

博士学位论文

基于软件演化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 SOFTWARE EVOLUTION**

张凡龙

哈尔滨工业大学

2017 年 6 月

国内图书分类号：TP311.5
国际图书分类号：004.41

学校代码：10213
密级：公开

工学博士学位论文

基于软件演化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6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 SOFTWARE EVOLUTION

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: June, 2017

Degree-Conferring-Institution: Harbin Institute of Technology

摘 要

克隆代码是软件系统中存在的彼此相似的代码片段。在软件开发过程中，开发人员通过复制粘贴既有代码会向系统中引入大量的克隆代码。克隆代码会随着时间和软件系统更新而进行演化，使软件系统变得越来越臃肿、难以维护，影响了软件的质量、可理解性和可维护性。这引发了一系列对克隆代码的研究，如克隆检测、克隆分析和克隆维护等。克隆检测帮助开发人员收集系统中的克隆代码，克隆分析帮助开发人员理解系统中存在的克隆代码，克隆维护帮助开发人员解决克隆代码已经引发或者可能引发的问题。因此，克隆代码研究对帮助提高软件系统的质量、增强软件系统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，克隆代码可能会被开发人员修改而发生变化，而进一步加剧克隆代码问题。演化中克隆代码的变化使得克隆代码难以理解，降低了软件的可理解性。克隆代码的一致性变化会导致额外的维护代价，而遗忘这种变化会导致一致性违背缺陷，降低了软件质量和可维护性。因此，本文基于软件演化研究克隆代码分析与一致性维护方法，通过提取分析克隆代码的演化特征，帮助软件开发人员理解克隆代码及其演化过程，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，并结合软件开发过程实现边开发、边维护克隆代码，提高软件质量、降低软件维护代价。

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，为克隆代码一致性维护需求研究奠定了基础。首先，使用克隆检测工具检测系统中的克隆代码，并构建克隆家系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维度提取相应的属性值描述克隆代码及其演化过程。最后，聚类系统中的克隆代码并挖掘克隆代码演化特征。实验结果发现大部分的克隆代码在演化过程中是稳定的，但也存在相当数量发生变化的克隆代码，其中发生一致性变化的克隆代码比率高于发生不一致性变化的克隆代码。

针对新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，研究并提出了克隆代码创建一致性维护需求预测方法。将系统中最早出现的克隆代码称为克隆创建实例，并将其在未来演化过程中所发生的一致性变化称为克隆代码创建一致性维护需求。首先，通过检测软件系统的克隆代码并构建其克隆家系，收集系统中新创建的克隆代码。然后，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。最后，使用机器学习方法训练预测模型，并在克隆代码创建时预测克隆代码的一致性维护需求。实验结果表明本文所提出的方法可以高效地预测克隆代码的一致性维护需求，可以帮助软件开发人员降低克隆代码的一致性维护代价。

针对演化中的克隆代码的一致性变化可能会导致克隆一致性违背缺陷问题，研究并提出了克隆代码变化一致性维护需求预测方法。本文将软件系统中发生变化的克隆代码称为克隆变化实例，并将由此变化引发的未来演化中的一致性变化称为克隆代码变化一致性维护需求。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，从克隆组的角度提取三组不同的属性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，使用机器学习模型训练预测模型，并在克隆代码发生变化时预测克隆代码变化的一致性维护需求。实验结果表明本文所提出的方法可以有效地预测克隆代码的一致性维护需求，可以帮助软件开发人员避免克隆代码一致性违背缺陷。

针对在软件开发初期软件系统中因数据不足而无法预测克隆一致性维护需求的问题，研究并提出了跨项目克隆代码一致性维护需求预测方法。将克隆代码创建、变化实例统称为克隆实例，并将相应的一致性维护需求统一为克隆代码一致性维护需求。首先，对不同的软件系统通过构建其克隆家系收集克隆实例，使用不同的属性表示克隆实例。然后，将不同的软件系统划分为训练系统和测试系统，使用训练系统的数据训练机器学习模型，在测试系统上验证跨项目克隆一致性维护需求的预测效果。实验结果表明跨项目的克隆一致性维护需求预测，可以在软件开发初期帮助预测克隆代码的一致性维护需求。最后，结合软件开发过程，设计并实现了一个 eclipse 插件预测克隆代码一致性维护需求，可以边开发、边预测克隆代码的一致性维护需求，有助于降低软件的维护代价。

综上所述，本文提出的基于软件演化的克隆代码分析与一致性维护方法，为在软件开发过程中解决克隆代码分析和理解、克隆代码一致性维护、避免克隆代码相关缺陷、降低克隆代码维护代价、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：实证软件工程；软件维护；软件演化；克隆代码；一致性维护

Abstract

Code clones are the similar code fragments to each other in software. During the software development, developers introduce a large number of code clones through reusing the existing code fragments from software. Code clone always evolving as the time and the update of software, and software become more and more bloated and difficult to maintain, and that affect the quality, comprehensibility, and maintainability of the software. Hence, there are a variety of code clone research, such as clone detection, clone analysis, clone maintenance and etc. Clone detection help developers gather the code clones from software, and clone analysis help developers understand the presence of code clones in software, and clone maintenance help developers solve the issues caused by code clones. Consequently, code clone research has great scientific theoretical significance and practical applicative value for improving the quality of the software, enhancing the comprehensibility, and maintainability of the software.

During the process of code clone evolution, code clones may be modified by developers that resulting the changes to clones, that will exacerbate the issue of code clones. The changes to the evolving code clone make it difficult to understand, and decrease the comprehensibility of the software. The consistent changes of code clones give rise to additional maintenance costs and clone consistency-defects, that thereby reducing software quality and maintainability. Therefore, this thesis studies on the code clone analysis and consistency maintenance based on software's evolution. Analyzing and extracting the clone evolutionary characteristics of code clones help the developers understand the code clones. Predicting the consistency-requirement of code clones solve the issues of code clone consistency maintenance. Our methods can help developers improve software quality, and reduce the cost of software maintenance through maintain the code clones at the development time.

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 lay the foundation for the prediction of clone consistency-requirement. We firstly detect all code clones with detection tool from software's repository, then build all the clone genealogies for 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, including clone fragment, clone group and clone genealogy. Finally, the clustering method is employed for excavating and analyzing the clone evolutionary characteristics from all the code clones and their evolution. The experimental results show that most of the code clones are stable during evolution, but there are also a significant number of code clones that occur changes. What's more, the percentage rate of clone codes that having consistent change is high than code clones that having inconsistent change.

Aiming to address the problem of additional maintenance cost caused by the clone consistent change in future evolution of a clone creating operation, we propose an approach for predicting clone creating consistency-requirement in this thesis. We call clone creation operation as “clone creating instance”, and call the consistent change that occurs in future evolution of thus creating instance as “clone creating consistency-requirement”. Firstly, we collect all the clone creating instances through detecting all code clones and building all clone genealogies from software's repository. And then, we extract code attribute set to represent the copied code clone and context attribute set for pasted code clones. Finally, the machine learning models are trained with the collection of clone creating instances, and are employed to predict clone creating consistency-requirement. The experimental results show that our approach can effectively predict the consistency-requirement for creating instances, that can help the developers to reduce the consistency maintenance cost of the code clone at clone creating time.

Aiming to address the problem of consistency-defect caused by the clone consistent change in their evolution, an approach for predicting clone changing consistency-requirement is proposed. In this thesis, we call the changing 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 from the perspective of clone group are extracted to represent the clone changing instance with three attribute sets including code attribute set, the context attribute set, and the evolutionary attribute set. Lastly, the machine learning models are trained with the collection of clone changing instances, and are employed to predict clone changing consistency-requirement. The experimental results show that the proposed approach can reasonable predict the consistency-requirement for changing instances, that

can help the developers avoid clone consistency-defects at clone changing time.

Aiming to the problem that the insufficient clone instances is not enough to predict clone consistency-requirement at the early software development phase, an empirical study on clone consistency-requirement prediction for cross-project is constructed in this thesis. We unify the clone creating-instances and changing-instances as “clone instances”, and unify the corresponding consistency-requirement as “clone consistency-requirement”. Firstly, for the different software repositories, we collect their clone instances by detecting code clones and building clone genealogist from software repository, and represent all instance with different attribute sets. Then, these different softwares are divided into training softwares and testing softwares. We employ the data from training softwares to train the machine learning models, and verify the ability of cross-project clone consistency-requirement on the testing softwares. Finally, combining with the software development, we develop and implement an eclipse plug-in for predicting clone consistency requirement, that can help maintain the code clone at the developing time, that thereby reducing the cost of software maintenance. The experimental results show that the cross-project prediction can be employed to predict clone consistency requirement at the early stage of software development. Our approach can help to maintain code clones at the development phase that reducing the cost of software maintenance.

In summary, this thesis presents the code clone analysis and consistency maintenance approach based on software evolution, that providing 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 during at the software development phase.

Keywords: Empirical software engineering; software maintenance; software evolution; code clones; consistency maintenance

目 录

| | |
|--------------------------------------|-----|
| 摘 要..... | I |
| ABSTRACT | III |
| 第 1 章 绪论 | 1 |
| 1.1 课题背景及研究目的和意义..... | 1 |
| 1.2 国内外研究现状及其分析 | 3 |
| 1.2.1 研究热点与趋势..... | 3 |
| 1.2.2 克隆代码检测研究 | 6 |
| 1.2.3 克隆代码分析研究 | 9 |
| 1.2.4 克隆代码维护研究 | 13 |
| 1.2.5 当前研究中存在的问题分析 | 16 |
| 1.3 研究内容与论文结构..... | 17 |
| 1.3.1 研究内容..... | 17 |
| 1.3.2 论文结构..... | 18 |
| 第 2 章 基于 X-means 聚类的克隆代码演化特征分析 | 20 |
| 2.1 引言 | 20 |
| 2.2 克隆代码演化特征分析 | 20 |
| 2.2.1 现有研究存在的问题 | 20 |
| 2.2.2 本文的解决思路..... | 22 |
| 2.3 克隆代码演化以及相关定义..... | 23 |
| 2.4 软件克隆家系的构建..... | 26 |
| 2.4.1 克隆检测与 CRD 描述 | 26 |
| 2.4.2 克隆家系构建与克隆模式识别..... | 27 |
| 2.5 克隆演化实体的特征描述 | 29 |
| 2.5.1 克隆片段实体的特征 | 29 |
| 2.5.2 克隆组实体的特征 | 30 |
| 2.5.3 克隆家系实体的特征 | 30 |

| | |
|--------------------------------|----|
| 2.6 克隆演化特征挖掘 | 31 |
| 2.6.1 克隆实体聚类向量的生成 | 32 |
| 2.6.2 基于 X-means 的克隆实体聚类 | 32 |
| 2.7 实验结果与分析 | 35 |
| 2.7.1 实验设置..... | 35 |
| 2.7.2 克隆片段演化特征分析实验 | 36 |
| 2.7.3 克隆组演化特征分析实验 | 37 |
| 2.7.4 克隆家系演化特征分析实验 | 40 |
| 2.8 本章小结 | 46 |
| 第 3 章 克隆代码创建一致性维护需求预测方法 | 47 |
| 3.1 引言 | 47 |
| 3.2 克隆代码额外维护代价的相关研究..... | 47 |
| 3.2.1 克隆代码的维护代价问题 | 47 |
| 3.2.2 现有方法存在的问题 | 49 |
| 3.2.3 本文的解决思路..... | 50 |
| 3.3 克隆创建一致性维护需求的定义 | 52 |
| 3.4 复制和粘贴操作的样本获取..... | 54 |
| 3.5 复制粘贴操作的特征描述 | 55 |
| 3.5.1 被复制克隆代码的特征..... | 56 |
| 3.5.2 被粘贴克隆代码的特征..... | 56 |
| 3.6 基于机器学习的克隆创建一致性维护需求预测 | 57 |
| 3.7 实验结果与分析 | 60 |
| 3.7.1 实验设置..... | 60 |
| 3.7.2 有效性验证实验..... | 63 |
| 3.7.3 使用模式实验 | 65 |
| 3.7.4 与其它方法的对比 | 70 |
| 3.7.5 讨论 | 73 |
| 3.8 本章小结 | 74 |
| 第 4 章 克隆代码变化一致性维护需求预测方法 | 75 |
| 4.1 引言 | 75 |
| 4.2 克隆代码一致性维护的相关研究 | 75 |
| 4.2.1 克隆变化对软件质量影响问题..... | 75 |
| 4.2.2 现有研究存在的问题 | 78 |

| | |
|---------------------------------------|------------|
| 4.2.3 本文的解决思路..... | 79 |
| 4.3 克隆变化一致性维护需求的定义 | 80 |
| 4.4 克隆变化实例的获取 | 83 |
| 4.5 克隆变化实例的特征描述 | 85 |
| 4.5.1 代码属性特征 | 85 |
| 4.5.2 上下文属性特征..... | 86 |
| 4.5.3 演化属性特征 | 87 |
| 4.6 模型的构建与预测 | 88 |
| 4.7 实验结果与分析 | 89 |
| 4.7.1 实验设置..... | 89 |
| 4.7.2 有效性验证实验..... | 92 |
| 4.7.3 使用模式实验 | 95 |
| 4.7.4 讨论 | 100 |
| 4.8 本章小结 | 102 |
| 第 5 章 跨项目克隆代码一致性维护需求预测研究 | 103 |
| 5.1 引言 | 103 |
| 5.2 跨项目克隆一致性维护问题..... | 103 |
| 5.2.1 问题的提出 | 103 |
| 5.2.2 本文的解决思路..... | 104 |
| 5.3 克隆一致性维护需求的定义..... | 105 |
| 5.4 跨项目数据集的获取..... | 107 |
| 5.4.1 跨项目数据来源..... | 107 |
| 5.4.2 属性特征与数据集生成..... | 108 |
| 5.5 跨项目模型的构建与预测 | 110 |
| 5.6 实验结果与分析 | 111 |
| 5.6.1 实验设置..... | 111 |
| 5.6.2 跨项目有效性验证实验..... | 113 |
| 5.6.3 跨项目训练集规模实验..... | 116 |
| 5.6.4 跨项目使用模式实验 | 118 |
| 5.7 克隆一致性维护需求预测插件的设计与实现 | 127 |
| 5.7.1 克隆一致性维护需求预测插件的总体框架 | 127 |
| 5.7.2 克隆一致性维护需求预测插件的功能模块设计 | 128 |
| 5.7.3 克隆一致性维护需求预测插件的实现..... | 131 |

| | |
|----------------------------|-----|
| 5.8 本章小结 | 132 |
| 结 论..... | 135 |
| 参考文献..... | 137 |
| 攻读博士学位期间发表的论文及其他成果..... | 150 |
| 哈尔滨工业大学学位论文原创性声明和使用权限..... | 152 |
| 致 谢..... | 153 |
| 个人简历..... | 154 |

Contents

| | |
|---|-----|
| Abstract (In Chinese) | I |
| Abstract (In English) | III |
| | |
| Chapter 1 Introduction | 1 |
| 1.1 Background, Research Objective and Significance | 1 |
| 1.2 Related Work | 3 |
| 1.2.1 Research Hot-spots and Trends | 3 |
| 1.2.2 Code Clone Detection Research | 6 |
| 1.2.3 Code Clone Analysis Research | 9 |
| 1.2.4 Code Clone Maintenance Research | 13 |
| 1.2.5 Problems in Current Research | 16 |
| 1.3 Main Research Contents and Structure of This Dissertation | 17 |
| 1.3.1 Main Research Contents | 17 |
| 1.3.2 Structure of This Dissertation | 18 |
| | |
| Chapter 2 Analyzing Clone Evolutionary Characteristic Based on X-means | |
| Clustering | 20 |
| 2.1 Introduction | 20 |
| 2.2 Analyzing Clone Evolutionary Characteristic | 20 |
| 2.2.1 Problems in Current Research | 20 |
| 2.2.2 The Proposed Method | 22 |
| 2.3 Clone Evolution and its Related Definitions | 23 |
| 2.4 Constructing Clone Genealogy for Software Repository | 26 |
| 2.4.1 Clone Detection and Representation with CRD | 26 |
| 2.4.2 Building Clone Genealogy and Identifying Clone Evolutionary Pattern ... | 27 |
| 2.5 The Attributes for Three Clone Evolution Entities | 29 |
| 2.5.1 The Attributes for Clone Fragment Entity | 29 |
| 2.5.2 The Attributes for Clone Group Entity | 30 |
| 2.5.3 The Attributes for Clone Genealogy Entity | 30 |
| 2.6 Mining Clone Evolutionary Characteristics | 31 |

| | | |
|---|--|----|
| 2.6.1 | Generating the Clustering Vectors for Clone Entities | 32 |
| 2.6.2 | Clustering All Clone Entities with X-means Method | 32 |
| 2.7 | The Experimental Results and Analysis..... | 35 |
| 2.7.1 | Experimental Methodology | 35 |
| 2.7.2 | Experiment for Clone Fragment Evolutionary Characteristics..... | 36 |
| 2.7.3 | Experiment for Clone Group Evolutionary Characteristics | 37 |
| 2.7.4 | Experiment for Clone Genealogy Evolutionary Characteristics | 40 |
| 2.8 | Summary of this Chapter | 46 |
| Chapter 3 Predicting Clone Creating Consistency-Requirement at Copy-and-Paste Time | | |
| 3.1 | Introduction | 47 |
| 3.2 | Related Work for Code Clone of Extra Maintenance..... | 47 |
| 3.2.1 | The Problem of Maintenance Cost for Code Clones | 47 |
| 3.2.2 | Problems in Current Research | 49 |
| 3.2.3 | The Proposed Method | 50 |
| 3.3 | The Definitions for Clone Creating Consistency-Requirement | 52 |
| 3.4 | Collecting Copy-and-Paste Operations..... | 54 |
| 3.5 | The Attributes for Copied and Pasted Code Clones | 55 |
| 3.5.1 | The Attributes for the Copied Code Clone..... | 56 |
| 3.5.2 | The Attributes for Pasted Code Clone | 56 |
| 3.6 | Predicting Clone Creating Consistency-Requirement base on Machine Learning | 57 |
| 3.7 | Experimental Results and Analysis | 60 |
| 3.7.1 | Experimental Methodology | 60 |
| 3.7.2 | The Effectiveness Experiments for Employed Machine Learning Methods | 63 |
| 3.7.3 | The Experiments for Two Usage Scenarios | 65 |
| 3.7.4 | Comparing with Wang' s Method | 70 |
| 3.7.5 | Discussion | 73 |
| 3.8 | Summary of this Chapter | 74 |
| Chapter 4 Predicting Clone Changing Consistency-Requirement in A Clone Group | | |
| 4.1 | Introduction | 75 |
| 4.2 | Related Work for Code Clone Consistency-Maintenance | 75 |

Contents

| | |
|---|------------|
| 4.2.1 The Clone Change Effect for Software Quality | 75 |
| 4.2.2 Problems in Current Research | 78 |
| 4.2.3 The Proposed Method | 79 |
| 4.3 The Definitions for Clone Changing Consistency-Requirement..... | 80 |
| 4.4 Collecting Clone Changing Instance..... | 83 |
| 4.5 The Attributes for Clone Changing Instance | 85 |
| 4.5.1 The Code Attribute Set | 85 |
| 4.5.2 The Context Attribute Set | 86 |
| 4.5.3 The Evolutionary Attribute Set..... | 87 |
| 4.6 Building the Models and Predicting | 88 |
| 4.7 Experimental Results and Analysis | 89 |
| 4.7.1 Experimental Methodology | 89 |
| 4.7.2 The Effectiveness Experiments for Employed Machine Learning Methods | 92 |
| 4.7.3 The Experiments for Two Usage Scenarios | 95 |
| 4.7.4 Discussion | 100 |
| 4.8 Summary of this Chapter | 102 |
| Chapter 5 Cross-Project Clone Consistency-Requirement Prediction | 103 |
| 5.1 Introduction | 103 |
| 5.2 The Problem for Cross-Project Clone Consistency Maintenance..... | 103 |
| 5.2.1 Problem in this Chapter | 103 |
| 5.2.2 The Proposed Method | 104 |
| 5.3 The Definitions for Clone Consistency-Requirement | 105 |
| 5.4 Collecting the Dataset for Cross-Project Prediction | 107 |
| 5.4.1 Data Sources for Cross-Project Prediction..... | 107 |
| 5.4.2 The Attributes and Generation for Data Set | 108 |
| 5.5 Building the Cross-Project Models and Predicting | 110 |
| 5.6 Experimental Results and Analysis | 111 |
| 5.6.1 Experimental Methodology | 111 |
| 5.6.2 The Effectiveness Experiments for Cross-Project Prediction | 113 |
| 5.6.3 The Experiments of Cross-Project Prediction for Training Size | 116 |
| 5.6.4 The Experiment of Cross-Project Prediction for Usage Scenarios | 118 |
| 5.7 Design and Implementation for the Plug-in of Clone Consistency-Requirement Prediction..... | 127 |

| | |
|---|-----|
| 5.7.1 The Framework for the Plug-in of Clone Consistency Prediction | 127 |
| 5.7.2 The Design for Plug-in Function Modules of Clone Consistency Prediction | 128 |
| 5.7.3 The Implementation for Plug-in of Clone Consistency Prediction | 131 |
| 5.8 Summary of this Chapter | 132 |
| Conclusions | 135 |
| References | 137 |
| Papers published in the period of PH.D. education | 150 |
| Statement of copyright and Letter of authorization | 152 |
| Acknowledgements | 153 |
| Resume | 154 |

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，软件质量问题日益引起了人们的广泛重视。保证软件质量、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。软件开发人员会向软件系统中不断添加新的功能，这将导致软件规模越来越大、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下降。

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，可以提高软件开发效率，但同时也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]。大量的克隆代码，使得软件变得越来越难以理解，降低了软件的可理解性。与此同时，克隆代码也会随着软件系统演化，在演化中可能会被软件开发人员修改而发生变化，从而降低软件可维护性。更进一步，克隆代码的一致性变化也可能导致软件缺陷，从而降低软件的质量。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，例如克隆代码缺陷研究 [7][8][9]、克隆代码有害性研究 [10][11][12] 等。因此，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，并对其进行有效地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。

为解决克隆代码问题，人们展开了大量且深入的研究，取得了较多的研究成果。目前对克隆代码的研究有三个主要的研究内容，即克隆检测研究、克隆分析研究和克隆维护研究。其中，克隆检测研究是最早也是研究最为充分的一个研究内容，旨在从软件系统中收集已有的克隆代码。到目前为止，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]。但是，由于克隆代码大量存在并且情况复杂，克隆检测的结果并不能完全令人满意。更重要的是，克隆检测研究既无法消除克隆代码对软件的不利影响，也无法直接帮助提高软件的质量以及可维护性。而对克隆代码的分析和维护研究可以弥补这一不足之处。对克隆代码的分析研究，

例如克隆代码产生的原因、对系统产生的不利影响等，可以辅助开发人员理解和维护克隆代码。对克隆代码的维护研究，例如克隆代码重构、克隆代码变化维护等，可以帮助开发人员维护克隆代码。

因此，本文通过对克隆代码的分析和维护研究，旨在帮助开发人员理解和维护软件系统中的克隆代码，并帮助提高软件质量、软件可理解性和可维护性。

克隆代码分析是目前克隆代码研究中较为活跃的一个分支，其主要目的是辅助开发人员理解克隆代码。研究发现克隆代码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，可以提高软件的可理解性。

克隆维护研究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、解决克隆代码所引发的问题。在克隆代码的演化过程中，克隆代码可能会被程序开发人员修改而发生变化，即致性变化或者不一致变化 [21][22]。克隆代码的一致性变化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；其次，遗忘克隆代码的一致性变化会导致克隆一致性违背缺陷（Consistency Defect）的产生 [7][9]，会进一步增加软件维护代价。因此，如何避免和预测克隆代码的一致性维护（Consistency Maintenance），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，可以有效地避免克隆一致性违背缺陷，也可以降低克隆所导致的维护代价，并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，本文研究基于软件演化的克隆代码分析和一致性维护方法，旨在帮助开发人员在软件开发过程中理解和维护软件系统中的克隆代码。首先，本文结合克隆代码演化的研究成果，研究并提取了克隆代码演化特征，揭示克隆代码及其演化过程所隐含的信息，为克隆一致性维护研究奠定了基础。然后，针对克隆代码一致性变化可能导致额外维护代价和缺陷的问题，分别在克隆代码创建时和克隆代码变化时，预测克隆代码的一致性维护需求，可以降低克隆代码的维护代价和避免相关的一致性违背缺陷。最后，结合软件开发过程，针对软件开发初期系统中缺少相应的数据的问题，研究跨项目的克隆代码一致性维护需求预测方法，使用不同系统的数据预测其它系统的克隆一致性维护需求。本文的方法可在软件开发过程中帮助软件开发人员边开发、边维护克隆代码，对提高软件质量，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

克隆代码（简称克隆）是根据某种相似性的定义彼此之间相似的代码片段 [1]。在克隆代码研究领域中，目前尚没有完善的克隆代码定义及其分类方式。目前使用最为广泛的是按照克隆代码的语法和语义的相似程度，将克隆代码划分为如下四种类型 [23]：

- 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，前三类克隆代码是语法相似的克隆代码。Type-4 克隆是语义相似的克隆代码。目前对克隆代码的研究主要集中在前三类的克隆代码上。除此之外，也有其它类型的克隆代码的研究，如函数克隆 [24]、模型克隆 [25] 和结构克隆 [26][27] 等。

1.2.1 研究热点与趋势

迄今为止，克隆代码的研究已有超过 20 年的研究时间。为方便研究人员查阅相关研究，阿拉巴马大学维护了一个克隆代码领域的文献库（目前已停止维护¹）。该文献库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

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。

评估相关的研究内容。

Roy 分析了此文献库中每个研究方向的论文分布和研究进展情况 [28], 但此文献库仅更新至 2013 年。为了分析和反映克隆代码研究的研究热点和发展趋势, 本文检索和收集了近几年的克隆代码相关文献, 并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。

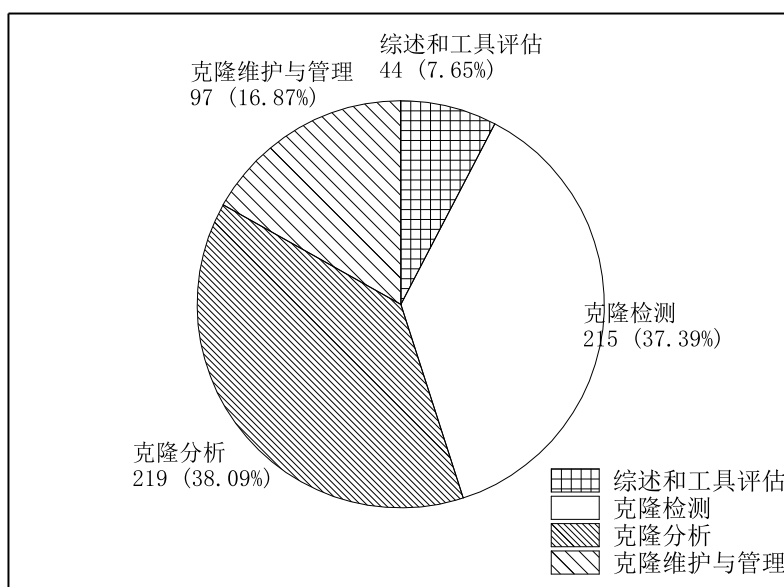


图 1-1 克隆代码研究领域文献分布情况

Fig. 1-1 Literature distribution of code clone research

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, 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%)。

为分析克隆代码研究领域的发展趋势, 本文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 和图 1-3 可以看出, 克隆研究可以划分为两个阶段: 1990-2003 年和

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

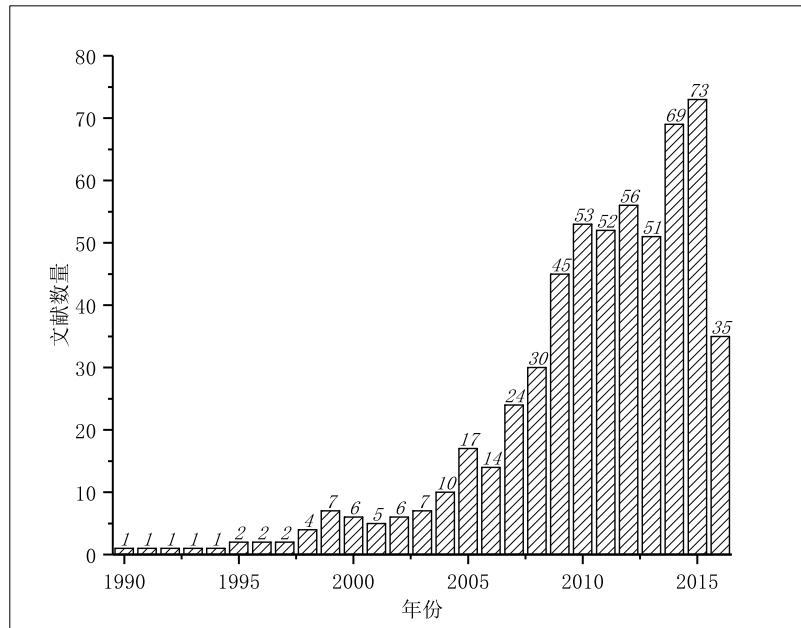


图 1-2 克隆领域每年发表的论文数量

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

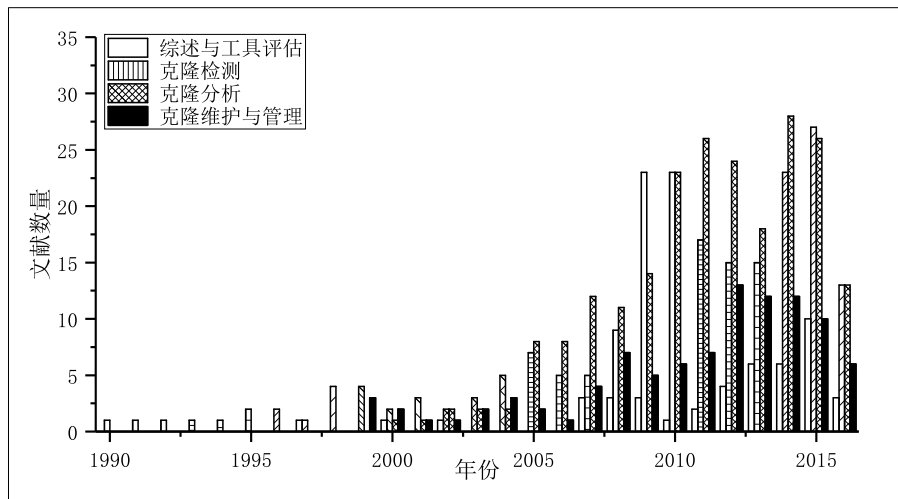


图 1-3 各个克隆研究活动每年发表的文献数量

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

2004-2016 年。在 2003 年以前, 克隆代码研究领域的文献数量较少。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。在 2003 年以前, 克隆代码研究主要集中在克隆检测方向。在 2004 年以后, 各个研究方向的文献数量都有了快速增长, 其中尤以克隆检测和分析方向的发展最为迅速。这说明克隆检测和分析是目前研究最为充分和最为活跃的研究方向。相比之下, 克隆维护和管理仅在近几年才引起人们的关注, 文献数量占比相对较少, 但已呈现出逐年增长和后来者居上的趋势。这说明克隆维护和管理正在逐渐成为克隆代码研究领域的新热点。

1.2.2 克隆代码检测研究

克隆检测是指从软件系统中检测克隆代码, 并向软件开发人员报告克隆代码的活动。

1.2.2.1 克隆检测方法

迄今为止, 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 克隆代码的检测。目前, 使用较多的基于文本的克隆检测工具主要有 duploc[5]、Simian³、DuDe[29]、SDD[30]、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[31]、iClone[32]、SourcererCC[33] 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

³Simian - Similarity Analyser: <http://www.harukizaemon.com/simian/index.html>

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[34]、SimScan⁴、Deckard[15]、CloneDigger[35] 等。

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[36] 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 [26]。White 等人也使用深度学习技术检测系统中的克隆代码 [37]。

1.2.2.2 克隆检测方法及其工具评估

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。因此, 通过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 [38][39][40][41]。

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没有列出。

从表 1-1 可以看出, 基于 Text 的检测工具不支持 Type-4 克隆的检测。对 Type-1 克隆的检测效果最好, 对 Type-3 克隆的检测效果一般。而对 Type-2 克隆支持较弱, 仅有两个工具可以支持。原因是 Type-2 克隆是标识符重命名的克隆代码, 基于

⁴SimScan: <http://www.blue-edge.bg/simscan/>

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][42]，缺少相应的检测工具。

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[29] | 1、3 | 较好 1，一般 3 |
| | SDD[30] | 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[31] | 1、2、3 | 较好 1、2，一般 3 |
| | iClone[32] | 1、2 | 较好 1，2 |
| Tree | CloneDr[34] | 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[35] | 1、2、3 | 较好 1、3，一般 2 |
| PDG | DupliX[36] | 1、2、3、4 | 较好 1、2，一般 3，较差 4 |
| | Gabel[43] | 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[42] | 1、2、3、4 | 较好 1、2，较差 3、4 |

1.2.3 克隆代码分析研究

克隆分析是指使用各种技术手段分析系统中的克隆代码，旨在帮助软件开发人员更好地理解和维护克隆代码。目前，克隆分析研究主要集中在克隆表示、克隆演化、克隆评价和克隆可视化上。

1.2.3.1 克隆代码表示

目前的很多克隆检测工具，如 NiCad、CCFinder、iClone 和 Decard 等，都使用检测报告的形式提供克隆代码的检测结果。由于不同的检测工具采用不同的检测技术来检测克隆代码，因此不同的检测工具的克隆检测结果往往是不同的。而不同的检测工具又采用不同的克隆表示方法来报告克隆检测结果，缺乏统一的克隆表示形式，因此要实现各种克隆工具的检测结果信息的共享和融合，是非常困难的。

为了实现各个检测工具的克隆检测结果的共享，克隆检测工具 iClone 使用 Rich Clone Format(RCF) 表示克隆代码 [44]。RCF 定义了多种函数接口，以方便用户和其它检测工具检索克隆代码⁵。RCF 不仅可以表示克隆代码的基本位置信息，还允许用户对其进行扩展，表示克隆代码的扩展信息，但是并没有指出可以表示哪些具体的扩展信息以及以何种形式来表示这些信息。

为了表示克隆代码的上下文信息，Duala-Ekoko 出了另一种克隆表示形式 Clone Region Description(CRD)[45]。与其它表示形式不同的是，CRD 采用一种与位置无关的克隆表示方法。CRD 不保存克隆代码的位置信息，仅使用克隆代码本身及其所在文件的语法信息、结构信息等表示克隆代码。CRD 的这种表示方法可以详细地描述克隆代码的上下文信息，尤其是语法信息。CRD 使用上下文描述克隆代码，有助于辅助开发人员跟踪克隆代码。

各种检测工具的克隆检测结果缺乏统一的表示形式，使得现有的克隆表示方法不能满足对克隆代码分析和维护的需求。为此，Harder 在文献 [46] 中分析了统一克隆表示的挑战和限制，并给出了一个克隆表示的概念模型，但未给出具体的克隆表示方法。Kapsner 也认为需要对克隆代码进行统一的表示，并提出了 Unified Clone Model(UCM) 模型 [47]。UCM 模型是一种较为完备的克隆表示方法，不仅可以表示克隆代码的基本位置信息，还可以表示克隆代码的上下文信息、演化信息和度量值信息等，具有较强的理论和实际意义⁶。

⁵<http://www.softwareclones.org/rcf.php>, RCF 格式文档。RCF 是克隆检测工具 iClone 的默认结构形式，其支持多版本克隆代码的演化信息，RCF 开放 API 给其它用户使用。

⁶http://www.softwareclones.org/ucm/index.php/Main_Page, UCM 格式文档。UCM 旨在建立一个统一模型用于组织不同克隆检测工具的检测结果，也可使得检测工具进行信息共享。

1.2.3.2 克隆代码演化

克隆代码往往存在于软件系统的多个版本中，并随着软件系统进行演化。克隆代码演化分析就是通过分析克隆代码的演化过程，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 [48]，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，提升了构建克隆家系的效率 [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 [49]。Harder 对现有的演化模型进行分析 [50]，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。

表 1-2 克隆演化特征分析

Table 1-2 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 |
| [51] | 克隆寿命 | 克隆寿命与克隆修改次数、新增和减少有关 |
| [52] | 克隆寿命 | 克隆代码的寿命比非克隆的寿命更长 |
| [53][54] | 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 |
| [55] | 克隆稳定性 | 克隆代码比非克隆代码更稳定 |
| [56][57] | 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 |
| [58][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 |
| [59] | 延后传播 | Type-3 更容易发生延后传播并导致缺陷 |

表 1-2 列出了目前对克隆演化特征的研究情况。由表 1-2 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。

1.2.3.3 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维护克隆代码。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。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。

1.2.3.4 克隆代码可视化

克隆检测工具通常使用文本形式来保存克隆代码的检测结果。但是，用文本形式保存大量的检测结果，既不形象直观，也不利于深入分析克隆代码的分布情况和关联关系。因此，展开了对克隆代码进行了可视化研究。

目前，使用较多的可视化技术主要有捆绑图、散点图和树图。捆绑图是从宏观的角度展示克隆文件之间的关系以及克隆关系的强弱。一般以克隆文件为粒度，采用同心圆环、扇形和曲线相结合的形式可视化克隆代码 [71][72][73]。例如可视化工具 ClonEvol 使用分层捆绑图分析软件的演化情况，以辅助开发人员寻找自己感兴趣的克隆代码 [71]。Hauptmann 则使用捆绑图分析软件各模块之间的耦合关系 [72]。散点图是从微观的角度使用矩阵的形式对软件中克隆代码的分布情况进行可视化 [74][75][76]。Cordy 使用散点图分析软件之间的相似性 [74]，Higo 则使用散点图分析软件中的克隆模式 [75][76]。树图是以二维矩形图的形式对克隆代码在软件中的分布情况和层次结构进行可视化，以树图形式可视化克隆代码的工具 VisCad[77][78]、SoftGUESS[79]。

克隆可视化的作用不仅仅是可视化克隆代码，更重要的作用是通过克隆代码

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 | [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 |
| | [60] | 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 |
| | [61] | 积极 | 克隆代码含有更少的缺陷, Type 3 所含有的缺陷最少 |
| | [62] | 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 |
| | [63] | 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 |
| 维护代价 | [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 |
| | [64] | 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 |
| | [65] | 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 |
| | [66] | 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 | [11] | 积极 | 克隆代码并不比非克隆代码具有更高的有害风险 |
| | [67][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 |
| | [68] | 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 |
| | [69] | 中立 | 使用分类模型抽取有问题的克隆代码, 只有少部分是问题的克隆 |
| | [70] | 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 |

的可视化辅助软件开发人员分析克隆代码对软件产生的影响。例如可视化工具 Doppel-Code 可以用于分析克隆代码对系统全局和局部产生的影响，并将不同克隆代码组对系统的影响进行排序 [80]。Jiang 提出一个克隆可视化框架，辅助开发人员分析系统内各模块间的耦合和内聚情况 [81][82]。Zhang 等人通过对结构克隆进行过滤和可视化分析，帮助维护人员寻找对开发人员有用的克隆代码以便进行代码复用 [83]。

1.2.4 克隆代码维护研究

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。

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 [84]。将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。克隆代码的重构研究是对克隆代码维护研究的一个主要的研究内容。

由于重构所需的条件较为苛刻，同时重构的代价也较大。因此，在重构前对克隆代码进行可重构性分析和重构排序显得尤为重要。可重构性分析旨在识别适于重构的克隆代码候选集合，以提高克隆重构的效率 [85][86][87][88]。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 [85]。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 [86]。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 [87]。Choi 等人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 [88]。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 [89][90][91]。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 [89]。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 [90]。Zibran 提出一种极限编程方法对克隆代码进行重构调度 [91]。另外，Liu 等人提出了一个重构调度方法帮助优化调度过程 [92]。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 [93]，应该结合可重构性分析指导克隆重构。Tsantalis 也对克隆代码的可重构性进行分析，并获得一些有用的结论帮助人们重构克隆代码 [94]。

克隆重构最主要的目的是消除系统中的克隆代码，因此提出了许多具体的重构方法 [95][96][97][98]。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 [95]。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地重构，然后通过检测和参数化克隆代码的差异点对克隆代码进行重构，取得了较好的效果 [96]。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 [97]。Ettinger 等人提出了一个高效的方法可以消除系统中的 Type-3 克隆代码 [98]。

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 [99][100][101][101]。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 [99]。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 [100]。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 [101]。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 [102]。

1.2.4.2 克隆代码复用

近些年来，随着对克隆代码评价分析研究的深入，人们已经意识到复用健壮性好的克隆代码不仅有助于缩短软件开发周期，还有利于提高软件的健壮性和可维护性。因此，也出现了少量的克隆代码复用的研究。

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 [105]。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 [106]。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 [107]。在软件开发过程中，向开发人员提供一个可复用的克隆代码库，是一种较为为主动的克隆代码复用形式。例如，Yang[108] 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，从不同粒度对可复用的代码提供搜索支持 [109]。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，从而决定如何复用和维护既有的克隆代码 [110]。

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 [111]。Cheng 等人研究了跨项目的克隆代码检测技术 [112]。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 [113]。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。

1.2.4.3 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。

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 [103]，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]。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 [104]。

由于受软件复用的影响和编程语言的限制，大部分克隆代码是无法避免的。克隆规避可以作为一种辅助的手段，通过限制开发人员的复制粘贴操作帮助减少克隆代码的产生，但却无法规避全部的克隆代码。

1.2.4.4 克隆代码管理

克隆代码管理的概念最早是 1997 年由 Koschke 提出的 [114]，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。

早在 2012 年的面向工业界的克隆管理会议⁷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

⁷Software Clone Management Towards Industrial Application

界是目前克隆研究领域急需解决的问题 [115]。Koschke 将克隆管理划分为预防性、补偿性和改正性三种类型 [114]。预防性克隆管理的目标是避免克隆代码，关注于避免新的克隆代码的产生，而不是对已有克隆代码的管理。补偿性克隆管理的目标是发现克隆代码所引发的问题，并对其对系统造成的不利影响进行补偿。改正性克隆管理的目标是以主动的方式消除可能对系统产生不利影响的克隆代码。Roy 在文献 [28] 中对克隆管理研究进行了详尽的阐述，指出目前对克隆管理的研究依然较少，尚缺少有效的克隆管理方法，尤其是对 Type 3 和 Type 4 克隆代码进行管理的研究还不够充分，未来应针对这两种类型的克隆代码的管理进行深入研究。

研究人员也提出了一些克隆代码管理的工具。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 [116]。Nguyen 实现了一种可用于管理克隆代码的 eclipse 插件 JSync[117]。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。而另一个较早提出的 eclipse 插件 CeDAR[118]，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。同时，Dang 等人将克隆代码检测和分析与软件开发过程相结合，也具有一点克隆管理的功能 [119]。

1.2.5 当前研究中存在的问题分析

目前，克隆代码研究已经成为软件工程领域的一个热点问题，也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。但是，目前研究中仍然存在一些问题和不足之处。

(1) 尽管对克隆检测的研究已经较为深入，已经开发和提出了许多的克隆检测方法和工具。但是，克隆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。于此同时，对克隆代码的分析和维护研究是解决克隆代码问题的主要内容。

(2) 克隆分析研究是目前较为活跃的一个研究内容，可以辅助开发人员理解克隆代码。在克隆分析的研究中，克隆代码演化是最为重要的研究内容，是克隆评价研究以及其它相关研究的基础。但是，目前克隆演化分析研究仍不能令人满意。首先，目前的研究往往集中到某一个具体的内容上，例如克隆寿命、稳定性等，从而缺少宏观角度的克隆演化分析方法。其次，目前的克隆演化研究也往往带有较强的主观性，甚至得出了一些完全相悖的研究结论。例如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。

(3) 克隆维护研究可以帮助解决或者避免克隆代码对系统产生的影响，进行了大量的克隆维护研究，包括克隆代码重构、复用和管理等等。但是，目前克隆维护研究也无法令人满意。目前克隆维护研究没有解决克隆难以维护的根本原因，即克隆代码在其演化过程中的一致性变化问题。同时，克隆代码的一致性变化问题，也是影响软件质量的一个重要因素。克隆代码的一致性变化，不仅会使得软件系统的维护代价增大，也可能会导致克隆相关缺陷。因此，如何避免和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。

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克隆代码，以及在其演化过程中的克隆一致性维护问题，本文结合软件演化、程序分析和机器学习方法，研究克隆代码分析与一致性维护方法，旨在帮助软件开发人员理解克隆代码、避免克隆代码一致性违背缺陷和降低克隆代码的维护代价，并在软件开发过程中边开发、边维护克隆代码，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全面且客观地理解和分析的问题，本文将克隆代码在其演化过程中所表现出的特征称之为克隆演化特征。研究基于聚类的克隆代码演化特征分析方法，提取相应的属性值表示克隆代码及其演化过程，分析克隆代码的演化特征，帮助软件开发人员理解克隆代码，并为克隆一致性维护研究奠定基础。

(2) 针对新创建的克隆代码的一致性变化往往会导致额外维护代价的问题，研究克隆代码创建一致性维护需求预测方法。将系统中最早出现的克隆代码称为克隆创建实例，将其演化过程中的一致性变化称为克隆创建一致性维护需求。通过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，并使用机器学习方法预测克隆代码创建的一致性维护需求，帮助软件开发人员降低克隆代码的额外维护代价。

(3) 针对克隆代码演化中的一致性变化会导致克隆一致性违背缺陷问题，研究克隆代码变化一致性维护需求预测方法。将演化中发生变化的克隆代码称为克隆变化实例，并将其未来演化中所发生的一致性变化称为克隆变化一致性维护需求。通过从克隆组的角度提取了代码属性、上下文属性和演化属性用于表示克隆变化

实例，并使用机器学习方法预测克隆代码变化一致性维护需求，帮助程序开发人员避免克隆代码一致性违背缺陷。

(4) 针对软件开发初期因数据不足而无法预测克隆一致性维护需求的问题，研究跨项目的克隆代码一致性维护需求预测方法。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。使用训练系统的数据训练机器学习模型，并预测测试系统的克隆一致性维护需求。同时，与软件开发过程相结合，设计并实现一个 eclipse 插件预测克隆代码的一致性维护需求，帮助实现边开发、边维护克隆代码，并帮助提高软件的质量和可维护性。

1.3.2 论文结构

基于软件演化的克隆代码分析与一致性维护方法，综合考虑软件演化过程、程序分析方法和机器学习方法，实现对克隆代码的一致性维护。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，给出论文的主要研究内容及其关系示意图，如图 1-4 所示。

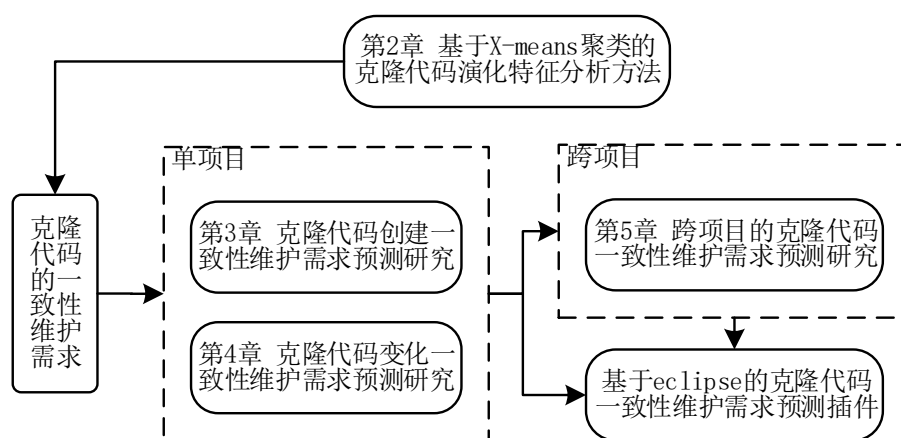


图 1-4 论文主要研究内容及其关系示意图

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

本文主要研究四个内容，首先第 2 章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，并指导克隆代码一致性维护研究。经分析发现：软件中存在相当数量发生一致/不一致变化的克隆代码，会导致额外的维护代价，并有可能引发一致性违背缺陷。因此，在软件开发过程中，需要对克隆代码进行一致性维护，从而实现边开发、边维护克隆代码，降低软件的维护代价。为了实现上述目标，

针对克隆代码的一致性变化会导致额外维护代价和一致性维护缺陷问题，分别在克隆代码创建时（第3章）和变化时（第4章）预测克隆代码的一致性维护需求，帮助降低软件维护代价。最后，针对软件开发初期软件缺乏自身历史数据的问题，第5章研究跨项目的克隆代码一致性维护需求预测问题，并结合软件开发过程，基于 eclipse 实现了一个克隆代码一致性维护需求预测插件，帮助软件开发人员边开发、边维护克隆代码。

各章节研究内容安排如下：

第2章研究基于 X-means 聚类的克隆代码演化特征分析方法，提取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，并为下一步克隆一致性维护研究奠定基础。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，提取相应的度量值从克隆片段、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。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章研究克隆代码创建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。最后，使用机器学习方法训练模型，预测克隆代码创建时一致性维护需求。

第4章研究克隆代码变化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，从克隆组的角度提取代码属性、上下文属性和演化属性三组属性值表示克隆变化实例。最后，使用机器学习方法训练模型，预测克隆代码变化时一致性维护需求。

第5章研究跨项目克隆代码一致性维护需求预测实证研究，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，检测不同软件系统的克隆代码并构建克隆家系，收集并表示克隆代码实例。然后，将软件系统划分为训练系统和测试系统，并使用训练系统数据训练机器学习模型，并在测试系统上预测克隆代码的一致性维护需求。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，帮助实现边开发、边预测克隆代码的一致性维护需求。

第 2 章 基于 X-means 聚类的克隆代码演化特征分析

2.1 引言

研究表明软件中存在大量的克隆代码,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,克隆代码也会随着软件系统进行演化,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分析方法。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。

为帮助程序开发人员获得克隆代码演化特征,在提取克隆代码的属性特征的基础上,使用机器学习中的聚类方法挖掘克隆代码的演化特征,为本文后续章节的克隆代码一致性维护奠定基础。首先,通过映射相邻版本的克隆代码,构建软件系统的克隆家系。然后,从克隆片段、克隆组和克隆家系三个不同的角度描述克隆代码及其演化过程,并提取不同属性值表示克隆代码及其演化过程。最后,根据所提取的属性值生成相应的聚类向量,使用聚类方法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,结果表明:在演化过程中,大部分克隆代码是稳定的,但也存在一定数量发生变化的克隆代码;且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 克隆代码演化特征分析

2.2.1 现有研究存在的问题

为描述克隆代码随着软件的演化过程, Kim 等人首先提出了克隆家系模型 [16], 并引发了对克隆代码的演化以及演化规律的大量研究。目前, 引发人们关注的演化规律主要包括: 克隆寿命、克隆稳定性与一致性变化等研究 (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。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 longer [16]; 进一步对长寿命的克隆代码进行研究后, 发现克隆代码的变化会使得其寿命变短 [51]。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 [52]。通过对克隆寿命的研究发现, 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 [53]。另外, 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 [54]。因此, 从上述研究中不难得出结论, 克隆代码会长时间的存在于系统中。

长时间存在于系统中的克隆代码, 可能会被程序人员修改而发生变化, 因此引发了对克隆代码的稳定性研究。其中一个普遍的观点是寿命较长的克隆代码是稳定的 [55][56][57], 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 [58][19]。因此, 对克隆代码的稳定性问题尚未达成共识, 仍需要进一步研究。

在克隆代码演化中, 克隆代码的一致性变化和 inconsistency 变化往往会引发人们的强烈关注。原因在于克隆一致性变化可能会引发相关的软件缺陷, 如标识符重命名缺陷等。Göde 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 inconsistency 变化的克隆代码比例大约各占一半, 并且大部分发生 inconsistency 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 [59]。因此, 尽管已经有了一些克隆代码的一致性变化研究, 但是克隆代码如何变化也没有得出明确的结论。

上述研究的结论并不是相互独立的, 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。然而, 目前克隆演化分析研究仍然不能令人满意, 存在以下两点不足:

(1) 在现有的克隆代码演化分析研究中, 大多数研究关注的是局部的演化特征, 缺少全局的克隆代码演化特征的分析方法。

(2) 在现有的克隆演化分析研究中, 大多数研究带有较强的主观性, 即认为克隆代码是有害的, 并从克隆代码有害性的角度验证克隆代码的有害性。

综上, 克隆代码在随着软件系统的演化过程中, 可能会被开发人员修改而发生变化, 因而对理解系统中的克隆代码带来了新的挑战。如何从大量的克隆代码及

其演化过程中，全面且客观地分析克隆代码演化以及变化规律（即克隆代码的演化特征）是一个值得研究的问题。这不仅对帮助程序开发人员理解克隆代码及其演化过程具有积极意义，还可以帮助开发人员维护系统中的克隆代码。

2.2.2 本文的解决思路

本章拟通过聚类的方法分析和提取克隆演化特征，先给出克隆演化特征的描述，如下所示：

定义 2.1 (克隆代码演化特征) 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。

为分析克隆代码及其演化特征，本章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，本章将克隆代码及其演化过程抽象成为“特征向量”，并借助机器学习中的聚类方法挖掘克隆代码的演化特征。

本章所提出的基于聚类的克隆代码演化特征分析框架如图 2-1 所示。从图所示，该方法可以划分为三个部分，分别是构建克隆家系、克隆演化实体表示和演化特征挖掘。在构建克隆家系阶段中，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地描述克隆代码的演化过程，同时也可以快速有效地识别克隆代码的演化模式。然后，使用三种不同的克隆实体从三个不同的角度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。

本章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属性值生成，克隆实体的属性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

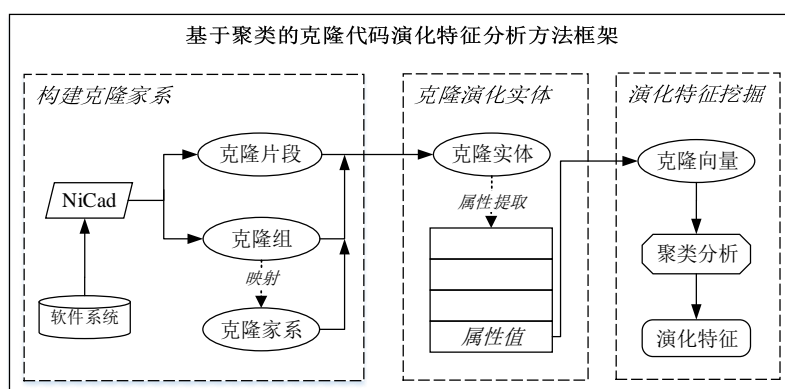


图 2-1 基于聚类的克隆演化特征提取方法框架

Fig.2-1 The framework for clone characteristic analysis based on clustering

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克隆代码（即全部克隆家系）的演化过程以及演化特征。

本章所采用的聚类分析方法为 X-means[120] 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，影响克隆代码演化特征的分析 and 提取。因此，本章采用 X-means 聚类方法，不需要人为的指定聚类数量，X-means 方法会自动地选择出最佳的聚类数量。

因此，本章基于软件演化和机器学习方法分析和挖掘克隆代码的演化特征，将重点分析讨论如下问题：

- （1）如何结合软件演化过程，定义和模型化克隆代码的演化过程？
- （2）如何从不同的角度描述和表示克隆代码及其演化过程，即如何从克隆片段、克隆组和克隆家系三个不同的角度分析克隆代码的演化情况？
- （3）如何全面且客观地分析和挖掘克隆代码演化特征，并帮助开发人员理解和维护软件系统中的克隆代码？

2.3 克隆代码演化以及相关定义

软件工程实践会产生大量的克隆代码，存在的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，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（参见本文绪论第 1.2.2 节克隆代码检测）。

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,并向程序开发人员报告检测结果。克隆检测结果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片段的集合。克隆片段和克隆组可以描述如下:

定义 2.2 (克隆片段) 克隆片段 (Clone Fragment, CF) 是一段代码片段,包括若干连续的代码行。根据某种相似性,克隆片段与一些其它的代码片段彼此相似,并将它们称之为克隆代码片段,简称为克隆代码。

定义 2.3 (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。克隆组揭示了该克隆组内的克隆片段之间的克隆关系,即克隆组内的克隆片段互为克隆关系。

如绪论中所述,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 [48],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,是迄今为止最好的用于描述克隆代码演化情况的模型 [16]。因此,本文也使用克隆家系模型描述克隆代码演化过程。Kim 所提出的克隆家系可以描述如下:

定义 2.4 (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) 随着软件系统进行演化的过程。CGE 图中的某一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),即该克隆组 CG 由上一版本 V_{i-1} 的 CG_{i-1} 演化至下一版本 V_i 中的 CG_i 。

在克隆代码的演化过程中,克隆代码可能会被开发人员修改而发生变化,因而也会导致两个相邻版本间的同一克隆组的变化。对于相邻版本之间的克隆组的演化情况,可以使用“克隆演化模式”进行描述。克隆组在两个相邻版本之间的克隆演化模式可以描述如下:

定义 2.5 (克隆演化模式) 克隆演化模式 (Clone Evolution Pattern, CEP) 是某一克隆组相邻两个软件版本的演化情况。CEP 描述该克隆组在相邻版本的变化情况,根据不同的演化情况具有 7 种不同的演化模式。假设该克隆组 CG 从系统版本 V_{i-1} 演化至 V_i ,其演化关系 (CG_{i-1}, CG_i) 可以定义如下:

- 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。

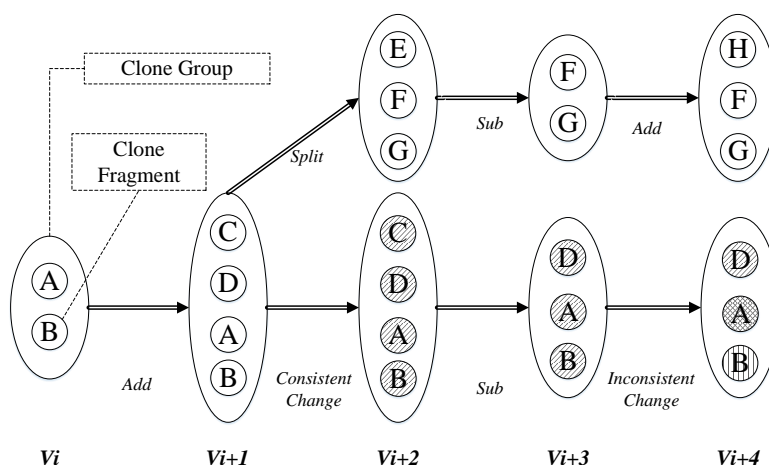


图 2-2 克隆家系示意图

Fig.2-2 An Example for Clone Genealogy

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 2-2 所示。从图中可以看出，克隆家系 (CGE) 是克隆组 (CG) 随着软件系统演化的一个有向无环图，图的节点表示克隆组 CG，图中边表示克隆组的演化关系 (CG_{i-1}, CG_i)，演化模式可以通过比较 CG_{i-1} 和 CG_i 中克隆代码片段获取。

图 2-2 同时也描述了一个克隆组 CG 在五个版本 (V_i, V_{i+4}) 中的演化过程。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增加模式 (Add Pattern)。从 V_{i+1} 到 V_{i+2} 中，克隆组首先分裂为两组，因此其演化模式为分裂模式 (Split Pattern)，同时下方的克隆组发生了相一致的变化，其演化模式为一致性变化模式 (Consistent Change Pattern)。从版本 V_{i+2} 到 V_{i+3} 中，演化模式

一个是减少模式 (Subtract Pattern), 另一个是不一致变化模式 (Inconsistent Change Pattern)。从版本 V_{i+2} 到 V_{i+3} , 克隆模式分别是增加模式 (Add Pattern) 和减少模式 (Subtract Pattern)。

2.4 软件克隆家系的构建

在构建克隆家系阶段, 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并识别克隆演化模式。

2.4.1 克隆检测与 CRD 描述

本节使用克隆检测工具 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) 描述克隆代码的相关信息。

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 [121][122], 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 (函数克隆也是块克隆代码), 本文使用块粒度检测克隆代码。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 [45]。CRD 能够充分地反映出克隆代码本身的信息, 还能够反映克隆代码区域的特征, CRD 的描述如图 2-3 所示。CRD 克服了使用文件行描述克隆代码的局限, 不依赖于克隆区域的具体位置, 同时也独立于具体文本。使用 CRD 可以映射相邻版本之间的克隆代码, 慈蒙等人改

¹NiCad 可从此网站获取: <http://www.txl.ca/nicadownload.html>。

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 [123][124]。同时，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。

```
<CRD> ::= <file> <class> <CM> [<method>]
<method> ::= <signature> <CM> <block>*
<block> ::= <btype> <anchor> <CM>
<btype> ::= 'for' | 'while' | 'do' | 'if' | 'switch' | 'try' | 'catch'
```

图 2-3 克隆区域描述符示意

Fig. 2-3 The Definition of Clone Region Description

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 [123][124]。假设给定一个软件系统的两个相邻版本 V_i 和 V_{i+1} ，并分别给出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，可以通过遍历映射结果生成系统中的所有的克隆家系。

Input: All software's source code and all code clones with N versions

Output: All clone genealogies $CGEs$

```

1 for  $i=1$  to  $N$  do
2     Generating CRD for represent each  $CF_i$  in version  $i$ ;
3     Saving all results in result_  $i$  files;
4 end
5 for  $i=1$  to  $N - 1$  do
6     Mapping all the  $CFs$  and  $CGs$  between adjacent versions  $i$ 
       and  $i + 1$ ;
7     Saving all mapping results in mapping_  $i$  files;
8 end
9 for  $i=1$  to  $N - 1$  do
10    for each mapping  $CGs$  in version  $i$  and  $i + 1$  do
11        Identify all clone patterns CEPs in thus two versions
           according to Definition 2.5 ;
12        Saving all mapping results in mapping_  $i$  files;
13    end
14 end
15 Generating all clone genealogies  $CGEs$  through traversing all
       mapping files;
16 return  $CGEs$ ;
```

算法 2-1 克隆家系构建与演化模式识别算法

Algo. 2-1 The algorithm for building clone genealogies and identifying clone evolution patterns

于此同时，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_{i+1} 的克隆演化模式。

构建克隆家系的算法如 2-1 所示。算法中第 1-4 行是为所检测到的每一个克隆代码生成 CRD 描述，并将结果保存。第 5-8 行是使用基于 CRD 克隆群映射算法

映射相邻版本的克隆代码，同样将结果保存起来。第 9-14 行根据克隆演化模式定义，识别所有映射的相邻版本的克隆组的克隆演化模式。第 15 行通过遍历映射文件生成系统中全部的克隆家系。假设一个版本中有 m 个克隆代码，生成 CRD 所需时间为 $O(m)$ 。软件系统的 n 版本需时间为 $O(m * n)$ 。映射克隆相邻版本的克隆代码以及识别演化模式的时间复杂度同样为 $O(m * n)$ 。所以，该算法的复杂度为 $O(m * n)$ 。

2.5 克隆演化实体的特征描述

为挖掘克隆代码的演化特征，本节从三个不同的角度表示克隆代码及其演化情况，即克隆片段、克隆组和克隆家系，并将之称为克隆演化实体 (简称为克隆实体)。正如在机器学习中通常所做的那样，本节还提取不同的属性特征表示克隆代码实体。对克隆片段实体，重点关注克隆代码片段的变化情况。对克隆组实体，重点关注克隆组的克隆演化模式分布情况。对于克隆家系实体，重点关注克隆组在整个演化过程中的演化情况。

2.5.1 克隆片段实体的特征

克隆代码片段是最小的克隆演化实体单元。克隆片段实体描述了克隆代码本身的一些特征。在克隆片段的生命期间，克隆片段可能被软件人员开发人员修改 (特别是在软件维护期间)。因此，克隆片段在演化过程中可能会发生变化，甚至在其演化过程中可能发生不止一次的变化。

因此，对于克隆片段实体，将重点考虑其变化情况。某一版本 V_i 的克隆片段实体，将提取其两个属性特征表示克隆片段实体，即历史变化次数 (截止到版本 V_i) 和是否发生了变化 (从上一版本 V_{i-1} 演化到此版本 V_i 时)。同时，也会提取克隆片段实体的寿命作为另一个重要的属性特征。因此，对某一克隆片段实体 CF ，其所在的软件版本为 V_i ，该克隆片段实体 CF_i 的属性特征可描述如下：

- 克隆片段寿命 (Clone Fragment Life): 截止到当前版本 V_i ，克隆片段 CF_i 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} 演化到当前版本 V_i 时，克隆片段 CF_i 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_i 在其演化过程中所发生的变化次数。

2.5.2 克隆组实体的特征

克隆片段实体仅从单个克隆代码的角度描述了克隆演化中的被修改的情况。而克隆组实体可以提供一些克隆代码的区域性特征。克隆代码的变化往往会导致其所在克隆组的变化，可以使用克隆演化模式进行描述。克隆代码演化模式描述了相邻版本之间的克隆组的演化情况，因此，可以使用克隆组当前的演化模式描述克隆组实体的变化情况。

因此，对于某一个演化中的克隆组实体，从上一版本演化至当前版本时，其克隆演化模式可以作为克隆组实体的属性特征。同时，其存在于软件中的时间（克隆组寿命），同样也是一个重要的属性特征。克隆组寿命不但揭示了其在系统中存在的时间长短，也可能与克隆演化模式息息相关。因此，本节将克隆组的寿命和当前的演化模式视为克隆组实体的属性特征。对某一克隆组 CG ，其所在的软件文版本为 V_i ，该克隆组实体 CG_i 的属性特征可以描述如下：

- 克隆组寿命 (Clone Group Life): 截止到当前版本 V_i ，克隆组 CG_i 所经历的所有版本数量称之为克隆寿命。
- 当前静态模式 (Current Static Pattern): 克隆组 CG_i 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)。
- 当前相同模式 (Current Same Pattern): 克隆组 CG_i 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)。
- 当前增加模式 (Current Add Pattern): 克隆组 CG_i 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)。
- 当前减少模式 (Current Subtract Pattern): 克隆组 CG_i 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)。
- 当前一致性变化模式 (Current Consistent Change Pattern): 克隆组 CG_i 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)。
- 当前不一致变化模式 (Current Inconsistent Change Pattern): 克隆组 CG_i 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)。
- 当前分裂模式 (Current Split Pattern): 克隆组 CG_i 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 克隆家系实体的特征

克隆家系实体是克隆代码演化的全局视角。该实体提供了某一克隆组在其整个演化过程中的演化情况，可以帮助捕获软件系统中全部克隆代码的演化情况。如

前文所述，一个克隆组在所有的版本中的全部演化过程是一个克隆家系。因此，在其整个演化过程中，其所经历的所有的演化模式数量，可以描述克隆家系实体的在其整个演化过程中的整个变化历史演化过程。同时，对于一个克隆家系，克隆寿命是克隆家系的另一个重要的指标，描述了克隆代码在系统中存在的时间。

因此，克隆家系实体的寿命和克隆演化模式数量可以作为克隆家系实体的属性特征。对于某一克隆家系实体 *CGE*，该克隆家系实体 *CGE* 的属性特征可以描述如下：

- 克隆家系寿命 (Clone 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 Pattern 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 Pattern 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 Pattern 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 Pattern 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 Change Pattern 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 Change Pattern 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 Pattern 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 克隆演化特征挖掘

为了从克隆代码及其演化过程中挖掘演化特征，上一节使用不同的属性特征分别描述克隆代码实体、克隆组实体和克隆家系实体。在上一节的基础上，本节将使用 WEKA (“Waikato Environment for Knowledge Analysis” [125]) 中提供的聚类方法来分析克隆演化实体，将从克隆片段、克隆组和克隆家系三个不同的角度分别对克隆代码及其演化过程进行聚类分析。

2.6.1 克隆实体聚类向量的生成

使用 X-means 聚类克隆代码实体，需要生成与之对应的克隆聚类向量空间。这一过程可以划分为两个子步骤：首先，为每一个克隆代码实体成一个“克隆聚类向量”，该向量包含某个克隆实体的所有属性值。然后，使用克隆实体全部的“克隆聚类向量”生成所有克隆实体的“聚类空间”，包含克隆片段、克隆组和克隆家系三个聚类空间。

(1) 为每一个克隆代码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 $Vector$ ：即 $Vector = (v_1, v_2, \dots, v_m)$ ，其中 v_i 表示该克隆实体的一个特定属性值。以某一克隆组 CG_i 为例，为该克隆组实体 CG_i 生成一个 8 维向量 $Vector(CG_i) = (v_1, v_2, \dots, v_8)$ ，其中 v_i （对于所有 $i, i \leq 8$ ）是此克隆组 CG_i 所对应的属性值。 $Vector(CG_i)$ 即为克隆组实体 CG_i 的克隆聚类向量。

(2) 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，还生成系统的全部克隆聚类向量空间。假设由所有克隆实体的聚类向量生成的聚类向量空间为 X ，则 X 可由 $X = (x_1, x_2, \dots, x_n)$ 表示，其中 n 是全部克隆实体的数量。本章从克隆片段实体、克隆组实体和克隆家系实体三个不同角度，聚类分析克隆代码及其演化过程。因此，本节也将生成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实体的聚类空间， X_{CG} 表示所有克隆组实体的聚类空间， X_{CGE} 表示所有克隆家系实体的聚类空间。

2.6.2 基于 X-means 的克隆实体聚类

使用 X-means 方法聚类所有的克隆实体向量（ X_{CF} 、 X_{CG} 和 X_{CGE} ），并根据聚类结果分析和提取克隆代码演化特征。

本章将使用 WEKA 中实现的聚类方法，实现对克隆实体的聚类。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²。由于上一节所生成克隆代码的向量空间为 X_{CF} 、 X_{CG} 和 X_{CGE} ，分别代表了系统中全部的克隆片段实体、克隆组实体和克隆家系实体。因此，也将使用 X-means 方法聚类这三个不同的向量空间，并分析得到克隆代码

²WEKA 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/>

的演化特征。

本文使用的聚类方法是 X-means 聚类 [120]，X-means 聚类是 K-means 聚类 [126] 的改进方法。后者可以将向量空间 X 聚类为事前指定的 K 个 Cluster 中，并将彼此相似的向量分配到同一个 Cluster 中。但是，K-means 聚类方法必须指定所需要聚类的 Cluster 的数量。然而，对于克隆代码演化的聚类而言，由于缺少必要的基本信息，难以确定所需的 Cluster 的数目。因此，本章选择使用 X-means 聚类方法。X-means 聚类改进了 K-means 聚类方法，聚类时并不需要指定具体的 Cluster 数量。X-means 聚类会自动搜索最佳的 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 [120]。给定一个克隆实体的克隆向量空间 $X = (x_1, x_2, \dots, x_n)$ ，其中每个 x_i 是 d 维克隆演化实体的向量。X-means 聚类将向量空间 X 中的 n 个向量划分为 k 个 Clusters，即 $C = (c_1, c_2, \dots, c_k)$ ， c_i 表示一个 Cluster。 c_i 则表示了彼此比较相似的克隆代码实体，根据聚类结果可以进行克隆演化特征的挖掘。使用 X-means 聚类仅需要指定 K 的范围即可，聚类算法会自动搜索最佳的 Cluster 数量 K 。聚类分析的结果见后文的实验分析部分。

最后，根据聚类分析的结果，可以进行挖掘和分析克隆代码的演化特征。

克隆演化特征的聚类算法如 2-2 所示。算法的第 1-3 行分别收集系统中全部的克隆实体，分别为克隆家系实体、克隆组实体和克隆片段实体。第 4 行初始化三种克隆实例的聚类向量空间。第 5-8 行、9-12 行、13-16 行分别使用上文描述的属性生成克隆实体的聚类向量，并将其添加到聚类向量空间中。最后，第 17 行调用 WEKA 聚类克隆实体。收集不同克隆实体的算法复杂度为 $O(m)$ 。算法的主要时间消耗在了生成克隆实例的向量空间上，其时间复杂度为 $O(n * m)$ ， n 为系统的版本数量。所以，该算法的时间复杂度为 $O(n * m)$ 。

Input: All clone genealogies $CGEs$ and all source code

Output: All the clusters for clone fragments, groups, and clone genealogies

- 1 Collecting all entities of clone genealogy with number N_{cge} ;
 - 2 Collecting all entities of clone group with number N_{cg} ;
 - 3 Collecting all entities of clone fragment with number N_{cf} ;
 - 4 Initializing the clustering space of X_{CF} , X_{CG} , and X_{CGE} ;
 - 5 **for** $i=1$ to N_{cge} **do**
 - 6 Generating the vector V_i for CGE_i ;
 - 7 Appending the vector V_i to X_{CGE} ;
 - 8 **end**
 - 9 **for** $i=1$ to N_{cg} **do**
 - 10 Generating the vector for CG_i ;
 - 11 Appending the vector V_i to X_{CG} ;
 - 12 **end**
 - 13 **for** $i=1$ to N_{cf} **do**
 - 14 Generating the vector V_i for CF_i ;
 - 15 Appending the vector V_i to X_{CF} ;
 - 16 **end**
 - 17 Calling WEKA to clustering X_{CF} , X_{CG} and X_{CGE} with X-means method;
 - 18 Mining clone evolutionary characteristic with the clusters of X_{CF} , X_{CG} and X_{CGE} ;
 - 19 **return** *All the clusters for clone fragment, group, and genealogy;*
-

算法 2-2 克隆演化特征聚类算法

Algo. 2-2 The algorithm of clustering for clone characteristics

2.7 实验结果与分析

2.7.1 实验设置

本章选择了两个开源软件作为本章的实验系统：分别为 ArgoUML 和 jEdit。ArgoUML 是一个领先的开源 UML 建模工具，包括对所有标准 UML 1.4 图的支持。jEdit 是程序员开发所使用的编辑器，其特点是具有易于使用的接口，类似于许多流行的文本编辑器。

表 2-1 描述了这两个实验系统的基本信息。从表中第 2-4 列给出了系统的版本信息，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 5-7 列则给出了实验系统的克隆实体的数量（即克隆聚类空间大小）。其中，“Clone Fragment”列出了所聚类的克隆片段的数量，“Clone Group”和“Clone Genealogy”则分别是所聚类的克隆组和克隆家系数数量。

表 2-1 两个开源软件实验系统信息

Table 2-1 The information of two open sources experimental projects

| 实验系统 | 版本数 | 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 |

本章从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、克隆组和克隆家系。因此，克隆演化特征的实验也可以划分成为三个部分，分别为：

- 克隆片段演化特征分析实验：在克隆片段实验中，聚类克隆片段实体的向量空间，将分析克隆片段在演化过程中的变化情况，挖掘克隆片段变化的演化特征。
- 克隆组演化特征分析实验：在克隆组实验中，聚类克隆组实体的向量空间，将分析克隆组在演化过程中的克隆演化模式情况，并挖掘克隆组实体的一致性和不一致变化模式。
- 克隆家系演化特征分析实验：在克隆家系的实验中，聚类克隆家系实体的向量空间，从全局的角度分析了系统中全部克隆代码的演化规律，并挖掘在整个演化过程中克隆代码的稳定性以及一致性变化的演化特征。

在每个实验中，又将克隆代码演化特征的挖掘分成两个子任务：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。因此，使用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属性值分布情况，进而发现一些基本的分布特征。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在演化中克隆代码的演化特征。

2.7.2 克隆片段演化特征分析实验

克隆片段是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。考虑克隆片段实体的三个属性特征，分别是克隆寿命 (Clone Life)、是否发生变化 (Ischanged) 和历史变化次数 (Change Times)，将帮助挖掘克隆片段在其演化过程中的真实变化情况。

2.7.2.1 克隆片段统计分析实验

对克隆片段“历史变化次数”进行统计分析，分析结果如表 2-2 和表 2-3 所示。表中统计了克隆代码片段的变化情况，其中数字“0”表示从未发生变化，数字“N”表示了截止到当前版本克隆片段发生了 N 次变化。

从表 2-2 和表 2-3 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，仅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。但是，依然存在一定数量的克隆代码片段发生了变化。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 |
|--------------|-------|------|-----|---|
| 数量 | 24327 | 982 | 109 | 4 |
| 总数 | 24327 | 1095 | | |

2.7.2.2 克隆片段聚类分析实验

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。从表中可以看出，X-means 聚类将 ArgoUML 和 jEdit 的克隆代码片段实体分成 4 个

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 |
|--------------|------|-----|-----|----|----|----|----|---|
| 数量 | 5885 | 533 | 135 | 47 | 14 | 10 | 11 | 1 |
| 总数 | 5885 | 751 | | | | | | |

Cluster: Cluster0-Cluster3。根据其聚类结果分别统计了四个 Cluster 的属性特征的信息, 并使用平均值 (Mean)、标准差 (Standard Deviation, SD) 和中位数 (Median) 来描述每一个 Cluster 中的属性的分布情况。

从表中可以看出, 在 ArgoUML 和 jEdit 两个系统中, 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), 将这些克隆片段称为“发生变化的”(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), 克隆代码发生变化的时刻往往是其存在于系统中一段时间后 (Cluster0 的 Clone Life 数值要大于 Cluster3 的 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将它们称为“未发生变化”的克隆代码片段。对于未发生变化的克隆代码片段, 从表中第 2 列 (数量和比例) 可以看出, 该类克隆代码片段, 占到全部克隆片段数量的大多数。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, 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。

2.7.3 克隆组演化特征分析实验

克隆片段实体仅提供了克隆片段本身的变化情况, 而通过对克隆组实体进行实验分析, 则可以提供了克隆代码以克隆组为单位的在两个相邻版本的演化情况。

表 2-4 ArgoUML 中克隆片段的聚类结果

Table2-4 Clustering results of clone fragment for ArgoUML

| Cluster | 数量 (比例) | 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 | 数量 (比例) | 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 |

在克隆组实体实验中,通过统计和聚类克隆组在演化过程中的克隆演化模式,可以揭示克隆代码的演化特征。

2.7.3.1 克隆组统计分析实验

统计克隆组实体的“克隆演化模式”(Clone Pattern)分布情况,其结果如表 2-6 和 2-7 所示。表中第二行和第三行使用“Present”和“Absent”来标识某一克隆组实体是否具有某种具体克隆演化模式。其中,“Present”表示拥有某演化模式,“Absent”表示没有³。同时,本节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)。表中最后一行给出具有某一克隆演化模式的克隆组实体的比例。

从表中可看出,在两个实验系统中,大多数的克隆组实体(比例约 72%-85%)具有有稳定的克隆演化模式(“Static”和“Same”),只有一小部分克隆组(其比例小于 7%)具有动态的克隆模式,即拥有“Add”,“Sustract”,“Split”,“Consistent/Inconsistent Change”模式。

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”。

值得注意的是,在 ArgoUML 和 jEdit 中有数百个克隆组实体具有 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——特别是一致性变化模式(Consistent Change)——会导致额外的维护代价甚至会引发相关的克隆缺陷。在这些发生变化的克隆组实体中,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

| | 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 |
| 比例 | 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

| | 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 |
| 比例 | 79.3% | 85.20% | 1.99% | 1.60% | 6.21% | 1.82% | 0.84% |

2.7.3.2 克隆组聚类分析实验

使用 X-means 方法对克隆组实体进行聚类分析,聚类分析的结果如表 2-8 和 2-9 所示。与克隆片段聚类实验相似,聚类结果也分成了四类:Cluster0-3。相似地,也分别统计了每一 Cluster 中的克隆组实体的演化模式的分别情况,并使用“Mean”表示平均值、“SD”表示标准差、“Median”表示中位数。

从上述两个表中可以看出,Cluster1 的克隆组实体的数量最多(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)。

从表中还可以看出, Cluster0 和 Cluster2 都是动态克隆组, 因为这些克隆组实体都具有动态的克隆演化模式。在发生变化的克隆组实体中, 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。值得注意的是, 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。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实体中, 但是它们的数量也会非常小。

从 Cluster0 和 Cluster2 中克隆组的绝对数量上看, 具有一致性变化模式 (Consistent Change Pattern) 的克隆组 (Cluster2) 的数量大于具有不一致变化模式 (Inconsistent Change Pattern) 的克隆组数量 (Cluster0)。这意味着一致性变化模式相比于不一致变化模式更容易发生。因此, 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。

最后, 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需要考虑克隆组的变化情况以及对系统的影响问题。

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。

2.7.4 克隆家系演化特征分析实验

克隆家系可以提供系统中克隆代码的全局视角, 因此本节对克隆家系实体进行实验分析。在克隆家系实验中, 依然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。

2.7.4.1 克隆家系统计分析实验

本节统计克隆家系的所有属性值 (包括克隆寿命和克隆演化模式数量), 统计结果如图 2-4 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, 即所选用的克隆家系的属性值, 纵坐标表示克隆家系属性值的数值范围。

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

表 2-8 ArgoUML 中克隆组的聚类结果

Table2-8 Clustering results of clone group for ArgoUML

| Cluster | Metric | Life | Static | Same | Add | Subtract | Consistent | Inconsistent | Split |
|----------|--------|-------|--------|-------|-------|----------|------------|--------------|---------|
| 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 |

表 2-9 jEdit 中克隆组的聚类结果

Table2-9 Clustering results of clone group for jEdit

| Cluster | Metric | Life | Static | Same | Add | Subtract | Consistent | Inconsistent | Split |
|----------|--------|---------|--------|-------|-------|----------|------------|--------------|---------|
| 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 |

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。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的。

2.7.4.2 克隆家系聚类分析实验

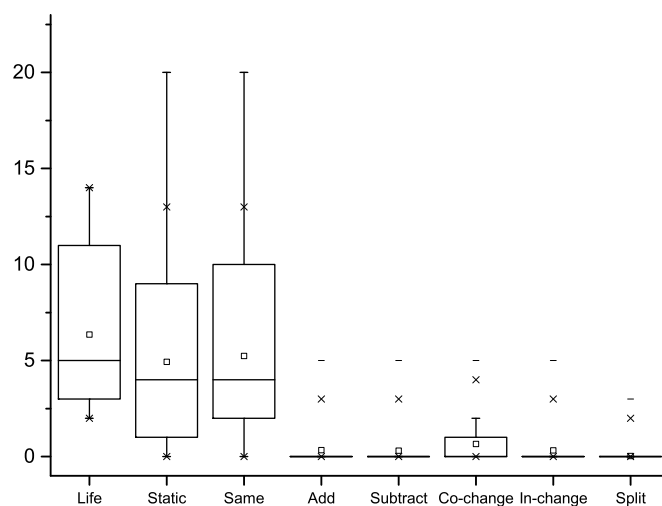
使用 X-means 对系统中克隆家系实体的演化情况进行聚类分析(即克隆寿命和克隆演化模式数量), 进而挖掘更多的克隆演化特征, 结果如表 2-10 和 2-11 所示。

表中定义了一个“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死亡。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, 本节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 4 个 Cluster: Cluster0-Cluster3。

为了分析克隆家系的演化情况, 本节非正式给出“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(Stactic)”和“相同(Same)”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。

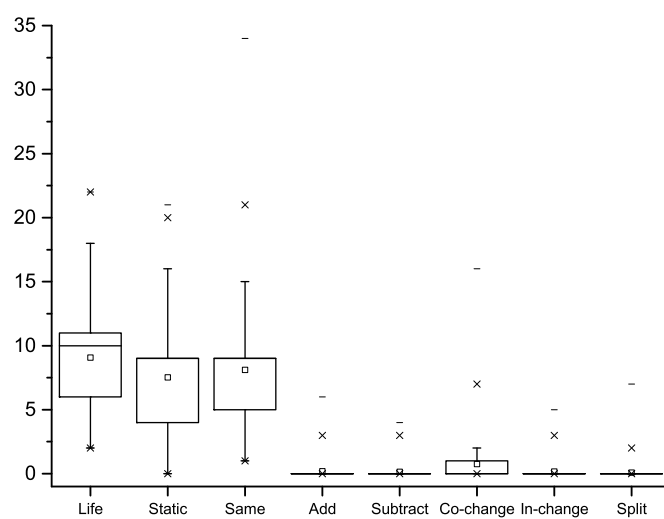
从图 2-4 和表 2-10 和 2-11 中, 可以很明显的看出, 大多数的克隆家系是“稳定的”克隆家系(Cluster1, 2, 3), 并且克隆家系寿命和“稳定的”克隆模式之间存在较强的正相关关系。另一方面, “动态的”克隆家系(Cluster0)数量十分稀少, 并且具有较长的寿命且依然存在于系统中。这表明“动态的”克隆演化模式—特别是一致性变化和 inconsistency 变化—通常发生在寿命较长的克隆家系中。因此, 软件开发人员应该对动态的克隆家系采取一些必要的措施, 因为不一致性的变化可能导致软件缺陷。

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 inconsistency 变化演化模式。



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 2-4 克隆家系的演化模式统计

Fig.2-4 The statistics of clone genealogy evolution

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。

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 |

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 |

2.8 本章小结

为挖掘克隆代码及其演化过程的克隆代码演化特征，本章提出一个挖掘克隆代码演化特征的分析方法。该方法可以分析多版本软件的演化情况，使用聚类分析 (X-means) 的方法挖掘和分析克隆代码演化特征，帮助程序开发人员理解克隆代码。从克隆片段、克隆组和克隆家系实体三个不同角度，分析和表示克隆代码及其演化情况，并提取相应属性值生成克隆演化特征分析的聚类向量。克隆片段角度重点关注克隆代码的实际变化情况，克隆组角度关注克隆组在相邻两个版本的演化情况，克隆家系则提供了系统中全部克隆代码演化的全局视角。在 ArgoUML 和 jEdit 两个软件系统上的实验结果表明：克隆代码在其演化过程中通常是非常稳定的，不会频繁的发生变化。同时，在其演化过程中，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。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。本章所挖掘的克隆演化特征，不仅可以帮助开发人员更好地理解克隆，提高系统的可理解性；还可以提供一些建议指导开发人员维护系统中克隆代码，为本文后续的克隆代码一致性维护研究奠定了基础。

第3章 克隆代码创建一致性维护需求预测方法

3.1 引言

在软件开发过程中，程序开发人员通过复制和粘贴操作复用软件中既有的代码，从而向系统中引入新的克隆代码。新产生的克隆代码在随着软件系统的演化过程中，可能会发生一致性变化而导致软件的额外维护代价。遗忘该一致性变化还会引入与之相关的克隆一致性违背缺陷，从而将进一步增大系统的维护代价。因此，在克隆代码创建时（复制粘贴操作发生时），预测在其演化过程中是否会发生一致性变化，将有助于降低软件的维护代价。因此，如何在克隆代码创建时，预测克隆代码的一致性变化是一个值得研究的问题，可以帮助提高软件质量和可维护性。

鉴于此，本章在定义克隆代码创建的一致性变化和一致性维护需求的基础上，在克隆代码创建时（复制和粘贴时），使用机器学习方法预测克隆代码的一致性维护需求。首先，通过构建软件系统的克隆家系，收集系统中所有的克隆创建实例（复制粘贴操作），并识别克隆创建实例中被复制和粘贴的克隆代码。然后，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。最后，使用机器学习方法训练预测模型，并在克隆代码创建时预测其一致性维护需求。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可以高效地预测克隆代码创建的一致性维护需求，F 值在 79.4%-95.7% 之间，精确率和召回率分别介于 79.3%-95.8% 和 79.4%-95.8%。本章所提出的预测方法可以帮助程序开发人员在克隆代码创建时预测克隆代码的一致性变化，降低克隆代码导致的额外的维护代价。

3.2 克隆代码额外维护代价的相关研究

3.2.1 克隆代码的维护代价问题

研究表明，复制粘贴操作是导致克隆代码产生的主要原因，但是所引入的克隆代码可能会导致额外的软件维护代价。图 3-1 是会导致额外维护代价的克隆代码。开发人员复制了 23 行的代码片段 (Clone Fragment 1)，然后粘贴至其它地方 (Clone Fragment 2)。在通过此复制粘贴操作引入该克隆代码后，在其演化的第 2、12、32 和 33 个月，这两个克隆代码片段发生了如下 4 次修改：（1）开发人员修改了变量

cockpitServers, cockPorts 和 collectionDirs (图中 Line 2、4、6), 将其重命名为其它的变量 (如 cockpitServers 变为 m_cockpitServers)。由于这两个代码片段存在于同一个方法中, 并且访问了局部变量, 因此图中 Line 2、4、6 也发生了相应的变化。(2) 开发人员在 Line 3 后插入了一个函数调用 Config.GetParameter 获取额外的配置信息 m_cockpitEnvs。(3) 为了遵守新的命名规则, 开发人员对 Clone Fragment 1 的变量 PPE 新增加前缀 StaticRank3, 并且对 Clone Fragment 2 的变量 PROD 做了相同的操作。(4) 开发人员将 Line 21 的 LogLevel.Warning 更改为 LogLevel.Error。上述操作要求开发人员同时修改两个克隆代码片段, 从而会引发克隆代码的额外维护代价。

| Code Fragment 1: | Code Fragment 2: |
|---|--|
| <pre> 1 try{ 2 cockpitServers ["PPE"]=Config.GetParameter(c_iqmFile, 3 "PPE", "cockpitServer", c_ppeCockpitServer); 4 cockpitPorts ["PPE"]=Config.GetIntParameter(5 c_iqmFile, "PPE", "cockpitPort", c_ppeCockpitPort); 6 collectionDirs ["PPE"]=Config.GetParameter(c_iqmFile, 7 "PPE", "collectionDir", c_ppeCollectionDir); 8 if (Config.GetIntParameter(9 c_iqmFile, "PPE", "rankDataAvailable", 10 c_ppeRankDataAvailable) == 1){ 11 m_numCollections++; 12 rankDataAvailable ["PPE"] = true; 13 } 14 if (Config.GetIntParameter (15 c_iqmFile, "PPE", "crawlDataAvailable", 16 c_ppeCrawlDataAvailable) == 1) { 17 m_numCollections++; 18 crawlDataAvailable ["PPE"] = true; 19 } 20 }catch{ 21 Logger.Log (LogID.IQM, LogLevel.Warning, "Unable 22 to get cockpit Server or cockpitPort for "PPE"); 23 }</pre> | <pre> 1 try{ 2 cockpitServers ["PROD"]=Config.GetParameter(c_iqmFile, 3 "PROD", "cockpitServer", c_productionCockpitServer); 4 cockpitPorts ["PROD"]=Config.GetIntParameter(5 "PROD", "cockpitPort", c_productionCockpitPort); 6 collectionDirs ["PROD"]=Config.GetParameter(c_iqmFile, 7 "PROD", "collectionDir", c_productionDir); 8 if (Config.GetIntParameter(9 c_iqmFile, "PROD", "rankDataAvailable", 10 c_productionRankDataAvailable) == 1){ 11 m_numCollections++; 12 rankDataAvailable ["PROD"] = true; 13 } 14 if (Config.GetIntParameter (15 c_iqmFile, "PROD", "crawlDataAvailable", 16 c_productionCrawlDataAvailable) == 1) { 17 m_numCollections++; 18 crawlDataAvailable ["PROD"] = true; 19 } 20 }catch{ 21 Logger.Log (LogID.IQM, LogLevel.Warning, "Unable 22 to get cockpit Server or cockpitPort for "production"); 23 }</pre> |

图 3-1 会导致额外维护代价的克隆代码 [127]

Fig.3-1 An example for code fragments that resulting extra maintenance cost

但是并非所有的克隆代码都会导致额外的软件维护代价, 图 3-2 是不会导致额外维护代价的克隆代码。开发人员复制一段代码片段, 并将其粘贴至其它位置。在引入此克隆代码后, 该克隆代码一直安静的存在于系统中长达三年的时间, 直到包含克隆代码的功能模块被移除。因此, 对于此类复制粘贴所导致的克隆代码, 并不会引发额外的维护代价。

```

1  if ( c >= sSource.Length ||
2      (c == sSource.Length - 1 && sSource[c] == `&` )){
3      break;
4  }
5  if ( sSource[c] == `&` ) {
6      try{
7          if ( sSource[c+1] == `q` && sSource[c+2] == `=` &&
8              sTag.ToLower() == "&q=" ){
9              c += 3;
10             }else{
11                 break;
12             }
13         }catch ( Exception e){
14             Console.WriteLine (e.Message);
15         }
16 }

```

图 3-2 不会导致额外维护代价的克隆代码 [127]

Fig.3-2 An example for code fragments that not resulting extra maintenance cost

3.2.2 现有方法存在的问题

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 [23]。复用既有代码可以减少软件开发时间、提高软件开发效率，但同时也会向软件系统中引入大量的克隆代码。复制和粘贴导致的克隆代码可能引发额外的维护代价，如图 3-1 所示。

针对克隆代码的维护代价问题，当前最主流的解决方式是使用重构的方法消除系统中的克隆代码。拟通过重构的手段消除克隆代码，进而避免克隆代码维护代价。目前的重构研究主要集中在可重构分析 [85][86][87][88]、重构调度 [89][90][91]、重构方法 [95][96][97][98] 等方面（见本文第 1.2.4.1 节克隆代码重构）。

此外，为了有效地管理克隆代码，也对克隆代码进行跟踪，如跟踪克隆代码的产生。由于复制粘贴操作是导致克隆产生的最主要原因，因此通过监测程序员的复制粘贴操作可以跟踪克隆代码的产生。例如，许多克隆跟踪工具 CLONEBOARD[128]、CnP[129]、CPC[130]、CReN[131]、CSeR[132] 等。但是，上述方法对新产生的克隆代码，并没有进行是否会导致额外维护代价的判断，而是将所有的克隆代码全部引入系统中。因此，也无法帮助降低克隆代码的额外维护代价。

目前对克隆代码研究无法有效地降低克隆代码的维护代价，依然存在以下两点不足：

（1）现有的克隆代码维护方法主要采用先检测后重构的方法，目标是消除克隆代码，而不是规避克隆代码的产生。这种基于重构的克隆代码维护方法不利于实

现边开发边维护克隆代码，不能从源头上规避克隆代码的产生进而降低软件维护代价，而且重构本身的代价也很高，并非所有的克隆代码都适合重构，重构无法从根本上解决克隆代码的维护问题。

(2) 在现有的克隆管理中的克隆代码跟踪研究中，通过跟踪复制粘贴操作跟踪克隆代码的产生，但是并未对复制粘贴操作加以控制，对这种操作是否需要一致性维护缺少必要分析和判断。

综上，由于开发人员的复制粘贴操作会向系统中引入克隆代码，而克隆代码的一致性变化会导致额外的维护代价，这会降低软件系统的质量和可维护性。在软件开发过程中，帮助开发人员规避导致额外维护代价的克隆代码的产生，有助于降低克隆代码的额外的维护代价。因此，结合软件开发过程，在克隆代码创建时（复制和粘贴操作发生时），预测克隆代码的一致性维护需求是一个值得研究的问题，可以帮助规避会引入额外维护代价的克隆代码，从而降低软件的维护代价，提高软件质量和可维护性。

3.2.3 本文的解决思路

为了在克隆代码创建时，避免其可能导致额外的维护代价，本章将解决克隆一致性需求维护预测问题。在克隆代码演化的基础上，结合克隆代码一致性变化所导致的额外维护代价，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一致性维护需求。然后，本文在 Wang[127] 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属性值。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，可以更为细致地表示新创建的被复制和被粘贴的克隆代码。最后，使用五种不同的机器学习方法预测克隆代码创建的一致性维护需求。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，提高软件的质量和可维护性。

本章的克隆代码创建一致性维护需求预测框架如图 3-3 所示。方法可以划分为三个阶段，复制和粘贴操作（克隆创建实例）收集阶段、特征提取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复制和粘贴操作，将其用于使用机器学习方法中来训练预测模型。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。首先通过构建系统的克隆家系，然后，将第一次出现在系统的克隆代码标记为由复制和粘贴操作导致的克隆创建实例。最后，通过遍历系统全部克隆家系的根节点，可以收集

系统所有的克隆创建实例。

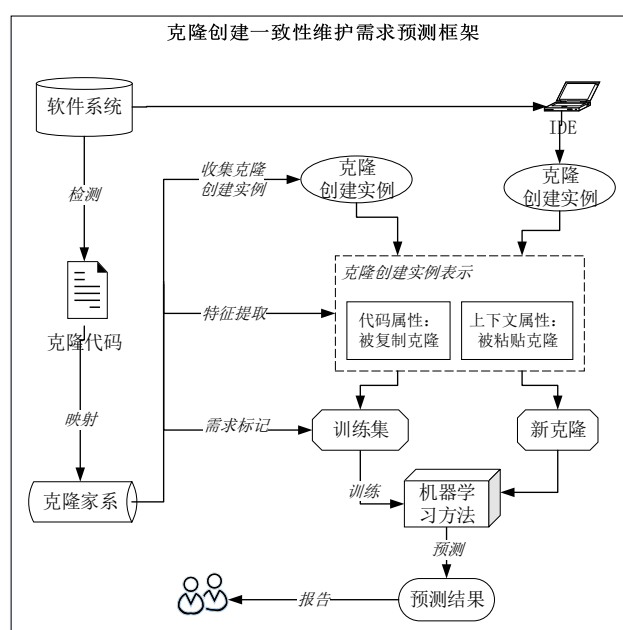


图 3-3 克隆创建一致性维护需求预测框架

Fig.3-3 The framework for clone creating consistency prediction

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，因此在特征提取阶段中将提取相应的属性特征表示克隆创建实例。分别提取了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。

在预测步骤中，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。如果该复制和粘贴操作会导致一致性变化，程序开发人员可以据此拒绝此粘贴被复制的克隆代码，从而避免额外的维护代价。如果该复制粘贴操作不会导致一致性变化（不会引发额外的维护代价），程序开发人员可以放心的复用该克隆代码，从而提高软件开发的效率。

本章在克隆代码创建时，预测克隆代码的一致性维护需求，本章将重点分析和讨论如下问题：

- （1）在克隆代码创建时（复制和粘贴克隆代码时），结合其可能导致的额外维护代价，如何描述和定义其演化中的一致性变化与一致性维护需求？
- （2）对于克隆创建实例（复制和被粘贴操作导致的克隆代码），如何进一步深入的描述和表示创建实例中的被复制的克隆代码和被粘贴的克隆代码？
- （3）在预测克隆创建一致性维护需求时，如何结合软件开发过程，帮助程序开

发人员避免额外的维护代价？

3.3 克隆创建一致性维护需求的定义

如图 3-1 所示，克隆代码的演化过程中，克隆代码可能会发生一致性变化，从而引发额外的维护代价。为了描述这种克隆代码的变化，先定义了克隆代码创建的一致性变化，如下所示：

定义 3.1 (克隆创建一致性变化) 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的一致性变化 (Creating Consistent Change)，

$$textSim(CF_i, CF'_i) < 1 \quad \forall i \in \{1, 2\}$$

在该定义中，克隆代码的变化使用 $textSim(CF_i, CF'_i) = 1 - UPI(CF_i, CF'_i)$ 描述。其中，UPI (Unique Percentage Items[13]) 表示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%。值得注意的是，创建时的一致性变化，仅要求克隆代码片段 CF_1 和 CF_2 被同时修改。

在上述定义中，仅要求克隆代码片段被同时的修改。原因在于：在向系统中引入新的克隆代码时（克隆代码创建时），本章的目标是避免由新创建的克隆代码所导致额外的维护代价。而本章认为只要克隆代码同时发生变化，即导致系统的维护代价。因此，该定义仅要求两个克隆片段在其演化过程同时发生变化即可。

克隆创建所导致的克隆代码，会随着系统的演化而演化。在演化过程中，克隆代码片段之间的变化，会引发其所在克隆组的变化，本文使用克隆演化模式描述克隆组的变化。因此，根据克隆片段的一致性变化定义，可以将克隆组的一致性变化模式定义如下：

定义 3.2 (克隆创建一致性变化模式) 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)。

本章定义的克隆组的一致性变化模式，仅仅要求组内至少存在两个克隆代码片段被同时修改。对于一个克隆组来讲，组内至少存在两个克隆代码片段。假定要

求组内全部的克隆代码片段都同时变化，为一致性变化模式。那么此种情况下的定义过于狭隘。原因在于：即使组内仅有两个克隆代码片段同时发生变化，即会导致额外的维护代价；而要求组内全部克隆片段全部同时变化，将无法准确的识别两个片段变化所导致的维护代价。

如前文所述，系统中的大部分的克隆代码是由复制和粘贴操作所导致的。并且，使用克隆家系还可以描述克隆代码的演化过程。因此，在一个克隆家系中，本文认定克隆家系的根节点就是克隆代码创建的时刻，根节点的克隆组是由程序开发人员的复制粘贴操作所创建的克隆代码（克隆创建实例）。因此，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) 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，且该组的克隆代码是由复制粘贴操作引入的。

给定一个由复制粘贴操作所创建的克隆组，在其演化过程中，可能会发生一致性变化，进而引发额外的维护代价。假设在创建该克隆代码时可以预测其是否可以发生一致性变化，可以避免额外的维护代价。因此，本文将由复制粘贴操作创建的克隆代码，其演化过程中发生的一致性变化，称为克隆创建一致性维护需求，定义如下：

定义 3.4 (创建一致性维护需求) 给定版本 j 中一个克隆创建实例 CG ， 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 Consistency-Free，或 Free)。

根据上面的定义描述，只要克隆代码在其演化过程中，发生过一致性变化模式，即认定其满足克隆一致性维护需求。最终，可以将本章的研究问题可以表述如下：给定一个克隆创建实例 CG ，例如由复制粘贴操作导致的克隆代码，预测该创建实例 CG 是否满足克隆创建的一致性维护需求。

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。

3.4 复制和粘贴操作的样本获取

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克隆创建实例，即复制粘贴操作导致的克隆代码。因此，通过检测系统的克隆代码并构建克隆家系，可以收集系统中的复制和粘贴操作。本章将使用 NiCad 来检测软件版本中的所有克隆。通过在相邻版本的克隆组之间进行映射来构建克隆家系。

(1) 构建系统的克隆家系

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*[45]，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 [123][124]。根据克隆映射结果，构建系统的克隆家系。

(2) 收集复制粘贴操作并标识一致性维护需求

克隆家系是一个克隆组演化的有向无环图，图中根节点是由复制粘贴操作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，即复制和粘贴操作。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（定义 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) 确认被复制和被粘贴的克隆代码

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

¹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。

Input: All source code and code clones from N versions

Output: All copy-and-paste operations

```

1 for  $i=1$  to  $N$  do
2   Generating CRD to represent each code cloneCFs;
3   Mapping all the  $CF$ s and  $CG$ s between the adjacent
   versions  $i$  and  $i+1$ ;
4 end
5 Generating all clone genealogies  $CGEs$  with number  $N\_cge$ ;
6 for  $i=1$  to  $N\_cge$  do
7   Collecting the clone creating instance from  $CGE\_i$ 
   according to Definition 3.3 ;
8   Identify all clone consistent change pattern in thus two
   versions according to Definition 3.2 ;
9   Labeling its consistency-requirement according to
   Definition 3.4 ; Labeling the copied and pasted code
   fragments by mapping code clones to the last version;
10 end
11 return All copy-and-paste operations;

```

算法 3-1 复制粘贴操作收集算法

Algo. 3-1 The algorithm for collecting copy-and-pasted operations

收集系统中复制和粘贴操作算法如 3-1 所示。算法中第 1-4 行是为克隆代码生成 CRD，并使用 CRD 映射相邻版本的克隆代码假设一个版本中有 m 个克隆代码，生成 CRD 所需时间为 $O(m)$ ， n 个版本所需时间为 $O(m * n)$ 。映射克隆相邻版本的克隆代码时间复杂度为 $O(m)$ 。算法的第 5 行为根据映射结果生成系统的克隆家系。第 6-10 行，先根据克隆创建实例定义，通过遍历克隆家系收集系统复制和粘贴操作。然后，根据克隆演化模式定义，识别克隆演化模式，并标记其一致性维护需求。最后，确认复制粘贴操作中被复制和被粘贴的克隆代码。算法复杂度为 $O(n)$ 。所以，该算法的复杂度为 $O(m * n)$ 。

3.5 复制粘贴操作的特征描述

在使用机器学习中分类方法进行分类或者预测时，例如本章预测克隆代码创建的一致性维护需求，需要提前构建和训练机器学习模型。但是，实际的复制粘贴

操作无法直接应用于机器学习中。

因此，本节将提取相应的属性值表示克隆创建实例（复制和粘贴操作）。一个克隆创建实例，即复制粘贴操作，由两个克隆代码片段构成：被复制的克隆代码代码和被粘贴的克隆代码。因此，本节将提取不同的属性用于描述复制和粘贴操作。提取代码属性表示被复制的克隆代码、提取上下文属性属性表示被粘贴的克隆代码。

3.5.1 被复制克隆代码的特征

在克隆创建实例中，被复制的克隆代码代表系统中最初始的代码片段。因此，可以从代码本身的角度去描述被复制的克隆代码特征，即代码属性。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。
- 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。

3.5.2 被粘贴克隆代码的特征

对于克隆创建实例中的被粘贴的克隆代码，其包含了一些与克隆关系相关的特征，即上下文属性。上下文属性是代克隆创建实例中克隆代码之间的关系属性，描述了两者的克隆关系信息。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。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 [133] 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, \dots, P_m) 和 (Q_1, Q_2, \dots, 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, \dots, P_m) 和 (Q_1, Q_2, \dots, 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, \dots, P_m) 和 (Q_1, Q_2, \dots, 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 基于机器学习的克隆创建一致性维护需求预测

本章将克隆代码创建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的一致性维护需求。本章使用五种机器学习方法对克隆创建实例进行分类，从而预测克隆一致性维护需求。

在本章的研究中，使用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) [134]、朴素贝叶斯方法 (Native Bayesian, 本文简称为 Native) [135]、支持向量机方法 (Support Vector Machine, 简称为 SVM) [136]、K 近邻方法 (K-Nearest Neighbors, 简称为 KNN) [137] 和决策树方法 (Decision Tree, 本文简称为 Tree) [138]。

(1) 贝叶斯网络方法

贝叶斯网络 (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 [134]。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, \dots, X_n , 贝叶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。

(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。决策树是一个树结构（可以是二叉树或非二叉树）。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。

本章没有对机器学习方法进行改进和研究，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 中的机器学习算法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：

- 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。

基于机器学习的克隆创建一致性维护需求预测算法如 3-2 所示。算法的第 1 行为初始化训练集。第 2-6 行提取相应的属性组表示克隆创建实例，并生成训练集。

最后第 7 行，使用训练集调用 WEKA 训练机器学习模型。算法的主要时间消耗在表示克隆创建实例上，其时间复杂度为 $O(m)$ ，其中 m 为复制粘贴操作的数量。

Input: All copy-and-paste operations and source code

Output: The predictive model

```

1 Initializing the training data set Sets for all the operations;
2 for  $i=1$  to  $M$  do
3     Generating all the code attributes for copied code clones;
4     Generating all the context attributes for pasted code clones;
5     Appending all the attributes as one item to the Sets;
6 end
7 Calling WEKA to train the model on training set Sets;
8 return The predictive model;
```

算法 3-2 克隆创建一致性维护需求预测算法

Algo. 3-2 The algorithm for predicting clone creating
consistency-requirement

3.7 实验结果与分析

3.7.1 实验设置

3.7.1.1 实验系统

本章选取了四个开源软件进行实验，表 3-1 给出了实验系统的克隆创建实例信息。其中，第 2 列和第 3 列列出了克隆创建实例的数量和比例，其中第 2 列是不需要一致性维护的实例，第 3 列是需要一致性维护实例的数量和比例。对于需要一致性维护的克隆创建实例，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。这说明复用既有代码引入克隆代码，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2章得出的克隆演化特征相一致。但是值得注意的是：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复用技术。

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 |
|------------|---------------|--------------|------|
| | 不需要维护 | 需要维护 | |
| 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 |

为了详细评估本章所提出的克隆创建一致性维护需求预测方法，并回答本章所提出的两个子问题。本节实验可以划分为两个实验：有效性验证实验和使用模式实验，如下所示：

- 有效性验证实验（机器学习方法对比实验）：在此实验中，使用五种不同机器学习方法验证本章方法的预测能力。可以帮助程序开发人员选择一个最优的机器学习模型，用于克隆代码创建的一致性维护需求预测。在此实验中，将同时预测两种类别的克隆创建实例。

- 使用模式实验（贝叶斯网络方法实验）：因为有两种不同类别的克隆创建实例，即需要一致性维护和不需一致性维护。本节以贝叶斯网络方法为例，详细给出了两种状态下的具体预测结果。将分别预测两种类别的克隆创建实例，以帮助程序开发人员选择合适的“使用模式”进行克隆创建的一致性维护需求预测。

同时，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。为了评估本章所提取的属性值的有效性，本章还将每一个实验进一步划分为全属性实验和属性组实验两个部分，如下所示：

- 全属性实验：在此实验中，使用本章所提取的所有属性进行预测实验，以评估本文方法的整体预测能力。

- 属性组实验：在此实验中，分别使用两组不同的属性值进行预测实验，以评估所提取的属性值对预测结果的影响程度。

3.7.1.2 实验评估指标

在机器学习领域中，往往使用混淆矩阵（Confusion Matrix）描述一个模型的预测能力。混淆矩阵如表 3-2 所示。在表中，True positives(TP) 表示被正确地划分为正例的个数，即实际为正例且被分类器划分为正例的实例数。False positives(FP) 表示被错误地划分为正例的个数，即实际为负例但被分类器划分为正例的实例数。False negatives(FN) 表示被错误地划分为负例的个数，即实际为正例但被分类器划分为负例的实例数。True negatives(TN) 表示被正确地划分为负例的个数，即实际为负例且被分类器划分为负例的实例数。

表 3-2 混淆矩阵示意图
Table3-2 The Confusion Matrix

| 真实类别 | 预测类别 | |
|------|--------------------|--------------------|
| | 类别 0 | 类别 1 |
| 类别 0 | True Positive(TP) | False Negative(FN) |
| 类别 1 | False Positive(FP) | True Negative(TN) |

根据混淆矩阵，使用计算精确率（Precision Rate）、召回率（Recall Rate）和 F 值（F-measure）用于评价模型的预测能力。对于类别 0 来说，其精确率可如下计算：

$$Precision = \frac{TP}{TP + FP} \quad (3-1)$$

召回率可如下计算：

$$Recall = \frac{TP}{TP + FN} \quad (3-2)$$

F-Measure 可如下计算：

$$F-measure = 2 \times \frac{precision \times recall}{precision + recall} \quad (3-3)$$

在本章的一致性维护预测中，将同时预测两种不同的状态的创建实例的预测效果。首先，对于满足和不满足一致性维护需求的克隆创建实例，分别计算其精确率 (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。然后，将这两组预测结果根据实际实例的数量进行加权平均，计算平均精确率 (Average Precision)、平均召回率 (Average Recall) 和平均 F 值 (Average F-measure) 作为评价指标，计算方式如下所示：

- 平均精确率：该指标评估克隆创建实例的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的预测。首先，计算预测中满足一致性维护需求的精确率“ 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的精确率“ P_2 ”。最后，将这两个值进行加权平均，作为整个模型的精确率。“ P_1 ”是满足一致性维护需求的精确率，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精确率，且“ N_2 ”为其数量。平均精确率（Average Precision）计算如下，

$$Ave-Precision = \frac{P_1 \times N_1 + P_2 \times N_2}{N_1 + N_2} \quad (3-4)$$

• 平均召回率：该指标评估克隆创建实例的一致性维护预测的查全能力，也包含了满足和不满足一致性维护需求的预测。同精确率，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，

$$Ave-Recall = \frac{R_1 \times N_1 + R_2 \times N_2}{N_1 + N_2} \quad (3-5)$$

• 平均 F 值：该指标可以评估所有克隆创建实例的精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。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，

$$Ave-F-measure = \frac{F_1 \times N_1 + F_2 \times N_2}{N_1 + N_2} \quad (3-6)$$

3.7.2 有效性验证实验

在本节的实验中，讨论五种机器学习方法的预测能力，帮助程序开发人员选择合适的机器学习模型。因此，对每一个实验系统，同时使用五种不同的机器学习方法，同时对需要和不需要一致性维护的克隆实例进行预测。

值得注意的是，本章没有对机器学习方法本身做进一步的改进，直接调用 WEKA 中的机器学习算法进行评估和预测。在使用不同的机器学习方法时，需要对其进行一些基本的参数设置，以构建不同的机器学习模型。本文也没有对全部参数进行调优，大多选择 WEKA 中默认的参数。通过调整机器学习的参数，可能进一步增强机器学习方法的预测能力。

所采用方法的参数设置如下：贝叶斯网络的父节点数设置为 3，并且设置阈值

为 0.5。Native 方法只有一个父节点，同样设置阈值为 0.5。在 SVM 方法中，本章选择用 *polynomial kernel* 作为核函数。对 Decision Tree 方法，采用了 J48 算法作为预测方法，并设置置信度为 0.75。对 KNN 方法，使用 *Euclidean Distance* 为距离函数，并且设置邻居个数为 1。

在本节实验中，使用 10-Folds Cross-Validity 将数据集分为训练集和测试集，评估不同机器学习方法的预测能力。使用平均精确率 (Average Precision)、平均召回率 (Average Recall) 和平均 F 值 (Average F-measure) 作为评价指标。采用五种不同的机器学习方法，预测克隆创建的克隆代码的一致性维护需求，实验分成了两个部分，分别为全属性实验和属性组实验，实验结果如表 3-3 和 3-4 所示。

3.7.2.1 全属性实验结果

表 3-3 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，精确率从 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。

3.7.2.2 属性组实验结果

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 3-4 所示。表中“All”表示全部属性实验结果，属性组实验结果使用在“Code”和“Context”表示。

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著差异。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。

表 3-3 克隆创建实例一致性维护需求预测效果

Table3-3 The effectiveness of clone creating consistency for all attribute

| 指标 | 方法 | ArgoUML | jEdit | jFreeChart | Tuxguitar |
|--------------|----------|---------|-------|------------|-----------|
| 平均 Precision | 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 |
| 平均 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 |
| 平均 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 |

3.7.3 使用模式实验

与 Wang 的方法相似，本节使用贝叶斯网络方法，分别预测两种不同类别的克隆创建实例。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，同时使用 SimpleEstimator 估计贝叶斯网络的条件概率表。在实验时，采用十倍交叉验证（10-folds Cross-Validity）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。

使用贝叶斯网络方法进行预测时，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、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

表 3-4 克隆创建实例的属性组预测效果

Table3-4 The effectiveness of attribute set for clone creating consistency

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

要一致性维护。

3.7.3.1 一致性维护自由实验

对不需要一致性维护的克隆创建实例进行预测评估，帮助程序开发人员放心的执行复制和粘贴操作，从而快速安全的开发软件。实验使用三个度量评估预测效果，分别为：推荐率、精确率和召回率。其中，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。

(1) 全属性实验结果

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 3-5 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，四个系统的精确率介于 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 3-5 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，本章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。

(2) 属性组实验

实验结果如表 3-6 所示，其中左侧为代码属性，右侧为上下文属性。由表 3-6 和表 3-5 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，精确率除 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 系统外精确率提高，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精确率影响较大，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，本章建议在进行预测时，保留所有的属性进行预测。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。

3.7.3.2 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、精确率和召回率。其中，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

表 3-5 全属性组维护自由实验效果

Table3-5 The effectiveness for all attribute of clone consistency free

| 系统 | 阈值 | 推荐率 (%) | 精确率 (%) | 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 |

表 3-6 属性组维护自由实验效果

Table3-6 The effectiveness for attribute set of clone consistency free

| 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 |

中全部实例的比值。

(1) 全属性实验结果

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 3-7 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: 精确率介于 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精确率和召回率, 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, 本文构建模型的精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。

(2) 属性组实验结果

类似的, 对于一致性维护需求实例, 表 3-8 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。

3.7.4 与其它方法的对比

Wang[127] 等人也使用贝叶斯网络对克隆代码创建时的一致性维护需求进行了研究。其在四个实验系统上进行了实验, 其中两个实验系统是微软内部的系统, 另外两个系统为开源系统 jFreeChart 和 Tuxguitar。因此, 本节将本章的实验结果与 Wang 的方法进行对比, 从而验证本章所提出方法的有效性。方法对比的实验结果如表 3-9 和表 3-10 所示。

表 3-9 是所收集到的克隆创建实例信息对比。由表中可以看出, 本章所收集的克隆创建实例数量要多于 Wang 所收集的。

表 3-7 需要维护的全属性实验效果

Table3-7 The effectiveness for all attributes of clone consistency

| 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 |

表 3-8 需要维护的属性组实验效果

Table3-8 The effectiveness for attribute set of clone consistency

| 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 |

表 3-9 实验系统的克隆创建实例信息统计对比

Table3-9 The comparing 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 |
|------------------|----------------|--------------|------|
| | 不需要维护 | 需要维护 | |
| jFreeChart(本章) | 2013(59.80%) | 1353(40.20%) | 3366 |
| jFreeChart(Wang) | 1059(92.2%) | 89(7.8%) | 1148 |
| Tuxguitar(本章) | 1016(71.10%) | 413(28.90%) | 1429 |
| Tuxguitar(Wang) | 384(86.5%) | 60(13.5%) | 444 |

表 3-10 是本章方法与 Wang 的方法的预测效果对比。表中使用平均精确度 (Average Precision) 和平均召回率 (Average Recall) 评估预测效果，即加权平均需要和不需要一致性维护的预测结果。由表中可以看出，本章所提出的方法优于 Wang 所提出的方法。

表 3-10 实验系统的克隆创建实例信息统计对比

Table3-10 The comparing effectiveness for clone creating instances in two projects

| 实验系统 | 平均精确率 | 平均召回率 |
|------------------|--------|--------|
| jFreeChart(本章) | 88.30% | 88.20% |
| jFreeChart(Wang) | 58.30% | 63.00% |
| Tuxguitar(本章) | 83.10% | 83.60% |
| Tuxguitar(Wang) | 61.50% | 60.10% |

3.7.5 讨论

本节从不同的角度评估了本章所提出的克隆代码创建的一致性维护需求预测，同时对不需要和需要一致性维护需求的克隆创建实例进行了预测，并从不同的角度评估了不同的机器学习方法的预测能力。

在机器学习方法对比中，在克隆代码创建时，五种机器学习方法都可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，预测结果具有较高的精确率、召回率和 F 值。更重要的是，不同机器学习模型的预测能力仅具有较小的差异，但 SVM 方法拥有相对最佳的预测效果。因此，建议优先推荐使用 SVM 方法，预测克隆代码创建的一致性维护需求。与此同时，所提取的属性组作为一个整体在不同机器学习方法的预测中，同样起到了积极的作用。因此，建议软件开发人员在预测时保留所有的属性组进行创建时一致性预测。

在使用贝叶斯网络方法的预测中，全属性实验结果表明：本章所构建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，所产生的影响程度不一致。因此，建议维护人员使用全属性组进行克隆创建实例的一致性维护需求预测，从而将其适用到其它未验证的系统中，使之达到最佳的预测效果。

此外，与 Wang 提出的方法进行了对比，结果表明本章所提出的方法具有更好的预测能力。

3.8 本章小结

通过复制和粘贴操作复用既有代码可以向系统中引入新创建的克隆代码，然而在其演化过程中可能会发生一致性变化，从而导致额外的维护代价。为帮助软件开发人员避免克隆代码的额外维护代价，本章提出了一个克隆代码创建一致性维护需求预测方法。可在克隆代码创建时（复制和粘贴时），预测新创建的克隆代码的一致性维护需求，根据预测结果决定是否执行该复制粘贴操作。构建了软件系统的克隆家系，并通过识别克隆家系的根节点收集克隆创建实例（复制和粘贴操作）。使用不同的属性组表示克隆创建实例，使用代码属性表示被复制克隆代码，使用上下文属性表示被粘贴的克隆代码。在四个开源软件系统上进行实验，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精确率和召回率高效地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，但是对精确率和召回率有不同的影响。

第4章 克隆代码变化一致性维护需求预测方法

4.1 引言

在本文第3章的研究中，在克隆代码创建时预测其一致性维护需求，帮助程序开发人员避免会导致额外维护代价的克隆代码。但是，软件系统中已经存在相当数量的克隆代码，在随着软件系统演化的过程中，克隆代码可能会被开发人员修改而导致一致性变化；遗忘这种一致性变化，会引入与之相关的克隆一致性违背缺陷，从而增大系统的维护代价。针对软件中已存在的克隆代码，在克隆代码发生变化时（被开发人员修改时），预测该变化是否会引发其所在克隆组的一致性维护，可以避免由于遗忘该变化而导致的克隆代码一致性违背缺陷。因此，如何在克隆代码变化时，预测克隆代码的一致性维护需求是一个值得研究的问题，可以帮助提高软件质量和可维护性。

与第3章不同的是，第3章克隆代码一致性维护的目的是为了规避可能导致额外维护代价的克隆代码产生，需要监测开发人员的复制粘贴操作，关注的代码特征与复制粘贴操作相关。而本章克隆代码一致性维护的目的是为了避免克隆代码引发的一致性违背缺陷，需要监测开发人员对克隆代码的修改，关注的代码特征还包括变化所在克隆组的上下文和演化历史特征。因需求和关注点不同，所以需要从不同的角度提取不同的特征。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可以有效地预测克隆代码变化一致性维护需求。本章所提出的预测方法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违背缺陷。

4.2 克隆代码一致性维护的相关研究

4.2.1 克隆变化对软件质量影响问题

克隆代码的存在不仅有可能导致额外的维护代价，更重要的是还有可能引发相关缺陷。有研究表明，长期存在于软件中不发生变化的克隆代码既不会增加软件维护的代价，也不会引发软件缺陷。而发生一致性变化的代码尽管增加了软件维护的代价，但是不会引发软件缺陷。然而，如果忘记对需要一致性变化的克隆代码进行一致性维护时，会向系统中引入克隆一致性违背缺陷（Consistency Defect）

[60] [7]。

先给出系统 jEdit 中存在的克隆代码一致性变化示例，如图 4-1 和 4-2 所示。图 4-1 中给出了 3.1.0 版本中的两个克隆代码片段。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。如图 4-2 所示，3.1.0 版本中的第 9-12 行和第 15 行代码在版本 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。开发人员对上述两个克隆代码进行了一致性维护，从而避免了克隆代码的一致性违背缺陷。

| | |
|--|--|
| <pre> 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 }</pre> | <pre> 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 }</pre> |
|--|--|

图 4-1 版本 3.1.0 中未发生一致性变化前的同组克隆代码

Fig.4-1 Two clone fragments in one clone group before consistent change in version 3.1.0

图 4-3 是一个克隆代码一致性违背缺陷的示例。图中 *CloneA* 是一段代码，将数量为 *nRegs* 的内存数据转移值数组 `ppTotal` 中。程序开发人员将 *CloneA* 复制并粘贴至另一个位置成为 *CloneB*。*CloneB* 具有相似的功能，将数量为 *nRegs* 的内存数据转移值数组 `ppTaken` 中，并将代码中的 `ppTotal` 重命名为 `ppTaken`。在随后的演化中，发现 *CloneA* 存在潜在的边界问题，并在第 5 行添加条件语句 `if (i+1<nRegs)`，将其修改为 *CloneA'*。代码 *CloneB* 也需要添加相似的判断语句，然而遗憾的是开发人员遗忘了该一致性变化，导致 *CloneB'* 中存在一个一致性违背缺陷。因此，开发人员未对上述两个克隆代码进行一致性维护，而导致了克隆代码的一致性违背缺陷，降低了软件的质量。

| | |
|---|---|
| <pre> 1 public void delete() 2 { 3 if(!buffer.isEditable()) 4 { 5 getToolkit().beep(); 6 return; 7 } 8 9 <u>if(selection.size() != 0)</u> 10 { 11 <u>setSelectedText(null);</u> 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 } </pre> | <pre> 1 public void backspace() 2 { 3 if(!buffer.isEditable()) 4 { 5 getToolkit().beep(); 6 return; 7 } 8 9 <u>if(selection.size() != 0)</u> 10 { 11 <u>setSelectedText("");</u> 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 } </pre> |
|---|---|

图 4-2 版本 3.2.0 中发生一致性变化后的同组克隆片段

Fig.4-2 Two clone fragments in clone group after consistent change in version 3.2.0

| | |
|--|---|
| <p>Clone A</p> <pre> 1 for (i= 0; i < nRegs; i++){ 2 ppTotal[i].start = prMem[i].addr; 3 ppTotal[i].nBytes = prMem[i].size; 4 ppTotal[i].more = ppTotal[i+1]; 5 } </pre> | <p>Clone B</p> <pre> 1 for (i= 0; i < tRegs; i++){ 2 ppTaken[i].start = prMem[i].addr; 3 ppTaken[i].nBytes = prMem[i].size; 4 ppTaken[i].more = ppTaken[i+1]; 5 } </pre> |
| <p>Clone A'</p> <pre> 1 for (i= 0; i < nRegs; i++){ 2 ppTotal[i].start = prMem[i].addr; 3 ppTotal[i].nBytes = prMem[i].size; 4 if (i+1 < nRegs) 5 ppTotal[i].more = ppTotal[i+1]; 6 } </pre> | <p>Clone B'</p> <pre> 1 for (i= 0; i < tRegs; i++){ 2 ppTaken[i].start = prMem[i].addr; 3 ppTaken[i].nBytes = prMem[i].size; 4 5 ppTaken[i].more = ppTaken[i+1]; 6 } </pre> |

图 4-3 导致一致性违背缺陷的克隆代码 [117]

Fig.4-3 An example for code clones that resulting consistency defect[117]

4.2.2 现有研究存在的问题

在克隆代码随着软件系统进行演化的过程中，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，并引发克隆代码的一致性变化 [17]，如图 4-1 和 4-2 所示。对于需要一致维护的克隆代码变化，如果遗忘克隆的一致性变化，会导致克隆代码的一致性违背缺陷 [22][60]，如图 4-3 所示。

为了维护演化中的发生变化的克隆代码，研究人员进行了克隆代码一致性维护的研究。Cheng 等人 [140] 提出了一个基于 Token 的克隆代码一致性维护方法，当克隆组内的一个代码发生变化时，能够同时修改组内的其他克隆代码，保证克隆代码的一致性。但是，该方法仅未对克隆代码是否需要一致性维护进行判定。在克隆管理工具 JSync[117] 中，也支持克隆代码的一致性维护，但是该方法仅能支持标识符的一致性维护。Yamanaka 等人 [139] 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。但是，该方法会将所有的克隆代码变化通知开发人员，也没有区分对是否需要一致性维护进行判定，仍需开发人员手工验证。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 [141]。但并不是在克隆代码发生变化时进行预测，无法实时的帮助维护克隆代码。Murakami 等人在克隆代码未发生变化时，预测克隆代码是否会发生下一次变化 [142]。该方法仅仅预测了是否会发生变化，既没有在变化发生时进行预测，也没有区分克隆代码的一致性和不一致性变化。

现有对克隆代码变化维护研究依然存在以下两点不足之处：

(1) 现有的克隆代码一致性维护研究中，假定需要一致性维护的克隆代码是已知的，即已知哪些克隆代码是需要一致性维护的。在此假设条件下，研究克隆代码一致性维护方法，即如何维护此类克隆代码保证其变化的一致性。因此，当前的研究中，缺少对发生变化的克隆代码是否需要一致性维护的判定研究。

(2) 现有的克隆变化的预测方法中，仅预测哪些克隆代码会发生变化，既没有预测克隆代码发生变化的时刻，也没有预测其是否需要一致性的维护。因此，预测结果不能指导开发人员对发生变化的克隆代码进行一致性维护。

综上，当软件中的克隆代码发生变化时，由于克隆组内的克隆片段的相似性可能会引发一致性变化。遗忘一致性变化会导致克隆代码的一致性违背缺陷 (Consistency Defect)，从而降低软件系统的质量和可维护性。因此，结合软件开发过程，在克隆代码发生变化时，预测克隆代码的一致性维护需求是一个值得研究的问题，可以帮助开发人员避免一致性违背缺陷，使得开发人员可以边开发、边维护克隆

代码，提高软件质量和可维护性。

4.2.3 本文的解决思路

为在克隆代码克隆变化时，避免其可能导致的一致性违背缺陷，本章将预测克隆变化的一致性维护需求。首先，定义克隆变化时的克隆代码一致性变化以及一致性变化需求，从而可以将其应用于发生变化的克隆代码中。然后，对于发生变化的克隆代码，从其所在的克隆组的角度分别提取了代码属性和上下文属性表示该克隆组，从演化的角度提取了该克隆组的历史演化特征。最后，构建克隆变化的预测模型并预测克隆变化的一致性维护需求。本章方法可以帮助程序开发人员避免克隆一致性缺陷，提高软件的质量和可维护性。

本章的克隆变化一致性维护需求预测框架如图 3-3 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。

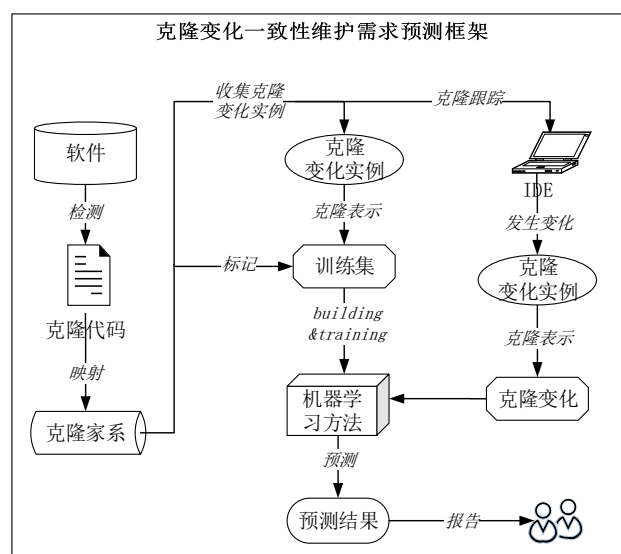


图 4-4 克隆代码变化一致性维护需求预测框架

Fig.4-4 The framework for clone changing consistency prediction

收集阶段旨在收集系统中全部的发生变化的克隆代码，可将其用于使用机器学习方法中来训练预测模型。首先，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。然后，并通过分析所映射的克隆组之间的克隆演化模式（一致性变化模式），从而收集系统中发生变化的克隆代码，并将该克隆组标记为克隆变化实例。最后，通过遍历克隆家系中的演化

模式从软件中收集全部的克隆变化实例。

在克隆变化表示阶段中，将提取相应的属性值表示克隆变化实例。与第 3 章不同的是，将从克隆组的角度提取代码属性、上下文属性，并新增演化属性一共三组属性组表示克隆变化实例。同时，还提取了关于克隆变化的相关信息，用于描述此次发生的变化。

在预测阶段中，使用收集到的克隆变化实例训练机器学习模型，并在克隆变化时预测克隆一致性维护需求。程序开发人员根据预测结果，可以采取不同的操作。对于会导致一致性变化的克隆代码变化，其在将来的演化中可能会引发缺陷。因此，需要通知程序开发人员检测整个克隆组的一致性。对于不会导致一致性变化的克隆代码变化，其不会引发克隆一致性违背缺陷。因此，在程序开发人员可以放心的对该克隆代码进行修改，不必考虑克隆代码的一致性问题。

本章针对软件中存在的克隆代码发生的变化，预测其一致性维护需求，将重点分析讨论如下问题：

- (1) 在克隆代码发生变化时（克隆片段被修改时），结合其可能导致的克隆一致性违背缺陷，如何描述和定义克隆代码的一致性变化和一致性维护需求？
- (2) 对于克隆变化实例（发生变化的克隆组），如何描述和表示发生变化的克隆组及其所发生的变化？
- (3) 在预测克隆变化一致性维护需求时，如何结合软件开发过程帮助程序开发人员避免可能导致的一致性违背缺陷？

4.3 克隆变化一致性维护需求的定义

在第 3 章中，本文给出了一个克隆创建的一致性变化定义。但是，当预测克隆代码变化时的一致性维护需求时，该定义仅仅要求同一克隆组克隆代码片段发生变化，并没有限制发生什么样的变化。该定义无法应用于本章的研究问题中。因此，对克隆变化时的一致性变化进行了重新定义，如下所示：

定义 4.1 (克隆变化一致性变化) 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的一致性变化 (Changing Consistent Change)，

$$\text{textSim}(CF_i, CF'_i) < 1 \quad \forall i \in \{1, 2\} \quad (1)$$

$$| \text{textSim}(CF_1, CF'_1) - \text{textSim}(CF_2, CF'_2) | < \tau \quad (2)$$

与第 3 章的定义不同，克隆变化时的一致性变化包含两个约束条件。第一个约

束条件要求 CF_1 和 CF_2 同时被修改，第二个约束条要求 CF_1 和 CF_2 发生相同的变化。在第二个约束条件中，一致性变化使用一个非常小的阈值 τ 描述，要求克隆片段发生了完全一致的修改（本章中 τ 取值为 0.003）。

在克隆代码变化时，不仅要求克隆代码片段被同时修改，还要求发生相同的变化。原因在于：在克隆代码变化时，本章的目的是避免由于克隆变化可能导致的克隆一致性违背缺陷。因此，不仅要求两个克隆代码片段同时变化，还需要发生相同的变化。假如软件开发人员遗忘该变化，将会引入克隆一致性违背缺陷。

克隆片段的变化必然也会导致克隆组的变化，而克隆组的一致性变化模式定义，如下所示：

定义 4.2 (克隆变化一致性模式) 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)。

同第3章相似，本节定义的克隆组的一致性变化模式，也只要求至少存在两个克隆代码片段同时发生相同的变化。其原因在于：即使发生变化的克隆组有多于两个以上的克隆代码片段，仅有两个克隆代码变化的一致性变化，也可能导致一致性违背缺陷。若要求组内全部的克隆片段发生相同的变化，将会导致部分的缺陷发生。

通过识别克隆家系中的一致性变化模式，也可以获得软件中的发生变化的克隆代码（克隆变化实例）。克隆变化实例的定义如下所示：

定义 4.3 (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）。

为更好的理解发生变化的克隆代码，即克隆变化实例。现给出一个克隆组的克隆家系以及克隆变化实例，如图 4-5 所示。如图所示，图中给出一个克隆组 CG 的克隆家系，该克隆组在版本 j 到 $j+1$ 的演化过程中，克隆代码片段发生了变化，因此克隆组 CG_j 是一个克隆变化实例。值得注意的是，该变化可能是一致性变化也可能是非一致性变化。

当演化中的克隆代码变化时，其所在的克隆组是一个克隆变化实例，该变化可能会导致克隆组的一致性变化，进而可能引发克隆一致性违背缺陷。假设在克隆代码变化时预测其是否会导致一致性变化，则帮助避免一致性违背缺陷。因此，本

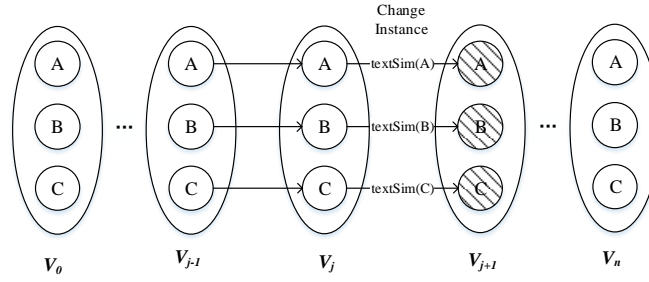


图 4-5 克隆变化实例示意图

Fig.4-5 An example for clone changing instance

文将克隆变化所导致的一致性变化，称为克隆变化一致性维护需求，定义如下：

定义 4.4 (克隆变化一致性维护需求) 给定版本 j 中一个克隆变化实例 CG ， 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 (Consistency-Requirement Free，或 Consistency-Free，或 Free)。

根据上面的定义，对于某一个克隆变化实例 CG_j ，假如其会导致一致性变化，要求在 CG_j 在未来的演化中发生一致性变化模式即可，而不是仅仅限定 CG_{j+1} 拥有一致性变化模式。原因在于：Mondal 等人的研究发现克隆代码的一致性变化可能具有延迟传播现象。延迟传播指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 [59]。值得注意的是，本章的定义可以同时涵盖 CG_{j+1} 拥有一致性变化模式和可能发生延迟传播的情况。

根据上面的定义，本章的研究问题可以表述如下：给定一个克隆变化实例 CG ，即克隆组中 CG 的克隆代码片段 CF 被修改，确定 CG 是否满足克隆变化一致性维护需求。

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。可以使用机器学习模型解决此分类问题，具体方法见下文。

4.4 克隆变化实例的获取

收集克隆变化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 [45]，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 [123][124]。根据克隆映射结果，构建系统的克隆家系。

(2) 识别克隆演化模式并收集克隆变化实例

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 2.5 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（定义 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（定义 4.2），则该实例满足一致性维护需求，否则不满足维护需求。

Input: All source code and code clones from N versions

Output: All the clone changing instances

```

1 for  $i=1$  to  $N$  do
2   Generating CRD to represent each code clone CFs;
3   Mapping all the  $CFs$  and  $CGs$  between the adjacent
   versions  $i$  and  $i+1$ ;
4   Identifying all clone evolution pattern in thus two versions
   according to Definition 2.5 ;
5   Identify all clone consistent change pattern in thus two
   versions according to Definition 4.2 ;
6 end
7 Generating all clone genealogies  $CGEs$  with number  $N_{cge}$ ;
8 for  $i=1$  to  $N_{cge}$  do
9   Collecting the clone changing instance from  $CGE_i$ 
   according to Definition 4.3 ;
10  Labeling its consistency-requirement according to
   Definition 4.4 ;
11 end
12 return All clone changing instances;
    
```

算法 4-1 克隆变化实例收集算法

Algo. 4-1 The algorithm for collecting clone changing instances

克隆创建实例收集算法如 4-1 所示。算法中第 1-6 行是为克隆代码生成 CRD, 并使用 CRD 映射相邻版本的克隆代码, 并根据克隆演化模式定义, 识别克隆演化模式。假设一个版本中有 m 个克隆代码, 生成 CRD 所需时间为 $O(m)$, n 个版本所需时间为 $O(m * n)$ 。映射克隆相邻版本的克隆代码时间复杂度为 $O(n)$ 。第 7 行为是生成系统的克隆家系。第 8-11 行, 收集克隆变化实例并标识其一致性维护需求。首先, 根据克隆家系中的一致性变模式化, 收集克隆变化实例。然后, 根据克隆变化实例的后续演化过程中的克隆组的是否发生一致性变化, 标记其一致性维护需求。算法复杂度为 $O(n)$ 。所以, 该算法的复杂度为 $O(m * n)$ 。

4.5 克隆变化实例的特征描述

本章使用机器学习中分类方法预测克隆代码变化的一致性维护需求，并使用软件中既有的克隆变化实例训练不同的机器学习模型。但是，实际的克隆变化实例无法直接应用于机器学习中。

因此，本节将提取相应的属性值表示克隆变化实例。对于一个克隆变化实例，其本质上是一个发生变化的克隆组。对于该发生变化的克隆组，将从克隆组的角度分别提取代码属性特征、上下文属性特征和演化属性特征描述该克隆变化实例。值得注意的是，克隆变化实例的代码属性和上下文属性与本文第3章中克隆创建实例的属性不同，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的在发生变化前的整个演化过程。

4.5.1 代码属性特征

代码属性从代码自身的角度描述了克隆变化实例的克隆组的克隆代码特征。代码属性描述了克隆组的词法、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3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。

4.5.2 上下文属性特征

上下文属性是克隆变化实例的克隆组的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信息。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 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 [133] 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, \dots, P_m) 和 (Q_1, Q_2, \dots, 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, \dots, P_m) 和 (Q_1, Q_2, \dots, 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, \dots, P_m) 和 (Q_1, Q_2, \dots, 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。

4.5.3 演化属性特征

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。

最后，对于克隆变化实例，本节还收集了与此次变化相关的一些变化信息，用于描述克隆变化实例所发生的变化。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，变化信息如下：

- 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 模型的构建与预测

本章将克隆代码变化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的一致性维护需求。本章使用五种机器学习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，即贝叶斯网络方法、朴素贝叶斯方法、支持向量机方法、K 近邻方法和决策树方法（方法简介可参考本文第 3.6 节）。

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：

- 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的，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，也不会导致一致性违背缺陷。

Input: All clone changing instances and source code

Output: The predictive model

```

1 Initializing the training data set Sets for clone changing
  instances;
2 for  $i=1$  to  $M$  do
3   Generating all the code attributes for clone instance;
4   Generating all the context attributes for clone instance;
5   Generating all the evolution attributes for clone instance;
6   Generating all the change attributes for clone instance;
7   Appending all the attributes as one item to the Sets;
8 end
9 Calling WEKA to train the model on training set Sets;
10 return The predictive model;
```

算法 4-2 克隆变化一致性维护需求预测算法

Algo. 4-2 The algorithm for predicting clone changing
consistency-requirement

克隆变化一致性维护需求预测算法如 4-2 所示。算法的第 1 行为初始化训练

集。第 2-8 行提取相应的属性组表示克隆变化实例，并生成训练集。最后第 9 行，使用训练集调用 WEKA 训练机器学习模型。算法的主要时间消耗在提取克隆变化实例的特征上，其时间复杂度为 $O(m)$ ，其中 m 为克隆变化实例的数量。

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

4.7.1 实验设置

4.7.1.1 实验系统

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 2 列和第 3 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。

表 4-2 给出了四个实验系统中克隆变化实例所包含的克隆代码的数量的统计信息。表中最后一列显示了克隆代码数量为 2。因此，系统中虽然有大量的克隆代

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 |
|------------|---------------|-------------|------|
| | 不需要维护 | 需要维护 | |
| 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 |

码，但是变化实例所包含的克隆数量并不是很多。同时，ArgoUML 和 jFreeChart 系统中存在少量的超大克隆组，从而导致了平均值和标准差较大。

表 4-2 克隆变化实例的克隆代码数量信息统计

Table4-2 Statistics of the size of clone change instances

| 实验系统 | 数量 | 平均值 | 标准差 | 中位数 |
|------------|------|-------|--------|-----|
| ArgoUML | 427 | 7.593 | 26.224 | 2 |
| jEdit | 159 | 2.465 | 0.913 | 2 |
| jFreeChart | 1040 | 7.853 | 26.924 | 2 |
| Tuxguitar | 354 | 3.404 | 3.452 | 2 |

表 4-3描述了克隆变化实例发生变化时的克隆寿命统计信息。如前所述，克隆变化实例寿命年龄是自其所在的克隆组在发生变化时，所经历的所有软件版本的数量（从 0 开始计算）。从表中可以看出，克隆变化实例的寿命普遍偏小。考虑到克隆变化实例的本质，即被开发人员修改的克隆代码。克隆代码变化可能会较早的出现，因此需要尽可能早地执行克隆的一致性要求，从而避免引入过多的克隆一致性违背缺陷。

表 4-3 克隆变化实例寿命信息统计

Table4-3 Statistics for clone life of clone change instances

| 实验系统 | 数量 | 平均值 | 标准差 | 中位数 |
|------------|------|-------|-------|-----|
| ArgoUML | 427 | 2.094 | 2.331 | 1 |
| jEdit | 159 | 4.484 | 3.644 | 3 |
| jFreeChart | 1040 | 4.935 | 4.995 | 3 |
| Tuxguitar | 354 | 1.557 | 1.503 | 1 |

4.7.1.2 评估指标

与第3章的实验方式相似，本节实验也划分有效性验证实验和使用模式实验。有效性验证实验中，使用五种不同机器学习方上评估方法的预测能力，试图帮助程序开发人员选择一个最优的机器学习模型。使用模式实验中，分别对需要一致性维护和不需要一致性维护的实例进行实验，详细给出了贝叶斯网络方法的预测结果，可以帮助程序开发人员选择合适的使用模式。同时，为全面评估本文方法，将每一个实验还进一步划分为全属性实验和属性组实验。

在克隆变化的一致性预测实验中，将同时预测两种不同的状态的实例的预测效果。分别计算两种类型变化实例的（满足和不满足一致性维护需求）的精确率 (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（计算方法见第3章第3.7.1.2实验评估小节）。然后将这两组预测结果根据实际实例的数量进行加权平均，计算平均精确率 (Average Precision)、平均召回率 (Average Recall) 和平均 F 值 (Average F-measure) 作为评价指标，如下所示：

- 平均精确率：该指标评估克隆变化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精确率，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精确率。最后，将这两个值进行加权平均，作为整个模型的精确率，计算如下：

$$Ave-Precision = \frac{precision_1 \times number_1 + precision_2 \times number_2}{number_1 + number_2} \quad (4-1)$$

其中， $precision_1$ 和 $precision_2$ 分别是两种实例的精确率， $number_1$ 和 $number_2$ 是系统中两种实例的数量。

- 平均召回率：该指标评估克隆变化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。同精确率，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，计算如下，

$$Ave-Recall = \frac{recall_1 \times number_1 + recall_2 \times number_2}{number_1 + number_2} \quad (4-2)$$

其中， $recall_1$ 和 $recall_2$ 分别是两种实例的召回率， $number_1$ 和 $number_2$ 是系统中两种实例的数量。

- 平均 F 值：该指标可以评估所有克隆变化实例的精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变化实例的 F 值，然后根据实例的数量进行加权平均，计算如下，

$$Ave-F-measure = \frac{F_1 \times number_1 + F_2 \times number_2}{number_1 + number_2} \quad (4-3)$$

其中, F_1 和 F_2 分别是两种实例的 F 值, $number_1$ 和 $number_2$ 是系统中两种实例的数量。

4.7.2 有效性验证实验

在本节实验中, 使用 10-Folds Cross-Validity 方式将数据集分为训练集和测试集, 使用平均精确率 (Average Precision)、平均召回率 (Average Recall) 和平均 F 值 (Average F-measure) 作为评价指标评估不同机器学习方法的预测能力。在此实验中, 五种机器学习方法的参数设置如下: 贝叶斯网络的父节点数设置为 4, Native 方法只有一个父节点, 同样设置阈值为 0.5。SVM 方法使用 *polynomial kernel* 作为核函数。并使用 J48 算法作为决策树, 设置置信度为 0.75。KNN 方法的邻居个数为 1, 使用 *Euclidean Distance* 为距离函数。本节实验分成了两个部分, 分别为全属性实验和属性组实验, 实验结果如表 4-4 和表 4-5 所示。

4.7.2.1 全属性实验结果

表 4-4 给出了使用全属性组在五种不同机器学习方法上的预测结果。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。精确率的范围从 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, 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。因此可以得出结论, 在进行克隆变化的一致性预测是, 预测效果将取决于不同项目的训练集规模大小。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。具体来说, SVM 对 jEdit, jFreeChart 和 Tuxguitar 具有最好的结果, KNN 和 SVM 对于 ArgoUML 是最好的。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。

表 4-4 克隆变化实例的一致性维护需求预测结果

Table4-4 The effectiveness of clone changing consistency for all attribute

| 指标 | 系统 | ArgoUML | jEdit | jFreeChart | Tuxguitar |
|--------------|----------|---------|-------|------------|-----------|
| 平均 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 |
| 平均 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 |
| 平均 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 |

4.7.2.2 属性组实验结果

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 4-5 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接受精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，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。

表 4-5 克隆变化实例的属性组一致性维护需求预测效果

Table4-5 The effectiveness of attribute set for clone changing consistency

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

4.7.3 使用模式实验

与第3章相似，本节也使用贝叶斯网络方法，对两种不同状态的实例分别进行预测。贝叶斯网络使用 K2 算法构建网络结构，使用 SimpleEstimator 来计算网络的概率表。设置不同的阈值进行实验分析。对于需要一致性维护的实例，选取阈值为 0.5、0.6、0.7、0.8 和 0.9。对于不需要一致性维护的实例，选取阈值为 0.01、0.05、0.10、0.15 和 0.20。

4.7.3.1 一致性维护需求实验

在本节中，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，避免克隆一致性违背缺陷。实验采用三个度量对方法进行评估，即警告率、精确率和召回率。其中，警告率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。

(1) 全属性组实验结果

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 4-6 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，但其精确率仍然可以达到 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精确率和召回率，但影响并不是十分的剧烈，对召回率的影响要比精确率的影响更大。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 3 章提出的克隆创建时克隆一致性预测方法。

(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

表 4-6 需要维护的全属性组实验效果

Table4-6 The effectiveness for all attribute sets of clone consistency

| 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 |

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 4-7 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，本章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，发现全属性组实验结果要优于最佳属性组实验结果，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著的影响，但是其也不具有消极的影响。本章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作用。

4.7.3.2 一致性维护自由实验

本节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、精确率和召回率。其中，推荐率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。

(1) 全属性实验结果

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 4-8 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，精确率在 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

表 4-7 需要维护的属性组实验效果

Table4-7 The effectiveness for attribute set of clone consistency

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

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精确率和召回率，但是阈值对于召回率的影响要大于大于精确率的影响。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（精确率差不大），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 4-8 全属性组维护自由实验效果

Table4-8 The effectiveness for all attribute of clone consistency free

| 系统 | 阈值 | 推荐率 (%) | 精确率 (%) | 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 |

（2）属性组实验结果

同样，对不需要一致性维护的克隆变化实例，本章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 4-9 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著的影响,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。

4.7.4 讨论

本节从不同的角度评估了本章所提出的克隆代码变化一致性维护需求预测,同时对需要不需要一致性维护需求的克隆变化实例进行了预测,并从不同的角度评估了不同的机器学习方法的预测能力。

在机器学习方法对比中,在克隆代码变化时,五种机器学习方法都可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,预测结果具有合理的精确率、召回率和 F 值。更重要的是,不同机器学习模型的预测能力仅具有较小的差异,但 SVM 方法拥有相对最佳的预测效果。因此,建议优先推荐使用 SVM 方法,预测克隆代码变化的一致性维护需求。与此同时,所提取的属性组作为一个整体在不同机器学习方法中,同样起到了积极的作用。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。

在使用贝叶斯网络的预测中,全属性实验结果表明: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,属性组实验表明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,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,从而将其运用到其它未验证的系统中,使之达到最佳的预测效果。

表 4-9 属性组维护自由实验效果

Table4-9 The effectiveness for attribute set of clone consistency free

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

4.8 本章小结

软件中存在的克隆代码，在其演化过程中的一致性变化，可能会引发克隆一致性违背缺陷。为了帮助软件开发人员避免克隆一致性违背缺陷，本章提出了一个克隆代码变化一致性维护需求预测方法。在克隆代码发生变化时，预测该变化的一致性维护需求，从而帮助软件开发人员避免克隆代码的一致性违背缺陷。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，并使用其构建和训练不同的机器学习模型预测其一致性维护需求。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，分别为代码属性、上下文属性和演化属性。在四个开源软件系统上进行实验，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合理的精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。

第 5 章 跨项目克隆代码一致性维护需求预测研究

5.1 引言

由于日益增长的软件开发需求，复用既有代码作为一种常见的开发手段会引入大量克隆代码。随软件系统演化的过程中，克隆代码的一致性变化不仅会引发额外的维护代价，甚至可能会导致克隆一致性违背缺陷。因此，本文第 3 章和第 4 章在克隆代码创建和变化时，预测克隆代码的一致性维护需求。但是，第 3 章和第 4 章的方法无法应用于软件开发初期阶段的系统中。因为其未经过充分的演化，缺少历史演化数据训练机器学习模型。为解决此问题，本章在第 3 章和第 4 章研究的基础上，使用跨项目的数据预测克隆代码一致性维护需求预测，可以在软件开发初期，帮助避免克隆额外维护代价和一致性违背缺陷。

针对软件开发初期项目缺乏训练数据的问题，本章研究跨项目的克隆代码一致性维护需求预测。首先，统一了克隆代码创建、变化的一致性维护需求定义。然后，将不同的软件系统划分为测试集和训练集，使用机器学习方法在训练系统上训练机器学习模型，在测试系统上验证跨项目模型的预测能力。在四个开源软件系统上进行了跨项目实证研究，实验结果表明跨项目的克隆一致性维护需求预测，可以应用于软件开发初期时的预测中。同时，跨项目预测的效果依赖于具体的系统以及训练数据的规模。最后结合软件开发过程，基于 eclipse 设计并实现了克隆代码一致性维护需求预测插件，可以帮助程序人员边开发、边维护克隆代码，从而帮助提高软件质量和可维护性。

5.2 跨项目克隆一致性维护问题

5.2.1 问题的提出

为了解决克隆代码的一致性维护问题，本文的第 3 章和第 4 章分别在克隆代码创建时和变化时，预测克隆代码的一致性维护需求。但是，上述方法仅仅适用于软件经过若干版本演化，系统具有充足的克隆创建和变化实例数据的情况。

然而，在软件开发初期阶段，软件系统刚开始进行演化，不具有充足的演化历史数据（第一个版本甚至没有相关数据），从而无法预测克隆代码的一致性维护需求。因此，本章考虑使用跨项目的克隆代码演化历史数据对克隆代码的一致性维

护需求进行预测。目前，尚未有人对跨项目的克隆代码一致性维护问题进行研究。本章利用跨项目的数据，为软件开发初期进行克隆代码的一致性维护需求预测，提供了一种可能性。

最后，为将第 3、4 和 5 章的研究方法实际应用到软件开发过程中，实现边开发、边维护克隆代码，还设计并实现了一个克隆代码的一致性维护需求预测插件。该插件可以集成到软件开发环境 eclipse 中，实时的监测克隆代码的产生和变化，并对其进行一致性维护需求预测。根据预测结果，程序开发人员可对克隆代码采取相应的维护措施。

5.2.2 本文的解决思路

为解决本章所提出的跨项目的克隆代码一致性需求维护预测问题，本节给出本章方法的框架。跨项目克隆代码一致性需求预测框架如图 5-1 所示。跨项目的克隆一致性维护需求预测方法可以划分为三个阶段，分别为训练数据收集阶段、测试数据收集阶段和跨项目预测阶段。收集阶段旨在收集训练系统和测试系统中全部的克隆实例（包括创建实例和变化实例），并将训练系统的数据集用于机器学习模型的构建，测试系统的数据用于测试跨项目预测的效果。在预测阶段，使用训练系统数据所构建的机器学习模型预测测试系统的克隆代码一致性维护需求。其中，在表示所收集到的克隆实例时，也会提取相应的属性值描述所提取的克隆实例，所提取的属性值与第 3 章和第 4 章中的相同。

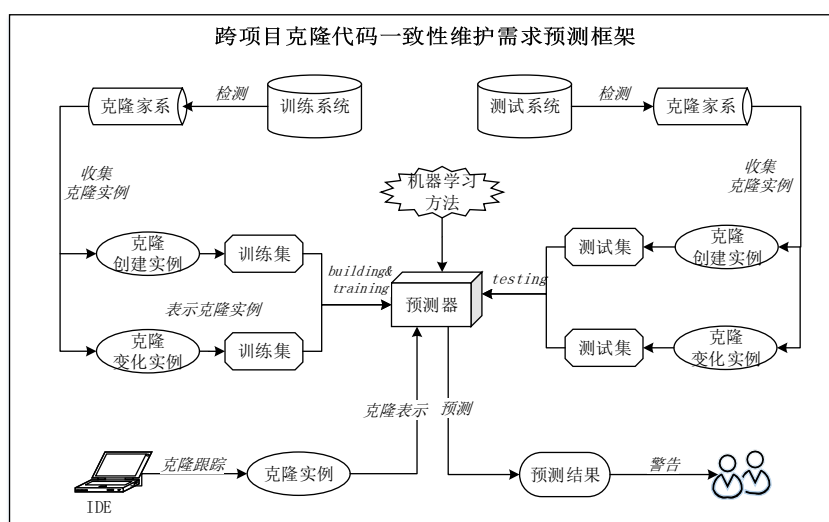


图 5-1 跨项目克隆代码一致性维护需求预测框架

Fig.5-1 The framework for clone cross-project consistency prediction

为了收集不同系统中的克隆代码创建和变化实例，同样通过构建系统的克隆家系来完成收集工作。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，用于收集克隆实例。然后，通过提取属性值表示克隆创建和变化实例，分别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。最后，使用收集到的训练系统的克隆实例训练机器学习模型，并使用其预测测试系统的克隆代码一致性维护需求。

与第 3 章和第 4 章相似，每一个克隆实例也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的一致性。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，程序开发人员不需要采取相应的操作，可以放心自由的接受新创建的克隆实例和修改克隆代码。

本章在第 3 章和第 4 章的研究基础上，在进行跨项目的克隆一致性维护需求预测时，将同时预测克隆代码创建时和变化时的跨项目一致性维护需求，将重点分析和讨论如下问题：

- (1) 在软件开发初期阶段，在项目自身训练数据不充分的情况下，能否使用其它系统的数据训练模型，并对该系统进行克隆一致性需求预测？
- (2) 在进行跨项目的克隆一致性维护需求预测时，训练数据的规模将如何影响跨项目的预测效果？
- (3) 在进行跨项目的克隆一致性维护需求预测时，应该预测哪一类别的克隆代码实例，是否对两种不同类别的实例都具有相一致的预测效果？

5.3 克隆一致性维护需求的定义

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 3 章和第 4 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。为了进一步研究跨项目的一致性预测问题，本节统一了第 3 章和第 4 章的定义，以应用于跨项目的克隆代码一致性维护需求预测中。因此，克隆代码的一致性变化如下：
定义 5.1 (克隆一致性变化) 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)，

$$\text{textSim}(CF_i, CF'_i) < 1 \quad \forall i \in \{1, 2\} \quad (1)$$

$$|\text{textSim}(CF_1, CF'_1) - \text{textSim}(CF_2, CF'_2)| < \tau \quad (2)$$

如果克隆代码变化仅满足条件 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-2 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 3 章和第 4 章计算方式相同。该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，由阈值 τ 指定。

其中，Type-1 一致性变化是克隆创建一致性变化，Type-2 一致性变化是克隆变化的一致性变化。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 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违背缺陷。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违背缺陷，因此，在克隆代码变化时使用 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。现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 (克隆一致性变化模式) 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“Type-1 或者 Type-2 一致性变化” (Type-1 or Type-2 Consistent Change)，则称克隆组 CG' 具有 Type-1 或者 Type-2 一致性变化模式 (Type-1 or Type-2 Consistent Change Pattern)。

回顾本章地研究内容是要在两种不同的时刻，跨项目的预测克隆代码的一致性维护需求，结合第 3 章和第 4 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统一为克隆实例，如下所示：

定义 5.3 (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。在一个克隆家系 CG 中，克隆家系的根节点为克隆创建实例，并且发生变化的克隆组为克

隆变化实例。

克隆实例在演化过程中可能会发生一致性变化模式,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违背缺陷。对于克隆创建实例, Type-1 一致性变化可能会导致额外的克隆维护代价。对于克隆变化实例, Type-2 一致性变化可能会导致克隆一致性违背缺陷。因此,需要在不同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违背缺陷。本章结合第3章和第4章的克隆代码一致性维护需求的定义,将克隆创建和变化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) 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“Type-1 或者 Type-2 一致性变化模式” (Type-1 or Type-2 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, 或者 Free)。

最终, 克隆一致性维护需求预测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。

5.4 跨项目数据集的获取

在跨项目的预测中, 使用其它系统的数据训练机器学习模型, 并预测另外系统的一致性维护需求。因此, 需要构建跨项目的数据集, 将从训练系统中提取克隆实例作为训练集, 从测试系统中提取克隆实例作为测试集。

对于某一个软件系统来讲, 通过收集和表示该系统中的克隆实例, 可以生成其数据集。因此, 生成实验所需的数据集, 可以划分为两个部分: 收集系统中的克隆实例和表示所收集到的克隆实例

5.4.1 跨项目数据来源

跨项目的数据由软件系统中的克隆实例组成, 因此收集系统中的克隆实例, 即可以完成收集跨项目的数据。通过构建系统的克隆家系, 通过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。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。通过构建和遍历克隆家系, 可以收集克隆创建和变化实例。

首先，下载系统所有版本的源代码，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。然后，通过对比相邻版本的克隆代码，识别克隆家系的克隆演化模式，尤其是克隆一致性变化模式（参考定义 2.5 和 5.2）。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。在收集克隆创建实例后，还需确认该实例中的被复制和被粘贴代码（参见本文第 3 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。

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，标识克隆实例的一致性维护需求。如果克隆实例在其演化过程中发生了一致性变化模式（定义 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 属性特征与数据集生成

本文收集到的数据是系统中的克隆实例，因此同样提取相应的属性值表示克隆代码实例。将提取代码、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，表示了创建时克隆代码的特征；克隆变化实例中的属性，表示了发生变化的克隆组的特征。

对于克隆创建实例，即被复制和粘贴克隆代码，使用不同的属性进行描述。使用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。使用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 3 章相关属性值部分（第 3.5 节克隆创建实例表示）。

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 4 章相关属性值部分（第 4.5 节克隆变化实例表示）。

Input: All source code and code clones from N versions

Output: The data set of software

```

1 for  $i=1$  to  $N$  do
2   Generating CRD to represent each code cloneCFs;
3   Mapping all the  $CFs$  and  $CGs$  between the adjacent
   versions  $i$  and  $i+1$ ;
4   Identifying all clone evolution pattern in thus two versions
   according to Definition 2.5 and Definition 5.2 ;
5 end
6 Generating all clone genealogies  $CGEs$  with number  $N_{cge}$ ;
7 for  $i=1$  to  $N_{cge}$  do
8   Collecting the clone instance that both including
   clone-creating and changing instances from  $CGE_i$ 
   according to Definition 5.3 ;
9   Labeling all consistency-requirement according to
   Definition 5.4 ;
10 end
11 Initializing all training data set  $Creating\_Set$  and
    $Changing\_Set$  for clone creating and changing instances;
12 for each clone instance do
13   Generating all attributes for clone creating or changing
   instance;
14   Appending all the attributes to the  $Creating\_Set$  or
    $Changing\_Set$ ;
15 end
16 return The data set of software;

```

算法 5-1 数据集生成算法

Algo. 5-1 The algorithm for generating dataset

数据集生成算法如 5-1 所示。算法中第 1-5 行是为克隆代码生成 CRD，并使用 CRD 映射相邻版本的克隆代码，并根据克隆演化模式定义，识别克隆演化模式。

第 6 行是生成系统的克隆家系。第 7-10 行，分别是收集克隆实例，并标记其一致性维护需求。第 11 行是初始化数据集。第 12-15 行是提取相应的属性值表示克隆实例，并且生成数据集。在表示克隆实例时，算法将消耗大量的时间，其时间复杂度为 $O(m * n)$ ，其中 m 为克隆变化实例的数量。因此，该算法的时间复杂度为 $O(m * n)$ 。

5.5 跨项目模型的构建与预测

使用收集到的训练系统的克隆实例训练机器学习模型，并预测测试系统的克隆代码的一致性维护需求。从而验证跨项目的克隆一致性需求的预测能力。值得注意的是，将对两种不同的时刻的克隆代码的一致性维护需求分别进行预测（即克隆创建时和克隆变化时），因此会训练两种不同的模型以适用于两个时刻。

对于所测试的软件系统，首先，通过收集训练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的预测器，预测克隆代码的一致性维护需求。

跨项目克隆一致性维护需求预测算法如 5-2 所示。算法的第 1 行为划分训练集和测试集。第 2-3 行是生成跨项目的数据集。第 4 行是使用训练集训练机器学习模型。第 5 行是使用训练的模型在测试集上验证跨项目预测的有效性。

Input: All the projects

Output: The predictive models

- 1 Dividing the projects into training projects and testing project;
- 2 Generating the *Training_Set* for training projects;
- 3 Generating the *Testing_Set* for testing project;
- 4 Calling WEKA to train the models applying training set
Training_Set;
- 5 Employed this trained model to predict on the *Testing_Set*;
- 6 **return** *The predictive models*;

算法 5-2 跨项目克隆一致性维护需求预测算法

Algo. 5-2 The algorithm for cross-project predicting clone
consistency-requirement

与第 3 章和第 4 章相似，根据定义 5.4，克隆实例会有两种不同的状态：需要一致性维护和不需一致性维护。如下所示：

- 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。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的，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，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，也不会导致一致性违背缺陷。

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和跟踪克隆代码，识别新产生的克隆实例（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 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进行跨项目的一致性维护预测。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。

5.6.1 实验设置

本节在四个开源系统上进行实验评估，如表 5-1 所示。从该表可以看出，系统中克隆创建实例的数据集的规模要大于克隆变化实例的数据集。克隆创建实例数据集规模为 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。

在进行跨项目的预测实验时，使用三个系统的数据作为训练集，使用第四个系统作为测试集。跨项目预测的训练集如表 5-2 所示。表中使用（To Project）表示被测试的系统为“Project”，并使用除“Project”系统外其它的系统作为训练集。从表中发现克隆创建的训练集规模要远远大于克隆变化的训练集规模，其中克隆创

表 5-1 跨项目测试集信息统计

Table5-1 The statistics for cross-project testing data

| 类型 | 测试系统 | 测试集规模 | | |
|--------|------------|--------------|--------------|------|
| | | 不需要维护 | 需要维护 | 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 |

建的训练集规模为 5402-8135，克隆变化的训练集规模为 940-1821。

表 5-2 跨项目训练集信息统计

Table5-2 The statistics for cross-project training data

| 类型 | 训练系统 | 训练集规模 | | |
|-------|-----------------|--------------|--------------|------|
| | | 不需要维护 | 需要维护 | 总数 |
| 克隆创建时 | (To ArgoUML) | 3589(66.12%) | 1839(33.88%) | 5428 |
| | (To jEdit) | 5603(68.88%) | 2532(31.12%) | 8135 |
| | (To jFreeChart) | 4150(76.82%) | 1252(23.18%) | 5402 |
| | (To Tuxguitar) | 5147(70.13%) | 2192(29.87%) | 7339 |
| 克隆变化时 | (To ArgoUML) | 621(39.99%) | 932(60.01%) | 1553 |
| | (To jEdit) | 831(45.63%) | 990(54.37%) | 1821 |
| | (To jFreeChart) | 457(48.62%) | 483(51.38%) | 940 |
| | (To Tuxguitar) | 818(50.31%) | 808(49.69%) | 1626 |

为解决跨项目克隆代码一致性预测问题，本章将研究问题划分为三个子研究问题。因此本节也从三个不同的角度进行实验，并回答提出的三个子研究问题：

- 跨项目的有效性验证实验：将回答第一个子研究问题，使用不同的机器学习模型进行跨项目克隆一致性预测，以验证是否能使用跨项目的数据进行跨项目一致性维护需求预测。

- 跨项目的数据集规模实验：将回答第二个子研究问题，使用不同的数据集训练机器学习模型，并在同一系统上测试跨项目预测模型的预测能力，从而探讨训练集规模对跨项目模型预测能力的影响。

• 跨项目的使用模式实验：将回答第三个子研究问题，将探讨在进行跨项目预测时，不同类别克隆实例的一致性预测的效果。实验将使用贝叶斯网络预测不同类别的克隆实例。

5.6.2 跨项目有效性验证实验

本节将回答本章提出的第一个研究问题：在软件开发初期阶段，在项目自身训练数据不充分的情况下，能否使用其它系统的数据训练模型，并对该系统进行克隆一致性维护需求预测？

在五种机器学习方法上进行了跨项目的预测，并使用同本文第 3 章和第 4 章相同的指标评估预测效果，即采用平均精确率 (Average Precision)、平均召回率 (Average Recall) 和平均 F 值 (Average F-measure) 进行评价（见本文第 3.7.1.2 节和第 4.7.1.2 节）。

5.6.2.1 克隆创建时的实验结果

对克隆代码创建实例，使用不同的机器学习方法进行跨项目预测，实验结果如 5-3 所示。在表中，对于 SVM 的实验结果使用 * 进行标记表。原因在于：在使用 SVM 方法进行跨项目预测时，不具备相对应的预测能力，其会将所有的测试数据划分到同一个类别中（规模比较大的类中）。

从表 5-3 中可以看出，除了 SVM 方法外，其它的四种机器学习方法，对不同的项目均具有不错的预测效果，并且彼此之间也不存在显著的差异。ArgoUML 系统的预测结果较好，其平均 F 值介于 65.3%-66.7% 之间。对 jEdit 系统的预测效果最好，为 76.2%-82.6%，jFreeChart 最差，也达到了 48.9%-60.9%。对于 Tuxguitar 来说，预测效果也较好，F 值为 60.3%-68.3%。

尽管如此，不同的机器学习方法在不同的跨项目的预测中，其预测能力不是完全一致。对系统 ArgoUML 和系统 Tuxguitar 来说，决策树具有最好的预测效果，其 F 值分别为 66.7% 和 68.3%。贝叶斯网络对 jEdit 系统具有最好的预测效果，F 值为 82.6%。而 KNN 方法对于系统 jFreeChart 具有最好的预测效果，F 值为 60.9%。

因此，针对不同的系统，不推荐程序开发人员选用 SVM 的方法；但可以使用其它的机器学习方法进行跨项目的一致性维护需求预测；针对不同的软件系统，程序开发人员可以根据预测结果选择最合适的机器学习方法。

5.6.2.2 克隆变化实例的实验结果

对克隆代码变化实例，使用不同的机器学习方法进行跨项目预测，实验结果如 5-4 所示。在表中，对于 SVM 的实验结果使用 * 进行标记。原因相同：在使用

表 5-3 克隆创建时跨项目预测效果

Table5-3 The effectiveness for cross-project prediction at creating time

| 指标 | 方法 | To ArgoUML | To jEdit | To jFreechart | To Tuxguitar |
|--------------|----------|------------|----------|---------------|--------------|
| 平均 Precision | BayesNet | 0.638 | 0.812 | 0.575 | 0.648 |
| | Natvie | 0.631 | 0.783 | 0.486 | 0.615 |
| | SVM | 0.594* | 0.783* | 0.358* | 0.506* |
| | KNN | 0.629 | 0.786 | 0.648 | 0.581 |
| | Tree | 0.629 | 0.796 | 0.566 | 0.704 |
| 平均 Recall | BayesNet | 0.675 | 0.844 | 0.602 | 0.698 |
| | Natvie | 0.693 | 0.747 | 0.54 | 0.647 |
| | SVM | 0.771* | 0.885* | 0.598* | 0.711* |
| | KNN | 0.687 | 0.741 | 0.649 | 0.663 |
| | Tree | 0.74 | 0.844 | 0.599 | 0.731 |
| 平均 F-Measure | BayesNet | 0.654 | 0.826 | 0.508 | 0.65 |
| | Natvie | 0.656 | 0.764 | 0.489 | 0.627 |
| | SVM | 0.671* | 0.831* | 0.448* | 0.591* |
| | KNN | 0.653 | 0.762 | 0.609 | 0.603 |
| | Tree | 0.667 | 0.818 | 0.503 | 0.683 |

SVM 方法进行跨项目预测时，不具备相对应的预测能力，其会将所有的测试数据划分到同一个类别中（规模比较大的类中）。

从表中可以看出，其它的四种机器学习方法，对不同的项目的预测效果一般，并且彼此之间也不存在显著的差异。分析其预测能力一般的原因，可能是系统中存在的克隆变化实例较少，而项目之间的克隆代码变化也千差万别，所构建的跨项目预测模型并不具有较强的泛化能力。

尽管如此，在克隆变化时进行跨项目预测时，对于不同的系统，存在某一个机器学习方法可以是预测结果的 F 值达到一个相对最好的预测效果。如（To ArgoUML）中 KNN 方法的 F 值为 54.6%，（To jEdit）中 Decision Tree 的 F 值为 56.7%，（To jFreeChart）的 Native Bayesian 方法为 53.3%。因此，对不同的系统可以选择最合适的机器学习方法进行预测。

不同的机器学习方法在不同的跨项目的预测中，其预测能力不是完全一致。对系统 ArgoUML 和系统 Tuxguitar 来说，KNN 具有最好的预测效果，其 F 值分别为 54.6% 和 48.7%。决策树对 jEdit 系统具有最好的预测效果，F 值为 56.7%。而朴素贝叶斯方法对于系统 jFreeChart 具有最好的预测效果，F 值为 53.3%。

结合上一节的贝叶斯网络方法的实验结果，在跨项目的预测中，不推荐开发人员选用 SVM 的方法；但依然可以使用其它的机器学习方法进行跨项目一致性预测，但需要根据项目自身特征，去选择所预测的类别（样本不平衡），即预测数量较多的克隆变化实例类别；在对不同的软件系统进行预测时，程序开发人员可以根据预测结果选择最合适的机器学习方法。

表 5-4 克隆变化时跨项目预测效果

Table 5-4 The effectiveness for cross-project prediction at changing time

| 度量 | 方法 | To ArgoUML | To jEdit | To jFreechart | To Tuxguitar |
|--------------|----------|--------------|--------------|---------------|--------------|
| 平均 Precision | BayesNet | 0.591 | 0.549 | 0.537 | 0.602 |
| | Natvie | 0.629 | 0.547 | 0.566 | 0.556 |
| | SVM | 0.106* | 0.26* | 0.611* | 0.524* |
| | KNN | 0.624 | 0.542 | 0.533 | 0.623 |
| | Tree | 0.629 | 0.648 | 0.546 | 0.638 |
| 平均 Recall | BayesNet | 0.471 | 0.547 | 0.512 | 0.418 |
| | Natvie | 0.464 | 0.547 | 0.538 | 0.353 |
| | SVM | 0.326* | 0.509* | 0.596* | 0.322* |
| | KNN | 0.534 | 0.541 | 0.509 | 0.452 |
| | Tree | 0.37 | 0.604 | 0.567 | 0.387 |
| 平均 F-Measure | BayesNet | 0.474 | 0.546 | 0.507 | 0.441 |
| | Natvie | 0.45 | 0.545 | 0.533 | 0.361 |
| | SVM | 0.16* | 0.344* | 0.522* | 0.32* |
| | KNN | 0.546 | 0.54 | 0.505 | 0.478 |
| | Tree | 0.273 | 0.567 | 0.489 | 0.383 |

5.6.2.3 讨论

综上所述，可以回答第一个研究子问题：

首先，克隆创建时的跨项目预测取得了相对较好的预测效果；在选择合适的机器学习模式前提下，克隆变化时的预测效果也可达到相对不错的预测效果。因此，在软件开发初期，可以使用跨项目的数据进行跨项目的克隆一致性维护需求预测。

然后，在进行跨项目预测时，不建议选择 SVM 方法进行预测。不同机器学习方法的预测能力不具有显著性差异，可以根据不同的系统选取合适的机器学习方法。

5.6.3 跨项目训练集规模实验

在进行克隆代码一致性预测时，训练集的规模会影响到预测的效果。为了探索数据集规模对预测效果的影响，本节实验使用不同的训练集训练模型，并预测克隆代码的一致性。将回答本章提出的第二个研究问题：在进行跨项目的克隆一致性维护需求预测时，训练数据的规模将如何影响跨项目的预测效果？

在五种机器学习方法上进行了跨项目的预测，并使用同本文第 3 章和第 4 章相同的指标评估预测效果，即采用平均精确率 (Average Precision)、平均召回率 (Average Recall) 和平均 F 值 (Average F-measure) 进行评价（见本文第 3.7.1.2 节和第 4.7.1.2 节）。

首先，进行“一对一”跨项目预测。“一对一”跨项目预测指的是仅使用一个系统的数据作为训练集，另一个系统的数据作为测试集进行跨项目预测。并对目标系统的预测结果进行平均，得出“一对一”预测结果（使用“Average”表示）。然后，进行“多对一”跨项目预测。“多对一”跨项目预测指的是，使用三个系统的数据集合称为一个大的训练集，并在另一个系统上进行跨项目预测，得出多对一预测结果（使用“All”表示）。实验结果如表 5-5 和表 5-6 所示。

5.6.3.1 克隆创建实例的实验结果

在克隆创建时的一对一预测实验中，系统 jFreeChart 和系统 ArgoUML 的规模最大，克隆创建实例分别为 3366 和 3340 个；系统 Tuxguitar 规模居中，为 1429 个；而 jEdit 具有最小的规模，实例个数仅为 159 个。在克隆创建时的多对一预测实验中，训练集规模最大的系统为 jEdit (8135)，其次为系统 Tuxguitar (7339)，系统 ArgoUML 和 jFreeChart 系统的训练集规模最小（分别为 5428 和 5402）。

对克隆创建实例进行跨项目预测，实验结果如表 5-5 所示。其中“All”列表示多对一实验结果，“Average”列表示“一对一”实验结果。

通过对比不同系统的跨项目预测结果，可以发现 (To jEdit) 的实验效果最好，其 F 值在 80% 左右；(To ArgoUML) 和 (To Tuxguitar) 的跨项目预测效果次之，其 F 值在 60-70% 左右；(To jFreeChart) 的预测效果最差，F 值在 50% 左右。根据其训练集的规模大小，可以发现跨项目预测效果和训练集规模存在正相关关系，即训练集的大小影响跨项目的预测能力。因此，在进行跨项目克隆创建预测时，建议扩大其训练集的大小，以达到较好的预测效果。

通过对比同一个系统的“Average”和“All”列，在 4 个系统中全部的 F 值中，“All”列具有较好的预测效果（表中大多数“All”列的数值要大于“Average”列）。但是，通过比较“Average”和“All”列中数据，发现两列的数据相差不大。

因此, 在对于某一个具体的测试系统, 在跨项目预测中, 对于同一个测试系统而言, 由于项目之间的差异, 训练集规模的大小并不能起到决定性作用。因此, 可以考虑在保证一定规模训练数据的基础上, 对训练数据进行进一步的过滤, 缩小项目之间的差异, 从而提升跨项目的预测效果。但是, 如何过滤和优选跨项目预测的训练数据, 从而缩小项目之间的差异, 仍需进一步的研究确认。

表 5-5 创建时跨项目预测数据集规模实验

Table5-5 The effectiveness of cross-project for training size at creating time

| 指标 | 方法 | To ArgoUML | | To jEdit | | To jFreeChart | | To Tuxguitar | |
|--------------|----------|------------|-------|----------|-------|---------------|-------|--------------|-------|
| | | Average | All | Average | All | Average | All | Average | All |
| 平均 Precision | BayesNet | 0.633 | 0.638 | 0.810 | 0.812 | 0.556 | 0.575 | 0.633 | 0.648 |
| | Natvie | 0.629 | 0.631 | 0.792 | 0.783 | 0.502 | 0.486 | 0.606 | 0.615 |
| | SVM | 0.657 | 0.594 | 0.783 | 0.783 | 0.358 | 0.358 | 0.506 | 0.506 |
| | KNN | 0.629 | 0.629 | 0.815 | 0.786 | 0.617 | 0.648 | 0.587 | 0.581 |
| | Tree | 0.686 | 0.629 | 0.782 | 0.796 | 0.530 | 0.566 | 0.594 | 0.704 |
| 平均 Recall | BayesNet | 0.670 | 0.675 | 0.752 | 0.844 | 0.606 | 0.602 | 0.674 | 0.698 |
| | Natvie | 0.669 | 0.693 | 0.754 | 0.747 | 0.564 | 0.54 | 0.643 | 0.647 |
| | SVM | 0.809 | 0.771 | 0.885 | 0.885 | 0.598 | 0.598 | 0.711 | 0.711 |
| | KNN | 0.681 | 0.687 | 0.762 | 0.741 | 0.635 | 0.649 | 0.655 | 0.663 |
| | Tree | 0.764 | 0.74 | 0.803 | 0.844 | 0.595 | 0.599 | 0.688 | 0.731 |
| 平均 F-Measure | BayesNet | 0.645 | 0.654 | 0.768 | 0.826 | 0.516 | 0.508 | 0.631 | 0.65 |
| | Natvie | 0.645 | 0.656 | 0.774 | 0.764 | 0.488 | 0.489 | 0.617 | 0.627 |
| | SVM | 0.724 | 0.671 | 0.831 | 0.831 | 0.448 | 0.448 | 0.591 | 0.591 |
| | KNN | 0.651 | 0.653 | 0.781 | 0.762 | 0.579 | 0.609 | 0.601 | 0.603 |
| | Tree | 0.703 | 0.667 | 0.789 | 0.818 | 0.484 | 0.503 | 0.612 | 0.683 |

5.6.3.2 克隆变化实例的实验结果

对克隆变化实例进行预测, 实验结果如表实验结果表 5-6 所示。

根据数据集规模的统计分析(表 5-1 和表5-2), 克隆创建实例的数量要远多于克隆变化实例的数量, 即创建实例具有上千的规模(633-3366 个), 变化实例仅有数百的规模(159-1040)。结合对两种实例的跨项目预测结果, 发现前者的跨项目预测效果要优于后者。因此, 尽管在不同的时刻进行预测, 训练数据的规模同样会影响到跨项目的预测效果。建议在可能的情况下, 增加训练集的数据。

在克隆变化时的一对一预测实验中, 系统 jFreeChart 规模最大, 克隆变化实例为 1040 个; 系统 ArgoUML 和系统 Tuxguitar 的规模居中, 分别为 427 和 3549 个;

而 jEdit 具有最小的规模, 实例个数仅为 159 个。在克隆变化时的多对一预测实验中, 训练集规模最大的系统为 jEdit (1821), 其次为系统 ArgoUML (1553) 和系统 Tuxguitar (1626), 系统 jFreeChart 的训练集规模最小, 为 940。

通过对比不同系统的跨项目预测结果, 可以发现 (To jEdit) 的实验效果最好, 其 F 值除 SVM 外介于 54%-56.7% 之间; (To jFreeChart) 的预测效果次之, F 值在 48.9%-53.3% 之间; 而 (To ArgoUML) 和 (To Tuxguitar) 预测效果最差, 大部分的 F 值低于 50%。根据其训练集的规模大小, 可以发现跨项目预测效果和训练集规模也存在一定的正相关关系。但是, 其相关性不如克隆创建实例。究其原因可能是因为: 克隆变化实的训练集规模仅仅在几百之间, 和创建实例的数量相差较大。克隆变化时跨项目预测的数据集规模较小, 导致模型无法较好的训练。

通过对比每个项目的 “Average” 和 “All” 列, 在 4 个系统中全部的 F 值中, “All” 列具有较好的预测效果。具体地, 对系统 (To jEdit) 和 (To jFreeChart), “All” 列的预测效果均优于 “Average” 列。但是, 通过比较 “Average” 和 “All” 列中数据, 发现两列的数据相差不大。因此, 在对于某一个具体的测试系统, 在跨项目预测中, 由于项目之间的差异, 训练集的大小并不能起到决定性作用。因此, 可以考虑在保证一定规模训练数据的基础上, 对训练数据进行进一步的过滤, 缩小项目之间的差异, 从而提升跨项目的预测效果。但是, 如何过滤和优选跨项目预测的训练数据, 从而缩小项目之间的差异, 仍需进一步的研究确认。

5.6.3.3 讨论

综上所述, 可以回答第二个研究子问题:

首先, 在跨项目预测中, 训练集的大小会影响到预测的效果, 较大的训练集可以达到较强的预测能力。但是, 如何确定训练集大小仍需进一步的研究确定。

其次, 但针对某特定的系统来讲, 训练集的大小并不能起到决定性的作用, 原因可能在于项目自身的差距较大。因此, 如何对训练集项目的数据进行优选和过滤, 可能会提高跨项目的预测能力, 但是如何进行优选跨项目预测的训练数据, 从而降低训练系统和测试系统之间的差异, 仍需进一步的研究确定。

5.6.4 跨项目使用模式实验

在贝叶斯网络的实验中, 本节将回答本章提出的第三个研究问题: 在进行跨项目的克隆一致性维护需求预测时, 应该预测哪一类别的克隆代码实例, 是否对两种实例都具有一致的预测效果?

相似地, 使用三个项目的克隆实例训练贝叶斯网络, 并在第四个实验系统上

表 5-6 克隆变化的跨项目预测数据集规模实验

Table5-6 The effectiveness of cross-project for training size at changing time

| 指标 | 方法 | To ArgoUML | | To jEdit | | To jFreeChart | | To Tuxguitar | |
|--------------|----------|------------|-------|----------|-------|---------------|-------|--------------|-------|
| | | Average | All | Average | All | Average | All | Average | All |
| 平均 Precision | BayesNet | 0.595 | 0.591 | 0.480 | 0.549 | 0.552 | 0.537 | 0.575 | 0.602 |
| | Natvie | 0.596 | 0.629 | 0.541 | 0.547 | 0.548 | 0.566 | 0.548 | 0.556 |
| | SVM | 0.106 | 0.106 | 0.254 | 0.26 | 0.276 | 0.611 | 0.390 | 0.524 |
| | KNN | 0.605 | 0.624 | 0.517 | 0.542 | 0.515 | 0.533 | 0.617 | 0.623 |
| | Tree | 0.616 | 0.629 | 0.490 | 0.648 | 0.485 | 0.546 | 0.616 | 0.638 |
| 平均 Recall | BayesNet | 0.513 | 0.471 | 0.482 | 0.547 | 0.522 | 0.512 | 0.389 | 0.418 |
| | Natvie | 0.496 | 0.464 | 0.541 | 0.547 | 0.531 | 0.538 | 0.398 | 0.353 |
| | SVM | 0.343 | 0.326 | 0.491 | 0.509 | 0.503 | 0.596 | 0.581 | 0.322 |
| | KNN | 0.533 | 0.534 | 0.512 | 0.541 | 0.498 | 0.509 | 0.449 | 0.452 |
| | Tree | 0.387 | 0.37 | 0.501 | 0.604 | 0.516 | 0.567 | 0.550 | 0.387 |
| 平均 F-Measure | BayesNet | 0.491 | 0.474 | 0.456 | 0.546 | 0.501 | 0.507 | 0.383 | 0.441 |
| | Natvie | 0.481 | 0.45 | 0.533 | 0.545 | 0.526 | 0.533 | 0.407 | 0.361 |
| | SVM | 0.160 | 0.16 | 0.337 | 0.344 | 0.360 | 0.522 | 0.457 | 0.32 |
| | KNN | 0.453 | 0.546 | 0.446 | 0.54 | 0.459 | 0.505 | 0.474 | 0.478 |
| | Tree | 0.296 | 0.273 | 0.441 | 0.567 | 0.407 | 0.489 | 0.500 | 0.383 |

进行一致性预测。采用和第 3 章和第 4 章相同的评价指标，评估跨项目预测的能力。对需要一致性维护的克隆实例，使用警告率 (Warning Rate)、精确率 (Precision Rate) 和召回率 (Recall Rate) 进行评估；对不需要一致性维护的克隆实例，使用推荐率 (Recommendation Rate)、精确率 (Precision Rate) 和召回率 (Recall Rate) 进行评估，详见本文第 3.7.1.2 节和第 4.7.1.2 节。实验也在两个时刻进行了预测，即克隆代码创建时和克隆代码变化时。

5.6.4.1 克隆创建实例实验结果

本节在克隆代码创建时，四个系统上的实验结果如表 5-7 和 5-8 所示。

(1) 不需要一致性维护

对于克隆创建实例，由统计结果可以看出（表 5-1），系统中大部分的实例都是不需要进行一致性维护的克隆实例，其比例为 59.80-88.47%。对其进行跨项目一致性预测，其预测结果如表 5-7 所示。

从表中可以看出，四个系统的精确率和召回率依然达到了较高的水平，其中精确率在 60.01%-91.20% 之间，召回率 56.06%-91.43% 之间。同时，通过对比发现 jEdit 的预测效果最好（精确率较高），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。

将实验结果与第 3 章的全属性实验进行对比（表 3-5），可以发现四个系统的预测效果都有了不同程度的下降，但是目跨项目的预测效果依然是可以接受的。

因此，预测模型的预测能力会依赖于自身系统数据，建议优先选用系统自身的数据进行一致性为需求预测；在软件开发初期，自身系统训练数据较少，而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跨项目预测。

(2) 需要一致性维护

对于克隆创建实例，由统计结果可以看出（表 5-1），系统中仅有少量的实例满足一致性维护需求，其比例为 11.53-40.20%。对其进行跨项目一致性预测，实验结果如 5-8 所示。

从表中可以看出，四个系统的预测效果都比较差，精确率在 15.36%-61.01% 之间，召回率则更低，仅在 10% 徘徊。分析原因是克隆创建实例的样本不平衡，其中大部分的数据是不需要一致性维护。所以，所训练的模型不够完善，跨项目预测能力无法达到实际应用的水平。

与第 3 章全属性实验进行对比（表 3-7），发现四个系统的跨项目预测效果下降的均十分严重。因此，对需要一致性维护的克隆创建实例，不建议使用跨项目的方式进行克隆一致性预测。

因此，对于需要一致性维护的克隆创建实例，本文同样不建议使用跨项目的方法

表 5-7 克隆创建的贝叶斯网络跨项目预测实验结果（不需维护）

Table5-7 The effectiveness for cross-project with Bayesian network at creating time(free)

| 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 |

式对需要一致性维护的克隆创建实例进行预测，而是选择自身数据进行预测。在开发初期由于缺乏数据问题，建议程序开发人员选择对不需要的克隆实例进行预测。

表 5-8 克隆创建的贝叶斯网络方法跨项目预测实验结果（需要维护）

Table 5-8 The effectiveness for cross-project with Bayesian network at creating time(meeting)

| 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）小结

在克隆创建的跨项目的预测中，所构建的预测模型的有效性会依赖于系统自身的某些特征。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，由于系统具有较多的不需要一致性维护的克隆创建实例，而需要实例较少。所训练的跨项目预测模型可以较好的预测不需要一致性维护的克隆实例，而对需要维护的预测效果较差。因此，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。

在软件开发初期,可以使用跨项目预测的方式,对克隆创建实例进行不需要一致性维护需求预测,仅仅允许此类的克隆实例产生,从而避免一致性维护代价。

5.6.4.2 克隆变化实例的实验结果

本节给出在克隆代码变化时,四个系统上的实验结果如表 5-9 和 5-10 所示。

(1) 不需要一致性维护

对于克隆变化实例,根据统计结果(表5-1),在系统 ArgoUML 中,不需要维护的变化实例较多(比例为 67.45%),其它三个系统中不需要一致性维护的实例较少(比例为 25.71-49.06%)。其中,系统 Tuxguitar 具有最少的需要维护的实例,比例为 25.71%,剩余两个系统中需要和不需要维护的实例数量相差不大。

对不需要一致性维护的克隆变化实例,跨项目一致性预测结果如表 5-9 所示。从表中可以看出,ArgoUML 的精确率最好,达 71.72% 以上,系统 jEdit 和 jFreeChart 的预测效果在 50% 左右,而 Tuxguitar 的预测效果最差。结合数据的不平衡性,发现预测效果向需要一致性维护的实例偏移。因此,精确度预测效果与样本自身的特征有关联关系,即最好选择向数据较多的一侧进行预测。同时,四个系统的召回率都不尽如意,未能达到较好的预测效果。分析原因,是由于样本之间具有较大的差异,跨项目数据无法较好的拟合被测试系统的样本数据。

结合全属性实验的实验结果(表 4-8),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,预测模型有效的。因此,本文建议使用系统自身的数据进行一致性预测,不建议使用跨项目的形式进行预测。

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 3 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维护需求。

(2) 需要一致性维护

对于克隆变化实例,根据统计结果(表5-1),在系统 ArgoUML 中,需要维护的变化实例较少(比例为 32.55%),其它三个系统中不需要一致性维护的实例较少(比例为 50.94-74.29%)。其中,系统 Tuxguitar 具有最多的需要维护的实例,比例为 74.29%,剩余两个系统中需要和不需要维护的实例数目相差不大。对需要一致性维护的克隆变化实例,跨项目一致性预测结果如表 5-10 所示。

从表中可以看出,Tuxguitar 的精确率最好,达 70% 以上,系统 jEdit 和 jFreeChart 的预测效果在 60% 左右,而 Tuxguitar 的预测效果最差,仅在 33% 左右。结合数据的不平衡性,发现预测效果向需要一致性维护的实例偏移。因此,精确度预测效果

表 5-9 克隆变化的贝叶斯网络方法跨项目预测实验结果（不需维护）

Table5-9 The effectiveness for cross-project with Bayesian network at changing time(free)

| 测试系统 | 阈值 | 推荐率 (%) | 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 |

与样本自身的特征有关联关系，即最好选择向数据较多的一侧进行预测。同时，四个系统的召回率都不尽如意，未能达到较好的预测效果。分析原因，是由于样本之间具有较大的差异，跨项目数据无法较好的拟合被测试系统的样本数据。

结合全属性实验的实验结果（表 4-6），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，预测模型有效的。因此，本文建议优先使用系统自身的数据进行一致性预测，当数据不足以预测自身项目时，可以使用跨项目的形式进行预测。

同时，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，开发人员依然可以使用本文第 3 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。

（3）小节

对于克隆变化实例，实验表明跨项目预测模型的有效性会依赖于系统自身的某些特征。本文建议开发人员优先选用自身的数据进行模型训练，以达到满意的预测效果。

当自身系统的数据太少而不足以训练模型时，跨项目预测效果将依赖于数据集的分布情况。对于被测试的软件系统，需要尽量选择哪些具有相似分别地测试系统，从而训练模型进行跨项目预测。以本文使用的实验系统为例，对于其中三个系统，其需要维护的克隆变化实例较多，因此所训练的跨项目模型，在需要一致性维护的预测中预测效果要优于不需要维护的预测。

同时，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。当跨项目的模型也无法使用时，建议建议开发人员使用本文第 3 章的方法，在克隆代码创建时进行一致性维护需求预测，尽量避免会导致一致性变化的克隆代码。

5.6.4.3 讨论

综上所述，可以回答第三个研究子问题：

首先，于项目之间的差异性，测试集和训练集也具有较大差异，跨项目预测能力达不到项目内预测的能力。因此，建议开发人员优先选择项目内的预测方式。

其次，在软件开发初期，由于缺乏系统自身的数据而无法训练机器学习模型时，可以使用跨项目的克隆一致性预测，帮助避免由于克隆代码所导致的维护代价。但由于训练数据的样本不平衡性，跨项目预测能力会向数据较多的一侧倾斜，因此建议选择数据较多的一个角度进行预测。以本文为例：（1）在克隆代码创建时，建议对不需要一致性维护的克隆实例进行预测。（2）克隆代码变化时，建议对

表 5-10 克隆变化的贝叶斯网络方法跨项目预测实验结果（需要维护）

Table5-10 The effectiveness for cross-project with Bayesian network at changing time(meeting)

| 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 |

需要一致性维护的克隆实例进行预测。

最后,在进行选择跨项目预测时,软件开发人员可以根据自身需求,选择具有相似分布的训练系统,以达到较好的预测效果。如何选择相似的系统,将是本文未来的研究工作。

5.7 克隆一致性维护需求预测插件的设计与实现

为了与软件开发过程相结合,本章还设计并实现了一个克隆代码的一致性维护需求插件。所设计的插件可以嵌入到软件开发环境中 (eclipse),帮助软件开发人员实现边开发、边预测、边维护克隆代码的一致性维护需求。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违背缺陷,从而提高软件质量和可维护性。

5.7.1 克隆一致性维护需求预测插件的总体框架

本文所设计的克隆代码一致性维护需求预测插件的体系结构如图 5-2 所示,从上到下依次分为交互层、控制层、功能层、传输层、数据层。

交互层是面向开发人员的展示层,开发人员可以通过该层与插件进行交互。通过“数据载入”可以载入相应的数据,并在操作界面中选择不同的“功能按钮”来调用插件相应的功能,还可以将一些中间数据进行保存。同时,交互层还可以向开发人员提供克隆一致性维护需求预测的结果,通知开发人员采取相应的操作。

控制层主要负责交互层与功能层之间的接口实现。通过控制层对开发人员的交互层请求进行响应,并然后调用功能层完成特定的功能。当完成特定功能后,控制层还会将功能层所返回的结果反馈给交互层。

功能层是本文插件的核心所在,实现了克隆代码一致性维护需求预测的核心功能,在整个插件中起到了最关键的作用。功能层实现了五个不同的功能模块,包括预处理模块、属性特征提取模块、模型训练模块、克隆代码跟踪模块和一致性维护需求预测模块。功能层通过接受用户所指定的输入数据,并实现相应的功能完成用户的需求。

传输层负责数据层与功能层之间的数据交换与传输。在考虑代码的一致性维护需求预测中,需要一些输入数据的支持,比如系统源代码和克隆检测工具。传输层所负责的即是数据的传输。

插件的最后一层是数据层,可以向插件提供基本的数据以及一些必要的输出等。在本文设计的插件中,数据层数据主要包括系统源代码数据、克隆代码检测结

果数据、功能层生成的机器学习训练集数据、模型文件以及其它的一些中间结果等。

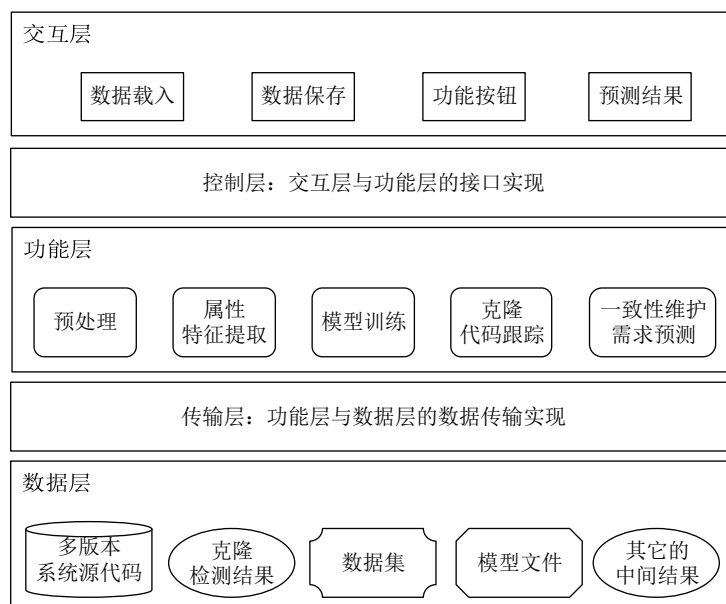


图 5-2 克隆一致性维护需求预测插件的体系结构

Fig.5-2 The framework for the plug-in of clone consistency-requirement prediction

5.7.2 克隆一致性维护需求预测插件的功能模块设计

本文的克隆代码一致性维护需求预测插件主要包括五个模块：预处理模块、属性提取模块、模型训练模块、克隆跟踪模块和预测模块。插件的模块设计如图 5-3 所示。预处理模块构建克隆家系并识别克隆演化模式，帮助程序开发人员收集系统中的克隆实例。属性提取模块实现对克隆实例的表示，提取不同的属性组表示相对应的克隆实例。预测模块可以根据不同用户需求，通过调用 WEKA 中的机器学习算法，构建和训练克隆一致性维护需求的预测器。克隆跟踪模块可以跟踪系统中的克隆代码产生和变化。预测模块可以根据克隆跟踪的过程，对新产生的克隆代码和克隆代码的变化进行一致性维护需求预测。

(1) 预处理模块

预处理模块通过识别克隆检测工具的检测结果，并构建克隆家系和识别克隆演化模式，从而收集并生成相应系统的克隆实例。首先，需要人工的使用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

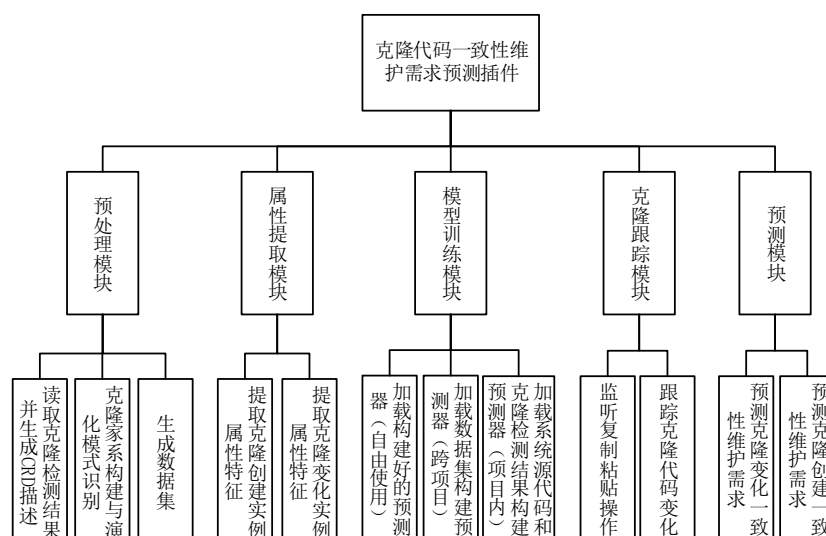


图 5-3 克隆一致性维护插件的功能模块

Fig. 5-3 The modules for clone consistency-requirement prediction

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。然后，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。最后，根据所构建的克隆家系和识别的演化模式，可以方便的生成相应系统的克隆创建和变化实例，从而生成其相应的数据集。值得注意的是，完整的数据集需通过调用属性提取模块生成具体的属性特征。

（2）属性提取模块

属性提取模块可以提取克隆实例的属性特征，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。对克隆变化实例，从克隆组的角度提取代码属性、上下文属性、演化属性和相对应的克隆变化情况。首先，对代码属性和上下文属性的提取，可以使用抽象语法树 (Abstract Syntactic Tree, AST) 对程序源代码进行解析并提取。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 AST¹。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，对于演化属性的提取，可以取通过遍历该克隆实例的克隆家系，根据其属性的描述计

¹抽象语法树可参见：http://www.eclipse.org/articles/Article-JavaCodeManipulation_AST/

算相应演化属性。

(3) 模型训练模块

模型训练模块根据用户的请求，调用 WEKA 构建和训练机器学习模型。为灵活的使用本文方法，本文提供三种构建和训练预测模型的方式：

(a) 项目内模型训练，该方式用项目自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机器学习模型，并将训练好的模型应用于克隆代码的一致性维护需求预测中。

(b) 跨项目模型训练，该方式使用其它项目数据作为训练集来训练预测模型。在软件开发初期，由于演化时间较短导致其历史的克隆实例较少，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。首先，用户指定相应的训练系统，并导入相应的输入数据。然后，通过调用预处理和属性特征提取模块生成所需的训练集。最后，使用训练集训练跨项目的预测模型，并将其应用于当前测试系统的一致性维护需求预测中。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，本文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。

(c) 加载已有训练模型，该方式直接加载已经训练好的机器学习模型。由于机器学习模型的构建和训练往往需要大量的时间，开发人员不应该也不需要每次开发时都训练机器学习模型。因此，在实际的开发过程中，模型的训练和开发时的预测可以分开进行。首先通过 (a) 和 (b) 两种方式进行模型的训练，然后使用 (c) 加载训练好的模型文件，最后进行克隆代码一致性维护需求预测。

(4) 克隆代码跟踪模块

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）。

首先，该模块可以跟踪开发人员的复制和粘贴操作。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，监测复制和粘贴操作即可跟踪克隆创建实例的产生。但是，在实际的软件开发过程中，大部分的复制和粘贴操作是对一些具体的变量进行的，这些操作不会导致克隆代码。因此，本文对复制粘贴操作进行了筛选，认定复制和粘贴的代码行数最少为 5 行的操作，才是会导致克隆代码的复制和粘贴操作。

然后，该模块还可以跟踪系统中克隆代码的变化。对克隆代码跟踪的功能有两

种实现的策略，一是根据 CRD 跟踪演化的克隆代码代码，而是通过 eclipse 开发平台所提供的文档内片段位置更新功能（位置更新器）实现。对于第一种方式，克隆代码的 CRD 中存储了其上下文信息，通过比较开发人员当前修改的代码片段是否和系统中的克隆代码区域重合，认定是否对克隆代码片段进行了修改。对于第二种方式，对于一个打开的文档，先确定该文档内是否含有克隆代码，如果含有克隆代码，则需要对其包含的克隆代码进行跟踪。位置跟踪器通过克隆代码片段的首字符在文档中的偏移量和片段长度来表示要跟踪的范围。将要跟踪的代码片段位置和跟踪其位置的更新器通过更新器种类绑定在一起，便可实现克隆片段的位置跟踪。然后，通过判断该克隆代码片段是否被修改（与上一版本比较），识别克隆代码的变化。

（5）预测模块

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维护需求。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当前版本中所有的克隆组的演化属性均提取出来。当克隆变化实例产生时，直接使用其作为属性，从而避免属性提取的时间。

5.7.3 克隆一致性维护需求预测插件的实现

本文基于 eclipse 实现了克隆一致性维护需求预测插件。eclipse 是一个基于 JAVA 语言的开源集成开发环境，具有良好的可扩展性和跨平台性。eclipse 是一个具有极小内核的开发平台，其所有功能都以插件的形式添加到此内核上。同时，eclipse 具有极为友好的插件开发环境，用户可以根据自身需要开发所需的功能，并以插件的形式发布在 eclipse 平台上。因此，本文也选择 ecilpse 实现克隆代码的一致性维护需求预测插件。

同时，本文设计的插件需要对不同的机器学习模型进行训练，并对在软件开发过程进行一致性维护需求预测。为了实现此功能，本插件也集成了 WEKA 的软件开发包。WEKA 是一个 JAVA 语言实现的数据挖掘和机器学习开源工具，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，从而极为灵活的进行模型的训练和预测²。

²使用 WEKA 可参见：<http://weka.wikispaces.com/>。

本文所设计的插件运行截图如图 5-4 所示。由图中看出，本章实现的插件会在 eclipse 开发环境的菜单栏中，新增加一个新的菜单项“Clones”。该菜单具备克隆一致性维护需求预测的功能。

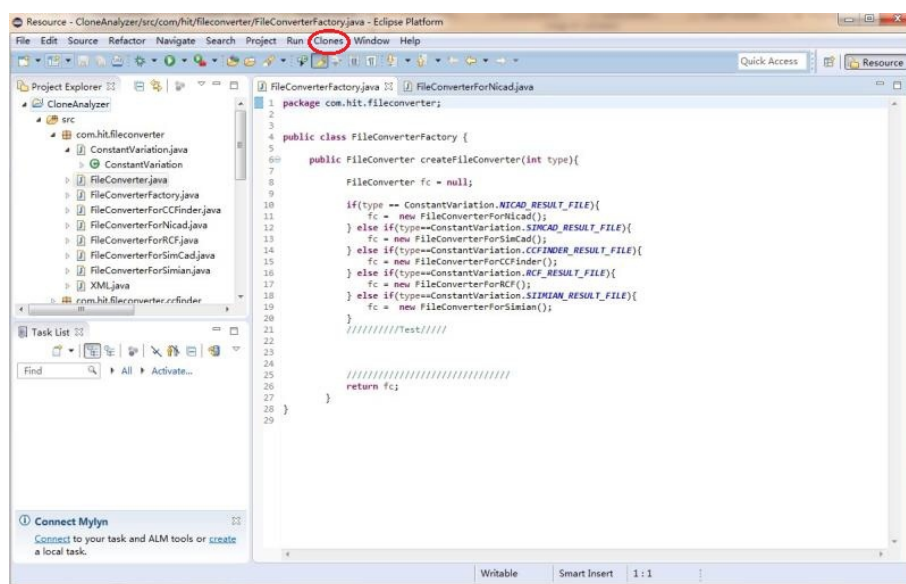


图 5-4 插件安装完成后 Eclipse 界面

Fig.5-4 The screen-shot for eclipse with installed plug-in of clone consistency prediction

在使用本文所设计的一致性维护需求预测插件时，需要提前加载已训练好的机器学习模型。为方便用户使用，本章插件提供三种方式加载预测模型：（1）加载系统源代码及克隆检测工具的检测结果构建预测模型（项目内）；（2）加载已有软件历史版本特征向量训练预测模型（跨项目）。（3）加载已有预测模型。加载所需功能截图如图 5-5 所示。

成功加载预测模型后，该插件便可以预测克隆代码的一致性维护需求。通过监听开发过程中的克隆创建和变化实例，并预测克隆代码的一致性维护需求。图中 5-6 是一个会导致额外维护代价的克隆创建操作的警告示例。如图所示，本插件会警告软件开发人员，其操作会引发额外的维护代价。根据预测结果，开发人员可以采取拒绝此复制粘贴操作，以避免在其未来演化过程中的克隆代码的额外维护代价。

5.8 本章小结

在软件开发过程中，克隆代码的一致性变化会增加软件维护的代价。然而，软件开发初期由于克隆实例较少，无法训练机器学习模型进行一致性预测。因此，本章在第 3 章和第 4 章研究的基础上，统一了克隆创建和变化的一致性维护需求定

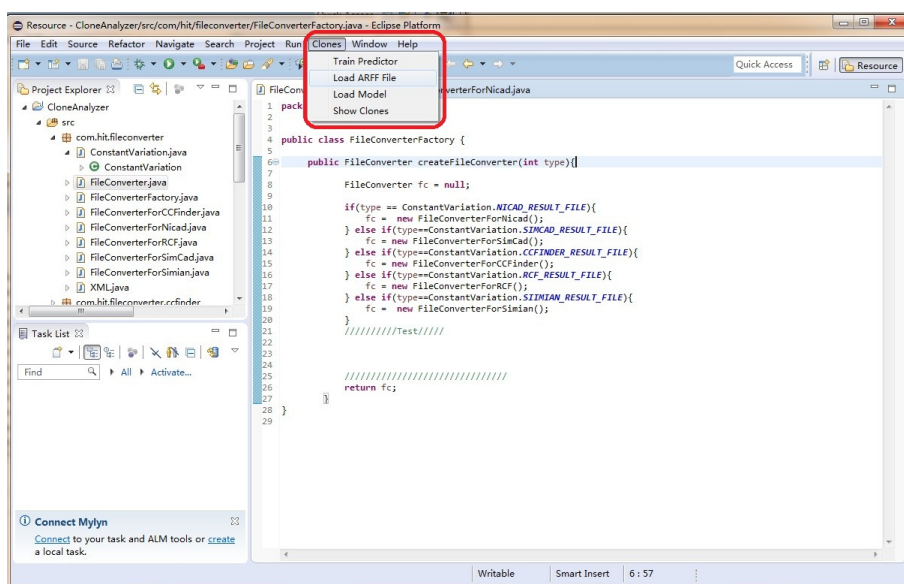


图 5-5 插件加载预测模型的截图

Fig.5-5 The screen-shot for loading the models of clone consistency prediction

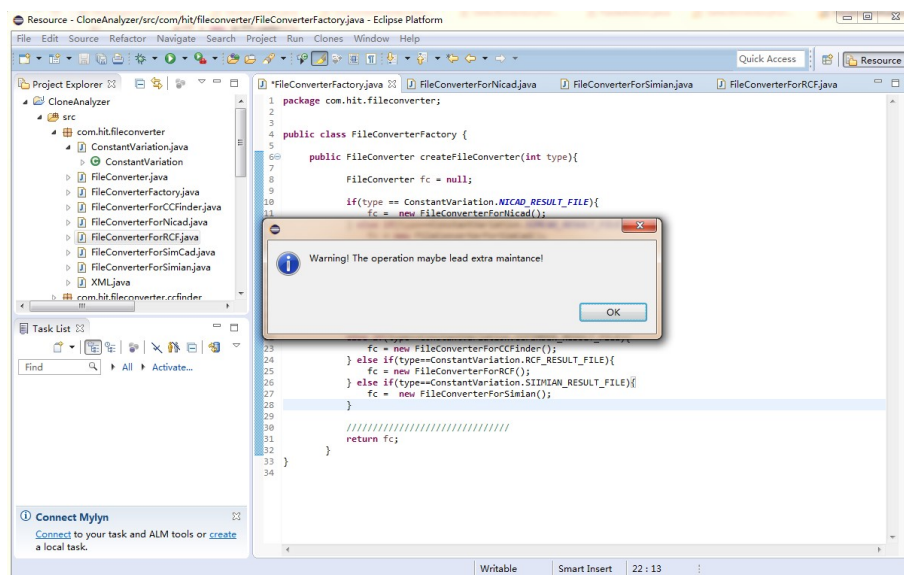


图 5-6 需要一致性维护的预测结果示例

Fig.5-6 The screen-shot for a prediction example that meeting consistency

义,并结合软件开发过程,对跨项目的克隆代码一致性维护需求预测进行研究。在四个开源系统上进行了跨项目实验,从三个不同的角度回答了本章所提出的研究问题。跨项目的预测实验结果表明:在软件开发初期阶段,可以使用跨项目的训练数据训练机器学习模型,并应用于其它系统的克隆一致性需求预测中。同时,由于样本的不平衡性,不推荐使用 SVM 模型进行跨项目预测。训练集的规模会对跨项目的预测能力产生影响,建议增大训练集的规模,但是仍然无法弥补项目之间的差异性。最后,本章设计和开发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开发过程中预测克隆代码的一致性,帮助提高软件的可维护性和软件质量。

结 论

克隆代码在随着软件系统进行演化的过程中，克隆代码的演化与其所发生的一致性变化，导致软件越来越难以理解和维护，已经成为了影响软件质量的一个重要因素。然而，现有的方法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维护问题。因此，本文提出了基于软件演化的克隆代码分析与一致性维护方法，结合软件开发过程重点研究了克隆代码演化特征分析、克隆代码创建和变化的一致性维护需求预测和跨项目的克隆代码一致性维护需求预测，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且客观的理解克隆代码及其演化过程，提出了一个基于 X-means 聚类的克隆代码演化特征分析方法。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。从克隆片段、克隆组和克隆家系三个不同的角度，提取相应的属性值表示克隆代码及其演化情况。在此基础上使用 X-means 聚类方法分析和挖掘克隆代码演化特征。实验结果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代码要多于发生不一致变化的克隆代码。该方法为后续克隆代码的一致性维护研究奠定了基础。

(2) 针对新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，提出了克隆代码创建一致性维护需求预测方法。通过定义克隆代码创建一致性维护需求，将问题转化为可以应用机器学习方法解决分类问题。通过构建系统克隆家系并识别克隆创建一致性变化模式，从系统中收集克隆创建实例用于训练机器学习模型，提取代码属性表示被复制的克隆代码、上下文属性表示被粘贴的克隆代码。在四个软件系统上进行了实验评估，实验结果表明该方法以较高的准确率和召回率有效地预测克隆代码创建一致性维护需求，可以帮助软件开发人员避免克隆代码的额外维护代价。

(3) 针对演化中的克隆代码的一致性变化可能会导致一致性缺陷问题，提出了克隆代码变化一致性维护需求预测方法。通过克隆代码变化的一致性维护需求，将问题转化为可以应用机器学习方法解决分类问题。通过构建系统的克隆家系和识别其全部的演化模式（尤其是克隆变化的一致性变化模式），从系统中收集克隆变化实例用于训练机器学习模型。从克隆组的角度提取了代码属性、上下文属性，从演化的角度提取了演化属性和变化信息共同表示发生变化的克隆代码。在四个

软件系统上进行了实验评估，实验结果表明该方法以合理的准确率和召回率有效地预测克隆代码变化的一致性维护需求，可以帮助软件开发人员避免克隆代码的一致性违背缺陷。

(4) 针对软件开发初期系统训练数据不足的问题，提出了跨项目克隆代码一致性维护需求预测方法，并统一克隆创建、变化的一致性维护需求为克隆一致性维护需求。将软件系统划分为训练系统和测试系统，通过收集训练系统的历史数据训练机器学习模型，在测试系统上进行跨项目一致性维护需求预测。在四个软件系统上进行了实证研究，实验结果表明在软件开发初期可以对系统进行跨项目的克隆一致性维护需求预测，并根据实验结果给出了一些建议帮助提高跨项目的预测能力。同时，结合软件开发过程，设计并实现了一个基于 eclipse 的克隆一致性维护需求预测插件，帮助开发人员边开发、边维护克隆代码，从而提高软件质量和可维护性。

在本文工作的基础上，还有以下工作有待于进一步研究：

(1) 在克隆变化一致性维护需求预测中，所构建模型的预测能力仍然有一定的改进空间。在未来研究中，拟通过提取新的属性用于表示克隆代码的真实变化，进一步提高模型的预测能力，以更高的准确率和召回率预测克隆代码一致性维护需求。

(2) 在跨项目的克隆一致性维护需求预测中，所构建模型的预测能力尚未达到令人满意的效果。在未来的研究中，拟通过提取与项目自身相关的属性、筛选与测试系统相似的数据等方法，进一步提高跨项目预测模型的预测能力，从而在软件开发初期帮助缺乏训练数据的新系统预测克隆一致性维护需求。

(3) 在克隆代码变化时，使用本文方法可以判断该变化的一致性维护需求，并提醒开发人员对克隆代码进行一致性维护。但是，并没有帮助开发人员维护克隆代码的一致性。在未来的研究中，拟结合程序分析技术，在判别需要一致性维护的基础上，对克隆代码自动地进行一致性维护，从而降低软件的维护代价、提高软件的可维护性。

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, CORDY J R. An empirical study of function clones in open source software[C] // Reverse Engineering, 2008. WCRE'08. 15th Working Conference on. 2008 : 81 – 90.
- [25] ALALFI M H, CORDY J R, DEAN T R, et al. Models are code too: Near-miss clone detection for Simulink models[C] // Software Maintenance (ICSM), 2012 28th IEEE International Conference on. 2012 : 295 – 304.
- [26] 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.
- [27] BASIT H A, JARZABEK S. Detecting higher-level similarity patterns in programs[C] // ACM Sigsoft Software engineering notes : Vol 30. 2005 : 156 – 165.
- [28] 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.
- [29] 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.
- [30] 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.
- [31] 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.
- [32] GÖDE N, KOSCHKE R. Incremental clone detection[C] // Software Maintenance

- and Reengineering, 2009. CSMR'09. 13th European Conference on. 2009 : 219 – 228.
- [33] SAJNANI H, SAINI V, SVAJLENKO J, et al. SourcererCC: scaling code clone detection to big-code[C] // Proceedings of the 38th International Conference on Software Engineering. 2016 : 1157 – 1168.
- [34] 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.
- [35] BULYCHEV P, MINEA M. Duplicate code detection using anti-unification[C] // Spring Young Researchers Colloquium on Software Engineering. 2008 : 51 – 54.
- [36] KRINKE J. Identifying similar code with program dependence graphs[C] // Reverse Engineering, 2001. Proceedings. Eighth Working Conference on. 2001 : 301 – 309.
- [37] WHITE M, TUFANO M, VENDOME C, et al. Deep learning code fragments for code clone detection[C] // Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. 2016 : 87 – 98.
- [38] BELLON S, KOSCHKE R, ANTONIOL G, et al. Comparison and evaluation of clone detection tools[J]. IEEE Transactions on software engineering, 2007, 33(9).
- [39] RATTAN D, BHATIA R, SINGH M. Software clone detection: A systematic review[J]. Information and Software Technology, 2013, 55(7) : 1165 – 1199.
- [40] 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.
- [41] SVAJLENKO J, ROY C K. Evaluating modern clone detection tools[C] // Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on. 2014 : 321 – 330.
- [42] 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.
- [43] 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.
- [44] HARDER J, GÖDE N. Efficiently handling clone data: RCF and cyclone[C]

- // Proceedings of the 5th International Workshop on Software Clones. 2011 : 81 – 82.
- [45] 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.
- [46] HARDER J. The limits of clone model standardization[C] // Software Clones (IWSC), 2013 7th International Workshop on. 2013 : 10 – 11.
- [47] KAPSER C J, HARDER J, BAXTER I. A common conceptual model for clone detection results[C] // Proceedings of the 6th International Workshop on Software Clones. 2012 : 72 – 73.
- [48] 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.
- [49] BAKOTA T. Tracking the evolution of code clones[C] // International Conference on Current Trends in Theory and Practice of Computer Science. 2011 : 86 – 98.
- [50] HARDER J, GÖDE N. Modeling clone evolution[J]. Proc. IWSC, 2009 : 17 – 21.
- [51] CAI D, KIM M. An empirical study of long-lived code clones[C] // International Conference on Fundamental Approaches to Software Engineering. 2011 : 432 – 446.
- [52] KRINKE J. Is cloned code older than non-cloned code?[C] // Proceedings of the 5th International Workshop on Software Clones. 2011 : 28 – 33.
- [53] BAZRAFSHAN S. Evolution of near-miss clones[C] // Source Code Analysis and Manipulation (SCAM), 2012 IEEE 12th International Working Conference on. 2012 : 74 – 83.
- [54] 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.
- [55] 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.
- [56] GODE N, HARDER J. Clone stability[C] // Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on. 2011 : 65 – 74.
- [57] HARDER J, GÖDE N. Cloned code: stable code[J]. Journal of Software: Evolution and Process, 2013, 25(10): 1063 – 1088.
- [58] 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.
- [59] 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.
- [60] 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.
- [61] 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.
- [62] 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.
- [63] 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.
- [64] HARDER J, TIARKS R. A controlled experiment on software clones[C] // Program Comprehension (ICPC), 2012 IEEE 20th International Conference on. 2012 : 219 – 228.
- [65] JUERGENS E, DEISSENBOECK F. How much is a clone[C] // Proceedings of the 4th International Workshop on Software Quality and Maintainability. 2010.
- [66] 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.
- [67] KAPSER C, GODFREY M W. "Cloning considered harmful" considered harmful[C] // 2006 13th Working Conference on Reverse Engineering. 2006 : 19 – 28.
- [68] RAHMAN F, BIRD C, DEVANBU P. Clones: What is that smell?[J]. Empirical Software Engineering, 2012, 17(4-5): 503 – 530.
- [69] 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.
- [70] HORDIJK W, PONISIO M L, WIERINGA R. Harmfulness of code duplication-a structured review of the evidence[J], 2009.
- [71] HANJALIC A. ClonEvol: Visualizing software evolution with code clones[C] // Software Visualization (VISSOFT), 2013 First IEEE Working Conference on. 2013 : 1 – 4.
- [72] HAUPTMANN B, BAUER V, JUNKER M. Using edge bundle views for clone visualization[C] // Proceedings of the 6th International Workshop on Software Clones. 2012 : 86 – 87.
- [73] VOINEA L, TELEA A C. Visual clone analysis with SolidSDD[C] // Software Visualization (VISSOFT), 2014 Second IEEE Working Conference on. 2014 : 79 – 82.
- [74] CORDY J R. Exploring large-scale system similarity using incremental clone detection and live scatterplots[C] // Program Comprehension (ICPC), 2011 IEEE 19th International Conference on. 2011 : 151 – 160.
- [75] HIGO Y, KAMIYA T, KUSUMOTO S, et al. Method and implementation for investigating code clones in a software system[J]. Information and Software Technology, 2007, 49(9) : 985 – 998.
- [76] LIVIERI S, HIGO Y, MATUSHITA M, et al. Very-large scale code clone analysis and visualization of open source programs using distributed CCFinder: D-CCFinder[C] // Proceedings of the 29th international conference on Software Engineering. 2007 : 106 – 115.
- [77] ASADUZZAMAN M, ROY C K, SCHNEIDER K A. VisCad: flexible code clone analysis support for NiCad[C] // Proceedings of the 5th International Workshop on Software Clones. 2011 : 77 – 78.
- [78] UDDIN M S, GAUR V, GUTWIN C, et al. On the comprehension of code clone visualizations: A controlled study using eye tracking[C] // Source Code Analysis and Manipulation (SCAM), 2015 IEEE 15th International Working Conference on. 2015 : 161 – 170.
- [79] ADAR E, KIM M. SoftGUESS: Visualization and exploration of code clones in context[C] // Software Engineering, 2007. ICSE 2007. 29th International Conference on. 2007 : 762 – 766.
- [80] FORBES C, KEIVANLOO I, RILLING J. Doppel-Code: A clone visualization

- tool for prioritizing global and local clone impacts[C] // Computer Software and Applications Conference (COMPSAC), 2012 IEEE 36th Annual. 2012 : 366–367.
- [81] JIANG Z M, HASSAN A E. A framework for studying clones in large software systems[C] // Source Code Analysis and Manipulation, 2007. SCAM 2007. Seventh IEEE International Working Conference on. 2007 : 203–212.
- [82] JIANG Z M, HASSAN A E, HOLT R C. Visualizing clone cohesion and coupling[C] // Software Engineering Conference, 2006. APSEC 2006. 13th Asia Pacific. 2006 : 467–476.
- [83] ZHANG Y, BASITH A, JARZABEK S, et al. Query-based filtering and graphical view generation for clone analysis[C] // Software Maintenance, 2008. ICSM 2008. IEEE International Conference on. 2008 : 376–385.
- [84] KERIEVSKY J. 重构与模式 [J]. 程序员, 2006, 3 : 047.
- [85] 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.
- [86] 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.
- [87] 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.
- [88] 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.
- [89] MANDAL M, ROY C K, SCHNEIDER K A. Automatic ranking of clones for refactoring through mining association rules[C] // Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on. 2014 : 114–123.
- [90] 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.
- [91] 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.
- [92] 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.
- [93] 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.
- [94] TSANTALIS N, MAZINANIAN D, KRISHNAN G P. Assessing the refactorability of software clones[J]. IEEE Transactions on Software Engineering, 2015, 41(11): 1055 – 1090.
- [95] 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.
- [96] 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.
- [97] 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.
- [98] ETTINGER R, TYSZBEROWICZ S, MENAIA S. Efficient method extraction for automatic elimination of type-3 clones[C] // Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on. 2017 : 327 – 337.
- [99] GÖDE N. Clone removal: Fact or fiction?[C] // Proceedings of the 4th International Workshop on Software Clones. 2010 : 33 – 40.
- [100] 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.
- [101] 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.
- [102] TAIRAS R, GRAY J. Sub-clones: Considering the Part Rather than the Whole.[C] // Software Engineering Research and Practice. 2010: 284–290.
- [103] 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.
- [104] 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.
- [105] KRUTZ D E, LE W. A code clone oracle[C] // Proceedings of the 11th Working Conference on Mining Software Repositories. 2014: 388–391.
- [106] ISHIHARA T, HOTTA K, HIGO Y, et al. Reusing reused code[C] // 2013 20th Working Conference on Reverse Engineering (WCRE). 2013: 457–461.
- [107] 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.
- [108] 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.
- [109] 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.
- [110] 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.
- [111] 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.
- [112] 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.
- [113] 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.
- [114] KOSCHKE R. Frontiers of software clone management[C] // Frontiers of Software Maintenance, 2008. FoSM 2008.. 2008 : 119 – 128.
- [115] KOSCHKE R, BAXTER I D, CONRADT M, et al. Software clone management towards industrial application (dagstuhl seminar 12071)[J]. Dagstuhl Reports, 2012, 2(2).
- [116] 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.
- [117] 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.
- [118] 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.
- [119] DANG Y, ZHANG D, GE S, et al. Transferring code-clone detection and analysis to practice[C] // Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track. 2017 : 53 – 62.
- [120] PELLEG D, MOORE A W, OTHERS. X-means: Extending K-means with Efficient Estimation of the Number of Clusters.[C] // ICML : Vol 1. 2000.
- [121] CORDY J R. The TXL source transformation language[J]. Science of Computer Programming, 2006, 61(3) : 190 – 210.
- [122] DEAN T R, CORDY J R, MALTON A J, et al. Agile parsing in TXL[J]. Automated Software Engineering, 2003, 10(4) : 311 – 336.
- [123] 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.
- [124] 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.
- [125] 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.
- [126] 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.
- [127] WANG X, DANG Y, ZHANG L, et al. Predicting Consistency-Maintenance Requirement of Code Clones at Copy-and-Paste Time.[J], 2014 : 773 – 794.
- [128] 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.
- [129] 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.
- [130] 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.
- [131] 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.
- [132] 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.
- [133] LEVENSHTAIN V I. Binary codes capable of correcting deletions, insertions and reversals[C] // Soviet physics doklady : Vol 10. 1966 : 707.
- [134] FRIEDMAN N, GEIGER D, GOLDSZMIDT M. Bayesian network classifiers[J]. Machine learning, 1997, 29(2-3): 131 – 163.
- [135] JOHN G H, LANGLEY P. Estimating continuous distributions in Bayesian classi-

- fiers[C] // Proceedings of the Eleventh conference on Uncertainty in artificial intelligence. 1995 : 338 – 345.
- [136] PLATT J C. 12 fast training of support vector machines using sequential minimal optimization[J]. Advances in kernel methods, 1999 : 185 – 208.
- [137] AHA D W, KIBLER D, ALBERT M K. Instance-based learning algorithms[J]. Machine learning, 1991, 6(1) : 37 – 66.
- [138] QUINLAN J R. C4. 5: programs for machine learning[M]. [S.l.] : Elsevier, 2014.
- [139] YAMANAKA Y, CHOI E, YOSHIDA N, et al. Applying clone change notification system into an industrial development process[C] // Program Comprehension (ICPC), 2013 IEEE 21st International Conference on. 2013 : 199 – 206.
- [140] 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.
- [141] 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.
- [142] 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.

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. (已发表, 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. (在审, 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 3 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] 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. (已录用, EI 收录, 一级学报, 对应于第 1 章, 第二作者导师为第一作者)
- [6] Zhang Fanlong, Yuan yue, Khoo Siau-cheng, Su Xiaohong*. Clone Creating Consistency Prediction Based on Bayesian Network[J]. Chinese Journal of Electronics. (待投, 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)
- [7] Zhang Fanlong, Su Xiaohong*. An empirical study of clone consistency prediction for cross-project[J]. (待投, 对应于第 5 章, 第一作者)
- [8] 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). (已在线发表, 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)
- [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 5 章, 第二作者)
- [10] 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)
- [11] 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《基于软件演化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 悉心地指导我进行学术研究，细致地与我讨论学术问题，认真的帮我修改学术论文。Prof. KHOO 严谨的治学态度、卓越的学术能力将是我终生学习的方向！

衷心感谢所有对本文提出宝贵意见的专家们，尤其是责任专家、外审专家以及答辩专家！在所有专家的帮助、批评和指正下，使得本文更加完善。

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！

衷心感谢我的父母！感谢他们将我抚养成成人，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。

博士只是人生不断学习的一个阶段，我将继续开启新的人生。以一句话勉励自身：“天行健，君子以自强不息；地势坤，君子以厚德载物！”

个人简历

张凡龙，男，生于 1987 年 11 月 06 日，山东省兖州市人。

2006 年 09 月——2010 年 07 月，就读于 东北林业大学 信息与工程学院 计算机科学与技术专业，并获得 工 学学士学位。

2010 年 09 月——2012 年 07 月，就读于 哈尔滨工业大学 计算机科学与技术学院 计算机科学与技术学科，并获得 工 学硕士学位。

2015 年 08 月——2016 年 08 月，于 新加坡国立大学 计算学院 计算机科学系 访问，Visiting Student。

2012 年 09 月——至今，就读于 哈尔滨工业大学 计算机科学与技术学院 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、克隆代码、软件仓库挖掘等。在攻读学位博士期间，以第一作者或第二作者身份发表论文 11 篇，其中 SCI 期刊 4 篇，国内一级学报 1 篇，国际会议 4 篇，国内期刊 2 篇，至今 SCI 收录 2 篇，EI 收录 4 篇。