中山大学硕士学位论文

代码修改影响分析及修改周期预测方法研究

Research on the Methods of Code Change Impact Analysis and Change Period Prediction

学位申请人: _____

指导教师: _____

专业名称: 软件工程

答辩委员会主席(签名): _____

答辩委员会委员(签名): _____

_____

_____

_____

二零一九年四月九日

论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究作出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：

学位论文使用授权声明

本人完全了解中山大学有关保留、使用学位论文的规定，即：学校有权保留学位论文并向国家主管部门或其指定机构送交论文的电子版和纸质版，有权将学位论文用于非赢利目的的少量复制并允许论文进入学校图书馆、院系资料室被查阅，有权将学位论文的内容编入有关数据库进行检索，可以采用复印、缩印或其他方法保存学位论文。

学位论文作者签名：　　导师签名：

日期：　年　月　日　　日期：　年　月　日

论文题目： 代码修改影响分析及修改周期预测方法研究
专 业： 软件工程
硕 士 生：
指导教师：

摘要

软件维护作为保障软件质量的有效手段，是软件生命周期中最耗人力和时间的阶段。由于代码实体之间关系复杂，当开发人员进行代码修改工作时，一部分代码的修改可能会影响到软件中的其他部分。如何确定代码影响范围，让开发人员能快速确定需要修改的软件实体，成为影响代码修改效率的重要因素。另外，当开发人员完成代码修改后，代码需要通过审核才能提交至代码库中。审核不通过，需要开发人员再次对代码进行修改。“修改-审核”的次数直接关系着整个代码修改周期的长短，从而影响代码修改的完成效率。本文主要关注于如何辅助开发人员快速和准确确定修改的影响范围，以及预测修改周期的长短。

随着开源软件的飞速发展，互联网上积累了大量软件维护数据（如：代码修改提交数据）。这些维护数据记录了已有软件修改中元素之间如何互相影响，以及软件修改的审核过程。利用这些数据中隐含的规律，能够帮助更准确预测修改的影响范围以及修改周期。本文针对如何确定代码修改的影响范围，提出一种基于历史修改模式的影响分析方法。该方法收集了大量开源项目的历史修改提交数据，通过代码修改相似度及修改目标相似度筛选与当前修改任务相似的提交；再利用相似提交中的修改传播模式指导影响分析。实验结果表明，我们的方法对传统影响分析结果具有很好的提升效果，在多个项目上影响集前5和前10的召回率、精确率平均提升超过15%。针对代码修改周期预测问题，本文提出一种基于可判别性特征的修改周期预测方法，该方法通过代码修改过程中“修改-审核”的次数衡量修改周期的长短。具体地，我们从代码修改数据和审查数据中提取多维度的可判别特征，再结合机器学习方法，训练预测模型。我

们在两个真实的开源项目的提交数据集中对模型进行了测试，实验结果表明，我们的预测模型在修改周期长短的预测上具有较高的准确性，对开发人员提前预估代码修改和审核时间有较大的帮助。

关键词：影响分析，代码提交，代码审核，修改周期，机器学习

Title: Research on the Methods of Code Change Impact Analysis and Change Period Prediction

Major: Software Engineering

Name:

Supervisor:

Abstract

As an effective method to guarantee the quality of software, software maintenance is the most labor intensive and time consuming work in the software life cycle. When developers make code modification, some code modifications may affect other entities of the software due to the complex relationships between code entities. How to determine the scope of the modification, so that developers can quickly determine the software entities that need to be modified, becomes an important factor affecting the efficiency of code modification. Furthermore, the code changes made by developers needed to be reviewed before integrating the changes to software depository. If the code change submitted by developers is not accepted, the code change will be sent back to developers. The number of "modification-review" is directly related to the code modification time and efficiency. In this papers, we study the scope and time cycle of the code modification, and proposed some solutions to help developers improve software maintenance efficiency. We focus on how to assist developers to quickly and accurately determine the scope of impact of code change, as well as the change period.

With the rapid development of open source software, the Internet has accumulated a huge amount of software maintenance data, such as code change commits. These data record how the elements interact with each other in modification and software review process. Using the rules hiding in these data can help to more accurately predict the scope of the code change and the change period. In this paper, we proposed an impact analysis method based on historical change mode. We collected a large number of historical commit data from open source projects. Then, we search the similar commits based on the similarity of code change fragment and comment (modify target). Finally, we use the change mode of commits to optimize the impact set. Our results indicated that our method have better effect than other impact analysis tools in term of recall and

precision with more than 15% improvement on average. Regarding the predicion of code change time cycle, our paper proposed a method based on multi-dimension discriminative features. We extracted multi-dimensional features from code modification data, and then employ machine learning methods to train predictive models. We tested the model on the datasets from two open source projects. The results indicated that our model achieved high accuracy in predicting the length of the code change period.

Keywords: Impact Analysis, Code Commit, Code Review , Code Change Period , Machine Learning

目 录

第 1 章 综述

随着互联网与计算机的发展，已经产生了数以亿万计的软件项目。以在开源软件库Github[①]上托管的项目为例，2018年官方统计[②] 数量已超过9600000 个，且增速达到40%。海量的软件项目也引出了繁重的软件维护工作，对开发人员提出了巨大的挑战。因此，如何辅助开发人员在软件维护过程更高效的完成代码修改工作，成为软件维护领域的研究热点。开发人员在修改代码过程中，会对软件中其他代码实体产生潜在的影响，必需对这些代码做相应的修改。同时，代码修改工作通常需要经过多次“修改-审核”的过程，才能最终完成代码修改任务。因此，分析代码修改会产生的影响范围以及预估代码修改的完成周期对提高软件维护的效率具有重要价值。大数据背景下，如何从软件维护过程中产生的数据里挖掘出有用信息，用于辅助后续的代码修改任务，成为非常关键的研究问题。在版本控制系统中，存储了海量软件项目演化过程中产生的代码修改数据，这些数据以提交(Commit)的形式保存。另外，在代码审核过程中也存在大量代码修改和审核信息。因此，本文针对代码修改的影响分析以及代码修改的完成周期进行深入研究，通过挖掘软件维护过程中的历史数据，辅助代码修改影响分析和代码修改的完成周期预测，提高代码修改任务的效率和质量。

1.1 论文研究背景与意义

软件的可维护性是软件固有的重要特性。软件维护是整个软件生命周期中最关键的一环，占据着70%以上的比重，其主要任务是迎合市场和用户的新需求、修复软件运行过程中的存在的错误、以及对软件性能的优化。软件维护工作被认为是软件生命周期中，最困难和最费人力的工作[1]。软件维护过程中的核心是代码修改工作，由于软件系统的整体性以及系统中各部件的相互依赖关系，代码修改将不可避免地对修改以外的部分产生影响。为了预估软件维护和代码修改过程的影响范围和程度，修改影响分析成为代码修改任务中的重要一步[2]。另外，代码修改任务中，开发人员完成代码修改后的下一步是代码审核，

① Github, https://github.com
② Github, https://octoverse.github.com

通常代码修改需要经过多次“修改－审核”的环节才能最终完成，多次“修改-审核”对代码修改工作的时间周期有直接的影响。项目管理人员和开发人员提前对代码修改任务的所需时间进行预估，调整人力及物力的安排，能有效降低软件维护工作的成本。随着软件系统变得越来越庞大和复杂，修改影响分析和修改完成周期的预测是软件维护过程中提高维护效率和质量的有效方法。

同时，我们调研发现，大数据背景下，通过挖掘历史数据中的有效信息，再根据历史信息对研究对象进行分析的研究方式得到了广泛的应用，而且研究结果往往有较高的适用性和准确性[3–5]。在ICSE、ASE 和FSE 等计算机软件工程顶级会议中，关于数据挖掘在软件工程中应用的研究也吸引了广泛的关注。软件仓库中记录着软件演化的完整历史数据，包括：程序运行数据，缺陷跟踪数据，历史代码修改数据，代码审查数据。这些数据可用于挖掘重要信息，例如项目如何演变[6,7]，开发人员如何合作[8,9]，代码修改可能影响的范围[10,11] 等。如图1-1所示，我们总结了在软件工程研究问题中运用数据挖掘方法的总体流程。当前，开源项目广泛托管在版本控制系统中、开源代码审核软件也应用更加广泛，使得软件维护和演化的相关数据获取更加便利。同时，机器学习取得飞速的发展，并广泛应用在各个领域中，对各领域的研究工作起到推动作用。在软件工程领域，结合机器学习方法的研究工作，也取得了丰硕的成果。开发人员面对的大多数软件修改工作在本项目或者其他项目的历史维护过程中都存在相似的工作，这些历史工作为开发人员完成相似工作时提供了借鉴作用。挖掘历史维护信息，可以辅助相似软件修改工作的影响分析和完成周期预测。在本文，我们利用机器学习方法，研究开源项目中的代码修改数据和代码审核数据对代码修改影响分析以及对代码修改的完成周期预测的辅助作用。

修改影响分析领域已经有长达30年的研究历史[12]，传统的研究方法关注系统中代码实体之间的耦合关系以及代码运行信息，以此来分析可能受影响的范围。然而，本文尝试利用大量开源项目的历史修改数据辅助开发人员确定修改的影响范围。开源项目历史修改信息，都是通过代码提交（Commit）的形式存储在版本控制系统中。代码提交中的数据包括（如图1-2所示①）：（1）提交的注释文本，（2）修改前后的代码版本，（3）修改涉及范围，（4）提交编号，（5）提交作者和时间。这些数据包含软件演化过程中修复和改进信息，对后续维

① Spring Boot, https://github.com/spring-projects/spring-boot/commit,2018

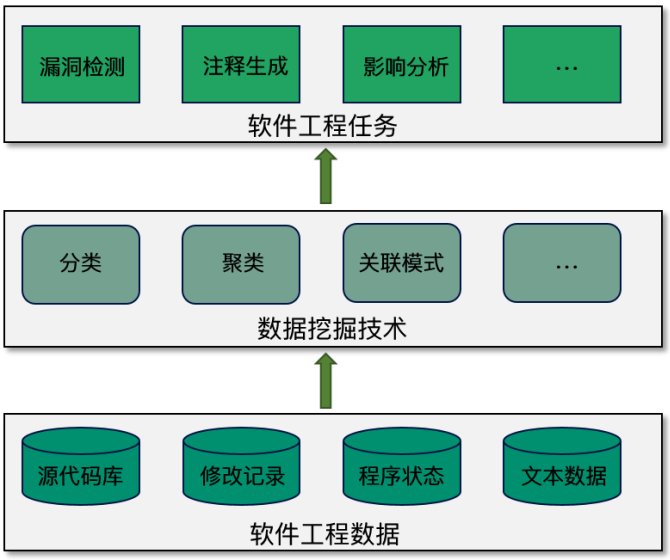


图 1-1 软件工程数据挖掘流程

护工作起着重要作用[13]。代码修改数据在多个软件工程的研究领域都起到了辅助作用，例如：代码审核评论的自动生成[14,15]，缺陷预测[16,17]，修改影响分析[10,11] 等。另外，我们研究发现，大量代码修改工作诸如需求变更、缺陷修复等，都能在软件存储库（本项目或其他项目）中找到相似的工作，相似的代码修改工作中的修改影响范围也存在相似性。因此，本文提出一种基于挖掘代码提交（Commit）信息中的修改模式来辅助修改影响分析的方法，通过关键类判定方法，将提交中核心修改的类等价为当前修改类，利用关键类的修改传播模式辅助分析当前修改的影响范围。

软件修改工作的完成，要求开发人员的修改内容通过审查人员的审查并合入代码库中。我们的调研发现，代码修改的时间周期随着“修改-审查”次数的增加而延长，预估代码修改的“修改-审查”次数能反应代码修改的时间周期。整个代码修改工作由代码修改环节和代码审核环节两部分构成。在代码审核环节，对于开发人员而言，软件维护工作中的主要关注点是“如何最大限度的使代码修改提交通过审核人员审核，并最大限度地缩短修改完成时间”。原则上，代码审查是一个透明的过程，旨在评估修改代码的质量。但是，代码审查的执行过程可能受到各种因素的影响，包括技术因素如代码修改难度，也包括非技术因素如项目复杂度，开发人员和审核人员数量等，这些因素很大程度上影响着代码审核通过率和完成时间。而在代码修改环节，其耗时主要由代码修改难度及代码修改工作量决定。随着开源代码审查工具的广泛使用，使得代码审查

图 1-2 代码提交示例

图 1-3 代码审核示例

数据（如图1-3 所示①）容易被收集，分析和使用。代码审查数据中包含的信息包括：代码审核过程中的信息（项目信息、人员信息等）；代码修改信息（代码文件）；审查周期（“修改-审核”次数）。在本文中，我们从代码审查系统中收集代码修改和审查数据，提取有效特征，用于预估代码修改的完成周期。

1.2 国内外研究现状

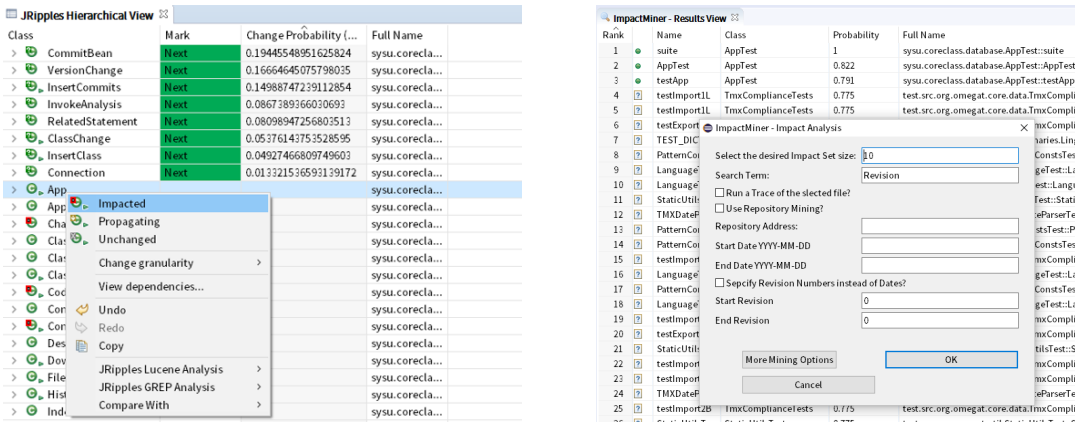
影响分析领域已有多年的研究历史，许多学者通过不同的方法对此问题进

① OPEN DAYLIGHT, https://git.opendaylight.org/gerrit/#/c/62848/1,2018

行研究，积累了大量研究成果。此外，得益于机器学习方法的飞速发展以及开源代码托管和审查工具的广泛使用，数据挖掘在软件维护中的应用也受到更多研究人员的关注。本文将从修改影响分析、修改周期预测、软件存储库挖掘以及机器学习在软件维护中的应用四个方面介绍国内外研究现状以及与本文研究相关的工作。

1.2.1 修改影响分析

修改影响分析可以帮助开发人员理解代码修改，预测修改的影响范围和修改的潜在代价。代码修改往往会由于系统中各部件的关联性而相互波及，如果没有对受波及的部件做相应的调整，会造成各部件之间程序的不一致性[18]。修改影响分析的目的是提前预估代码修改可能造成的影响范围和程度。Bohner等人[19]将影响分析定义为评估软件变更中所有要修改代码的工作。近年来，对于修改影响分析的研究工作不断增多，研究人员提出了许多影响分析的研究方法和支持工具(如图1-4）。



(a) JRipples (b) ImpactMiner

图 1-4 修改影响分析工具JRipples①和ImpactMiner②

修改影响分析方法主要分为静态影响分析[20–23]和动态影响分析[24–27]。静态影响分析通过分析代码之间的语法、语义信息，获取软件部件之间依赖关系，计算修改影响范围。软件项目中某个代码片段的修改通过代码间的依赖关系，可能会传播给其他代码片段，因此分析代码修改的传播机制是静态分析的关键。Breech等人[28]总结了代码修改中的传播机制，并基于这些传播机制建立

① JRipples, http://jripples.sourceforge.net
② ImpactMiner, http://www.cs.wm.edu/semeru/ImpactMiner

影响传播图作为中间表示，通过传播图分析影响范围，使得静态分析的精确度得到很大提高。动态影响分析使用特定测试样例，收集程序运行过程中的数据信息（执行轨迹信息，数据流信息，控制流信息等）计算影响范围。Law 等人[29]提出的PathImpact方法通过收集程序运行时的轨迹信息，构建程序运行路径图，对路径图进行遍历得到修改影响范围。PathImpact方法也是动态分析中使用最广且精度较高的方法。总的来说，静态分析方法由于依赖关系复杂，得到影响范围过大，使得影响分析精度较低；而动态分析方法必须在测试样例运行结束后，才能收集程序运行信息，造成成本过高。针对这个问题，Cai 等人[30]提出了DIVER方法，在PathImpact基础上引入静态依赖图，对程序运行轨迹进行修剪，在较低的成本下得到更高精度影响分析结果。

根据使用技术的不同，修改影响分析还可以划分为基于静态语法依赖关系的影响分析[31,32]，基于耦合关系的影响分析[33,34]，以及基于软件存储库挖掘的影响分析[35,36]等。基于静态语法依赖关系的影响分析是最常用的影响分析技术，通常在代码层次建立依赖图作为中间件，将依赖图中各代码实体之间的可达性作为影响传播的依据。代码层次获得的信息更加丰富，分析结果也更加精确。基于耦合关系的影响分析，计算代码实体之间的结构耦合、概念耦合等，通过耦合关系的强弱计算修改影响范围。基于软件存储库挖掘的影响分析则是根据软件存储库中的历史修改信息，挖掘历史修改中的修改影响范围，将历史修改的影响范围作为当前修改的影响范围。

Li等人[37]的研究表明，基于挖掘软件存储库的影响分析技术越来越受关注。挖掘软件存储库可以发现软件各部件之间重要的历史依赖关系，例如类之间，函数之间或者文档文件。软件维护者可以使用历史依赖关系分析历史变更中，修改影响是如何传播的，而不是仅依赖于传统的静态或动态代码依赖关系。例如，对一段将数据写入文件的代码进行修改，可能需要对从这个文件读取数据的代码进行相应的调整，但是由于两段代码之间不存在数据和控制流信息，传统静态和动态分析方法将无法捕获这些重要的依赖信息。因此，在传统静态或动态分析技术的基础上，挖掘软件存储库是影响分析方法的良好补充。已有的挖掘软件存储库的方法，仅根据当前修改代码实体，挖掘历史修改记录中该代码修改所产生的影响范围，没有涉及具体的修改内容，即对同一代码做不同的修改，通过历史修改记录所获得影响范围是相同的。本文针对这一问题，提出

一种新的历史数据挖掘思路。通过计算当前修改需求、修改代码与历史修改提交中修改描述、修改代码的相似度，得到与当前修改相似的修改提交信息，再使用关键类判定方法判断提交中核心修改的类，将关键类等价为当前修改的类，用提交中关键类的修改模式辅助确认当前修改的影响范围。

1.2.2 代码修改周期预测

代码修改周期有两个阶段组成，分别是修改代码阶段和审核代码阶段。修改代码阶段可以根据代码修改的难度及工作量预估所需时间，而审核代码阶段则存在更多不确定因素。开发人员在完成代码修改后需要经过第三方审查人员的检查，审查人员根据代码修改情况提出修复建议，以便在代码集成之前识别和修复缺陷。很多研究人员发现，代码审查环节中的许多因素会对代码修改的完成周期产生影响。Patanamon等人[38]研究发现查找合适的代码审查人员是代码修改任务中的关键步骤，不合适的审查人员将严重提高代码维护工作的成本。同时，Patanamon等人[38]还提出一种代码审查人员的推荐方法，该方法根据修改代码的文件路径的相似性来推荐合适的代码审查人员，位于相似文件路径中的文件将由类似的代码审查人员进行管理和审查。与Patanamon等人[38] 类似，Oleksii等人[39]的实证研究表明开发人员和审查人员的个人因素会很大程度影响代码修改提交的审查通过率。Baysal等人[40]研究发现诸多非技术因素会影响代码修改的审查通过率，包括：代码修改量，修改提交时间，修改需求的优先次序，代码修改人员和代码审查人员等。他们的方法从WebKit① 项目的问题跟踪和代码审查系统中提取信息，验证各因素的重要性。本文方法是从开源项目的历史修改数据中提取代码修改阶段及代码审核阶段的信息，再利用机器学习方法，训练预测模型。

1.2.3 软件存储库挖掘

软件工程中对软件存储库的挖掘已经有很长的历史，开发人员通过软件存储库数据可以理解软件项目的演化历史，从而更好的完成软件维护和更新工作。软件存储库挖掘的一个方向是从版本控制系统中挖掘历史修改信息。历史修改信息可以帮助开发人员了解软件演化进程中的变更模式从而更高效的完成维护工作。代码修改模式挖掘目的是挖掘历史修改信息中代码实体间是否在修改过

① WebKit, https://webkit.org/

程中有关联关系。代码修改中最常见的是同步修改模式，例如，如果从历史信息发现代码实体e_1总是与代码实体e_2同时进行修改，当开发人员再次对e_1进行修改时，可预估e_2需做相应的修改。根据代码同步修改模式，开发人员可以在面对代码修改需求时，预测需要共同修改的代码实体。Bouktif等人[18]定义了同步修改模式的一般概念，即描述在小时间范围内共同变化的代码实体，并使用模式识别中的动态时间扭曲技术对历史修改数据进行分组，提取相似的修改模式。Ying等人[41]通过基于频率计数的频繁模式挖掘技术从源代码更改历史中挖掘同步修改的源代码。Zimmermann等人[42]利用历史修改代码中的关联规则挖掘出代码实体的同步修改模式，并实现了一个原型系统ROSE①，为开发人员预测需共同变化的代码，预防因不完整修改而导致的错误，并且能检测出用传统程序分析方法无法得到的耦合关系。Ajienka等人[43]的研究表明，频繁同步修改的代码实体之间存在很强的耦合关系，挖掘语义耦合关系有助于识别修改模式。除了同步修改模式外，代码修改中还存在许多其他修改模式。Jaafar等人[44]的研究中提出了两种新的代码修改模式，代码异步修改模式和代码移相修改模式。代码异步修改模式指的是在大时间区间内共同修改的代码；代码移相修改模式指的是频繁在相同时间间隔进行修改的代码。Stephane等人[45]通过聚类的方法，对一定时间周期内完成相似修改的代码进行归类，划分不同的修改模式。与Stephane等人[45]类似，Fluri等人[46]利用层次聚类实现了修改模式的半自动挖掘。本文通过挖掘历史修改提交中的修改模式，辅助当前修改的影响分析。

软件存储库挖掘的另一个研究方向是对源代码的挖掘。Michail等人[47]运用数据挖掘技术检测在不同程序中如何复用代码。了解代码的复用模式可以有效减少开发人员工作量。Lin等人[48]利用频繁挖掘技术在源代码中提取编程规则，他们的研究表明，违反编程规则的代码可能存在缺陷。Holmes等人[49]通过挖掘源代码库中代码的结构上下文，问开发人员展示相关API的用法。类似地，Bruch 等人[50]提出从代码库中学习从而提升IDE中代码的补全效果。Zaidman等人[51]通过挖掘软件演化数据，研究工程代码和测试代码在软件演化中如何共同发展。

① ROSE: http://www.st.cs.uni-sb.de/softevo/

1.2.4 机器学习在软件维护中的应用

随着机器学习算法的飞速发展，研究人员已经将机器学习应用于软件维护领域的各个研究工作中。Murali等人[52]提出一个贝叶斯框架，从代码语料库中学习程序规范，使用这些规范检测可能存在缺陷的程序行为。该方法主要的观点是将语料库中的所有规范与实现这些规范的程序语法相关联。Mills等人[53]通过二分类方法实现可追踪性链接恢复，能够自动将所有潜在链接集合中的每个链接分类为有效或者无效。Rath等人[54]利用修改提交的相关信息训练分类器，以识别修改提交所针对的修改问题。Karim等人[55] 提出从软件度量指标中提取特征并使用支持向量机和随机森林建立模型来预测软件故障的方法，他们将软件度量指标划分为静态代码度量指标和过程度量，从静态代码度量指标中提取代码行数，循环复杂度以及对象耦合等特征；从过程度量中提取源代码历史变化等特征。Shimonaka等人[56]提出利用机器学习方法从源代码中识别自动生成的代码，该方法认为通过朴素贝叶斯和支持向量机模型从源代码中学习代码的语法信息，可以预测代码是否为自动生成。Nguyen等人[57]提出一种自动映射不同编程语言之间API的方法，该方法从不同编程语言的原代码库中学习API的关联关系。Mario 等人[58]从项目源代码以来的API中提取特征，使用机器学习算法实现对软件项目的自动分类。在我们的研究工作中，代码修改周期预测模型的构建也是基于机器学习算法。

1.3 本文的研究内容与主要贡献点

本文基于海量开源项目软件维护过程中的代码修改数据，利用机器学习方法，针对代码修改影响分析以及代码修改完成周期预测进行深入研究。本文的主要研究内容如下所述：

(1) 针对代码修改影响分析，本文提出一种基于历史修改模式的影响分析辅助方法。本文从开源项目中获取代码提交历史数据构建提交语料库；计算当前修改目标与提交注释信息以之间以及当前修改前后代码与提交中修改前后代码之间的相似度匹配最相似的提交；然后通过传统影响分析方法获取当前修改的初始影响集，根据结构耦合关系的相似度将相似提交的修改模式映射回当前修改，对初始影响集中的类进行重排序得到最终修改集。最后，通过实验对比初

始影响集与最终影响集的影响分析效果。

(2) 针对修改完成周期的预测，本文提出一种基于可判别性特征的修改完成周期预测方法。本文从开源代码审查软件中获取海量代码修改历史数据，提取可判别特征，并结合机器学习方法建立代码完成周期预测的模型。可判别特征的提取包括：描述代码审核过程中影响因素的特征；描述代码修改量的特征；描述代码修改难度的特征。

本文的主要创新点如下：

(1) 在修改影响分析方面，本文引入开源项目中相似修改提交的影响传播模式来辅助影响分析。与其他挖掘项目历史修改信息的影响分析方法不同的是，本文从多个开源项目中基于修改内容相似度匹配历史提交，而传统方法仅从当前修改类的项目修改历史中检索与当前类共同修改的类集，不涉及具体修改内容。实验结果表明，本文提出的影响分析辅助方法，在多个开源项目上都能提升传统影响分析工具的精确率和召回率。

(2) 在修改完成周期预测方面，本文创新地提出利用代码修改和审查信息中的可判别特征，预测代码修改周期的方法。该方法开源项目的代码修改数据中，提取了多维度的可判别特征，从代码修改及代码审核两个角度衡量代码修改周期，并结合机器学习方法，构建代码修改周期预测模型。实验结果表明，本文的代码修改周期预测模型在多个开源项目的数据集上都取得了较好的预测效果。

1.4 本文的论文结构与章节安排

本文共分为五章，章节内容安排如下：

第一章主要阐述了代码修改影响分析和代码修改完成周期预测在软件维护过程中的重要性，概述了代码修改影响分析、代码修改模式挖掘、代码修改周期预测以及软件存储库挖掘的国内外研究现状，以及介绍了本文的研究内容和主要贡献点。

第二章提出了基于历史修改模式的影响分析方法。该章主要介绍代码修改提交语料库的构建、相似提交的筛选、历史修改模式的映射以及影响集的优化等。

第三章提出了基于可判别性特征的修改周期预测方法。该章主要介绍代码修改和审查数据的处理、特征的提取、算法选择以及模型优化和评估等。

第四章的主要内容是介绍本文提出的修改影响分析辅助方法和代码修改周期预测方法的实验分析与评估，以及具体工具的设计。

第五章对本文的工作进行总结。主要总结本文提出的方法，并分析这些方法存在的不足与局限；最后，指出在将来的研究工作中如何进行改进。

1.5 本章小结

本章首先介绍了代码修改工作在软件维护过程中的重要性，并对与本文相关的技术和研究领域进行了概述。具体地，代码修改是软件维护中开发人员最频繁的接触工作内容，本文研究工作从不同侧面提出了两种方法，辅助开发人员更高质量、高效率的完成代码修改工作，分别是修改影响分析与修改周期预测。已有研究中，许多学者也提出了关于辅助开发人员完成代码修改的方案，本文对相关技术进行了总结和分类，分别介绍了代码修改影响分析、代码修改周期预测、软件存储库挖掘以及机器学习在软件维护中的应用以及国内外研究现状。最后总结了本文的主要研究工作以及本文的创新点和贡献点，并介绍了对本文的章节安排。

第 2 章　基于历史修改模式的影响分析方法

对于一个软件系统来说，经过多年的开发历史，系统中各个软件实体之间存在非常复杂关联关系，当开发人员需要对其中一个软件实体进行修改时，必然会影响到其他软件实体，且影响范围会随着实体间的关联关系不断传播。因此，修改影响分析成为软件修改工作中的关键一步。修改影响分析的目的是评估一项修改任务可能带来的风险以及受影响的范围和程度。软件修改的主要目的包括增加新功能、修复缺陷或者适应新的用户需求。在同一项目或不同项目的演化历史中往往存在着相似的修改需求，这些相似修改的修改模式对于当前的修改任务有辅助作用。传统的静态影响分析以及动态影响分析方法没有利用修改内容的相似性，引入历史修改模式信息可以对传统影响分析的结果进行优化，提升影响分析效果。已有的挖掘历史修改信息的影响分析方法没有涉及具体修改内容，本文从版本控制系统中收集优质开源项目的修改提交数据构建提交语料库，通过修改代码相似度以及修改需求的相似度检索相似的修改提交，再借助关键类判定方法将相似提交中的关键类作为当前修改类的等价类，引入关键类的修改模式对传统影响分析结果进行优化，从而获得最终的影响集。

2.1　基于历史修改模式的影响分析方法概述

图2-1是基于历史修改模式的影响分析辅助方法总览，方法主要分为三个步骤：（1）构建历史提交库，（2）检索相似提交，（3）辅助影响分析。第一步，首先从版本控制器中收集大量不同项目下的代码修改提交数据，并将这些提交数据存储于本地仓库中，提交数据包括修改前代码片段、修改后代码片段以及修改注释信息。分别对每一条提交数据进行文本预处理，构建历史提交库。第二步，使用第一步中预处理后文本信息构建文本语料库，语料库中每一条语料包含相应提交中的代码修改信息和注释信息，使用词嵌入方法Word2vec 训练得到词嵌入模型；对当前修改工作中的修改需求(文本描述)和修改前后的代码片段做相同的文本预处理；利用词嵌入模型计算当前修改与历史提交的向量相似度，得到相似修改提交列表。第三步，使用关键类判定方法识别相似提交中的关键

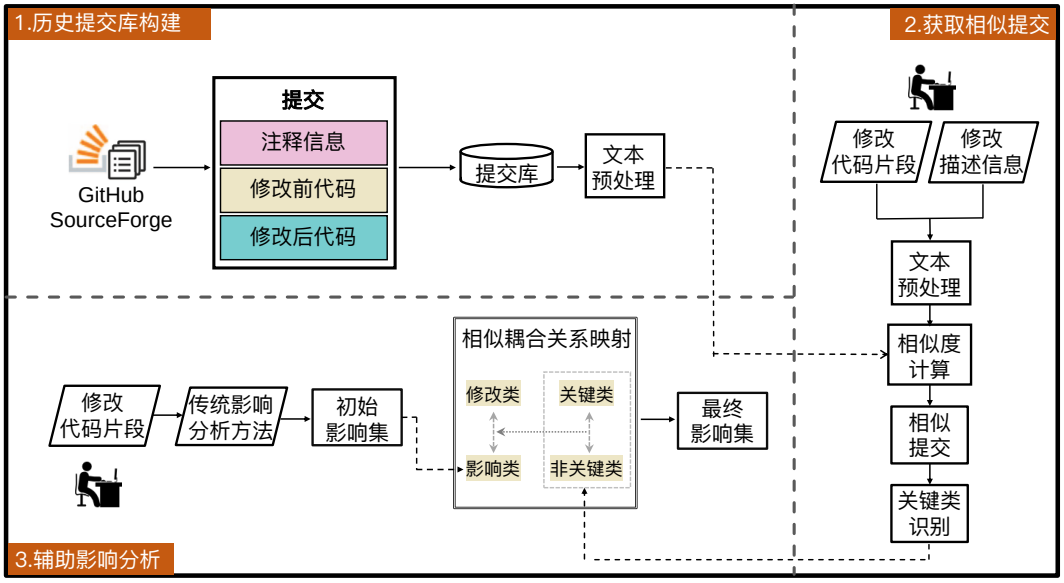


图 2-1　基于历史修改模式的影响分析方法总览

类，将关键类作为当前修改类的等价类；提取关键类与提交中其他类的耦合关系；使用传统影响分析方法得到当前修改的初始影响集，提取当前修改类与初始影响集中所有类的结构耦合关系；利用相似的耦合关系将提交中关键类的修改模式映射回当前修改类，对初始影响集进行优化，得到最终影响集。

2.2　历史提交语料库构建方法

本文中用于构建提交库的数据来源于版本控制系统Github、Sourceforge中的开源项目，这些开源项目经过长期维护存在大量代码修改的提交数据。我们从版本控制系统中筛选出182个开源项目，并从这些开源项目中收集了94778个提交数据用于构建提交库。

2.2.1　提交数据筛选

本文从开源项目中收集修改提交数据（Commit）用于构建提交库及验证的数据，而开源项目中代码提交数据质量良莠不齐，为了防止质量较差的提交数据对影响分析产生负面的优化效果，我们需要对用于构建提交库的提交数据进行去噪。我们对大量提交数据进行观察后，发现开源项目中修改提交主要存在以下问题(如图2-2所示)：（1）修改提交中注释信息缺失或注释文本较短(少于2个单词)，这类提交缺乏对修改内容的准确描述信息，会在相似提交检索中

对修改目标的相似性计算产生负面影响；（2）提交注释文本过长（多于200个单词），这类提交的注释文本中通常罗列了该修改工作中大部分琐碎的修改内容，难以判断其核心修改部分；（3）提交中只涉及一个类的代码修改，这类提交数据由于只包含一个类，不存在可借鉴的修改模式；（4）提交中涉及多于二十个类的代码修改，这类提交数据往往是包含了多个普通提交，一个提交中存在多个不相关的修改内容。

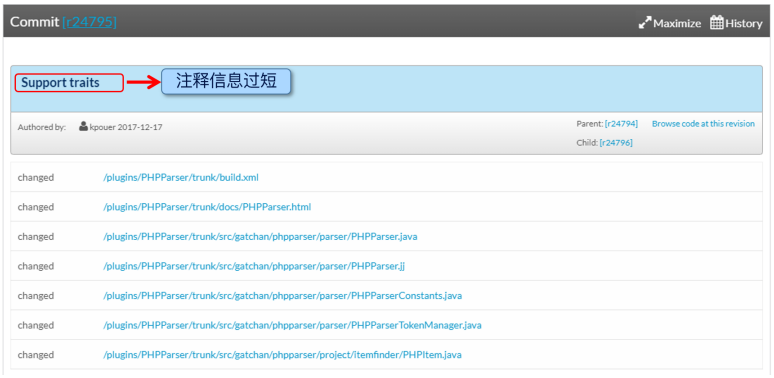
通过对存在问题的提交数据筛选后，我们收集的提交数据包含182个开源项目共94778 条。一条提交数据主要包含：提交编号、作者信息、提交时间、注释文本、修改前代码版本和修改后代码版本等。在相似提交检索任务中，本文通过修改目标的相似度以及代码修改片段的相似度来衡量当前修改与提交的相似度，因此，我们收集的提交数据仅需保留提交注释文本、修改前代码数据和修改后代码数据。

2.2.2　文本预处理

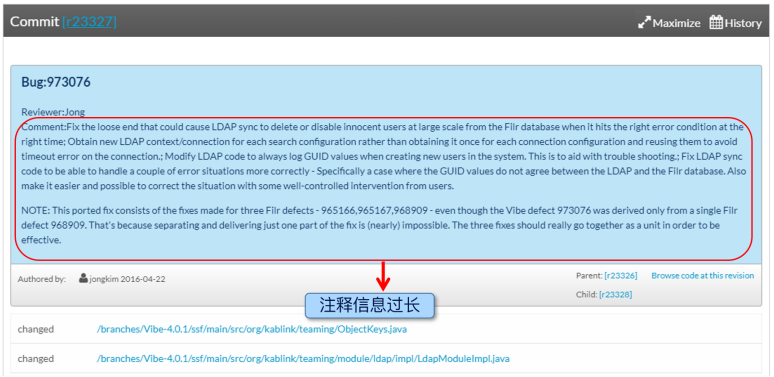
提交中代码数据和注释文本中通常存在许多噪声字符，这些字符可能会削弱文本中原有的语义信息。从开源项目中收集的提交数据直接用于相似提交检索有可能起负面作用，我们需要对提交中的代码及注释文本进行预处理。

第一步，对代码及注释文本使用相同预处理方法。过滤掉代码及注释文本中的标点符号、特殊字符、数字等，并利用空格、换行符将代码及注释文本分别转化为一系列字符串和或者单词。将文本中所有单词的大写字母统一规范化为小写，如“*Text*”转换为“*text*”。另外，开发人员喜欢在编写注释和代码时，使用缩略词，在文本预处理中，需要对缩略词进行补全，如“*Info*”转换为“*imformation*”。同时，需要对代码及注释文本进行词形还原和词干提取，利用单词规范化的方式，能有效提高文本相似度计算中的精确度。本文借助WordNet 对所有的文本做词形还原和词干提取，WordNet 由英语词汇数据库构成，通过同义词网络汇集不同词性的英语词汇。词形还原的目的是将不同形式和不同时态的单词统一为一般形式，如复数“*classes*” 还原为“*class*”，进行时的“*running*” 还原为”*run*”等。词干提取的目的是提取文本中单词的词干或词根表示，如“*effective*”转换为“*effect*”，“*happiness*”转换为“*happy*”。

第二步，由于代码数据与自然语言形式的注释文本之间存在明显差异，

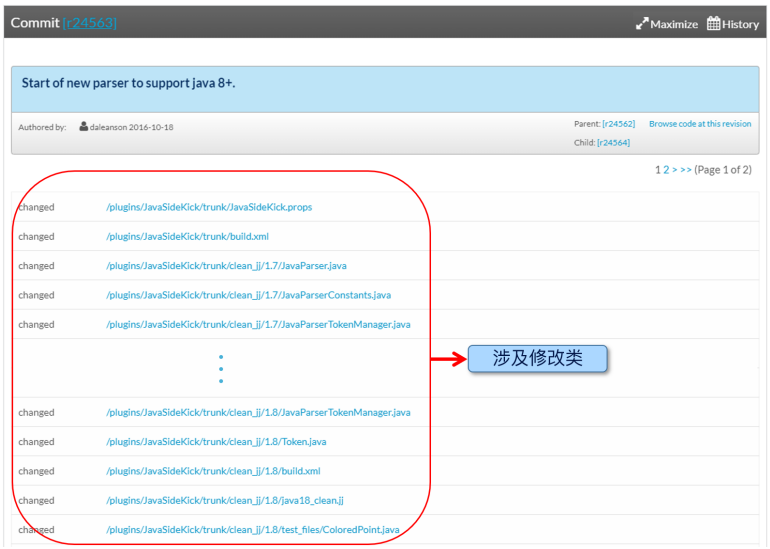


(a) 提交注释信息过短



(b) 提交注释信息过长

(c) 提交只涉及一个类修改



(d) 提交涉及超过二十个类修改

图 2-2　存在问题的提交图例

需要对代码数据使用额外的文本处理方法。语法分析从程序逻辑的角度衡量代码间的相似度，而语义分析是直接基于代码中的标识符判定代码的相似度。但是，代码文本中许多标识符并不能对语义相似度的分析起到促进作用，相反会起到一定的负面影响（例如，“*asaa*”，“*a*”，“*b*”等等）。本文通过使用一系列方法过滤代码文本中特定的标识符，包括：(1) 过滤代码文本中的虚词，如“*and*”，“*a*”，“*an*”等；（2）过滤不包含具体语义信息的字符串，如“*ttt*”，“*hgkk*”等等；(3)将使用驼峰命名法的词汇分割成多个的单词，如“*removeContextInfo*”分割成“*remove*”，“*Context*”和“*Info*”。经过以上文本预处理后，提交中的原始代码片段转换成文本，且仅保留了包含语义信息的单词。

另外，本方法除了需要对历史提交数据进行文本预处理之外，还需要对当前影响分析对象进行相同的文本预处理。当前修改目标的自然语言描述相当于提交中的注释文本，当前修改的代码变更片段相当于提交中代码修改片段。

2.2.3　提交语料库构建

本文提出的方法需要从本地提交库中检索与当前修改任务最相似的代码提交，其中关键点是计算当前修改代码、修改目标与提交中修改代码、提交注释之间的相似度。提交中的注释文本和代码文本经过预处理后可以融合成一条代表提交的语料信息，我们通过所有提交的语料信息构建提交语料库。其中，值得注意的是，对于代码文本（无论是提交中代码，还是当前修改任务中代码）首先需要识别其中涉及修改的代码段。另外，一个提交中通常涉及多个类的修改，需要识别其中核心修改的类，作为当前修改类的等价类。

本文使用*ChangeDistiller*方法[59] 中修改前后两个版本的代码文本中提取涉及修改的代码段。*ChangeDistiller*方法通过对比修改前后版本的代码对应的抽象语法树之间的差异，来获取涉及修改的代码段。由于提交中存在多种修改任务，*ChangeDistiller*方法还能识别提交中不同的代码修改类型，例如：参数名变更（“*Parameter Renaming*”），参数增加/删除（“*Parameter Insert/Delete*”），方法名变更（“*Method Renaming*”），语句增加/删除（“*Statement Insert/Delete*”）等。另外，我们使用关键类判定方法识别提交中核心修改的类。关键类判定方法将在后文介绍。

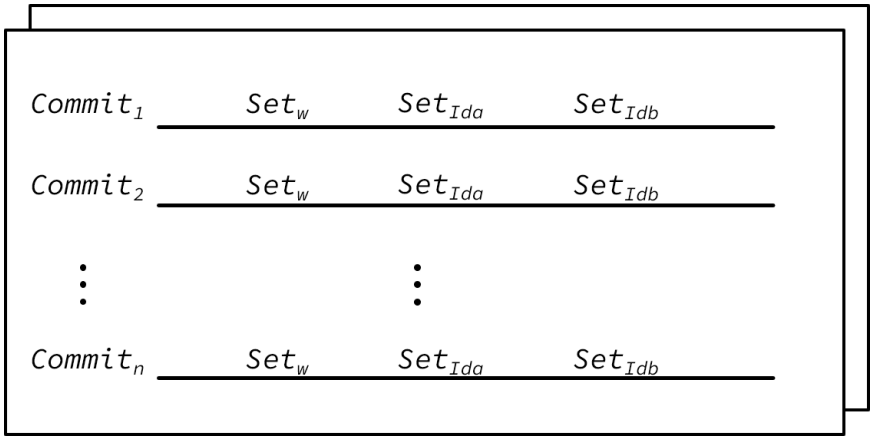


图 2-3　提交文本语料库构建方式图示

对于提交中注释文本、代码修改前片段及代码修改后片段，我们分两方面处理，首先将三个文本储存于提交库中用于后续相似提交检索，此外，将提交库中所有提交的文本语料融合成语料库，用于训练词嵌入模型。本文语料库的组织形式如下：

对于注释文本中每个单词w_i，从修改前代码文本中随机选取四个标识符Id_{a1}、Id_{a2}、Id_{a3}、Id_{a4}，组合成Id_{a1}、Id_{a2}、w_i、Id_{a3}、Id_{a4}，记为Com_1；从修改后代码文本中随机选取四个标识符Id_{b1}、Id_{b2}、Id_{b3}、Id_{b4},组合成Id_{b1}、Id_{b2}、w_i、Id_{b3}、Id_{b4},记为Com_2；从而，针对注释文本中每个单词得到组合语料$Com_{w_i} = \{Com_1, Com_2\}$，所有$Com_{w_i}$组合得到注释文本对应语料$Set_w$。

对于修改前代码中每个标识符Id_{ai}，则随机从注释文本中选取四个单词w_1、w_2、w_3、w_4，与Id_{ai} 组合成w_1、w_2、Id_{ai}、w_3、w_4，记为Com_{Id_a}，所有的Com_{Id_a}组合得到修改前代码对应语料Set_{Id_a}；类似地，对于修改后代码中每个标识符Id_{bi}，随机从注释文本中选取四个单词w_1、w_2、w_3、w_4，与Id_{bi}组合成w_1、w_2、Id_{bi}、w_3、w_4，记为Com_{Id_b}，所有的Com_{Id_b}组合得到修改前代码对应语料Set_{Id_b}。

最后，将每个提交对应的Set_w、Set_{Id_a}、Set_{Id_b}组合成一条语料，所有提交对应的语料信息组合得到完整的语料库（如图2-3所示）。

2.3　基于词嵌入的相似提交检索

本节详细介绍了从本地提交库中检索相似提交的方法。我们通过提交语料

库训练词向量模型，再根据词向量模型计算提交与当前修改对象之间的相似度，筛选出相似度最高的前20 个提交用于后续影响分析结果优化。提交与当前对象的相似度结合了修改代码相似度以及修改文本描述的相似度。

2.3.1　词向量模型训练

当前，文本语义相似度的计算方法主要分为两种，一种是基于WordNet[①]的语义相似度计算方法。WordNet由庞大的词汇数据库构成，通过同义词集形成词汇网络。同义词的集合表示一个独立的语义信息。而两个不同单词间的语义相似度，可以使用单词在WordNet网络中的位置距离计算得到。

另一种是本文所采用的基于词嵌入技术的Word2vec方法[60] 训练词向量模型。Word2vec的计算方法是通过训练将每个词映射为K维向量（K是人为设定的超参数），再根据词与词之间的向量距离来确定它们的语义相似度。其模型由三层神经网络构成，输入层-隐藏层-输出层，通过三层神经网络对语言模型进行建模，从而得到单词在向量空间上的表示。与潜在语义分析（Latent Semantic Index）、潜在狄利克雷分配（Latent Dirchlet Allocation）不同的是，Word2vec充分利用了单词的上下文信息，使得语义信息的表示更加准确。Word2vec的训练模型主要分为两种，CBOW 模型和Skip-Gram 模型（如图2-4 所示）。这两种模型主要的区别在于词向量模型训练过程中，CBOW模型根据上下文信息预测目标单词的概率分布，而Skip-Gram 模型则是通过当前单词预测其上下文出现的概率[61,62]。

本文在相似提交检索中采用Skip-Gram模型，模型训练的目标函数为公式2.1：

$$\sum_{i=1}^{n} \sum_{-k \le j \le k, j \neq 0} \log p(w_{i+j}|w_i) \tag{2.1}$$

其中，w_i和w_{i+j}分别表示长度为$2k+1$的上下文滑动窗口中的中心词和中心词的上下文(本文中取$k=2$，即滑动窗口大小为5)，n代表语句的长度。式子$\log p(w_{i+j}|w_i)$表示一个条件概率, 该条件概率由softmax函数定义，如下所示:

$$\log p(w_{i+j}|w_i) = \frac{\exp(v'^{T}_{w_{i+j}} v_{w_i})}{\sum_{w \in W} \exp(v'^{T}_{w} v_{w_i})} \tag{2.2}$$

[①] https://wordnet.princeton.edu/

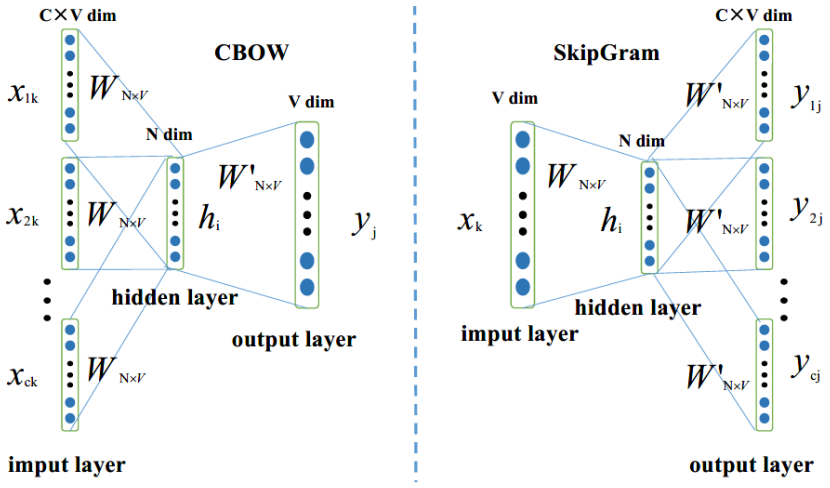


图 2-4　CBOW与Skip-Gram模型区别

其中，v_w表示输入向量，v'_w表示模型中的单词w的输出向量。W表示所有单词的词汇。而$p(w_{i+j}|w_i)$为在中心词w_i的上下文中出现的单词w_{i+j} 的归一化概率。我们采用负抽样方法来计算这个概率。

2.3.2　相似提交列表获取

通过以上方法得到词向量模型后，我们可以得到提交语料中每个单词的向量表示。本文根据向量的余弦距离度量两个文本之间的相似度，再筛选出相似度最高的前20 个提交构成相似提交列表。文本相似度的计算方法如下：

对于文本C、文本S中任意单词w_c和w_s，其中$w_c \in C$，$w_s \in S$，w_c和w_s相似度为：

$$sim(w_c, w_s) = \cos(\mathbf{w}_c, \mathbf{w}_s) = \frac{\mathbf{w}_c^T \mathbf{w}_s}{||\mathbf{w}_c|| ||\mathbf{w}_s||} \tag{2.3}$$

对于单词w_c与文本S的语义相似度，则取单词w_c 与文本S中所有单词之间相似度的最高值，公式如下：

$$sim(w_c, S) = \max_{w_s \in S} sim(w_c, w_s) \tag{2.4}$$

由于在文本相似度计算中TF-IDF值作为单词权重的方法会造成近似相交性[61]。本文采用文本C中单词与文本S相似度的平均值作为文本C与文本S的相

似度。另外，我们忽略与文本相似度为0的单词，作如下集合定义：

$$Set(C \to S) = \{w_c \in C | sim(w_c, S) \neq 0\} \tag{2.5}$$

文本C到文本S的相似度为：

$$sim(C \to S) = \frac{\sum_{w_c \in Set(C \to S)} sim(w_c, S)}{|Set(C \to S)|} \tag{2.6}$$

同理，文本S到文本C的相似度为：

$$sim(S \to C) = \frac{\sum_{w_s \in Set(S \to C)} sim(w_s, C)}{|Set(S \to C)|} \tag{2.7}$$

由此，我们可以得到文本C与文本S的相似度：

$$sim(C, S) = \frac{sim(C \to S) + sim(S \to C)}{2} \tag{2.8}$$

本文中提交与影响分析对象的相似度有三部分度量：（1）提交注释文本与影响分析对象修改目标的相似度$CommentSim$；（2）提交中旧版本代码修改片段与影响分析对象修改前代码片段的相似度$OldCodeSim$；（3）提交中新版本代码修改片段与影响分析对象修改后代码片段的相似度$NewCodeSim$。三者赋予不同的权重得到提交的综合相似度$CommitSimi$，公式如下：

$$CommitSimi = \alpha \cdot CommentSimi + \beta \cdot OldCodeSimi + \gamma \cdot NewCodeSimi \tag{2.9}$$

在实验测试中得到最优权重组合为：$\alpha = 0.4$、$\beta = 0.3$、$\gamma = 0.3$。

2.4　代码修改提交关键类判定方法

本文提出的影响分析辅助方法中一个关键步骤是从相似提交中找出影响分析对象的等价类，使用等价类的修改模式对初始影响分析结果进行优化。本文通过关键类判定方法识别提交中核心修改的类，将关键类作为影响分析对象的等价类。关键类判定方法参考于我们之前的研究工作[63,64]。我们从代码结构耦合、代码修改以及提交类型三个维度提取特征（如表2-1 所示），训练机器学习模型，通过机器学习模型预测提交中的关键类。本节主要介绍三个维度的特征提取方法。

代码耦合特征指的是在代码修改提交中，不同个类之间的结构耦合关系。为了表示类之间的耦合特征，我们通过入度和出度刻画提交中类与类之间的关联关系。一个类的入度代表这个类被其他类调用的次数；而出度则表示这个类调用其他类的次数。另外，我们通过特征表示了提交中不同类的出度大小、入度大小之间的关系。如果一个类的入度值大于提交中其他类的入度值，表示这个类更频繁被其他类调用，则这个类发生修改将更可能影响到其他类，更大概率是一次修改提交中的关键类。相似地，如果一个类的出度值大于提交中其他类的出度，表明这个类比较大概率受到其他类的影响，这个类也比较可能是非关键类。

对于一个代码修改提交，类的代码修改量往往体现了这个类在提交中的重要程度，关键类在一次提交中往往是被主要修改的类。从修改代码量角度观察，关键类的代码修改量也往往要比提交中其他类多。对于一个类的代码修改量，我们通过两个值来度量：类中方法的修改量以及类中代码行的修改量。另外，我们还考虑了提交中多个类的代码相对修改量。相对修改量指的是提交中，一个类的代码修改量与提交中所有类的平均代码修改量的比值。此外，我们还对类的修改类型进行划分，包括：新增，修改和删除三种。我们定义的代码修改特征如表。

根据代码修改的目标不同，我们将提交分为几种类型：（1）添加新功能（前向工程）；（2）删除过时代码（逆向工程）；（3）修复代码缺陷（纠错工程）；（4）非可执行文件修改（非代码修改类型），如配置文件，说明文件等。划分提交类型的目的是，如果提交类型不同，则提交中关键类的差异也会较大。

例如：当提交类型为添加新功能，提交中核心修改的类一般为新增加的类。

表 2-1　关键类判别特征

类型	编号	描述
代码耦合特征	01	入度
	02	入度/提交中类最大入度
	03	入度/提交中类平均入度
	04	出度
	05	出度/提交中类最大出度
	06	出度/提交中类平均出度
	07	入度是否为零
	08	出度是否为零
代码修改特征	09	代码修改行数
	10	代码修改行数/提交中类最大代码修改行数
	11	代码修改行数/提交中类平均代码修改行数
	12	方法修改数
	13	方法修改数/类中方法总数
	14	方法修改数/提交中最大方法修改数
	15	方法修改数/提交中平均方法修改数
	16	修改类型
提交类型特征	17	是否为前向工程
	18	是否为逆向工程
	19	是否为纠错工程
	20	是否非代码类型修改
	21	提交中类的个数

2.5　基于结构耦合相似性的修改影响集优化

本节详细介绍基于相似提交对传统影响分析的修改影响集进行优化的过程。我们先使用传统影响分析方法得到当前修改类的初始影响集，初始影响集中的类称之为初始影响类。从提交库中检索得到前20 个最相似的提交，提取提交中关键类与非关键类的结构耦合关系以及修改类与初始影响类的结构耦合关系，计算结构耦合关系的相似度，对相似度高于指定阈值的初始影响类，调整其在初始影响集的位置，得到最终影响集。

2.5.1　软件实体间结构耦合关系提取

我们提出的基于历史修改模式的影响分析辅助方法中，通过类与类之间结构耦合关系的相似性，将历史修改模式映射回当前修改类中。当相似提交中关键类A和非关键类B 的结构耦合关系与当前修改类和初始影响类C 的耦合关系的相似度很高时，我们认为修改类更可能将修改影响传播给初始影响类C。我们分析的软件实体包括类、方法和属性。对于两个类A和B，我们从类层次、方法层次以及属性层次提取了A、B 的18 种耦合关系（如表2-2所示）：

1) 类层次。主要包括两种耦合关系：继承和实现接口。

2) 方法层次。主要包括5种耦合关系：类型检查（instanceOf）、强制类型转换（Type-Casting）、类实例、函数返回值类型、方法体抛异常（Throws Exception）。

3) 属性层次（包括：成员属性、方法参数、方法局部属性）指的是用类B定义类A的一个属性attr，由于属性的使用方式可能不同，我们考虑以下三种耦合关系：类A中的方法直接使用attr；类A中的方法通过attr引用类B中的属性；类A的方法通过attr引用类B中的方法。结合属性的不同位置（包括：成员属性、方法参数、方法局部属性），属性层次共提取9种耦合关系。

4) 另外，我们总结其他情况提取了两种耦合关系：在类A的方法（或者类A的静态代码块、非静态代码块、属性）调用类B 中的构造方法或静态方法。

对于两个类之间的结构耦合关系，我们使用一个长度为18的特征向量表示，且向量中的每个维度对应表2-2的编码。每个维度采用二元编码的形式表示，若存在某种耦合关系，则该维度为1，否则，该维度为0。

2.5.2　初始影响集优化

通过传统影响分析方法获得初始影响集后，我们对影响集列表中的类赋予初始分数如表2-3所示。影响集中类的排序代表着其受影响的概率，因此，我们记倒数第一个类为0分，逆序为排在前面的类依次增加20 分。

对于初始影响集中的每个类，首先，提取其与修改类之间的结构耦合关系，记耦合关系为V_i（18维向量）；然后，提取每个相似提交中关键类与其他类的耦合关系，记为V_j；最后，计算V_i与所有V_j相似度，若V_i与提交中有n个V_j 相似度大于指定阈值η（由实验测试得出），则V_i所对应的初始影响类分数增加$n*20$分，

表 2-2　类与类（接口）耦合类型表

层次	编码	耦合类型	
类层次	01	A类继承B 类	
	02	A类实现B接口	
方法层次	03	强制类型转换	
	04	类型检查instanceOf	
	05	B类作为A方法返回值类型	
	06	类实例b.Class	
	07	方法抛异常（B为异常类）	
属性层次	08	A成员属性attr	直接调用attr
	09		通过attr调用其成员属性（attr为B类的实例）
	10		通过attr调用其成员方法（attr为B类的实例）
	11	A方法局部属性attr	直接调用attr
	12		通过attr调用其成员属性（attr为B类的实例）
	13		通过attr调用其成员方法（attr为B类的实例）
	14	A方法形参attr	直接调用attr
	15		通过attr调用其成员属性（attr为B类的实例）
	16		通过attr调用其成员方法（attr为B类的实例）
其他情况	17	A类调用B 类的静态方法	
	18	A类调用B类的构造方法	

调整其在影响集中的位置。

本文通过*Jaccard*相似系数度量两组耦合关系向量之间的相似性，*Jaccard*相似系数相对于其他相似性度量方法，更适合处理非对称二元变量。在两组结构耦合关系中，某一维度上的耦合关系编码正匹配（两者都取值为1）比负匹配（两者都取值为0）更有意义，在相似性度量上，负匹配的数量被认为是不重要的，*Jaccard*相似系数可以忽略负匹配的影响。初始影响集优化公式如下：

$$JaccardSimi(V_i, V_j) = \frac{V_i \cap V_j}{V_i \cup V_j} \tag{2.10}$$

$$Score_i = \begin{cases} Score_i + 20, & JaccardSimi(V_i, V_j) > \eta \\ Score_i, & else \end{cases} \tag{2.11}$$

表 2-3　初始影响集分数赋值示例

排序	类名	影响概率	初始分数
1	*Buffer.java*	0.7374193	240
2	*EditServer.java*	0.6746630	220
3	*Macros.java*	0.6488870	200
4	*EditPlugin.java*	0.6138285	180
5	*GUIUtilities.java*	0.5360856	160
6	*BeanShell.java*	0.5250749	140
7	*Abbrevs.java*	0.4440035	120
8	*BufferIORequest.java*	0.3989099	100
9	*FileVFS.java*	0.3953681	80
10	*Mode.java*	0.2678907	60
11	*View.java*	0.1140896	40
12	*EditPane.java*	0.0990951	20
13	*HistoryModel.java*	0.0719621	0

2.5.3　基于历史修改模式的影响分析算法

算法2-1是本文提出的影响分析辅助方法的算法整体流程，本节主要阐述算法的实现流程及细节。算法的输入是当前修改目标的自然语言描述*describe*和代码修改片段*codefragList*；输出目标为初始影响集优化后得到的最终影响集*finImpctedSet*。

算法3-15行为相似提交的检索过程。通过2.3节中提交相似度计算方法，对比提交库中的所有提交，得到一个大小为20的相似提交列表*SimiCommits*。在检索提交库过程中，若相似提交列表中提交数量低于20，则直接加入当前检索的提交；若提交列表中提交数量为20，则对比提交列表中最后一个提交（相似度最低的提交）的相似系数与当前提交的相似系数的大小，如果当前提交的相似系数更大，则更新提交列表。

算法16-30行为初始影响集*initImpactedSet*的优化过程。*initImpactedSet*由传统影响分析方法得到，并对影响集中的类根据受影响概率不同赋予不同的初始分数*Score*。对于*initImpactedSet*中一个受影响的类*impactClass*，提取其与修改类的结构耦合关系*impactClassCoupleVec*；然后遍历相似提交列表*SimiCommits*中所有提交*commit*，识别提交中的关键类*coreclass*，对于提交中的所有非

算法2-1: 影响分析算法

```
1:  Input: describe: 修改目标，codefragList: 修改代码片段;
2:  Output: finImpactedSet: 最终影响集;
3:  SimiCommits = {}
4:   foreach commit ∈ commitCorpus
5:    simi = getTextSimi(commit, describe, codefragList)
6:    if SimiCommits.size() <20 do
7:      SimiCommits.add(commit)
8:    end if
9:    Sorted(SimmiCommits)
10:   if SimiCommits.size() == 20 && simi > SimiCommits.get(SimiCommits.Size()-1).simi do
11:     SimiCommits.add(commit)
12:     Sorted(SimmiCommits)
13:     SimiCommits.remove(SimiCommits.Size()-1)
14:   end if
15: end foreach
16: initImpactedSet = getInitImpactedSet(describe, codefragList)
17: foreach impactClass ∈ initImpactedSet do
18:   impactClassCoupleVec = getCoupleVec(changeClass,impactClass)
19:   foreach commit ∈ SimiCommits do
20:     coreclass = getCoreClass(commit)
21:     foreach otherclass ∈ commit do
22:       otherClassCoupleVec = getCoupleVec(coreclass, otherclass)
23:       JaccardSimi = getJaccard(impactClassCoupleVec, otherClassCoupleVec)
24:       if JaccardSimi > 0.8 do
25:         impactClass.Score + = 20
26:       end if
27:     end foreach
28:   end foreach
29: end foreach
30: finImpactedSet = sorted(initImpactedSet)
31: return finImpactedSet
```

关键类*otherclass*，提取其与关键类的结构耦合关系*otherClassCoupleVec*，再计算*impactClassCoupleVec* 与*otherClassCoupleVec* 的*Jaccard*相似系数，若*JaccardSimi* 大于指定阈值（实验测试最优阈值为0.8），则当前受影响类*impactClass* 对应的分数*Score*增加20；最后对初始影响集根据分数大小，进行重排序得到最终影响

集*finImpactedSet*。

2.6 本章小结

在软件代码修改工作中，由于软件实体之间相互关联，代码修改会影响到软件中的其他部分。如何确定影响范围，是提高代码修改工作效率的关键。在本章中，我们提出了一种基于历史修改模式的影响分析辅助方法，引入相似的代码修改工作中的影响传播方式，对影响分析的影响集进行优化，提高准确性。具体地，该方法收集版本控制系统中开源项目的历史提交数据构建提交语料库；再借助词向量模型，在提交库中检索与当前修改任务相似的提交，并通过关键类判别技术识别提交中的核心修改的类，作为修改类在提交中的等价类；最后，基于相似提交中关键类和非关键类之间与当前修改类与其他类之间的结构耦合关系相似性，将提交中关键类的修改模式映射回修改类，完成对初始影响集的优化。

第 3 章　基于可判别特征的代码修改周期预测

开发人员在修改完代码后需提交给审核人员，如果代码审核不通过，开发人员需对代码进行再次改进。我们通常认为代码修改工作完成的标志是通过代码审核人员的审查。为了提高代码质量，软件代码修改工作通常包含多次“修改-审核-再修改”的过程，代码审核的次数直接体现该任务周期的长短。因此，我们定义代码修改的时间周期是从修改代码开始至代码通过审核并提交代码库。本文提出一种通过预测代码修改需要经过的“修改-审核”次数来评估该修改时间周期的方法。代码修改周期的预测有助于开发人员及时发现代码修改中存在的问题，以及有助于项目管理人员重新评估该代码修改任务的工作量和难度，并及时做出调整，如增加开发人员等，进而缩短代码修改任务的完成周期。在本文中，我们从代码审核工具中收集开源项目维护过程中的代码修改和审核信息，并从中提取可判别特征用于训练机器学习分类模型。我们从审核原特征（meta-feature）、代码耦合特征（coupling feature）以及代码修改特征（modifying feature）三个维度衡量代码修改及审核中各因素对修改周期的影响。

3.1　修改周期预测问题描述及方法总览

代码修改周期指的是开发人员完成一项代码修改工作的时间周期，包括代码修改阶段及代码审核阶段。其中代码修改阶段所需时间主要由修改难度，修改工作量等因素的影响，代码审核阶段所需时间还受到审核过程中一些非技术因素的影响，例如审核人员、项目信息等。代码修改工作完成的标志是修改的代码经过审查人员的审核，若代码审核不通过，将导致修改周期延长。在代码修改工作中提前预估修改完成周期，有助于开发人员及项目管理人员更加高效的完成代码修改工作。在本文中，我们通过代码“修改-审核”次数来评估代码修改周期。我们从3000条代码审核数据中调研了代码修改工作中代码“修改-审核”次数与其对应的平均修改周期的关系，如图3-1 所示。从中可以发现，代码审核次数与代码修改完成时间之间存在强相关性，代码审核次数增加意味着代码修改完成时间增加，代码修改周期的长短可以由代码审核次数反应。

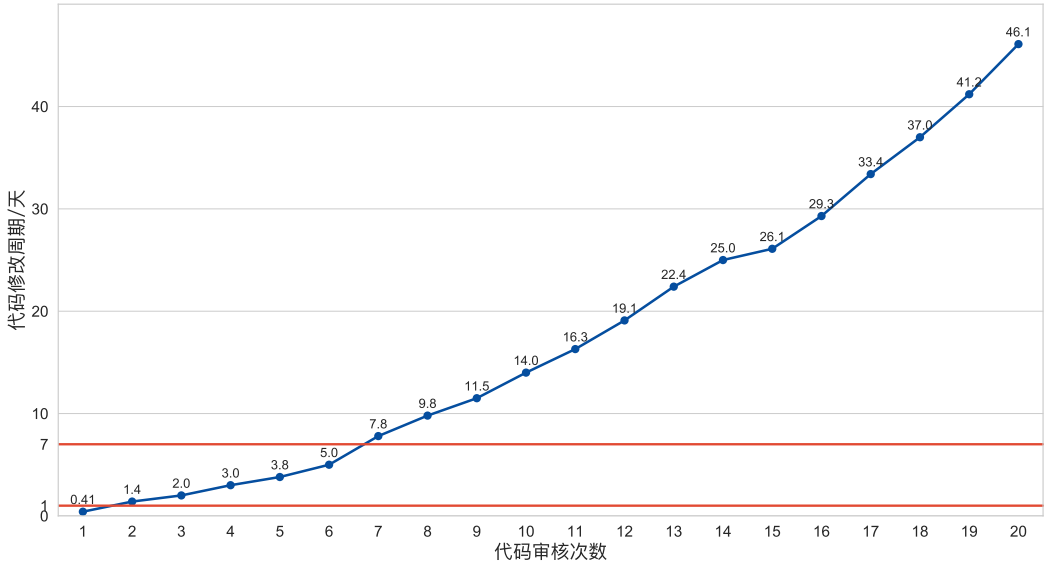


图 3-1　代码审核次数与平均修改周期关系图

图3-2展示了本文方法的总览。该方法主要分为三个阶段：代码修改和审核数据收集阶段、可判别特征提取阶段和机器学习模型构建阶段。代码审核软件的广泛使用，使得审核过程中的历史信息更加容易收集。本文从Gerrit代码审核软件中获取开源项目维护过程中代码审核数据，主要包括代码审核信息以及代码修改信息。在可判别特征提取阶段，我们从审核原特征、代码耦合特征以及代码修改特征等三个维度提取了40维特征。审核元特征用于衡量代码审核过程中的一些非修改因素，包括审核人员信息、审核提交时间、项目信息等。代码耦合特征用于衡量修改代码实体之间的结构耦合关系，如类之间的继承关系，类与方法的调用关系等，这些结构耦合特征体现了修改工作复杂程度。代码修改特征用于衡量修改内容以及修改的工作量，如代码修改行数，所涉及方法的数量等。在模型训练阶段，本文选用在当前机器学习领域效果突出的XGBoost模型作为预测模型，并对比了多个机器学习模型的预测效果。

3.2　多维度的可判别特征提取

本节详细介绍了修改完成周期预测模型中可判别特征提取的方法，我们将从三个不同的维度提取特征来描述代码修改工作，这些维度特征为：审核元特征、代码耦合特征以及代码修改特征。审核元特征描述代码审核过程中的因素对整个时间周期的影响；代码耦合特征从修改代码的结构分析该修改工作的复

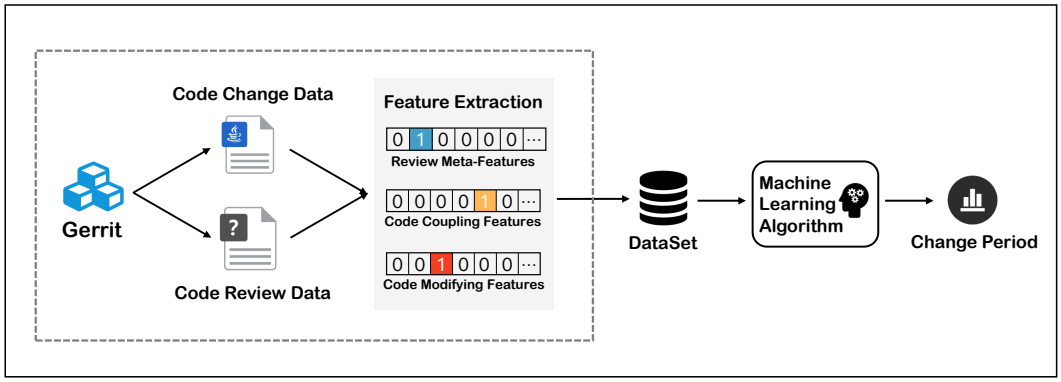


图 3-2　基于可判别特征的代码修改完成周期预测方法总览

杂度；代码修改特征用于衡量代码修改工作的工作量以及修改内容。

3.2.1　代码审核元特征提取

原则上，代码审核是一个透明的过程，代码审核应该对提交代码存在的缺陷具有预防性，审核人员的目标是及时评估提交代码的质量，然而，在实践中代码审核的执行过程会受到各种因素的影响[40]。代码修改过程中的非修改因素指的是除修改代码以外的非技术因素，包括负责修改代码的开发人员、代码审核人员、代码修改任务所归属的项目等，本文中，将这些非修改因素归属为审核元特征。代码修改及审核过程中存在诸多非修改因素，这些非修改因素对代码审核的通过率产生了直接的影响。审核人员的工作量和经验、开发人员的经验和参与度会直接关系到代码审核流程的质量[39]。本文从代码审核数据中提取了10 种特征，用于描述代码修改及审核过程中非修改因素对代码修改完成时间的影响。

本文提取的代码审核元特征如表3-1所示。MF_0表示该修改提交的管理人员，MF_1表示该修改提交的作者，MF_2 表示修改提交的审核人员，我们用这三个特征来描述人力因素对完成周期的影响。通常同个软件项目下的开发人员和审核人员数量有限，不同的审核人员和开发人员对于软件项目熟悉程度的差异会导致软件修改工作所需要时间产生很大差距。同时，我们调研发现，审核人员对缺陷审查具有倾向性，有些审核人员容易忽略一些特定的代码缺陷；不同审核人员提供反馈意见的详细程度也存在差异，将直接影响开发人员的再次修改。MF_3表示该修改提交归属的项目，MF_4表示该修改提交所在的项目分支，我们用这两个特征衡量不同项目以及不同分支中代码修改工作的差异。不同项目以

表 3-1　代码审核文本特征及描述

类型	符号	描述
审核文本特征	MF_0	修改提交的管理人员
	MF_1	修改提交的作者
	MF_2	修改提交的审核人员
	MF_3	修改所归属的项目
	MF_4	修改所归属的项目分支
	MF_5	修改提交的时间
	MF_6	审核人员的数量
	MF_7	修改提交作者在项目中总的提交次数
	MF_8	修改提交作者在项目中近一个月的提交次数
	MF_9	相同审核人员和作者的修改的平均审核次数

及分支中软件实体复杂程度不同，常见修改目标也不同，这些差异将体现在开发人员完成修改工作所需要的时间。另外，我们统计还发现，通常一项修改的审核工作存在多位审核人员，因此，我们收集了所有的审核人员信息，并增加了特征MF_6，用于标记参与该修改提交的审核人员数量。

Eyolfson等人[65]研究了修改的提交时间与提交代码的正确性之间的关系，他们发现在深夜到凌晨4点之间提交的代码更容易出错，同时，在早上7点到中午之间提交的代码错误更少。另外，该研究还发现开发人员提交代码的频率也影响着代码的正确率。参考该研究的结论，我们从代码审核数据中提取了代码提交时间MF_5以及开发人员在项目中的活跃度特征，活跃度特征通过开发人员在项目中总的提交次数MF_7和近一个月的提交次数MF_8来衡量。

我们直接从审核数据中提取的特征为文本特征，如人名，项目名等。在输入预测模型前，需要将文本特征转换成数值特征，常用的编码方法有直接编码和独热编码（one-hot）。以特征MF_2为例，存在600多个不同的审核人员，使用直接编码的方式，将转换成一维特征和600多种取值；使用独热编码的方式，对于MF_2特征下600多个可能取值，将产生600 多个二元特征，这些特征互斥且每次仅有一个特征激活，即每个样本仅有少数特征取值为1（由于一个提交存在多为审核人员），其余特征取值都为0。两种编码方式都存在明显缺陷，审核人员之间相互独立，而直接编码导致取值相差小的两位审核人员比取值相差大的更加具有相似性，并且直接编码使得不同样本之间的区分度变低；采用独热编码

的方式虽然能让特征之间的距离更加合理，但是独热编码会导致特征向量过于稀疏，仅MF_2经编码后就产生600多个特征，不易于模型训练。针对这些问题，我们对直接编码和独热编码进行了结合和调整。如图3-3 所示，首先，将文本类别特征按首字母划分为26 类，再对每个类下的文本进行直接编码，这种方式增加了样本差异性并且降低了向量维度。

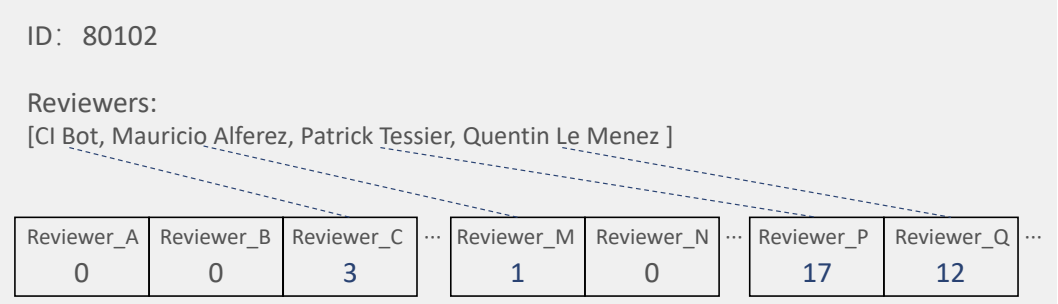


图 3-3　文本类别特征编码示例

3.2.2　代码结构耦合特征提取

软件系统经过多年的开发和维护，软件实体间存在复杂的关联关系。由于这些关联关系，代码修改会在软件实体间传播，从而增加了代码修改的工作的难度。根据代码修改传播机制[66]，如果两个类之间的结构耦合关系越复杂，则当其中一个类变更时，另一个类存在更大的可能被影响。因此，我们从修改代码中提取结构耦合特征，用于描述代码修改任务的复杂度。

在软件实体间，入度和出度是反应软件实体的结构耦合特征的有效方式，入度指当前实体引用修改提交中其他实体的次数；出度指的是其他软件实体引用当前实体的次数。修改所涉及的入度和出度能衡量该修改的复杂程度以及时间代价。本文从不同粒度提取代码实体之间的耦合特征共12维，如表3-2所示。下面我们按不同粒度分别介绍四种耦合规则，其中符号C、M、A分别表示类、方法、属性。

Class to Class: 根据面向对象编程语法，类层次的耦合关系最常见的有继承（Inheritance）和实现接口（Implementing Interface），C_i继承C_j或者实现C_j的接口，称C_j为C_i 的入度；反之，则称C_j 为C_i 的出度。本文提取四个特征用于衡量修改提交在类与类层次的耦合特征：特征CF_0为当前修改提交中类的平均入度；特征CF_1为当前修改提交中类的入度总和；特征CF_2为当前修改提交中类的平均

表 3-2　代码结构耦合特征及描述

类型	层次	符号	描述
代码耦合特征	Class to Class	CF_0	修改提交的类与类平均入度
		CF_1	修改提交的类与类入度总和
		CF_2	修改提交的类与类平均出度
		CF_3	修改提交的类与类出度总和
	Method to Class	CF_4	修改提交的方法与类平均入度
		CF_5	修改提交的方法与类入度总和
	Method to Attribute	CF_6	修改提交的方法与属性平均入度
		CF_7	修改提交的方法与属性入度总和
	Method to Method	CF_8	修改提交的方法与方法平均入度
		CF_9	修改提交的方法与方法入度总和
		CF_{10}	修改提交的方法与方法平均出度
		CF_{11}	修改提交的方法与方法出度总和

出度；特征CF_3为当前修改提交中类的出度总和。

Method to Class: 在软件实体中，类包含了方法集合和属性集合。类与方法的耦合关系指的是，在类C_i的某个方法M_i中使用了另一个类C_j，且不是使用C_j中的方法或属性，这时类C_j为方法M_i的入度这种耦合关系常见的实例包括：强制类型转换（Type-Casting），异常抛出类型（Exception Throws），参数类型（Parameter Type），返回类型（Return Type）和类型验证（Instanceof）等。本文提取两个特征衡量修改提交在方法与类层次的耦合特征：特征CF_4为修改提交中所有方法（*to Class*）的平均入度；特征CF_5为修改提交中所有方法（*to Class*）的入度总和。

Method to Attribute: 在面向对象编码示例中，方法与属性层次也经常存在耦合规则，它指的是位于类C_i中的某个方法M_i与类C_j中某个属性A_j之间的耦合关系，其中最常见的是静态属性调用（Static Attribute Invoking）。这种情况定义属性A_j为方法M_i的入度，我们提取两个特征用于描述方法与属性之间的耦合关系：特征CF_6表示修改提交中所有方法（*to Attribute*）的平均入度；特征CF_7为修改提交中所有方法（*to Attribute*）的入度总和。

Method to Method: 方法与方法层次指的是位于不同类C_i、C_j的两个方法M_i和M_j之间建立的耦合关系，满足这种耦合关系的实例有静态方法调用

（Static Method Invoking），构造方法调用（Construction Method Invoking）等。这种耦合关系具有方法性，因此，我们通过四个特征来表达方法间的耦合特征：特征CF_8为当前修改提交中方法（*to Method*）的平均入度；特征CF_9 为当前修改提交中方法（*to Method*）的入度总和；特征CF_{10}为当前修改提交中方法（*to Method*）的平均出度；特征CF_{11}为当前修改提交中方法（*to Method*）的出度总和。

3.2.3　代码修改特征

代码修改特征对于代码修改工作的时间代价具有很强的指向作用，我们从两个维度来提取修改提交中的代码修改特征，分别是代码修改量和代码修改内容。代码修改量直接体现了一项修改的工作量，而代码修改内容则是描述修改需要解决的问题和目标。代码修改量与修改内容都直接影响着代码修改阶段以及代码审核阶段所需的时间。

具体地，在代码修改量维度，可以细粒度地划分为类层次的修改量、方法层次的修改量、语句层次的修改量；而从代码变更的类型，还可以划分为增加（*add*）、修改（*change*）和删除（*delete*）。如表3-3所示，本文提取了18维特征用于描述代码修改任务中修改量。

特征$MF_0 \sim MF_2$表示涉及变更的代码语句数量，包括新增代码语句的数量、修改代码语句的数量和删除代码语句的数量。特征$MF_3 \sim MF_6$描述涉及修改的类的数量，分别代表新增类的数量、修改类的数量、删除类的数量以及类名修改的数量。其中类名的修改会引入一系列类引用上的修改问题。在方法层次，特征$MF_7 \sim MF_{10}$分别代表方法新增的数量，方法删除的数量，方法名变更数量以及方法返回类型修改的数量。特征$MF_{11} \sim MF_{14}$代表涉及变更的参数数量，包括参数类型修改，参数名修改以及参数增加和删除。特征$MF_{15} \sim MF_{16}$分别表示属性类型与属性名的修改。另外，我们还发现提交表达式的修改是代码修改工作中频率较高的修改类型，并且条件语句的修改会对代码流程产生较大影响，因此我们用特征MF_{17}表示涉及修改的条件表达式的数量。

在修改内容维度，我们采用与本文2.2节中相同的方法。我们对所有修改提交中涉及修改的代码片段进行文本预处理，再以预处理后的文本构建语料库。

表 3-3　代码修改量特征及描述

类型	符号	描述
代码修改量特征	MF_0	新增代码行数
	MF_1	修改代码行数
	MF_2	删除代码行数
	MF_3	新增类的数量
	MF_4	修改类的数量
	MF_5	删除类的数量
	MF_6	重命名类的数量
	MF_7	新增方法的数量
	MF_8	删除方法的数量
	MF_9	重命名方法数量
	MF_{10}	变更返回类型的方法数量
	MF_{11}	参数类型变更的数量
	MF_{12}	参数名变更数量
	MF_{13}	新增参数数量
	MF_{14}	删除参数数量
	MF_{15}	属性名变更数量
	MF_{16}	属性类型变更数量
	MF_{17}	条件表达式变更数量

最后，通过Word2vec方法训练词向量模型，以50维的特征向量表示修改提交中的修改内容。

3.3　机器学习算法选择及模型评估方法

本文提出的代码修改周期预测方法基于有监督机器学习算法，具体地，我们通过代码修改的“修改-审核”次数评估代码修改的周期长短。我们将代码修改周期预测视为分类问题，然后按时间长短将代码修改周期划分为三个层次：较短、一般和较长，分别对应时间为1天内、一周内和一周以上。结合图3-1，我们根据时间长短将“修改-审核”次数与修改周期长短相对应。我们从开源项目的历史修改和审核数据中提取审核元特征，代码耦合特征以及代码修改特征，再基于这些特征训练机器学习模型。最后使用训练好的模型对新的修改任务预测修改周期。

3.3.1　机器学习算法选择

在本文中，采用有监督机器学习算法。常见的分类算法有逻辑回归（*LR*，Logistic Regression），支持向量机（*SVM*，Support Vector Machines），随机森林（*RF*，Random Forest），人工神经网络（*ANN*，Artificial Neural Networks）以及*XGBoost*（Extreme Gradient Booting）。在实验中，我们对比各种分类算法在两个开源项目中的分类效果后，选择表现最优的*XGBoost* 作为我们的修改周期预测模型。

XGBoost 是梯度提升树模型（Gradient Boosting Trees）的一种高效实现方式。*XGBoost* 集成了K颗*CART* 树（Classification and Regression Trees）$\{Tr_1(x_1, y_1), Tr_2(x_2, y_2)..., Tr_k(x_k, y_k)\}$。其中，$x_i$ 表示第i^{th} 个样本的特征，y_i 则表示该样本对应的标签。*CART* 树将分配给每个叶子节点一个分数，而最终的预测结果将由所有*CART* 树上对应叶子节点的分数通过加法模型得到，如公式所示3.1。

$$\hat{y} = \sum_{k=1}^{K} f_k(x_i), \qquad f_k \in F \tag{3.1}$$

在上述公式中，$f_k(x_i)$ 为第k^{th} 颗树的预测分数，F 则代表函数空间中所包含的所有树模型。相比于传统的梯度提升树模型，*XGBoost* 一个重要的改进是在目标函数中加入了正则项。公式3.2为*XGBoost*的目标函数。

$$Obj(\Theta) = \sum_{i=1}^{n} l(y_i, \hat{y_i}) + \sum_{k}^{K} \Omega(f_k) \tag{3.2}$$

其中，l 为损失函数，描述的是预测标签$\hat{y_i}$ 与真实标签y_i 之间的误差。而在表达式$\sum_k^K \Omega(f_k)$ 中，$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} \omega_j^2$ 用于描述模型的复杂度。T 和ω 分别表示叶子数量及对应叶子节点上的取值。γ 和λ 则用于控制正则化的惩罚程度。

3.3.2　模型评估方法

为了评估修改周期预测模型的有效性，我们采用了分类模型评价中常用的四个指标：召回率、精确率、准确率以及F_1值。召回率用于度量模型对特定类别样本的预测安全性，如对一些修改周期较长的样本，是否能准确预测；精确率用于度量分类模型的预测结果的精确性；而准确率及F_1则考察模型对多个类

别预测的综合效果。四个指标的定义如下：

$$precision = \frac{TP}{TP + FP} \tag{3.3}$$

$$recall = \frac{TP}{TP + FN} \tag{3.4}$$

$$F_1 = 2 * \frac{precision * recall}{precision + recall} \tag{3.5}$$

$$accuracy = \frac{TP + FN}{TP + FN + TN + FP} \tag{3.6}$$

以二分类为例（多分类视为多个二分类问题），TP 和TN 分别表示正确预测的正样本数量和负样本数量，FP 和FN分别表示错误预测的正样本数量和负样本数量。

3.4　本章小结

软件维护中的代码修改工作通常包括了代码修改与代码审核两个阶段，这两个阶段共同决定了完成代码修改工作所需要的时间。在本章，我们提出了一种修改周期的预测方法，我们从代码修改和代码审核两个角度考虑代码修改可能需要的时间长短。具体地，我们从开源项目的历史代码修改数据和代码审核数据中，提取可判别特征，包括代码审核元特征、代码耦合特征及代码修改特征。最后，我们再基于机器学习算法训练预测模型，该模型对一项代码修改工作预测其可能需要经过“修改- 审核”的次数，再以次数评估其修改周期的长短。

第 4 章 代码修改影响分析与修改周期预测实验分析及工具应用

本章结合第二章与第三章的研究内容，介绍代码修改影响分析与修改周期预测的实验设计与评估，以及从技术角度讨论相关应用工具的设计与实现。在修改影响分析实验中，我们对两个传统影响分析工具在8个项目上的影响分析结果进行优化调整，实验结果证实了我们方法的有效性。在代码修改周期预测实验中，我们收集了两个开源项目的修改历史信息，通过两个数据集训练模型，实验结果也表明了我们的方法对修改周期预测的可行性。另外，工具应用的实现采用MVC（Model-View-Controller）模式，实现平台为Java，系统框架为Spring Boot，系统前端则使用Thymeleaf引擎，后端数据库采用MongoDB。

4.1 代码修改影响分析实验设计与评估

在本节中将介绍影响分析辅助方法的实验设计与评估。分别介绍实验数据的收集和去噪过程，实验的评估方法，以及实验结果。

4.1.1 数据集收集

我们从开源项目中选取8个项目进行影响分析实验。这些项目均保持着长期维护，有大量优质提交信息，并且是在软件工程领域常用的开源项目。这些项目分别是FreeCol①，HSQLDB②，HtmlUnit③，JAMWiki④，jEdit⑤，jHotDraw⑥，Makagiga⑦，OmegaT⑧。这些开源项目代表了不同的应用领域和开发环境。

同时，为了保证实验的可靠性，需要使用本文2.2.1节中提出的提交数据优化方法对实验数据集进行优化，去除只涉及一个修改类的提交（这种类型的提交不包含影响集）以及涉及20个修改类的提交（该类提交通常由多个普通提交

① http://www.freecol.org/
② http://hsqldb.org/
③ http://htmlunit.sourceforge.net/
④ http://www.jamwiki.org/
⑤ http://www.jedit.org/
⑥ http://www.jhotdraw.org/
⑦ https://makagiga.sourceforge.io/
⑧ https://omegat.org/

表 4-1 实验项目数据统计信息

项目名	版本跨度	提交个数	项目名	版本跨度	提交个数
FreeCol	0.7-1.0	1150	HSQLDB	2.20-2.33	637
HtmlUnit	2.0-2.21	737	JAMWiki	0.82-1.3	499
jEdit	4.0-4.5	1601	JHotDraw	7.1-7.5	378
Makagiga	3.82-4.12	2072	OmegaT	2.3-3.54	627

组合而成）。项目的详细信息如表4-1所示。

4.1.2 实验评估准则

影响分析研究中广泛使用召回率和精确率作为结果的度量指标，这两个指标能评估实验产生的预估影响集与真实修改产生的实际影响集之间的差异。给定软件实体e_s（本文影响分析实体为类），用E_{imp}表示影响分析结果中软件实体的集合（预估影响集），用R_{imp} 表示实际修改中被影响的软件实体的集合（实际影响集）。则实验结果的召回率由公式4.1定义，精确率由公式4.2定义。此外，为了更细致的评估实验结果，我们对预估影响集按不同截断点进行划分，分别评估实验影响集前5、前10、前20、前50的召回率和精确率，如公式4.3和公式4.4定义。

$$Recall = \frac{1}{n}\sum_{i=1}^{n}\frac{|E_{imp}\bigcap R_{imp}|}{|R_{imp}|} \times 100\% \tag{4.1}$$

$$Precision = \frac{1}{n}\sum_{i=1}^{n}\frac{|E_{imp}\bigcap R_{imp}|}{|E_{imp}|} \times 100\% \tag{4.2}$$

$$Recall(cutpoint) = \frac{1}{n}\sum_{i=1}^{n}\frac{|E_{imp}(cutpoint)\bigcap R_{imp}|}{|R_{imp}|} \times 100\% \tag{4.3}$$

$$Precision(cutpoint) = \frac{1}{n}\sum_{i=1}^{n}\frac{|E_{imp}(cutpoint)\bigcap R_{imp}|}{cutpoint} \times 100\% \tag{4.4}$$

为了验证本文提出的影响分析辅助方法的有效性，我们选取两个影响分析工具$JRipples$和$ImpactMiner$作为传统影响分析方法，用于产生初始影响集。$JRipples$ 工具是Eclipse官方插件，该插件是Java代码影响分析领域中被广泛使用的工具，也是实际开发中开发人员常用的影响分析工具。而$ImpactMiner$则

是可用工具中影响分析效果较好的一款工具，另外，$ImpactMiner$工具结合了多种影响分析中的研究方法，包括：信息检索，软件历史库挖掘以及动态影响分析。

本文实验中的验证数据为开源项目中提交数据，无法直观判断其修改集和影响集。因此，我们通过关键类判定方法识别提交中的关键类，将关键类作为实验的修改集，提交中所有非关键类作为影响集。

在实验中，我们主要关注以下四个方面的问题：

研究问题1:我们的方法是否能提高传统影响分析的效果? 该研究问题是本文工作的出发点，我们试图通过相似的历史修改信息来指导当前修改工作的影响分析。同时，考虑到开发人员经常需要修改完成前，进行影响分析，在研究问题1中，相似提交检索只使用注释文本及修改前代码片段。

研究问题2：相似提交检索中加入修改后代码片段是否能提高召回率和精确率? 在本文提出的影响分析方法的相似提交检索步骤中，默认是通过代码修改前片段及修改注释信息（修改目标描述）的相似度来衡量提交与当前修改的相似度。在研究问题2中，我们考虑在相似提交检索中加入修改后代码片段的相似度，研究影响分析的效果是否有提升。若效果提升，则在可使用代码修改后片段（需完成目标修改）的影响分析中，加入修改后代码片段的相似度。

研究问题3：以关键类作为修改的等价类是否能提高影响分析的效果? 相似提交检索以及相似耦合关系的检索中，本文方法都是以提交中的关键类作为修改的等价类进行相似性的测量，研究问题3目标是测试关键类是否能提高影响分析的效果。

研究问题4：结构耦合关系相似程度是否影响召回率和精确率? 在影响集的优化调整过程中，我们基于结构耦合关系的相似性实现对相似提交中关键类的修改模式的映射。结构耦合关系相似度指的是相似提交中关键类与非关键类的耦合关系和当前修改类与初始影响集中类的耦合关系的相似度。若相似度大于阈值η，则调整初始影响集中该类的位置。研究问题4关注于结构耦合关系相似程度是否影响召回率和精确率，即阈值η对影响分析结果的影响。

4.1.3 实验结果分析

研究问题1:我们的方法是否能提高传统影响分析的效果?

表 4-2 基于$JRipples$的影响分析辅助结果

项目名	方法	召回率（%）					精确率（%）				
		5	10	20	30	50	5	10	20	30	50
FreeCol	JRipples	13.68	20.52	25.50	28.15	28.72	12.65	9.53	6.01	4.37	2.68
	Our Method	14.65	20.42	25.49	28.72	28.72	13.65	9.70	28.72	28.72	2.68
HSQLDB	JRipples	16.43	24.76	42.03	55.67	61.05	22.56	18.33	15.13	13.04	8.93
	Our Method	18.10	30.53	47.12	55.67	61.05	24.56	21.14	16.18	13.04	8.93
HtmlUnit	JRipples	15.17	18.93	25.19	26.76	28.63	11.55	7.25	4.79	3.48	2.28
	Our Method	17.00	19.89	25.91	27.30	28.63	13.14	8.74	4.82	3.52	2.28
JAMWiki	JRipples	24.08	34.12	38.60	40.07	40.16	16.30	12.35	7.35	5.15	3.11
	Our Method	32.27	36.36	38.90	40.07	40.16	23.19	13.69	13.69	13.69	3.11
jEdit	JRipples	23.59	34.18	43.77	47.28	50.10	16.15	11.15	6.90	5.19	3.31
	Our Method	29.41	37.23	44.83	48.34	50.10	18.07	12.12	7.02	5.32	3.11
JHotDrow	JRipples	12.82	16.86	17.29	18.34	18.82	11.07	8.39	4.38	3.21	2.04
	Our Method	15.10	18.45	17.39	18.34	18.82	13.77	13.77	4.44	3.21	2.04
Makagiga	JRipples	21.78	30.67	39.54	46.34	51.74	24.04	17.41	12.13	9.78	6.99
	Our Method	23.22	31.32	40.09	49.83	51.48	25.74	17.93	12.30	9.91	6.99
OmegaT	JRipples	39.78	43.93	44.54	44.54	44.54	22.04	12.04	6.02	6.02	6.02
	Our Method	41.41	44.03	44.54	44.54	44.54	23.65	13.04	6.02	6.02	6.02

我们分别基于$JRipples$工具和$ImpactMiner$工具在8个项目上评估了影响分析辅助方法效果，实验结果如表4-2和表4-3所示。实验结果显示，我们的方法在8 个项目上对两个工具的初始影响集都起到了辅助效果，证实了本文提交出方法都有效性。其中，在$JRipples$基础上，我们的方法在8个项目上影响集前5和前10召回率平均提升了14.5%和6.9%，影响集前5和前10的精确率平均提升了15.4%和16.6%。而在$ImpactMiner$的基础上，我们的方法在8个项目上的实验结果显示，影响集前5和前10的召回率平均提升了16.64%和12.3%，影响集前5和前10的精确率平均提升了25.1%和16.1%。观察可以发现，我们的方法对$ImpactMiner$的辅助效果更好，我们研究发现这是由于$ImpactMiner$ 的初始影响集中包含更多实际影响类，给我们的影响分析辅助方法提供了更大的调整空间。

从表4-2中可以发现，在所有项目的前50召回率和精确率中，我们的方法的结果于基础影响分析方法的结果相同，这是由于我们实际设置的初始影响集大小为50，在影响集优化调整过程中，最终影响集也只包含初始影响集的50个结果。此外，在一些项目的实验结果中前30 的召回率和精确率是相同的，例如JAMWiki 和OmegaT 两个项目。我们从实验数据中发现，这是由于$JRipples$

表 4-3 基于$ImpactMiner$的影响分析辅助结果

项目名	方法	召回率（%）					精确率（%）				
		5	10	20	30	50	5	10	20	30	50
FreeCol	ImpactMiner	18.22	24.78	31.23	34.78	38.39	18.15	14.63	12.92	9.63	5.31
	Our Method	22.47	28.68	33.21	35.98	38.39	22.12	17.35	14.21	10.39	5.31
HSQLDB	ImpactMiner	34.33	39.03	47.99	58.33	63.87	30.13	24.11	21.69	17.56	14.02
	Our Method	41.09	45.42	51.08	60.77	63.87	33.77	27.65	23.00	18.14	14.02
HtmlUnit	ImpactMiner	18.33	22.43	28.06	29.97	31.70	15.48	11.98	9.97	7.53	5.89
	Our Method	20.17	24.08	29.61	31.15	31.70	17.44	13.70	11.31	9.65	5.89
JAMWiki	ImpactMiner	33.80	38.21	41.26	45.04	49.19	19.09	16.38	12.82	9.95	7.38
	Our Method	39.10	42.44	45.37	47.39	49.16	23.41	18.64	14.63	11.85	7.38
jEdit	ImpactMiner	34.39	47.13	54.22	57.33	62.12	15.13	21.32	18.54	15.23	11.33
	Our Method	41.44	53.33	58.83	60.34	62.12	30.11	28.49	23.12	19.83	11.33
JHotDrow	ImpactMiner	16.72	21.02	24.26	27.11	29.30	13.07	11.55	9.87	8.05	5.15
	Our Method	18.25	24.54	27.24	28.12	29.30	14.25	12.54	11.06	8.49	5.15
Makagiga	ImpactMiner	28.31	35.76	40.01	48.21	53.87	29.33	23.54	17.65	13.33	10.94
	Our Method	33.64	39.09	44.84	51.59	53.87	33.04	27.81	22.02	16.52	10.94
OmegaT	ImpactMiner	44.34	50.43	53.66	56.71	61.63	26.54	22.76	19.42	16.21	13.11
	Our Method	51.39	54.93	58.01	59.82	61.63	29.45	24.35	21.10	18.61	13.11

在这两个项目上影响分析产生的初始影响集大小小于30，导致我们取前30评估实验结果是相同的。同时，我们可以发现，当传统影响分析方法前50召回率或精确率更高时，我们的方法的辅助效果更加明显。这是因为初始影响集中包含更多的实际影响类，在影响集重排序时，我们的方法有更大几率将实际影响类的排序位置往前移动。

综上所述，本文提出的基于历史修改模式的影响分析辅助方法在多个项目上能提升传统影响分析工具的效果，验证了本文方法的有效性。

研究问题2：相似提交检索中加入修改后代码片段是否能提高召回率和精确率?

对于研究问题2，我们以$ImpactMiner$影响分析工具为基础在3个项目上进行影响分析测试，并且在相似提交的检索中加入了修改后代码的相似性。实验结果如表4-4所示，其中，$OurMethod$对应相似提交检索中只使用注释文本（修改目标）和修改前代码两部分的相似度。$OurMethod2$对应的是在前面方法的基础上增加修改后代码的相似度，综合三部分的相似度从提交库中检索相似提交。

观察实验结果可以发现，$OurMethod2$在三个项目上对$ImpactMiner$的影响

表 4-4 增加修改后代码相似度的实验结果

项目名	方法	召回率（%）					精确率（%）				
		5	10	20	30	50	5	10	20	30	50
jEdit	ImpactMiner	34.39	47.13	54.22	57.33	62.12	15.13	21.32	18.54	15.23	11.33
	Our Method	41.44	53.33	58.83	60.34	62.12	30.11	28.49	23.12	19.83	11.33
	Our Method 2	42.56	56.81	59.34	60.55	62.12	31.58	30.32	25.66	19.87	11.33
JHotDraw	ImpactMiner	16.72	21.02	24.26	27.11	29.30	13.07	11.55	9.87	8.05	5.15
	Our Method	18.25	24.54	27.24	28.12	29.30	14.25	12.54	11.06	8.49	5.15
	Our Method 2	19.89	27.45	27.96	28.24	29.30	15.11	13.69	11.26	9.04	5.15
HSQLDB	ImpactMiner	34.33	39.03	47.99	58.33	63.87	30.13	24.11	21.69	17.56	14.02
	Our Method	41.09	45.42	51.08	60.77	63.87	33.77	27.65	23.00	18.14	14.02
	Our Method 2	42.19	48.78	52.65	61.34	63.87	33.77	28.42	25.64	18.21	14.02

分析结果都起到了提升效果，证明通过注释文本、修改前代码以及修改后代码相似度检索相似提交的方法是有效的。同时，$OurMethod2$ 与$OurMethod$的影响分析结果相比，也有明显的提高，说明结合修改前后两个版本的代码片段的相似度，能更准确的反应历史提交与当前修改的相似性。值得注意的是，本文默认只使用修改前代码片段，这是由于大多数影响分析都是应用于修改之前。另外，如果开发人员在修改之后影响分析，我们的方法也证明了增加修改后的代码，能得到更准确的影响分析结果。

研究问题3：以关键类作为修改的等价类是否能提高影响分析的效果?

研究问题3的对比实验中，我们从提交中随机选择一个类作为当前修改类的等价类。在相似提交检索中，相似度计算使用该类的修改代码片段，且在影响集优化中的结构耦合关系相似度计算中，也使用该类与提交中其他类的耦合关系。

在实验中，我们在3个项目上对两种方法进行了对比，其中基础影响分析工具选择的是$ImpactMiner$。实验结果如表4-5所示，其中，$Comparison$代表对比的方法。实验结果显示，对比方法的影响分析效果要明显低于我们的方法。并且随机选择一个类作为等价类的对比方法，对基础影响分析工具$ImpactMiner$的结果不仅没有提高，反而有所降低。其原因是显而易见的，如果要引入历史提交的修改模式，必须找到提交中核心修改的类，以该类作为修改的起点，而其他类作为修改传播的结果。利用关键类识别方法有助于我们从提交中找到其核心修改的类。

表 4-5　使用关键类识别的实验结果

项目名	方法	召回率（%）					精确率（%）				
		5	10	20	30	50	5	10	20	30	50
jEdit	ImpactMiner	34.39	47.13	54.22	57.33	62.12	15.13	21.32	18.54	15.23	11.33
	Our Method	41.44	53.33	58.83	60.34	62.12	30.11	28.49	23.12	19.83	11.33
	Comparison	31.21	42.55	49.32	54.35	62.12	13.44	19.23	16.58	15.03	11.33
JHotDraw	ImpactMiner	16.72	21.02	24.26	27.11	29.30	13.07	11.55	9.87	8.05	5.15
	Our Method	18.25	24.54	27.24	28.12	29.30	14.25	12.54	11.06	8.49	5.15
	Comparison	15.12	20.15	23.44	25.89	29.30	11.49	10.78	8.62	7.88	5.15
HSQLDB	ImpactMiner	34.33	39.03	47.99	58.33	63.87	30.13	24.11	21.69	17.56	14.02
	Our Method	41.09	45.42	51.08	60.77	63.87	33.77	27.65	23.00	18.14	14.02
	Comparison	28.33	31.09	38.89	49.54	63.87	22.39	19.57	18.00	16.31	14.02

研究问题4：结构耦合关系相似程度是否影响召回率和精确率?

对于第四个研究问题，我们以$JRipples$为传统影响分析工具在3个项目上测试不同结构耦合关系相似程度对召回率和精确率的影响。如本文2.5.2节所述，相似度阈值为η，当提交中关键类和一个非关键类的耦合关系与修改类与一个初始影响集中影响类的耦合关系的相似度大于阈值，认为修改类的修改影响会传播给该影响类，并增加该影响类的分数。实验测试不同η下影响分析的召回率和精确率。实验结果如表4-6所示，我们可以发现当相似度阈值η从0.3，0.5，0.8一次增加时，影响分析的召回率和精确率在不断提升，但η从0.8增加到0.9时，影响分析的召回率和精确率是下降的。同时我们可以发现，η取值为0.3时，影响分析效果低于$JRipples$，说明实验对$JRipples$ 产生的初始影响集产生负面调整。

JRipples

排名	类名	分数
1	*SettingsReloader.java*	240
2	*jEdit.java*	260
3	*MiscUtilities.java*	220
4	*Log.java*	200
5	*FileVFS.java*	180
6	*GUIUtilities.java*	160
7	*EditServer.java*	140
8	*VFSManager.java*	120
9	*GrabKeyDialog.java*	100
10	*JARClassLoader.java*	80
11	*VFSBrowser.java*	60
12	*ToolBarManager.java*	40
13	*SearchDialog.java*	20
14	*JEditTextArea.java*	0

$\eta = 0.3$

排名	类名	分数
1	*MiscUtilities.java*	580
2	*Log.java*	520
3	*EditServer.java*	480
4	*SettingsReloader.java*	460
5	*jEdit.java* ↓	400
6	*GUIUtilities.java*	340
7	*FileVFS.java* ↓	320
8	*VFSManager.java*	280
9	*GrabKeyDialog.java*	260
10	*ToolBarManager.java*	260
11	*JEditTextArea.java* ↑	240
12	*SearchDialog.java* ↓	180
13	*VFSBrowser.java*	140
14	*JARClassLoader.java*	140

$\eta = 0.8$

排名	类名	分数
1	*jEdit.java* ↑	380
2	*Log.java*	340
3	*FileVFS.java* ↑	320
4	*SettingsReloader.java*	280
5	*JEditTextArea.java* ↑	240
6	*SearchDialog.java* ↑	220
7	*MiscUtilities.java*	160
8	*GUIUtilities.java*	160
9	*JARClassLoader.java* ↑	140
10	*EditServer.java*	140
11	*VFSManager.java*	120
12	*GrabKeyDialog.java*	100
13	*VFSBrowser.java*	60
14	*ToolBarManager.java*	60

$\eta = 0.9$

排名	类名	分数
1	*jEdit.java* ↑	300
2	*Log.java*	260
3	*SettingsReloader.java*	240
4	*MiscUtilities.java*	220
5	*FileVFS.java*	180
6	*EditServer.java*	160
7	*GUIUtilities.java*	160
8	*VFSManager.java*	120
9	*GrabKeyDialog.java*	100
10	*JARClassLoader.java*	80
11	*SearchDialog.java* ↑	80
12	*VFSBrowser.java*	60
13	*ToolBarManager.java*	40
14	*JEditTextArea.java*	40

图 4-1　不同η下影响集优化示例

为了进一步分析耦合关系相似度如何影响实验结果，我们随机从实验数

表 4-6 不同相似阈值η下实验结果

项目名	方法	η	召回率（%）					精确率（%）				
			5	10	20	30	50	5	10	20	30	50
jEdit	JRipples		23.59	34.18	43.77	47.28	50.10	16.15	11.15	6.90	5.19	3.31
	Our method	0.3	17.23	27.32	35.11	42.65	50.10	13.17	9.88	6.34	5.19	3.31
		0.5	23.62	33.75	42.68	46.44	50.10	16.21	11.66	6.90	5.24	3.31
		0.8	29.41	37.23	44.83	48.34	50.10	18.07	12.12	7.02	5.32	3.31
		0.9	26.67	35.54	43.12	47.19	50.10	16.46	12.08	6.95	5.32	3.31
JHotDraw	JRipples		16.72	21.02	24.26	27.11	29.30	13.07	11.55	9.87	8.05	5.15
	Our method	0.3	13.31	16.76	20.01	25.59	29.30	9.81	7.65	6.02	5.52	5.15
		0.5	16.76	20.87	25.35	28.81	29.30	12.84	11.41	9.25	8.13	5.15
		0.8	18.25	24.54	27.24	28.12	29.30	14.25	12.54	11.06	8.49	5.15
		0.9	17.09	23.84	25.59	27.36	29.30	13.57	12.38	10.88	8.21	5.15
HSQLDB	JRipples		12.82	16.86	17.29	18.34	18.82	11.07	8.39	4.38	3.21	2.04
	Our method	0.3	8.54	12.10	15.89	16.34	18.82	8.95	6.34	3.21	2.54	2.04
		0.5	11.99	15.88	17.16	17.55	18.82	11.15	8.41	4.25	3.21	2.04
		0.8	15.10	18.45	17.39	18.34	18.82	13.77	9.26	4.44	3.21	2.04
		0.9	15.04	17.84	17.99	18.29	18.82	12.67	9.06	4.35	3.23	2.04

据中提取一个提交的影响分析结果进行分析（如图4-1所示），我们可以发现阈值η取值越低，影响集中每个类的最终的得分越高，说明类被调整的次数也越多，η取值越高，则相反。这是由于阈值低的时候，能找到更多相似耦合关系对（关键类-非关键类），这同时也造成了更多的负面调整。结合图中实际影响类在不同影响集中的位置可以发现，阈值越高，实际影响类被有效调整几率越高，排名越有可能提高。但是，阈值太高时（例如η取0.9），由于符合相似性的耦合关系对太少，导致影响集得到的调整也变少。

综上所述，结构耦合关系的相似性可以用于映射修改模式，当相似度较低时，会对影响集产生较大调整，包括大量负面调整，当相似度过高时，则对影响集产生的调整过小。实验验证，当相似度阈值η 取值为0.8时，基于机构耦合关系相似性的历史修改模式映射取得最好效果。

4.2 代码修改周期预测实验设计与评估

本节首先描述了实验数据的收集及去噪处理方法，然后介绍了在修改周期预测实验中关注的三个研究问题，最后分析了模型预测的实验结果。

4.2.1 数据收集及去噪处理

我们从Gerrit代码审核工具中收集开源项目的历史提交信息以及代码审核信息。我们选择了Eclipse[①]和OpenDaylight[②]两个项目作为实验数据集。其中，Eclipse是使用最广泛的集成开发环境之一，而OpenDaylight是一种高度可用、模块化以及可扩展的多协议控制器基础架构，用于供应商网络上的SDN部署。Eclipse和OpenDaylight都拥有丰富的代码修改历史数据，可用于代码修改周期的分析。表格展示了实验数据集的详细信息。

表 4-7 代码修改周期预测实验数据集

Projects	Time span	No. of reviews
Eclipse	2016.1-2018.5	9455
OpenDaylight	2017.1-2018.8	6403

另外，我们对大量代码修改数据进行观察，发现其中存在一些噪声数据，这些数据对模型的训练产生负面影响。我们设置了一下几条规则对噪声数据进行过滤：

1) 在代码修改周期的预测中，我们需要从被修改的代码中提取代码修改特征及代码耦合特征。许多提交中不涉及代码修改，我们将过滤这部分的提交数据。

2) 我们同时去除数据集中，明显存在异常的修改提交。例如，有一个修改提交需要经过150次的反复修改和审核，而所有修改数据的平均修改和审核次数为5次。

3) 另外，我们还去除了项目中不活跃的开发人员的修改提交。我们认为修改提交次数低于两次的开发人员为项目中的非核心维护人员，这些开发人员只在项目中完成极少数的代码修改工作。

4.2.2 代码修改周期预测的问题研究

文章的主要目的是从代码修改及代码审核两个阶段分析代码修改周期，试着通过代码修改及审核次数预测代码修改周期的长短。本节从多个角度研究，

① https://www.eclipse.org/
② https://www.opendaylight.org/

评估本文提出的代码修改周期预测方法的有效性。我们关注的研究问题有以下三个：

研究问题1：本文方法对代码修改任务能否在较短周期内完成的预测效果如何?

开发人员都希望代码修改任务能在较短的时间周期内完成，即一次修改和审核周内，其中关键的是代码修改不存在问题并且在代码审核过程被接受。值得注意的是，在这个研究问题中，我们将审核次数为1的提交数据视为正样本，其余提交数据视为负样本。因此，我们使用本文提出的方法，训练预测模型，预测代码修改任务的审核次数是否为1。数据集中正负样本分布如表4-8所示。

表 4-8 按周期是否较短划分的样本分布

Projects	Positive	Negative
Eclipse	2905	6551
OpenDaylight	1598	4807

研究问题2：本文方法对代码修改周期的长短的预测效果如何?

观察图3-1可以发现，一项代码修改任务所花费时间周期的长短可以根据代码修改和审核次数来划分。在这个研究问题中，我们通过划分审核次数区间，来对应代码修改周期长短的层次。审核次数为1的提交的平均修改周期为1天，我们将这类修改对应为“较短”的代码修改周期；审核次数在2-6区间的提交的平均修改周期为1周内，我们认为这类修改任务对应代码修改周期长短为“一般”；而审核次数在7-20区间的提交的平均修改周期大于一周，我们则认为这类修改任务需要“较长”的修改周期。经过划分后的数据分布如表4-9所示。

表 4-9 不同修改周期的样本分布

Projects	较短	一般	较长
Eclipse	2905	5203	1287
OpenDaylight	1598	3327	1338

研究问题3：三种类型的特征对模型预测效果的影响如何?

本文方法的核心是从代码修改的历史信息中提取审核元特征、代码耦合特征及代码修改特征，用于度量代码修改的时间周期长短。三种特征代表了代码

修改阶段及代码审核阶段对代码修改周期的影响。在这个研究问题中，我们关注于三种类型各自的特征对模型预测效果的影响。

4.2.3 实验结果与分析

研究问题1：本文方法对代码修改任务能否在较短周期内完成的预测效果如何?

表 4-10 对修改周期是否较短的预测实验结果

项目名	方法	正样例			负样例			准确率
		精确率	召回率	F1	精确率	召回率	F1	
Eclipse	LR	69.39	33.72	45.38	74.84	92.98	82.93	73.99
	SVM	69.89	42.98	53.22	77.24	91.27	83.67	75.79
	RF	75.85	40.00	52.55	76.85	94.23	84.69	76.85
	ANN	63.21	49.42	55.47	78.37	86.44	82.21	74.58
	XGBoost	71.63	54.80	62.10	82.16	90.56	86.16	79.72
OpenDayLight	LR	60.47	33.66	43.24	78.36	78.36	84.66	74.86
	SVM	41.38	19.41	26.43	77.52	90.10	83.72	73.34
	RF	51.75	43.04	46.99	82.32	86.86	84.53	76.06
	ANN	49.18	9.71	16.22	75.59	96.71	85.49	75.26
	XGBoost	50.39	50.78	50.58	87.57	87.40	87.49	80.03

表格4-10展示了本文方法对代码修改任务能否在较短周期内完成的预测效果，包括正负样本的召回率、精确率和F_1值，以及综合预测效果的准确率。在实验中，我们测试对比了本文提出方法在多个机器学习模型上的效果，并采用10折交叉验证方法。针对数据集样本分布不平衡问题，使用SMOTE算法生成训练集中占比较低的样本，测试集中不做改变。

观察表4-10可以发现，在两个数据集上不同机器学习模型的预测准确率分布为73.34%至80.03%，这体现了本文提出方法对代码修改任务能否在较短周期内完成的预测的准确性。对比不同机器学习模型的效果，可以发现*XGBoost*取得了最优的表现。在Eclipse数据集上，*XGBoost*对负样本预测的召回率、精确率和F_1值分别是82.16%，90.56%和86.16%，意味着大部分负样本都能被准确预测。负样本代表可能需要重复多次修改的提交，对该类样本的预测有助于开发人员和项目管理提前做出调整和安排。另一方面，在正样本的预测效果上，多种机器学习模型的效果都要低于在负样本上表现。我们认为原因是SMOTE算法并不

表 4-11 不同修改周期的预测实验结果

项目名	方法	较短			一般			较长			准确率
		精确率	召回率	F1	精确率	召回率	F1	精确率	召回率	F1	
Eclipse	LR	66.67	22.64	33.79	61.20	60.90	61.05	29.70	74.24	42.42	50.72
	SVM	64.04	43.92	52.10	64.07	57.18	60.04	33.57	71.21	45.63	54.98
	RF	60.00	54.73	57.24	64.96	72.14	68.36	46.31	35.61	40.26	61.52
	ANN	63.81	40.20	49.33	62.57	72.73	67.27	41.32	49.62	45.09	59.23
	XGBoost	69.59	54.90	61.38	66.45	80.94	72.98	57.23	35.99	44.19	66.42
OpenDayLight	LR	52.14	23.62	32.52	59.73	61.68	60.69	44.22	65.89	52.93	53.31
	SVM	44.48	23.95	31.03	57.58	65.11	61.11	47.08	55.96	51.13	52.75
	RF	48.57	44.01	46.18	59.10	69.31	63.80	54.09	39.40	45.59	55.84
	ANN	55.10	17.47	26.54	57.88	62.31	60.01	43.10	66.23	52.22	52.19
	XGBoost	60.10	37.54	46.22	61.03	77.57	68.31	58.61	47.35	52.38	60.42

能完全解决样本不平衡的问题。

另外，我们可以发现*XGBoost* 的负样本召回率要稍微低于其他机器学习模型，但是，从正负样本的综合预测效果来看，*XGBoost* 的表现还是明显优于其他机器学习算法。因此，我们选择*XGBoost* 作为我们的预测模型。

表格4-11是本文方法对代码修改周期的长短的预测结果。在研究问题2中，我们同样对比了多种机器学习模型，以及通过SMOTE算法来一定程度样本不平衡造成的影响。观察实验结果可以发现，*XGBoost* 的综合准确在两个数据集上分别达到66.41% 和60.42%，高于其他机器学习模型的预测效果。同时，我们发现，所有机器学习模型的综合准确率分布为50.72%至66.42%，表明了我们方法在代码修改周期预测上的有效性。在Eclipse数据集上，*XGBoost*对于不同时间周期的F_1值分别达到61.38%，72.68%，44.19%。而在OpenDaylight上的效果则有一些下降，我们认为原因是OpenDaylight数据集中样本低于Eclipse。

另外，我们发现*XGBoost*在修改周期“较长”的样本上，召回率要略低于一些机器学习算法，但从综合评价指标F_1值可以发现，*XGBoost* 在该研究问题中是效果最好且稳定的模型。

为了研究三种特征对模型预测效果的影响，我们在研究问题1上进行对比实验。具体地，我们对三种特征进行两两组合，分别测试每对特征训练出来的模型的预测效果。例如，若审核元特征与代码耦合特征的组合训练出来的模型预测效果最优，则说明审核元特征与代码耦合特征对预测结果的重要性要高于代

表 4-12 不同特征组合对比的实验结果

项目名	特征	F1		准确率
		正样例	负样例	
Eclipse	审核元特征&代码耦合特征	48.31	83.77	75.30
	审核元特征&代码修改特征	51.04	85.17	77.24
	代码耦合特征&代码修改特征	54.78	85.08	77.56
	所有特征	62.10	86.16	79.72
OpenDayLight	审核元特征&代码耦合特征	42.86	83.67	74.60
	审核元特征&代码修改特征	35.90	85.38	76.19
	代码耦合特征&代码修改特征	44.09	85.88	77.46
	所有特征	50.58	87.49	80.03

码修改特征。在实验中，我们对比了不同特征组合下模型预测效果的F_1值和准确率，实验结果如表4-12所示。

观察表4-12中数据可以发现，任意两种特征组合的预测效果都低于使用三种特征时的预测效果，这说明三种对模型预测都起着重要作用。同时也证明了本文提取的三种可判别特征类型可用于代码修改周期的预测。使用审核元特征与代码耦合特征的组合训练得到的模型在F_1值和准确率上都低于其他两组特征组合，意味着代码修改特征在修改周期预测中的重要性高于另外两种特征。另一方面，代码耦合特征与代码修改特征训练的模型取得了最高的F_1 值和准确率，这表明审核元特征在周期预测中的作用低于另外两种特征。

4.3 代码修改影响分析及修改周期预测工具的具体实现

本节详细描述代码修改影响分析及修改周期预测工具的实现，图4-2为该工具的系统功能结构图，系统中影响分析模块与修改周期预测模块分别对应第二章与第三章研究内容，基础服务模块主要提供用户的管理功能及数据的管理功能。下面我们分别介绍几个功能模块的实现。

4.3.1 代码修改影响分析模块设计

代码修改影响分析模块在历史提交库中检索与当前修改工作最相似的提交，利用历史提交中关键类的修改模式，对当前修改类进行影响集的推荐。该模块包含一下几个功能：（1）关键类识别功能；（2）代码修改提取功能；（3）相似

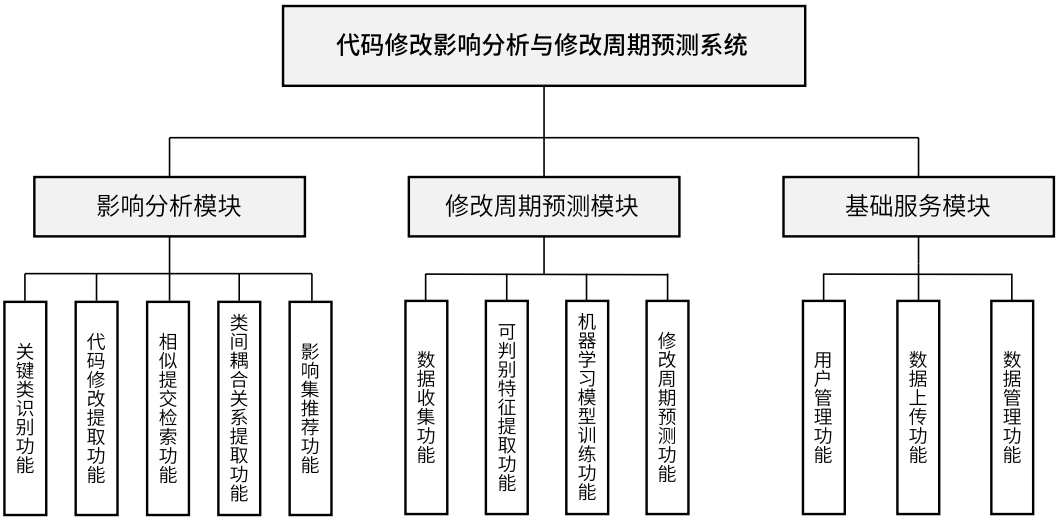


图 4-2 系统功能结构图

提交检索功能；（4）类间耦合关系提取功能；（5）影响集推荐功能。用户上传涉及修改的代码和修改目标后，系统通过文本预处理对两份数据进行处理，得到与历史提交库中相同的文本形式；通过词向量模型，对比用户输入数据与提交库每个提交中关键类的相似度，得到最相似的提交列表；用户选择初始影响分析方法，得到初始影响集，系统计算相似提交中关键类与非关键类的耦合关系以及修改类与初始影响集中类的耦合关系，再基于两部分耦合关系中相似的部分，对初始影响集进行优化排序，得到最终影响集。

表4-13中为代码修改影响分析模块中主要的功能类。*ClassChange*的功能是提取提交前后两个版本的代码中同个类的修改信息，而*VersionChange*则是提取提交两个版本的总修改信息。*Word2VecModel*用于封装有提交库中所有提交数据训练得到的词向量模型。关键类模型则封装在*CoreClassModel*中。*InitImpactSecGenerator*用于提供多种传统影响分析方法，生成用户修改的初始影响集。*CouplingRelations*的功能是提取类与类之间的结构耦合关系。*ImpactSetOptimizer*实现了对初始影响集的优化调整，并返回影响分析结果给用户。

下面简要阐述代码修改影响分析模块中各个子功能的实现：

（1）关键类识别功能

在本文影响分析方法中，关键类识别功能作为我们系统的核心模块之一。关键类识别功能是为了识别提交中核心修改的类。在相似提交检索及基于结构耦合相似性的影响集调整等两个步骤中，我们都是将提交中的关键类作为修改

表 4-13　代码修改影响分析模块主要类及对应功能

类名称	类功能描述
ClassChange	提取提交两个版本中同个类的修改信息
VersionChange	提取提交两个版本的修改信息
Word2VecModel	封装词向量模型
CoreClassModel	封装关键类识别模型
InitImpactSetGenerator	封装多种传统影响分析方法，生成初始影响集
CouplingRelations	提取类间结构耦合关系
ImpactSetOptimizer	初始影响集排序的调整，返回最终影响集

类的等价类。相似提交检索中，衡量指标是用户上传的代码修改片段与关键类的代码修改片段之间的相似度以及修改目标与注释文本的相似度。在对初始影响集的调整中，我们利用关键类的修改模式，对影响分析结果进行优化重排序。关键类识别功能如图4-3，用户上传提交数据后，系统给出提交中每个类为关键类的概率。

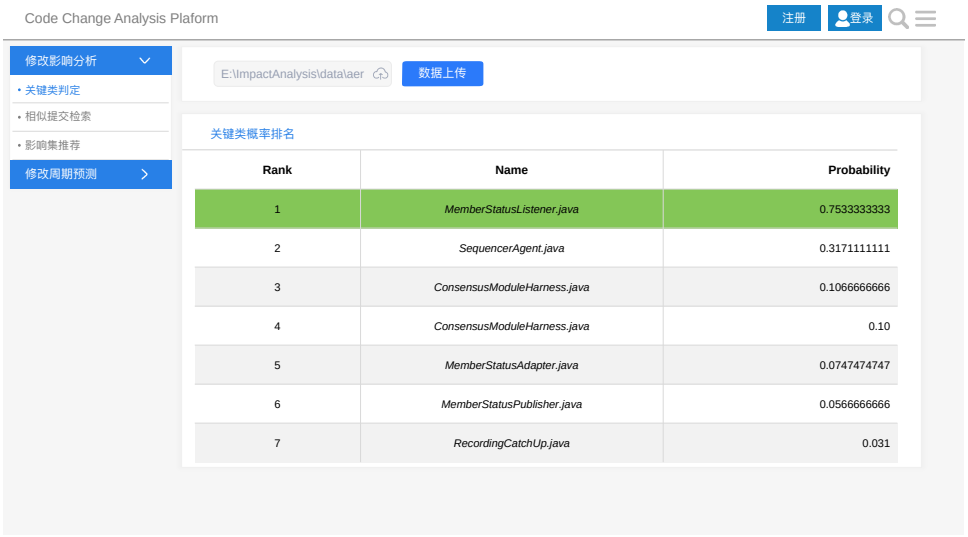


图 4-3　关键类识别功能示例

由于关键类判别模型已离线训练，系统中可以直接使用关键类识别模型。同时，由于提交数据库庞大，关键类的识别可离线完成。在用户使用阶段，可直接读取每个提交中的关键类。

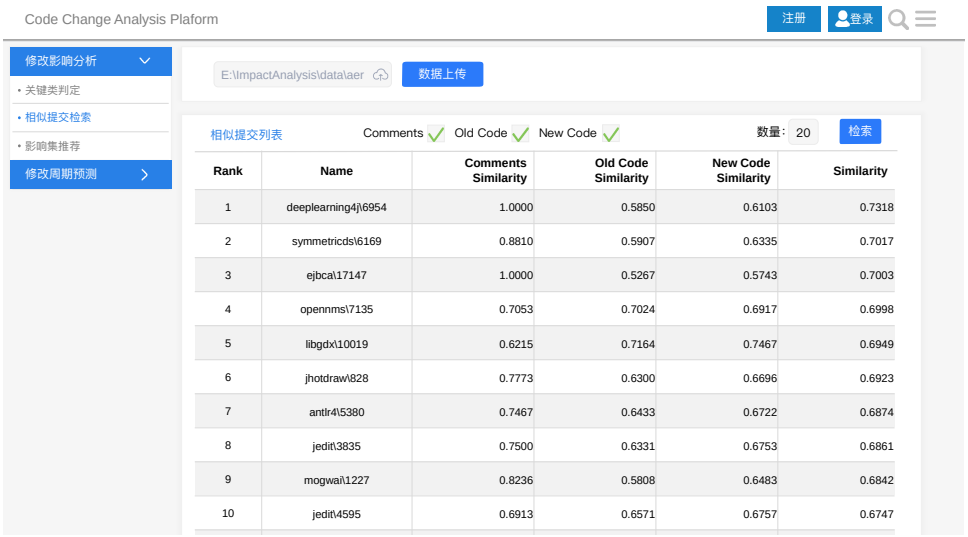
（2）代码修改提取功能

代码修改提取功能作为相似提交检索功能的基础模块，其主要功能是对用户上传的文件进行修改片段提取及不同版本代码对比。修改代码片段包括了类

层次、方放层次和语句层次等。同时，若影响分析中引入修改后的代码片段，代码修改提取功能中还实现了新旧代码的对比。

（3）相似提交检索功能

相似提交检索功能的核心利用提交库中的代码数据以及提交注释文本数据训练词向量模型，再基于词向量模型计算用户上传数据与提交库中提交的相似度，得到相似提交列表。相似提交检索功能如图4-4所示，用户可根据需要，选择衡量提交相似度的因素，包括注释相似度、修改前代码相似度和修改后代码相似度；还可以选择需要的相似提交数量。另外，在相似提交列表中，系统根据相似度从大到低展示相似提交，展示的数据包括了，注释文本相似度，修改前代码相似度，修改后代码相似度（若影响分析中引入修改后代码片段）以及综合相似度。另外，用户可以点击相似提交查看该提交的信息，包括注释信息，包含类的数量。



Rank	Name	Comments Similarity	Old Code Similarity	New Code Similarity	Similarity
1	deeplearning4j\6954	1.0000	0.5850	0.6103	0.7318
2	symmetricds\6169	0.8810	0.5907	0.6335	0.7017
3	ejbca\17147	1.0000	0.5267	0.5743	0.7003
4	opennms\7135	0.7053	0.7024	0.6917	0.6998
5	libgdx\10019	0.6215	0.7164	0.7467	0.6949
6	jhotdraw\828	0.7773	0.6300	0.6696	0.6923
7	antlr4\5380	0.7467	0.6433	0.6722	0.6874
8	jedit\3835	0.7500	0.6331	0.6753	0.6861
9	mogwai\1227	0.8236	0.5808	0.6483	0.6842
10	jedit\4595	0.6913	0.6571	0.6757	0.6747

图 4-4　相似提交检索功能示例

（4）类间耦合关系提取功能

本文的影响分析中初始影响集的优化重排序的基础是结构耦合关系的相似性。类间耦合关系的提取主要分为两部分，一是修改类与初始影响集中各个类的结构耦合关系，另一部分是相似提交中关键类与非关键类的耦合关系。通过该功能模块，用户可以选择查看不同类间的结构耦合关系。

（5）影响集推荐功能

影响集推荐功能如图4-5所示，影响集推荐功能模块用于展示影响分析结果，

包括初始影响分析结果和最终影响分析结果。最终影响分析结果中，绿色底纹的类是相对与初始影响集，排名上升的类。

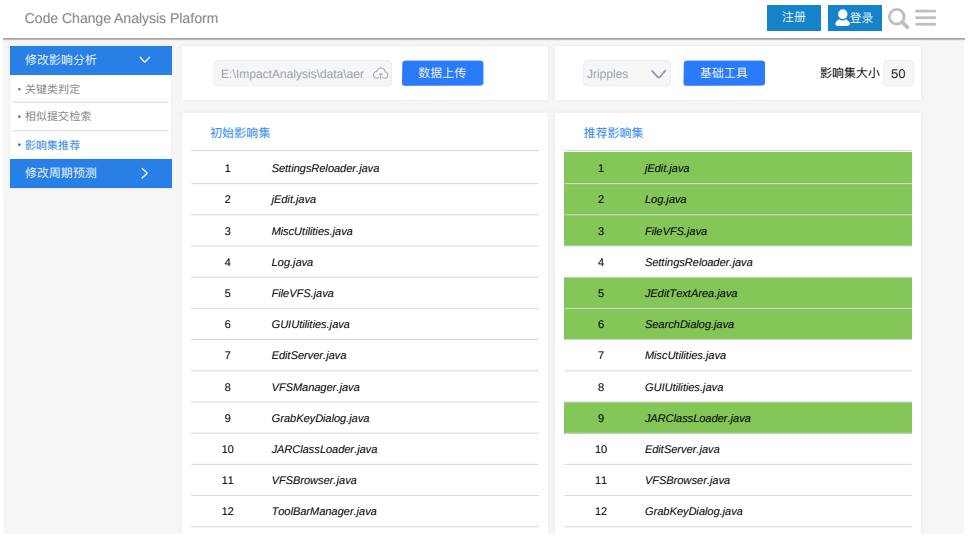


图 4-5　影响集推荐功能示例

4.3.2　代码修改周期预测模块实现

代码修改周期预测模块包括以下几个功能：（1）数据收集功能；（2）可判别特征提取功能；（3）机器学习模型训练功能；（4）修改周期预测功能。该模块主要实现对用户上传的数据，进行可判别特征的提取；提供机器学习模型的训练，以及修改周期预测等功能。用户上传代码修改数据及审核信息（另外，用户可上传项目历史修改数据集，也可以接入项目的代码审核平台，若用户未上传该部分数据，则系统默认使用已有数据集训练预测模型）。系统对用户上传数据进行一些文本预处理，然后提取代码审核元特征、代码耦合特征、代码修改特征。在预测模型训练部分，系统提供了多种机器学习模型供用户选择；同时，用户也可以直接使用已训练好的模型，对上传数据进行修改周期预测。下面我们简要介绍各功能模块。

表4-14为该代码修改周期预测模块主要类及对应功能。*DataSet*用于提供已有代码修改数据集，并接收用户上传数据集，其中，系统提供数据集包括Eclipse和OpenDaylight。*ReviewMetaFeatures*、*CodeCouplingFeatures*以及*CodeModifyingFeatures*功能分别是提取数据样本的代码审核元特征、代码耦合特征和代码修改特征。*FeatureProcessing*实现一些对数据样本的特征处理方法。*Model*提供了多种机器学习模型，用于实现模型训练。

表 4-14　代码修改周期预测模块主要类及对应功能

类名称	类功能描述
DataSet	封装已有代码修改及审核数据集，可替换成用户上传数据集
ReviewMetaFeatures	提取代码审核元特征
CodeCouplingFeatures	提取代码耦合特征
CodeModifyingFeatures	提取代码修改特征
FeatureProcessing	封装一些特征工程的处理方法
Model	封装多种机器学习模型

下面简要阐述代码修改周期预测分析模块中各个子功能的实现：

（1）数据收集功能

数据收集模块主要用于收集用户的上传的代码修改数据。系统中已有两个开源项目Eclipse和OpenDaylight的数据集用于训练机器学习模型，同时，用户也可以上传本项目的历史修改数据集。另外，如果修改涉及项目的使用了Gerrit审核平台，用户可以通过提供Gerrit项目地址的方式，提供项目代码修改的历史数据信息。数据收集功能如图4-6。

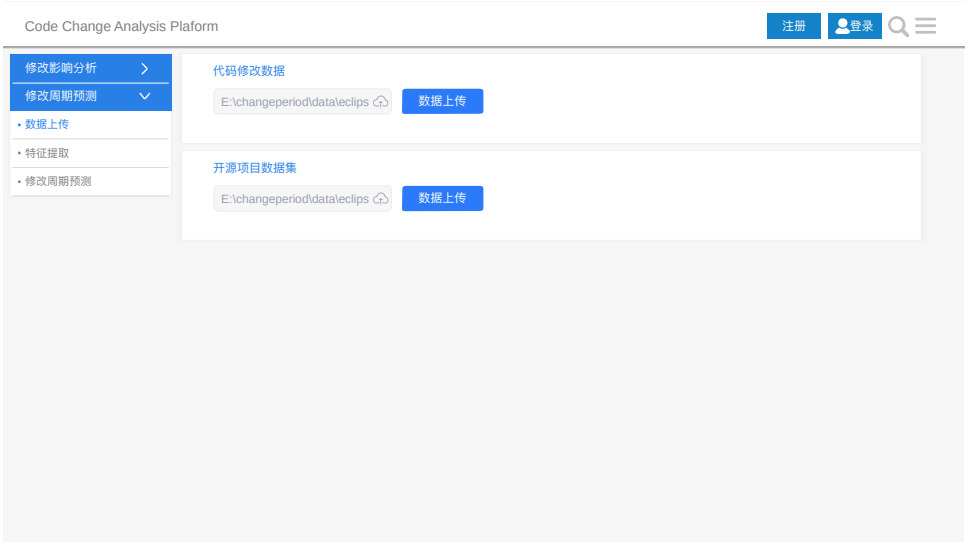


图 4-6　数据收集功能示例

（2）可判别特征提取功能

可判别特征提取是系统实现周期预测的核心功能，其主要目标是从用户上传的数据中提取本文3.2节中描述的40维特征，另外如果用户上传了自己的数据集，该模块还需对数据集中的代码和审核数据进行处理。在该模块中，用户可

以查看审核元特征、代码耦合特征和代码修改特征下各维特征的详细数据。可判别特征提取功能示例图如图4-7所示。

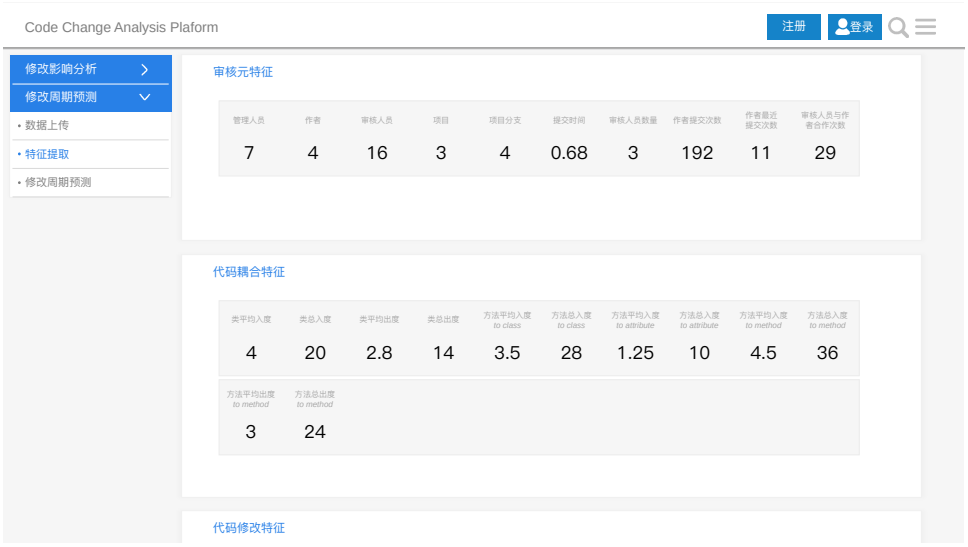


图 4-7　可判别特征提取功能示例

（3）机器学习模型训练功能

机器学习模型训练模块中实现了多种机器学习模型，其中默认使用通过已有数据训练的*XGBoost*模型。另外，用户也可选择训练其他常用的机器学习模型，如*LR*、*SVM*、*RF*和*ANN*等。在数据集使用方面，用户可以选择已有数据，或者自行上传数据集。

（5）修改周期预测功能

修改周期预测功能如图4-8所示，修改周期预测功能主要是向用户展示模型预测结果。其中，较短表示修改周期在一天内完成，一般表示修改周期在一周内完成，较长则表示修改周期超过一周。

4.3.3　基础服务模块的实现

基础服务模块的主要功能是用户管理、数据管理等。其中，用户管理包括提供用户注册和登录功能；数据管理功能主要完成用户数据的管理及用户操作日志的保存等。

4.4　本章小结

在本章中，我们详细介绍了修改影响分析方法和修改周期预测方法的实验

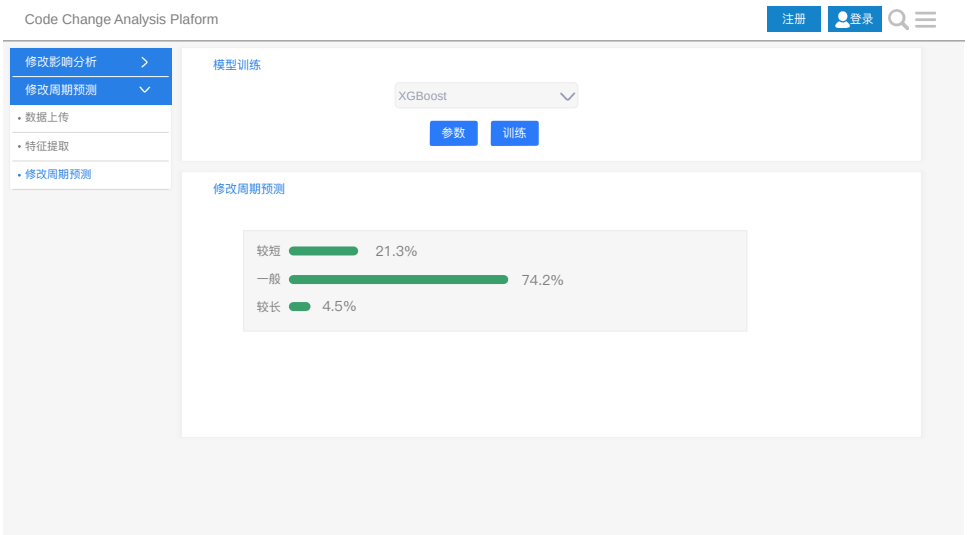


图 4-8　修改周期预测功能示例

设计及实验评估，并简要介绍了修改影响分析及周期预测工具的设计。在基于历史修改模式的影响分析方法实验中，我们从多个开源项目的数据集中评估了方法的有效性，并验证了是否使用关键类及耦合关系相似度高低对影响分析效果的影响。实验结果表明，我们的方法对传统影响分析结果具有很好的提升效果，在所有开源项目数据集上影响集前5和前10的召回率、精确率平均提升超过15%。在基于可判别特征的代码修改周期预测方法的实验中，我们通过对比了不同机器学习模型的效果以及不同特征对预测效果的影响。实验结果显示，不同机器学习模型的预测结果都有较好的准确性，验证了我们提取的可判别特征在周期预测中的有效性，并且结果表明XGBoost的预测效果优于其他机器学习模型。在工具设计中，我们通过两个功能模块实现了影响分析和代码修改周期预测功能，其中工具中的代码修改影响分析模块和代码修改周期预测模块分别对应第二章和第三章的研究内容。

第 5 章 总结与展望

代码修改作为软件维护工作的重要部分，是软件生命中期中关键的一环。如何辅助开发人员更高质量、更快速的完成代码修改成为软件维护过程中备受关注的问题。本文的研究目的是为了辅助开发人员在软件维护中能更高效率的完成代码修改工作。我们分别从修改影响分析及修改周期预测两个方面深入研究，提出了一些辅助开发人员完成代码修改工作的方法。本章将总结前文的研究工作，并对未来的研究工作展望。

5.1 工作总结

软件项目的发展周期中，需要不断应对新的问题及需求变化，使得代码修改工作成为软件维护中开发人员最频繁面对的工作。同时，软件项目的高速发展，互联网积累了更多数据资源，如软件维护过程中代码修改数据，代码审核数据等，为软件资源挖掘、影响分析等研究工作提供了更多的研究角度。如何高效的利用这些代码维护过程中数据辅助开发人员提高开发和维护的效率是非常有意义的工作。因此，本文立足于软件维护过程中数据，围绕如何提高开发人员的代码修改效率展开研究。首先，本文提出了一种基于历史修改模式的影响分析辅助方法，帮助开发人员在代码修改过程中，快速定位修改影响范围，保证修改的一致性。其次，本文还提出一种基于可判别特征的代码修改周期预测方法，辅助开发人员提前预估代码修改工作完成周期的长短，引导开发人员做出合适的调整。主要完成工作如下：

本文分析和总结了国内外关于修改影响分析、修改周期预测相关的研究，对两个研究点中具有代表性的方法进行了对比。文中还对比分析了现在方法和工具的不足，并针对修改影响分析、修改周期预测提出了新的研究方法。

软件实体的整体性导致软件中代码发生变动时，会影响其他部分的代码，这些问题增加了开发人员完成代码修改工作的难度。本文针对代码修改导致的软件实体不一致性，提出了一种基于历史修改模式的影响分析辅助方法。我们的方法在传统影响分析方法的基础上，借助历史修改信息对影响集进行调整，

从而提高影响分析的效果。具体地，我们从版本控制系统中筛选了182个开源项目，并从中收集94778条历史修改提交数据，用于构建历史提交库。然后，通过修改代码及修改目标（注释文本）的相似度在提交库中检索最相似的提交，在相似度匹配中，我们对比的是当前修改类与提交中关键类的相似度。最后，我们基于结构耦合关系的相似性，将关键类与提交中其他类的修改传播模式，映射到当前修改类与初始影响集中的类上，从而调整初始影响集的排序，得到最终影响集。实验结果表明，我们的方法对传统影响分析结果具有很好的提升效果，在多个项目上影响集前5和前10的召回率、精确率平均提升超过15%。

为了引导开发人员在代码修改工作中作出更准确的安排，本文提出一种基于可判别特征的修改周期预测方法。我们调研了开源项目的一些修改提交数据后，把修改周期按时间长短划分为三个层次，从而将修改周期预测定义为分类问题。该方法的关键是从提交数据中提取多维度可判别特征，包括：代码元特征、代码耦合特征及代码修改特征。从多维度衡量代码修改阶段及代码审核阶段的诸多影响因素。然后利用机器学习算法预测修改周期的长短。实验中，我们通过多种机器学习算法，验证了我们方法的可行性。

最后，我们围绕修改影响分析和修改周期预测两个方法，进行工具设计。工具的目标是在工作实践中有效地辅助软件开发和维护人员更加高效率的完成代码修改及代码维护工作。

5.2 研究展望

本文围绕修改影响分析和修改周期预测深入研究，提出了两种新的研究方法，经过实验验证，本文方法是实际有效的。然而，该方法仍然存在一些不足，包括：

1）在相似提交检索中关键的一步是识别提交中的关键类，而本文关键类识别方法仍存在一定的误差，从而不可避免地影响相似提交匹配。关键类判定模型的训练，需要提取更多有效特征，提高其识别准确率。

2）本文两个研究点都是基于软件修改和维护过程中数据，其中主要是修改提交数据。提交数据的数量和质量对实验结果会产生较大影响，在影响分析中，提交库覆盖了182个项目共94778条提交，而修改周期预测的数据集包括了2个项

目共15861条数据。在后续工作中，我们需要增加更多数据。

3）在修改周期预测方面，目前从三个维度共提取了40维特征，在以后研究中，需要总结提取更多衡量代码修改的特征。其次，当前方法仅采用了常规的机器学习算法，在算法上存在改进空间。

4）在修改影响分析和修改周期预测的工具设计方面，为了更有方便开发人员使用，在后续工作中，可以将两方面工具集成到Eclipse插件中。在开发人员在实际开发工作中，能快速分析代码修改。

参考文献

[1] Szabo C. Novice code understanding strategies during a software maintenance assignment [C]. Proceedings of the 37th International Conference on Software Engineering-Volume 2, 2015: 276∼284.

[2] Rovegård P, Angelis L, Wohlin C. An empirical study on views of importance of change impact analysis issues [J]. IEEE Transactions on Software Engineering, 2008 (4): 516–530.

[3] Kim S, Ernst M D. Prioritizing warning categories by analyzing software history [C]. Proceedings of the Fourth International Workshop on Mining Software Repositories, 2007: 27.

[4] Bradley A W, Murphy G C. Supporting software history exploration [C]. Proceedings of the 8th working conference on mining software repositories, 2011: 193–202.

[5] Wittenhagen M, Cherek C, Borchers J. Chronicler: Interactive exploration of source code history [C]. Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, 2016: 3522–3532.

[6] D’ Ambros M, Gall H, Lanza M, *et al*. Analysing software repositories to understand software evolution [M] // D’ Ambros M, Gall H, Lanza M, *et al*. Software evolution. Springer, 2008: 2008: 37–67.

[7] Thomas S W, Adams B, Hassan A E, *et al*. Studying software evolution using topic models [J]. Science of Computer Programming, 2014, 80: 457–479.

[8] Gousios G, Kalliamvakou E, Spinellis D. Measuring developer contribution from software repository data [C]. Proceedings of the 2008 international working conference on Mining software repositories, 2008: 129–132.

[9] Dabbish L, Stuart C, Tsay J, *et al*. Social coding in GitHub: transparency and collaboration in an open software repository [C]. Proceedings of the ACM 2012 conference on computer supported cooperative work, 2012: 1277–1286.

[10] Canfora G, Cerulo L. Impact analysis by mining software and change request repositories [C]. Software Metrics, 2005. 11th IEEE International Symposium, 2005: 9–pp.

[11] Kagdi H, Gethers M, Poshyvanyk D, *et al*. Blending conceptual and evolutionary couplings to support change impact analysis in source code [C]. Reverse Engineering (WCRE), 2010 17th Working Conference on, 2010: 119–128.

[12] 孙小兵, 李斌, 陈颖, *et al*. 软件修改影响分析研究与进展 [J]. 电子学报, 2014, 42 (12): 2467–2476.

[13] Sun X, Li B, Leung H, *et al*. Msr4sm: Using topic models to effectively mining software repositories for software maintenance tasks [J]. Information and Software Technology, 2015, 66: 1–12.

[14] Wong E, Liu T, Tan L. Clocom: Mining existing source code for automatic comment generation [C]. Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on, 2015: 380–389.

[15] Huang Y, Zheng Q, Chen X, *et al*. Mining version control system for automatically generating commit comment [C]. Proceedings of the 11th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, 2017: 414–423.

[16] Okutan A, Yıldız O T. Software defect prediction using Bayesian networks [J]. Empirical Software Engineering, 2014, 19 (1): 154–181.

[17] Moeyersoms J, de Fortuny E J, Dejaeger K, *et al*. Comprehensible software fault and effort

prediction: A data mining approach [J]. Journal of Systems and Software, 2015, 100: 80–90.

[18] Bouktif S, Gueheneuc Y-G, Antoniol G. Extracting change-patterns from cvs repositories [C]. Reverse Engineering, 2006. WCRE'06. 13th Working Conference on, 2006: 221–230.

[19] Bohner S A, Arnold R. Software change impact analysis for design evolution [C]. Proceedings of 8th International Conference on Maintenance and Re-engineering, 1991: 292–301.

[20] 刘华虓, 金英, 马鹏飞. 一种需求变更影响分析方法 [J]. 计算机研究与发展, 2013, 50 (8): 1769–1777.

[21] Park S, Bae D-H. An approach to analyzing the software process change impact using process slicing and simulation [J]. Journal of Systems and Software, 2011, 84 (4): 528–543.

[22] Kagdi H, Gethers M, Poshyvanyk D. Integrating conceptual and logical couplings for change impact analysis in software [J]. Empirical Software Engineering, 2013, 18 (5): 933–969.

[23] Abdeen H, Bali K, Sahraoui H, *et al*. Learning dependency-based change impact predictors using independent change histories [J]. Information and Software Technology, 2015, 67: 220–235.

[24] Apiwattanapong T, Orso A, Harrold M J. Efficient and precise dynamic impact analysis using execute-after sequences [C]. Proceedings of the 27th international conference on Software engineering, 2005: 432–441.

[25] Huang L, Song Y-T. Precise dynamic impact analysis with dependency analysis for object-oriented programs [C]. Software Engineering Research, Management & Applications, 2007. SERA 2007. 5th ACIS International Conference on, 2007: 374–384.

[26] Lin C-Y, Wu T-Y, Huang C-C. Nonlinear dynamic impact analysis for installing a dry storage canister into a vertical concrete cask [J]. International Journal of Pressure Vessels and Piping, 2015, 131: 22–35.

[27] Cai H, Thain D. Distia: A cost-effective dynamic impact analysis for distributed programs [C]. Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, 2016: 344–355.

[28] Breech B, Tegtmeyer M, Pollock L. Integrating influence mechanisms into impact analysis for increased precision [C]. Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on, 2006: 55–65.

[29] Law J, Rothermel G. Whole program path-based dynamic impact analysis [C]. Proceedings of the 25th International Conference on Software Engineering, 2003: 308–318.

[30] Cai H, Santelices R. Diver: Precise dynamic impact analysis using dependence-based trace pruning [C]. Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, 2014: 343–348.

[31] Briand L, Labiche Y, Soccar G. Automating impact analysis and regression test selection based on UML designs [C]. icsm, 2002: 0252.

[32] Díaz J, Pérez J, Garbajosa J, *et al*. Change impact analysis in product-line architectures [C]. European Conference on Software Architecture, 2011: 114–129.

[33] Poshyvanyk D, Marcus A, Ferenc R, *et al*. Using information retrieval based coupling measures for impact analysis [J]. Empirical software engineering, 2009, 14 (1): 5–32.

[34] Beszedes A, Gergely T, Farago S, *et al*. The dynamic function coupling metric and its use in software evolution [C]. Software Maintenance and Reengineering, 2007. CSMR'07. 11th European Conference on, 2007: 103–112.

[35] Rolfsnes T, Di Alesio S, Behjati R, *et al*. Generalizing the analysis of evolutionary coupling for software change impact analysis [C]. 2016 IEEE 23rd International Conference on Software

Analysis, Evolution, and Reengineering (SANER), 2016: 201–212.

[36] Hattori L, dos Santos Jr G, Cardoso F, *et al*. Mining software repositories for software change impact analysis: a case study [C]. Proceedings of the 23rd Brazilian symposium on Databases, 2008: 210–223.

[37] Li B, Sun X, Leung H, *et al*. A survey of code-based change impact analysis techniques [J]. Software Testing, Verification and Reliability, 2013, 23 (8): 613–646.

[38] Thongtanunam P, Tantithamthavorn C, Kula R G, *et al*. Who should review my code? A file location-based code-reviewer recommendation approach for modern code review [C]. Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on, 2015: 141–150.

[39] Kononenko O, Baysal O, Guerrouj L, *et al*. Investigating code review quality: Do people and participation matter? [C]. Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on, 2015: 111–120.

[40] Baysal O, Kononenko O, Holmes R, *et al*. Investigating technical and non-technical factors influencing modern code review [J]. Empirical Software Engineering, 2016, 21 (3): 932–959.

[41] Ying A T, Murphy G C, Ng R, *et al*. Predicting source code changes by mining change history [J]. IEEE transactions on Software Engineering, 2004, 30 (9): 574–586.

[42] Zimmermann T, Zeller A, Weissgerber P, *et al*. Mining version histories to guide software changes [J]. IEEE Transactions on Software Engineering, 2005, 31 (6): 429–445.

[43] Ajienka N, Capiluppi A, Counsell S. An empirical study on the interplay between semantic coupling and co-change of software classes [J]. Empirical Software Engineering, 2018, 23 (3): 1791–1825.

[44] Jaafar F, Guéhéneuc Y-G, Hamel S, *et al*. Detecting asynchrony and dephase change patterns by mining software repositories [J]. Journal of Software: Evolution and Process, 2014, 26 (1): 77–106.

[45] Vaucher S, Sahraoui H, Vaucher J. Discovering new change patterns in object-oriented systems [C]. Reverse Engineering, 2008. WCRE'08. 15th Working Conference on, 2008: 37–41.

[46] Fluri B, Giger E, Gall H C. Discovering patterns of change types [C]. Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, 2008: 463–466.

[47] Michail A. Data mining library reuse patterns in user-selected applications [C]. ase, 1999: 24.

[48] Lin Y, Dig D. Check-then-act misuse of java concurrent collections [C]. 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, 2013: 164–173.

[49] Holmes R, Walker R J, Murphy G C. Approximate structural context matching: An approach to recommend relevant examples [J]. IEEE Transactions on Software Engineering, 2006 (12): 952–970.

[50] Bruch M, Monperrus M, Mezini M. Learning from examples to improve code completion systems [C]. Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, 2009: 213–222.

[51] Zaidman A, Van Rompaey B, Demeyer S, *et al*. Mining software repositories to study co-evolution of production & test code [C]. Software Testing, Verification, and Validation, 2008 1st International Conference on, 2008: 220–229.

[52] Murali V, Chaudhuri S, Jermaine C. Bayesian specification learning for finding API usage errors [C]. Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering,

2017: 151–162.

[53] Mills C. Automating traceability link recovery through classification [C]. Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, 2017: 1068–1070.

[54] Rath M, Rendall J, Guo J L, *et al*. Traceability in the wild: automatically augmenting incomplete trace links [C]. Proceedings of the 40th International Conference on Software Engineering, 2018: 834–845.

[55] Karim S, Warnars H L H S, Gaol F L, *et al*. Software metrics for fault prediction using machine learning approaches: A literature review with PROMISE repository dataset [C]. Cybernetics and Computational Intelligence (CyberneticsCom), 2017 IEEE International Conference on, 2017: 19–23.

[56] Shimonaka K, Sumi S, Higo Y, *et al*. Identifying Auto-Generated Code by Using Machine Learning Techniques [C]. Empirical Software Engineering in Practice (IWESEP), 2016 7th International Workshop on, 2016: 18–23.

[57] Nguyen A T, Nguyen H A, Nguyen T T, *et al*. Statistical learning approach for mining API usage mappings for code migration [C]. Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, 2014: 457–468.

[58] Linares-Vásquez M, McMillan C, Poshyvanyk D, *et al*. On using machine learning to automatically classify software applications into domain categories [J]. Empirical Software Engineering, 2014, 19 (3): 582–618.

[59] Fluri B, Wuersch M, PInzger M, *et al*. Change distilling: Tree differencing for fine-grained source code change extraction [J]. IEEE Transactions on software engineering, 2007, 33 (11).

[60] Goldberg Y, Levy O. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method [J]. arXiv preprint arXiv:1402.3722, 2014.

[61] Mikolov T, Sutskever I, Chen K, *et al*. Distributed representations of words and phrases and their compositionality [C]. Advances in neural information processing systems, 2013: 3111–3119.

[62] Mikolov T, Chen K, Corrado G, *et al*. Efficient estimation of word representations in vector space [J]. arXiv preprint arXiv:1301.3781, 2013.

[63] 黄袁, 刘志勇, 陈湘萍, *et al*. 基于关键类判定的代码提交理解辅助方法 [J]. 软件学报, 2017, 28 (6): 1418–1434.

[64] Huang Y, Jia N, Chen X, *et al*. Salient-class location: help developers understand code change in code review [C]. Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2018: 770–774.

[65] Eyolfson J, Tan L, Lam P. Do time of day and developer experience affect commit bugginess? [C]. Proceedings of the 8th Working Conference on Mining Software Repositories, 2011: 153–162.

[66] Hassan A E, Holt R C. Predicting change propagation in software systems [C]. Software maintenance, 2004. proceedings. 20th ieee international conference on, 2004: 284–293.

致谢