属性平均：在克隆变化实例中，所有克隆片段的结构属性属性平均，包括`if_then`, `if\_else`, `switch`, `while`, `do`, `for`, `this\_or\_super` 等。
- 参数访问数量平均：在克隆变化实例中，所有克隆片段的参数访问数量统计的平均。

• 总函数调用次数平均：在克隆变化实例中，所有克隆片段的所有函数调用的次数统计。

• 本地函数调用次数平均：在克隆变化实例中，所有克隆片段的调用函数与被复制克隆片段在相同类的调用次数统计平均。

• 库函数调用次数平均：在克隆变化实例中，所有克隆片段的库函数的调用次数统计平均，包括 java 库函数的调用、eclipse 库函数的调用以及第三方包函数的调用。

• 其它调用次数平均：在克隆变化实例中，所有克隆片段中既不是库函数调用、也不是本地函数调用的其它调用次数统计平均，如同项目内其它包函数调用或同包内其它类中的函数调用。

（2）上下文属性

上下文属性是克隆变化实例中克隆代码之间的关系属性，描述了克隆代码之间的克隆关系。上下文属性包括克隆变化实例中的平均代码相似度、克隆分布、所包含克隆代码之间的一些相似度等等。值得注意的是，上下文属性也是从克隆组的角度进行计算，首先计算组内任意两个克隆代码片段的上下文属性，然后将上下文属性加权平均。在本文所使用的克隆组中，大部分的克隆组仅包含两个克隆代码片段，少部分克隆组包含多个克隆片段。具体的上下文属性如下所示：

• 代码相似度平均：在克隆变化实例中，所有的克隆代码的代码相似度平均，使用和 NiCad 相同的方式计算 [13]。

• 局部克隆标识：克隆变化实例中的克隆片段是否在同一个文件中。

• 文件名相似度平均：在克隆变化实例中，所有的克隆代码的文件的名相似度平均。假定文件名分别为 M_1 和 M_2，则文件名相似度为 $Sim(M_1, M_2)$，采用编辑距离 [115] 计算 (剩余度量中相似度采用相同方法计算)。

• 文件名相似度变量：当克隆变化实例中所有克隆片段是局部克隆时，其文件名相似度为 1，为非局部克隆时为 0。该属性决定文件名相似度是否起效。

• 方法名相似度平均：在克隆变化实例中，所有的克隆代码所在方法的方法名字相似度平均。

• 总参数名相似度平均：在克隆变化实例中，计算任意两个克隆片段的总参数名相似度，然后对所有的相似度进行加权平均。假定两个克隆代码片段所在方法分别为 M 和 N，M 和 N 分别包含 m 和 n 个参数，即 $(P_1, P_2, ..., P_m)$ 和 $(Q_1, Q_2, ..., Q_n)$，则总参数名相似度为 $Sum(Sim(P_i, Q_j))$。

• 最大参数名相似度平均：在克隆变化实例中，计算任意两个克隆片段的最大参数名相似度，然后对所有的相似度进行加权平均。假定两个克隆代码片段所

在方法分别为 M 和 N，M 和 N 分别包含 m 和 n 个参数，即 $(P_1, P_2, ..., P_m)$ 和 $(Q_1, Q_2, ..., Q_n)$，该最大参数名相似度为 $Max(Sim(P_i, Q_j))$。

• 总参数类型相似度平均：在克隆变化实例中，计算任意两个克隆片段的总参数类型相似度，然后对所有的相似度进行加权平均。假定两个克隆代码片段所在方法分别为 M 和 N，M 和 N 分别包含 m 和 n 个参数，其参数类型分别为 $(P_1, P_2, ..., P_m)$ 和 $(Q_1, Q_2, ..., Q_n)$，该参数类型相似度为 $Sum(Sim(P_i, Q_j))$。

• 块信息标识：在克隆变化实例中，所有的克隆代码的上下文信息是否相同，相同为 1，反之为 0。

（3）演化属性

克隆变化实例的最后一组属性是，截止到克隆变化实例发生时，克隆变化实例所在的克隆组的历史演化情况。克隆变化实例的演化属性从克隆家系的角度描述了其所在克隆组的演化情况。演化属性包括，克隆寿命、历史演化模式、上次演化时的演化模式克以及克隆组的历史变化等。（具体的演化属性如下所示：

• 变化实例寿命：截止到克隆变化实例发生时，变化实例所在的克隆组的寿命，即在系统中存在的版本数量。

• 历史演化模式统计：截止到克隆变化实例发生时，变化实例所在的克隆组的历史演化模式统计，包括 7 种克隆演化模式。

• 当前演化模式：截止到克隆变化实例发生时，变化实例所在的克隆组的当前所具有的演化模式，包括 7 种克隆演化模式。

• 历史变化统计：截止到克隆变化实例发生时，变化实例所在的克隆组的所有的历史变化统计。计算方式如下：使用克隆变化实例所在克隆组的代码属性变化进行计算。假设克隆变化实例所在的克隆组 CG_j 存在于软件版本 j 中，统计该克隆组从开始出现到此次变化发生时的每一次代码属性的变化。针对每一个代码属性，相邻两个版本之间的代码属性每增加一次，记录为一次正向变化，反之，记录为一次逆向变化。最后，统计每一个代码属性的所有变化，极为历史变化统计。

（4）变化信息

最后，对于克隆变化实例，本文还收集了与此次变化相关的一些变化信息，用于描述克隆变化本身。计算方式和演化属性中的“历史变化统计”属性中的计算方法一样。

• 克隆变化信息：此次克隆变化发生时，克隆变化实例的变化信息统计。演化属性中的“历史变化统计”属性。

4.6 克隆代码变化时一致性需求预测

本章将克隆代码变化时的一致性维护需求问题，转化成了克隆变化实例的分类问题，即给定一个克隆变化实例，判别其是否满足克隆变化时的一致性维护需求。本文使用贝叶斯网络方法作为机器学习模型，并使用其预测克隆一致性维护需求。因此，本节先简单介绍贝叶斯网络方法。随后，使用属性化的克隆变化实例构建和训练贝叶斯网络模型，并使用训练好贝叶斯网络在克隆代码变化时预测其一致性维护需求。

4.6.1 贝叶斯网络方法

同第二章一样，本章使用贝叶斯网络作为预测方法。贝叶斯网络是一个是一种概率图型，使用已经观察到的事件来预测将来可能发生的事件 [116]。关于贝叶斯网络的信息可以参考本文第三章节 3.6.1 贝叶斯网络。

4.6.2 训练与预测

接下来，使用收集到的克隆变化实例训练贝叶斯网络，并在克隆变化时预测克隆一致性维护需求。

对于每个软件系统，首先，通过收集克隆变化实例并提取相应的属性，用于构建模型训练所需的数据集。然后，调用 WEKA 中的贝叶斯网络实现构建和训练克隆一致性预测模型。

根据定义 4.4，克隆变化实例有两种不同的状态：需要一致性维护和不需要一致性维护。因此在进行一致性维护需求预测时，克隆变化实例也具有两种不同的预测结果：

• 需要一致性维护：若克隆变化实例的预测结果为“需要”，软件开发人员需要检测克隆变化实例所在的克隆组的一致性问题，考虑一致地修改组内其它的克隆代码。因为，该克隆变化实例，在未来演化的过程中可能会引发一致性变化，遗忘这种变化会向系统中引入缺陷，从而降低软件质量。

• 不需要一致性维护：若克隆创建实例的预测结果为“不需要”，软件开发人员可以自由的修改克隆变化实例所在克隆组的克隆片段。因为，该克隆变化实例，在未来演化的过程中不会引发一致性变化，也不会导致一致性缺陷。

在使用已训练好的模型进行预测时，可以与软件开发过程相结合，将该模型嵌入到软件开发环境中，帮助程序开发人员实现边开发边预测克隆变化实例的一致

性维护需求。首先，在软件开发环境中需监测程序员对克隆代码的修改，识别由此产生的克隆变化实例。然后，根据上文描述的代码、上下文和演化属性，提取相应的特征表示该克隆变化实例。最后，使用训练好的预测器预测该克隆变化实例的一致性维护需求，根据预测结果提醒程序开发人员采取进一步的操作。

4.7 实验结果与分析

本节给出本章的实验结果与分析，首先简单介绍了实验所使用的实验系统和评估方法，然后详细给出每个实验的结果与分析。

4.7.1 实验系统与实验设置

为评估本章方法，本章选取了四个开源软件进行实验。四个实验系统的克隆变化实例统计情况如表 4-1 所示。具体来说，第 3 列和第 4 列分别给出了不需要一致性维护和需要一致性维护的克隆变化实例的数量和比例。不需要一致性维护的克隆变化实例，该克隆变化不会使其所在的克隆组在未来演化中发生一致性变化，因此程序人员无需关注其克隆组的一致性问题。需要一致性维护的克隆变化实例，该克隆变化可能使其所在克隆组在未来演化过程中发生一致性变化，而遗忘这种变化会导致克隆缺陷，从而降低软件质量。

从表中 4-1 可以得出两个发现。第一，每个实验系统中都有数百到上千的克隆变化实例，数量从 159 到 1040 个，其中项目 jEdit 是含有最少的克隆变化实例。第二，在这些变化实例中，需要一致性维护的克隆实例比例占相当大的一部分，其比例为 33% 到 74%。其中，ArgoUML 中大部分的克隆变化实例不需要一致性维护。jEdit 和 jFreeChart 需要和不需要一致性维护的克隆变化实例数量相差不大。而 Tuxguitar 中的克隆变化实例大部分都需要一致性维护。尽管需要一致性维护的克隆变化实例远远少于克隆创建实例（表 3-1 ），但是仍在存在数百个需要一致性维护的实例，同时克隆变化实例的危害也更大。因为对克隆变化实例来讲，需要一致性维护的克隆实例比例更大，也更容易导致缺陷。

因为有两种类型的克隆变化实例，即需要一致性维护实例和不需要一致性维护实例。分别对上述两种克隆变化实例进行预测，可将实验划分为“一致性维护需求”实验（需要一致性维护）和“一致性维护自由”实验（不需要一致性维护）。同时，本章使用贝叶斯网络模型预测克隆创建实例的一致性，预测时贝叶斯网络会计算克隆变化实例的概率值，表示该实例需要一致性维护需求的概率 (在 0 1 之间，数值接近于 1 表示需要一致性维护，接近于 0 表示不需要一致性维护)。

表 4-1 实验系统的克隆变化实例信息统计

Table4-1 The statistics for clone creating instances in four projects

实验系统	克隆变化实例的数量（比例）		总数
	不需要一致性维护	需要一致性维护	
ArgoUML	288(67.45%)	139(32.55%)	427
jEdit	78(49.06%)	81(50.94%)	159
jFreeChart	452(43.46%)	588(56.54%)	1040
Tuxguitar	91(25.71%)	263(74.29%)	354

针对不同的克隆变化实例，与第三章类似，也设置了不同的阈值进行实验分析。对于需要一致性维护的实例，其贝叶斯网络预测值较大（接近于 1），选取阈值为 0.5、0.6、0.7、0.8 和 0.9，当预测值大于等于该阈值认定该实例需要一致性维护。对于不需要一致性维护的实例，其预测值较小（接近于 0），选取阈值为 0.01、0.05、0.10、0.15 和 0.20，当预测值小于等于给定阈值时认定该实例不需要一致性维护。

相似的，对上述两个实验中也从三个角度进行全面评估本文方法，将实验进一步的划分为全属性实验、属性组实验和交叉验证实验三个部分。

•

• 全属性实验：使用本章提取的所有属性对实验系统进行分析，评估本文方法的预测能力。

• 属性组实验：分别使用两组不同的属性组对实验系统进行分析，评估所提取的属性组对预测能力的影响。

• 交叉验证实验：使用其它系统数据作为训练集训练模型，并使用该模型在新系统上进行实验分析，从而探索模型在应用到新系统的预测能力。

本文使用使用开源软件工具 WEKA 训练和预测贝叶斯网络。在构建贝叶斯网络时，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 4，使用 SimpleEstimator 来估计贝叶斯网络的条件概率表。在实验时将克隆变化实例的数据集划分为训练集和测试集，使用训练集训练贝叶斯网络生成预测模型，然后使用测试集评估贝叶斯网络的预测能力。在全属性实验和属性组实验中，对每一个实验系统在数据集上使用十倍交叉验证（10 Cross-Validity[]）评估本文方法。在交叉验证实验中，使用不同系统的克隆创建实例分别作为训练集和测试集。

4.7.2 一致性维护需求实验

在本节中，对需要一致性维护的克隆变化实例进行评估。关注需要一致性维护的克隆变化实例，由于该变化可能会导致其所在的克隆组在演化过程中的一致性变化模式，因此需要警告程序开发人员检查克隆组的一致性问题，避免克隆一致性缺陷。实验同样采用三个度量对方法进行评估，即警告率、准确率和召回率：

• 警告率 (Warning Rate)：指所警告的需要一致性维护的克隆变化实例，即预测为需要一致性维护的实例与系统中全部实例的比值。这些克隆变化实例可能会导致一致性缺陷问题，需要确保克隆组的一致性。

• 准确率 (Precision)：指警告为需要一致性维护的克隆变化实例的准确率，即在所预测为需要一致性维护的变化实例中，正确预测的实例与全部警告实例的比值。

• 召回率 (Recall)：指所警告的需要一致性维护的克隆变化实例的召回率，即预测为需要一致性维护的实例与系统中的真实的需要一致性维护实例的比值。

4.7.2.1 全属性组实验

全属性实验使用本章提取的全部属性组在四个实验系统上进行评估，实验结果如表 4-2所示。

所创建的模型在预测 jFreeChart 和 Tuxguitar 的一致性需求时，预测结果显示非常有效，其中预测的准确度和召回率在 80% 左右。同时，jEdit 系统的预测效果虽然不如上面两个系统好，但是也相当不错。原因可能是因为 jEdit 系统的克隆变化实例规模较小，没有对模型实现较好的预测，但这需要进一步研究。尽管 ArgoUML 的预测效果不如其它系统的预测效果，但其准确率仍然可以达到 65% 左右，且召回率在 45% 左右。最后，对于四个实验系统，本文所构建的模型均具有有十分合理的警告率，警告率十分接近于满足一致性维护需求的克隆变化实例的比例（如表 4-1 中所示）。

虽然阈值的变化可以影响预测的准确率和召回率，但影响并不是十分的剧烈，对召回率的影响要比准确率的影响更大。这意味着本文所创建的模型可以提供相当准确的预测效果，但仍需进一步增强预测模型的召回能力。因此，开发人员可以较为自信地依赖于本文的预测结果，但需要其它方法来避免那些预测为不需要一致性维护的错误实例，从而帮助提高模型的预测效果。一个简单简单有效的改进方法是在克隆创建时尽量避免避免那些可能会引发一致性维护的克隆实例，具体细节可参考本文第三章提出的克隆创建时克隆一致性预测方法。

表 4-2 四个实验系统全属性的实验效果

Table4-2 The effectiveness for all attribute sets on four projects

实验系统	阈值	警告率 (%)	精确率 (%)	召回率 (%)
ArgoUML	0.9	20.14	69.77	43.17
	0.8	21.31	68.13	44.60
	0.7	24.12	65.05	48.20
	0.6	24.59	63.81	48.20
	0.5	25.29	62.04	48.20
jEdit	0.9	43.40	73.91	62.96
	0.8	47.17	70.67	65.43
	0.7	49.06	70.51	67.90
	0.6	50.31	70.00	69.14
	0.5	50.94	69.14	69.14
jFreeChart	0.9	52.50	82.05	76.19
	0.8	55.19	81.36	79.42
	0.7	56.83	80.71	81.12
	0.6	58.65	80.33	83.33
	0.5	60.10	79.68	84.69
Tuxguitar	0.9	76.55	80.44	82.89
	0.8	78.53	80.22	84.79
	0.7	80.51	80.00	86.69
	0.6	81.64	79.24	87.07
	0.5	83.33	79.32	88.97

4.7.2.2 属性组实验

本章提取的了三组属性表示克隆变化实例，从不同的角度描述了变化实例的特征。为评估不同的属性组对预测效果的作用，本节进行了属性组实验。表 4-3 中给出了属性组的评估实验结果。其中，分别给出了使用全部属性和删除每一组属性后的实验结果，即“全部属性”、“无代码属性”、“无上下文属性”和“无演化属性”。

从表中可以看出，属性组对预测结果的影响并无一致性结论。但是，代码属性和上下文属性与全属性组对比表明，这两组属性对预测的召回率有显著影响。但是，无演化组属性的实验结果表明其对实验结果的影响没有预想的那么大。尽管如此，这些属性组依然对预测效果有影响，但需要进一步的研究去寻找这种差异背后的真正原因。

此外，本文还对所选用的属性组进行了属性选择实验，使用 WEKA 提供的功能选取了最佳属性集。然而，通过比对全属性组的实验结果，发现发现全属性组实验结果要优于字属性组实验结果，所采用的属性具有积极的意义。虽然一些属性组可能对实验结果没有显着的影响，但是其也不具有消极的影响。本文建议在进行一致性预测时，需要保留全部属性组，因为在应用到其它系统是可能会起到积极的左右。

4.7.2.3 项目交叉实验

在系统开发的初始阶段，系统内可能没有足够的克隆变化实例，从而导致模型训练不完全，进一步使得预测结果不够理想。为解决此问题，本文在不同的系统上进行了交叉验证实验，即使用已有系统的克隆变化实例训练一致性预测模型，并用于预测其它系统的一致性维护需求。在此实验中，使用中三个系统的变化实例作为训练集，然后使用另外一个系统作为测试集测试模型的有效性。

实验结果如表 4-4 所示。从表中可以看出，与全属性实验相比，交叉验证实验的预测效果也大大下降，准确率和召回率都下降。因此，本文建议使用本文的预测模型预测自身系统的克隆变化实例。如果在软件开发的初始阶段，项目可能没有足够的克隆变化实例，本文建议使用第三章提出的方法在创建克隆代码时检测克隆代码的一致性问题。当随着软件演化以后，系统自身的变化实例将会变得更多，此时可以使用自身数据训练模型并预测克隆代码的一致性。

表 4-3 在四个实验系统上属性组实验效果

Table4-3 The effectiveness for attribute set on four projects

实验系统	阈值	全部属性 (%)			无代码属性 (%)			无上下文属性 (%)			无演化属性 (%)		
		警告率	准确率	召回率	警告率	准确率	召回率	警告率	准确率	召回率	警告率	准确率	召回率
ArgoUML	0.9	20.14	69.77	43.17	14.29	73.77	32.37	17.56	72.00	38.85	21.78	69.89	46.76
	0.8	21.31	68.13	44.60	20.37	70.11	43.88	19.91	69.41	42.45	24.12	66.02	48.92
	0.7	24.12	65.05	48.20	23.89	65.69	48.20	21.78	64.52	43.17	26.00	63.96	51.08
	0.6	24.59	63.81	48.20	25.76	63.64	50.36	24.12	63.11	46.76	28.10	60.00	51.80
	0.5	25.29	62.04	48.20	29.27	61.60	55.40	26.00	59.46	47.48	29.51	60.32	54.68
jEdit	0.9	43.40	73.91	62.96	38.99	75.81	58.02	40.88	75.38	60.49	44.03	71.43	61.73
	0.8	47.17	70.67	65.43	43.40	72.46	61.73	46.54	70.27	64.20	47.80	69.74	65.43
	0.7	49.06	70.51	67.90	47.17	70.67	65.43	47.80	71.05	66.67	49.69	68.35	66.67
	0.6	50.31	70.00	69.14	48.43	71.43	67.90	47.80	71.05	66.67	51.57	65.85	66.67
	0.5	50.94	69.14	69.14	53.46	69.41	72.84	50.94	67.90	67.90	52.83	65.48	67.90
jFreeChart	0.9	52.50	82.05	76.19	44.42	84.63	66.50	51.06	81.92	73.98	50.19	81.42	72.28
	0.8	55.19	81.36	79.42	49.42	82.10	71.77	54.13	80.99	77.55	53.56	80.07	75.85
	0.7	56.83	80.71	81.12	53.17	80.83	76.02	56.92	79.39	79.93	56.06	78.39	77.72
	0.6	58.65	80.33	83.33	55.10	80.10	78.06	59.33	78.12	81.97	57.50	77.93	79.25
	0.5	60.10	79.68	84.69	57.79	78.20	79.93	60.48	78.06	83.50	59.13	77.56	81.12
Tuxguitar	0.9	76.55	80.44	82.89	68.64	79.01	73.00	75.14	78.57	79.47	71.75	80.71	77.95
	0.8	78.53	80.22	84.79	74.01	79.01	78.71	78.53	78.42	82.89	74.01	80.92	80.61
	0.7	80.51	80.00	86.69	78.25	78.34	82.51	80.79	77.97	84.79	76.55	80.81	83.27
	0.6	81.64	79.24	87.07	79.94	78.09	84.03	82.20	77.32	85.55	78.53	80.94	85.55
	0.5	83.33	79.32	88.97	83.33	77.63	87.07	84.18	76.85	87.07	79.66	80.50	86.31

表 4-4 四个实验系统上项目交叉实验效果

Table4-4 The effectiveness for cross-project on four projects

测试系统	阈值	警告率 (%)	精确率 (%)	召回率 (%)
ArgoUML	0.9	40.28	33.72	41.73
	0.8	49.88	35.21	53.96
	0.7	57.85	34.01	60.43
	0.6	62.06	34.34	65.47
	0.5	65.34	34.41	69.06
jEdit	0.9	26.42	59.52	30.86
	0.8	31.45	54.00	33.33
	0.7	37.11	54.24	39.51
	0.6	42.14	56.72	46.91
	0.5	45.91	56.16	50.62
jFreeChart	0.9	21.15	62.73	23.47
	0.8	26.44	62.55	29.25
	0.7	31.44	60.86	33.84
	0.6	35.38	60.60	37.93
	0.5	38.65	59.95	40.99
Tuxguitar	0.9	18.08	70.31	17.11
	0.8	22.60	70.00	21.29
	0.7	26.27	70.97	25.10
	0.6	31.07	71.82	30.04
	0.5	35.88	72.44	34.98

4.7.3 一致性维护自由实验

本实验对不需要一致性维护的克隆变化实例进行预测评估。将关注不需要一致性维护的克隆实例，从而让程序开发人员可以更加自由的修改克隆代码，而不需要考虑克隆变化所引发的一致性维护问题，可以更加快速有效的开发系统。此实验使用三个度量评估预测效果，分别为：推荐率、准确率和召回率。在本实验中，我们计算以下三个指标，用于评估我们的模型的一致性无需求的可预测性：

• 推荐率 (Recommendation Rate)：指的是所推荐的不需要一致性维护的克隆变化实例与所有实例的比例，即预测为不需要一致性维护的克隆变化实例与系统中全部变化实例的比值。

• 精确率 (Precision)：指所预测的不需要一致性维护克隆变化实例的准确率，即在预测为不需要一致性维护的克隆变化实例中，正确预测的实例与所预测实例的比值。

• 召回率 (Recall)：指所推荐的不需要一致性维护的克隆变化实例的查全率，即预测为不需要一致性维护的变化实例，与系统中的不需要一致性维护的实例的比值。

4.7.3.1 全属性组实验

全属性实验使用全部属性在四个实验系统上进行评估，实验结果如表 4-5 所示。由表中可以看出，本文方法在预测不需要一致性维护的克隆变化实例时，在四个系统上均取得了不错的预测效果。其中，在系统 ArgoUML 和 jFreeChart 取得了较好的效果，精确率在 80% 左右，并且召回率在 70% 左右。在系统 jEdit 的预测效果尽管没有上面两个系统的预测效果好，但也具有不错的预测效果。然而不幸的是，本文的方法在系统 Tuxguitar 上预测效果不够理想，准确率在 50% 左右，召回率更低。可能是由于 Tuxguitar 的不满足一致性维护需求的克隆变化实例太少，没有对预测模型训练完全，但是仍需进一步的研究确认。同时，本文所构建的模型在四个系统上都具有十分合理的推荐率，推荐率和系统本身的不需要一致性维护的克隆变化的比例相差不大（如表 4-1 中所列）。

不同的阈值会影响到准确率和召回率，但是阈值对于召回率的影响要大于大于准确率的影响。这意味着本文所构建的模型可以提高比较稳定的预测效果（准确率差不大），但系统的召回率仍需要进一步的提升。简而言之，本文的方法所构建的预测模型可以对不需要一致性维护的克隆变化实例提供一个相对稳定的预测，并且具有相当好的准确度和召回率，软件开发人员可以据此对克隆变化实例进行有效的预测。

表 4-5 四个实验系统的全属性组实验效果

Table4-5 The effectiveness for all attribute on four projects

系统	阈值	推荐率 (%)	准确率 (%)	召回率 (%)
ArgoUML	0.01	51.99	83.33	64.24
	0.05	60.42	82.95	74.31
	0.1	63.93	81.32	77.08
	0.15	65.81	80.43	78.47
	0.2	67.45	80.21	80.21
jEdit	0.01	20.13	84.38	34.62
	0.05	30.19	79.17	48.72
	0.1	33.96	74.07	51.28
	0.15	38.36	70.49	55.13
	0.2	39.62	69.84	56.41
jFreeChart	0.01	28.65	84.56	55.75
	0.05	32.50	83.73	62.61
	0.1	34.23	82.02	64.60
	0.15	35.58	80.81	66.15
	0.2	36.06	80.53	66.81
Tuxguitar	0.01	7.91	53.57	16.48
	0.05	9.32	57.58	20.88
	0.1	9.89	54.29	20.88
	0.15	12.15	48.84	23.08
	0.2	13.28	48.94	25.27

4.7.3.2 属性组实验

同样，对不需要一致性维护的克隆变化实例，本文同样进行了属性组实验，以评估不同的属性组对预测效果的影响。实验结果如表 4-6 所示，其中，分别给出了使用全部属性和删除每一组属性后的实验结果，即“全部属性”、“无代码属性”、“无上下文属性”和“无演化属性”。

从表中可以看出，属性组对“一致性维护自由”实验的预测结果的影响并无一致性结论。但是，代码属性和上下文属性与全属性组对比表明，这两组属性对预测的召回率有显著影响。但是，无演化组属性的实验结果表明其对实验结果的影响没有预想的那么大。尽管如此，这些属性组依然对预测效果有影响，但需要进一步的研究去寻找这种差异背后的真正原因。

此外，本文还对所选用的属性组进行了属性选择实验，使用 WEKA 提供的功能选取了最佳属性集。然而，通过比对全属性组的实验结果，发现发现全属性组实验结果要好于最佳属性子集的实验结果，因此，所采用的属性具有积极的意义。虽然一些属性组可能对实验结果没有显着的影响，但是其也不具有消极的影响。

尽管一些属性组可能对实验结果没有极为重要的贡献，但是属性组同样也没有产生消极的影响。因此，本文建议将所有属性保留在构建预测模型中，因为某些属性可能对一些未经实验验证的系统具有重要且积极的影响。

4.7.3.3 交叉验证实验

在这个实验中，我们探讨使用跨项目在预测不需要一致性维护的克隆变化实例。同样，使用其它软件系统的克隆变化实例作为训练集，并使用另外系统作为测试集。实验结果如表 4-7 所示。

从表中可以看出，所有的准确率和召回率都受到了严重的影响。结合全属性实验的实验结果（表 4-5 ），可以发现当使用自身系统的数据作为训练集且训练集足够大时，本文的预测模型有效的，例如软件系统经过长时间的演化具有了大量的克隆变化实例。因此，本文建议使用系统自身的数据进行一致性预测。然而，在软件开发的初期阶段，由于缺乏足够数量的克隆变化实例，不能很好的训练模型时，开发人员可以使用本文第三章提出的方法在克隆创建时预测克隆代码的一致性，从而仅允许那些不需要一致性维护的克隆实例产生。在软件演化到了足够的版本时，随着来自其自己的克隆变化数据的增加，开发人员可以使用自身数据重新训练模型以预测项目中克隆变化的一致性。

表 4-6 在四个实验系统上属性组实验效果

Table4-6 The effectiveness for attribute set on four projects

实验系统	阈值	全部属性 (%)			无代码属性 (%)			无上下文属性 (%)			无演化属性 (%)		
		警告率	准确率	召回率	警告率	准确率	召回率	警告率	准确率	召回率	警告率	准确率	召回率
ArgoUML	0.01	51.99	83.33	64.24	36.77	89.17	48.61	48.01	83.41	59.38	44.03	85.64	55.90
	0.05	60.42	82.95	74.31	49.18	85.71	62.50	59.48	79.92	70.49	51.99	84.23	64.93
	0.1	63.93	81.32	77.08	56.67	83.06	69.79	63.00	79.18	73.96	58.31	83.53	72.22
	0.15	65.81	80.43	78.47	60.42	83.72	75.00	65.11	78.78	76.04	60.66	83.01	74.65
	0.2	67.45	80.21	80.21	62.53	82.77	76.74	65.81	78.65	76.74	62.76	83.21	77.43
jEdit	0.01	20.13	84.38	34.62	18.87	80.00	30.77	22.64	80.56	37.18	23.27	81.08	38.46
	0.05	30.19	79.17	48.72	27.04	79.07	43.59	30.19	77.08	47.44	27.04	79.07	43.59
	0.1	33.96	74.07	51.28	30.82	79.59	50.00	33.96	74.07	51.28	28.93	78.26	46.15
	0.15	38.36	70.49	55.13	33.96	77.78	53.85	36.48	70.69	52.56	32.08	76.47	50.00
	0.2	39.62	69.84	56.41	37.11	76.27	57.69	38.36	70.49	55.13	32.70	76.92	51.28
jFreeChart	0.01	28.65	84.56	55.75	18.94	89.34	38.94	23.75	84.62	46.24	26.73	84.53	51.99
	0.05	32.50	83.73	62.61	24.71	84.44	48.01	28.65	81.54	53.76	31.35	82.21	59.29
	0.1	34.23	82.02	64.60	28.75	82.61	54.65	31.44	81.35	58.85	32.79	80.94	61.06
	0.15	35.58	80.81	66.15	30.77	81.56	57.74	32.88	81.58	61.73	34.62	78.89	62.83
	0.2	36.06	80.53	66.81	32.98	80.17	60.84	34.13	79.72	62.61	35.87	78.02	64.38
Tuxguitar	0.01	7.91	53.57	16.48	3.11	27.27	3.30	5.37	47.37	9.89	9.89	57.14	21.98
	0.05	9.32	57.58	20.88	8.19	41.38	13.19	8.19	48.28	15.38	11.58	51.22	23.08
	0.1	9.89	54.29	20.88	9.89	48.57	18.68	9.89	48.57	18.68	13.28	48.94	25.27
	0.15	12.15	48.84	23.08	11.02	46.15	19.78	10.73	44.74	18.68	14.97	52.83	30.77
	0.2	13.28	48.94	25.27	11.86	45.24	20.88	11.86	42.86	19.78	15.82	55.36	34.07

表 4-7 四个实验系统上项目交叉实验效果

Table4-7 The effectiveness for cross-project on four projects

测试系统	阈值	警告率 (%)	精确率 (%)	召回率 (%)
ArgoUML	0.01	6.09	73.08	6.60
	0.05	13.11	76.79	14.93
	0.1	16.63	74.65	18.40
	0.15	20.84	75.28	23.26
	0.2	23.19	71.72	24.65
jEdit	0.01	9.43	60.00	11.54
	0.05	21.38	50.00	21.79
	0.1	31.45	44.00	28.21
	0.15	37.74	50.00	38.46
	0.2	42.14	49.25	42.31
jFreeChart	0.01	26.44	46.91	28.54
	0.05	37.60	46.55	40.27
	0.1	43.27	47.56	47.35
	0.15	46.92	46.72	50.44
	0.2	49.33	46.39	52.65
Tuxguitar	0.01	16.67	27.12	17.58
	0.05	33.62	24.37	31.87
	0.1	42.37	26.67	43.96
	0.15	46.61	26.67	48.35
	0.2	49.72	26.14	50.55

4.7.4 讨论

本节从不同的角度评估了所建立的贝叶斯网络模型的预测能力，同时对不需要一致性维护和需要一致性维护的克隆变化实例进行了一致性预测。实验结果表明，当系统使用自身的克隆变化实例进行全属性预测时，本章的模型在一致性维护需求和一致性维护自由的实验上均具有有效的预测能力。同时，属性组实验中，发现每一个属性组在不同角度和程度上对预测模型的效果有着积极的作用。因此，本文鼓励开发人员使用全属性组进行克隆变化实例的一致性维护需求预测，从而将其适用到其它系统是达到最佳的预测效果。

然而，跨项目的实验表明，本章所提出的预测模型的有效性会依赖于系统自身的某些特征。这意味着使用本章所提取的属性构建一个“通用”预测模型时，可能较为困难的预测所某一特定系统的一致性维护需求。换句话说，本文建议开发人员为不同的系统构建和训练不同的模型，已达到满意的预测效果。试图构建一个全新的预测器，并可以适用于其它的项目中也并非完全不可取的。在软件缺陷预测领域中存在一些跨项目预测的研究，可以考虑对其进行进一步的研究使之运用到跨项目的克隆一致性需求维护中。

在软件开发初期，系统自身缺乏克隆变化实例而无法构建预测模型，同时跨项目的模型也无法对其进行预测。为了解决此问题，本文建议建议开发人员在软件开发的早期阶段使用本文第二章提出的克隆代码创建时克隆一致性维护需求预测方法，尽量避免会导致一致性变化的克隆代码，或者仅仅允许不会导致一致性变化的克隆代码的产生。

4.8 本章结论

在软件演化过程中，发生变化的克隆代码可能会导致一致性变化而引发克隆一致性缺陷，从而增加了软件维护的代价。为了帮助程序开发人员避免克隆一致性缺陷，降低软件维护代价，本章提出了一个克隆代码变化时一致性维护需求预测方法，在克隆代码发生变化时预测该变化是否会导致克隆一致性缺陷，从而提醒程序开发人员对克隆代码进行一致性维护。通过构建软件系统的克隆家系收集系统中的克隆变化实例，并用于构建和训练克隆代码一致性预测的贝叶斯网络模型。同时，为了表示克隆变化实例，本章从克隆组的角度提取了三组属性描述克隆变化实例的特征，分别代码属性、上下文属性和演化属性。对四个开源软件项目进行了一致性维护需求和自由实验，从而验证所构建模型的预测能力。实验结果表明，

实验结果表明本章方法可以以合理的准确率和召回率有效地预测克隆代码的一致性维护需求。此外，提取的三组属性对预测效果起到了积极的作用，并对预测结果的召回率有较强的影响。同时，对于克隆变化实例不足的系统，跨项目所构建的模型还不能够达到让人满意的预测效果，因此建议采用第二章提出的方法在克隆代码创建时预测其一致性。

第 5 章 克隆代码一致性维护需求预测实证研究

5.1 引言

克隆代码在随着软件演化的过程中，可能会被程序开发人员修改而引发克隆代码的一致性问题。为解决此问题，本文在第三章和第四章分别在克隆代码创建时和变化时，对克隆代码的一致性维护需求进行了预测。但是，上述方法中仅考虑了贝叶斯网络方法，能否将其它机器学习方法（如支持向量机、决策树等）应用到此克隆需求预测中是一个值得研究的问题。此外，上述预测也没有和软件开发过程相结合，在开发过程中预测克隆一致性需求，可以切实的帮助程序开发人员避免克隆代码导致的额外维护代价和克隆一致性缺陷，从而帮助提高软件质量和可维护性。

鉴于此，在第三章和第四章研究的基础上，本章统一了克隆代码创建时和变化时的克隆代码一致性变化和维护需求定义，并使用五种不同的机器学习方法进行克隆一致性维护需求预测实证研究，并与软件开发过程相结合预测克隆一致性需求。首先通过构建软件系统的克隆家系来收集系统中所有的克隆实例（克隆创建实例和克隆变化实例），并提取不同的属性值表示不同的克隆实例。然后，使用五种不同的机器学习方法，分别在克隆创建时和变化时预测克隆代码的一致性维护需求。最后，结合软件开发过程，设计并实现了克隆代码一致性维护需求预测插件（CCRP），并可以嵌入到集成开发环境 eclipse 中预测克隆代码一致性需求。本章在四个开源软件系统上进行了实证研究，实验结果表明本文所提取的属性值可以适用于不同的机器学习方法上，且支持向量机方法更适合于克隆代码的一致性维护需求预测中。本章基于 eclipse 所实现的插件，可以帮助开发人员在软件开发过程中预测克隆代码的一致性，降低克隆代码导致的额外的维护代价，避免克隆变化导致的一致性缺陷，从而提高软件质量和可维护性。

5.2 克隆代码一致性维护需求

5.2.1 研究问题

在软件开发过程中，通过复制粘贴操作复用既有代码已经成为一种常见的软件开发手段，但是也会向软件系统中引入大量的克隆代码。这些新创建的克隆代码以及系统已经存在的克隆代码并不是静止不变的，会随着软件系统演化。在演

化过程中，克隆片段片段可能会被软件开发员修改而发生变化，并可能进一步引发克隆组的一致性变化。克隆代码的一致性变化问题会影响软件质量和可维护性，原因在于：程序开发人员需要对发生变化的克隆进行一致性维护，会导致额外的一致性维护代价。而遗忘克隆的一致性变化，更会会导致克隆不一致缺陷，从而进一步增加软件的维护代价。

为了解决此问题，本文的第三章和第四章分别在克隆代码创建时和变化时，基于贝叶斯网络对克隆代码的一致性维护需求进行了预测，并取得了不错的预测效果。上述方法中仅仅使用了贝叶斯网络作为预测模型，但是在机器学习领域中仍然存在着其它的机器学习方法。因此，这启发了本章对克隆一致性预测的深入研究。首先，能否将克隆代码的一致性维护需求预测应用到其它的机器学习方法中。然后，在这些机器学习方法中，能否为开发人员确定一种通用的机器学习方法，可以同时应用于克隆创建时和克隆变化时的一致性需求预测中。最后，对于两个不同的预测时刻（创建时和变化时），如何在软件开发过程帮助程序开发人员选择合适的预测时间，并达到最好的预测效果。更为重要的是，上述克隆一致性维护预测研究尚未与软件开发过程相结合，不能帮助程序开发人员在开发时预测克隆代码的一致性。

为了更好地帮助软件开发人员维护克隆代码，本章将结合其它机器学习方法和软件开发过程，对克隆代码进行一致性维护需求预测的实证研究。具体地，本章地研究问题如下：

Research Problem：克隆代码的一致性维护需求预测是否可以应用于其它的机器学习方法中，软件开发人员应如何结合软件开发过程中执行克隆一致性需求预测？

为了解决本章的研究问题，将使用五种不同机器学习方法来预测克隆代码的一致性维护需求，并且分别在克隆代码创建时和克隆代码变化时进行预测。同时，在预测克隆一致性的过程中充分考虑并结合软件开发过程，比较和讨论两种不同预测的时间，从而帮助开发人员实际开发环境中执行克隆代码一致性预测，具体地，本文章研究问题可以细分为以下三个子问题，如下所示：

RQ1： 在预测克隆代码创建时的一致性维护需求时，哪些机器学习方法可以用于该预测中，且所提取得度量值能否应用于其他的机器学习方法中，不同的机器学习方法预测效果是否一致，哪一种机器学习的方法能够取得最好的结果？

RQ2： 在预测克隆代码变化时的一致性维护需求时，哪些机器学习方法可以用于该预测中，且所提取得度量值能否应用于其他的机器学习方法中，不同的机器

学习方法预测效果是否一致，哪一种机器学习的方法能够取得最好的结果？

RQ3： 在实际开发过程中，程序开发人员应该如何结合软件开发过程选择合适的机器学习模型和预测时间，并对克隆代码进行一致性维护需求进行预测，从而都够达到最佳的预测效果？

鉴于此，本章基于不同的机器学习方法对克隆代码的一致性维护需求预测进行了一个实证研究，同时在克隆代码创建时和变化时预测克隆代码的一致性，并结合软件开发过程帮助程序开发人员选择合适和机器学习模型和预测时间已达到最佳的预测效果。首先，统一了克隆代码创建时和变化时的一致性变化及其一致性维护需求定义，可以在一个框架下预测克隆代码的一致性维护需求。然后，充分考虑了机器学习领域中五种不同的机器学习方法，并将其应用到克隆代码的一致性维护需求中。最后，本章结合软件开发过程，将克隆一致性维护预测方法嵌入到软件开发环境（eclipse）中，帮助程序开发人员实现边开发、边预测、边维护克隆代码，从而可以避免克隆一致性缺陷，并降低克隆代码的一致性维护代价。

5.2.2 克隆一致性维护需求定义

在克隆代码的整个演化周期中，克隆片段可能会被开发人员修改，从而导致克隆代码的一致性变化。在本文的第三章和第四章分别提供了两种不同形式的克隆代码一致性变化定义，从而适应于不同时间的克隆一致性维护需求预测中。本章将统一第三章和第四章的定义，以应用于本章地实证研究中。具体来说，克隆代码的一致性变化如下：

定义 5.1 (**一致性变化**（Consistent Change）) 给定两个克隆代码片段 CF_1 和 CF_2，且它们被分别地修改为 CF_1' 和 CF_2'。如果对于一个非常小的阈值 τ，如果克隆代码 CF_1 和 CF_2 的变化满足以下条件，称此变化为一致性变化（Consistent Change）,

$$\textit{textSim}(CF_i, CF_i') < 1 \qquad \forall i \in \{1,2\} \ (1)$$

$$|\ \textit{textSim}(CF_1, CF_1') \ - \ \textit{textSim}(CF_2, CF_2')\ | \ < \ \tau \ (2)$$

更具体地，如果克隆变化仅满足条件 1，将其称为 Type-1 一致性变化（Type-1 Consistent Change）；如果克隆变化同时满足条件 1 和条件 2，将其称为 Type-1 一致性变化（Type-2 Consistent Change）。

定义中克隆代码 CF_1 和 CF_2 的变化情况由相似性度量 *textSim* 进行定义，$textSim$ 与第三章和第四章计算方式相同。定义中的两个约束条件共同定义了克隆

代码的一致性变化，约束条件 1 确保了克隆代码片段同时被修改，约束条件 2 克隆代码片段发生了一致性的变化，由变化阈值 τ 指定。

其中，Type-1 一致性变化又可以称为克隆创建时一致性变化，Type-2 一致性变化又称为克隆变化时的一致性变化。这两种不同的一致性变化，将分别应用于两种不同时间的克隆一致性预测中。在克隆代码创建时，目标是避免新创建的克隆代码在其未来演化过程中的一致性变化，及其所导致额外的维护代价。所以，只要两个克隆片段同时变化，即认为会导致额外维护代价。因此，在克隆代码创建时使用使用 Type-1 一致性变化进行预测。在克隆代码变化时，目的是避免克隆变化可能导致的未来演化中的一致性变化，及其因此所引发的克隆一致性缺陷。所以，不仅要求两个克隆代码片段同时变化，还需要发生相似的变化，否则将会引入克隆一致性缺陷，因此，在克隆代码变化时使用使用 Type-2 一致性变化进行预测。

在克隆演化过程中，克隆片段是以克隆组的形式出现在软件系统中。克隆代码的变化情况必然会导致克隆组的变化，而克隆组的变化使用克隆演化模式进行描述，即一致性变化模式。本文给出演化过程中克隆组的一致性变化模式定义，如下所示：

定义 5.2 (**一致性变化模式**（Consistent Change Pattern）) 在软件版本 $j+1$ 中存在一个克隆组 CG'，假设克隆组内至少存在两个克隆代码片段 CF_1' 和 CF_2' 可以与映射到上一版本 j 的克隆组 CG 中，且 CG 中与之对应的克隆代码片段的 (CF_1, CF_2) 被修改为 (CF_1', CF_2')。如果克隆片段之间的变化（(CF_1, CF_2) 变化至 (CF_1', CF_2') 满足克隆片段的“一致性变化（Consistent Change）”，则称克隆组 CG' 具有一致性变化模式（Consistent Change Pattern）。

其中“一致性变化”为 Type-1 或 Type-2 一致性变化，相应的克隆组变化模式则为 Type-1 和 Type-2 一致性变化模式。在相邻的两个软件版本中，克隆组一致性变化模式可以描述克隆演化中由于克隆片段变化所引发的克隆组的变化情况。

回顾本章地研究内容是要在两种不同的时刻预测克隆代码的一致性维护需求，结合第三章和第四章的克隆实例的定义，这里将统克隆创建实例和克隆变化实例为克隆代码实例，如下所示：

定义 5.3 (**克隆实例**（Clone Instance）) 克隆创建实例：软件版本 j 中的一个克隆组 CG 是克隆创建实例，如果该克隆组 CG 是其克隆家系 CGE 的根节点。克隆变化实例：软件版本 j 中的一个克隆组 CG 是克隆变化实例，如果克隆组 CG 中至少两个克隆片段发生变化且版本 $j+1$ 中至少存在一个克隆组 CG' 与之对应（在同一克

隆家系 CGE 中）。克隆实例：将克隆创建实例和克隆变化实例统称为克隆实例。

克隆实例在演化过程中可能会引发的克隆一致性变化，如果不能确保克隆的一致性，将会导致导致额外的维护代价和一致性缺陷。具体来说，对于克隆创建实例，本文认为 Type-1 一致性变化及其演化模式在克隆演化过程中可能会导致额外的克隆维护代价。对于克隆变化实例，本文认为 Type-2 一致性变化及其演化模式在克隆演化过程可能会导致克隆一致性缺陷。因此，本章尝试在不同的时间预测克隆实例的一致性维护需求，以避免额外的克隆维护代价和一致性缺陷。本章结合第三章和第四章的克隆代码一致性维护需求的定义，将克隆创建时和变化时的一致性维护需求统一为克隆一致性维护需求定义，如下所示：

定义 5.4 (克隆一致性维护需求（Clone Consistency-Requirement）) 给定版本 j 中一个克隆实例，CG 满足克隆一致性维护需求（Consistency-Requirement），如果在版本 k 中存在一个克隆实例 CG'（$k > j$）满足以下条件：(1) 在 CG' 中至少存在两个克隆片段在其克隆家系 CGE 中可以映射到克隆实例 CG 中，(2) CG' 具有“一致性变化模式”（Consistent Change Pattern）。反之，假如克隆实例 CG 不满足克隆一致性维护需求条件，称该克隆实例不需要一致性维护（consistency-requirement free，或者 consistency-free）。

其中，“一致性变化模式”为 Type-1 或 Type-2 一致性变化模式，相应的克隆一致性需求为 Type-1 和 Type-2 一致性维护需求。其中，Type-1 克隆一致性维护需求可以用于预测克隆创建实例，Type-1 克隆一致性维护需求可以用于预测克隆变化实例。

最终，克隆一致性维护需求预测任务可以转化为以下问题：给定一个克隆实例，克隆创建实例或者克隆变化实例，判断该克隆实例是否满足克隆一致性维护需求。

更进一步，本文将克隆代码一致性维护需求预测问题转换成为一个分类问题，因而可以使用机器学习中的方法来解决此问题。在本文第三章和第四章的研究中，仅仅考虑了贝叶斯网络方法，为了更深入研究克隆代码的一致性预测问题，本文将会使用五种不同的机器学习方法在预测克隆实例的一致性维护需求。将在后文中详细介绍。

5.3 基于机器学习的克隆代码一致性需求预测框架

为解决本章所提出的克隆一致性需求维护预测问题，本文首先给出了一个方法框架。基于机器学习的克隆代码一致性需求预测框架如图 5-1 所示。方法可以划分为三个阶段，克隆实例收集阶段、克隆实例表示阶段和一致性维护需求预测阶段。收集阶段旨在收集系统中全部的克隆实例（包括创建实例和变化实例），可将其用于使用机器学习方法中来训练预测模型。由于实际的克隆实例无法直接应用于机器学习方法中，因此在表示步骤中将提取不同的属性值表示克隆实例，所提取的属性值将包含克隆实例的有意义的信息。接下来，在预测步骤，使用属性化的克隆实例构建和训练机器学习模型，并使用其预测克隆实例的克隆一致性维护需求。

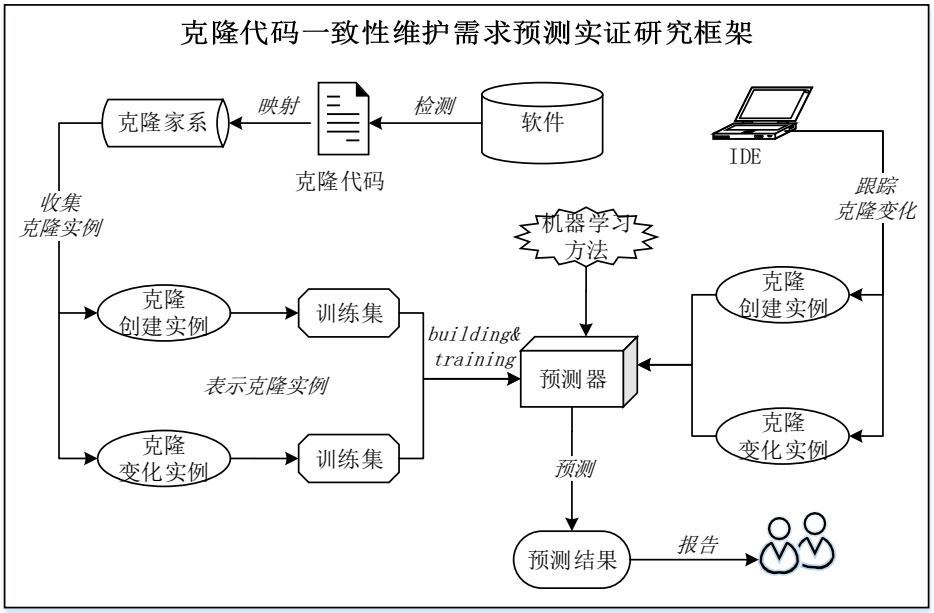


图 5-1 克隆代码一致性维护需求预测实证研究框架

Fig.5-1 The framework for empirical study on clone consistency prediction

具体来说，在收集阶段中，通过构建系统的克隆家系从软件中收集所有的克隆实例。使用 NiCad 来检测软件版本中的所有克隆，并通过在相邻版本的克隆组之间进行映射来构建克隆家系，用于识别克隆实例。在表示阶段中，通过提取属性值表示克隆创建和变化实例，提取代码属性和上下文属性表示克隆创建实例，提取代码属性、上下文属性和历史属性表示克隆变化实例。在预测阶段中，使用收集到的克隆创建实例训练贝叶斯网络，并在克隆创建时预测克隆一致性维护需求。在使用已构建好的模型进行预测时，可将该模型嵌入到软件开发环境中。软件开发环

境需要实时监测克隆创建实例和变化实例并提取克隆实例的度量值。最后使用模型预测其一致性维护需求，根据预测结果提醒程序开发人员采取进一步的操作。

克隆实例有两种不同的预测结果，即满足一致性维护需求和不满足维护需求。对于满足一致性维护需求实例来说，其在将来的演化中可能会引发一致性变化，程序开发人员需要采取相应的操作。例如，拒绝克隆创建实例或者检查克隆变化实例一致性。对于不满足一致性维护需求实例来说，其在将来的演化中不会引发一致性变化，程序开发人员需要采取相应的操作，如接受克隆创建实例。

本章使用多个不同的机器学习模型预测克隆实例的一致性。所提取的克隆实例的属性即为机器学习模型中的输入特征，用于构建不同机器模型的结构。所收集到的克隆变化实例可作为机器学习的训练集，用于训练机器学习模型的相关参数，细节可参考本文后续章节。本章使用和比较五种不同的机器学习方法，以帮助开发人员选择最佳的预测手段。

5.4 克隆实例收集和表示

收集和表示克隆实例可以生成克隆一致性预测的训练集，为将克隆实例应用于机器学习中，提取不同的属性值表示克隆实例，从而可以训练机器学习模型。

5.4.1 克隆实例收集

收集克隆实例的目的在于生成克隆一致性预测的训练集，并将其用于训练机器学习模型。通过构建系统的克隆家系并识别其中的克隆演化模式，可以从软件中收集所有的克隆实例。首先使用 NiCad 来检测软件版本中的所有克隆，然后通过在相邻版本的克隆组之间进行映射来构建克隆家系，最后识别 Type-1 和 Type-1 克隆一致性演化模式识别系统中的克隆创建和变化实例。

根据定义 5.3，克隆家系 CGE 中的初始节点即是克隆创建实例，发生变化的克隆组是变化实例，因此因此通过构建克隆家系可以收集克隆创建和变化实例。

（1）构建克隆家系。首先，下载系统所有版本的源代码，并使用 NiCad 的默认配置检测检测每一版本的中 Type1-3 的克隆代码。然后，通过映射所有相邻版本的克隆代码，构建系统中全部克隆家系。为完成版本间的映射，为每个克隆片段生成一个克隆区域描述符 CRD[99]，使用基于 CRD 的克隆映射算法映射两个连续版本之间的所有克隆片段和克隆组 [111][112]。根据克隆映射结果，构建系统的克隆家系。

（2）识别克隆演化模式和收集克隆实例。首先，识别克隆家系中的克隆演化模

式，尤其是克隆一致性变化模式。构建克隆家系后，通过对比相邻版本的克隆代码，可以识别克隆家系的克隆演化模式（参考定义 2.4 和5.2 ）。所识别的克隆演化模式有三个作用：（a）克隆演化模式可以帮助收集克隆变化实例。根据定义 2.4 可以识别系统中发生一致性变化的克隆代码和克隆组，从而确定克隆家系中的克隆变化实例。（b）可以用于表示克隆变化实例，本文使用克隆演化模式作为表示克隆变化实例的部分演化属性。因此，克隆演化模式可以用于克隆变化实例表示中。（c）克隆演化模式可以帮助确认克隆一致性维护需求。根据定义 5.4 ，一致性维护需求，可以通过遍历克隆家系 CGE 是否发生了一致性变化模式进行确定。

然后，收集克隆实例。根据定义 5.3 通过遍历克隆家系的根节点，可收集系统中所有的克隆创建实例。在收集克隆变化实例后，还需确认该实例中的被复制和被粘贴代码（参见本文第三章收集克隆创建实例小节 3.4 ）。根据定义 5.3通过识别克隆家系中的变化克隆组，便可以收集系统中的克隆变化实例。

（3）标识克隆变化实例的一致性维护需求。在收集所有的克隆实例后，还需确认相关实例的一致性维护需求。根据定义 5.4 ，通过遍历克隆实例所在的克隆家系 CGE 的演化情况确定其一致性维护需求。如果克隆实例在其演化过程中发生了一致性变化模式（ 5.2 ），则该实例满足一致性维护需求，否则不满足维护需求。

5.4.2 克隆实例表示

本章使用机器学习方法预测克隆代码的一致性维护需求，并使用软件中既有的克隆实例训练机器学习模型。但是，实际的克隆实例无法直接应用于机器学习中。因此，本文提取相应的属性值表示克隆代码实例，即提取代码、上下文两组属性代表克隆创建实例，提取代码属性、上下文属性和演化属性代表克隆变化实例。其中代码属性和上下文属性相似，但从不同的角度表示克隆创建实例和变化实例，克隆创建实例中的属性表示了创建时的被复制和被粘贴代码的特征，变化实例中的属性则表示了发生变化的克隆组的特征。

为表示克隆创建实例，对创建实例的被复制和粘贴克隆代码使用不同的属性。代码属性用于表示被复制的克隆的特征，包括：克隆代码粒度、Halstead 属性、结构属性、参数访问数量、总函数调用次数、本地函数调用次数、库函数调用次数、其它调用次数。上下文属性用于表示被粘贴的克隆代码的特征，包括：代码相似度、局部克隆标识、文件名相似度、文件名相似度标识、方法名相似度、总参数名相似度、最大参数名相似度、总参数类型相似度、块信息标识。属性具体信息可以参考本文第二章相关属性值部分（ 3.5 ）。

对于克隆变化实例，从克隆组的角度重新提取了代码属性和上下文属性，并从演化的角度新增演化属性。代码属性从代码自身的角度描述了克隆变化实例中克隆代码特征，包括克隆粒度、代码行平均、Halstead 属性平均、结构属性平均、总函数调用次数平均、本地函数调用次数平均、库函数调用次数平均、其它调用次数平均。上下文属性描述了克隆变化实例所在的克隆组的克隆关系特征，包括代码相似度平均、文件名相似度平均、文件名相似度变量、方法名相似度平均、总参数名相似度平均、最大参数名相似度平均、总参数类型相似度平均、块信息标识。历史属性描述了克隆变化实例所在克隆组在克隆变化发生前的历史演化特征，包括变化实例寿命、历史演化模式统计、当前演化模式、历史变化统计。同时，还提供了克隆变化实例的变化属性。属性具体信息可以参考本文第二章相关属性值部分（4.5）。

5.5 克隆一致性预测

本文将克隆代码的一致性维护需求问题，转化成了克隆创建实例和克隆变化实例的分类问题，即给定一个克隆实例，判别其是否满足克隆一致性维护需求。为了验证本文方法的有效性，本文使用了五种不同的机器学习方法，并将它们应用于克隆一致性维护需求的预测中。因此，本节先简单介绍所选择的机器学习方法。随后，使用属性化的克隆创建实例和克隆变化实例构建和训练不同的克隆一致性预测模型，并使用训练好的机器学习模型，在克隆代码创建时和变化时预测其一致性维护需求。

5.5.1 机器学习方法

在本章的实证研究中，为解决本文所提出的研究问题，使用了五种不同的机器学习方法，即：贝叶斯网络方法（Bayesian Network，简称为 BayesNet）[116]、朴素贝叶斯方法（Native Bayesian，本文简称为 Native）[117]，支持向量机方法（Support Vector Machine，简称为 SVM）[118] 、K 近邻方法 (K-Nearest Neighbors，简称为 KNN) [119] 和决策树方法 (Decision Tree，本文简称为 Tree)[120]。

（1）贝叶斯网络方法

贝叶斯网络是一个是一种概率图型，使用已经观察到的事件来预测将来可能发生的事件 [116]。关于贝叶斯网络的信息可以参考本文第三章 3.6.1 节贝叶斯网络。

（2）朴素贝叶斯方法

朴素贝叶斯方法和贝叶斯网络类似，是运用贝叶斯定理为基础的简单概率分类器。

但与贝叶斯网络不同的是，朴素贝叶斯方法的特征之间是强（朴素）独立的，因此称为朴素贝叶斯，即假定样本每个特征与其他特征都不相关。

（3）支持向量机方法

支持向量机是另一种常见的机器学习方法，可以应用在分类与回归问题中。SVM 模型将实例表示为空间中的点，并且试图构造一个超平面将不同类的实例（点）间隔开。更正式地来说，支持向量机在高维或无限维空间中构造超平面或超平面集合，可以用于分类问题中。直观来说，分类边界距离最近的训练数据点越远越好，因为这样可以缩小分类器的泛化误差。

以本文的克隆一致性需求分类为例，每一个克隆代码实例会抽象称为高维空间中的一个“点”，空间维数等同于所提取的属性数量。在使用 SVM 分类克隆实例时，将构造一个超平面分割开两种类别的克隆代码实例。

（4）K 近邻方法

KNN 方法是一种用于分类和回归的非参数统计方法。KNN 是一种基于实例的学习方法，是局部近似和将所有计算推迟到分类之后的惰性学习。KNN 会推迟对训练数据的建模，直到需要分类样本时才进行。在 KNN 分类中，输出是一个分类族群。一个实例的分类是由其邻居的“多数表决”确定的，K 个最近邻居（k 为正整数，通常较小）中最常见的分类决定了赋予该对象的类别。若 k = 1，则该对象的类别直接由最近的一个节点赋予。邻居都取自一组已经正确分类（在回归的情况下，指属性值正确）的对象。

以本文克隆一致性预测为例，每一个克隆代码实例是 KNN 中的一个实例。在进行预测时，被预测的克隆实例的类别，将会有其最近的 K 个邻居进行表决，从而确定其一致性维护需求。

（5）决策树方法

机器学习中另一个常见的分类方法是决策树。决策树是一种简单但是广泛使用的分类器。通过训练数据构建决策树，可以高效的对未知的数据进行分类。决策树代表的是属性值与对象类别之间的一种映射关系。树中每个节点表示某个属性，而每个分叉路径则代表的某个可能的权重，而每个叶结点则对应从根节点到该叶节点所经历的路径所表示的对象的类别。决策树仅有单一输出，若欲有复数输出，可以建立独立的决策树以处理不同输出。

以克隆一致性需求预测为例，克隆实例所提取的属性即是决策树中的属性，最后的克隆一致性维护需求则是对象的类别。

5.5.2 训练与预测

接下来，使用收集到的克隆实例训练不同的机器学习模型，并预测克隆代码的一致性维护需求。本章将在两个不同的时刻预测克隆代码的一致性维护需求，分别为为克隆创建时和克隆变化时，因此针对每一种机器学习方法，也会训练两种不同的模型。

本章没有对机器学习方法进行改进和研究，所使用模型的构建和训练通过调用现有机器学习工具包 WEKA 完成。本文使用 WEKA 机器学习工具包内提供的机器学习算法进行一致性维护需求预测工作，通过实验对比上述五种机器学习算法的预测效果，从而帮助程序人员选择最好的机器学习模型。

对于每个软件系统，首先，通过收集系统中所有的克隆实例（创建实例和变化实例），并提取相应的属性，用于构建模型训练所需的数据集。然后，调用 WEKA 中的机器学习算法，分别构建克隆创建和变化时的预测器。对每一种机器学习方法，将会构建和训练两种模型一起预测克隆代码的一致性维护需求。

根据定义 5.4，克隆实例有两种不同的状态：需要一致性维护和不需要一致性维护。因此在进行一致性维护需求预测时，克隆实例也具有两种不同的预测结果：

• 需要一致性维护：若克隆创建实例的预测结果为“需要”，软件开发人员需要谨慎的执行克隆创建操作（复制和粘贴）。因为，该克隆创建实例，在未来演化的过程中可能会引发一致性变化，从而向系统中引入额外的维护代价。

若克隆变化实例的预测结果为“需要”，软件开发人员需要检测克隆变化实例所在的克隆组的一致性问题，考虑一致地修改组内其它的克隆代码。因为，该克隆变化实例，在未来演化的过程中可能会引发一致性变化，遗忘这种变化会向系统中引入缺陷，从而降低软件质量。

• 不需要一致性维护：若克隆创建实例的预测结果为“不需要”，软件开发人员可以自由的执行克隆创建操作（复制和粘贴），从而节约开发时间提高开发效率。因为，该克隆创建实例，在未来演化的过程中不会引发一致性变化，也不会导致额外的维护代价。

若克隆创建实例的预测结果为“不需要”，软件开发人员可以自由的修改克隆变化实例所在克隆组的克隆片段。因为，该克隆变化实例，在未来演化的过程中不会引发一致性变化，也不会导致一致性缺陷。

在使用已训练好的模型进行预测时，可以与软件开发过程相结合，将该模型嵌入到软件开发环境中，帮助程序开发人员实现边开发边预测克隆实例的一致性维护需求。首先，在软件开发环境中需监测程序员的复制粘贴操作和对克隆代码的

修改，识别由此产生的克隆实例（克隆创建实例和变化实例）。然后，根据上文描述的代码、上下文和演化属性，提取相应的特征表示相应的克隆实例。最后，使用训练好的预测器预测相应克隆实例的一致性维护需求，根据预测结果提醒程序开发人员采取进一步的操作。

5.6 基于 eclipse 的克隆一致性维护需求预测插件

为了与软件开发过程相结合，本章设计并实现了一个克隆代码的一致性维护需求插件，可以帮助开发人员预测克隆一致性维护需求。所设计的克隆一致性预测插件可以嵌入到软件开发环境中（eclipse），实现边开发、边预测、边维护克隆代码的一致性。基于 eclipse 的预测插件，可以帮助程序开发人员避免克隆变化导致的一致性维护代价及一致性缺陷，从而帮助提高软件质量和可维护性。

5.6.1 插件基本模块

为了满足克隆代码的一致性需求预测，本文基于 eclipse 所设计的插件具有三个基本模块，可以帮助程序开发人员解析用 java 语言实现的软件系统。三个基本模块分别是：预处理模块、属性提取模块和一致性预测模块[1]。

(1) 预处理模块。

预处理模块可以识别大多数克隆检测工具的结果，并构建克隆家系和识别克隆演化模式，从而实现对系统克隆实例的收集。首先，需要人工的使用克克隆检测工具检测系统中的所有的克隆代码，并将检测结果作为插件的输入。（本文中使用的克隆检测工具是 NiCad [？]）对于软件系统的每一个版本，克隆代码将会被组成克隆组的形式，并使用文件名、起始行号表示所有的克隆代码，即“file _name”、“start _line”和“end _line”。然后，使用 CRD 重新描述克隆代码并重新组织为新的数据结构，从而方便构建克隆家系。最后，本插件将基于 CRD 对所有版本中的克隆代码，构建构建系统的克隆家系，并识别克隆演化模式。方法参加本文克隆家系构建和模式识别部分。根据所构建的克隆家系和识别的演化模式，可以方便的获取系统中已经存在的克隆创建和变化实例。值得注意的是，构建克隆家系时需要系统所有版本的源代码，用于生成克隆代码的 CRD 表示。

(2) 属性提取模块

属性提取模块将不同的克隆代码实例抽象成相应的属性值，对克隆创建实例提取代码属性和上下文属性，对克隆变化实例提取代码属性、上下文属性、演化属

[1]本文插件目前仅支持基于 java 语言的源代码，但对于其它语言的扩展也较为容易。

性和变化情况。首先，代码属性和上下文属性提取，可以使用抽象语法树（Abstract Syntactic Tree，AST）对程序源代码进行解析获取。对每一个克隆代码以及克隆代码所在文件，使用 Eclipse AST 中的 ASTParser 类将克隆片段所在源代码解析成 AST[2]。通过遍历语法树访问相应关键节点，根据本文描述的属性值计算克隆实例的代码属性和上下文属性。然后，历史属性提取，历史属性提取通过遍历克隆实例的克隆家系，根据属性描述计算相应历史属性。值得注意的是，属性提取时同样需要系统所有版本的源代码，用于生成克隆代码的抽象语法树和属性计算。

(3) 克隆一致性预测模块。

一致性预测模块实现了对相关机器学习模型的训练，并可以对在软件开发过程新产生的克隆创建实例和变化实例进行一致性维护需求预测。本插件并没有具体实现机器学习方法，而是通过调用 WEKA 中的 API 完成对机器学习模型的构造、训练和预测功能。WEKA 是一个 Java 语言实现的数据挖掘和机器学习开源工具。WEKA 提供了丰富的接口帮助程序开发人员调用相关的机器学习方法，可以使用极为灵活的方式对机器学习模型进行训练和预测[3]。

5.6.2 克隆实例跟踪

在软件开发过程中预测克隆代码的一致性需求维护，还需要实时的捕获软件中产生的克隆实例，即跟踪克隆实例的产生（克隆创建实例和克隆变化实例）。

（1）克隆创建实例跟踪

研究表明，软件中的克隆代码主要是由于复制和粘贴操作导致，即克隆创建实例产生的直接原因是程序开发人员的复制和粘贴操作。因此，测复制和粘贴操作即可跟踪克隆创建实例的产生。

为在 eclipse 中监测程序开发人员的复制粘贴操作，

（2）克隆变化实例跟踪

为了跟踪克隆变化的产生，需要实现跟踪系统中克隆代码以及变化。

Tracking clones changes

5.6.3 克隆一致性预测插件使用

由于机器学习模型的构建和训练往往需要大量的时间，不应该也不需要在每次开发时训练机器学习模型。因此，在实际的开发过程中，模型的训练和预测是分

[2]抽象语法树可参见：http://www.eclipse.org/articles/Article-JavaCodeManipulation_AST/

[3]使用 WEKA 可参见：http://weka.wikispaces.com/。

开进行的。

为了更为灵活的使用本文方法，提供两种构建和训练预测模型的方式。第一个是用项目本身的历史数据来训练预测模型。在这种情况下，本文的插件首先调用预处理模块，构建项目本身所有克隆家系和收集项目中的所有克隆实例。然后调用属性提取模块，将提取收集到的克隆实例的属性值，并生成训练集。最后，使用该训练集建立和训练克隆一致性预测模型。第二个其它项目数据作为训练集。原因在于，对于某些项目而言，由于实际版本的限制，其历史的克隆实例可能较少，不足以较好的训练所需要的模型。因此，需要使用其它项目的历史数据作为训练集。需要注意的是，随着时间的推移，当项目自身可以收集到足够的数据时，建议开发人员重新使用项目自身的数据训练机器学习模型，从而达到较好的预测效果。

当监测到具体的克隆实例产生时，调用已经训练好的机器学习模型进行预测。软件开发过程中，监测到克隆实例产生时，调用属性提取模块提取本次实例的属性，并使用训练好的一致性模型预测其一致性，根据预测结果通知程序开发人员采取相应措施。值得注意的是，在实际预测时，克隆变化实例预测需要项目的历史版本源代码，因为历史属性中包含克隆变化实例的历史变化过程。为了轻量化预测过程，程序开发人员可以将克隆变化实例中的历史变化属性移除。

5.7 实验结果与分析

本节给出本章的实验结果与分析，使用五种不同的机器学习方法，对克隆代码创建时和变化时同时进行一致性维护预测。首先简单介绍了实验所使用的实验系统和评估方法，然后详细给出每个实验的结果与分析。

5.7.1 实验系统与实验设置

为了解决本章的研究问题，本章在四个开源项目上进行了实验。表 5-1 给出了实验系统中克隆实例的统计信息，包括克隆创建实例和克隆变化实例。从该表可以看出，系统中克隆创建实例的数量要多于克隆变化实例的数量。克隆创建实例的数量范围为 633 到 3666 个，克隆变化实例的数量范围在 159 到 1040 个，系统 jEdit 是克隆规模最小的系统。表中第 3 和 4 列给出了不需要和需要一致性维护的克隆实例的数量和比例。不需要一致性维护的克隆实例，在其未来的演化中不会导致一致性变化和额外的维护代价。需要一致性维护的克隆实例，在其演化过程中可能导致一致性变化及克隆一致性缺陷。

从表 5-1 中可以观察到两个现象。第一，软件系统中存在大量的克隆创建实例

表 5-1 实验系统的克隆实例信息统计

Table5-1 The statistics for clone instances in four projects

类型	系统	不需要一致性维护	需要一致性维护	总数
克隆创建实例	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
克隆变化实例	ArgoUML	288(67.45%)	139(32.55%)	427
	jEdit	78(49.06%)	81(50.94%)	159
	jFreeChart	452(43.46%)	588(56.54%)	1040
	Tuxguitar	91(25.71%)	263(74.29%)	354

（三个项目中有上千的例子），同时大部分的克隆创建实例在其演化过程中不满足一致性维护要求 (比例从 59.8% 到 88.47%)。这表明克隆创建操作（即复制和粘贴操作）已经成为开发人员的常用技术，并且它们中的大多数在演化过程中不会在将来引入任何一致的变化，建议开发人员可以使用复制和粘贴操作节省开发时间。第二，在克隆变化实例中，三个系统中只有数百个变化实例，仅有 jFreeChart 中有 1040 变化实例。这表明相对于克隆代码在演化过程中不会频繁地发生变化，但也有相当数量的变化。值得注意的，这些变化中需要一致性维护的变化实例比例占相当大的一部分，其比例为 33% 到 74 %。克隆代码的一致性变化更容易引发一致性变化，从而导致增加克隆代码一致性缺陷的风险，因此开发人员在软件开发过程中应该更加注意软件开发过程中的克隆变化。

本章对克隆代码的一致性需求预测进行实证研究，目的是解决本文提出的研究问题：克隆代码的一致性维护需求预测是否可以应用于其它的机器学习方法中，软件开发人员应如何结合软件开发过程中执行克隆一致性需求预测？

本章将此研究问题划分为三个子问题，因此实验也相应的划分为三个部分以回答三个子问题。对于第一个研究问题，进行了“克隆代码创建时一致性预测实验”，用于验证五种不同机器学习方法对克隆代码创建时一致性预测的有效性。类似地，为了回答第二个研究问题，本章设计了“克隆代码变化时一致性预测”实验。对克隆变化实例进行实验，也分析了五种不同的机器学习方法在克隆变化实例上的有效性。最后，第三个研究问题没有具体的实验，但通过结合软件开发过程，并对比克隆创建实验和变化实验，回答了本章提出的第三个研究问题。同时，给出了一些相关的建议，帮助程序开发人员在开发实践中使用这些预测模型和技术。

在克隆创建实验和克隆变化实验中，与本文第三章和第四章相似，又分别进行了两种实验[4]：全属性实验和属性组实验，从两个不同的角度评估五种机器学习方法的有效性。全属性实验，使用所提取的全部属性进行预测，并对比五种机器学习方法。属性组实验中，使用不同的属性组进行实验，分析不同属性组在五种不同机器学习方法上的预测能力。

在每一个实验中，本章使用五种不同的机器学习方法，并同时对需要一致性维护的克隆实例和不需要一致性维护的克隆实例进行预测。在实验中，首先，将数据集分为 10 个数据集作 10 倍交叉验证，用于评估预测效果。然后，分别计算满足一致性维护需求和不满足一致性维护需求的准确率（Precision）、召回率（Recall）和 F 值（F-measure），用于评估不同类型实例的预测效果。最后，将这两组预测结果进行加权平均，使用平均准确率、平均召回率和平均 F 值作为实验指标，评估预测模型的预测能力。

因此，对克隆创建实例和克隆变化实例的评价指标为：准确率、召回率和 F 值，如下所示：

• 准确率（Precision）：该指标评估克隆实例一致性维护需求预测的准确程度，包含了满足和不满足一致性维护需求的实例。首先，计算预测中满足一致性维护需求的克隆实例的准确率 “P_1”，即预测为满足一致性维护需求的克隆实例中正确预测的数量与全部预测数量的比值。然后，相似的计算不满足一致性维护需求克隆实例的准确率 “P_2”。最后，将这两个值进行加权平均，作为整个模型的准确率。“P_1” 是满足一致性维护需求的准确率，并且训练集中所有满足一致性的实例数量为 “N_1”；类似地，“P_2” 为不满足一致性维护需求的准确率，且 “N_2” 为实例数量。平均准确率（Average Precision）计算如下,

$$\textit{Ave-Precision} = \frac{P_1 \times N_1 + P_2 \times N_2}{N_1 + N_2}$$

在实验中，使用此 “Average Precision” 作为模型准确率（Precision）。

• 召回率（Recall）：该指标评估克隆实例一致性维护预测的查全能力，包含了满足一致性维护需求和不满足一致性维护需求的实例。同准确率，分别计算满足和不满足一致性维护需求的实例的召回率，然后对两者进行加权平均。R_1” 是满足一致性维护需求的召回率，并且训练集中所有满足一致性的实例数量为 “N_1”；类似地，“R_2” 为不满足一致性维护需求的召回率，且 “N_2” 为实例数量。平均召回率（Average Recall）计算如下,

$$\textit{Ave-Recall} = \frac{R_1 \times N_1 + R_2 \times N_2}{N_1 + N_2}.$$

[4]值得注意的是，本章没有进行交叉验证实验。

在实验中，使用此 “Average Recall” 作为模型召回率（Recall）。

• F 值（F-measure）： 该指标可以评估所有克隆实例的准确率和召回率的平均有效性。相似地，先分别计算满足和不满足一致性维护需求的克隆实例的 F 值，然后根据实例的数量进行加权平均。给定一个预测的准确率为 precision、召回率为 recall，则该预测的 F-Measure 可计算如下,

$$F\text{-}measure = 2 \times \frac{precision \times recall}{precision + recall}$$

为计算本章预测的平均 F-measure，“F_1” 是满足一致性维护需求的克隆实例的 F 值，且实例的个数为 “N_1”；“F_2” 是不满足一致性维护需求的克隆实例的 F 值，且实例的个数为 “N_1”，则平均 F 值（Average F-measure）可计算如下,

$$Ave\text{-}F\text{-}measure = \frac{F_1 \times N_1 + F_2 \times N_2}{N_1 + N_2}.$$

在实验中，使用此 “Average F-measure” 作为模型 F 值（F-measure）。

5.7.2 项目内克隆一致性需求预测

5.7.2.1 克隆创建时一致性预测

在本实验中，我们解决了克隆克隆实例的第一个子研究问题。子问题 1：在克隆克隆时间，其他机器学习方法是否可以用于克隆一致性要求预测？属性集是否对此预测产生积极或消极的影响？

我们采用五种不同的机器学习方法来预测这四个项目的克隆创建时的克隆代码的一致性维护需求，实验结果如表5-2 和 5-3 所示。

表 5-2 给出了克隆创建实例的使用全部属性组的预测结果。从表中可以看出，五种不同的机器学习方法均可以高效地预测克隆一致性要求，准确率从 79.3% 到 95.8%，召回率从 79.4% 到 95.8% ，F 值从 79.4% 至 95.7%。同时，对比不同机器学习方法的有效性，通过对比发现，SVM 在这四个实验系统上具有最佳的预测能力。具体来说，SVM 在系统 ArgoUML、jEdit 和 jFreeChart 上有最好的结果。对于 Tuxguitar 系统来说，决策树具有最好的结果，SVM 具有第二好的预测结果。与此同时，贝叶斯网络和朴素贝叶斯方法在这四个项目中几乎具有相对较差的预测结果，但是相差并不明显。因此，建议开发人员在需要预测克隆创建实例时首先考虑 SVM，其它的机器学习方法也可以作为选择进行考虑。

为了探索不同属性组对预测效果的影响，在五个机器学习方法上进行了属性组实验，实验结果如表 5-3 所示。表中“全部”列是全部属性实验结果，属性组实验结果在“代码”列和“上下文”列中。

表 5-2 克隆创建实例一致性维护需求平均预测效果

Table 5-2 The Average Effectiveness of Creating Instances for Consistency-Requirement

指标	方法	ArgoUML	jEdit	jFreeChart	Tuxguitar
Percision	BayesNet	0.935	0.889	0.883	0.831
	Native	0.886	0.871	0.869	0.793
	SVM	0.958	0.924	0.906	0.888
	KNN	0.94	0.886	0.9	0.848
	Tree	0.939	0.898	0.893	0.889
Recall	BayesNet	0.936	0.877	0.882	0.836
	Native	0.888	0.82	0.868	0.794
	SVM	0.958	0.921	0.904	0.883
	KNN	0.94	0.889	0.9	0.848
	Tree	0.94	0.905	0.892	0.891
F-measure	BayesNet	0.935	0.882	0.881	0.832
	Native	0.887	0.84	0.867	0.794
	SVM	0.957	0.903	0.903	0.876
	KNN	0.94	0.887	0.9	0.848
	Tree	0.939	0.901	0.892	0.89

从这个表中可以看出，属性组实验结果和全部属性组实验结果没有显着差异。这表明所使用的属性组不会对克隆创建时一致性预测产生负面影响。尽管如此，在全部实验结果中，效果最差的实验结果往往出现在属性组的实验结果中，即在仅使用代码属性或仅适用上下属属性预测时。具体来说，对系统 ArgoUML、jFreeChart 和 Tuxguitar，其仅使用代码属性进行预测时五种机器学习方法的预测效果最差；而对系统 jEdit 仅仅使用“上下文”属性的五种机器学习方法预测效果最差。

最后，分析不同机器学习方法之间的预测能力，可以得出与全属性实验相似的结论，即五种机器学习方法的预测能力相差不大，但 SVM 方法具有相对最好的预测能力。因此建议程序开发人员在预测时保留所有的属性，并优先选择 SVM 方法。

根据表 5-2 和 5-3 中的实验结果，可以给出对子问题 RQ1 的回答和建议。在克隆代码创建时，五种机器学习方法都可以应用于克隆代码的一致性维护需求预测中，并具有相似的预测能力。另外，其预测能力仍然具有一点差异，SVM 似乎拥有相对最佳的预测效果，因此优先推荐使用 SVM 技术应用在一致性预测中。与此同时，所使用的属性组在预测中起到了积极的作用，建议程序开发人员在预测时保留所有的属性组进行变化时一致性预测。

表 5-3 克隆创建实例的属性组平均预测效果

Table5-3 The Average Effectiveness of Attribute Set on Creating Instances for Consistency-Requirement

指标	方法	ArgoUML			jEdit			jFreeChart			Tuxguitar		
		All	**Code**	**Context**	**All**	**Code**	**Context**	**All**	**Code**	**Context**	**All**	**Code**	**Context**
Percision	BayesNet	0.935	0.912	0.917	0.889	0.885	0.83	0.883	0.808	0.903	0.831	0.811	0.843
	Native	0.886	0.866	0.877	0.871	0.865	0.832	0.869	0.755	0.873	0.793	0.747	0.824
	SVM	0.958	0.928	0.942	0.924	0.924	0.909	0.906	0.819	0.906	0.888	0.834	0.873
	KNN	0.94	0.911	0.933	0.886	0.903	0.877	0.9	0.808	0.898	0.848	0.806	0.861
	Tree	0.939	0.904	0.93	0.898	0.882	0.876	0.893	0.802	0.891	0.889	0.8	0.881
Recall	BayesNet	0.936	0.914	0.919	0.877	0.869	0.852	0.882	0.803	0.903	0.836	0.817	0.846
	Native	0.888	0.868	0.88	0.82	0.836	0.826	0.868	0.752	0.873	0.794	0.756	0.816
	SVM	0.958	0.929	0.943	0.921	0.921	0.918	0.904	0.806	0.904	0.883	0.837	0.874
	KNN	0.94	0.912	0.934	0.889	0.912	0.885	0.9	0.803	0.898	0.848	0.81	0.862
	Tree	0.94	0.905	0.931	0.905	0.896	0.889	0.892	0.796	0.89	0.891	0.807	0.882
F-measure	BayesNet	0.935	0.912	0.917	0.882	0.876	0.84	0.881	0.797	0.902	0.832	0.811	0.844
	Native	0.887	0.867	0.878	0.84	0.848	0.829	0.867	0.741	0.872	0.794	0.75	0.819
	SVM	0.957	0.927	0.941	0.903	0.903	0.907	0.903	0.797	0.903	0.876	0.827	0.869
	KNN	0.94	0.912	0.933	0.887	0.905	0.881	0.9	0.796	0.897	0.848	0.807	0.862
	Tree	0.939	0.904	0.93	0.901	0.887	0.881	0.892	0.788	0.889	0.89	0.802	0.881

5.7.2.2 克隆创建时一致性预测

在本节中解决本章提出的第二个研究问题。子问题 2：在克隆更改时间，其他机器学习方法是否可以用于克隆一致性要求预测？属性集是否对我们的预测产生积极或消极的影响？

为了解决这个问题，使用五种不同的机器学习方法在克隆代码变化时进行一致性维护预测，实验结果如表 5-4 和表 5-5 所示。

表 5-4 给出了使用全属性组在五种不同机器学习方法上的克隆变化实例的预测结果。从这个表可以看出，克隆变化实例在这五种机器学习方法上具有可以接受的预测效果。准确率的范围从 58.1% 到 79.3 %，召回率从 57.9% 到 79.1 %，而 F-measure 从 57.3 % 到 79 %。

根据这四个系统的预测结果，通过比较发现预测效果最好的系统是 jFreeChart，三个评价指标中拥有最多的最大值，同时 jFreeChart 中的克隆变化实例的个数也是最多的。然而，系统 jEdit 是所有系统中预测效果最差的。其原因在于预测模型需要足够的训练集进行训练，然而 jEdit 中的训练数据太少导致所建立的模型训练不充分，jEdit 仅有 159 个克隆变化实例。本文建议在对克隆代码进行一致性预测时，需要对模型进行充分的训练，以达到最佳的预测效果。

此外，通过对比五种机器学习方法在四个系统上的有效性，除基于决策树的方法外，另外三种方法的预测能力十分相似，没有明显的差异。同时，相对而言 SVM 方法具有相对较好的预测效果。基于贝叶斯的方法（贝叶斯网络和朴素贝叶斯方法）具有十分友好的预测结果，其预测结果是可以接受的。应该注意的是，KNN 和 Decisions Tree 这两种机器学习方法相对而言预测能力不如其他的机器学习方法，尤其是 Decisions Tree 方法。

因此，建议开发人员在预测克隆变化实例时也可以考虑 SVM，但是贝叶斯的方法也具有不错的预测能力，仅比 SVM 相差一点点。

为探索属性组对预测能力的影响，同样进行在五种不同的机器学习方法上进行了属性组实验，实验结果如表找到属性的贡献，我们还进行属性集的实验，依次删除一个属性集。属性集的有效性如表 5-5 所示。依次移除代码属性、上下文属性和演化属性，“All”列是使用全部属性组的实验结果，“Code”、“Cont”、“Evo”分别是是删除 Code，Context，Evolution 属性组后的实验结果。

从表中可以看出，三个属性集中任何一个都没有在预测中起到决定性的作用，同时它们任何一个也没有起到消极的作用。尽管如此，三个属性组中的任何一个可能在预测中仅起到了有限的一点积极作用，但三个属性组作为一个整体却仍然可以以可以接受的准确率和召回率有效的预测克隆代码的一致性维护需求。另外，

表 5-4 克隆变化实例的一致性维护需求平均预测结果

Table5-4 The average effectiveness of changing instances

指标	系统	ArgoUML	jEdit	jFreeChart	Tuxguitar
Precision	BayesNet	0.724	0.686	0.791	0.72
	Native	0.734	0.696	0.778	0.729
	SVM	0.744	0.704	0.793	0.733
	KNN	0.733	0.597	0.772	0.672
	Tree	0.682	0.581	0.742	0.637
Recall	BayesNet	0.735	0.686	0.791	0.746
	Native	0.726	0.692	0.778	0.737
	SVM	0.752	0.704	0.791	0.734
	KNN	0.731	0.597	0.77	0.706
	Tree	0.698	0.579	0.742	0.672
F-Measure	BayesNet	0.726	0.686	0.79	0.726
	Native	0.729	0.689	0.778	0.733
	SVM	0.729	0.704	0.789	0.733
	KNN	0.732	0.597	0.771	0.683
	Tree	0.632	0.573	0.739	0.651

不同的属性集在预测中可能会发挥不同的作用。通过比较五种机器学习方法的预测结果，可以得出与全属性实验相似的结论，即方法之间的差异并不会导致预测结果的明显差异，并取得来的相一致的预测能力。但是，支持向量机似乎拥有相对最佳的预测能力，同样贝叶斯的方法同样也可以取得不错的预测效果。因此在预测中本文需要保留所有属性集，并优先选择 SVM 方法

根据对克隆变化实例的预测结果（表 5-4 和 5-5 ），可以对 RQ2 进行回答。当克隆代码发生变化时，五种机器学习方法在现有的属性值上都具有可以接受的预测能力，合理的准确率、召回率和 F 值。尽管 SVM 方法具有相对较好的预测效果，但是彼此之间的相差并不大。同时，所提起的属性组作为一个整体在预测中起到了积极的作用。所有属于整体的属性都对这一预测产生了积极的影响。因此建议开发人员在预测时使用全部的属性组，并使用 SVM 作为预测模型。

5.7.3 跨项目克隆代码一致性预测实验

本节给出跨项目预测的实验结果。

表 5-5 克隆变化实例的属性组一致性维护需求平均预测效果

Table 5-5 The average effectiveness of attribute set for changing instances

Metric	Method	ArgoUML				jEdit				jFreeChart				Tuxguitar			
		All	Code	Cont	Evo	All	Code	Cont	Evo	All	Code	Cont	Evo	All	Code	Con	Evo
Percision	BayesNet	0.724	0.737	0.712	0.727	0.686	0.698	0.673	0.654	0.791	0.76	0.773	0.76	0.72	0.686	0.672	0.727
	Native	0.734	0.743	0.693	0.723	0.696	0.662	0.636	0.676	0.778	0.756	0.731	0.747	0.729	0.7	0.69	0.719
	SVM	0.744	0.737	0.736	0.758	0.704	0.749	0.687	0.642	0.793	0.742	0.769	0.775	0.733	0.678	0.726	0.699
	KNN	0.733	0.692	0.688	0.725	0.597	0.522	0.617	0.68	0.772	0.703	0.744	0.741	0.672	0.639	0.659	0.669
	Tree	0.682	0.689	0.713	0.696	0.581	0.579	0.571	0.595	0.742	0.746	0.711	0.733	0.637	0.621	0.658	0.634
Recall	BayesNet	0.735	0.742	0.724	0.733	0.686	0.698	0.673	0.654	0.791	0.761	0.774	0.761	0.746	0.718	0.709	0.743
	Native	0.726	0.738	0.681	0.71	0.692	0.66	0.635	0.673	0.778	0.757	0.732	0.742	0.737	0.703	0.686	0.737
	SVM	0.752	0.742	0.74	0.766	0.704	0.748	0.686	0.642	0.791	0.739	0.768	0.775	0.734	0.678	0.718	0.698
	KNN	0.731	0.698	0.689	0.733	0.597	0.522	0.616	0.679	0.77	0.7	0.742	0.738	0.706	0.681	0.689	0.689
	Tree	0.698	0.7	0.726	0.703	0.579	0.579	0.553	0.591	0.742	0.745	0.711	0.734	0.672	0.653	0.672	0.678
F-Measure	BayesNet	0.726	0.739	0.715	0.729	0.686	0.698	0.673	0.654	0.79	0.76	0.772	0.76	0.726	0.695	0.683	0.732
	Native	0.729	0.74	0.686	0.714	0.689	0.659	0.634	0.671	0.778	0.756	0.731	0.743	0.733	0.702	0.688	0.725
	SVM	0.729	0.712	0.707	0.75	0.704	0.748	0.684	0.642	0.789	0.733	0.765	0.773	0.733	0.678	0.721	0.698
	KNN	0.732	0.694	0.688	0.727	0.597	0.522	0.616	0.678	0.771	0.701	0.743	0.739	0.683	0.655	0.671	0.677
	Tree	0.632	0.634	0.694	0.635	0.573	0.577	0.533	0.584	0.739	0.741	0.711	0.731	0.651	0.635	0.664	0.651

5.7.3.1 跨项目克隆创建时一致性预测

5.7.3.2 跨项目克隆变化时一致性预测

5.7.4 软件开发过程中的克隆代码一致性预测

在这个讨论中，我们比较克隆和变化预测来解决最后一个研究问题。子问题3：他们应该采用哪种机器学习技术作为他们的偏好？而且，开发人员如何执行这些克隆预测来实现实践中的最佳效果？

根本节将据上面两节的实验结果，并结合软件开发过程，帮助程序开发人员在实践中运用克隆代码一致性维护需求预测，从而回答本章提出的上面第三个研究问题。

从机器学习方法的角度上来看，本文使用机器学习方法均取得了不错的预测能力，预测结果具有很大的相似性，表明本文所选择的属性可以很好的表示克隆代码创建实例和变化实例，并不依赖于某个具体的机器学习方法。因此，程序开发人员可以根据自己的喜好或者情况选择合适的机器学习方法。当程序开发人员无法决策使用何种机器学习方法，本文强烈建议使用 SVM 机器学习方法。因为上面的两个小节的实验结果表明，SVM 在克隆创建时和变化时均取得了相对较好的预测能力。除此之外，本文还建议开发人员在克隆创建的时候选择使用于 KNN 和 Tree 模型，以及在克隆变化时使用 BayesNet 和 Native Bayes 方法，从而可以保证这些方法具有相同或更好的预测能力。对于属性集的选择，本文还建议开发人员应该选择并使用本文所提取的全部属性集，因为这些属性值在预测中起到了积极的作用。除此之外，本文还建议如果开发者能够保证获得有效的结果，开发人员可以探索更多属性和机器学习方法构建自己的预测。

通过对比克隆创建实例和克隆变化时的预测结果（表 5-2 和 表5-4），发现预测效果的好坏会依赖于具体的系统。具体来说，克隆创建实例的预测效果要好于克隆变化实例的预测效果，而在这两个预测中，系统 jEdit 就几乎都具有最差的预测结果。原因在于 jEdit 的克隆实例的规模太小，不能对其自身的预测模型进行较好的训练，从而导致预测效果最差。因此，本文建议开发人员在进行克隆创建和变化实例时，需要在软件演化一定版本后收集到足够的训练数据后再进行训练和预测。在软件开发初始阶段，由于系统中缺少训练数据，可以使用跨项目预测的方法解决该问题，但需要对跨项目预测做进一步的研究，从而达到较好的预测效果。

为了结合软件开发过程，本文还从软件开发的不同角度给出相关的建议，让开发人员在实践中应用克隆代码一致性维护需求时实现最佳的预测效果。

在软件开发的初始阶段，主要任务是快速的进行软件开发，并同时向系统中添

加新的功能。这种特点可能会促使程序开发人员大量的进行复制粘贴操作，复用既有的代码进行快速开发。因此，本文建议开发人员在软件开发的初始阶段使用克隆代码创建时的一致性维护预测方法。进一步地，由于要向软件中引入新的功能，因此，建议使用该模型预测克隆代码的一致性维护自由的克隆，仅仅避免那些可能会引发一致性变化的克隆代码，进而快速的开发软件。但是，由于在最初始的阶段，软件系统可能仅仅具有几个版本，因此系统中的克隆创建实例不够多，导致预测数据不足的数据不足，无法完全的训练预测模型。为了解决这个问题，可以使用跨项目预测方法用于克隆代码创建时的预测，见本文第三章中的跨项目预测部分。

在开发的中间阶段，主要任务是添加新功能并修复包括克隆一致性缺陷在内的暴露缺陷。因此，我们建议开发人员应该同时考虑两者来预测克隆创建实例和克隆变化实例。在大多数开发任务的软件演进之后，可能有足够的克隆克隆实例的培训数据和用于更改实例的训练数据不足。这些足够的数据可以训练克隆预测器，以帮助开发人员关心克隆一致性。对于变化的预测，开发人员应该在 work [102] 中描述的跨项目预测中建立和训练模型。

在开发的最后阶段，主要任务是维护可能导致大量克隆更改一致性的软件。因此，我们建议开发人员对克隆更改实例执行克隆一致性预测。由于软件的不断发展，其项目克隆更改实例的数据将足以对预测器进行良好的训练。

但是，这种方法只能预测变化的一致性要求，不能帮助开发人员维护克隆更改的一致性。为了实现克隆一致性的目标，我们还建议开发人员采取措施，通过一致性维护和管理技术（如 [101] 和 [106]）执行克隆一致的更改。

5.8 结论

在软件开发过程中，克隆创建实例（复制粘贴操作）和克隆变化实例（克隆修改操作）可能会导致克隆代码在演化过程中的一致性变化，从而可能增加软件维护的代价。因此，本章统一了克隆创建和变化时的一致性维护需求定义，并结合软件开发过程对克隆代码一致性维护需求预测进行了实证研究。为帮助程序开发人员在实际应用中预测克隆代码的一致性维护，本章结合软件开发过程提出了一个研究问题，并使用五种不同的机器学习方法预测克隆代码的一致性维护需求；同时设计和开发了了一个克隆代码一致性预测插件，可以帮助程序开发人员在实际的开发过程中预测克隆代码的一致性，切实的提高软件的可维护性和软件质量。在四个开源系统上分别在克隆代码创建时和变化时进行了实验分组，从三个不同的角度回答了本文所提出的研究问题。实验结果表明使用不同的机器学习方法所构建

的预测模型在克隆一致性维护需求问题上具有相一致的预测效果，可以以高效地预测克隆创建实例一致性且有效地预测克隆变化的一致性问题。尽管如此，本文建议开发人员采用 SVM 方法进行预测，因为 SVM 在两种时刻均具最佳的预测能力。同时，为帮助软件开发人员在实际中对克隆一致性进行预测，本文结合软件开发过程在不同的软件开发阶段给出了不同的克隆一致性预测的建议。最后，本文结合实际的软件开发过程，基于 eclipse 实现了一个克隆代码一致性维护插件，可以在软件开发过程中帮助程序开发人员预测一致性，帮助降低软件维护代价，提高可维护性。

结　论

软件中克隆代码随着系统演化，克隆代码的存在以及在演化过程中的变化，会导致软件难以理解和维护，已经成为了影响软件质量的一个重要因素。现有的研究方法中既不能客观全面的理解克隆代码，也不能有效地解决克隆代码一致性变化问题。针对以上问题，本文提出了基于软件演化和机器学习的克隆代码分析与一致性维护方法，重点研究了克隆代码演化特征分析、克隆代码创建时和变化时的一致性需求预测，并结合软件开发过程中对克隆代码一致性预测进行实证研究，取得了如下创新性成果：

(1) 针对现有的克隆分析方法不能全面客观地理解克隆代码及其演化过程，提出了一个基于聚类的克隆代码演化特征分析方法。方法通过检测软件版本中的克隆代码，并构建系统克隆家系描述克隆代码的演化过程。为表示克隆代码及其演化情况，从克隆片段、克隆组和克隆家系三个不同的角度提取相应的属性值。在此基础上使用 X-means 聚类分析和挖掘克隆代码的演化特征。实验表明软件中存在的大部分克隆代码在其演化过程中是较为稳定的，并不会发生频繁的变化。同时，也存在一定规模的发生变化的克隆代码，其中发生一致性变化的克隆要多于不一致变化的克隆代码。

(2) 针对复制粘贴创建的克隆代码在其演化过程可能会引发一致性变化，从而导致额外的维护代价问题，提出了一个基于贝叶斯网络的克隆代码创建时一致性维护需求预测方法。提供了一个克隆代码创建时的一致性维护需求定义，将问题转化为一个可以使用机器学习解决的分类问题。通过构建系统克隆家系并识别克隆演化模式，从系统中收集克隆创建实例用于训练贝叶斯网络模型，并提取代码属性表示被复制的克隆代码、提取上下文属性表示被粘贴的克隆代码。在四个实验系统的实验表明，该方法可以以较高的准确率和召回率高效地预测克隆代码创建时的一致性维护需求。

(3) 针对演化过程中的克隆代码的变化会导致其所在克隆组在演化过程中的一致性变化，从而可能引发的一致性缺陷问题，提出了一个基于贝叶斯网络的克隆代码变化时一致性维护需求预测方法。提供了一个克隆代码变化时的一致性维护需求定义，将问题装化为可以应用机器学习方法解决的分类问题。通过构建系统的克隆家系和识别其演化模式，从系统中收集克隆变化实例用于训练贝叶斯网络模型。从克隆组的角度提取了代码属性、上下文属性，并同时从克隆演化的角度提取了

一组演化属性共同表示克隆变化实例。在四个实验系统的实验表明，该方法可以以合理的准确率和召回率有效地预测克隆代码变化使得一致性维护需求。

(4) 针对如何结合软件开发过程并帮助程序开发人员选择合适的机器学习方法对克隆代码进行一致性维护的问题，对克隆代码的一致性维护需求进行了一个实证研究。从实际应用的角度提出一个克隆一致性预测的研究问题，并统一克隆创建和变化时的一致性维护需求为克隆代码一致性维护需求定义。结合软件开发过程，基于 eclipse 设计并实现了一个克隆代码一致性维护需求预测插件，可以在实际开发中帮助程序开发人员预测克隆代码的一致性维护需求。并使用五种不同的机器学习方法预测克隆代码的一致性维护需求，从而给出在实际开发过程中如何预测克隆代码一致性维护需求。该方法可以在软件开发过程中帮助程序开发人员边开发、边预测克隆代码的一致性维护需求。

在本文工作的基础上，还有以下工作有待于进一步研究：

(1) 在克隆变化时一致性维护需求预测中，仍然有一定的改进空间。在未来研究中，拟改进现有的属性值用于表示克隆代码变化实例，从而以更高的准确率和召回率更加高效的对克隆代码的变化进行一致性维护需求预测。

(2) 本文方法在对跨项目的克隆一致性维护需求预测时，现有方法的预测能力远远达不到令人满意的效果。在未来的研究中，拟通过添加与项目自身相关的属性用于跨项目的克隆代码一致性维护需求预测中，从而帮助缺乏训练数据的新系统进行一致性维护需求预测。

(3) 在软件开发过程中，本文方法仅可以警告程序开发人员克隆代码的一致性维护需求，并不能切实的对克隆代码进行一致性维护。在未来的研究中，拟结合程序分析技术在需要时对克隆代码自动地进行一致性维护，保证克隆代码的一致性。

参考文献

[1] ROY C K, CORDY J R. A survey on software clone detection research[J]. Queen’s School of Computing TR, 2007, 541(115): 64–68.

[2] BAKER B S. On finding duplication and near-duplication in large software systems[C] // Reverse Engineering, 1995., Proceedings of 2nd Working Conference on. 1995: 86–95.

[3] KONTOGIANNIS K A, DEMORI R, MERLO E, et al. Pattern matching for clone and concept detection[G] // Reverse engineering. [S.l.]: Springer, 1996: 77–108.

[4] LAGUE B, PROULX D, MAYRAND J, et al. Assessing the benefits of incorporating function clone detection in a development process[C] // Software Maintenance, 1997. Proceedings., International Conference on. 1997: 314–321.

[5] DUCASSE S, RIEGER M, DEMEYER S. A language independent approach for detecting duplicated code[C] // Software Maintenance, 1999.(ICSM’99) Proceedings. IEEE International Conference on. 1999: 109–118.

[6] FOWLER M. Refactoring: improving the design of existing code[M]. [S.l.]: Pearson Education India, 2009.

[7] JUERGENS E, DEISSENBOECK F, HUMMEL B, et al. Do code clones matter?[C] // Proceedings of the 31st International Conference on Software Engineering. 2009: 485–495.

[8] GAUTHIER F, LAVOIE T, MERLO E. Uncovering access control weaknesses and flaws with security-discordant software clones[C] // Proceedings of the 29th Annual Computer Security Applications Conference. 2013: 209–218.

[9] WAGNER S, ABDULKHALEQ A, KAYA K, et al. On the relationship of inconsistent software clones and faults: an empirical study[C] // Software Analysis, Evolution, and Reengineering (SANER), 2016 IEEE 23rd International Conference on: Vol 1. 2016: 79–89.

[10] KAPSER C J, GODFREY M W. “Cloning considered harmful” considered harmful: patterns of cloning in software[J]. Empirical Software Engineering, 2008, 13(6): 645.

[11] SELIM G M, BARBOUR L, SHANG W, et al. Studying the impact of clones on

software defects[C] // Reverse Engineering (WCRE), 2010 17th Working Conference on. 2010 : 13 – 21.

[12] WANG X, DANG Y, ZHANG L, et al. Can I clone this piece of code here?[C] // Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. 2012 : 170 – 179.

[13] ROY C K, CORDY J R. NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization[C] // Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on. 2008 : 172 – 181.

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

[15] JIANG L, MISHERGHI G, SU Z, et al. Deckard: Scalable and accurate tree-based detection of code clones[C] // Proceedings of the 29th international conference on Software Engineering. 2007 : 96 – 105.

[16] KIM M, SAZAWAL V, NOTKIN D, et al. An empirical study of code clone genealogies[C] // ACM SIGSOFT Software Engineering Notes : Vol 30. 2005 : 187 – 196.

[17] SAHA R K, ROY C K, SCHNEIDER K A. An automatic framework for extracting and classifying near-miss clone genealogies[C] // Software Maintenance (ICSM), 2011 27th IEEE International Conference on. 2011 : 293 – 302.

[18] GÖDE N, KOSCHKE R. Frequency and risks of changes to clones[C] // Proceedings of the 33rd International Conference on Software Engineering. 2011 : 311 – 320.

[19] MONDAL M, ROY C K, SCHNEIDER K A. Dispersion of changes in cloned and non-cloned code[C] // Proceedings of the 6th International Workshop on Software Clones. 2012 : 29 – 35.

[20] RAHMAN M S, ROY C K. A Change-Type Based Empirical Study on the Stability of Cloned Code[C] // Source Code Analysis and Manipulation (SCAM), 2014 IEEE 14th International Working Conference on. 2014 : 31 – 40.

[21] KRINKE J. A study of consistent and inconsistent changes to code clones[C] // 14th working conference on reverse engineering (WCRE 2007). 2007 : 170 – 178.

[22] AVERSANO L, CERULO L, DI PENTA M. How clones are maintained: An em-

pirical study[C] // 11th European Conference on Software Maintenance and Reengineering (CSMR'07). 2007 : 81 – 90.

[23] KOSCHKE R. Survey of research on software clones[C] // Dagstuhl Seminar Proceedings. 2007.

[24] ROY C K, ZIBRAN M F, KOSCHKE R. The vision of software clone management: Past, present, and future (keynote paper)[C] // Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on. 2014 : 18 – 33.

[25] WETTEL R, MARINESCU R. Archeology of code duplication: Recovering duplication chains from small duplication fragments[C] // Symbolic and Numeric Algorithms for Scientific Computing, 2005. SYNASC 2005. Seventh International Symposium on. 2005 : 8 – pp.

[26] LEE S, JEONG I. SDD: high performance code clone detection system for large scale source code[C] // Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. 2005 : 140 – 141.

[27] LI Z, LU S, MYAGMAR S, et al. CP-Miner: Finding copy-paste and related bugs in large-scale software code[J]. IEEE Transactions on software Engineering, 2006, 32(3) : 176 – 192.

[28] GÖDE N, KOSCHKE R. Incremental clone detection[C] // Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on. 2009 : 219 – 228.

[29] BAXTER I D, YAHIN A, MOURA L, et al. Clone detection using abstract syntax trees[C] // Software Maintenance, 1998. Proceedings., International Conference on. 1998 : 368 – 377.

[30] BULYCHEV P, MINEA M. Duplicate code detection using anti-unification[C] // Spring Young Researchers Colloquium on Software Engineering. 2008 : 51 – 54.

[31] KRINKE J. Identifying similar code with program dependence graphs[C] // Reverse Engineering, 2001. Proceedings. Eighth Working Conference on. 2001 : 301 – 309.

[32] BASIT H A, JARZABEK S. A data mining approach for detecting higher-level

clones in software[J]. IEEE Transactions on Software engineering, 2009, 35(4): 497–514.

[33] BELLON S, KOSCHKE R, ANTONIOL G, et al. Comparison and evaluation of clone detection tools[J]. IEEE Transactions on software engineering, 2007, 33(9).

[34] RATTAN D, BHATIA R, SINGH M. Software clone detection: A systematic review[J]. Information and Software Technology, 2013, 55(7): 1165–1199.

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

[36] SVAJLENKO J, ROY C K. Evaluating modern clone detection tools[C] // Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on. 2014: 321–330.

[37] GABEL M, JIANG L, SU Z. Scalable detection of semantic clones[C] // Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on. 2008: 321–330.

[38] MAYRAND J, LEBLANC C, MERLO E. Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics.[C] // icsm: Vol 96. 1996: 244.

[39] ANTONIOL G, PENTA M D, CASAZZA G, et al. Modeling clones evolution through time series[C] // Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01). 2001: 273.

[40] BAKOTA T. Tracking the evolution of code clones[C] // International Conference on Current Trends in Theory and Practice of Computer Science. 2011: 86–98.

[41] HARDER J, GÖDE N. Modeling clone evolution[J]. Proc. IWSC, 2009: 17–21.

[42] CAI D, KIM M. An empirical study of long-lived code clones[C] // International Conference on Fundamental Approaches to Software Engineering. 2011: 432–446.

[43] KRINKE J. Is cloned code older than non-cloned code?[C] // Proceedings of the 5th International Workshop on Software Clones. 2011: 28–33.

[44] BAZRAFSHAN S. Evolution of near-miss clones[C] // Source Code Analysis and

Manipulation (SCAM), 2012 IEEE 12th International Working Conference on. 2012 : 74 – 83.

[45] GÖDE N. Evolution of type-1 clones[C] // Source Code Analysis and Manipulation, 2009. SCAM'09. Ninth IEEE International Working Conference on. 2009 : 77 – 86.

[46] KRINKE J. Is cloned code more stable than non-cloned code?[C] // Source Code Analysis and Manipulation, 2008 Eighth IEEE International Working Conference on. 2008 : 57 – 66.

[47] GODE N, HARDER J. Clone stability[C] // Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on. 2011 : 65 – 74.

[48] HARDER J, GÖDE N. Cloned code: stable code[J]. Journal of Software: Evolution and Process, 2013, 25(10) : 1063 – 1088.

[49] MONDAL M, ROY C K, RAHMAN M S, et al. Comparative stability of cloned and non-cloned code: An empirical study[C] // Proceedings of the 27th Annual ACM Symposium on Applied Computing. 2012 : 1227 – 1234.

[50] MONDAL M, ROY C K, SCHNEIDER K A. A comparative study on the intensity and harmfulness of late propagation in near-miss code clones[J]. Software Quality Journal, 2016, 24(4) : 883 – 915.

[51] BETTENBURG N, SHANG W, IBRAHIM W, et al. An empirical study on inconsistent changes to code clones at release level[C] // 2009 16th Working Conference on Reverse Engineering. 2009 : 85 – 94.

[52] ELISH M O, AL-GHAMDI Y. Fault density analysis of object-oriented classes in presence of code clones[C] // Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering. 2015 : 10.

[53] LO D, JIANG L, BUDI A, et al. Active refinement of clone anomaly reports[C] // Proceedings of the 34th International Conference on Software Engineering. 2012 : 397 – 407.

[54] KAMEI Y, SATO H, MONDEN A, et al. An empirical study of fault prediction with code clone metrics[C] // Software Measurement, 2011 Joint Conference of the 21st Int'l Workshop on and 6th Int'l Conference on Software Process and Product Measurement (IWSM-MENSURA). 2011 : 55 – 61.

[55] HARDER J, TIARKS R. A controlled experiment on software clones[C] // Program

Comprehension (ICPC), 2012 IEEE 20th International Conference on. 2012 : 219 – 228.

[56] JUERGENS E, DEISSENBOECK F. How much is a clone[C] // Proceedings of the 4th International Workshop on Software Quality and Maintainability. 2010.

[57] MONDEN A, NAKAE D, KAMIYA T, et al. Software quality analysis by code clones in industrial legacy software[C] // Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on. 2002 : 87 – 94.

[58] KAPSER C, GODFREY M W. ” Cloning considered harmful” considered harmful[C] // 2006 13th Working Conference on Reverse Engineering. 2006 : 19 – 28.

[59] RAHMAN F, BIRD C, DEVANBU P. Clones: What is that smell?[J]. Empirical Software Engineering, 2012, 17(4-5) : 503 – 530.

[60] HIGO Y, SAWA K-I, KUSUMOTO S. Problematic code clones identification using multiple detection results[C] // Software Engineering Conference, 2009. APSEC’09. Asia-Pacific. 2009 : 365 – 372.

[61] HORDIJK W, PONISIO M L, WIERINGA R. Harmfulness of code duplication-a structured review of the evidence[J], 2009.

[62] KERIEVSKY J. 重构与模式 [J]. 程序员, 2006, 3 : 047.

[63] LIN Y, XING Z, XUE Y, et al. Detecting differences across multiple instances of code clones[C] // Proceedings of the 36th International Conference on Software Engineering. 2014 : 164 – 174.

[64] MENDE T, KOSCHKE R, BECKWERMERT F. An evaluation of code similarity identification for the grow-and-prune model[J]. Journal of Software Maintenance and Evolution: Research and Practice, 2009, 21(2) : 143 – 169.

[65] SCHULZE S, KUHLEMANN M, ROSENMÜLLER M. Towards a refactoring guideline using code clone classification[C] // Proceedings of the 2nd Workshop on Refactoring Tools. 2008 : 6.

[66] CHOI E, YOSHIDA N, ISHIO T, et al. Extracting code clones for refactoring using combinations of clone metrics[C] // Proceedings of the 5th International Workshop on Software Clones. 2011 : 7 – 13.

[67] MANDAL M, ROY C K, SCHNEIDER K A. Automatic ranking of clones for refactoring through mining association rules[C] // Software Maintenance, Reengi-

neering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on. 2014 : 114 – 123.

[68] LEE S, BAE G, CHAE H S, et al. Automated scheduling for clone-based refactoring using a competent GA[J]. Software: Practice and Experience, 2011, 41(5) : 521 – 550.

[69] ZIBRAN M F, ROY C K. A constraint programming approach to conflict-aware optimal scheduling of prioritized code clone refactoring[C] // Source Code Analysis and Manipulation (SCAM), 2011 11th IEEE International Working Conference on. 2011 : 105 – 114.

[70] LIU H, MA Z, SHAO W, et al. Schedule of bad smell detection and resolution: A new way to save effort[J]. IEEE Transactions on Software Engineering, 2012, 38(1) : 220 – 235.

[71] VENKATASUBRAMANYAM R D, GUPTA S, SINGH H K. Prioritizing code clone detection results for clone management[C] // Proceedings of the 7th International Workshop on Software Clones. 2013 : 30 – 36.

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

[73] KRISHNAN G P, TSANTALIS N. Unification and refactoring of clones[C] // Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on. 2014 : 104 – 113.

[74] BARBOSA F S, AGUIAR A. Removing code duplication with roles[C] // Intelligent Software Methodologies, Tools and Techniques (SoMeT), 2013 IEEE 12th International Conference on. 2013 : 37 – 42.

[75] GÖDE N. Clone removal: Fact or fiction?[C] // Proceedings of the 4th International Workshop on Software Clones. 2010 : 33 – 40.

[76] ZIBRAN M F, SAHA R K, ROY C K, et al. Evaluating the conventional wisdom in clone removal: a genealogy-based empirical study[C] // Proceedings of the 28th Annual ACM Symposium on Applied Computing. 2013 : 1123 – 1130.

[77] EUNJONG C, YOSHIDA N, INOUE K. An investigation into the characteristics

of merged code clones during software evolution[J]. IEICE TRANSACTIONS on Information and Systems, 2014, 97(5): 1244–1253.

[78] TAIRAS R, GRAY J. Sub-clones: Considering the Part Rather than the Whole.[C] // Software Engineering Research and Practice. 2010: 284–290.

[79] ALI A-F M, SULAIMAN S. Enhancing Generic Pipeline Model in Preventing Code Clone during Software Development[C] // e-Proceeding of Software Engineering Postgraduates Workshop (SEPoW). 2013: 56.

[80] WANG X, DANG Y, ZHANG L, et al. Predicting Consistency-Maintenance Requirement of Code Clones at Copy-and-Paste Time.[J], 2014: 773–794.

[81] VENKATASUBRAMANYAM R D, SINGH H K, RAVIKANTH K. A method for proactive moderation of code clones in IDEs[C] // Software Clones (IWSC), 2012 6th International Workshop on. 2012: 62–66.

[82] KRUTZ D E, LE W. A code clone oracle[C] // Proceedings of the 11th Working Conference on Mining Software Repositories. 2014: 388–391.

[83] ISHIHARA T, HOTTA K, HIGO Y, et al. Reusing reused code[C] // 2013 20th Working Conference on Reverse Engineering (WCRE). 2013: 457–461.

[84] OHTA T, MURAKAMI H, IGAKI H, et al. Source code reuse evaluation by using real/potential copy and paste[C] // Software Clones (IWSC), 2015 IEEE 9th International Workshop on. 2015: 33–39.

[85] YANG J, HOTTA K, HIGO Y, et al. Classification model for code clones based on machine learning[J]. Empirical Software Engineering, 2015, 20(4): 1095–1125.

[86] OHTANI A, HIGO Y, ISHIHARA T, et al. On the level of code suggestion for reuse[C] // Software Clones (IWSC), 2015 IEEE 9th International Workshop on. 2015: 26–32.

[87] KINTAB G A, ROY C K, MCCALLA G I. Recommending software experts using code similarity and social heuristics[C] // Proceedings of 24th Annual International Conference on Computer Science and Software Engineering. 2014: 4–18.

[88] ISHIHARA T, HOTTA K, HIGO Y, et al. Inter-project functional clone detection toward building libraries-an empirical study on 13,000 projects[C] // Reverse Engineering (WCRE), 2012 19th Working Conference on. 2012: 387–391.

[89] CHENG X, JIANG L, ZHONG H, et al. On the feasibility of detecting cross-

platform code clones via identifier similarity[C] // Proceedings of the 5th International Workshop on Software Mining. 2016 : 39 – 42.

[90] TAIRAS R, GRAY J. An information retrieval process to aid in the analysis of code clones[J]. Empirical Software Engineering, 2009, 14(1) : 33 – 56.

[91] KOSCHKE R. Frontiers of software clone management[C] // Frontiers of Software Maintenance, 2008. FoSM 2008.. 2008 : 119 – 128.

[92] KOSCHKE R, BAXTER I D, CONRADT M, et al. Software clone management towards industrial application (dagstuhl seminar 12071)[J]. Dagstuhl Reports, 2012, 2(2).

[93] DE WIT M, ZAIDMAN A, VAN DEURSEN A. Managing code clones using dynamic change tracking and resolution[C] // Software Maintenance, 2009. ICSM 2009. IEEE International Conference on. 2009 : 169 – 178.

[94] HOU D, JABLONSKI P, JACOB F. CnP: Towards an environment for the proactive management of copy-and-paste programming[C] // Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on. 2009 : 238 – 242.

[95] WECKERLE V. CPC: an eclipse framework for automated clone life cycle tracking and update anomaly detection[J]. Master's thesis, Freie Universität Berlin, Germany, 2008.

[96] JABLONSKI P, HOU D. CReN: a tool for tracking copy-and-paste code clones and renaming identifiers consistently in the IDE[C] // Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange. 2007 : 16 – 20.

[97] JACOB F, HOU D, JABLONSKI P. Actively comparing clones inside the code editor[C] // Proceedings of the 4th International Workshop on Software Clones. 2010 : 9 – 16.

[98] DUALA-EKOKO E, ROBILLARD M P. Clonetracker: tool support for code clone management[C] // Proceedings of the 30th international conference on Software engineering. 2008 : 843 – 846.

[99] DUALA-EKOKO E, ROBILLARD M P. Clone region descriptors: Representing and tracking duplication in source code[J]. ACM Transactions on Software Engineering and Methodology (TOSEM), 2010, 20(1) : 3.

[100] YAMANAKA Y, CHOI E, YOSHIDA N, et al. Applying clone change notifica-

tion system into an industrial development process[C] // Program Comprehension (ICPC), 2013 IEEE 21st International Conference on. 2013 : 199 – 206.

[101] CHENG X, ZHONG H, CHEN Y, et al. Rule-directed code clone synchronization[C] // Program Comprehension (ICPC), 2016 IEEE 24th International Conference on. 2016 : 1 – 10.

[102] ZHANG F, KHOO S-C, SU X. Predicting Consistent Clone Change[C] // Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on. 2016 : 353 – 364.

[103] MONDAL M, ROY C K, SCHNEIDER K A. Prediction and ranking of co-change candidates for clones[C] // Proceedings of the 11th Working Conference on Mining Software Repositories. 2014 : 32 – 41.

[104] MURAKAMI H, HOTTA K, HIGO Y, et al. Predicting Next Changes at the Fine-Grained Level[C] // Software Engineering Conference (APSEC), 2014 21st Asia-Pacific : Vol 1. 2014 : 119 – 126.

[105] ZHANG G, PENG X, XING Z, et al. Towards contextual and on-demand code clone management by continuous monitoring[C] // Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on. 2013 : 497 – 507.

[106] NGUYEN H A, NGUYEN T T, PHAM N H, et al. Clone management for evolving software[J]. IEEE Transactions on Software Engineering, 2012, 38(5) : 1008 – 1026.

[107] TAIRAS R, GRAY J. Increasing clone maintenance support by unifying clone detection and refactoring activities[J]. Information and Software Technology, 2012, 54(12) : 1297 – 1307.

[108] PELLEG D, MOORE A W, OTHERS. X-means: Extending K-means with Efficient Estimation of the Number of Clusters.[C] // ICML : Vol 1. 2000.

[109] CORDY J R. The TXL source transformation language[J]. Science of Computer Programming, 2006, 61(3) : 190 – 210.

[110] DEAN T R, CORDY J R, MALTON A J, et al. Agile parsing in TXL[J]. Automated Software Engineering, 2003, 10(4) : 311 – 336.

[111] CI M, SU X-H, WANG T-T, et al. A New Clone Group Mapping Algorithm for Extracting Clone Genealogy on Multi-version Software[C] // Instrumentation, Mea-

surement, Computer, Communication and Control (IMCCC), 2013 Third International Conference on. 2013 : 848 – 853.

[112] 慈萌. 基于 CRD 克隆群映射的克隆家系提取方法研究 [D]. [S.l.] : 哈尔滨: 哈尔滨工业大学, 2013.

[113] HALL M, FRANK E, HOLMES G, et al. The WEKA data mining software: an update[J]. ACM SIGKDD explorations newsletter, 2009, 11(1) : 10 – 18.

[114] ARTHUR D, VASSILVITSKII S. k-means++: The advantages of careful seeding[C] // Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms. 2007 : 1027 – 1035.

[115] LEVENSHTEIN V I. Binary codes capable of correcting deletions, insertions and reversals[C] // Soviet physics doklady : Vol 10. 1966 : 707.

[116] FRIEDMAN N, GEIGER D, GOLDSZMIDT M. Bayesian network classifiers[J]. Machine learning, 1997, 29(2-3) : 131 – 163.

[117] JOHN G H, LANGLEY P. Estimating continuous distributions in Bayesian classifiers[C] // Proceedings of the Eleventh conference on Uncertainty in artificial intelligence. 1995 : 338 – 345.

[118] PLATT J C. 12 fast training of support vector machines using sequential minimal optimization[J]. Advances in kernel methods, 1999 : 185 – 208.

[119] AHA D W, KIBLER D, ALBERT M K. Instance-based learning algorithms[J]. Machine learning, 1991, 6(1) : 37 – 66.

[120] QUINLAN J R. C4. 5: programs for machine learning[M]. [S.l.] : Elsevier, 2014.

攻读博士学位期间发表的论文及其他成果

（一）发表的学术论文

[1] Zhang Fanlong，Khoo Siau-Cheng，Su Xiaohong. Predicting Change Consistency in a Clone Group[J]. Journal of Systems and Software.（小修已修回，SCI 收录，CCF 推荐 B 类期刊，中科院 SCI 期刊分区 3 区，SCI 检索，IF=1.424，对应于第 4 章，第一作者）

[2] Zhang Fanlong，Khoo Siau-Cheng，Su Xiaohong. Predicting Consistent Clone Change[C]. Software Reliability Engineering (ISSRE), 2016 IEEE 27th International Symposium on. IEEE, 2016: 353-364.（EI:20170803379101，CCF 推荐 B 类会议，DOI:10.1109/ISSRE.2016.11，对应于第 4 章，第一作者）

[3] Zhang Fanlong，Khoo Siau-Cheng，Su Xiaohong. An empirical study on clone consistency-requirement prediction based on machine learning[J]. Journal of Computer Science and Technology.（在投，SCI 收录，CCF 推荐 B 类期刊，中科院 SCI 期刊分区 4 区，SCI 检索，IF=0.475，对应于第 5 章，第一作者）

[4] Zhang Fanlong，Khoo Siau-cheng, Su Xiaohong. Machine-Learning Aided Analysis of Clone Evolution[J]. Chinese Journal of Electronics. (录用待发表，SCI 收录，中科院分区四区，IF=0.319，对应于第 2 章，第一作者)

[5] Zhang Fanlong，Yuan yue, Khoo Siau-cheng, Su Xiaohong. Clone Creating Consistency Prediction Based on Bayesian Network[J]. Chinese Journal of Electronics. (在投，SCI 收录，中科院 SCI 期刊分区 4 区，IF=0.319，对应于第 3 章，第一作者)

[6] 苏小红，张凡龙. 面向管理的克隆代码研究综述 [J]. 计算机学报.（大修已修回，EI 收录，一级学报，对应于第 1 章，第二作者导师为第一作者）

[7] Zhang Fanlong ，Su Xiaohong，Zhao Wen，Ma Peijun. An empirical study of code clone clustering based on clone evolution[J]. Journal of Harbin Institute of Technology(New Series). (录用待发表，DOI：10.11916/j.issn.1005-9113.15316，工大学报英文版，对应于第 2 章，第一作者)

[8] Zhang Fanlong ，Su Xiaohong. An empirical study of clone consistency prediction[J]. Journal of Harbin Institute of Technology(New Series). (在投，工大学报英文版，对应于第 5 章，第一作者)

[9] Yuan Yue，Zhang Fanlong，Su Xiaohong. CloneAyz：An Approach for Clone Representation and Analysis[C]. Information Science and Control Engineering

(ICISCE), 2016 3rd International Conference on. IEEE, 2016: 252-256.（已发表，EI:20165003106894，DOI: 10.1109/ICISCE.2016.63，对应于第 4 章，第二作者）

[10] Zhang Fanlong，Su Xiaohong. CCP：An Plug-in for Clone Consistency Prediction[C] (在投，对应于第 5 章，第一作者)

[11] 张凡龙, 苏小红，李智超，马培军. 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): 112-115. (已发表，对应于第 3 章，第一作者)

[12] Su Xiaohong，Zhang Fanlong，Xia Li，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.（已发表，EI:20151600752325，DOI: 10.1007/978-81-322-1695-7_28，第二作者导师为第一作者）

（二）参与的科研项目及获奖情况

[1] 苏小红等. 无定型克隆代码的检测及重构方法. 国家自然科学基金项目. 课题编号：61173021.

[2] 苏小红等. 基于启发式选择变异和软件行为特征挖掘的软件错误定位方法. 国家自然科学基金项目. 课题编号：61672191.

[3] 苏小红等. 数据挖掘和静态分析相结合的克隆代码缺陷检测及重构方法. 国家自然科学基金项目. 课题编号：61073052.

[4] 苏小红等. 面向理解的软件错误定位方法. 国家自然科学基金项目. 课题编号：61202092.

哈尔滨工业大学学位论文原创性声明和使用权限

学位论文原创性声明

本人郑重声明：此处所提交的学位论文《基于机器学习的克隆代码分析与一致性维护方法研究》，是本人在导师指导下，在哈尔滨工业大学攻读学位期间独立进行研究工作所取得的成果，且学位论文中除已标注引用文献的部分外不包含他人完成或已发表的研究成果。对本学位论文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。

作者签名：　　日期：　　年　　月　　日

学位论文使用权限

学位论文是研究生在哈尔滨工业大学攻读学位期间完成的成果，知识产权归属哈尔滨工业大学。学位论文的使用权限如下：

（1）学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文，并向国家图书馆报送学位论文；（2）学校可以将学位论文部分或全部内容编入有关数据库进行检索和提供相应阅览服务；（3）研究生毕业后发表与此学位论文研究成果相关的学术论文和其他成果时，应征得导师同意，且第一署名单位为哈尔滨工业大学。

保密论文在保密期内遵守有关保密规定，解密后适用于此使用权限规定。

本人知悉学位论文的使用权限，并将遵守有关规定。

作者签名：　　日期：　　年　　月　　日

导师签名：　　日期：　　年　　月　　日

致　谢

在此论文完成之际，谨向给予我无私帮助和关怀的人们致以最诚挚的谢意！

衷心感谢我的导师 苏小红 教授！本文是在苏老师的悉心指导下完成的，几年来苏老师对我的科研工作给予了大力支持，也在日常生活等等方面给予了无微不至的关怀。在攻读博士期间，论文的选题、开题、中期以及博士论文撰写的各个阶段，苏老师给予了细致且精心的指导，在此向她表达最衷心的感谢，她的言传身教将使我终生受益。苏老师渊博的知识、远见的学术洞察力、严谨的治学态度和执着的敬业精神是我受益匪浅。没有苏老师的悉心指导和热情鼓励，我的博士论文工作不可能如此顺利的完成。在此，谨向恩师致以由衷的敬意和衷心的感谢！

衷心感谢新加坡国立大学的 Khoo Siau-Cheng 教授！在新加坡国立大学访学期间，Prof. Khoo 给予了我大力的支持，尤其是在学术了道路上给予我极大的帮助。感谢 Prof. Khoo

衷心感谢实验室全体老师和同窗们的热情帮助和支持！感谢王甜甜老师、张彦航老师、赵玲玲老师。感谢实验室的师兄师姐师弟师妹们。

衷心感谢我的女朋友王子萍！她始终陪在我身边，在我消沉时给我支持和鼓励，并帮助我润色修改博士论文。

衷心感谢我的父母！感谢他们将我抚养成人，尽力的创造最好的条件和资源让我接受最好的教育，是他们对我一直以来的支持、关心、理解和厚望，鼓励和激励着我全身心的投入学习，让我有了最坚实的后盾，在遇到困难时从而能有继续前行的决心、勇气和动力。

个人简历

1987 年 11 月 06 日出生于 山东省兖州市。

2006 年 09 月考入 东北林业大学 信息与工程学 院（系）计算机科学与技术 专业，2010 年 07 月本科毕业并获得 工 学学士学位。

2010 年 09 月——2012 年 07 月在 哈尔滨工业大学 大学 计算机科学与技术学 院（系）计算机科学与技术 学科学习并获得 工 学硕士学位。

2015 年 08 月——2016 年 08 月在 新加坡国立 大学 计算 学院（系）计算机科学 Visiting Student。

2012 年 09 月——至今在 哈尔滨工业大学 大学 计算机科学与技术学 院（系）计算机科学与技术 学科攻读博士学位。

主要研究领域为软件工程、程序分析，具体来说克隆代码分析与维护