**中山大学**

攻读硕士学位研究生

**硕士学位论文开题报告**

题目：数据驱动的修改影响分析与修改完成预测方法及应用

|  |  |
| --- | --- |
| 姓名： | 洪凯 |
| 学号： | 17214615 |
| 院系： | 数据科学与计算机学院 |
| 专业： | 软件工程 |
| 研究方向： | 软件工程 |
| 指导教师： | 陈湘萍助理研究员 |

中山大学研究生院

2018年 12月20日

目 录

[1 课题研究背景和意义 2](#_Toc532395375)

[2 国内外研究现状 3](#_Toc532395376)

[2.1修改影响分析 4](#_Toc532395377)

[2.2 代码修改完成周期预测 5](#_Toc532395378)

[2.3软件存储库挖掘 6](#_Toc532395379)

[3 主要研究内容 7](#_Toc532395380)

[3.1 基于历史修改模式的影响分析辅助方法 8](#_Toc532395381)

[3.2基于可判别特征的代码修改完成周期预测 9](#_Toc532395382)

[3.3 实验 9](#_Toc532395383)

[4 研究方法 10](#_Toc532395384)

[5 研究计划和时间安排 11](#_Toc532395385)

[6 参考文献 12](#_Toc532395386)

# 1 课题研究背景和意义

软件的可维护性和可修改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，从而影响软件的稳定性。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，代码修改任务的最后一步是代码审核，通常代码修改需要经过多次“修改—审核—再修改”的环节才能最终完成，审核轮次对代码修改的完成周期有直接的影响，提前对代码修改任务的完成周期进行预估，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。

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。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[3,4,5]。 在ICSE、ASE 和ICSM 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。软件维护和演化过程中存在大量软件工程过程中数据，特别是当前，开源项目广泛托管在Github等版本控制系统中、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，近年来，机器学习取得飞速的发展，在各个领域的应用，都起到了对研究分析的推动作用。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。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，但是主流的研究方法更加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。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）中找到相似的代码修改任务。相似的代码修改任务中的代码修改范围对于开发人员确认修改影响范围有直接的借鉴作用。开源项目在版本控制系统中的维护和演化，都是通过代码提交（Commit）的形式进行的。代码提交中的数据包括：（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，对后续维护工作起着重要作用[12]。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[13,14]，缺陷预测[15,16]，修改影响分析[10,11]等。其中，修改影响分析有助于开发人员在修改代码时，预估可能影响的其他代码实体，辅助开发人员更加高效的完成代码修改工作。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，将提交中的关键类等价为当前修改类，利用关键类的修改影响范围辅助分析当前修改的影响范围。

# 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，代码修改完成周期预测和软件存储库挖掘三个方面介绍国内外研究现状以及与本文研究相关的工作。

## 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[20,21]。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[22]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。

修改影响分析方法主要分为静态影响分析[23,24,25]和动态影响分析[26,27,28,29]。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[30]总结了代码修改中的传播机制，并基于这些传播机制建立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[31]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[32]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[33,34]，基于耦合关系的影响分析[35,36]，以及基于软件存储库挖掘的影响分析[37,38]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[39]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。

## 2.2 代码修改完成周期预测

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[40]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[40]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[40]类似，Oleksii等人[41]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[42]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WebKit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。本文使用机器学习算法从开源代码审查软件中提取审查信息，根据代码审查中各元素预测代码修改的完成周期。由于没有与代码审查周期预测直接相关的工作，接下来主要介绍机器学习在软件维护研究中的应用。

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[43]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[44]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[45]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[46]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[47]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[48]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[49]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。

## 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e1总是与代码实体e2同时进行修改，当开发人员再次对e1进行修改时，可预估e2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[21]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[50]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[51]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ROSE，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[52]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[53]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[54]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[54]类似，Fluri等人[55]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[56]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[57]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[58]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[59]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[60]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。

# 3 主要研究内容

本课题的主要研究内容包含两个方面，分别是：基于历史修改模式的影响分析辅助方法和基于可判别特征的代码修改完成周期预测方法。

## 3.1 基于历史修改模式的影响分析辅助方法

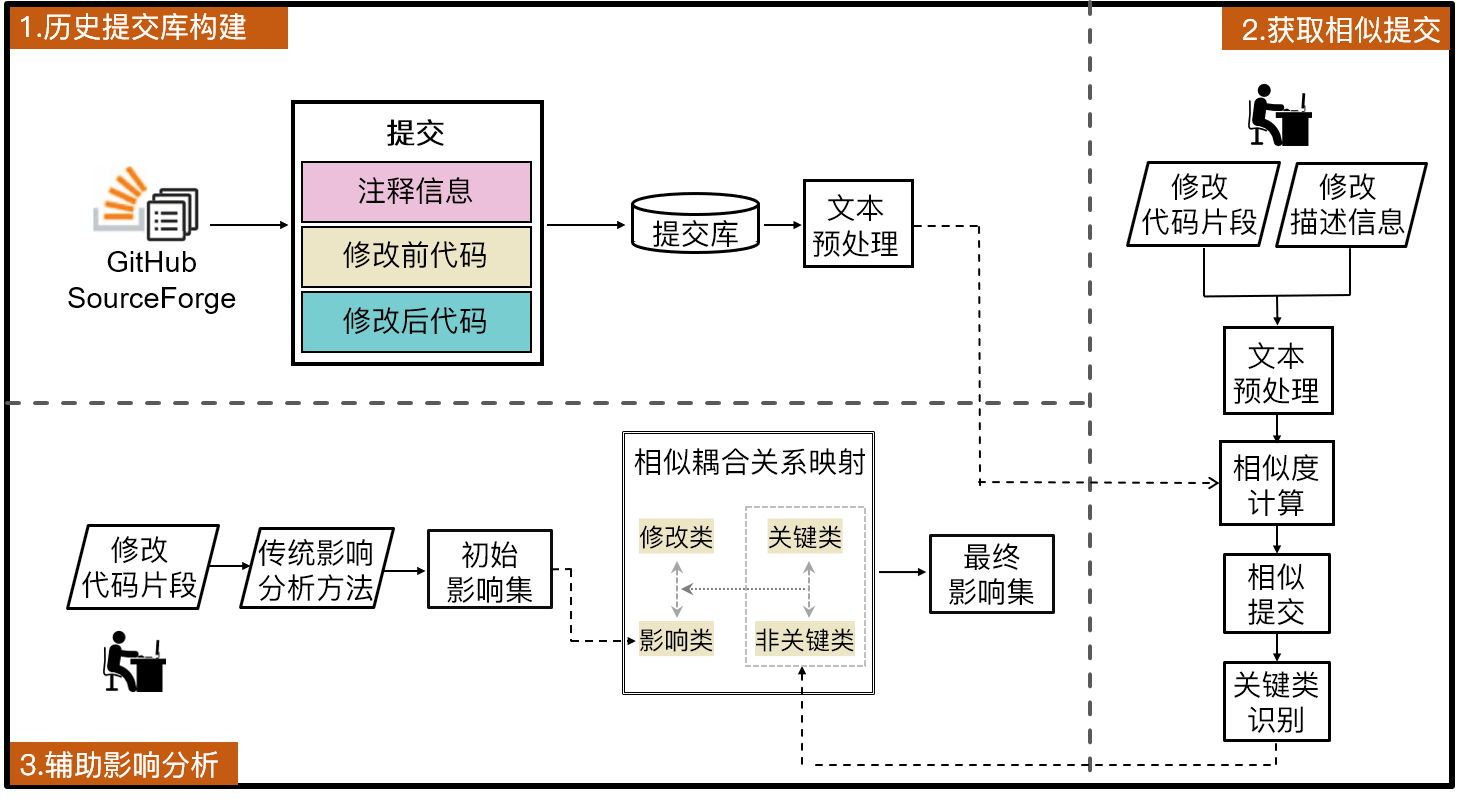
对于一个软件系统来说，经过多年的开发历史，系统中各个软件实体之间存在非常复杂关联关系，当开发人员需要对其中一个软件实体进行修改时，必然会影响到其他软件实体，且影响范围会随着实体间的关联关系不断传播。因此，修改影响分析成为软件修改工作中的关键一步。修改影响分析的目的是评估一项修改任务可能带来的风险以及受影响的范围和程度。软件修改的主要目的包括增加新功能、修复缺陷或者适应新的用户需求。在同一项目或不同项目的演化历史中往往存在着相似的修改需求，这些相似修改的修改模式对于当前的修改任务有辅助作用。传统的静态影响分析以及动态影响分析方法难以捕获软件项目中复杂的依赖关系，引入历史修改模式信息可以对传统影响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。

图3-1 基于历史修改模式的影响分析辅助方法总览

图3-1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其他类的耦合关系；利用耦合关系的相似度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。

## 3.2基于可判别特征的代码修改完成周期预测

一项软件代码修改工作通常包含多次“修改-审核-再修改”的过程，开发人员在修改完代码后需提交给审核人员，即一项代码修改工作完成的标志是通过代码审核人员的审查。代码审核不通过，则开发人员需对代码进行再次改进。因此，一项代码修改任务往往经过了多次的代码审核环节，代码审核的次数也能体现该任务周期的长短。审核次数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。本文以代码审核次数表示代码修改完成周期的长短，并从代码审核数据中提取可判别特征，训练机器学习模型，用于预测代码修改的审核次数。我们的方法从代码审核数据中提取代码审核信息、代码耦合信息以及代码修改信息作为可判别特征，再使用集成学习方法融合多个机器学习模型得到我们的分类模型。方法总览图如图2所示。

## 3.3 实验

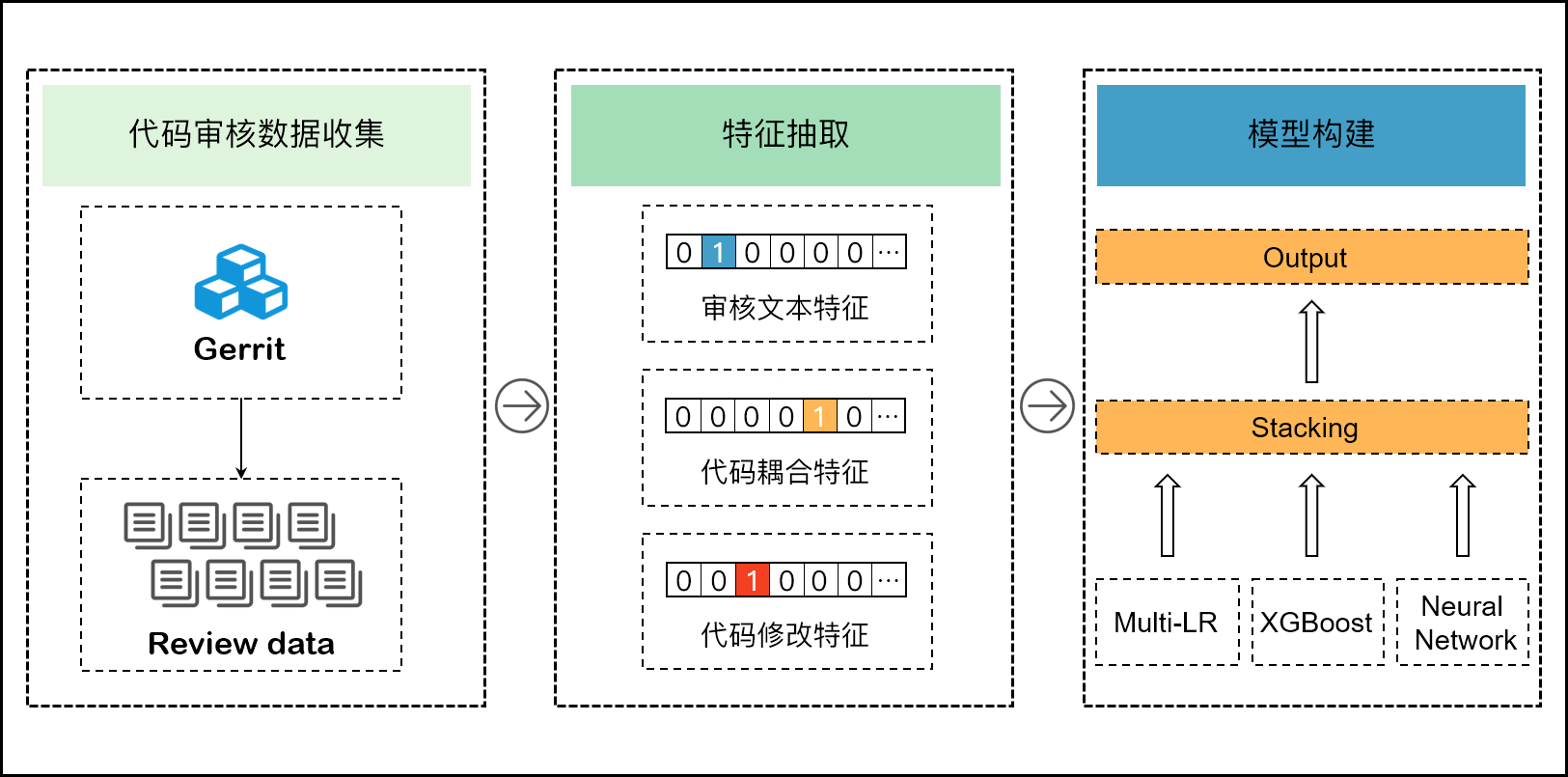
本课题为基于数据驱动修改影响分析与修改完成预测方法及应用，旨在提高开发人员在软件项目维护过程中效率和质量。为了验证方法的可行性和科学性，我们从Github、SourceForge源码库中获得开源软件的源代码及相关文档、从Gerrit代码审核工具中收集开源项目的代码审核信息，最后在Eclipse平台进行实验，并对涉及不同领域的开源软件做实例测试，验证结果的一般性。

图2 基于可判别特征的修改完成周期预测方法总览

# 4 研究方法

本课题定位于学术性质的研究，因此在研究方法上偏重于理论创新和实验方面，研究成果以学术论文为主，实验性演示系统为辅的形式。具体研究方式如下列所示：

1、关注并仔细阅读最新的文献，特别是在2013-2018年的FSE/ESEC、ICPC、ICSM、ICSE和ICPC等顶级及一流的软件工程会议和期刊上的最新研究成果。

2、在修改影响分析方面，本文通过代码修改内容和修改文本描述的相似性来检索与当前修改任务相似的历史提交，再利用代码类之间的耦合关系将历史提交的修改模式映射回当前修改。在修改完成周期预测方面，本文以代码审核次数来评估代码修改完成周期的时间长短，从代码审核数据中提取审核信息，代码修改信息以及代码耦合信息，结合机器学习模型，实现修改完成周期的预测。

3、在研究工作后期，形成用于检测代码注释质量以及生成注释的系统，进行质量分析和注释生成，反馈信息给软件开发维护者。

4、以目标为导向，本研究以产生有价值的学术成果为目标。

5、以科研合作的方式进行，尊重他人劳动成果，事实求是，力图创新。

# 5 研究计划和时间安排

本研究课题时间从2017年9月至2019年5月，历时约20个月，共分四个阶段进行，具体时间安排如下表所示。

本研究预期完成硕士毕业论文1篇。

表1 研究计划和时间安排表

| **时间** | **研究计划** |
| --- | --- |
| 2017.9-2017.12 | **资料搜集整理阶段：**   * 明确研究课题的方向、主要内容和需要解决的问题； * 通过阅读文献和搜集相关资源，清楚本研究课题的研究状况和特点； * 确定研究的重点和研究路线，形成基本的研究思路。 |
| 2018.01-2018.12 | **论文主要方法的实现阶段：**   * 有针对性地对现有方法进行实现； * 对比实验结果，整理实验数据； * 撰写学术论文和毕业论文的部分内容，并根据2018年的国际学术会议投稿时间进行论文投稿。 |
| 2019.01-2018.03 | **论文撰写和定稿阶段：**   * 完成论文的总体内容； * 对论文进行数次修改，包括论文格式、论文内容、参考文献可靠性审查和其他补充材料。 |
| 2019.03-2019.05 | **论文修改和完善阶段：**   * 进一步修改和完善论文； * 完成答辩用的演示材料，并进行答辩。 |

注：具体答辩事项的时间安排以学院公布为准。

# 参考文献

1. Per Rovegard, Lefteris Angelis, and Claes Wohlin. An empirical study on views of importance of change impact analysis issues. IEEE Transactions on Software Engineering, (4):516–530, 2008.
2. Szabo C. Novice code understanding strategies during a software maintenance assignment[C]//Proceedings of the 37th International Conference on Software Engineering-Volume 2. IEEE Press, 2015: 276-284.
3. Sunghun Kim and Michael D Ernst. Prioritizing warning categories by analyzing software history. In Proceedings of the Fourth International Workshop on Mining Software Repositories, page 27. IEEE Computer Society, 2007.
4. Alexander WJ Bradley and Gail C Murphy. Supporting software history exploration. In Proceedings of the 8th working conference on mining software repositories, pages 193–202. ACM, 2011.
5. Moritz Wittenhagen, Christian Cherek, and Jan Borchers. Chronicler: Interactive exploration of source code history. In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, pages 3522–3532. ACM, 2016.
6. Marco D’Ambros, Harald Gall, Michele Lanza, and Martin Pinzger. Analysing software repositories to understand software evolution. In Software evolution, pages 37–67. Springer, 2008.
7. Stephen W Thomas, Bram Adams, Ahmed E Hassan, and Dorothea Blostein. Studying software evolution using topic models. Science of Computer Programming, 80:457–479, 2014.
8. Georgios Gousios, Eirini Kalliamvakou, and Diomidis Spinellis. Measuring developer contribution from software repository data. In Proceedings of the 2008 international working conference on Mining software repositories, pages 129–132.ACM, 2008.
9. Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in github: transparency and collaboration in an open software repository. In Proceedings of the ACM 2012 conference on computer supported cooperative work,pages 1277–1286.ACM, 2012.
10. Gerardo Canfora and Luigi Cerulo. Impact analysis by mining software and change request repositories. In Software Metrics, 2005. 11th IEEE International Symposium, pages 9–pp. IEEE, 2005.
11. Huzefa Kagdi, Malcom Gethers, Denys Poshyvanyk, and Michael L Collard.Blending conceptual and evolutionary couplings to support change impact analysis in source code. In Reverse Engineering (WCRE), 2010 17th Working Conference on, pages 119–128. IEEE, 2010.
12. Xiaobing Sun, Bixin Li, Hareton Leung, Bin Li, and Yun Li. Msr4sm: Using topic models to effectively mining software repositories for software maintenance tasks.Information and Software Technology, 66:1–12, 2015.
13. Edmund Wong, Taiyue Liu, and Lin Tan. Clocom: Mining existing source code for automatic comment generation. In Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on, pages 380–389.IEEE, 2015.
14. Yuan Huang, Qiaoyang Zheng, Xiangping Chen, Yingfei Xiong, Zhiyong Liu, and Xiaonan Luo. Mining version control system for automatically generating commitcomment. In Proceedings of the 11th ACM/IEEE International Symposium onEmpirical Software Engineering and Measurement, pages 414–423. IEEE Press,2017.
15. Ahmet Okutan and Olcay Taner Y\_ld\_z. Software defect prediction using bayesian networks. Empirical Software Engineering, 19(1):154–181, 2014.
16. Julie Moeyersoms, Enric Junqu´ e de Fortuny, Karel Dejaeger, Bart Baesens, and David Martens. Comprehensible software fault and effort prediction: A data mining approach. Journal of Systems and Software, 100:80–90, 2015.
17. Stacy Nelson and Johann Schumann. What makes a code review trustworthy? In System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on, pages 10–pp. IEEE, 2004.
18. Anne Edmundson, Brian Holtkamp, Emanuel Rivera, Matthew Finifter, Adrian Mettler, and David Wagner. An empirical study on the effectiveness of security code review. In International Symposium on Engineering Secure Software and Systems, pages 197–212. Springer, 2013.
19. Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E Hassan. An empirical study of the impact of modern code review practices on software quality.Empirical Software Engineering, 21(5):2146–2189, 2016.
20. 孙小兵, 李斌, 陈颖, 李必信, and 文万志. 软件修改影响分析研究与进展. 电子学报, 42(12):2467–2476, 2014.
21. Salah Bouktif, Yann-Gael Gueheneuc, and Giuliano Antoniol. Extracting change-patterns from cvs repositories. In Reverse Engineering, 2006. WCRE’06. 13th Working Conference on, pages 221–230. IEEE, 2006.
22. Shawn A Bohner and RS Arnold. Software change impact analysis for design evolution. In Proceedings of 8th International Conference on Maintenance and Re-engineering, pages 292–301. IEEE CS Press Los Alamitos, CA, 1991.
23. Seunghun Park and Doo-Hwan Bae. An approach to analyzing the software process change impact using process slicing and simulation. Journal of Systems and Software, 84(4):528–543, 2011.
24. Huzefa Kagdi, Malcom Gethers, and Denys Poshyvanyk. Integrating conceptual and logical couplings for change impact analysis in software. Empirical Software Engineering, 18(5):933–969, 2013.
25. Hani Abdeen, Khaled Bali, Houari Sahraoui, and Bruno Dufour. Learning dependency-based change impact predictors using independent change histories. Information and Software Technology, 67:220–235, 2015.
26. Taweesup Apiwattanapong, Alessandro Orso, and Mary Jean Harrold. Efficient and precise dynamic impact analysis using execute-after sequences. In Proceedings of the 27th international conference on Software engineering, pages 432–441.ACM, 2005.
27. Lulu Huang and Yeong-Tae Song. Precise dynamic impact analysis with dependency analysis for object-oriented programs. In Software Engineering Research,Management & Applications, 2007. SERA 2007. 5th ACIS International Conference on, pages 374–384. IEEE, 2007.
28. Chin-Yu Lin, Tung-Yueh Wu, and Chin-Cheng Huang. Nonlinear dynamic impact analysis for installing a dry storage canister into a vertical concrete cask. International Journal of Pressure Vessels and Piping, 131:22–35, 2015.
29. Haipeng Cai and Douglas Thain. Distia: A cost-effective dynamic impact analysis for distributed programs. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, pages 344–355. ACM, 2016.
30. Ben Breech, Mike Tegtmeyer, and Lori Pollock. Integrating influence mechanisms into impact analysis for increased precision. In Software Maintenance, 2006. ICSM’06. 22nd IEEE International Conference on, pages 55–65. IEEE, 2006.
31. James Law and Gregg Rothermel. Whole program path-based dynamic impact analysis. In Proceedings of the 25th International Conference on Software Engineering, pages 308–318. IEEE Computer Society, 2003.
32. Haipeng Cai and Raul Santelices. Diver: Precise dynamic impact analysis using dependence-based trace pruning. In Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, pages 343–348. ACM,2014.
33. Lionel Briand, Yvan Labiche, and George Soccar. Automating impact analysis and regression test selection based on uml designs. In icsm, page 0252. IEEE,2002.
34. Jessica Daz, Jennifer Perez, Juan Garbajosa, and Alexander L Wolf. Change impact analysis in product-line architectures. In European Conference on Software Architecture, pages 114–129. Springer, 2011.
35. Denys Poshyvanyk, Andrian Marcus, Rudolf Ferenc, and Tibor Gyim´ othy. Using information retrieval based coupling measures for impact analysis. Empirical software engineering, 14(1):5–32, 2009.
36. Arpad Beszedes, Tamas Gergely, Szabolcs Farago, Tibor Gyimothy, and Ferenc Fischer. The dynamic function coupling metric and its use in software evolution. In Software Maintenance and Reengineering, 2007. CSMR’07. 11th European Conference on, pages 103–112. IEEE, 2007.
37. Thomas Rolfsnes, Stefano Di Alesio, Razieh Behjati, Leon Moonen, and Dave W Binkley. Generalizing the analysis of evolutionary coupling for software change impact analysis. In 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), pages 201–212. IEEE, 2016.
38. Lile Hattori, Gilson dos Santos Jr, Fernando Cardoso, and Marcus Sampaio. Mining software repositories for software change impact analysis: a case study. In Proceedings of the 23rd Brazilian symposium on Databases, pages 210–223. Sociedade Brasileira de Computacao, 2008.
39. Bixin Li, Xiaobing Sun, Hareton Leung, and Sai Zhang. A survey of code-based change impact analysis techniques. Software Testing, Verification and Reliability, 23(8):613–646, 2013.
40. Patanamon Thongtanunam, Chakkrit Tantithamthavorn, Raula Gaikovina Kula, Norihiro Yoshida, Hajimu Iida, and Ken-ichi Matsumoto. Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. In Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on, pages 141–150. IEEE, 2015.
41. Oleksii Kononenko, Olga Baysal, Latifa Guerrouj, Yaxin Cao, and Michael W Godfrey. Investigating code review quality: Do people and participation matter? In Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on, pages 111–120. IEEE, 2015.
42. Olga Baysal, Oleksii Kononenko, Reid Holmes, and Michael W Godfrey. Investigating technical and non-technical factors influencing modern code review. Empirical Software Engineering, 21(3):932–959, 2016.
43. Vijayaraghavan Murali, Swarat Chaudhuri, and Chris Jermaine. Bayesian specification learning for finding api usage errors. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pages 151–162. ACM, 2017.
44. Chris Mills. Automating traceability link recovery through classification. In Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pages 1068–1070. ACM, 2017.
45. Michael Rath, Jacob Rendall, Jin LC Guo, Jane Cleland-Huang, and Patrick Mader. Traceability in the wild: automatically augmenting incomplete trace links. In Proceedings of the 40th International Conference on Software Engineering,pages 834–845. ACM, 2018.
46. Syaeful Karim, Harco Leslie Hendric Spits Warnars, Ford Lumban Gaol, Edi Ab-durachman, Benfano Soewito, et al. Software metrics for fault prediction using machine learning approaches: A literature review with promise repository dataset.In Cybernetics and Computational Intelligence (CyberneticsCom), 2017 IEEE International Conference on, pages 19–23. IEEE, 2017.
47. Kento Shimonaka, Soichi Sumi, Yoshiki Higo, and Shinji Kusumoto. Identifying auto-generated code by using machine learning techniques. In Empirical Software Engineering in Practice (IWESEP), 2016 7th International Workshop on, pages 18–23. IEEE, 2016.
48. Anh Tuan Nguyen, Hoan Anh Nguyen, Tung Thanh Nguyen, and Tien N Nguyen. Statistical learning approach for mining api usage mappings for code migration. In Proceedings of the 29th ACM/IEEE international conference on Automated software engineering, pages 457–468. ACM, 2014.
49. Mario Linares-V´ asquez, Collin McMillan, Denys Poshyvanyk, and Mark Grechanik. On using machine learning to automatically classify software applications into domain categories. Empirical Software Engineering, 19(3):582–618,2014.
50. Annie TT Ying, Gail C Murphy, Raymond Ng, and Mark C Chu-Carroll. Predicting source code changes by mining change history. IEEE transactions on Software Engineering, 30(9):574–586, 2004.
51. Thomas Zimmermann, Andreas Zeller, Peter Weissgerber, and Stephan Diehl. Mining version histories to guide software changes. IEEE Transactions on Software Engineering, 31(6):429–445, 2005.
52. Nemitari Ajienka, Andrea Capiluppi, and Steve Counsell. An empirical study on the interplay between semantic coupling and co-change of software classes. Empirical Software Engineering, 23(3):1791–1825, 2018.
53. Fehmi Jaafar, Yann-Ga¨ el Gu´ eh´ eneuc, Sylvie Hamel, and Giuliano Antoniol. Detecting asynchrony and dephase change patterns by mining software repositories.Journal of Software: Evolution and Process, 26(1):77–106, 2014.
54. Stephane Vaucher, Houari Sahraoui, and Jean Vaucher. Discovering new change patterns in object-oriented systems. In Reverse Engineering, 2008. WCRE’08. 15th Working Conference on, pages 37–41. IEEE, 2008.
55. Beat Fluri, Emanuel Giger, and Harald C Gall. Discovering patterns of change types. In Proceedings of the 2008 23rd IEEE/ACM International Conference on Automated Software Engineering, pages 463–466. IEEE Computer Society, 2008.
56. Amir Michail. Data mining library reuse patterns in user-selected applications. In ase, page 24. IEEE, 1999.
57. Yu Lin and Danny Dig. Check-then-act misuse of java concurrent collections. In 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation, pages 164–173. IEEE, 2013.
58. Reid Holmes, Robert J Walker, and Gail C Murphy. Approximate structural context matching: An approach to recommend relevant examples. IEEE Transactions on Software Engineering, (12):952–970, 2006.
59. Marcel Bruch, Martin Monperrus, and Mira Mezini. Learning from examples to improve code completion systems. In Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pages 213–222. ACM, 2009.
60. Andy Zaidman, Bart Van Rompaey, Serge Demeyer, and Arie Van Deursen. Mining software repositories to study co-evolution of production & test code. In Software Testing, Verification, and Validation, 2008 1st International Conference on, pages 220–229. IEEE, 2008.