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 ANALYSIS AND CONSISTENCY MAINTENANCE OF CODE CLONE BASED ON SOFTWARE EVOLUTION

张凡龙

哈尔滨工业大学

2017 年 10 月

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

工学博士学位论文

基于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：计算机应用技术

所在单位：计算机科学与技术学院

答辩日期：2017 年 10 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 ANALYSIS AND CONSISTENCY MAINTENANCE OF CODE CLONE BASED ON SOFTWARE EVOLUTION

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

摘 要

在软件开发过程中，开发人员通过复制粘贴既有代码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，从而影响了软件的质量、可理解性和可维护性。这引发了对克隆代码的大量研究，例如克隆检测、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。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，具有重要理论意义和实际应用价值。

在克隆代码随着软件系统的演化过程中，克隆代码可能会被开发人员修改而发生变化，进一步地加剧了克隆代码的问题。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。由于克隆代码彼此之间的相似性，一个克隆代码的变化可能会导致其它克隆代码的变化，称为克隆代码的一致性变化。该一致性变化会导致系统额外的维护代价，而遗忘这种变化则会导致克隆代码一致性违背缺陷，降低了软件质量和可维护性。鉴于此，本文基于软件演化研究克隆代码分析与一致性维护方法，通过分析提取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，验证了克隆代码一致性维护需求预测的必要性，并通过预测克隆代码的一致性维护需求帮助解决克隆代码的一致性维护问题，最后结合软件开发过程实现对克隆代码的同步开发与维护，可以提高软件质量、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，验证了克隆代码一致性需求预测的必要性。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特征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，在发生变化的克隆代码中有超过一半的发生了一致性变化，为克隆代码一致性维护需求预测研究奠定了基础。

针对新创建的克隆代码在其演化中的一致性变化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，帮助开发人员从规避克隆代码产生的角度降低克隆代码的维护代价。将软件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是否发生一致性变化称为克隆代码创建时一致性维护需求。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，提取代码属性和上下文表示新创建的克隆代码。最后，使用机器学习方法训练预测模型，并预测克隆代码创建时的一致性维护需求。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额外维护代价。

针对演化中克隆代码的一致性变化可能会导致克隆一致性违背缺陷的问题，研究并提出了克隆代码变化一致性维护需求预测方法，帮助开发人员实现克隆代码的同步开发与维护。将软件系统中发生变化的克隆代码称为克隆变化实例，并将该变化是否会引发克隆代码的一致性维护称为克隆代码变化时的一致性维护需求。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特征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，并预测克隆代码变化时的一致性维护需求。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代码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，对软件系统通过构建其克隆家系收集克隆实例，并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一致性维护需求预测模型可以在软件开发初期预测其它项目的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分析和理解克隆代码、维护克隆代码的一致性、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：软件质量；软件维护；克隆代码；克隆演化分析；克隆一致性维护

Abstract

During the software development, developers introduce a large number of code clones into software through reusing the existing code fragments. As code clones evolving with the time and the update of software, the software become more and more bloated and difficult to maintain, which can impact on the quality, comprehensibility, and maintainability of software. Hence, it sparks a great deal of research on code clones, such as clone detection, clone analysis, clone maintenance and etc. Clone detection can help developers gather the code clones from software, and clone analysis can help developers understand the presence of code clones in software, and clone maintenance can help developers solve the issues caused by code clones. Consequently, code clone research has a great theoretical significance and practical applicative value for improving the quality of software, enhancing the comprehensibility, and maintainability of software.

During the process of code clone evolution, code clones may be modified by developers that resulting the changes to clones, which will exacerbate the issue of code clones. The changes to the evolving code clone make them difficult to understand, and decrease the comprehensibility of the software. Meanwhile, changes to one code clone may give rise to some related changes to other code clones in a clone group according to the similarity of these code clones. We term such changes as clone consistent change to a clone group. The consistent change of code clones will increase additional maintenance cost, and will lead to clone consistency-defect when failing in making the changes, which will reduce the quality and maintainability of the software. Therefore, this thesis studies on the analysis and consistency maintenance of code clone based on software evolution. Analyzing and exploring the clone evolutionary characteristics of code clones can help developers understand the code clones. Predicting the consistent change of code clones can help developers solve the issues of code clone consistency maintenance. Thus, this thesis can help developers improve software quality, and reduce the cost of software maintenance through synchronously maintain code clones at the development time.

Aiming to address the difficulty on analyzing and understanding the evolving code clones, an approach for exploring code clone evolutionary characteristics based on clustering method is proposed in this thesis, which lay the foundation on the prediction of clone consistency-requirement. We firstly detect all code clones with detection tool from

software's repositories, and build all the clone genealogies for software to describe the evolution of the code clones. After that, the corresponding attribute sets are 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; yet, there are also a significant number of changed code clones, and more than a half of them possess the consistent change.

Aiming to address the problem of additional maintenance cost that caused by the clone consistent change in the evolution of a clone creating operation, we propose an approach for predicting clone creating consistency-requirement in this thesis, which can help developers avoid the extra maintenance cost of code clones from the perspective of clone prevention. We call the earliest code clone in software as "clone creating instance", and call that of whether this creating instance will lead to consistent change in its evolution 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 repositories. And then, we extract code attribute set to represent the copied code clone and context attribute set for the pasted code clone. Finally, the machine-learning models are trained with these 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 clone creating instances, which can help the developers to reduce the consistency maintenance cost of the code clone.

Aiming to address the problem of consistency-defect that caused by the clone consistent change in their evolution, an approach for predicting clone changing consistency-requirement is proposed in this thesis, which can help developers to synchronize the development and maintenance of code clones. We call the changed clone code as "clone changing instance", and call that of whether such changes will lead to a consistent change to this changed clone group as "clone changing consistency-requirement". Firstly, through detecting all the code clones and building all the clone genealogies, all the clone changing instances can be collected from software's repositories. After that, three different attribute sets from the perspective of clone group are extracted to represent the clone changing instance, including code attribute set, the context attribute set, and the evolu-

tionary attribute set respectively. 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 reasonably predict the consistency-requirement for changing instances, which can help the developers avoid clone consistency-defects.

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 cross-project consistency-requirement prediction 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 software repositories, we collect their clone instances by detecting code clones and building clone genealogist from software repositories, and represent all clone instances with different attribute sets. Then, the 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 clone cross-project consistency-requirement on the testing softwares. 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. Combining with the software development, we develop and implement an eclipse plug-in for predicting clone consistency requirement, which can help maintain the code clone at the developing time that reducing the cost of software maintenance.

In summary, this thesis presents a method for code clone analysis and consistency maintenance based on software evolution, that providing new ideas and approaches 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: Software quality; software maintenance; code clone; clone evolution analysis; clone consistency maintenance

目 录

Contents

第 1 章 绪论

1.1 课题背景及研究目的和意义

随着计算机软件广泛应用于经济、军事、商业等各个领域中，软件质量问题日益引起了人们的广泛重视。保证软件质量、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。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。在软件开发和维护的过程中，软件开发人员会向软件系统中不断添加新的功能，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。

与此同时，代码复用作为一种常见的软件开发手段，尽管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，软件开发人员通过复制和粘贴操作向系统中引入多种类型的克隆代码（Code Clone）[1]。克隆代码在大型软件系统中会占到代码总量的 7%-23%[2–4]，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。克隆代码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。具体来说，由于克隆代码之间的相似性，对某一个克隆代码的修改可能会导致其它克隆代码的变化，将之称为克隆代码的一致性变化（Clone Consistent Change）。克隆代码的一致性变化问题不仅仅会导致软件系统额外的维护代价；遗忘克隆代码的一致性变化还会导致克隆代码的一致性违背缺陷（Clone Consistency-Defect），降低了软件的可维护性，从而进一步降低软件质量。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，例如对克隆代码相关缺陷的研究[7–9]、对克隆代码的有害性研究[10–12] 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，研究人员展开了大量且深入的研究，并取得了较多的研究成果。目前对克隆代码的研究包含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。克隆检测研究是最早也是研究最为充分的一个研究

内容，旨在从软件系统中检测已经存在的克隆代码。到目前为止，研究人员提出和开发了许多克隆代码检测的方法和工具，例如 NiCad[13]、CCFinder[14]、Deckard[15]等等。但是，由于系统存在着大量的克隆代码并且情况复杂，目前克隆检测的效果并不能完全令人满意。更为重要的是，克隆检测研究既不能消除克隆代码，也不能消除克隆代码对软件产生的不利影响，因此无法直接帮助提高软件的质量及其可维护性。而对克隆代码的分析和维护研究可以弥补这一不足之处，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研究内容，其主要目的是帮助软件开发人员理解系统中存在的克隆代码。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[18–20]。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内容，旨在帮助软件开发人员维护系统中的克隆代码、解决克隆代码可能引发或者已经所引发的问题。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，并可能引发克隆代码的一致性变化[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代码的一致性违背缺陷的产生[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是否需要一致性维护（Clone 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，不仅可以降低克隆所导致的维护代价，还可以有效地避免克隆代码的一致性违背缺陷，从而帮助提高软件质量和可维护性。

基于以上分析，本文研究基于软件演化的克隆代码分析与一致性维护方法，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，本文结合软件与克隆代码的演化过程，研究并提取克隆代码演化特征，揭示克隆代码及其演化过程所隐含的信息，为克隆代码的一致性维护需求预测研究奠定了基础。然后，针对克隆代码一致性变化导致的额外维护代价和一致性违背缺陷的问题，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，帮助开发人员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软件系统的克隆代码一致性维护需求。本文所提出的方法可在软件开发过程中帮助开发人员边开发、边维护克隆代码，帮助提高软件质量和软件的可理解性和可维护性。

1.2 国内外研究现状及其分析

在目前对克隆代码的研究中，尚没有统一的克隆代码的形式化定义。但研究人员普遍认为克隆代码是根在某种相似性定义的作用下彼此相似的代码片段[1]。按照克隆代码的语法和语义的相似度，研究人员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）。

研究人员将 Type-1 的克隆代码称为精确克隆，表示完全相同的克隆代码。将 Type-2 和 Type-3 的克隆代码称为近似克隆，表示在语法相似的克隆代码。将 Type-4 的克隆代码称为语义相似的克隆代码。除此之外，也有研究人员研究其它类型的克隆代码，如函数克隆[24]、模型克隆[25] 和结构克隆[26,27] 等。函数克隆指的是克隆代码为一个完整函数的克隆代码，函数克隆也可以按照语法语义分成 Type 1-4 克隆。同时，在模型驱动的软件开发过程中，会产生大量的模型文件。这类模型文件中相似的模型图（例如 Simulink 模型图）称为模型克隆。除此之外，有别于代码级别的克隆，结构克隆是较大粒度的克隆形式，一个结构克隆是一个包含若干个代码克隆的文件, 可帮助软件开发人员分析文件和模块之间的关联程度。

1.2.1 研究热点与趋势

国内外对克隆代码的研究并没有软件工程的其它领域那么广泛，克隆代码研究仅仅是软件工程领域的一个较为细小的分支，克隆代码领域的文献数量也未达到较大的规模。但是，克隆代码很早就引起了研究人员的注意，迄今为止研究人员对克隆代码的研究已有超过 20 年的时间。目前，克隆代码研究已经称为了软件工程领域的一个重要分支，每年均召开克隆代码研讨会（International Workshop on

Software Clones），并与软件维护的相关会议一起举办，如 SANER、WCRE 等等。

为方便研究人员查阅相关研究，截止到 2013 年，阿拉巴马大学维护了一个克隆代码领域的文献库[1]，并按照研究内容和目标将对克隆代码的研究划分为以下四个方向（如图 1-1 所示）：

• 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评估相关的研究内容。

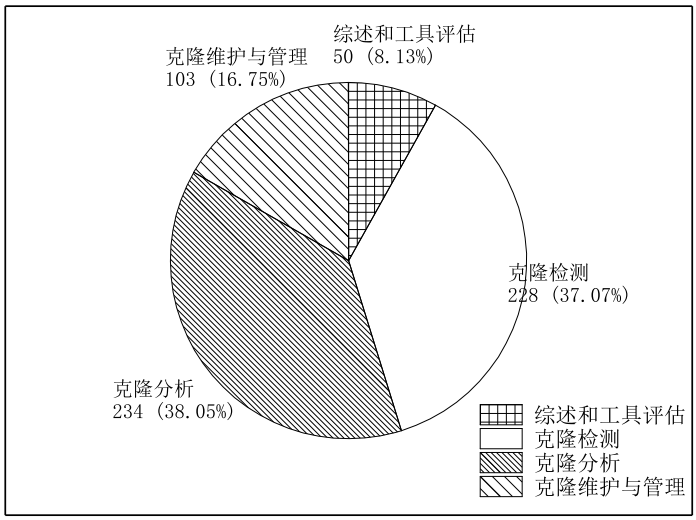


图 1-1 克隆代码研究领域文献分布情况

Fig.1-1 Literature distribution of code clone research

根据此文献库，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。为了进一步分析和反映克隆代码研究的研究热点和发展趋势，本文在该文献库的基础上，检索和收集了截止到 2016 年的克隆代码的研究论文。本文所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、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 461

[1] 阿拉巴马大学关于克隆代码的文献库：http://students.cis.uab.edu/tairasr/clones/literature/。

篇，期刊论文 120 篇，学位论文 34 篇，共计 615 篇[2] 。

同时，图 1-1 也给出了克隆代码研究领域的文献分布情况。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 37.07% 和 38.05%；而克隆维护与管理的研究论文占比较少 (占 16.75%)。

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 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。

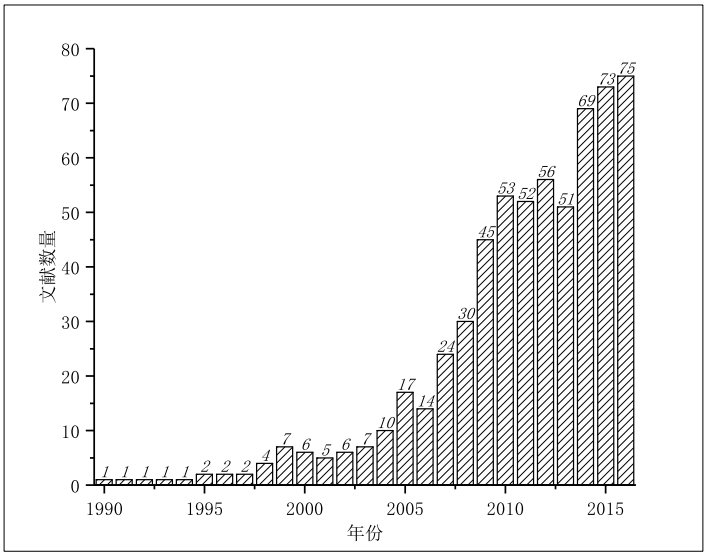


图 1-2 克隆领域每年发表的论文数量

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

[2]在收集的过程中，仅选取领域内权威期刊、会议以及学位论文（期刊和会议选自 CCF 所推荐的学术刊物目录以及克隆代码研讨会）。

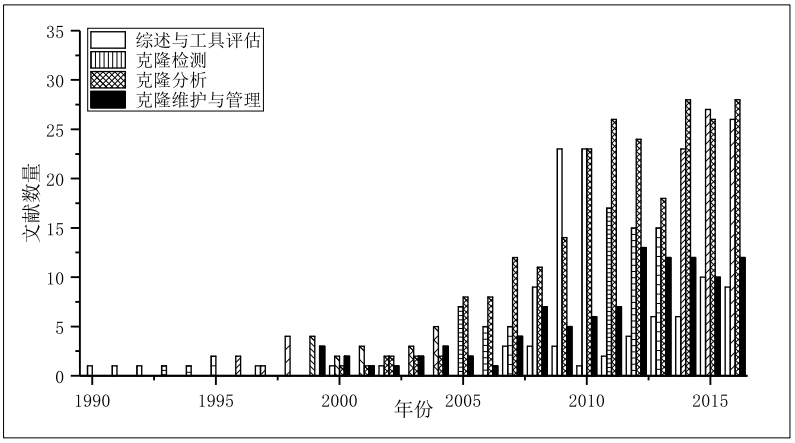


图 1-3 各个克隆研究活动每年发表的文献数量

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

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。

一般而言，克隆代码检测过程分为三个步骤：代码的中间表示、相似性匹配和报告检测结果。“代码的中间表示”是使用不同的方法对源代码进行抽象表示或转换，将其表示为抽象语法树等一些中间表示形式。“相似性匹配”是对代码的中间表示形式进行相似性匹配与检测，寻找相似的代码片段。“报告检测结果”是将彼此相似的克隆代码以某种克隆组织方式存储检测结果，并向分析人员提供克隆检测报告。

1.2.2.1 克隆检测方法

迄今为止，研究人员已提出了多种克隆检测方法，并同时也开发了相应的克隆检测工具。根据所使用的技术不同，可以将克隆检测划分基于文本 (Text)、基于 Token、基于树 (如 Abstract Syntax Tree，AST)、基于程序依赖图 (Program Dependency Graph, PDG) 和基于度量值 (Metric) 等方法。

基于 Text 的克隆检测方法是通过直接比较源代码文本，使用字符串匹配等算法来检测克隆代码。因其并没有对源程序进行词法分析，大部分仅可以较好地支持 Type-1 克隆代码的检测。目前，使用较多的基于文本的克隆检测工具主要有

duploc[5]、Simian[3]、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]、SimScan[4]、Deckard[15]、CloneDigger[35] 等。

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

基于 PDG 的克隆检测方法的主要思路是，将源代码转化成程序依赖图 (包括数据依赖图和控制依赖图)，然后通过寻找同构的子图来检测克隆代码。因程序依赖图表示了程序的语义信息，所以该方法可以支持语义相似的 Type-4 克隆代码的检测。但由于程序依赖图生成算法和图匹配算法的时间和空间复杂度极高，因而导致这类检测算法的时空开销过大，使其无法应用于大规模程序的克隆代码检测。目前基于 PDG 的克隆检测工具主要有 Duplix[36] 等。

基于度量值的方法将源代码转换为某种中间表示，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 等人使用深度学习技术检测系统中的克隆代码[37]。

[3]Simian - Similarity Analyser：http://www.harukizaemon.com/simian/index.html

[4]SimScan：http://www.blue-edge.bg/simscan/

1.2.2.2 克隆检测方法及工具评估

不同的克隆检测方法和检测工具各有其优缺点，分别适合检测不同类型的克隆代码，对同一类型的克隆代码的检测效果也不尽相同。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[38–41]。Bellon 对 6 个克隆检测工具进行了评估，分别对比了查准率、查全率以及时间和空间等性能[38]。Rattan 通过使用一种标准的系统文献综述方法，详细分析了克隆检测方面的 213 篇文献，并对不同的检测方法进行了评估分析，并给出了未来的研究方向[39]。此外，Roy 重点分析了检测工具的使用技术和适用环境，对克隆检测工具的检测效果进行了详细的对比，对帮助用户选择和使用克隆检测工具有重要的指导意义[40][41]。

本文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[39]：“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。

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
Token	NiCad[13]	1、2、3	较好 1、2、3
	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[42]	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[43]	1、2、3、4	较好 1、2，较差 3、4

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 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。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。基于度量值的检测方法目前仅有一些检测方法被提出[3,43]，缺少相应的检测工具。

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。克隆分析研究主要包括克隆表示、克隆演化、克隆评价和克隆可视化等。

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技术检测克隆代码，因此不同的检测工具的克隆检测结果往往是不同的。

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 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。RCF 还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]。但是，RCF 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]。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的语法信息、结构信息等表示克隆代码，有助于辅

[5]RCF 格式文档：http://www.softwareclones.org/rcf.php。

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，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。Kapser 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 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]。

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。

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

克隆演化过程分析最早是 2001 年由 Antoniol 等人提出的，使用时间序列描述克隆代码的演化模型[48]。2005 年，Kim 提出了克隆家系模型用于描述克隆代码的演化过程，是迄今为止最好的演化模型[16]。而 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。

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。克隆寿命是指克隆代码在系统中的存在时间克隆稳定性关注的是在克隆代码的生存期发生变化的问题。特别地，克隆代码的一致性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。

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

[6] UCM 格式文档：http://www.softwareclones.org/ucm/index.php/Main_Page。

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。

表 1-2 克隆演化特征分析

Table1-2 The analysis of clone evolutionary characteristic

文献	克隆模式、特征	结论
[16]	克隆寿命/一致性变化	观察并分析克隆家系寿命与一致性变化规律
[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	在发生变化的克隆代码中，约一半是一致性变化
[59] [60]	延后传播	Type-3 更容易发生延后传播并导致缺陷

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，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。

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，即克隆代码有害性分析。因此，克隆评价主要包括缺陷分析、维护代价分析、有害性分析等。

在早期，研究人员认为克隆代码是一种最刺鼻的代码坏味，是因为它有可能

会引发相关缺陷。因此，人们研究的关注点主要集中在克隆代码的缺陷分析上。Juergens 等人研究发现遗忘克隆代码的一致性变化容易引发相应的软件缺陷，从而降低了软件质量[7,61]。Gauthier 通过对克隆代码进行安全性分析，在开源软件 Joomla 和 Moodle 中发现了几个潜在的影响软件质量的安全漏洞和缺陷[8]。但也有另外一些证据表明克隆代码的不一致性变化并不一定会引发缺陷，因此不会对软件质量产生显著的影响。例如，Bettenburg 对不一致性变化是否引发缺陷的研究发现，仅有极少数的不一致性变化会引发缺陷[62]。文献[9] 的研究则表明 Type-3 克隆代码中大约有 17% 的代码含有缺陷，且 Type-3 克隆不容易发生不一致性变化。文献[63] 通过对缺陷密度的分析发现，面向对象程序中的克隆类含有更少的缺陷，并且 Type 3 克隆含有的缺陷最少。因此有人将克隆代码研究应用到缺陷检测中，但没有直接证据证明克隆代码与缺陷密切相关[64,65]。因此，克隆缺陷分析并不能直接证明克隆代码是有害的。

研究人员还从维护代价的角度去分析克隆代码对软件产生的影响[66]。Harder 通过实验发现克隆代码的存在并不会增加缺陷修复的时间，但是缺陷未被修复则可能导致维护代价的增加[67]。Lozano 通过研究克隆代码的可变性，也发现克隆代码的存在会增加软件维护的代价[68]。Juergens 不仅认为克隆代码会增加维护代价，还提出了一个计算模型来计算克隆维护代价[69]。Monden 的研究则发现包含少量克隆代码的软件模块比不含克隆代码的软件模块更可靠，但含有大量克隆代码的软件模块则正相反，实验表明克隆代码与软件可靠性和维护代价有一定的关联，但这种关联关系并不十分明确[70]。以上维护代价分析的研究表明克隆代码的存在有可能会增加软件的维护代价。

虽然缺陷分析和维护代价分析可用于评价克隆代码对软件产生的影响，但却不能作为评价克隆代码是否有害的直观证据。正因如此，近些年来又展开了克隆代码有害性分析的研究。Kapser 通过对比 11 种克隆模式在软件开发和维护过程中的优缺点，并在开源软件 Apache 和 Gnumeric 上进行实证研究，发现在 Apache 中大约有 71% 的克隆代码被分类为有益克隆，对系统维护具有积极的影响，是一种合理的存在方式，因此人们应当正视在开发过程中克隆代码的长期存在[10,71]。Kapser 通过对 Apache Web Server 中的克隆代码进行分析，研究发现其中一个子系统中聚集了大量的克隆代码；该子系统的克隆代码增加了系统的功能性，对软件开发过程是有益的[72]。Selim 使用风险模型判定克隆代码是否有害，研究结果发现克隆代码并不比非克隆代码具有更高的有害风险[11]。Wang 提出利用贝叶斯网络对克隆代码进行有害性预测的方法[12]，该方法可用于辅助开发人员决定是否可以通过复制粘贴的方式引入新的克隆代码。Higo 则提出一种提取有问题的克隆代码（有害克

隆）的方法，该方法先对程序进行标准化，然后利用检测工具检测克隆代码，最后对检测到的克隆代码进行过滤、合并等操作来提取有问题的克隆代码，在 Linux 内核 2.6.6 上的实验结果表明只有少数克隆代码是有问题的克隆代码[73]。Hordijk 提出了一个结构化的证据模型分析克隆代码的有害性，研究结果表明只有少部分的证据证明克隆代码是有害的，并且仍需更多的研究去支撑这一结论[74]。从上述对克隆代码有害性分析的结果不难得出结论，克隆代码并非都是有害的。

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。克隆可视化是使用可视化技术将克隆代码在软件中的分布情况以及克隆关系以图形的方式展现出来，以辅助开发人员更加方便直观地理解、分析和维护克隆代码。

目前，使用较多的可视化技术主要有捆绑图、散点图和树图。捆绑图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[76–78]。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[76]。Hauptmann 则使用捆绑图分析软件各模块之间的耦合关系[77]。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[79–81]。Cordy 使用散点图分析软件之间的相似性[79]，Higo 则使用散点图分析软件中的克隆模式[80,81]。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有 VisCad[82,83]、SoftGUESS[84]。

克隆可视化的作用不仅仅是可视化克隆代码，更重要的作用是通过克隆代码的可视化辅助软件开发人员分析克隆代码对软件产生的影响。例如可视化工具 Doppel-Code 可以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[85]。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[86,87]。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[88]。

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	研究发现克隆代码的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
	[62]	积极	克隆代码的变化并不会引入相关缺陷，不会对软件的质量产生影响
	[63]	积极	克隆代码含有更少的缺陷，Type-3 所含有的缺陷最少
	[64]	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
	[65]	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 Type-3 克隆更容易发生一致性变化
	[67]	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
	[69]	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
	[70]	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
	[71] [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
	[75]	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
	[73]	中立	使用分类模型抽取有问题的克隆代码，只有少部分是有问题的克隆
	[74]	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

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。克隆代码维护与软件开发过程结合得较为紧密，目前的研究主要包括克隆重构、克隆复用、克隆规避和克隆管理方法。

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[89]。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。

由于重构所需的条件较为苛刻，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[90–93]。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[90]。Mende 实现了一个工具在软件中识别可以被重构的函数克隆[91]。Schulze 通过计算克隆代码的可重构指数识别可重构的克隆代码[92]。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[93]。

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[94–96]。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[94]。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[95]。Zibran 提出一种极限编程方法对克隆代码进行重构调度[96]。Liu 等人提出了一个重构调度方法帮助优化调度过程[97]。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[98]，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了分析，并获得一些有用的结论[99]。

克隆重构最主要的目的是消除系统中的克隆代码，研究人员提出了许多具体的重构方法[100–103]。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[100]。Krishnan 通过分析克隆代码的程序依赖图，确定给定克隆代码能否被安全的重构，然后通过检测和参数化克隆代码的差异点对克隆代码进行重构，取得了较好的效果[101]。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[102]。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[103]。

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[104–106,106]。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[104]。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[105]。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[106]。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[107]。

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。因此也出现了对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[108]。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[109]。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[110]。Yang 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[111]。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[112]。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[113]。

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[114]。Cheng 等人研究了跨项目的克隆代码检测技术[115]。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[116]。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。克隆规避是一种预防性的克隆维护方法，通过阻止克隆代码产生来降低克隆维护的代价。

软件开发人员的复制粘贴活动是导致克隆代码产生的最主要原因。因此在复制粘贴时对新产生的克隆代码进行分析，帮助软件开发人员决定是否规避克隆代码，是目前常见的克隆规避方法。Ali 曾提出一个克隆规避的概念模型[117]，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[118]。

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[119]，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。

2012 年，在面向工业界的克隆管理会议（Software Clone Management Towards Industrial Application）中，研究人员指出克隆管理是未来的重要研究方向，如何将克隆管理与软件开发过程相结合并使之适用于工业界是目前克隆研究领域急需解决的问题[120]。目前，人们对克隆管理的研究仍处于起步和理论研究阶段，罕有从管理的角度维护克隆代码的研究报道。Koschke 将克隆管理划分为预防性、补偿性和改正性三种类型[119]。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。然而，Koschke 对克隆管理的这种划分方法的理论意义大于实际意义。目前克隆管理研究主要集中于补偿性克隆管理，而改正性克隆管理尚未具体应用到实际的克隆管理中。Roy 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。

克隆管理需要跟踪系统中的克隆代码，包括克隆代码的产生以及变化。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[121]、CnP[122]、CPC[123]、CReN[124]、CSeR[125] 等可以在软件集成开发环境中跟踪克隆代码的产生。另一方面，在软件开发过程中克隆代码可能随时发生变化，因此还要跟踪克隆代码的变化。Duala 使用克隆区域描述符生成克隆代码的上下文信息，然后根据这些上下文信息可以实时跟踪克隆代码的变化[126]。该方法仅通过实验验证可以跟踪克隆代码的变化，但是并没有实现对克隆代码的管理，也没有集成到软件开发环境中对克隆代码的边开发、边管理。

在跟踪克隆代码的基础上，也要对克隆代码的变化进行维护和管理。Yamanaka 等人将克隆变化与事件通知机制相结合，把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[127]。Cheng 等人提出了一个基于 Token 的克隆代码一致性维护方法，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[128]。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[129]。此外，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[130]。该方法通过分析历史版本中克隆代码的变化情况，提取相关的特征进行变化预测，以便在软件开发过程中提醒开发人员对有可能发生变化的克隆代码进行维护。克隆变化管理的关键不仅需要实时跟踪克隆代码的变化，更重要的是对发生变化的克隆代码进行及时的维护。

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[131]。Nguyen 实现了一种可用于管理克隆代码的 eclipse 插件 JSync[132]。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。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。克隆关系管理是指在软件开发过程中检测系统中的克隆代码，并对具有克隆关系的代码进行管理。一致性维护管理是指识别变化的克隆代码以及变化的克隆代码是否会导致不一致性缺陷，以便开发人员对克隆代码进行一致性维护。而另一个较早提出的 eclipse 插件 CeDAR[133,134]，虽然没有强调是一种克隆管理工具，但事实上也具有一定的克隆管理功能。CeDAR 将克隆检测和克隆重构融为一体，对现有克隆检测工具的克隆检测结果进行可重构性分析，寻找潜在的可重构的克隆代码，然后在软件开发过程中实现对克隆代码的重构。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，也具有了克隆管理的功能[135]。

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。因此，对克隆代码的分析和维护研究是解决克隆代码问题的最主要途径。

（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。最后，目前也缺乏对克隆代码一致性变化的研究，无法指导对克隆代码的维护过程。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，尤其是克隆代码稳定性与克隆代码变化相关的演化特征。

（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，也更容易导致克隆相关缺陷。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

针对软件中存在的大量克隆代码、以及克隆代码演化过程中的一致性变化问题，本文结合软件演化、程序分析和机器学习方法，研究基于软件演化的克隆代码

分析与一致性维护方法，不仅可以帮助软件开发人员理解克隆代码，还可以帮助降低克隆代码的维护代价以及避免克隆代码一致性违背缺陷，从而达到提高软件质量、可理解性和可维护性的目的。本文的主要研究内容可以简述如下：

（1）针对演化中的克隆代码难于理解和分析的问题，将克隆代码在其演化过程中所表现出的特征称之为克隆代码演化特征，研究基于聚类的克隆代码演化特征分析方法。通过提取相应的属性值表示克隆代码及其演化过程，并使用聚类方法挖据克隆代码的演化特征，帮助软件开发人员理解克隆代码，验证了克隆代码一致性维护的必要性，为后续克隆代码一致性维护需求预测研究奠定了基础。

（2）针对新创建的克隆代码其演化中的一致性变化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，将其在演化过程中是否会发生一致性变化称为克隆创建一致性维护需求。通过提取代码属性和上下文属性表示所创建的克隆代码，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）针对克隆代码演过中的一致性变化容易导致克隆代码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，并将该变化是否会引发克隆代码的一致性变化称为克隆代码变化一致性维护需求。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的一致性违背缺陷。

（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。由于软件开发的流程和规范大同小异，且不同软件系统的克隆实例也具有较强的关联性，使用跨项目的数据进行克隆代码一致性维护需求预测。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。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。

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，综合考虑了软件演化过程、程序分析方法和机器学习方法，不仅可以帮助软件开发人员分析和理解系统中存

在的克隆代码，还可以帮助维护系统中的克隆代码。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，现给出论文的主要研究内容及其关系示意图，如图 1-4 所示。

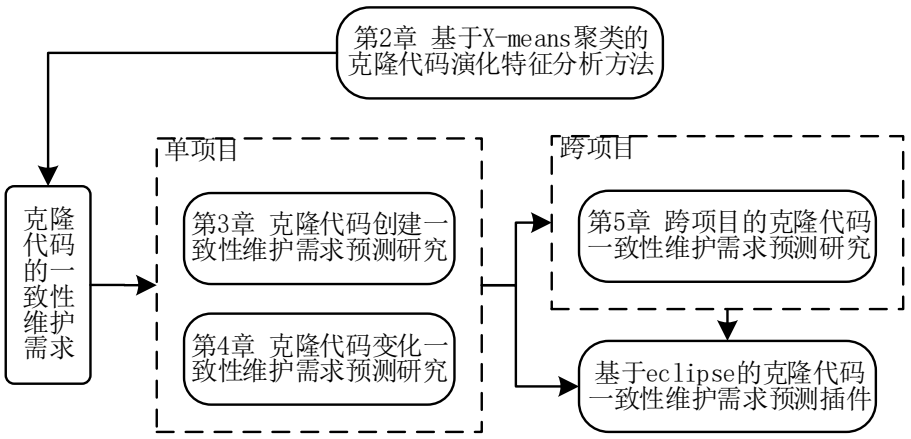


图 1-4 论文主要研究内容及其关系示意图

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

本文主要研究四个内容，第 2 章针对演化中的克隆代码难以理解和分析的问题，提出了一种基于 X-means 聚类的克隆代码演化特征分析方法，通过获取克隆代码演化特征指导克隆代码一致性维护需求预测研究。经分析发现：软件中存在相当数量发生变化的克隆代码，其中大部分变化是一致性变化。克隆代码所发生的一致性变化不仅会导致额外的维护代价，并有可能引发克隆代码的一致性违背缺陷。本文第 2 章研究验证了克隆代码一致性维护的必要性，即在软件开发过程中需要对克隆代码进行一致性维护，为后面第 3、4、5 章的克隆代码一致性维护需求预测研究奠定了基础。

针对克隆代码的一致性维护需求问题，即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）和克隆代码变化时（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和避免一致性违背缺陷。并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。结合软件开过程基于 eclipse 实现了一个克隆代码一致性维护需求预测插件，可以帮助软件开发人员同步的开发和维护克隆代码。

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，为分析克隆代

码演化规律提供了有效的手段，验证了克隆代码一致性维护的必要性，并为后续克隆一致性维护需求预测研究奠定了基础。首先，使用克隆检测工具检测系统中的克隆代码，并构建系统所有克隆代码的克隆家系，用于描述克隆代码的演化过程。然后，从克隆片段、克隆组和克隆家系三个不同角度提取相应的度量值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、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，从规避克隆代码产生的角度降低克隆代码的维护代价。通过定义克隆代码创建实例及其一致性维护需求，将问题转化为可以使用机器学习方法解决的分类问题。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，实现克隆代码的同步开发与维护。通过定义克隆代码变化实例及其一致性维护需求，将问题转化为可以使用机器学习方法解决的分类问题。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 插件帮助实现同步开发和维护克隆代码的一致性。

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，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，本章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，并验证了克隆代码的一致性维护的必要性。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，大部分是发生一致性变化的克隆代码。因此，建议程序开发人员关注发生变化的克隆代码并对其进行一致性维护，为本文后续的克隆代码一致性维护需求预测研究奠定了基础。

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 等人提出了克隆家系模型[16]，并引发了对克隆代码演化以及演化规律的研究，例如克隆寿命、克隆稳定性与一致性变化等研究 (如表 1-2 所示)。

在克隆代码随着软件演化的过程中，克隆代码会长时间的存在于系统中。研究

人员通过对比克隆和非克隆代码，发现克隆代码比非克隆代码的寿命更长[52]，且克隆代码的存在时间往往都会超过一年[53]。Kim 也研究发现克隆代码要比非克隆代码寿命更长，并且克隆代码的变化会使得其寿命变短[16,51]。

系统中的克隆代码会被程序人员修改而发生变化，从而引发了对克隆代码稳定性的讨论。其中一个普遍的观点是克隆代码是稳定的[18,55–57]，其存在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，在克隆代码是否稳定这个问题上还存在一定分歧。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，认为 Type-1、Type-2 克隆是不稳定的，Type-3 克隆是稳定的[19,58]。另外，也有研究人员认为克隆代码的变化规律与具体的软件系统相关[54]。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。同时，克隆代码的稳定性与其演化过程息息相关，如何将克隆代码的稳定性分析与克隆代码演化过程相结合，分析并提取其变化规律是一个难点问题。

另一方面，克隆代码的一致性变化也引起了人们的关注，因为克隆代码一致性变化不仅会导致额外的软件维护代价，遗忘一致性变化还会导致克隆代码的一致性违背缺陷[7,9]。研究人员发现软件系统中有较多的发生一致性变化的克隆代码[21,22]，且发生一致性变化的克隆代码占克隆代码的比例很小[18]。但是，究竟有多少克隆代码发生了一致性变化，并且一致性变化是否更容易发生也尚未可知。结合克隆代码演化过程，克隆代码的一致性变化规律也需要进一步的研究。这不仅可以帮助对软件进行维护代价分析，还可帮助程序开发人员避免克隆代码的一致性违背缺陷。

目前克隆代码演化分析研究仍然不能令人满意，存在以下不足之处：

（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）在现有的克隆代码演化分析研究中，大多数研究带有较强的主观性，甚至得出了相悖的结论，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。

（3）在现有的克隆演化分析研究中，缺乏对克隆代码一致性变化的研究，无法指导程序开发人员对发生变化的克隆代码进行必要的维护。

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，并且如何结合克隆代码演化过程对其分析是一个难点问题。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

克隆代码演化特征是指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；克隆代码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。

本章提出了一种探索和分析克隆代码演化特征的方法，拟通过聚类的方法分析和提取克隆演化特征。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。为了使用机器学习方法，将克隆代码及其演化情况表示成为特征向量，从三个不同的角度分析克隆代码以及演化情况，即克隆片段、克隆组和克隆家系（定义描述见本章第 2.3 节）。

本章所提出的基于聚类的克隆代码演化特征分析方法如图 2-1 所示。从图中可以看出，该方法可以划分为三个步骤，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。

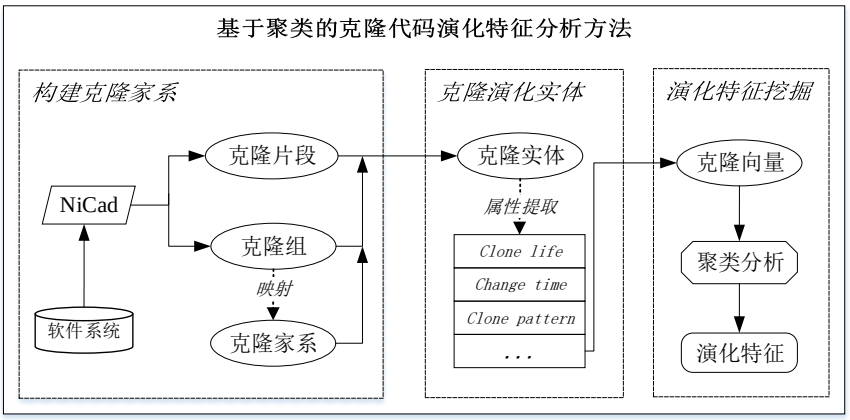


图 2-1 基于聚类的克隆演化特征提取方法

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

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。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[136] 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和提取。因此，本章采用 X-means 聚类方法，无需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。

综上所述，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 克隆代码及演化过程的定义

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（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 2.1 (克隆片段) 克隆片段 (Clone Fragment, CF) 是彼此相似的代码片段，包括若干行连续的代码。给定两个代码片段 CF_1 和 CF_2，如果 CF_1 和 CF_2 满足：

$$\mathit{TextSim}(CF_1, CF_2) > \mathit{sim\_th} \tag{2-1}$$

则称代码片段 CF_1 和 CF_2 为克隆代码（CFs），其中 *TextSim* 为相似性度量，*sim_th* 为相似性阈值。

本文使用 NiCad[13] 检测系统中存在的克隆代码，因此 *TextSim* 采用 NiCad 中所的定义方法。在 NiCad 中，克隆代码相似性定义为 $\mathit{TextSim}(CF_1, CF_2) = 1 - \mathit{UPI}(CF_1, CF_2)$。其中，UPI（Unique Percentage Items）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 $\mathit{UPI}(CF_1, CF_2) = 0.3$，表示两者的差异程度为 30%。同时，其相似度可根据 UPI 计算：$\mathit{SimText}(CF_1, CF_2) = 1 - \mathit{UPI} = 0.7$，表示两者的相似度为 70%。本文将克隆代码的相似性阈值设置为 0.7，当两个代码片段相似性大于 0.7 时，才被认定为是克隆代码。与此同时，NiCad 在检测克隆代码时会设定代码片段的长度范围为 10 - 2500 行，即代码行数在此范围内的代码片段会被检测为克隆代码。

定义 2.2 (克隆组) 克隆组 (Clone Group, CG) 是彼此相似的克隆代码片段的集合，包含若干个彼此相似的克隆片段。假定 CF_i 和 CF_j 是一个克隆组 CG 中的任意两个克隆代码片段，则 CF_i 和 CF_j 满足

$$\mathit{TextSim}(CF_i, CF_j) > \mathit{sim\_th} \tag{2-2}$$

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代码。

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，所使用的克隆家系可以描述如下：

定义 2.3 (克隆家系) 克隆家系 (Clone Genealogy, CGE) 是一个有向无环图，描述了一个克隆组 (CG) 随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也会发生相应的变化。为了描述相邻版本之间的克隆组的变化情况，可以使用“克隆演化模式”进行描述。一个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 2.4 (克隆演化模式) 克隆演化模式 (Clone Evolution Pattern, CEP) 是某一克隆组 (CG) 在相邻两个软件版本之间的演化情况。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相邻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。克隆组 CG_{i-1} 和 CG_i 是相邻两个版本的同一克隆组，克隆组内的克隆片段是一一对应的，且对组内任意的一个克隆片段 CF_{i-1} 和 CF_i，满足

$$\mathit{TextSim}(CF_{i-1}, CF_i) = 1 \tag{2-3}$$

• 相同模式 (Same Pattern)：相同模式表示在两个相邻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）。克隆组 CG_{i-1} 和 CG_i 是相邻两个版本的同一克隆组，且克隆组 CG_{i-1} 和 CG_i 中的克隆片段数量分别为 N_{i-1} 和 N_i，满足 $N_{i-1} = N_i$。

• 增加模式 (Add Pattern)：增加模式表示在两个相邻版本的演化中该克隆组内克隆片段增加。克隆组 CG_{i-1} 和 CG_i 是相邻两个版本的同一克隆组，克隆组 CG_{i-1} 和 CG_i 中的克隆片段数量分别为 N_{i-1} 和 N_i，且克隆组映射的克隆片段的数量为 N，增加模式满足 $N < N_i$。

• 减少模式 (Subtract Pattern)：减少模式表示在两个相邻版本的演化中该克隆组内的克隆片段减少。克隆组 CG_{i-1} 和 CG_i 是相邻两个版本的同一克隆组，克隆组 CG_{i-1} 和 CG_i 中的克隆片段数量分别为 N_{i-1} 和 N_i，且克隆组映射的克隆片段的数量为 N，满足 $N_{i-1} > N$。

• 一致性变化模式 (Consistent Change Pattern)：一致性变化模式表示在两个相邻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。在从版本 V_{i-1} 演化到 V_i 的过程中，克隆组 CG_i 中的全部克隆片段发生了相同的变化（数量为 n），即由 CF_{i-1}^j 变为 CF_i^j（$\forall j \in \{n\}$）。如果对于接近于 0 的阈值 τ，克隆片段 CF_i^j 的变化满足以下条件，称此克隆组 CG_i 发生了一致性变化模式，

$$\begin{aligned} &\textit{TextSim}(CF_{i-1}^{j}, CF_{i}^{j}) < 1, && \forall j \in \{n\}\ (1) \\ &|\ \textit{TextSim}(CF_{i-1}^{p}, CF_{i}^{p}) - \textit{TextSim}(CF_{i-1}^{q}, CF_{i}^{q})\ | < \tau, && \forall p, q \in \{n\}\ (2) \end{aligned} \tag{2-4}$$

• 不一致性变化模式 (Inconsistent Change Pattern)：不一致性变化模式表示在两个相邻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。CG_i 的克隆片段发生了变化，但是不满足一致性变化模式的条件，称此克隆组 CG_i 发生了不一致性变化模式。

• 分裂模式 (Split Pattern)：分裂模式表示在两个连续版本的演化中克隆组内克隆片段发生了变化，并且导致在下一个版本中该克隆组分裂成为两个不同的克隆组。版本 V_i 中存在两个不同的克隆组 CG_i^p 和 CG_i^q 均是由上一版本 V_{i-1} 的克隆组 CG_{i-1} 演化而来，对克隆组 CG_i^p 和 CG_i^q 中任意的 CF_i^p 和 CF_i^q 满足：

$$\textit{TextSim}(CF_i^p, CF_i^q) < \textit{sim\_th} \tag{2-5}$$

则称版本 V_i 中的克隆组 CG_i 发生了分裂模式。

为了更为形象的描述克隆家系及其演化模式，本文给出一个克隆家系的示意图，如图 2-2 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，图中的节点表示一个克隆组 CG，图中边表示该克隆组在两个连续版本之间的演化关系 (CG_{i-1}, CG_i)。同时，该克隆组的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。

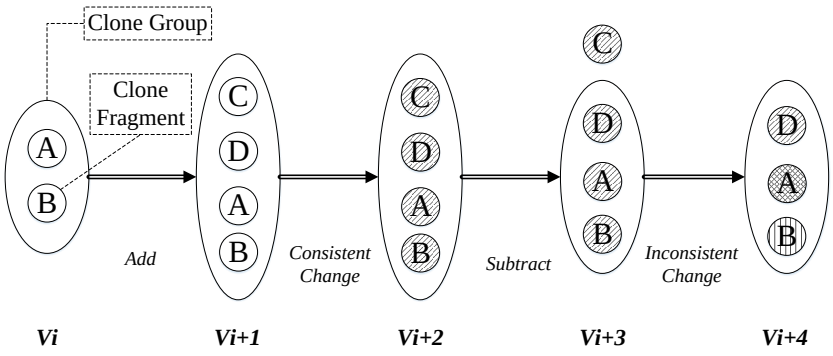


图 2-2 克隆家系示意图[16]

Fig.2-2 An example for clone genealogy[16]

图 2-2 也描述了一个克隆组 CG 在五个连续版本 (V_i, V_{i+4}) 中的演化过程。从版本 V_i 到 V_{i+1}，克隆组 CG 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）。从版本 V_{i+1} 到 V_{i+2} 时，克隆组 CG 中全部的克隆代码片段发生了一致地变化，其克隆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} 时，克隆组 CG 中减少了一个克隆代码片段，其演化模式是减少模式（Subtract Pattern）。从版本 V_{i+3} 到 V_{i+4} 时，克隆组中的克隆代

码片段发生了不一致地变化，因此克隆演化模式为不一致性变化模式（Inconsistent Change 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[137,138]，能够以较高的准确度和召回率系统中的 Type-1、Type-2 和 Type-3 克隆代码[1]。NiCad 可以从两个不同粒度检测系统中的克隆代码：函数粒度和块粒度。因为块粒度具有更为通用的检测效果（函数克隆也是是块克隆代码），本文使用块粒度检测系克隆代码。根据 NiCad 的默认配置，其将克隆代码之间的相似度阈值设置为 70%，将相似度高于此阈值的代码片段报告为克隆代码。

NiCad 将检测到的克隆代码保存于 XML 文件中，并使用“Filename + Start/End Line No.”标记克隆克隆代码，并将彼此相似的克隆片段保存在同一克隆组内。由于 NiCad 仅仅使用代码行表示克隆代码，不仅无法描述克隆代码的语法和语义信息，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[139]。同

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

<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

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 V_i 和版本 V_{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。

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[139]。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。

与此同时，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 - 3 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 4 - 6 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 7 - 10 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 11 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 $O(m*n)$。

算法 2-1 克隆家系构建与演化模式识别算法

Algo. 2-1 Algorithm for building CGEs and identifying CEPs

Input: All source code and code clones from 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 **for** i=*1 to* $N-1$ **do**
5 Mapping all the CFs and CGs between adjacent versions i and $i+1$;
6 Saving all mapping results in mapping_i files;
7 **for** i=*1 to* $N-1$ **do**
8 **for** *each mapping* CGs *in version* i *and* $i+1$ **do**
9 Identify all clone patterns CEPs in thus two versions according to Definition 2.4 ;
10 Saving all mapping results in mapping_i files;
11 Generating all clone genealogies $CGEs$ through traversing all mapping files;
12 **return** $CGEs$;

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)。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片段寿命。假定起始版本为 V_s，克隆寿命为 $i-s$。

• 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。如果 $\mathit{TextSim}(CF_{i-1}, CF_i) < 1$，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。假定克隆片段的初始版本为 V_s，变化次数为：$\sum_{j=s}^{i} ischanged_j$

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在相邻两个版本之间的变化情况。

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

所有的版本数量称之为克隆组寿命。

• 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)，若发生取值为 1，反之为 0。

• 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)，若发生取值为 1，反之为 0。

• 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)，若发生取值为 1，反之为 0。

• 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)，若发生取值为 1，反之为 0。

• 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)，若发生取值为 1，反之为 0。

• 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），若发生取值为 1，反之为 0。

• 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)，若发生取值为 1，反之为 0。

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，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

为分析克隆代码及其演化过程，使用不同的属性特征分别描述克隆片段实体、克隆组实体和克隆家系实体，并采用 X-means[136] 聚类方法挖掘和分析克隆代码演化特征。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和提取。X-means 聚类方法是 K-means 聚类方法的改进方法，在使用其聚类时无需指定聚类数量，方法会自动地选择出最佳的聚类数量。本节将使用 WEKA（“Waikato Environment for Knowledge Analysis”[140]）中提供的 X-means 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：

$Vector$ = (v_1，v_2，...，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，...，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，还行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，...，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 是由新西兰怀卡托大学开发的机器学习工具，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的演化特征。

本文使用的聚类方法是 X-means 聚类[136]，X-means 聚类是 K-means 聚类[141]的改进方法。K-means 方法可以将向量空间 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，对于克隆代码实体的聚类而言，由于缺少必要的基本信息，难以确定所需的 Cluster 的数目。同时，人为的指定聚类的个数也会引入主观的因素。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 聚类会自动的搜索最佳的 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[136]。

[2]WEKA 网站为：http://www.cs.waikato.ac.nz/ml/weka/。

给定一个克隆实体的克隆向量空间 $X=(x_1，x_2，...，x_n)$，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 $C=(c_1，c_2，...，c_k)$，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，聚类算法会自动的搜索最佳的 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。

算法 2-2 克隆演化特征聚类算法

Algo. 2-2 Algorithm of clustering for clone characteristics

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

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

1 Collecting all entities of clone genealogy with number N_cge;
2 Collecting all entities of clone group with numberN_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 **for** *i=1 to* N_cg **do**
9 Generating the vector for CG_i;
10 Appending the vector V_i to X_{CG};
11 **for** *i=1 to* N_cf **do**
12 Generating the vector V_i for CF_i;
13 Appending the vector V_i to X_{CF};
14 Calling WEKA to clustering X_{CF}, X_{CG} and X_{CGE} with X-means method;
15 Mining clone evolutionary characteristic with the clusters of X_{CF}, X_{CG} and X_{CGE};
16 **return** *All clusters for clone fragment, group, and genealogy;*

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 - 7 行、8 - 10 行、11 - 13 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 14 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 $O(n*m)$。

2.7 实验结果与分析

2.7.1 实验设置

本章选择了两个 Java 开源软件作为实验系统：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量。

表 2-1 两个开源软件实验系统信息

Table2-1 The information for 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：

- 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，本章将克隆代码演化特征的挖掘分成两个子任务。第一个子任务，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。第二个子任务，使用 WEKA 对克隆实体进行聚类分析，根据聚类结果挖掘和分析克隆代码的演化特征。因此，均使用两种方法分析和挖掘克隆代码的演化特征：统计分析方法和聚类分析方法。第一种统计分析每种克隆实体的属性值分布情况，帮助发现一些最基本的分布特征。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。本章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

对克隆片段的“历史变化次数”进行统计分析，分析结果如表 2-2（ArgoUML 系统）和表 2-3（jEdit 系统）所示。表中详细给出了克隆代码片段的变化情况，其中数字“0”表示未发生变化，数字“N”表示了截止到当前版本克隆片段发生了 N 次变化。同时，表中第三行给出了没有发生变化和发生变化的克隆片段的数量和比例。

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(95.69%)	1095(4.31%)		

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（其比例为 95.69%），在 jEdit 中为 5885 个（比例为 88.68%）。同时，只有仅仅一小部分克隆片段在演化

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(88.68%)	751(11.32%)						

过程中发生了变化，ArgoUML 为 1095 个（比例为 4.31%），jEdit 为 751 个（比例为 11.32%）。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，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过变化（比例分别为 95.69% 和 88.68%）。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（发生多次变化）。

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 Cluster：Cluster0 - Cluster3。根据其聚类结果分别统计了四个 Cluster 中克隆片段实体的属性特征信息，即克隆寿命（Clone Life）、是否发生变化（Ischanged）和变化次数（Change Time）。对上述每一种属性特征，表中使用平均值（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。

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	Clone Life			Ischanged			Change Times		
	(比例)	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

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

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

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 中 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）。

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，克隆代码在演化中是稳定的，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。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 2 行和第 3 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。

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%

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”、“Subtract”、“Split”、“Consistent/Inconsistent Change”模式。

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

[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。

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：在克隆代码的演化过程中，存在数百的克隆组会发生变化（一致性变化和不一致性变化）。其中，相比于不一致变化，软件系统中的克隆组更容易发生一致性变化。

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 - Cluster3。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。表中第 1 列则给出了每一个 Cluster 中包含的克隆组数量以及比例。

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

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，同时也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 (在两个软件系统中大约 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

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

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性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（刚刚出现在软件系统中），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而新出现在系统中时，程序开发人员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，尽管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。其中，系统中存在数百以上的发生变化的克隆组，其比例接近于 10%。由于克隆代码的一致性变化比不一致性变化更容易发生（一致性变化比例高于不一致性变化）。因此，当开发人员修改克隆组内的某一克隆片段时，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

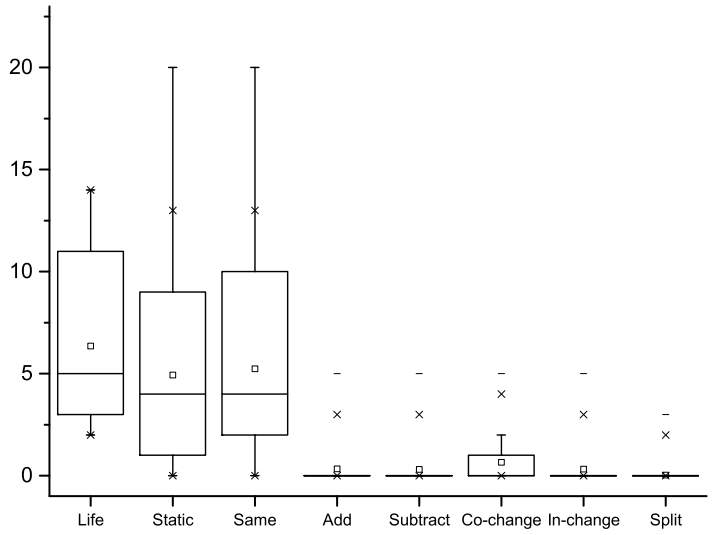
本节克隆家系进行统计分析，分别统计了克隆寿命和七种克隆演化模式数量，其统计结果如图 2-4 所示。图中使用“箱式图”展示，图中的横坐标表示克隆家系的演化情况，即所选用的克隆家系的属性特征，包含克隆寿命（Clone Life）以及 7 种克隆演化模式；图中纵坐标表示克隆家系属性值的数值范围。

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 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。

因此，克隆家系会在软件中存在较长的时间，并且在演化中克隆家系也是较为稳定的。尽管克隆家系不会发生太多的动态演化模式，但是其中也存在一定数量的一致性变化模式。

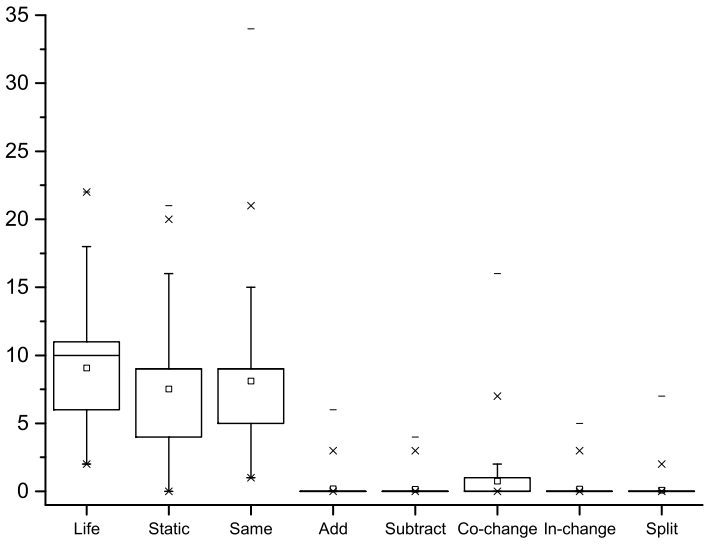
2.7.4.2 克隆家系聚类分析实验

使用 X-means 对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。从表



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

中可以看出，X-means 聚类方法将所有克隆家系可以聚类成 4 个 Cluster：Cluster0 - Cluster3。表中第 1 列给出了四个 Cluster 的基本情况，分别给出了 Cluster 中克隆家系的数量和比例。表中第 4 - 11 列给出了克隆家系每一种属性特征的分布情况，同样使用平均值（Mean）、标准差（SD）和中位数（Median）描述。

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。从表中的“Cluster”列和“Death”列中可以发现，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。

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

为了分析克隆家系的演化情况，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“增加（Add）”，“减少（Subtract）”，“一致/不一致性变化（Consistent/Inconsistent Change）”和“分裂（Split）”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

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

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

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

综上所述，克隆家系在整个演化过程中大多是稳定的，而较短寿命的克隆家系相比寿命较长的克隆家系更为稳定。同时，动态的演化模式通常发生在较长寿命的克隆家系中（即使其数量比较稀少），其中一致性变化模式比不一致性变化模式更

为频繁。因此，本文建议开发人员应该更加注意寿命较长的克隆家系，并且当克隆代码发生变化时需要考虑同组克隆代码的一致性变化。

2.8 本章小结

为挖掘克隆代码及其演化过程的克隆代码演化特征，本章提出了一种基于 X-means 聚类的克隆代码演化特征分析方法，为分析克隆代码演化规律提供了有效的手段。本章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及其演化过程。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。在发生变化的克隆代码中，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维护问题。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，还可以提供一些建议指导开发人员维护系统中克隆代码。本章研究不仅验证了克隆代码一致性维护的必要性，更为本文后续的克隆代码一致性维护需求预测研究奠定了基础。

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，程序开发人员会通过复制和粘贴操作复用软件中的既有代码，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。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，可以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时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，提取代码属性和上下文属性表示该创建的克隆代码。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一致性维护需求，预测结果的 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% 之间。本章所提出的预测方法从规避克隆代码产生的角度降低克隆代码的维护代价，提高了预测的准确率和召回率。

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），由此操作引入了克隆代码。在其随着软件系统演化的第 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，因此被复制的代码片段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:

```
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}
```

Code Fragment 2:

```
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}
```

图 3-1 会导致额外维护代价的克隆代码[142]

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，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。

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

```
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[142]

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

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[90–93]、重构调度[94–96]、重构方法[100–103]等方面（见本文第 1.2.4.1 节克隆代码重构）。由于重构的代价较大，仅有少数的克隆代码适合于重构，因此无法解决克隆代码所导致的额外维护代价问题。

此外，为了有效地管理和维护克隆代码，研究人员也对克隆代码进行跟踪，如跟踪新产生的克隆代码。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[121]、CnP[122]、CPC[123]、CReN[124]、CSeR[125] 等。但是，对新产生的克隆代码，上述方法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）在现有的克隆维护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，对这种操作是否会导致额外的维护代价缺少必要分析和判断。

综上所述，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。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。具体来说，通过规避会导致额外维护代价的克隆代码产生，可以降低克隆代码的额外维护代价；通过允许不会导致额外维护代价的克隆代码的产生，可以提高软件开发效率。

3.2.3 本文的解决思路

为了在克隆代码创建时避免其演化过程中可能导致的额外维护代价，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创建时的一致性变化以及一致性维护需求的定义。定义不仅可以描述克隆代码所导致的维护代价，还将问题转化为一个可以使用机器学习方法解决的分类问题。然后，本文在 Wang[142] 等人的研究基础上，改进了用于描述克隆代码的特征属性。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方法如图 3-3所示。从图中可以看出，该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。

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被复制粘贴的克隆代码，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地复用该克隆代码，从而提高软件开发的效率。

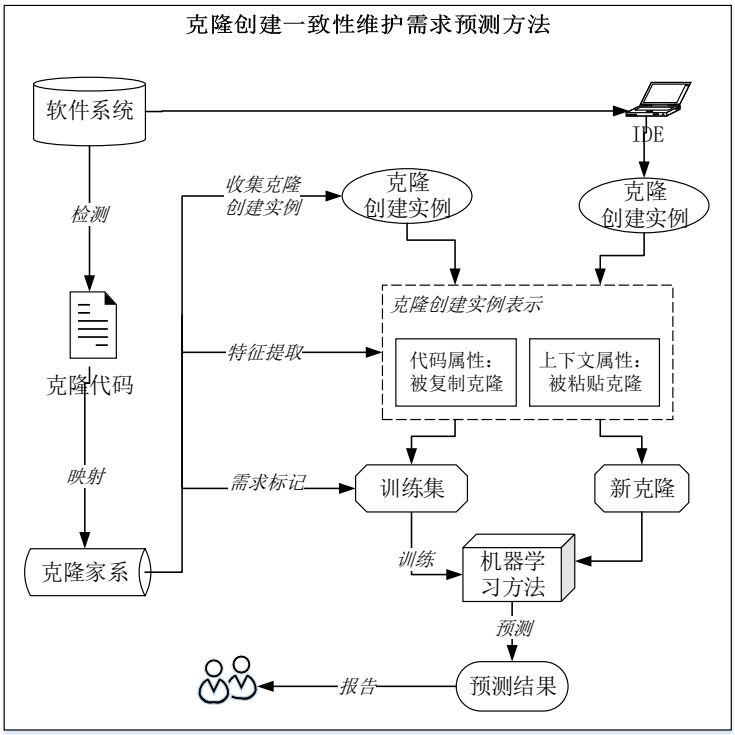


图 3-3 克隆创建一致性维护需求预测方法

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

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 克隆代码创建时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。为了描述克隆代码所发生的变化，本文在第 2 章中给出了克隆代码的一致性变化模式，见定义2.4。但是，第 2 章所定义的克隆代码的一致性变化模式过于严格，无法直接用于描述克隆代码可能引发的额外维护代价。

在第 2 章的定义中，要求克隆组的所有克隆代码片段都要发生相同的变化，才称之为一致性变化。然而，本章的目标是避免克隆代码所导致额外的维护代价，当克隆组内的两个克隆代码同时发生变化时，同样会导致软件的额外维护代价。因此，在向系统中引入新的克隆代码时（克隆代码创建时），本章仅要求至少两个克隆代码片段同时发生变化。

因此，针对克隆代码所导致的额外维护代价问题，将克隆代码的一致性变化模式重新定义如下：

定义 3.1 (克隆创建时一致性变化模式) 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) 被分别修改为 (CF_1', CF_2')。如果克隆代码 CF_1 和 CF_2 的变化满足：

$$\mathit{TextSim}(CF_i, CF_i') < 1, \forall i \in \{1, 2\} \tag{3-1}$$

则称克隆组 CG' 具有创建时一致性变化模式（Creating Consistent Change Pattern）。

在该定义中，克隆代码的变化同样使用 $\mathit{TextSim}(CF_i, CF_i') = 1 - \mathit{UPI}(CF_i, CF_i')$ 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。

本章定义的克隆组的一致性变化模式，仅要求克隆组内至少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严格。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，称为克隆创建实例。因此，给出克隆创建实

例的定义，如下所示：

定义 3.2 (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，且认为该克隆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 3.3 (克隆创建时一致性维护需求) 给定版本 j 中一个克隆创建实例 CG，如果在版本 k 中存在一个克隆组 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“克隆创建时一致性变化模式”（Creating Consistent Change Pattern），则称该克隆创建实例 CG 满足克隆创建一致性维护需求（Creating Consistency-Requirement）；反之，假如不存在满足上述条件的克隆组 CG'，则称该克隆创建实例 CG 不满足克隆一致性维护需求（Creating 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，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 3.2，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。

算法 3-1 给出了收集系统中复制和粘贴操作的步骤。算法中第 1 - 3 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 4 行为根据映射结果生成系统

的克隆家系。第 5 - 8 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 $O(m*n)$。

算法 3-1 复制粘贴操作收集算法

Algo. 3-1 Algorithm for collecting copy-and-pasted operations

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 CFs and CGs between the adjacent versions i and i+1;

4 Generating all clone genealogies $CGEs$ with number N_cge;

5 **for** i=*1 to* N_cge **do**

6 Collecting the clone creating instance from CGE_i according to Definition 3.2 ;

7 Identify all clone consistent change pattern in thus two versions according to Definition 3.1 ;

8 Labeling its consistency-requirement according to Definition 3.3 ; Labeling the copied and pasted code fragments by mapping code clones to the last version;

9 **return** *All copy-and-paste operations;*

（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[139]。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导致的克隆创建实例。因此，根据定义 3.2 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 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所发生的一致性变化。根据定义 3.3，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 3.1 ），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一版本中。根据映射结果，可能会存在两种情况：(a) 其中一个克隆代码可以映射，另一个未映射。认为映射代码为被复制代码，未映射为被粘贴代码。(b) 两者均没有映射。此情况下随机选取一个为被复制代码，另一个未未被粘贴代码。原因是两者互为克隆代码，彼此之间相似，故随机选取一个为被复制代码并不影响预测结果。注意不存在两者均可映射的情况，应在上一版本检测为克隆代码[1]。

3.5 复制粘贴操作的特征描述

在使用机器学习方法进行分类或者预测时，需要提前构建和训练机器学习模型，如本章预测克隆代码创建一致性维护需求。但是，实际的复制粘贴操作所导致的克隆代码片段无法直接应用于机器学习中。因此，本节将使用相应的属性值表示克隆创建实例。克隆创建实例是由复制粘贴操作所引入的克隆代码片段，克隆创建实例包含被复制的克隆代码和被粘贴的克隆代码。将提取不同的属性用于描述被复制和被粘贴的克隆代码。

3.5.1 属性特征分析

从本章第3.2.1节所给出的克隆代码例子中，可以观察到与克隆代码一致性变化有关联的一些因素。首先，如果两个克隆代码存在于同一个文件中，那么它们可能会共享一些相同的局部方法和局部变量。对其中一个克隆代码片段的修改可能会引发相似的变化。其次，克隆代码自身的一些语法信息，可能也会影响克隆代码的一致性变化。例如，在对 if 语句的条件判定中，往往隐含着一些配置信息和静态作用域相关的变量。如果两个克隆代码片段也包含相同的配置信息或者变量，则对此类变量的修改可能也会导致一致性的变化。最后，克隆代码之间以及克隆代码

[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。

与其它模块之间的依赖关系，也可能会对克隆代码的一致性变化产生影响。例如，被粘贴的克隆代码片段只包含了一些库函数调用，该克隆代码片段对其它模块的依赖相对较少。因此该被粘贴的克隆代码片段不太可能受到其它模块变化的影响，也就不太容易引发克隆代码的一致性变化。

在一个克隆创建实例中，包含着两个不同类型的克隆代码片段，即被复制的克隆代码片段以及被粘贴的克隆代码片段。对于被复制的克隆代码片段，程序开发人员往往不会对其进行修改。而对被粘贴的克隆代码片段，程序开发人员可能会对其进行一定程度的修改，从而使其适用于被粘贴克隆代码的所在位置的环境。因此，本章从这两个不同的角度，提取不同的属性特征表示克隆代码创建实例。对于被复制的克隆片段，从克隆代码本身的角度进行分析，提取代码属性对其进行描述。而对于被粘贴的克隆代码，其可能会被开发人员修改，从两个克隆代码片段之间的区别与联系的角度进行分析，即提取上下文属性进行描述。

3.5.2 被复制克隆代码的属性特征

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。因此本章从代码本身的角度去描述被复制克隆代码的特征，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，主要包括克隆代码粒度、Halstead 属性、结构属性、调用属性等属性。

• 克隆粒度：被复制克隆代码的粒度。使用代码行数表示克隆粒度，代码行数是最基本的表示代码规模的度量。代码规模作为基础度量可以帮助预测程序的特性，如可靠性和易维护性等[143]。通过计算克隆代码的位置信息，即可获取克隆粒度。假定该克隆代码在文件中的起始和结束行分别为：L_s 和 L_n，则克隆粒度为：$L_n - L_s$。

• Halstead 属性：被复制克隆代码的代码复杂度 Halstead 度量[144]。是由 Halstead 提出的一种基于程序源代码的度量，该度量可以估计程序复杂程度和程序员编写程序花费时间。分别统计了 Halstead 的四个基本属性值，即该克隆代码片段中的操作符种类、操作数种类、操作符总数和操作数总数。

• 结构属性：被复制克隆代码的语法结构信息。Jiang 等人使用代码的结构信息生成特性向量用于检测克隆代码，并取得了较好的效果[15]。因此，使用克隆代码的结构信息描述克隆代码也是可行的。本文使用的结构信息包含了克隆代码片段中关键语句的统计信息，对 if_then、if_else、switch、while、do、for、this_or_super 等语句进行数量统计。

克隆代码与其它模块的依赖关系也会影响克隆代码的一致性变化。因此，为了帮助分析和识别克隆代码与其它模块的依赖关系，本文也将克隆代码中所包含的函数调用次数作为属性特征，并将其细分为本地函数调用、库函数调用、其它函数调用和总函数调用。

• 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的函数调用。

• 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。

• 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。函数的参数个数也是影响软件质量的因素之一。函数的参数个数和函数调用次数类似，同样也能反应出代码间的依赖程度，因此也会影响克隆片段的一致性维护需求。该度量统计了该克隆代码片段中所有的参数访问次数之和。

3.5.3 被粘贴克隆代码的属性特征

克隆创建实例中被粘贴的克隆代码，在其被粘贴到其所在的位置后，程序开发人员可能会对其进行修改。因此，被复制的和被粘贴的克隆代码不仅存在着克隆关系，还存在着一些区别。为了描述这两个克隆代码之间的克隆关系以及它们之间区别。本章提取上下文属性用于表示被粘贴的克隆代码，描述了克隆创建实例中克隆代码之间的关系属性。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相似度等信息。

• 代码相似度：被复制与被粘贴的克隆代码之间的相似程度。代码片段越相似，它们之间的关系也就越密切，因而在修改时也就越可能需要一致性修改。假设 (CF_1, CF_2) 是两个克隆代码片段，其相似度可如下计算：$\mathit{SimText}(CF_1, CF_2) = 1 - \mathit{UPI}$。

• 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。存在同一个文件中，其可能会共享一些局部变量等信息，也意味着更容易发生一致性变化。若两个克隆代码片段存在于同一个文件中，则取值为 1，反之取值为 0。

• 文件名相似度：被复制和被粘贴克隆代码所在文件的文件名相似度。在一些

项目中，文件名也往往会隐含一些内在的联系，克隆代码所在文件的文件名越相似，则克隆代码之间的依赖关系可能会更紧密。假定两个克隆片段所在的文件名分别为 M_1 和 M_2，则文件名相似度为 $Sim(M_1, M_2)$。其中，*Sim* 使用李氏距离[145,146]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：当两个克隆代码是局部克隆时，其文件名相似度标识为 1，非局部克隆时为 0。该属性用于决定文件名相似度度量是否起效。

• 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相似度，计算方式同文件名相似度。程序员在命名方法时，往往也会采取一定的规则。而方法名相似度则可以帮助获取这种内在的联系。

函数中的参数信息也表明了函数与其它模块的依赖关系，也会影响克隆代码的一致性维护需求。根据程序员的编程习惯，通常他们会将变量命名成具有实际意义的变量，因此参数名的相似度隐含着函数之间的依赖关系。因此，对于包含克隆代码的函数参数，分别计算以下 3 种不同属性特征表示克隆代码的属性特征。

• 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 $(P_1, P_2, \ldots, P_m)$ 和 $(Q_1, Q_2, \ldots, Q_n)$，则总参数名相似度为 $\sum_{i=1}^{m}\sum_{j=1}^{n}(Sim(P_i, Q_j))$。

• 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 $(P_1, P_2, \ldots, P_m)$ 和 $(Q_1, Q_2, \ldots, Q_n)$，最大参数名相似度为 $Max(Sim(P_i, Q_j))$。

• 总参数类型相似度：被复制和被粘贴克隆代码所在方法分别为 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 $(P_1, P_2, \ldots, P_m)$ 和 $(Q_1, Q_2, \ldots, Q_n)$，总参数类型相似度 $\sum_{i=1}^{m}\sum_{j=1}^{n}(Sim(P_i, Q_j))$。

• 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，若相同为 1，反之为 0。

综上所述，本章使用代码属性和上下文属性表示克隆创建实例。给定一个克隆创建实例，使用一个 n 维向量表示该克隆创建实例，$Vector = (v_1, v_2, \ldots, v_m)$，其中 v_i 表示该克隆创建实例的一个特定属性。对于克隆创建实例，本章共提取了共计 28 维属性特征，因此 m 取值为 28。

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，简写为 BN）[147,148]、朴素贝叶斯方法（Naive Bayes，简写为 NB）[149]，支持向量机方法（Support Vector Machine，简称为 SVM）[150]、K 近邻方法（K-Nearest Neighbors，简写为 KNN）[151] 和决策树方法（Decision Tree，简写为 DT）[152]。

（1）贝叶斯网络方法

贝叶斯网络是一种概率图型，使用已经观察到的事件来预测将来可能发生的事件[147]。贝叶斯网络可以表示成一个有向无环图模型，图中的每一个节点表示一个随机事件，图中的边则表示随机事件发生的条件概率。因此，贝叶斯网络中的全部节点可以视为一组随机变量$X_1, X_2, ..., X_n$，贝斯网络的边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。支持向量机方法模型将实例表示为空间中的点，并且试图构造一个超平面将不同类的实例（点）间隔开。更正式地来说，支持向量机在高维或无限维空间中构造超平面或超平面集合，可以用于分类问题中。直观来说，分类边界距离最近的训练数据点越远越好，因为这样可以缩小分类器的泛化误差。

以本文的克隆一致性需求分类为例，每一个克隆代码实例会抽象称为高维空间中的一个“点”，空间维数等同于所提取的属性数量。在使用支持向量机分类克隆实例时，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

K 近邻方法是一种用于分类和回归的非参数统计方法。K 近邻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。K 近邻会推迟对训练数据的建模，直到需要分类样本时才进行。在 K 近邻分类中，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 K 近邻中的一个实例。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。

根据定义 3.3 ，克隆创建实例有两种不同的状态：

• 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，从而导致系统的额外维护代价。

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。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。

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 - 5 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 6 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。

算法 3-2 克隆创建一致性维护需求预测算法

Algo. 3-2 Algorithm for predicting clone creating consistency

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 Calling WEKA to train the model on training set Sets;
7 return The predictive model;
```

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 Java 开源软件进行实验，其中 ArgoUML 和 jEdit 与第 2 章中所使用的系统相同。ArgoUML 是一个领先的开源 UML 建模工具，jEdit 是程序员开发所使用的编辑器。并增加两个新的软件系统，分别为 jFreeChart 和 Tuxguitar。jFreeChart 是一个开源 Java 库，可以帮助软件开发人员显示高质量的图表。Tuxguitar 是一个开源的多轨的吉他指法编辑器，用户可以使用其创建自己的吉他六线谱。

表 3-1 给出了四个实验系统的基本信息情况，表中第 2 列给出了所分析系统的软件版本数量，第 3 和 4 列给出了软件系统的起始版本号和结束版本号。本文选择 14 个版本的 ArgoUML，22 版本的 jEdit，20 个版本的 jFreeChart 和 13 个版本的 Tuxguitar。

表 3-2 给出了实验系统的克隆创建实例信息统计。其中，第 2 列和第 3 列给出了每个系统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。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，因而会增加系统的额外维护代价。

表 3-1 四个开源软件实验系统信息

Table3-1 The information for four open sources experimental projects

实验系统	版本数	Start Version	End Version
ArgoUML	14	0.20.0	0.34.0
jEdit	22	3.0.0	5.0.0
jFreeChart	20	1.0.0	1.0.19
Tuxguitar	13	0.7.0	1.3.2

表 3-2 实验系统的克隆创建实例信息统计

Table3-2 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(77.1%)	766(22.9%)	3340
jEdit	560(88.5%)	73(11.5%)	633
jFreeChart	2013(59.8%)	1353(40.2%)	3366
Tuxguitar	1016(71.1%)	413(28.9%)	1429

从表 3-2 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。这说明通过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需求（比例从 59.8% 到 88.5%）。上述观察到的现象与已有研究的结论相一致[142][18]。这表明通过复制粘贴所引入的克隆代码，在演化过程中大部分不会发生一致性变化，因此开发人员可以采用这种技术提高软件开发效率。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。这警告程序开发人员需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该复用技术。

为了评估本章所提出的克隆创建一致性维护需求预测方法，本章将实验划分为两个部分，有效性验证实验和使用模式实验：

• 有效性验证实验：在此实验中，使用五种不同机器学习方法评估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方法，用于克隆代码创建的一致性维护需求预测。在此实验中，同时预测两种类别的克隆创建实例。

• 使用模式实验：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要一致性维护。本节以贝叶斯网络方法为例，详细给出了两种不同状态实例

的预测结果，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。

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。为了评估所提取的属性特征的有效性，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：

• 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）描述一个机器学习模型的预测能力。混淆矩阵如表 3-3 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 3-3 混淆矩阵示意图

Table3-3 The Confusion Matrix

真实类别	预测类别 0	预测类别 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，本章计算精确率（Precision Rate）、召回率（Recall Rate）和 F 值（F-measure）作为评价模型预测能力的指标。

对于不需要一致性维护的克隆创建实例（类别 0），其精确率指在预测结果为不需要一致性维护的复制粘贴实例中，正确预测的实例与所预测实例的比值。Precision 可计算如下：

$$Precision = \frac{TP}{TP + FP} \tag{3-2}$$

召回率指预测为不需要一致性维护的复制粘贴实例，与系统中的不需要一致性维护的实例的比值。Recall 可如下计算：

$$Recall = \frac{TP}{TP + FN} \tag{3-3}$$

F 值是综合精确率和召回率的评价指标，用于评估预测模型的综合能力。F-measure 可如下计算：

$$F\text{-}measure = 2 \times \frac{precision \times recall}{precision + recall} \tag{3-4}$$

相似地，对于需要一致性维护的克隆创建实例（类别 1），同样采用精确率（Precision Rate）、召回率（Recall Rate）和 F 值（F-measure）作为评价指标，计算方式相同。

在本章的一致性维护预测中，将同时预测两种不同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”，然后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,

$$Ave\text{-}Precision = \frac{P_1 \times N_1 + P_2 \times N_2}{N_1 + N_2} \tag{3-5}$$

• 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。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,

$$Ave\text{-}Recall = \frac{R_1 \times N_1 + R_2 \times N_2}{N_1 + N_2}. \tag{3-6}$$

• 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。首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,

$$\textit{Ave-F-measure} = \frac{F_1 \times N_1 + F_2 \times N_2}{N_1 + N_2}. \tag{3-7}$$

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。本文对相应机器学习方法的参数进行了简单的调整，并获得不错的预测效果。通过进一步的调整机器学习的参数，可以增强所构建机器学习模型的预测能力。

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。朴素贝叶斯方法只有一个父节点，设置阈值为 0.5。在支持向量机方法中，选择用 Polynomial Kernel 作为核函数。决策树方法采用了 J48 算法作为预测方法，并设置置信度为 0.75。对 K 近邻方法，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，实验结果分别如表3-4 和 3-5 所示。

3.7.2.1 全属性实验结果

表 3-4 给出了克隆创建实例使用全部属性组进行预测的实验结果。表中第 1 列和第 2 列分别为所使用的评估指标和五种不同机器学习方法，表中第 3 - 6 列为在四个实验系统上的实验结果。表中第 2 - 6 行为平均精确率，第 7 - 11 行为平均召回率，第 12 - 16 行为平均 F 值。

从 3-4 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维护需求，精确率从 79.3% 到 95.8%，召回率从 79.4% 到 95.8% ，F 值从 79.4% 到 95.7%。具体地，对 ArgoUML 系统的预测效果最好，F 值从 88.7% 到 95.7%，jEdit 和 jFreeChart 系统的预测效果次之，F 值从 84.0 到 90.3%，Tuxguitar 系统预测效果

最差，F 值从 79.4% 到 89.0%。

机器学习方法对于不同的项目具有不同的预测效果。从表 3-4 中可以看出，对 ArgoUML 系统具有最好的预测效果，对 jEdit 和 jFreeChart 的预测效果次之，对 Tuxguitar 的预测效果最差。对于 ArgoUML 和 Tuxguitar 系统来说，这两个系统的数据分布相似，但 ArgoUML 的预测效果要好于 Tuxguitar 系统。分析原因是由于 ArgoUML 系统具有更多的数据，因此所训练的机器学习模型更好。与此同时，jEdit 和 jFreeChart 系统的实验效果相差不大，但是 jEdit 的训练集规模是四个系统中最小的。jEdit 在训练集较小还可以达到不错的预测效果的原因，是由于 jEdit 系统的样本不平衡。由表 3-2 可以看到，jEdit 的数据会向一侧偏移，从而使得预测效果变好。

表 3-4 克隆创建实例一致性维护需求预测效果

Table3-4 The effectiveness of clone creating consistency for all attributes

评估指标	方法	ArgoUML	jEdit	jFreeChart	Tuxguitar
平均 Precision(%)	BN	93.5	88.9	88.3	83.1
	NB	88.6	87.1	86.9	79.3
	SVM	95.8	92.4	90.6	88.8
	KNN	94.0	88.6	90.0	84.8
	DT	93.9	89.8	89.3	88.9
平均 Recall(%)	BN	93.6	87.7	88.2	83.6
	NB	88.8	82.0	86.8	79.4
	SVM	95.8	92.1	90.4	88.3
	KNN	94.0	88.9	90.0	84.8
	DT	94.0	90.5	89.2	89.1
平均 F-measure(%)	BN	93.5	88.2	88.1	83.2
	NB	88.7	84.0	86.7	79.4
	SVM	95.7	90.3	90.3	87.6
	KNN	94.0	88.7	90.0	84.8
	DT	93.9	90.1	89.2	89.0

对比不同机器学习方法的有效性，通过对比发现支持向量机在这四个实验系统上具有最佳的预测能力。具体来说，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，支持向量机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，但是相差并不是特别明显。因此，建议开发人员在需

要预测克隆创建实例时优先考虑支持向量机方法，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。分别使用代码属性和上下文属性进行预测，实验结果如表 3-5 所示。表中第 1 列和第 2 列分别为实验评估指标和实验系统，表中第 3 列给出了不同系统使用不同属性组进行的实验。其中，“全部属性”表示使用全部属性组的预测结果，“代码属性”和“上下文属性”分别为仅使用一组属性的实验结果。表中第 4 - 8 列为五种不同的机器学习方法。

从表中可以看出，所提取的属性特征对克隆代码创建时的一致性维护需求具有积极的作用。首先，属性组实验结果和全部属性组实验结果并没有显著的差异。这表明本章所提取的属性特征不会对克隆创建实例的一致性维护需求预测产生负面影响。其次，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文属性时。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。因此提取的属性特征对预测效果具有积极的作用。

不同的属性特征对不同系统的预测效果具有不同的作用。对于 ArgoUML 和 jEdit 系统来说，使用全部属性的预测效果最好。代码属性和上下文属性在预测中均起到了积极的作用。而对系统 jFreeChart 和 Tuxguitar 来讲，仅使用上下文属性的预测效果最好，这说明上下文属性对这两个系统起到了更为关键的作用。但仅使用代码属性和使用全部属性的预测效果，与其预测的效果相差不大，也具有积极的作用。在预测克隆代码的一致性维护需求时，本文建议保留全部的属性进行预测，因为属性特征可能对未经验证的系统具有积极的作用。

同样对比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，但支持向量机方法具有相对最好的预测能力。因此建议程序开发人员优先选择支持向量机方法，并在预测时保留所有的属性特征进行预测。

3.7.3 使用模式实验

本节以贝叶斯网络方法为例，分别预测两种不同类别的克隆创建实例，即不满足一致性维护需求的实例（一致性维护自由实验）和满足一致性维护需求的实例

表 3-5 克隆创建实例的属性组预测效果

Table3-5 The effectiveness of clone creating consistency for attribute sets

评估指标	实验系统	实验组	BN	NB	SVM	KNN	DT
平均 Precision(%)	ArgoUML	全部属性	93.5	88.6	95.8	94.0	93.9
		代码属性	91.2	86.6	92.8	91.1	90.4
		上下文属性	91.7	87.7	94.2	93.3	93.0
	jEdit	全部属性	88.9	87.1	92.4	88.6	89.8
		代码属性	88.5	86.5	92.4	90.3	88.2
		上下文属性	83.0	83.2	90.9	87.7	87.6
	jFreeChart	全部属性	88.3	86.9	90.6	90.0	89.3
		代码属性	80.8	75.5	81.9	80.8	80.2
		上下文属性	90.3	87.3	90.6	89.8	89.1
	Tuxguitar	全部属性	83.1	79.3	88.8	84.8	88.9
		代码属性	81.1	74.7	83.4	80.6	80.0
		上下文属性	84.3	82.4	87.3	86.1	88.1
平均 Recall(%)	ArgoUML	全部属性	93.6	88.8	95.8	94.0	94.0
		代码属性	91.4	86.8	92.9	91.2	90.5
		上下文属性	91.9	88.0	94.3	93.4	93.1
	jEdit	全部属性	87.7	82.0	92.1	88.9	90.5
		代码属性	86.9	83.6	92.1	91.2	89.6
		上下文属性	85.2	82.6	91.8	88.5	88.9
	jFreeChart	全部属性	88.2	86.8	90.4	90.0	89.2
		代码属性	80.3	75.2	80.6	80.3	79.6
		上下文属性	90.3	87.3	90.4	89.8	89.0
	Tuxguitar	全部属性	83.6	79.4	88.3	84.8	89.1
		代码属性	81.7	75.6	83.7	81.0	80.7
		上下文属性	84.6	81.6	87.4	86.2	88.2
平均 F-measure(%)	ArgoUML	全部属性	93.5	88.7	95.7	94	93.9
		代码属性	91.2	86.7	92.7	91.2	90.4
		上下文属性	91.7	87.8	94.1	93.3	93.0
	jEdit	全部属性	88.2	84.0	90.3	88.7	90.1
		代码属性	87.6	84.8	90.3	90.5	88.7
		上下文属性	84.0	82.9	90.7	88.1	88.1
	jFreeChart	全部属性	88.1	86.7	90.3	90.0	89.2
		代码属性	79.7	74.1	79.7	79.6	78.8
		上下文属性	90.2	87.2	90.3	89.7	88.9
	Tuxguitar	全部属性	83.2	79.4	87.6	84.8	89.0
		代码属性	81.1	75.0	82.7	80.7	80.2
		上下文属性	84.4	81.9	86.9	86.2	88.1

（一致性维护实验）。在构建贝叶斯网络时，使用 K2 算法建立网络结构，并设置贝叶斯网络的最大父节点个数为 3，同时使用 SimpleEstimator 计算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，并评估预测模型的预测能力。

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），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），当预测值大于等于该阈值时认定该实例需要一致性维护。

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，帮助程序开发人员执行复制和粘贴操作，从而快速安全地进行软件开发。使用三个指标评估预测效果，分别为：推荐率、精确率和召回率。其中，推荐率（Recommendation Rate，RR）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

全属性实验同样使用全部属性在四个实验系统上进行预测，实验结果如图 3-4 所示。图中的横坐标表示阈值，设置阈值从 0.01 到 0.50，图中的纵坐标为评价指标的数值大小。

由图 3-4 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 86.2% - 97.2% 之间，同时召回率也达到较高值，介于 77.0% - 97.4% 之间。其中，ArgoUML 系统的预测效果最好，jEdit 和 jFreeChart 系统的预测效果次之，Tuxguitar 的预测效果相对最差，但也达到了一个较高的预测效果。ArgoUML 之所以有最好的预测效果是因为其拥有最多的克隆创建实例（2574 个）。jEdit 与 jFreeChart 的预测效果相差不大，但 jEdit 仅有 560 个不需要维护的创建实例，而 jFreeChart 有超过 2000 个实例。因此，预测的效果与训练集规模大小和训练集样本分布（不平衡性）均有关系。在预测克隆代码一致性维护需求时，应该尽量扩大训练集的规模，并且选择向数据较多的一类实例进行预测，从而达到最佳的预测效果。

由图 3-4 和表 3-2 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一致。同时，阈值会对预测结果产生一定

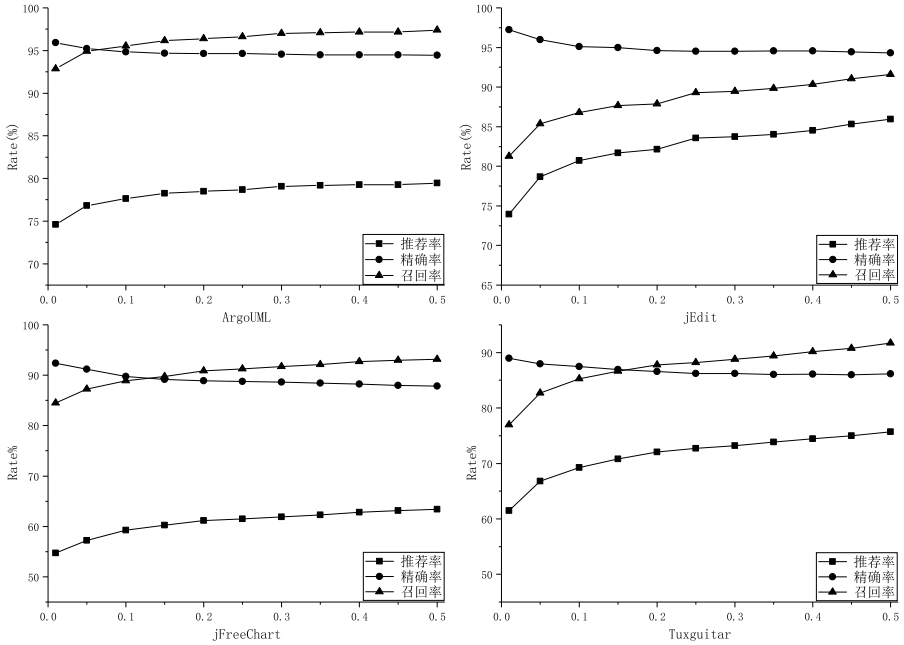


图 3-4 全属性组维护自由实验效果

Fig.3-4 The effectiveness for all attributes of clone consistency free

的影响，其中准确率会随着阈值的升高而缓慢降低，召回率会随着阈值的升高而缓慢增大。但是，阈值所产生的影响不大，四个系统在不同的阈值的准确率和召回率均达到了较高水平。这说明本章的方法以较好的稳定性预测满足一致性维护需求自由的克隆创建实例。因此，在使用全部属性预测不需要一致性维护的克隆创建实例时，可以达到一个较好地预测效果。

（2）属性组实验

针对不同的属性组进行了属性组实验，实验结果如图 3-5 所示。其中，使用“RR”表示推荐率，“P”表示精确率，“R”表示召回率。并同时使用不同的后缀表示不同属性组的实验结果，其中“All”为全属性组实验结果，“Code”表示仅使用代码属性的实验结果，“Context”表示上下文属性的实验结果。

由图 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，因此本章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

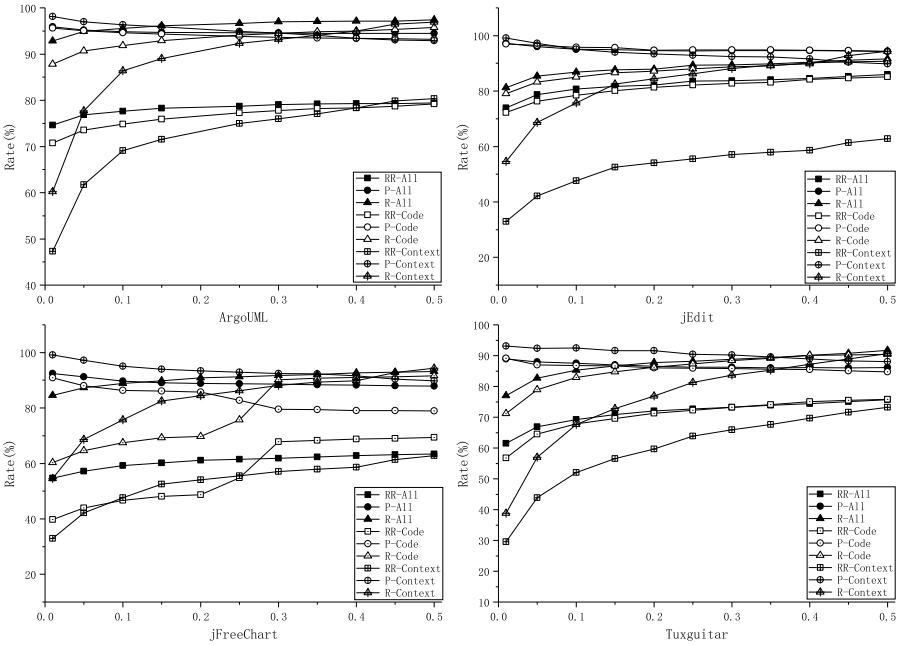


图 3-5 属性组维护自由实验效果

Fig.3-5 The effectiveness for attribute set of clone consistency free

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作用。

3.7.3.2 一致性维护实验

本节对需要一致性维护的克隆创建实例进行预测评估。将关注需要一致性维护的克隆创建实例，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，WR）指所警告的需要一致性维护的克隆创建实例，即预测为需要一致性维护的实例与系统中全部实例的比值。

（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，实验结果如图 3-6所示。图中的横坐标表示阈值，设置阈值从 0.50 到 1.0，图中的纵坐标为评价指标的数值。

由图中可以看出，本章方法在预测需要一致性维护的克隆创建实例时，也取得了较好的预测效果。首先，系统 ArgoUML 和 jFreeChart 在不同阈值下取得了不错

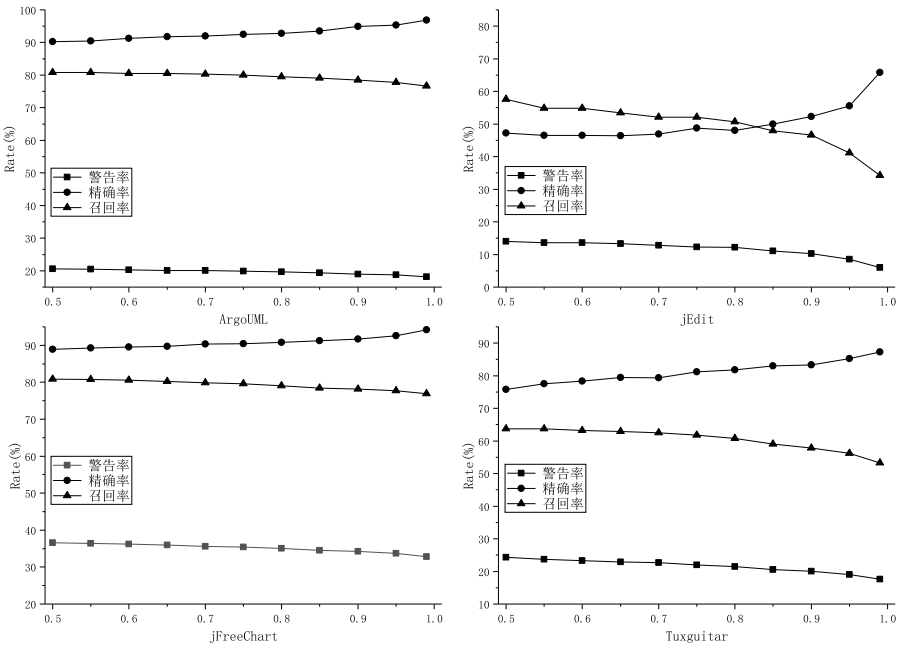


图 3-6 全属性组需要维护的全属性实验效果

Fig.3-6 The effectiveness for all attributes of clone consistency

的预测效果：精确率介于 75.8% - 96.9% 之间，召回率介于 76.6% - 80.9% 之间。系统 ArgoUML 和 jFreeChart 的训练集最大，分别拥有 766 和 1353 个需要维护的克隆代码创建实例。因此，可以较好的训练机器学习模型，并取得四个系统中最好的预测效果。然后，尽管 Tuxguitar 的预测效果没有另外两个系统的预测结果好，但是其精确率在 75.8% - 87.3%，召回率为 53.3% - 63.7%。最后，jEdit 系统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 个复制粘贴实例。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%），在 jEdit 系统上的预测依然是有效的。因此，在预测克隆代码一致性时，扩大训练集的规模对于提高预测效果具有积极意义。

对于四个实验系统，本章所构建的模型均具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 3-2 中所示）。虽然阈值的变化可以影响预测的精确率和召回率，但影响并不大，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，本章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章模型的预测结

果。然而，本章方法的召回率没有达到准确率的效果，但仍需进一步增强预测模型的召回能力，需要进一步的深入研究。

因此，在使用全部属性预测需要一致性维护的克隆创建实例时，可以达到一个较好地预测效果。但是，对于部分缺少训练数据的系统（如 jEdit 系统），本文建议预测不需要一致性维护的克隆创建实例，从而确保达到最佳的预测效果。

（2）属性组实验结果

类似的，对于一致性维护需求实例，表 3-7 给出了属性组实验结果。图中使用“WR”表示警告率，“P”表示精确率，“R”表示召回率。并同时使用不同的后缀表示不同属性组的实验结果，其中“All”为全属性组实验结果，“Code”表示仅使用代码属性的实验结果，“Context”表示上下文属性的实验结果。

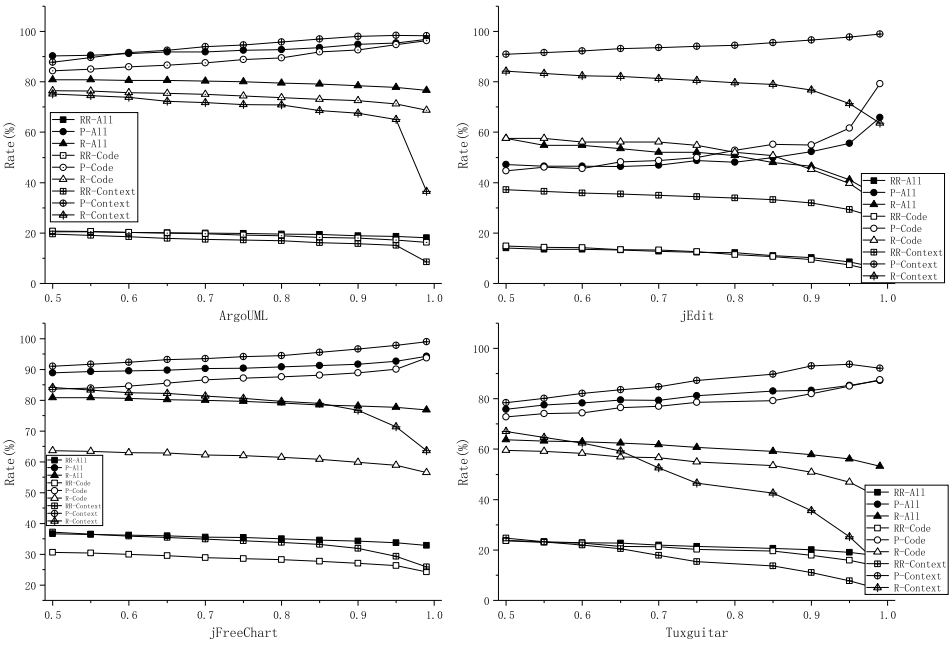


图 3-7 需要维护的属性组实验效果

Fig.3-7 The effectiveness for attribute set of clone consistency

由图中可以看出，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不相同。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 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预测研究[142]。Wang 等人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（无法获取实验系统信息），另外两个系统为开源系统 jFreeChart 和 Tuxguitar。因此，针对系统 jFreeChart 和 Tuxguitar，本节将本章的实验结果与 Wang 的方法进行对比，验证本章所提出方法的有效性。方法对比的实验结果如表 3-6 和表 3-7 所示。

表 3-6 是所收集到的克隆创建实例信息对比。其中，表中第 2 行和第 4 行为本文收集到的克隆创建实例。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 所收集的实例数量。

表 3-6 实验系统的克隆创建实例信息统计对比

Table3-6 The comparing statistics for clone creating instances in two projects

实验系统	不需要维护	需要维护	
jFreeChart(本章)	2013(59.8%)	1353(40.2%)	3366
jFreeChart(Wang)	1059(92.2%)	89(7.8%)	1148
Tuxguitar(本章)	1016(71.1%)	413(28.9%)	1429
Tuxguitar(Wang)	384(86.5%)	60(13.5%)	444

表 3-7 是本章方法与 Wang 的方法的预测效果对比情况。在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，本章所提出的方法具有更好的预测能力。具体地，对 jFreeChart 系统，本章方法的平均精确率为 88.3%，平均召回率为 88.2%，而 Wang 的方法仅为 58.3% 和 63.0%。对 Tuxguitar 系统，本章方法的平均精确率为 83.1%，平均召回率为 83.6%，而 Wang 的方法仅为 61.5% 和 60.1%。

表 3-7 克隆创建实例的预测效果对比

Table3-7 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)	88.3	88.2
jFreeChart(Wang)	58.3	63.0
Tuxguitar(本章)	83.1	83.6
Tuxguitar(Wang)	61.5	60.1

本章方法之所以优于 Wang 方法有两个可能的原因。第一个原因是本章方法改进了表示克隆创建实例的属性特征。首先，摒弃了与克隆创建实例中复制和粘贴操作关联较弱的历史属性。其次，扩展了表示克隆创建实例的代码属性和上下文属性，提取了更丰富的属性特征更为细致地刻画了克隆创建实例。第二个可能的原因是本章方法可以收集更多的克隆创建实例。本文使用经典的克隆检测工具 NiCad 检测系统存在的克隆代码，使用基于 CRD 的方法构建系统的克隆家系并收集克隆创建实例。在使用克隆创建实例进行训练学习模型时，较多的克隆创建实例使得所训练的机器学习模型更为完善，具有更好的预测效果（特别是对满足一致性维护需求的克隆创建实例）。

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一致性维护需求预测方法，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。

在有效性验证实验中，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，但支持向量机方法具有相对最佳的预测效果。因此，建议优先推荐使用支持向量机方法预测克隆代码创建一致性维护需求。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预测。

在使用模式实验中，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，均具有高效的预测能力。本文所提取的代码属性和上下文属性在两种使用模式上均起到了积极的作用。但是，对于不同系统中，所产生的影响程度并不一致。因此，建议维护人员使用全属性组进行一致性维护需求预测，从而将其使用到其它未验证的系统中，使之达到最佳的预测效果。

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从规避克隆代码产生的角度降低克隆代码的额外维护代价，本章提出了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验证了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

本文第 3 章研究克隆创建时的一致性维护需求预测，可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，其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克隆一致性违背缺陷。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，在克隆代码变化时如何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。

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，关注的是由复制粘贴操作产生的新的克隆代码，所提取的属性特征也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一致性变化的一致性违背缺陷，关注的是发生变化的克隆代码，因需求和关注点不同从不同的角度提取了不同的属性特征。首先，通过构建软件系统的克隆家系，并识别来收集系统中所有的克隆变化实例（发生变化的克隆组）。然后，提取了代码属性、上下文属性描述发生变化的克隆组，并新增一组演化属性描述其历史演化情况。最后，训练机器学习模型预测克隆代码变化的一致性维护需求。在四个开源软件系统上对本章方法进行了评估，实验结果表明本章方法可以有效地预测克隆代码变化一致性维护需求。本章的预测方法可以帮助开发人员同步开发与维护克隆代码，进而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的问题

克隆代码不仅可能会导致额外的维护代价，更重要的是还有可能引发相关缺陷。首先，发生一致性变化的克隆代码会增加软件的额外维护代价；其次，如果忘记对需要一致性变化的克隆代码进行一致性维护，会向系统中引入克隆一致性违背缺陷（Consistency Defect）[7,62]。

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。

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

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

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 是会导致一致性违背缺陷的克隆代码示例。图中 Clone A 是一段代码，将数量为 nRegs 的内存数据转移至数组 ppTotal 中。程序开发人员将 Clone A 复制并粘贴至另一个位置成为 Clone B。Clone B 具有和 Clone A 相似的功能，将数量为 nRegs 的内存数据转移至数组 ppTaken 中，并将原 Clone A 中的 ppTotal 重命名为 ppTaken。在随后的演化中，程序开发人员发现 Clone A 存在潜在的边界问题，因而在第 5 行添加条件语句 if (i+1<nRegs)，将其修改为 Clone A’。代码 Clone B 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 Clone B’ 中存在一个一致性违背缺陷。因此，如果开发人员未能在需要时对克隆代码进行一致性维护，会导致克隆代码的一致性违背缺陷，并降低软件的质量、增大其维护代价。

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

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

Clone A

```
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     }
```

Clone B

```
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     }
```

Clone A'

```
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     }
```

Clone B'

```
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  }
```

图 4-3 导致一致性违背缺陷的克隆代码[132]

Fig.4-3 An example for code clones that resulting consistency defect[132]

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[22,62]（如图 4-3 所示）。

为了维护演化中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[128]。但是，该方法未对克隆代码是否需要一致性维护进行判定，缺少必要的先行条件。同时，克隆管理工具 JSync 也支持克隆代码的一致性维护[132]。但是，该方法仅能支持标识符的一致性维护，也未在发生变化时对克隆代码进行一致性维护的判断。Yamanaka 等人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[127]。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[129]。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[130]。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。

综上所述，当软件中的克隆代码发生变化时，由于克隆组内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。因此，预测克隆代码变化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，并提高软件质量和可维护性。具体来说，如果克隆代码的

变化需要一致性维护，警告软件开发人员对变化所在的克隆组进行维护，从而避免一致性违背缺陷；如果克隆代码的变化不会引发一致性变化，则帮助开发人员可以更加自信地修改克隆代码，更快地对软件开发和维护。

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维护需求。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，并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方法如图 4-4 所示。从图中可以看出，该方法可以划分为三个阶段，克隆变化实例收集阶段、克隆变化实例表示阶段和一致性维护需求预测阶段。收集阶段旨在收集系统中全部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，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，程序开发人员可以放心地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发生变化的克隆代码，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

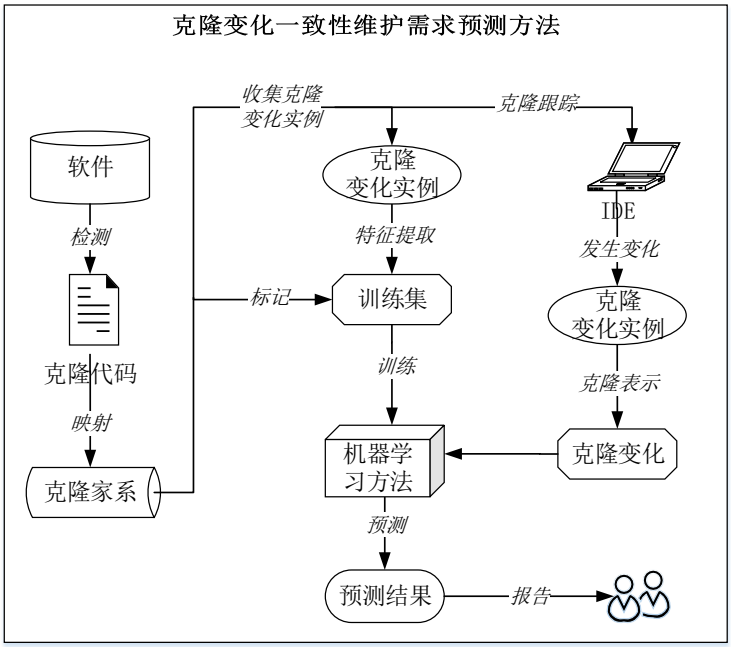


图 4-4 克隆代码变化一致性维护需求预测方法

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

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 克隆代码变化时一致性维护需求的定义

在第 3 章中，本文给出了一个克隆代码创建时的一致性变化模式定义。但是，当预测克隆代码变化一致性维护需求时，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克隆代码变化时的一致性维护需求预测中。因此，对克隆变化时的一致性变化模式进行了重新定义，如下所示：

定义 4.1 (克隆变化时一致性变化模式) 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')。如果对于一个的接近于 0 的阈值 τ，克隆代码 CF_1 和 CF_2 的变化满

足：

$$\begin{aligned} &\textit{TextSim}(CF_i, CF_i') < 1 \qquad\qquad \forall i \in \{1,2\}\ (1) \\ &\mid \textit{TextSim}(CF_1, CF_1') - \textit{TextSim}(CF_2, CF_2') \mid < \tau\ (2) \end{aligned} \tag{4-1}$$

则称克隆组 CG' 具有克隆变化一致性变化模式（Changing Consistent Change Pattern）。

在该定义中，克隆代码的变化同样使用 $\textit{TextSim}(CF_i, CF_i') = 1 - \textit{UPI}(CF_i, CF_i')$ 定义。其中，UPI（Unique Percentage Items) 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。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，即克隆代码的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接近于 0 的阈值 τ 限定，要求克隆片段发生了完全一致的修改。

要求发生相同变化的原因在于：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。

同第 3 章相似，本节定义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，也可能会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，将无法预测此类缺陷的发生。

本文将发生变化的克隆组称为克隆变化实例，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 4.2 (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不一致性变化。

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

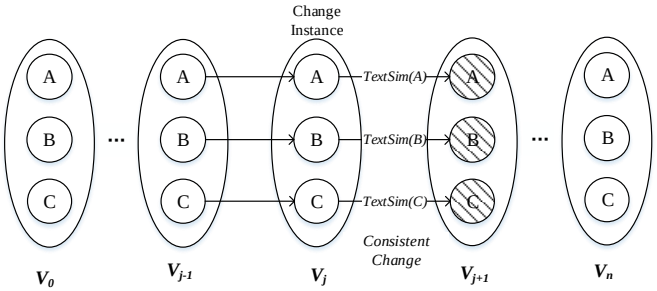


图 4-5 克隆变化实例示意图

Fig.4-5 An example for clone changing instance

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 4.3 (克隆变化时一致性维护需求) 给定版本 j 中一个克隆变化实例 CG，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“变化时一致性变化模式”（Changing Consistent Change Pattern），则称克隆变化实例 CG 满足克隆变化的一致性维护需求（Changing Consistency-Requirement）；反之，如果不存在一个克隆实例 CG'，称该克隆变化实例 CG 不需要一致性维护（Changing Consistency-Requirement Free，或 Consistency-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：研究发现克隆代码的一致性变化可能具有延迟传播现象[59,60]。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。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。从而将变化实例的一致性要求的预测转化为分类问题。

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维护需求预测的训练集，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 4.2 ，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[139]。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 2.4 和4.1 ）。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 4.3 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 4.1 ）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 4.1 ），根据定义 4.2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 4.3 ，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 4.1 ），则该实例满足一致性维护需求，否则不满足维护需求。

克隆创建实例收集算法如 4-1 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。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 6 行是生成系统的克隆家系。第 7 - 9 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 $O(m*n)$。

算法 4-1 克隆变化实例收集算法

Algo. 4-1 Algorithm for collecting clone changing instances

Input: All source code and code clones from N versions

Output: All the clone changing instances

```
for i=1 to N do
    Generating CRD to represent each code clone CFs;
    Mapping all the CFs and CGs between the adjacent
      versions i and i+1;
    Identifying all clone evolution pattern in thus two versions
      according to Definition 2.4 ;
    Identify all clone consistent change pattern in thus two
      versions according to Definition 4.1 ;
Generating all clone genealogies CGEs with number N_cge;
for i=1 to N_cge do
    Collecting the clone changing instance from CGE_i
      according to Definition 4.2 ;
    Labeling its consistency-requirement according to
      Definition 4.3 ;
return All clone changing instances;
```

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，同时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。因此，克隆组内的克隆代码片段均有可能会影响克隆代码的一致性维护。鉴于此，本章在第 3 章对克隆创建实例表示的基础上，通过从克隆组的角度提取代码属性和上下文属性表示克隆变化实例。同时，在第 2 章克隆演化特征分析中，本文发现克隆代码的变化

往往发生经过一段时间的演化后。因此，克隆变化实例的历史演化情况，也会影响克隆代码的一致性变化。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。

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。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。

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。第 3 章的代码属性描述了一个克隆代码片段的粒度、Halstead 属性、结构属性、调用属性等。因此，在第 3 章的基础上，本章提取克隆变化实例的代码属性，即从克隆组的角度提取克隆变化实例相应的属性（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，该克隆组中所有克隆片段的数量。
- 平均代码行数：在克隆变化实例中，所有克隆片段的平均代码行数。
- 平均 Halstead 属性：在克隆变化实例中，所有克隆片段的平均 Halstead 属性，同样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。
- 平均结构属性：在克隆变化实例中，所有克隆片段的平均结构属性，包括 if_then、if_else、switch、while、do、for、this_or_super 等。
- 平均参数访问数量：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。
- 平均总函数调用次数：在克隆变化实例中，所有克隆片段中所有函数调用的次数统计的平均。
- 平均本地函数调用次数：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的平均。
- 平均库函数调用次数：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的平均，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。
- 平均其它调用次数：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的平均，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（计算方法同第 3 章），然后将上下文属性进行加权平均。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：

• 平均代码相似度：在克隆变化实例中，所有的克隆代码的代码相似度的平均。

• 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，是的话取值为 1，反之为 0。

• 平均文件名相似度：在克隆变化实例中，所有的克隆代码的文件的名相似度的平均。假定文件名分别为 M_1 和 M_2，则文件名相似度为 $Sim(M_1, M_2)$，采用李式距离[145] 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。

• 平均方法名相似度：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的平均。

• 平均总参数名相似度：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 $(P_1, P_2, \ldots, P_m)$ 和 $(Q_1, Q_2, \ldots, Q_n)$，则总参数名相似度为 $\sum_{i=1}^{m}\sum_{j=1}^{n}(Sim(P_i, Q_j))$。

• 平均最大参数名相似度：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 $(P_1, P_2, \ldots, P_m)$ 和 $(Q_1, Q_2, \ldots, Q_n)$，该最大参数名相似度为 $Max(Sim(P_i, Q_j))$。

• 平均总参数类型相似度：在克隆变化实例中，首先计算任意两个克隆片段的总参数类型相似度，然后对所有的相似度进行加权平均。假定两个克隆代码片段

所在方法分别为 M 和 N，M 和 N 分别包含 m 和 n 个参数，其参数类型分别为 $(P_1, P_2, ..., P_m)$ 和 $(Q_1, Q_2, ..., Q_n)$，该参数类型相似度为 $\sum_{i=1}^{m}\sum_{j=1}^{n}(Sim(P_i, Q_j))$。

• 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。在本文第 2 章的克隆演化特征研究中，发现克隆代码的变化往往是发生在克隆代码演化后的几个版本之后，因此发生变化时的克隆代码的寿命对一致性变化有影响。与此同时，在发生变化的克隆组，在变化之前也可能发生过变化，其历史演化情况也会影响此次克隆组的变化情况。因此，本章从克隆代码演化的角度提取了相应的属性共同描述克隆变化实例。克隆家系是克隆组的演化情况的描述，借助于克隆家系描述克隆变化实例的演化情况。对一个克隆变化实例，本章提取的演化属性包括，克隆寿命、历史演化模式、上次演化时的演化模式克以及克隆组的历史变化等。

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

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。

综上所述，本章从克隆组的角度提取了代码属性、上下文属性以及历史属性表示克隆变化实例。给定一个克隆创建实例，同样使用一个 n 维向量表示发生变化的克隆组，$Vector$ = (v_1，v_2，...，v_m)，其中 v_i 表示该克隆变化实例的一个特定属性。

4.6 模型的训练与预测

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 4.3，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，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。

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。

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 - 7 行提取相应的属性组表示克隆变化实例，并生成训练集。对每一个克隆变化实例，分别提取其代码属性、上下文属性和演化属性，并生成一个向量表示克隆变化实例。第 8 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。

算法 4-2 克隆变化一致性维护需求预测算法

Algo. 4-2 Algorithm for predicting clone changing consistency

Input: All clone changing instances and source code

Output: The predictive model

```
Initializing the training data set Sets for clone changing instances;
for i=1 to M do
    Generating all the code attributes for clone instance;
    Generating all the context attributes for clone instance;
    Generating all the evolution attributes for clone instance;
    Generating all the change attributes for clone instance;
    Appending all the attributes as one item to the Sets;
Calling WEKA to train the model on training set Sets;
return The predictive model;
```

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，本章采用和第 3 章相同的四个 Java 开源软件进行实验（如表 3-1 和 4-1 所示）。通过遍历克隆家系中发生变化的克隆组（一致和不一致性变化模式）来收集克隆变化实例。所采用的实验系统是经过充分演化的开源系统，这些系统已经在克隆代码相关研究中被大量采用。同时，所选择的软件版本均为正式发布版本，所有的系统版本经专业开发团队测试后发布。因此，本章所收集的克隆变化是值得信赖的。但是，本文也无法保证所收集到的数据是绝对正确的，可能也会引入一些不可避免的噪声。

四个实验系统的克隆变化实例的数量与分布情况如表 4-1 所示，表中第 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。对于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。对于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。

表 4-1 实验系统克隆变化实例的信息统计

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(67.5%)	139(32.5%)	427
jEdit	78(49.1%)	81(50.9%)	159
jFreeChart	452(43.5%)	588(56.5%)	1040
Tuxguitar	91(25.7%)	263(74.3%)	354

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。因此，克隆代码的变化问题需要引起开发人员的关注。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。上述所观察到的现象与已有研究中所观察的情况相一致[21][22]。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 3-2 ），但是仍在存在数百个需要一致性维护的实例。对于这些需要一致性维护的克隆变化实例，更需要引起开发人员的关注，因为其更容易导致一致性违背缺陷。

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	7.59	26.22	2
jEdit	159	2.467	0.91	2
jFreeChart	1040	7.85	26.92	2
Tuxguitar	354	3.40	3.45	2

一个克隆变化实例是发生变化的一个克隆组，因此也统计了克隆变化实例所包含克隆代码片段的数量。表 4-2 给出了四个实验系统中克隆变化实例包含克隆代码数量的统计信息。表中第 2 列是系统所有的克隆变化实例的数量。表中第 3 - 5 列是克隆变化实例所包含的克隆代码的数量的统计信息。最后一列显示了克隆代码数量的中位数为 2，即绝大部分的克隆变化实例包含两个克隆代码。这进一步地验证了本章所提出的一致性变化定义中仅要求两个克隆代码片段同时发生变化的限定。系统中虽然有大量的克隆代码，但是变化实例所包含的克隆片段数量并不

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 4-3 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

由于克隆变化实例是已经存在于系统中的发生变化的克隆组，因此也对发生变化的克隆组进行了克隆寿命统计（截止到发生变化时的版本）。如前所述，本文将克隆组在软件中所经历的软件版本的数量称为克隆寿命（从 0 开始计算）。表 4-3描述了克隆变化实例的克隆寿命统计信息。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。

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，观察在不同机器学习方法上的有效性，并帮助程序开发人员选择一个最优的机器学习方法。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克隆变化实例进行实验，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。

本章提取了三组度量表示克隆创建实例，分别为代码属性、上下文属性和演化属性。为了评估所提取的属性特征的有效性，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。全属性实验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。

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。首先，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 Rate）、召回率（Recall Rate）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（计算方式同本文第 3 章第 3.7.1.2 实验评估指标小节），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。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。

• 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，包含满足一致性维护需求和不满足一致性维护需求的实例。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。

• 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。

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（BN）的父节点数设置为 4，朴素贝叶斯方法（NB）只有一个父节点，同样设置阈值为 0.5。支持向量机方法（SVM）使用 Polynomial Kernel 作为核函数。使用 J48 算法作为决策树（DT），设置置信度为 0.75。K 近邻方法（KNN）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，实验结果分别如表 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。表中第 1 列和第 2 列分别为所使用的评估指标和五种不同机器学习方法，表中第 3 - 6 列为在四个实验系统上的实验结果。表中第 2 - 6 行为平均精确率，第 7 - 11 行为平均召回率，第 12 - 16 行为平均 F 值。

从表 4-4 可以看出，克隆变化实例在这五种机器学习方法上具有良好的预测效果。精确率的范围从 58.1% 到 79.3 %，召回率从 57.9% 到 79.1%，而 F 值从 57.3% 到 79.0%。其中，jFreeChart 的预测效果最好，其 F 值从 73.9% 到 79.0%，ArgoUML

和 Tuxguitar 系统的预测效果次之，F 值从 63.2% 到 73.3%，系统 jEdit 的预测效果最差，F 值 57.3% 到 70.4%。结合实验系统的数据分布，可以看出本章方法的预测效果要高于其数据分布情况，因此克隆变化的一致性维护需求预测是有效的。

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

评估指标	系统	ArgoUML	jEdit	jFreeChart	Tuxguitar
平均 Precision(%)	BN	72.4	68.6	79.1	72.0
	NB	73.4	69.6	77.8	72.9
	SVM	74.4	70.4	79.3	73.3
	KNN	73.3	59.7	77.2	67.2
	DT	68.2	58.1	74.2	63.7
平均 Recall(%)	BN	73.5	68.6	79.1	74.6
	NB	72.6	69.2	77.8	73.7
	SVM	75.2	70.4	79.1	73.4
	KNN	73.1	59.7	77.0	70.6
	DT	69.8	57.9	74.2	67.2
平均 F-measure(%)	BN	72.6	68.6	79.0	72.6
	NB	72.9	68.9	77.8	73.3
	SVM	72.9	70.4	78.9	73.3
	KNN	73.2	59.7	77.1	68.3
	DT	63.2	57.3	73.9	65.1

在四个系统的预测结果中，机器学习方法对不同的项目具有不同的预测效果。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，而系统 jEdit 是所有系统中预测效果最差的，另外两个系统的预测效果居中。通过分析这四个实验系统的实验数据，可以发现系统 jFreeChart 中克隆变化实例的数量最多，系统 jEdit 的克隆变化实例最少。因此，预测效果与所使用训练集的规模有关系，其数据集越大预测效果越好。因此，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。

通过对比五种机器学习方法在四个系统上的预测效果，发现除决策树方法外，另外三种方法的预测能力较为相似，没有明显的差异。同时，相对而言支持向量机方法具有相对较好的预测效果。具体来说，支持向量机对 jEdit、jFreeChart 和 Tuxguitar 具有最好的结果，K 近邻和支持向量机对于 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，其预测结果可以接受。应该注意的是，K 近邻和决策树方法这两种机器学习方法相对而言预

测能力不如其它的机器学习方法，尤其是决策树方法。因此，建议开发人员在预测克隆变化实例时也可以考虑支持向量机方法，但是贝叶斯的方法也具有不错的预测能力，仅比支持向量机相差一点。

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，即依次删除一个属性集。属性集的实验结果如表 4-5 所示。表中第 1 列和第 2 列分别为实验评估指标和实验系统，表中第 3 列给出了不同系统使用不同属性组进行的实验。其中，“全部属性”是使用全部属性组的实验结果，“无代码属性”、“无上下文属性”、“无演化属性”分别是是删除相应属性组后的实验结果。

从表中可以看出，所提取的属性特征对克隆变化实例的一致性维护需求具有积极的作用。首先，属性组实验结果与全属性的实验结果较为接近。尽管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。其次，预测效果最差的预测结果往往出现在属性组的预测结果中。三个属性组中在预测中起到了积极作用，尤其是三个属性组作为一个整体以可以接受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。另外，不同的属性特征在预测中也会发挥不同的作用。

不同的属性组在预测中也发挥不同的作用。首先，对于 ArgoUML 和 jEdit 系统来说，无上下文属性的实验结果最差，因此上下文属性对这两个系统来说最为重要。对于 jFreeChart 系统来说，只有使用全属性组进行预测才可以达到最好的预测效果，因此所提取的全部属性均具有重要的作用。而对于 Tuxguitar 系统，代码属性所起到的作用最大，是最重要的属性组。因此，当使用本章方法进行预测时，程序开发人员可以根据项目特点选择不同属性特征，从而使之达到最佳的预测效果。

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。因此在预测中本章建议保留全部的属性集，并优先选择支持向量机方法。

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，对两种不同状态的克隆变化实例分别进行预测，即不满足一致性维护需求的实例（一致性维护自由实验）和满足一

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

评估指标	实验系统	实验组	BN	NB	SVM	KNN	DT
平均 Precision(%)	ArgoUML	全部属性	72.4	73.4	74.4	73.3	68.2
		无代码属性	73.7	74.3	73.7	69.2	68.9
		无上下文属性	71.2	69.3	73.6	68.8	71.3
		无演化属性	72.7	72.3	75.8	72.5	69.6
	jEdit	全部属性	68.6	69.6	70.4	59.7	58.1
		无代码属性	69.8	66.2	74.9	52.2	57.9
		无上下文属性	67.3	63.6	68.7	61.7	57.1
		无演化属性	65.4	67.6	64.2	68.0	59.5
	jFreeChart	全部属性	79.1	77.8	79.3	77.2	74.2
		无代码属性	76.0	75.6	74.2	70.3	74.6
		无上下文属性	77.3	73.1	76.9	74.4	71.1
		无演化属性	76.0	74.7	77.5	74.1	73.3
	Tuxguitar	全部属性	72.0	72.9	73.3	67.2	63.7
		无代码属性	68.6	70.0	67.8	63.9	62.1
		无上下文属性	67.2	69.0	72.6	65.9	65.8
		无演化属性	72.7	71.9	69.9	66.9	63.4
平均 Recall(%)	ArgoUML	全部属性	73.5	72.6	75.2	73.1	69.8
		无代码属性	74.2	73.8	74.2	69.8	70.0
		无上下文属性	72.4	68.1	74.0	68.9	72.6
		无演化属性	73.3	71.0	76.6	73.3	70.3
	jEdit	全部属性	68.6	69.2	70.4	59.7	57.9
		无代码属性	69.8	66.0	74.8	52.2	57.9
		无上下文属性	67.3	63.5	68.6	61.6	55.3
		无演化属性	65.4	67.3	64.2	67.9	59.1
	jFreeChart	全部属性	79.1	77.8	79.1	77.0	74.2
		无代码属性	76.1	75.7	73.9	70.0	74.5
		无上下文属性	77.4	73.2	76.8	74.2	71.1
		无演化属性	76.1	74.2	77.5	73.8	73.4
	Tuxguitar	全部属性	74.6	73.7	73.4	70.6	6.72
		无代码属性	71.8	70.3	67.8	68.1	65.3
		无上下文属性	70.9	68.6	71.8	68.9	67.2
		无演化属性	74.3	73.7	69.8	68.9	67.8
平均 F-measure(%)	ArgoUML	全部属性	72.6	72.9	72.9	73.2	63.2
		无代码属性	73.9	74.0	71.2	69.4	63.4
		无上下文属性	71.5	68.6	70.7	68.8	69.4
		无演化属性	72.9	71.4	75.0	72.7	63.5
	jEdit	全部属性	68.6	68.9	70.4	59.7	57.3
		无代码属性	69.8	65.9	74.8	52.2	57.7
		无上下文属性	67.3	63.4	68.4	61.6	53.3
		无演化属性	65.4	67.1	64.2	67.8	58.4
	jFreeChart	全部属性	79.0	77.8	78.9	77.1	73.9
		无代码属性	76.0	75.6	73.3	70.1	74.1
		无上下文属性	77.2	73.1	76.5	74.3	71.1
		无演化属性	76.0	74.3	77.3	73.9	73.1
	Tuxguitar	全部属性	72.6	73.3	73.3	68.3	65.1
		无代码属性	69.5	70.2	67.8	65.5	63.5
		无上下文属性	68.3	68.8	72.1	67.1	66.4
		无演化属性	73.2	72.5	69.8	67.7	65.1

致性维护需求的实例（一致性维护实验）。贝叶斯网络的构建使用 K2 算法，并设置贝叶斯网络的最大父节点个数为 4，使用 SimpleEstimator 来计算网络的概率表。同样采用十倍交叉验证（10-Folds Cross-Validity）评估预测模型的预测能力。

4.7.3.1 一致性维护实验

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，从而避免克隆代码的一致性违背缺陷。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，即预测结果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预测，实验结果如图 4-6 所示。图中的横坐标表示阈值，设置阈值从 0.50 到 1.0，图中的纵坐标为评价指标的数值大小。

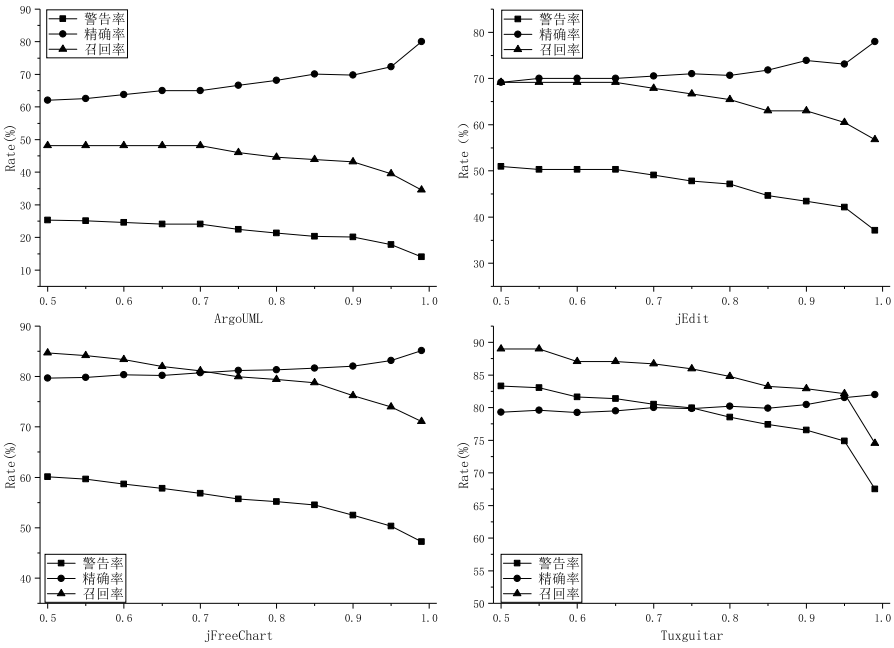


图 4-6 全属性组需要维护的实验效果

Fig.4-6 The effectiveness for all attribute sets of clone consistency

由图 4-6 中可以看出，所构建的模型在预测 jFreeChart 和 Tuxguitar 系统的一

致性需求时，预测效果非常有效，其中预测的精确率和召回率在 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，精确率和召回率在 70% 左右。原因可能是 jEdit 系统的克隆变化实例规模较小，没有对模型实现较好地训练，但这需要进一步验证研究。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。

分析这种差异背后的原因有两个可能的因素。首先，不同项目的预测效果与训练集规模有关系。jFreeChart 和 Tuxguitar 系统具有最多的需要一致性维护的克隆变化实例，可以较好的训练模型。然后，预测效果与样本不平衡性也有关系。尽管 ArgoUML 系统的训练数据要多于 jEdit 系统，但对 ArgoUML 系统不需要维护的克隆代码变化实例更多，导致所训练的模型向不需要维护的一侧偏移。因此，ArgoUML 系统在预测需要一致性维护需求的克隆变化实例时，预测效果最差。

对于四个实验系统，本章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。这意味着本章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章的方法，对需要一致性维护的克隆变化实例进行预测。同时，还可以采用其它方法来避免那些需要一致性维护的克隆实例，例如采用第 3 章的方法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。

（2）属性组实验结果

本章提取的了三组属性表示克隆变化实例，从不同的角度描述了变化实例的特征。为评估不同的属性组对预测效果的作用，本节进行了属性组实验。图 4-7 中给出了属性组的评估实验结果。其中，使用“All”表示全属性实验结果，“Code”表示移除代码属性的实验结果，“Context”表示上下文属性的实验结果，“Evo”表示移除演化属性的实验结果。并使用“WR”表示警告率，“P”表示精确率，“R”表示召回率。

从图中可以看出，属性组对不同系统的预测结果的影响并无一致性结论。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。为进一步验证但是，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

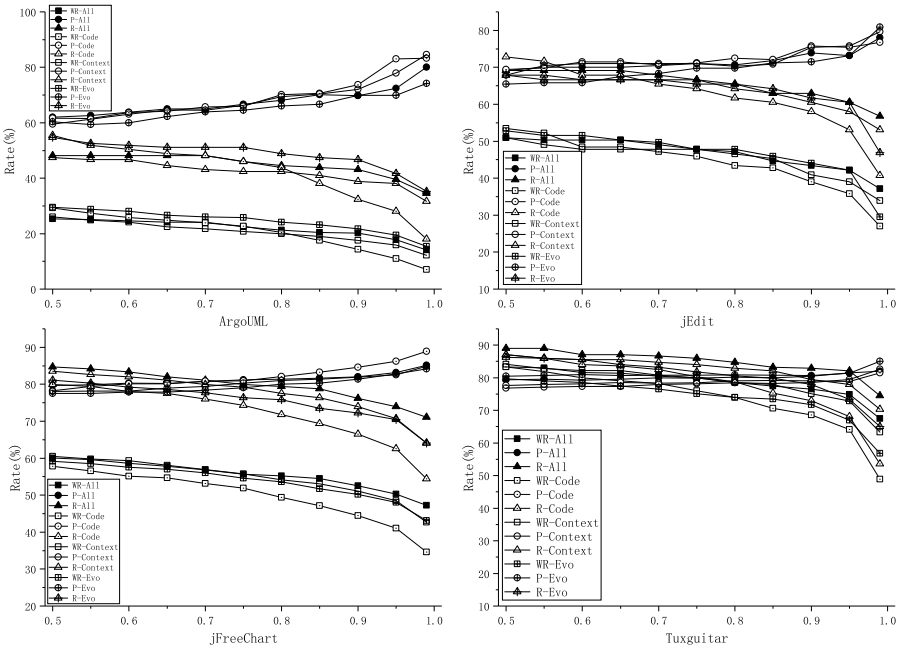


图 4-7 需要维护的属性组实验效果

Fig.4-7 The effectiveness for attribute sets of clone consistency

能选取了最佳属性集。然而，通过比对全属性组的实验结果，发现发现全属性组实验结果要优于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着的影响，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。关注不需要一致性维护的克隆变化实例，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。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，实验结果如图 4-8 所示。图中的横坐标表示阈值，设置阈值从 0.01 到 0.50，图中的纵坐标为评价指标

的数值大小。

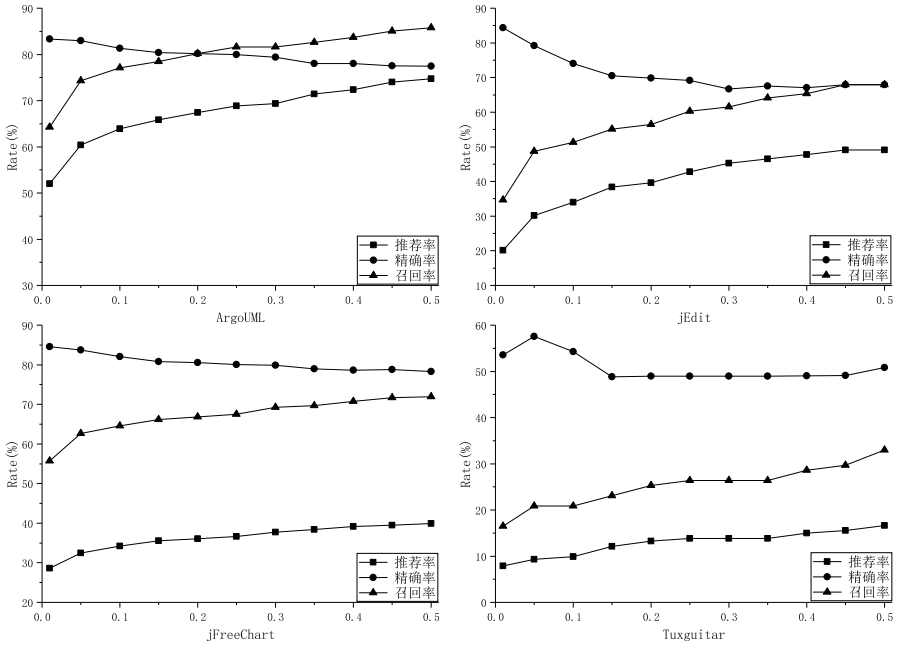


图 4-8 全属性组维护自由实验效果

Fig.4-8 The effectiveness for all attributes of clone consistency free

由图中 4-8 可以看出，本章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，本章的方法在系统 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，没有对预测模型进行完全训练。

分析这种差异背后的原因可以得出相似的结论，即对不同项目的预测效果会依赖于两个因素：训练集的规模以及训练数据的分布情况（样本的不平衡性）。jFreeChart 系统中拥有最多的不需要一致性维护的克隆变化实例，使得所训练的机器学习模型较为完善。而 ArgoUML 的样本数据中，有 67.5% 的不需要维护的变化实例，所训练的模型会向数据较多的一侧偏移。

同时，由图 4-8 和表 4-1 可以看出，本章所构建的模型在四个系统上都具有十分合理的推荐率，推荐率和系统本身的不需要一致性维护的克隆变化的比例相差

不大。与此同时，阈值会影响到预测的精确率和召回率，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较为稳定地进行一致性维护需求预测（精确率相差不大），但是系统的召回率仍需要进一步的提升。简而言之，本章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，并且具有较好的精确率和召回率，软件开发人员可以据此对克隆变化实例进行有效的预测。

（2）属性组实验结果

针对不需要一致性维护的克隆变化实例，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。实验结果如图 4-9 所示。其中，使用“RR”表示推荐率，“P”表示精确率，“R”表示召回率。同时，使用“All”表示全属性实验结果，“Code”表示移除代码属性的实验结果，“Context”表示上下文属性的实验结果，“Evo”表示移除演化属性的实验结果。

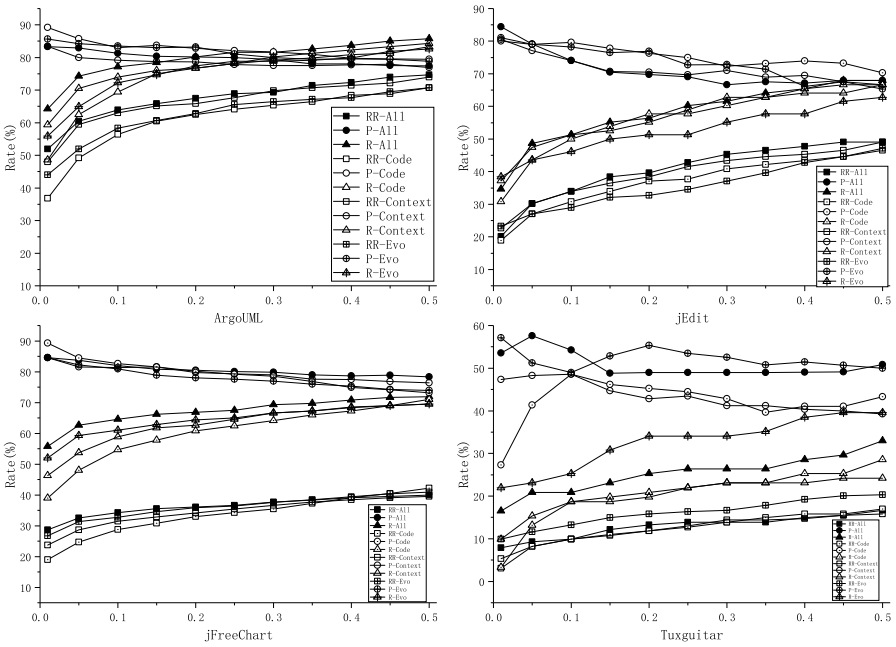


图 4-9 属性组维护自由实验效果

Fig.4-9 The effectiveness for attribute set of clone consistency free

从图中可以看出，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积极的影响，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，发现发现全属性组实验结果要好于最佳属性子集的实验结果，因此，所采用的属性具有积极的意义。虽然一些属性组可能对实验结果没有显着的影响，但是其也不具有消极的影响。

尽管一些属性组可能对实验结果没有极为重要的贡献，但是属性组同样也没有产生消极的影响。因此，本章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，同时对需要和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。

在有效性预测实验中，本章所提的方法是有效的。五种机器学习方法都可预测克隆变化的一致性维护需求，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，但支持向量机方法拥有相对最佳的预测效果。因此，建议优先推荐使用支持向量机方法预测克隆代码变化一致性维护需求。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。

在使用模式实验中，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。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，从而将其适用到其它未验证的系统中，使之达到最佳的预测效果。

4.8 本章小结

软件中存在的克隆代码的一致性变化，可能会引发克隆一致性违背缺陷。但是，现有的方法中尚无法在克隆代码变化时预测克隆代码的一致性变化，更无法帮助避免一致性违背缺陷。因此，本章提出了一个克隆代码变化一致性维护需求预测方法，帮助开发人员实现克隆代码的同步开发与维护。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。在随软件系统进行演化的过程中，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。因为在此阶段软件系统没有经过充分的演化，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，可以在软件开发初期帮助避免克隆代码的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，研究跨项目的克隆代码一致性维护需求预测。由于软件开发流程大同小异，且本文使用一致的属性特征表示克隆代码，因此跨项目的数据也具有统一的表示形式。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，并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，并给出了一些建议帮助提高跨项目的预测能力。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，可以帮助避免克隆代码的额外维护代价和一致性违背缺陷。

在软件开发的过程中，为了快速有效地进行软件开发，软件开发团队往往会制定出相应的规范和标准，用于标准化软件开发过程。同时，不同软件开发团队制定的标准和规范，往往是大同小异的，因此软件开发活动具有一定的相似性。与此同时，目前对克隆代码的研究中，也已经出现了一些跨项目的克隆代码检测和复用方法[114,115]，进一步说明了不同项目之间的克隆代码也具有一定的关联性。更重要的是，本章研究跨项目的克隆代码一致性维护需求预测，所提取的克隆实例的特征属性是一致的，使得不同项目之间的数据也具有相同的表示形式。因此，跨项目的克隆代码一致性维护需求预测是可行的。

为将克隆代码的一致性维护需求预测方法（第 3、4 和 5 章方法）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。结合软件开发过程，对克隆代码进行一致性维护需求预测可以帮助开发人员实现边开发、边维护克隆代码，提高软件质量和可维护性。

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，首先给出本章方法的框架。跨项目克隆代码一致性需求预测方法如图 5-1 所示。从图中可以看出，该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。

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

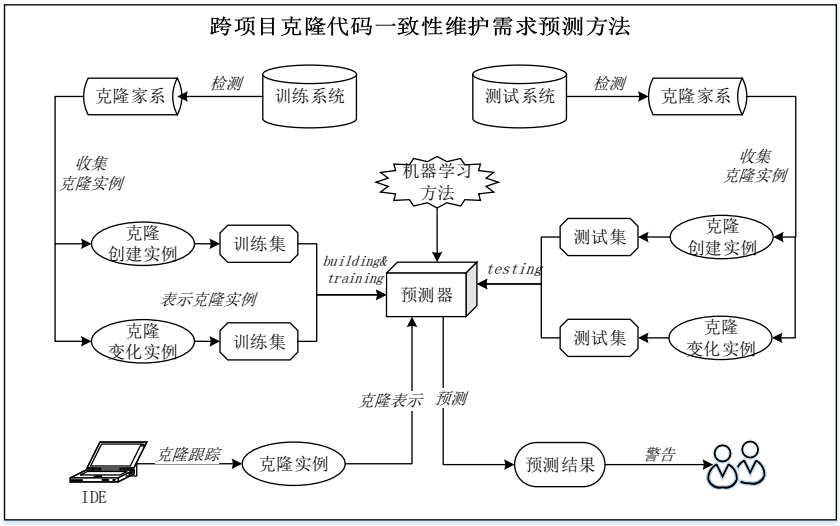


图 5-1 跨项目克隆代码一致性维护需求预测方法

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

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。

每一个克隆实例有两种不同的预测结果，即满足一致性维护需求和不满足维护需求。对于满足一致性维护需求实例来说，其在将来的演化中可能会引发一致性变化，程序开发人员需要采取相应的操作。例如，拒绝克隆创建实例或者检查克隆变化实例的一致性。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，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？

（3）在进行跨项目的克隆一致性维护需求预测时，训练数据将如何影响跨项目的预测效果？

5.3 克隆代码一致性维护需求的定义

本文第 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。首先，为更好的预测克隆代码的一致性维护需求，进一步提炼了克隆片段的一致性变化，如下所示：

定义 5.1 (克隆片段的一致性变化) 克隆组内两个克隆代码片段 CF_1 和 CF_2，被分别地修改为 CF_1' 和 CF_2'。如果对于一个接近于 0 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），

$$\begin{aligned} &\mathit{TextSim}(CF_i, CF_i') < 1 \qquad\qquad \forall i \in \{1,2\}\ (1) \\ &\mid \mathit{TextSim}(CF_1, CF_1') \ - \ \mathit{TextSim}(CF_2, CF_2') \mid \ < \ \tau\ (2) \end{aligned} \tag{5-1}$$

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，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 确保克隆代码片段发生了相同的变化。

其中，Type-1 一致性变化表示克隆创建时的一致性变化，Type-2 一致性变化表示克隆变化时的一致性变化。这两种不同的一致性变化，将分别应用于两种不同时间的克隆一致性维护需求预测中。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 的一致性变化。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，还需要发生相同的变化，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。

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一致性变化模式描述。根据克隆片段一致性变化定义，可给出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，在两种不同的时刻预测克隆代码的一致性维护需求，结合第 3 章和第 4 章的克隆实例的定义，这里将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，从而导致额外的维护代价；同时如果不能确保克隆组的一致性，也会导致一致性违背缺陷。具体来说，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，即克隆创建实例或者克隆变化实例，如果在版本 k 中存在一个克隆组 $CG'(k > j)$ 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)，则称克隆实例 CG 满足克隆一致性维护需求 (Consistency-Requirement)；反之，如果不存在克隆实例，称该克隆实例 CG 不需要一致性维护 (Consistency-Requirement Free，或者 Consistency-Free，或者 Free)。其中，克隆创建实例和克隆变化实例的一致性维护需求分别满足 Type-1 和 Type-2 一致性变化模式。

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（即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。所采用的方法与本文第 3 章和第 4 章的方法类似，但在跨项目预测中，需要同时收集克隆创建和克隆变化实例。

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 2.4 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 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。

数据集生成算法如 5-1 所示。算法中第 1 行是使用第 2 章的克隆家系构建算法生成克隆家系并识别克隆演化模式。第 2 - 4 行是根据第 3 和 4 章的算法收集克隆创建和变化实例，并标记其一致性维护需求。第 5 行是初始化数据集。第 6 - 8 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 $O(m*n)$。

算法 5-1 数据集生成算法

Algo. 5-1 The algorithm for generating dataset

Input: All source code and code clones from N versions

Output: The data set of software

```
Generating all clone genealogies CGEs with number N_cge
  according to Algo. 2-1 ;
for i=1 to N_cge do
    Collecting the clone instance that both including
      clone-creating and changing instances from CGE_i
      according to Algo. 3-1 and Algo. 4-1 ;
    Labeling all consistency-requirement according to
      Definition 5.4 ;
Initializing all training data set Creating_Set and
  Changing_Set for clone creating and changing instances;
for each clone instance do
    Generating all attributes for clone creating or changing
      instance according to Algo. 3-2 and Algo. 4-2 ;
    Appending all the attributes to the Creating_Set or
      Changing_Set;
return The data set of software;
```

对于克隆创建实例，提取与第 3 章相同的属性特征，具体信息可以参考本文第 3.5 节克隆创建实例表示。使用代码属性用于表示被复制的克隆代码的特征，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。

对于克隆变化实例，提取与第 4 章相同的属性特征，具体信息可以参考本文第 4.5 节克隆变化实例表示。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。

5.5 跨项目预测模型的训练与预测

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。

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要一致性维护。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，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。

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。

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

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;*

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

5.6.1 实验设置

本章在四个 Java 开源系统上进行实验评估，所收集的克隆实例的统计信息如表 5-1 所示。表中第 1 列给出不同的克隆实例，即克隆创建实例和克隆变化实例。第 2 列给出了测试系统，第 3 - 5 列给出了测试集的数据分布情况。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。

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。表中第 1 列给出不同的克隆实

表 5-1 跨项目的测试集信息统计

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

例。第 2 列给出了训练系统，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系统。第 3 - 5 列给出了训练集的数据分布情况。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 跨项目的训练集信息统计

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
克隆创建实例	(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
克隆变化实例	(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维护需求预测问题，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评估，并回答所提出的三个子研究问题：

• 跨项目有效性验证实验：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。

• 跨项目使用模式实验：将回答第二个子研究问题，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。

• 跨项目数据集规模实验：将回答第三个子研究问题，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。

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 克隆创建实例的实验结果

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。表中第 1 列为评估指标，第 2 列为五种不同的机器学习方法，第 3 - 6 列为在四个实验系统上的实验结果。同时，表中支持向量机方法（SVM）的预测结果使用 * 进行标记表。原因在于：在跨项目克隆创建实例的一致性维护需求预测中，支持向量机方法不具备相对应的预测能力，会将所有的克隆创建实例划分到一个类别中（数量规模较大的类中）。

从表5-3 中可以看出，除了支持向量机外，其它四种机器学习方法对四个不同的系统均具有不错的预测效果。对 jEdit 系统的预测效果最好，平均 F 值在 76.2% - 82.6% 之间。同时 ArgoUML 系统和 Tuxguitar 系统的预测结果也不错，平均 F 值分别介于 65.3% - 66.7% 和 60.3% - 68.3% 之间。jFreeChart 最差，也达到了 48.9% - 60.9%。

尽管不同机器学习方法的预测效果不存在显著的差异，但不同的机器学习方法对不同系统具有不同的预测能力。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 K 近邻方法对于系统 jFreeChart 具有最

表 5-3 克隆创建实例的跨项目预测效果

Table5-3 The effectiveness of cross-project prediction for clone creating instances

评估指标	方法	To ArgoUML	To jEdit	To jFreechart	To Tuxguitar
平均 Precision(%)	BN	63.8	81.2	57.5	64.8
	NB	63.1	78.3	48.6	61.5
	SVM	59.4*	78.3*	35.8*	50.6*
	KNN	62.9	78.6	64.8	58.1
	DT	62.9	79.6	56.6	70.4
平均 Recall(%)	BN	67.5	84.4	60.2	69.8
	NB	69.3	74.7	54	64.7
	SVM	77.1*	88.5*	59.8*	71.1*
	KNN	68.7	74.1	64.9	66.3
	DT	74	84.4	59.9	73.1
平均 F-measure(%)	BN	65.4	82.6	50.8	65
	NB	65.6	76.4	48.9	62.7
	SVM	67.1*	83.1*	44.8*	59.1*
	KNN	65.3	76.2	60.9	60.3
	DT	66.7	81.8	50.3	68.3

好的预测效果，F 值为 60.9%。

因此，本章不推荐程序开发人员选用支持向量机方法对克隆创建实例进行跨项目一致性维护需求预测，可以使用其它的机器学习方法进行跨项目预测；同时，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为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，使用五种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。表中部分支持向量机的实验结果使用 * 标记，原因同样是支持向量机将克隆变化实例划分到同一个类别中。

从表中可以看出，对于跨项目的克隆变化实例的预测效果较为一般，最好的预测结果的平均 F 值仅达到 55% 左右。分析其预测能力一般的原因，可能是克隆代码的变化情况也与项目之间的关联较大，因此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，对不同的系统存在某种机器学习方法使得预测效果可以达到一个相对较好的结果。例如（To ArgoUML）中 K 近邻方法的 F 值可以达到 54.6%，（To jEdit）中决策树方法的 F 值为 56.7%，（To jFreeChart）的朴素贝叶

表 5-4 克隆变化实例的跨项目预测效果

Table5-4 The effectiveness of cross-project prediction for clone changing instances

评估指标	方法	To ArgoUML	To jEdit	To jFreechart	To Tuxguitar
平均 Precision(%)	BN	59.1	54.9	53.7	60.2
	NB	62.9	54.7	56.6	55.6
	SVM	10.6*	26*	61.1	52.4
	KNN	62.4	54.2	53.3	62.3
	DT	62.9	64.8	54.6	63.8
平均 Recall(%)	BN	47.1	54.7	51.2	41.8
	NB	46.4	54.7	53.8	35.3
	SVM	32.6*	50.9*	59.6	32.2
	KNN	53.4	54.1	50.9	45.2
	DT	37	60.4	56.7	38.7
平均 F-measure(%)	BN	47.4	54.6	50.7	44.1
	NB	45	54.5	53.3	36.1
	SVM	16*	34.4*	52.2	32
	KNN	54.6	54	50.5	47.8
	DT	27.3	56.7	48.9	38.3

斯方法为 53.3%。因此，对不同的系统开发人员可以选择适合于该系统的机器学习方法进行预测。

不同的机器学习方法在跨项目预测中，其预测能力是不一致的。对系统 ArgoUML 和系统 Tuxguitar 来说，K 近邻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%。

因此，在跨项目的预测中，不推荐程序开发人员选用支持向量机的方法，并且需要根据项目自身特征和预测结果选择最合适的机器学习方法进行预测。

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；而在选择合适的机器学习模型的前提下，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，不建议选择支持向量机方法进行预测。同时，不同机器学习方法的预测能力不同，可以根据不同的系统选取合适的机器学习方法

进行预测。

最后，相比于克隆创建时的一致性维护需求跨项目预测，克隆变化实例的预测效果一般。结合第 4 章项目内的预测结果，建议最好在收集到项目自身足够的数据时进行项目内预测。针对项目初期数据不足的情况，建议开发人员在克隆代码创建时预测其一致性维护需求，避免可能会发生一致性变化的克隆代码。

5.6.3 跨项目使用模式实验

本节将回答本章提出的第二个研究问题：在进行跨项目克隆一致性维护需求预测时，开发人员应该预测哪一类别的克隆实例（需要维护的实例或者不需要维护的实例），跨项目预测是否对两种状态的实例具有一致的预测效果？

实验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进行跨项目预测。采用和第 3 章和第 4 章相同的评价指标评估预测效果，即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，评估指标见本文第 3.7.1.2 节和第 4.7.1.2 节。

5.6.3.1 克隆创建实例实验结果

本节对克隆创建实例进行跨项目预测，四个系统上的实验结果如图 5-2 和图 5-3 所示。

（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跨项目预测，其预测结果如图 5-2 所示。从图中可以看出，四个系统的精确率和召回率可以达到较高的水平，其中精确率在 60.01% - 91.20% 之间，召回率 56.06% - 94.84% 之间。将实验结果与第 3 章的全属性实验进行对比（图 3-4 ），可以发现尽管四个系统的预测效果都有了不同程度的下降，但跨项目预测效果依然是可以接受的。

通过对比四个实验系统的实验结果，不同系统的预测效果不是一致的。发现 jEdit 系统的预测效果最好，其精确率在 89.1% - 91.2% 之间，召回率在 81.4% - 93.8% 之间。而对 jFreeChart 系统的预测效果最差，其精确率仅在 60% 左右，召回率可以达到 90% 以上。而系统 ArgoUML 和 Tuxguitar 的预测效果差不多，精确率在 75% 左右，召回率最高分别可达 76.6% 和 90.84%。分析原因可能是由于 jEdit 系统的训练集规模最大，模型训练最充分，而 jFreeChart 则与之相反。

与此同时，贝叶斯网络的阈值会影响预测效果。从图中可以看出，当阈值接近

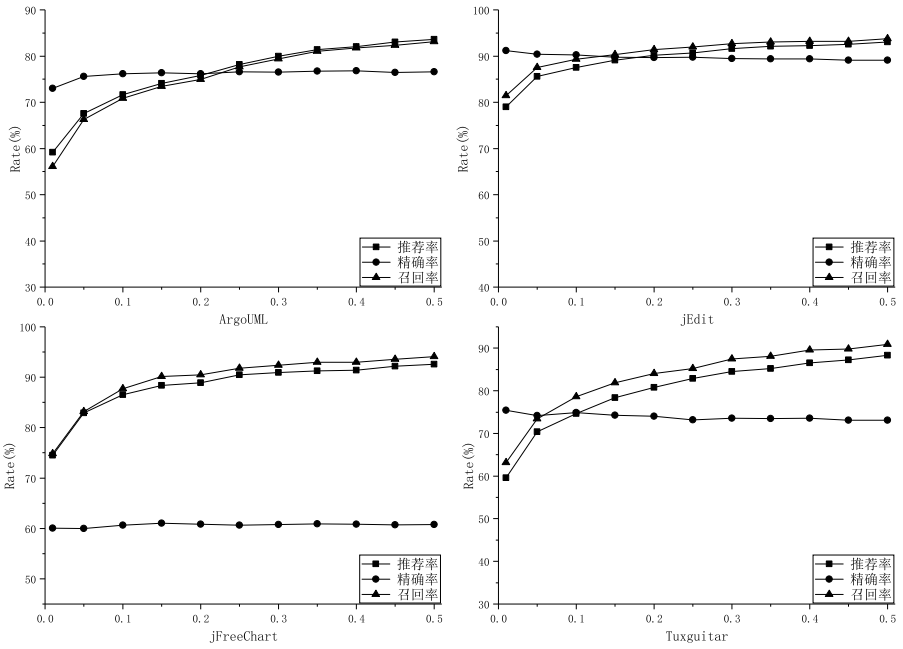


图 5-2 克隆创建实例的跨项目使用模式实验结果（不需维护）

Fig.5-2 The effectiveness of cross-project for usage on meeting creating consistency-free

于 0.5 时，跨项目的预测效果好，系统 ArgoUML 的精确率和召回率分别为 76.6% 和 83.1%，jEdit 为 89.1% 和 93.4%，jFreeChart 为 60.8% 和 94.8%，Tuxguitar 为 73.1% 和 90.8%。

在软件开发初期，当自身系统训练数据较少不足以训练模型的情况下，可以使用其它系统的数据对不需要一致性维护的克隆创建实例进行跨项目预测。所构建模型的预测能力会依赖于项目自身的特征，建议优先选用系统自身的数据进行一致性维护需求预测。同时，建议开发人员调整合适的阈值以达到最佳的预测效果。

（2）一致性维护需求实验

对于需要一致性维护的克隆创建实例，其跨项目的实验结果如图 5-3所示。从图中可以看出，四个系统的预测效果都比较差。其中，系统 ArgoUML 的精确率在 7.9% - 20.7% 之间，jEdit 的精确率略好，在 20.5% - 75.0% 之间，系统 jFreeChart 在 33.3% - 52.6% 之间，系统 Tuxguitar 为 44.3% - 51.9%。而所有系统的召回率均低于 18%。分析原因是需要维护的克隆创建实例往往会依赖于具体的系统，同时克隆创建实例的样本也不平衡，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 11.5% - 40.2%。因此，所训练的跨项目模型，无

法实际的预测需要一致性维护的克隆创建实例。

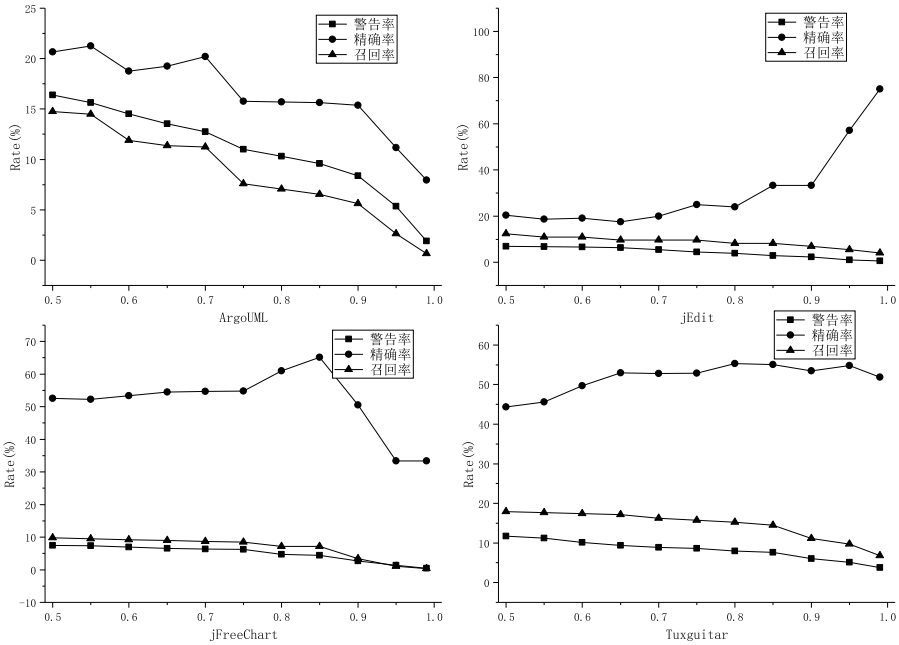


图 5-3 克隆创建实例的跨项目使用模式实验结果（需要维护）

Fig.5-3 The effectiveness of cross-project for usage on meeting creating consistency

同时，与第 3 章全属性实验进行对比（图 3-6 ），发现四个系统的跨项目预测效果下降得十分严重。因此，对需要一致性维护的克隆创建实例，强烈的依赖于具体的系统，不建议使用跨项目的方式对其进行一致性维护需求预测。

因此，对于需要一致性维护的克隆创建实例，本文不建议使用跨项目的方式对需要一致性维护需求预测。原因可能是由于克隆代码的一致性变化往往会依赖于具体的系统的特征，而跨项目之间的差异无法较好的训练相应的模型，因此建议开发人员选择自身数据进行训练和预测。在开发初期由于缺乏数据问题无法训练相应模型时，建议程序开发人员选择对不需要一致性维护的克隆实例进行跨项目预测。

（3）小结

综上，针对克隆创建实例的跨项目预测，实验结果表明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和和预测。

在软件开发初期，当自身系统的数据太少而不足以训练模型时，可以使用跨项

目预测的方式，对不需要一致性维护的克隆创建实例进行预测，仅仅允许此类的克隆实例产生，从而避免额外的一致性维护代价。由于系统具有较少的需要一致性维护的克隆创建实例，并且会依赖于具体系统的某些特征，对需要一致性维护的克隆创建实例的预测效果较差。因此，本文不建使用跨项目的方式对其进行一致性维护需求预测。

5.6.3.2 克隆变化实例的实验结果

本节对克隆变化实例进行跨项目一致性维护需求预测，在四个系统上对两种状态的变化实例的实验结果分别如图 5-4 和图 5-5所示。

（1）一致性维护自由实验

对不需要一致性维护的克隆变化实例，其跨项目一致性预测结果如图 5-4 所示。

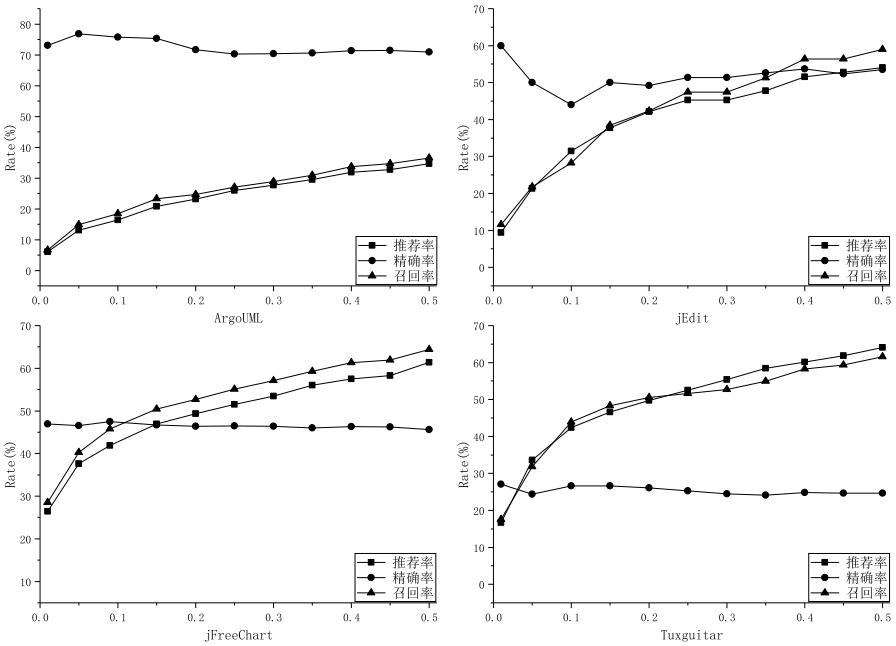


图 5-4 克隆变化实例的跨项目使用模式实验结果（不需维护）

Fig.5-4 The effectiveness of cross-project for usage on meeting changing consistency-free

从图中可以看出，跨项目预测在四个系统上预测效果较为一般。ArgoUML 系统的精确率最好，达到了 70.3% 以上，系统 jEdit 和 jFreeChart 的精确率在 50% 左右，而系统 Tuxguitar 的预测效果最差，仅在 25% 左右。同时，四个系统的召回率并不能令人满意。其中，系统 ArgoUML 的召回率最低（低于 36.46%），其它系统

的召回率最高可以达到 60% 左右。分析原因可能是克隆代码的变化情况较高的依赖于具体的系统的数据分布情况。

结合第 4 章项目内的预测结果（图 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维护需求预测。采用跨项目预测的情况下，所构建的模型无法较好地预测不需要一致性维护的克隆变化实例。

跨项目预测的效果是较为稳定的，阈值变化会影响到预测的效果，其中对召回率的影响大于对精确率的影响。从图中可以看出，推荐率和召回率具有正相关的关系，并受阈值的影响较大。跨项目预测的精确率与具体的系统有关，预测效果是较为稳定的，即精确率稳定在一个稳范围内。随着阈值的升高，所推荐的不需要克隆变化实例的数量变多，因而导致召回率的提高。

因此，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，不能很好地训练模型时，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）一致性维护需求实验

对需要一致性维护的克隆变化实例，跨项预测的实验结果如图 5-5 所示。

从图中可以看出，对需要一致性维护的克隆变化实例，预测效果也较为一般。Tuxguitar 系统的精确率最好，在 70% 左右，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 30% 左右。同时对于召回率来说，系统 Tuxguitar 和 jFreeChart 的召回率最低，最高分别可以达到 30% 和 40% 左右。jEdit 和 ArgoUML 的召回率较好，分别可以达到 50% 和 70% 左右。结合全属性实验的实验结果（图 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，再使用跨项目的形式进行预测。

跨项目预测的效果是较为稳定的，阈值变化会影响到预测的效果，其中对召回率的影响大于对精确率的影响。从图中可以看出，推荐率和召回率具有正相关的关系，并受阈值的影响较大。跨项目预测的预测效果是较为稳定的，即精确率稳定在一个稳范围内。跨项目预测的精确率与具体的系统有关。随着阈值的升高，所推荐的不需要克隆变化实例的数量变多，因而导致召回率的升高。

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，建议开发人员使用本文第 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

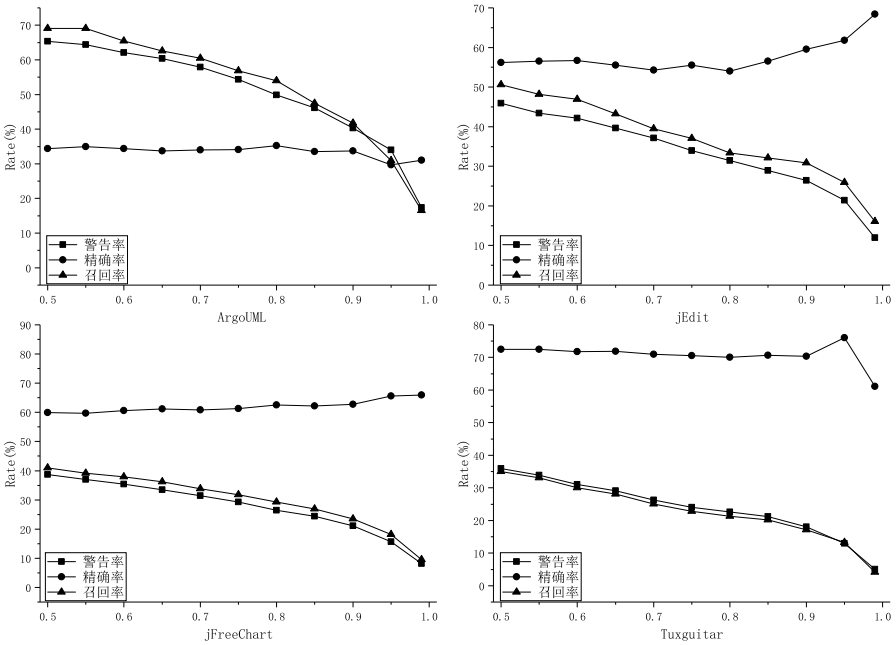


图 5-5 克隆变化实例的跨项目使用模式实验结果（需要维护）

Fig.5-5 The effectiveness of cross-project for usage on meeting changing consistency

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）小结

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。

在软件开发初期阶段，由于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（精确率）将依赖于数据集的分布情况。对于被测试的软件系统，需要尽量选择那些具有相似数据分布的测试系统，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。

5.6.3.3 讨论

综上所述，可以回答第二个研究子问题：

首先，由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，可以使用跨项目方式进行克隆一致性维护需求预测，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：（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，帮助快速的执行复制和粘贴操作。（2）克隆代码变化时，建议对需要一致性维护的克隆实例进行预测，帮助避免克隆一致性维护缺陷。

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。

5.6.4 跨项目训练集规模实验

在进行克隆代码一致性维护需求预测时，训练集的规模会影响预测效果。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维护需求。并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，训练数据的规模将如何影响跨项目的克隆一致性维护需求的预测效果？

在五种机器学习方法上进行了跨项目的数据集规模实验，并使用平均精确率（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”跨项目预测是指每次仅使用一个系统的数据作为训练集，在目标系统上进行跨项目预测，并对多个训练系统的预测结果进行平均，得出“一对一”的预测结果（使用“Average”表示，简写为 Ave）。然后，进行“多对一”跨项目预测。“多对一”跨项目预测是指使用三个系统的数据集共同作为训练集，并在目标系统上进行跨项目预测，得出多对一预测结果（使用“All”表示）。最后，进行“数据集扩大”实验，新增三个不同实验系统扩大跨项目预测的训练集（使用“Expansion”表示，简写为 Exp）。扩大训练集所采用的系统为 DNSjava、Carol 和 JabRef 三个实验系统。其中，选择了 34 个软件版

本的 DNSjava 系统，11 个版本的 Carol 系统和 19 个版本的 JabRef 系统。

对克隆创建实例和变化实例进行了实验评估，实验结果如表 5-5 和表 5-6 所示。表中“Precision”、“Recall”、“F-measure”分别指平均精确率、平均召回率和平均 F 值。

5.6.4.1 克隆创建实例的实验结果

对克隆创建实例进行跨项目预测，实验结果如表 5-5 所示。表中第 1 列和第 2 列分别为评估指标和机器学习方法，第 3 - 14 列为四个实验系统的实验结果。其中，“Ave”列表示“一对一”实验结果，“All”列表示“多对一”实验结果，“Exp”列表示扩大数据集实验结果。

表 5-5 克隆创建实例的跨项目数据集规模实验结果

Table5-5 The effectiveness of cross-project for training size on creating instances

评估指标	方法	To ArgoUML			To jEdit			To jFreeChart			To Tuxguitar		
		Ave	All	Exp	Ave	All	Exp	Ave	All	Exp	Ave	All	Exp
Precision(%)	BN	63.3	63.8	61.8	81.0	81.2	79.6	55.6	57.5	57.0	63.3	64.8	60.7
	NB	62.9	63.1	66.4	79.2	78.3	78.5	50.2	48.6	52.3	60.6	61.5	54.8
	SVM	65.7	59.4	62.8	78.3	78.3	78.2	35.8	35.8	41.0	50.6	50.6	60.0
	KNN	62.9	62.9	60.1	81.5	78.6	78.0	61.7	64.8	49.6	58.7	58.1	56.7
	DT	68.6	62.9	62.1	78.2	79.6	81.6	53.0	56.6	42.1	59.4	70.4	54.1
Recall(%)	BN	67.0	67.5	68.7	75.2	84.4	80.3	60.6	60.2	60.0	67.4	69.8	64.2
	NB	66.9	69.3	57.7	75.4	74.7	49.1	56.4	54.0	44.5	64.3	64.7	48.6
	SVM	80.9	77.1	72.1	88.5	88.5	88.2	59.8	59.8	59.2	71.1	71.1	70.2
	KNN	68.1	68.7	62.2	76.2	74.1	71.7	63.5	64.9	55.2	65.5	66.3	62.8
	DT	76.4	74.0	66.7	80.3	84.4	70.8	59.5	59.9	50.2	68.8	73.1	56.8
F-measure(%)	BN	64.5	65.4	64.7	76.8	82.6	79.9	51.6	50.8	51.5	63.1	65.0	62.0
	NB	64.5	65.6	60.7	77.4	76.4	58.0	48.8	48.9	40.0	61.7	62.7	50.8
	SVM	72.4	67.1	66.3	83.1	83.1	82.9	44.8	44.8	44.7	59.1	59.1	60.1
	KNN	65.1	65.3	61.1	78.1	76.2	74.6	57.9	60.9	49.4	60.1	60.3	58.9
	DT	70.3	66.7	64.1	78.9	81.8	75.1	48.4	50.3	44.1	61.2	68.3	55.3

在数据集规模实验中，“一对一”实验中的每一个系统的训练集规模都是最小的，仅使用一个项目作为训练集。同时，“多对一”预测的实验中训练集规模居中，使用三个项目作为训练集。而“数据集扩大”实验中所使用训练集的规模最大，使用 6 个项目作为训练集。

首先，对于某一具体的测试系统来说，根据其训练集的规模大小，发现跨项目

预测效果和训练集规模并不存在明显的正相关关系，即训练集的大小并不能明显的影响跨项目的预测能力。通过对比同一系统的“Ave”、“All”和“Exp”列，在 4 个系统中全部的平均 F 值中，所得到预测结果相差不大。分析其原因，可能是由于在跨项目预测中，项目之间的差异导致所训练的模型不够完善。因此，训练集规模的大小在跨项目预测中不能起到决定性作用。

其次，在预测跨项目的克隆创建实例时，预测效果可能会依赖于某具体的系统。通过对比不同系统的跨项目预测结果，可以发现 jEdit 系统的实验效果最好，其 F 值在 80% 左右；ArgoUML 系统和 Tuxguitar 系统的跨项目预测效果次之，其 F 值在 60% - 70% 左右；jFreeChart 系统的预测效果最差，F 值在 50% 左右。因此，在进行跨项目预测时，可以考虑在保证一定规模训练数据的基础上，对训练项目进行进一步的优选，选择与目标项目相似的项目作为训练集，提升跨项目的预测效果。但是，在跨项目预测中如何对项目进行优选仍需进一步的研究。

5.6.4.2 克隆变化实例的实验结果

对克隆变化实例进行跨项目预测，实验结果表 5-6 所示，其中“All”列表示“多对一”实验结果，“Ave”列表示“一对一”实验结果，“Exp”列表示扩大数据集实验结果。

在数据集规模实验中，“一对一”实验中的每一个系统的训练集规模都是最小的，仅使用一个项目作为训练集。同时，“多对一”预测的实验中训练集规模居中，使用三个项目作为训练集。而“数据集扩大”实验中所使用训练集的规模最大，使用 6 个项目作为训练集。

首先，根据其训练集的规模大小，发现跨项目预测效果和训练集规模并不存在明显的正相关关系。但是，对不同的系统而言，训练集的大小对预测效果的影响并不相同。对于 jEdit 系统、Tuxguitar 系统 jFreeChart 系统来说，跨项目预测的最好的预测结果总是出现在“All”列和“Exp”列中，数据集规模对预测有积极的作用；但是，对比三组实验的实验效果，预测效果相差不大，并不能起到决定性的作用。对于 ArgoUML 系统来说，预测效果与数据集关系和所采用的机器学习方法都有关系，并没有一致性的结论。分析其原因，可能是由于在跨项目预测中，项目之间的差异导致所训练的模型不够完善。因此，训练集规模的大小在跨项目预测中并不能起到决定性作用。

其次，通过对比不同系统的跨项目预测结果，预测效果可能会依赖于某具体的系统。例如，对 jEdit 系统和 jFreeChart 的预测效果最好，除支持向量机方法其平均 F 值大都高于 50%；对 ArgoUML 系统和 Tuxguitar 系统的预测效果较差，其平均 F 值大都低于 50%。因此，项目自身的特性可能会更多的影响跨项目的预测效

表 5-6 克隆变化实例的跨项目数据集规模实验

Table5-6 The effectiveness of cross-project for training size on changing instances

评估指标	方法	To ArgoUML			To jEdit			To jFreeChart			To Tuxguitar		
		Ave	All	Exp	Ave	All	Exp	Ave	All	Exp	Ave	All	Exp
Precision(%)	BN	59.5	59.1	66.0	48.0	54.9	58.5	55.2	53.7	54.8	57.5	60.2	60.7
	NB	59.6	62.9	55.6	54.1	54.7	47.8	54.8	56.6	51.7	54.8	55.6	55.8
	SVM	10.6	10.6	62.7	25.4	26.0	55.8	27.6	61.1	59.7	39.0	52.4	56.2
	KNN	60.5	62.4	58.6	51.7	54.2	52.2	51.5	53.3	52.9	61.7	62.3	66.1
	DT	61.6	62.9	60.1	49.0	64.8	54.0	48.5	54.6	52.0	61.6	63.8	64.6
Recall(%)	BN	51.3	47.1	43.8	48.2	54.7	58.5	52.2	51.2	54.9	38.9	41.8	45.2
	NB	49.6	46.4	35.3	54.1	54.7	47.8	53.1	53.8	50.4	39.8	35.3	39.3
	SVM	34.3	32.6	42.4	49.1	50.9	54.1	50.3	59.6	58.7	58.1	32.2	38.4
	KNN	53.3	53.4	49.4	51.2	54.1	52.2	49.8	50.9	51.3	44.9	45.2	51.7
	DT	38.7	37.0	52.9	50.1	60.4	54.1	51.6	56.7	49.6	55.0	38.7	49.4
F-measure(%)	BN	49.1	47.4	39.8	45.6	54.6	58.4	50.1	50.7	54.9	38.3	44.1	48.2
	NB	48.1	45.0	36.1	53.3	54.5	47.8	52.6	53.3	50.5	40.7	36.1	42.1
	SVM	16.0	16.0	38.4	33.7	34.4	48.9	36.0	52.2	58.8	45.7	32.0	40.7
	KNN	45.3	54.6	50.6	44.6	54.0	52.2	45.9	50.5	51.4	47.4	47.8	54.5
	DT	29.6	27.3	54.3	44.1	56.7	53.9	40.7	48.9	49.2	50.0	38.3	52.3

果。可以考虑在保证一定规模训练数据的基础上，对训练项目进行进一步的优选，选择与目标项目相似的数据作为训练集，从而缩小项目之间的差异，提升跨项目的预测效果。但是，在跨项目预测中如何对项目进行优选仍需进一步的研究。

5.6.4.3 讨论

综上所述，可以回答第三个研究子问题：首先，在跨项目的克隆一致性维护需求预测中，训练集的大小会影响到预测的效果，但是训练集的规模并不能起到决定性的作用。其次，对某特定的系统来讲，项目自身的特征可能会更多地影响到预测效果，但是系统的具体特征还有待更进一步的研究。最后，在保证一定规模训练数据的基础上，对训练项目进行进一步的优选可能会提升跨项目的预测效果，但是这也需要进一步的研究。

5.6.5 软件开发过程中的克隆一致性维护需求预测

本节将根据本文的研究内容，结合软件开发过程帮助程序开发人员在实践中运用克隆代码一致性维护需求预测方法。

通过对比克隆创建实例和克隆变化时的预测结果，发现预测效果的好坏会依赖于具体的系统，预测效果也会偏向于数量较多的克隆实例的类别上。具体来说，克隆创建实例的预测效果要好于克隆变化实例的预测效果。而在这两个预测中，系统 jEdit 就几乎都具有最差的预测结果。原因在于 jEdit 的克隆实例的规模太小，不能对其自身的预测模型进行较好地训练，从而导致预测效果最差。因此，本文建议开发人员在进行克隆一致性维护需求预测时，需要在软件演化一定版本后收集到足够的训练数据后再进行训练和预测。在软件开发初始阶段，由于系统中缺少训练数据，可以使用跨项目预测的方法进行预测。在预测时，建议在克隆代码创建时进行预测，尽量避免会导致一致性变化的克隆代码，并且建议预测数量较多的克隆实例的类别，即对克隆创建实例预测不需要一致性维护的实例。当确有必要在克隆变化时进行跨项目预测时，尽管预测效果不能令人满意，但是预测的精确度依然可以达到一个不错的水平，但需要谨慎地选择所预测的克隆变化实例的类别，建议预测需要一致性维护克隆变化实例，从而达到较好的预测效果。

在软件开发的初始阶段，主要任务是快速的进行软件开发，并同时向系统中添加新的功能。这种特点可能会促使程序开发人员大量的进行复制粘贴操作，复用既有的代码进行快速开发。因此，本文建议开发人员在软件开发的初始阶段使用克隆代码创建时的一致性维护预测方法。进一步地，由于要向软件中引入新的功能，因此建议使用该模型预测满足一致性维护自由的克隆代码，仅仅避免那些可能会引发一致性变化的克隆代码，进而快速的开发软件。但是由于在最初始的阶段缺乏项目自身的数据，可以使用跨项目预测方法用于克隆代码创建时的一致性维护需求预测。

在软件开发的中间阶段，主要任务是添加新功能并修复包括克隆一致性违背缺陷在内的一些相关缺陷。因此，本文建议开发人员应该同时考虑两者来预测克隆创建实例和克隆变化实例。在经过几个版本的演化后，大多数的系统已经具有充分的克隆创建实例训练机器学习模型，可以使用项目内的预测方式预测需要一致性维护的克隆创建实例，从而避免引入额外的维护代价。同时，此阶段系统可能存在较少的克隆变化实例，因此可以采用跨项目预测的方式预测克隆变化实例的一致性维护需求，帮助避免可能导致的一致性违背缺陷。

在软件开发的最后阶段，主要任务是维护系统的功能并且修复一些缺陷问题。这可能会导致对克隆代码的大量修改，进而引发克隆代码的一致性维护问题。因此，本文建议开发人员对克隆变化实例进行克隆一致性维护需求预测。由于软件系统已经经过了若干版本的演化，其项目自身已经具备了足够的克隆变化实例，建议使用项目内的一致性维护需求预测。同时，这个阶段主要需要避免由于代码变

化所导致的克隆代码的一致性违背缺陷，因此可以考虑仅针对需要一致性维护的克隆变化实例进行预测，同时建议开发人员采取措施保证克隆代码的一致性，例如克隆代码一致性维护方法[128,132]。

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，在软件开发过程中对克隆代码进行一致性维护需求预测，本文还设计并实现了一个克隆代码的一致性维护需求插件。

本文所设计的插件是基于 eclipse 的插件环境实现的，因此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。在软件开发过程中，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体系结构

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 5-6 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

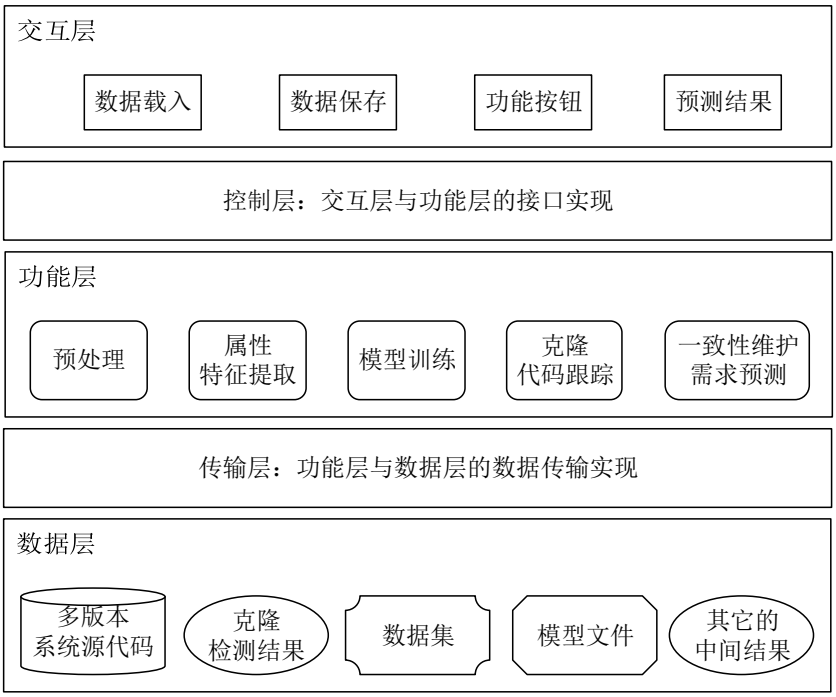


图 5-6 克隆一致性维护需求预测插件的体系结构

Fig.5-6 The architecture for the plug-in of clone consistency-requirement prediction

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。

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 5-7 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

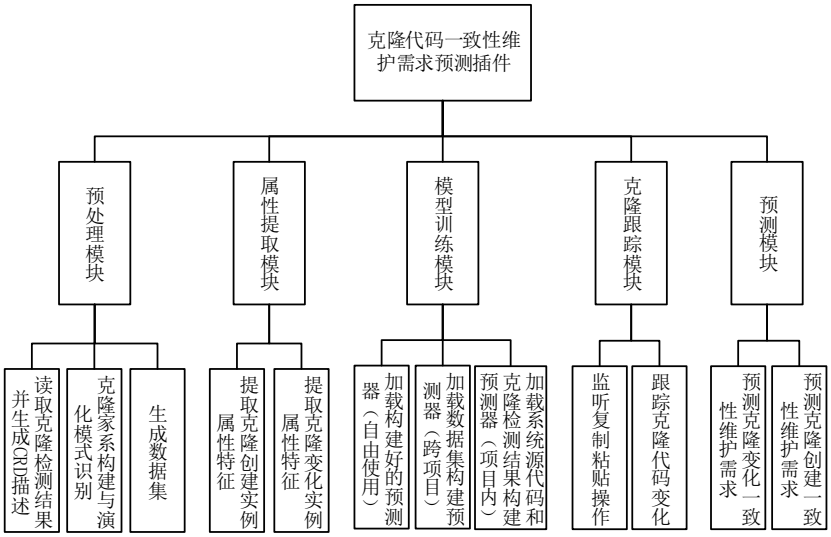


图 5-7 克隆一致性维护插件的功能模块

Fig.5-7 The modules for clone consistency-requirement prediction

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。方法见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 AST[1]。通过遍历语法树访问相应关键

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

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，根据对其属性的描述计算相应演化属性。

（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在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，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提前提取出来。当克隆变化实例产生时，直接使用已经提取的属性，从而降低属性提取的时间。

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]。

本文所设计的插件运行截图如图 5-8 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[3]。

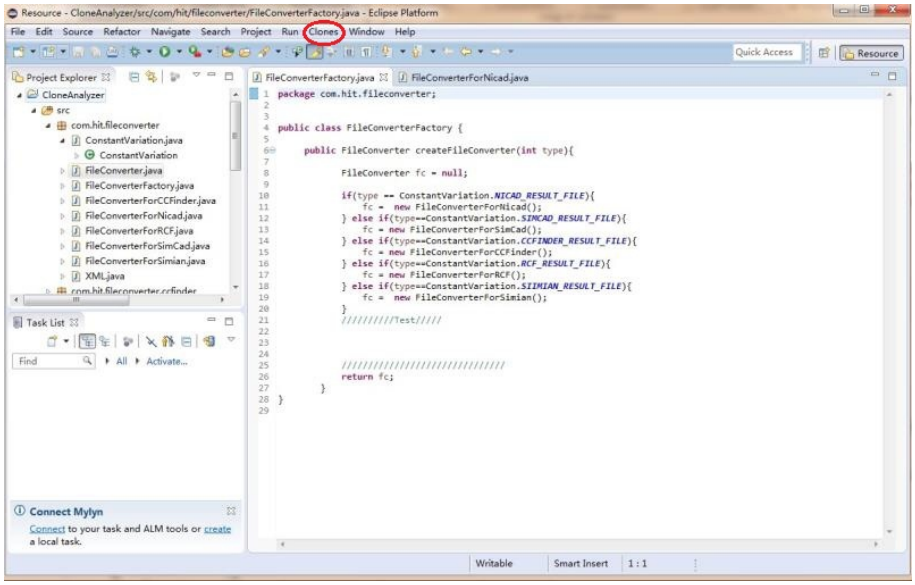


图 5-8 插件安装完成后 eclipse 界面

Fig.5-8 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 5-9 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 5-10 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。

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

[3]本文所设计的插件见：https://github.com/zhangfanlong/CloneControlPlug-in。欢迎开发人员在实际开发过程中使用本文的所提出的克隆代码一致性维护需求预测方法，并期望给出建议与意见。

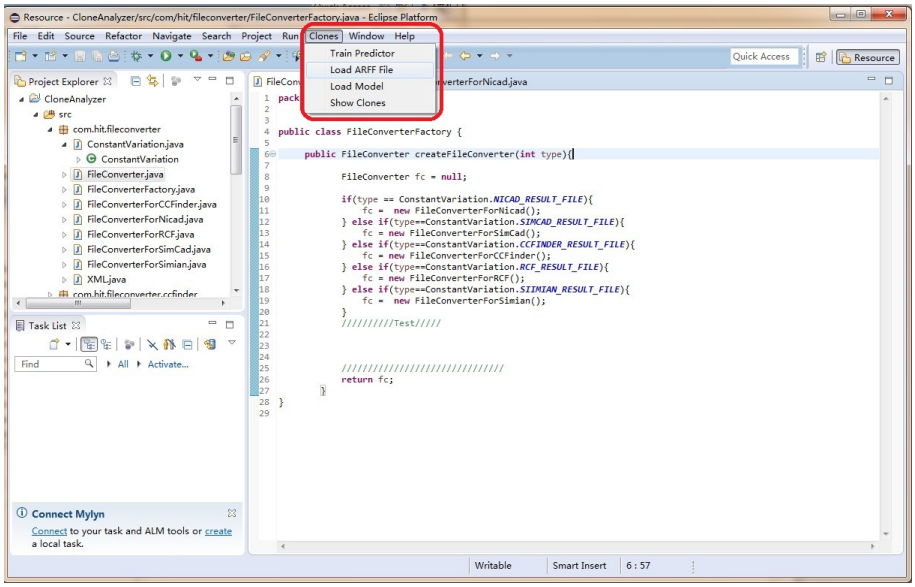


图 5-9 插件加载预测模型的截图

Fig. 5-9 The screen-shot for loading the models of clone consistency prediction

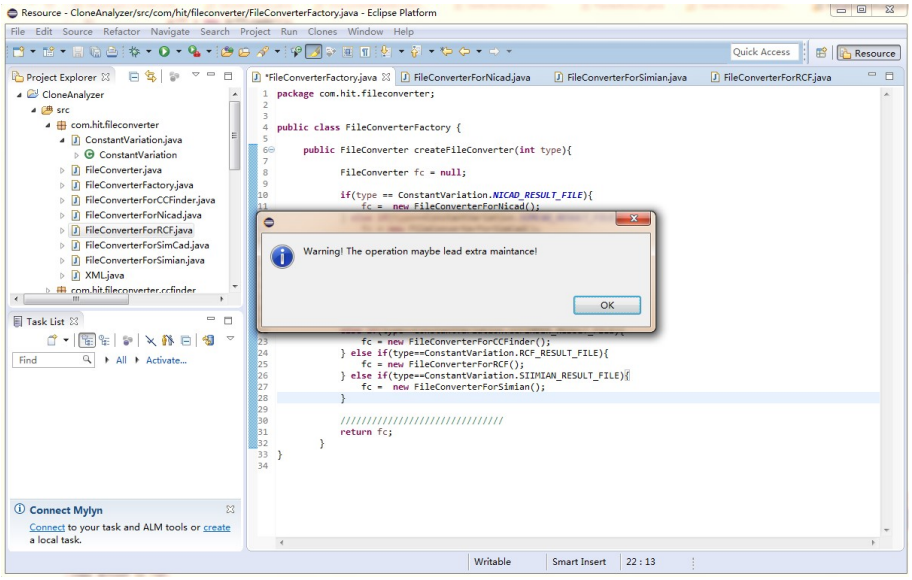


图 5-10 需要一致性维护的预测结果示例

Fig. 5-10 The screen-shot for a prediction example that meeting consistency

5.8 本章小结

在软件开发初期，由于克隆代码实例较少无法训练机器学习模型进行克隆一致性维护需求预测。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。训练集的规模会对跨项目的预测能力产生影响，但并不具备决定性的作用，无法弥补项目之间的差异性。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实际开发过程中同步的开发和维护克隆代码，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，会导致软件越来越难以理解和维护，克隆代码成为了影响软件质量的一个重要因素。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，结合软件演化和软件开发过程，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）针对演化中的克隆代码难于理解和分析的问题，提出了一个基于 X-means 聚类的克隆代码演化特征分析方法，为分析克隆代码演化规律提供了有效的手段。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，并不会频繁的发生变化。同时，也存在相当数量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。上述结论不仅验证了克隆代码一致性需求预测的必要性，还为后续克隆代码的一致性维护需求预测研究奠定了基础。

（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时一致性维护需求预测方法，从规避克隆代码产生的角度降低克隆代码的维护代价。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的分类问题。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，提出了一种克隆代码变化一致性维护需求预测方法，实现了克隆代码的同步开发与维护。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的分类问题。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跨项目实证研究，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帮助提高跨项目预测能力的建议。结合软件开发过程设计并实现了一个基于 eclipse 的克隆代码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）在克隆变化的一致性维护需求预测中，所构建模型的预测能力仍有一定的改进空间。在未来的研究中，拟通过结合程序分析技术提取新的属性，用于表示克隆代码的真实变化以及变化所在的克隆组，进一步提高所构建模型的准确率和召回率，更好地预测克隆代码变化的一致性维护需求。

（2）在跨项目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、构建新的跨项目预测模型等方法，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代码的一致性维护需求。

（3）本文方法可以帮助程序开发人员预测克隆代码的一致性维护需求，并提醒开发人员对克隆代码进行一致性维护。但是，本文没有直接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，自动地维护克隆代码的一致性，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[C] // Reverse engineering, 1996. Proceedings., International Conference on. 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[C] // Software Maintenance, 2000. Proceedings., International Conference on. 2000: 214–225.

[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] 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.

[43] 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.

[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] BARBOUR L, KHOMH F, ZOU Y. Late propagation in software clones[C] // Software Maintenance (ICSM), 2011 27th IEEE International Conference on. 2011 : 273 – 282.

[60] 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.

[61] INOUE K, HIGO Y, YOSHIDA N, et al. Experience of finding inconsistently-changed bugs in code clones of mobile software[C] // Proceedings of the 6th International Workshop on Software Clones. 2012 : 94 – 95.

[62] 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.

[63] 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.

[64] 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.

[65] 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.

[66] BAKOTA T, FERENC R, GYIMOTHY T. Clone smells in software evolution[C] // 2007 IEEE International Conference on Software Maintenance. 2007 : 24 – 33.

[67] HARDER J, TIARKS R. A controlled experiment on software clones[C] // Program Comprehension (ICPC), 2012 IEEE 20th International Conference on. 2012 : 219 – 228.

[68] LOZANO A, WERMELINGER M. Assessing the effect of clones on changeabil-

ity[C] // Software Maintenance, 2008. ICSM 2008. IEEE International Conference on. 2008 : 227 – 236.

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

[70] 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.

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

[72] KAPSER C J, GODFREY M W. Supporting the analysis of clones in software systems[J]. Journal of Software Maintenance and Evolution: Research and Practice, 2006, 18(2) : 61 – 82.

[73] 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.

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

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

[76] HANJALIC A. ClonEvol: Visualizing software evolution with code clones[C] // Software Visualization (VISSOFT), 2013 First IEEE Working Conference on. 2013 : 1 – 4.

[77] 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.

[78] VOINEA L, TELEA A C. Visual clone analysis with SolidSDD[C] // Software Visualization (VISSOFT), 2014 Second IEEE Working Conference on. 2014 : 79 – 82.

[79] 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.

[80] HIGO Y, KAMIYA T, KUSUMOTO S, et al. Method and implementation for in-

vestigating code clones in a software system[J]. Information and Software Technology, 2007, 49(9): 985–998.

[81] 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.

[82] 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.

[83] 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.

[84] 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.

[85] 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.

[86] 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.

[87] 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.

[88] ZHANG Y, BASIT H 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.

[89] KERIEVSKY J. Refactoring to Pattern[J]. Journal of Software Maintenance and Evolution: Research and Practice, 2009, 15(2): 18–39.

[90] 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.

[91] 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.

[92] 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.

[93] 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.

[94] 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.

[95] 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.

[96] 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.

[97] 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.

[98] 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.

[99] TSANTALIS N, MAZINANIAN D, KRISHNAN G P. Assessing the refactorability of software clones[J]. IEEE Transactions on Software Engineering, 2015, 41(11) : 1055 – 1090.

[100] 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.

[101] 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.

[102] 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.

[103] 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.

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

[105] 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.

[106] 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.

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

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

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

[110] 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.

[111] 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.

[112] 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.

[113] 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.

[114] 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.

[115] 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.

[116] 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.

[117] 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.

[118] 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.

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

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

[121] 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.

[122] 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.

[123] 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.

[124] 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.

[125] 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.

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

[127] 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.

[128] 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.

[129] 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.

[130] 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.

[131] 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.

[132] 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.

[133] 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.

[134] TAIRAS R A. Representation, analysis, and refactoring techniques to support code clone maintenance[D]. [S.l.] : Citeseer, 2010.

[135] DANG Y, ZHANG D, GE S, et al. Transferring code-clone detection and analy-

sis to practice[C] // Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track. 2017 : 53 – 62.

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

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

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

[139] 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, Measurement, Computer, Communication and Control (IMCCC), 2013 Third International Conference on. 2013 : 848 – 853.

[140] 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.

[141] 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.

[142] WANG X, DANG Y, ZHANG L, et al. Predicting consistency-maintenance requirement of code clonesat copy-and-paste time[J]. IEEE Transactions on Software Engineering, 2014, 40(8) : 773 – 794.

[143] RAMAMOORTHY C V, PRAKASH A, TSAI W T, et al. Software engineering: Problems and perspectives.[J]. Computer, 1984, 17(10) : 191 – 209.

[144] JONES C. Software metrics: good, bad and missing[J]. Computer, 1994, 27(9) : 98 – 100.

[145] LEVENSHTEIN V I. Binary codes capable of correcting deletions, insertions, and reversals[C] // Soviet physics doklady : Vol 10. 1966 : 707 – 710.

[146] NAVARRO G. A guided tour to approximate string matching[J]. ACM computing surveys (CSUR), 2001, 33(1) : 31 – 88.

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

[148] PEARL J. Bayesian networks: A model of self-activated memory for evidential

reasoning[C] // Proceedings of the 7th Conference of the Cognitive Science Society, 1985. 1985 : 329 – 334.

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

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

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

[152] SALZBERG S L. C4. 5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993[J]. Machine Learning, 1994, 16(3) : 235 – 240.

攻读博士学位期间发表的论文及其他成果

（一）已发表的学术论文

[1] Zhang Fanlong, Khoo Siau-Cheng, Su Xiaohong*. Predicting Change Consistency in A Clone Group[J]. Journal of Systems and Software. 134(2017), 105-119.（已发表, DOI:10.1016/j.jss.2017.08.045, SCI 收录, 5-Year IF=2.619, 对应第 4 章）

[2] Zhang Fanlong, Khoo Siau-Cheng, Su Xiaohong*. Predicting Consistent Clone Change[C]. Proceeding of the 27th International Symposium on Software Reliability Engineering (ISSRE), Ottawa, Canada, 2016: 353-364.（已发表, DOI:10.1109/ISSRE.2016.11, EI:20170803379101, 对应第 4 章）

[3] Zhang Fanlong, Khoo Siau-cheng, Su Xiaohong*. Machine-Learning Aided Analysis of Clone Evolution[J/OL]. Chinese Journal of Electronics(2017-09-28). http://kns.cnki.net/kcms/detail/10.1284.TN.20170928.1353.002.html.（已发表, DOI:10.1049/cje.2017.08.012, SCI 收录, 2016 年 IF=0.513, 对应第 2 章）

[4] 苏小红, 张凡龙*. 面向管理的克隆代码研究综述[J/OL]. 计算机学报 2017(2017-08-24). On Publishing: No.120, Vol.40. http://kns.cnki.net/kcms/detail/11.1826.TP.20170728.1305.046.html.（已发表, EI 收录, 对应第 1 章）

[5] 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). 2017, 24(2):10-18.（已发表, DOI:10.11916/j.issn.1005-9113.15316, 对应第 2 章）

[6] 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, 对应第 3 章）

[7] Su Xiaohong*, Zhang Fanlong, Xia Li, et al. Functionally Equivalent C Code Clone Refactoring by Combining Static Analysis with Dynamic Testing[C]. Proceeding of the International Conference on Soft Computing Techniques and Engineering Application. 2014: 247-256.（已发表, DOI:10.1007/978-81-322-1695-7_28, EI:20151600752325）

（二）审稿中的学术论文

[1] 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. （大修, SCI 收录, 2016 年 IF=0.956, 对应第 5 章）

[2] Zhang Fanlong, Khoo Siau-cheng, Su Xiaohong*. Improving Maintenance-Consistency Prediction During Code Clone Creation[J]. Software Quality Journal.（在审, SCI 收录, 2016 年 IF=1.816, 对应第 3 章)

[3] Zhang Fanlong , Su Xiaohong*. Clone Cross-Project Consistency Prediction[J]. Journal of Systems and Software. （在审, SCI 收录, 5-Year IF=2.619, 对应第 5 章）

（三）与他人合作的学术论文

[1] Yuan Yue, Zhang Fanlong, Su Xiaohong*. CloneAyz: An Approach for Clone Representation and Analysis[C]. Proceeding of the 3rd International Conference on Information Science and Control Engineering (ICISCE), 2016: 252-256.（已发表, DOI:10.1109/ICISCE.2016.63, EI:20165003106894）

（四）参与的科研项目

[1] 苏小红等. 建筑全性能联合仿真平台内核开发. “十三五”国家重点研发计划课题. 课题编号：2017YFC0702204.

[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. 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.

[4] 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.

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我无微不至的关怀。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使我受益匪浅。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。

衷心感谢我的女朋友王子萍！感谢她在我博士最后阶段的陪伴，在我沮丧、失落时一次次地鼓励和安慰我！

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 计算机科学与技术学科 ，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。

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。

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 访学，Visiting Student。

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。

在攻读博士学位期间，发表（含在审）论文 11 篇，其中 SCI 期刊 5 篇，国内一级学报 2 篇，国际会议 3 篇，国内期刊 2 篇。