南京大学

研 究 生 毕 业 论 文

（申请硕士专业学位）

论　文　题　目　一种结合代码依赖和用户反馈的软件可追踪生成方法

作　者　姓　名　张宗飞

学　科、专　业　计算机技术

研　究　方　向　软件维护

指　导　教　师　胡昊　副教授

2018 年 5 月 27 日

学　　号：**MF1533075**

论文答辩日期：**2018 年 5 月 27 日**

指 导 教 师：　　（签字）

An IR-based approach combining with code dependencies for updating requirements

By

Zongfei Zhang

Supervised by

Associate Professor **Hao Hu**

A Thesis

Submitted to the Department of Computer Science and Technology

and the Graduate School

of Nanjing University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Engineering

Institute of Computer Software

May 2018

南京大学研究生毕业论文中文摘要首页用纸

毕业论文题目：一种结合代码依赖和用户反馈的软件可追踪生成方法

计算机技术 专业 2015 级硕士生姓名： 张宗飞

指导教师（姓名、职称）： 胡昊 副教授

摘 要

在软件开发过程中，会产生多种软件制品，例如代码、需求文档、测试集合等。软件制品之间的追踪线索尤其是需求到代码的追踪线索对利益相关者有重要的作用。需求到代码的追踪线索能够描述高层到底层实现之间的关系，对于软件理解、影响分析、软件维护等活动都具有重要意义。但是建立需求到代码之间的追踪线索需要耗费大量时间和精力。软件工程师需要阅读并理解不同的软件制品，才能判断不同软件制品之间是否具有追踪关系。

当前获取获取需求到代码可追踪性的主流方法是信息检索方法，通过计算需求和代码之间的文本相似度来辅助用户判断两者之间是否存在追踪关系。由于需求和代码可能使用不同的术语表，这就使得两者存在单词失配问题。严重影响生成结果的精度。为了解决这个问题，有些工作提出了一些增求策略，例如考虑代码结构信息，考虑用户反馈等。对于只考虑代码结构信息的工作，其精度严重依赖于信息检索的结果；而对于考虑用户反馈的现有工作，往往需要大量的用户反馈，在实际中可操作性不强。

针对上述问题，本文完成了以下工作：

1. 基于代码依赖紧密度分析和用户反馈的可追踪性生成方法。我们提出了一种结合了代码紧密度分析和用户反馈的软件可追踪生成方法。一方面通过设置代码紧密度阈值划分代码域，使得功能相似的代码元素位于同一个代码域中；另一方面，将各代码域中有代表性的代码元素交由用户判断与需求相关性，根据用户反馈信息调整相关代码元素对应候选线索在排序表中的位置。

2. 方法的验证及技术来源分析。我们在四个研究案例下设计实验，验证了上述方法的有效性。并且，介绍了在实验所用不同类型软件系统的代码依赖捕获过程。此外，对于实验中所用开源系统，我们阐述其可追踪数据集的整理过程。

3. 基于代码依赖和用户反馈的软件可追踪生成工具的设计与实现。目前，领域内并没有相应的结合代码依赖和用户反馈的可追踪生成工具可以应用于实际的软件演化过程。因此，我们设计并实现软件可追踪生成工具，并集成了我们基于代码依赖紧密度分析和用户反馈的软件可追踪生成方法。

关键词： 需求可追踪性，信息检索，紧密度分析，用户反馈，代码依赖

南京大学研究生毕业论文英文摘要首页用纸

THESIS: An IR-based approach combining with code dependencies for updating requirements

SPECIALIZATION: Computer Software and Theory

POSTGRADUATE: Zongfei Zhang

MENTOR: Associate Professor Hao Hu

Abstract

There will be variety of software artifacts, such as code artifacts、 requirement document artifacts and test case artifacts and so on, during the process of software development. The traceability between software artifacts especially requirement-to-code play an good important role for stakeholders. The traceability between requirement and code make a great significance for software understanding, impact analysis and software maintenance. However, it takes a lot time and effort to establish the traceability between requirement and code.Software engineers need to read and understand different software artifacts to determine whether there is a trace between them.

The information retrieval is mainstream method of obtaining traceability between requirement and code, it assist user to judge whether there exist trace between requirement and code through calculating text similarity between them. It will cause word mismatch for the requirement and the code use different glossaries. In order to solve this problem, some work put forward some enhancing strategies, such as considering code structure information and user feedback. Unfortunately, on the one hand, the performance of work which considers the code structure information only depends heavily on the result of information retrieval; on the other hand, the work that only considers the user feedback will need a large amount of user feedback which is infeasible in practice.

For the above problems, we finished several works as following:

1. We proposed an IR-based approach combining user feedback with closeness analysis on code dependencies. On the one hand, we build candidate regions through setting closeness threshold. On the other hand, for a given requirement,

we choose the class that has the highest IR value in each region as the representative class. Then we ask user to iteratively verify these representative classes for each region and adjust the position of relevant candidate link base on user feedback.

2. We evaluated the above traceability recovery approach on four different case studies. Meanwhile, we introduce the process of code dependency capture about different experiment software system. In addition, we describe the process of mange requirement traceability dataset for open source system used in experiment.

3. We also developed an assistant tool for traceability recovery between requirement and code and integrating the above approach into it.

Keywords: traceability recovery, information retrieval, closeness analysis, user feedback, code dependencies

目　录

插　　图

第一章　绪言

1.1　研究背景

软件可追踪性是指在软件开发过程中建立和维护软件制品之间的关联关系，并利用这些关系对软件项目进行一系列分析的能力 [1]。如图，在软件开发过程中，会产生各种软件制品，包括需求文档、软件架构、设计文档、用户手册和源代码等 [2]。这些软件制品之间存在追踪关系。这些不同软件制品之间的追踪关系可以使得软件工程师更好的理解系统，保证软件的质量，同时也可以降低软件维护成本，有利于软件后期的维护和演化 [3–5]。需求文档是对软件功能的描述，代码是软件的底层实现方式。因此需求到代码的可追踪性尤为重要也是当前领域的热点。

通过对照实验发现，对于软件维护工作而言，在需求到代码可追踪性的支持下，维护的正确率提高了60%，效率提高了21% [6]。事实上，需求到代码的追踪数据描述了软件高层抽象和底层代码实现之间的对应关系。软件迭代过程中，当需求发生变更时，根据需求到代码的追踪数据可以快速找到要更新的代码位置。此外需求到代码的追踪信息可以应用于软件生命周期以支持变更影响分析、依赖影响分析、系统验证以及安全认证等活动 [7, 8]。然而，建立和维护需求到代码的追踪关系会耗费用户大量的精力并且很容易出错 [9]。在工业环境中，由于软件系统复杂度比较高，迭代速度比较快，人工的建立和维护需求到代码的追踪因关系代价高昂被很多组织放弃 [9, 10]。当前领域内建立需求到代码追踪关系的主要方法是信息检索方法（Information Retrieve），但是此类方法严重依赖需求文档和代码文档的文本质量，生成的追踪数据质量不高。为解决这个问题， [11, 12]考虑在信息检索方法的基础上考虑代码结构信息，但是这些方法往往受信息检索结果质量的影响。 [13–15]考虑用户反馈信息，但是这些方法往往需要大量的用户参与。因此，如何在减少用户参与的情况下，生成高质量的软件可追踪性数据是当前领域内研究的重点。

1.2　研究现状

在本节中，我们将会讨论软件可追踪生成领域的现状，主要包括当前可追踪生成的一些主流方法。我们所关注的软件可追踪性是需求到代码元素的可追踪性。为此，能够直接关联需求与代码的主要方式，就是需求可追踪性

（Traceability）。需求可追踪性能够追踪一个需求使用期限的全过程，提供了需求到产品整个过程范围的明确的查阅能力，以辅助利益相关者完成相应的软件活动。其中，需求到代码的可追踪性是领域内关注的热点。在近十年中，需求可追踪性是一个非常活跃的研究领域。在已有的工作中，涉及包括需求可追踪性问题分析 [10]，需求可追踪性相关模型 [9]，需求可追踪性的自动化生成，追踪关系演化的管理 [16]，追踪关系的自动化维护 [17]及追踪关系生成与管理工具 [18]等。本文的主要工作是需求可追踪性的自动生成。

尽管需求可追踪性对于软件系统演化的管理至关重要，但是它在实际中的使用仍然很有限。这主要是由于需求可追踪性的建立与维护需要高昂的人工与时间成本。为了解决这个问题，领域内采用一些半自动化的方法。当前主流的追踪关系生成方法是基于信息检索技术以计算实体间的相似性，并生成候选追踪关系 [3, 13, 19, 20]。用户自上而下遍历候选列表并判断候选线索的有效性。这种方法的精度严重依赖文本质量，当文本质量比较差时精度比较低。这是因为不同软件制品（例如需求和代码）经常采用不同的术语来描述同一个概念（单词试配问题）。为了解决这个问题，研究者在不同角度引入了多种增强策略 [21**?**]对字典进行了进一步分析。 [11, 12]加入了代码结构信息。

另一方面，大量用户使用用户反馈信息（user feedback）来提高精度，[13]要求用户迭代式判断追踪线索的有效性，然后根据用户反馈更改单词权重。[15]提出一种结合用户反馈和代码依赖的方法，这种方法要求用户遍历整个候选列表，这对用户来说是不实际的。

为了提高追踪关系生成方法的精度，信息检索技术被用于与机器学习 [22]，用户反馈[15]，动态分析 [23]等方式相结合。此类方法的一个局限在于，当候选追踪关系的召回率水平较高时，获取的候选追踪关系准确率较低（引入了大量的误报）。因此，维护人员需要人工检查并确认生成的追踪关系是否正确，从而使基于信息检索的追踪关系生成方法退化为半自动的方法。

1.3　本文工作

在对需求到代码可追踪性生成等相关领域进行了深入研究后，为了解决上述问题，本文完成了以下工作：

1. 基于代码依赖紧密度分析和用户反馈的可追踪性生成方法。我们提出了一种结合了代码紧密度分析和用户反馈的软件可追踪生成方法。该方法有效利用了代码依赖紧密度和用户反馈两部分信息，一方面通过设置代码紧密度阈值划分代码域，使得功能相似的代码元素位于同一个代码域中；另一

方面，将各代码域中有代表性的代码元素交由用户判断与需求相关性，根据用户反馈信息调整相关代码元素对应候选线索在排序表中的位置。

2. 方法的验证及技术来源分析。我们在四个研究案例下设计实验，验证了上述方法的有效性。并且，介绍了在实验所用不同类型软件系统的代码依赖捕获过程。此外，对于实验中所用开源系统，我们阐述其可追踪数据集的整理过程。

3. 基于代码依赖和用户反馈的软件可追踪生成工具的设计与实现。目前，领域内并没有相应的结合代码依赖和用户反馈的可追踪生成工具可以应用于实际的软件演化过程。因此，我们设计并实现软件可追踪生成工具，并集成了我们基于代码依赖和用户反馈的软件可追踪生成方法。

1.4　本文组织

针对当前软件可追踪生成领域存在的问题，本文研究软件可追踪性生成这一问题。综合考虑代码依赖和用户反馈两部分信息，以提高结果精度并使用户少量参与为目标，从代码依赖获取和软件可追踪生成两个场景出发，研究了代码依赖和用户反馈如何有效协作提高候选列表精度。论文总共六章，第一章作为绪论，提出了文本的研究背景及概要结构，其余五章的概要内容总结如下：

第二章主要介绍了与本文研究相关的概念与工作，包括软件可追踪生成，结合代码依赖关系，用户反馈和信息检索的相关技术。

第三章介绍基于代码依赖和用户反馈的软件可追踪生成方法。介绍如何通过代码依赖紧密度阈值划分代码域，使得功能相似的代码元素位于同一个代码域中以及如何结合代码域和用户反馈提高软件可追踪生成精度。

第四章介绍了我们对于结合代码依赖和用户反馈的软件可追踪生成方法的实验验证及技术来源。我们在四个研究案例下设计实验，验证了我们方法的有效性。并且，我们阐述了代码依赖的获取方法，提出了通过运行测试集获得代码依赖的方法。此外，对于实验中所用开源系统，我们阐述其可追踪数据集的整理过程。

第五章介绍了软件可追踪生成工具的设计实现，及其在实际软件活动过程中的应用。

第六章是对全文工作的总结，并基于我们已有的研究内容展望未来工作。

第二章　相关工作

2.1　需求可追踪性

需求跟踪是指跟踪一个需求使用期限的全过程，需求跟踪包括编制每个需求同系统元素之间的联系文档，这些元素包括其他类型的需求，体系结构，其他设计部件，源代码模块，测试，帮助文件等。需求跟踪为我们提供了由需求到产品实现整个过程范围的明确查阅的能力。Gotel对需求追踪问题进行了一系列的实际调查和分析，并给出了需求追踪的定义：在软件整个生命周期中，对某一特定需求从前后两个方向描述和追踪的能力。需求追踪分成两个方向，前向追踪是从书写文档形式的需求追踪到需求来源，后向追踪是从需求文档软件发布过程中的各种制品（例如测试集合、代码、设计文档等）[10]。

需求到其它任何软件制品的追踪关系都可以用一个二维矩阵来描述。例如，表2.1描述了需求到代码的可追踪关系。 行为代码项，列为需求项。每一行代表该代码项与哪些需求有关，每一列表示该需求由哪些代码项来完成。‘X’表示代码项和需求项之间存在实现关系（追踪关系）。在实际的软件生成过程中，代码的粒度通常是类或者函数，需求的粒度可以是用例（Use Case）或声明（requirement Statement）。

2.2　信息检索技术

信息检索技术是软件可追踪生成领域中研究最多、应用最广泛的分析方法。该方法通过计算查询文档和目标文档之间的文本相似度，进而检索出与查

表 2.1: 需求追踪矩阵（需求到代码）

	$Requirement_1$	$Requirement_2$	$Requirement_3$	...	$Requirement_n$
$Code_1$	X				
$Code_2$		X			
$Code_3$	X				X
$Code_4$			X		X
...					
$Code_n$		X			

询文档相关的目标文档，该方法具有人工参与少、自动化程度高、易于实现等特点。本小节主要介绍VSM、LSI和JS [24]三种常用的信息检索模型。VSM模型将对文本内容的处理简化为向量空间中的向量运算，用向量来表示查询和目标文档，目标文档根据向量之间的余弦距离排序；LSI是一个简单实用的主题模型，基于奇异值分解（SVD）的方法得到文本的主题，克服了VSM模型中存在的近义词和多义词问题。JS属于概率模型，估计文档和查询相关的概率。

2.2.1 VSM

VSM（Vector Space Model）在信息检索和搜索引擎中应用非常广泛，也是一种自然语言处理中常用的模型。该模型用向量来表示查询和目标文档，将对文本内容的处理转化为向量空间种的向量运算。查询和目标文档中的每个此项用向量中的一个维度来描述。

如果文档中没有出现某个词项，此时该查询向量与该词项对应维度的值就是零。这个值用来刻画词项在文档中的权重。有很多计算词项权重的方法，其中应用比较广泛的计算方法为TF-IDF，它有词项的局部权重和全局权重两部分组成。TF（term frequency）表示当前文档中的某个词项在当前文档中的出现频率。直觉上，某一词项在文档中出现的次数越多权重应该越大，但是这里有个缺陷在于，不同文档长度不同。长文档包含的词项比较多。因此我们用某词项出现的次数除以其所在文档的总词项数来表示该词项的频率。TF计算公式如下：

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \tag{2.1}$$

其中$n_{i,j}$表示词项t_i在文档d_j中出现的次数。分母表示文档d_j中出现的总词项数。结果$tf_{i,j}$表示词项t_i 在文档d_j中出现的频率。

对于词项的全局权重，如果一个词项普遍存在于各个文档中。极端条件下所有文档中都有该词项，则该词项没办法用来区分不同的文档，即其权重比较低。如果只有很少的文档出现了某词项，那么通过该词项很容易区分开出现过它的文档和未出现它的文档，所以此时应该赋予该词项较高的权重。IDF计算公式如下：

$$idf_i = \log \frac{|D|}{\{|j : t_i \in d_j|\}} \tag{2.2}$$

其中$|D|$ 表示总的文档个数，分母表示出现词项t_i 的文档数目

TF-IDF值由词项的局部权重值和全局权重值综合决定，其计算公式如下：

$$tfidf_{i,j} = tf_{i,j} \times idf_i \tag{2.3}$$

即词项的本地权重值乘以词项的全局权重值。

相似度计算的合理性在于，存在较多共享词项的两个文档，往往是在描述同样的信息。描述不同信息的文档往往使用不同的词项。由于向量空间模型将目标文档与查询文档用高维空间中的一个向量来表示，这两个文档之间的文本之间的相似度可以用这两个空间向量的相似度来表示，给定查询向量q, 目标文档向量d, 向量长度均为n, 那么其余弦相似度定义为：

$$sim(d_j, q) = \frac{\sum_{i=1}^{n} w_{i,j} \times w_{i,q}}{\sqrt{\sum_{i=1}^{n} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{n} w_{i,q}^2}} \tag{2.4}$$

其中，w表示对应文档中由TF-IDF计算而来的词项权重。

2.2.2　LSI

LSI（Latent semantic indexing，潜在语义索引） [25, 26] VSM模型中并没有考虑词项之间的关联性。例如，对于分别出现了“automobile”和“car”词项的两个文档，虽然这两个词项是同义词均表示“汽车”。但是在VSM 模型下，两个文本之间的相似度不会因为有大量同义词而变大。LSI模型能够捕获词项之间的相关性，如果两个词项之间具有很强的相关性，那么当一个词项出现时，往往意味着另一个词项也会出现（同义词）；反之，如果查询文档或者目标文档中的某个词项和其它词项的相关性都不大，那么这个词项很可能表示的是另一个意思（一词多义，比如Apple 在不同语境下可能是水果也可能是指手机）。LSI采用奇异值矩阵分解（SVD） [27]对VSM模型下的词项-文档矩阵（Term-by-Document）进行分解。奇异值矩阵分解可以看作是从单词- 文档矩阵中发现不相关的索引变量（因子），将原来的数据映射到语义空间内。在词项-文档矩阵中不相似的两个文档，可能在语义空间内比较相似。SVD背后的形式化比较复杂，详细内容可参考文献 [27]。简单来说，奇异值分解是对矩阵进行分解的一种方法。它能够将一个大矩阵表示为三个小矩阵的乘积，并且与特征分解不同，奇异值分解不要求被分解矩阵为方阵。假设我们的矩阵A 是一个$m \times m$的矩阵，其中的元素全部属于域K，也就是实数域或复数域。如此则存在一个分解使得：

$$A = U\Sigma V^T \tag{2.5}$$

其中U是$m \times m$阶酉矩阵，Σ是半正定$m \times n$ 阶对角矩阵，而V^T即V的共轭转置，是$n \times n$阶酉矩阵。这样的分解就称做M的奇异值分解。Σ对角线上的元素即为矩阵A的奇异值。常见的做法是对奇异值自大到小进行排序，这样Σ可以由矩阵A唯一确定。可以直观的理解为，矩阵U的列向量（左奇异向量）组成一套对矩阵A 的正交“输入”或“分析”的基向量。这些向量是AA^T的特征向量。矩阵V的列向量（右奇异向量）组成一套对矩阵A 的“正交”输出的基向量。这些向量是$M^T M$ 的特征向量。Σ 对角线上的元素是奇异值，可视为是输入与输出间进行的标量的“膨胀控制”。这些是MM^T及$M^T M$ 的奇异值，并与U和V的列向量相对应。对奇异值自大到小排序，取前k个A的奇异值利用$X_k = U_k \Sigma_k V_k^T$构造X的秩－k近似矩阵，重建的矩阵是一个最小二乘法拟合。其中U和V 的列向量正交，即$U^T U = V^T V = I_r$，其中r是原矩阵X的秩。X_k为原矩阵X的k维最佳近似矩阵，由k个最大的奇异三元组构造而成。

LSI通过奇异值矩阵分解将原始词项文档矩阵降维，投影到一个缩小的空间中，以消除词语使用中的“噪声”。在LSI模型中，检索是基于文档的语义内容而不是其词汇内容。为了取得较好的检索结果，选择合适的K值很重要。理想情况下，我们选择一个K值能包含数据中所有的主题模型，同时能过滤掉所有噪音。实际中，选择最优K值的方法还是一个公开问题[28]。当前K 值的选择通常由实验决定。降维之后的查询、文档向量通过将VSM 空间的相关向量投影至LSI 子空间生成，类似VSM模型，查询语句与目标文档之间的相似度即为对应向量的余弦距离。

2.2.3　JS

JS模型（Jensen-Shannon similarity model）是一种由概率方法和假设检验技术驱动的新型信息检索技术。在概率模型中，查询语句和目标文档都被假定具有潜在的概率分布。通过对查询和目标文档对应概率分布的距离进行排序的方法，达到检索的目的。根据假设检验 [**?**]的结论，将查询和目标文档看作是词项的概率分布，两者之间的差异通过JS散度来度量。公式如下：

$$JS(q,d) \triangleq H(\frac{\hat{p}_q + \hat{p}_d}{2}) - \frac{H(\hat{p}_q) + H(\hat{p}_d)}{2} \tag{2.6}$$

$$H(p) \triangleq \sum_{w \in W} h(p(w)),\ h(x) \triangleq -x \log x \tag{2.7}$$

其中$H(p)$代表概率分布p的熵，$\hat{p}_q$和$\hat{p}_d$ 分别为查询语句和目标文档的概率分布。我们注意到，根据定义当x 为0 时，h(0) 恒等于0。因此我们定义查询语句和目标文档之间的相似度为1-JS(q,d)。

2.3　结合用户反馈

一方面鉴于信息检索方法严重依赖检索语句和目标文档的质量；另一方面信息检索生成的目标文档排序列表需要用户验证每个文档和查询语句的相关性。这是个用户交互的重复过程。 [13, 14, 29, 30] 使用用户反馈对候选追踪线索的判断信息对目标文档排序表进行优化。 [13]假设q是一个查询向量，集合D_q 是用信息检索方法通过查询q 得到的文档集合。假设文档集合D 由两个子集合D_r与D_{irr}组成，其中D_r集合中的文档是用户判断结果为与查询语句相关的文档，D_{irr}为与查询语句无关的文档，它们所包含的文档个数分别为R 和S。很明显D_r集合和D_{irr}集合没有交集，并且由于这两个集合里面都是用户判断过的文档，所以它们的并集也未必是D_q，因为可能有些文档用户还没判断。 [13]使用标准Rochio [31] 反馈处理算法，在下一轮迭代中修改查询向量各词项的权重，公式如下：

$$q_{new} = \alpha \cdot q + (\frac{\beta}{R} \sum_{d_j \in D_r} d_j) - (\frac{\gamma}{S} \sum_{d_k \in D_{irr}} d_k) \tag{2.8}$$

直觉上，通过将相关文档中出现的词项加到查询语句中可以使得查询语句与其它相关文档共享更多的词项，从而提高完全率；削弱查询语句中在不相关文档中出现词项的权重，可以使得查询语句远离不相关文档，进而减小犯错的概率，增加准确率。上式子中，可以通过调整参数α 、β 、γ 来改变原始查询语句、相关文档和不相关文档对生成新查询语句的重要性。新查询语句生成之后，重新使用信息检索方法并对检索得到的相关文档按相似度排序，重复以上过程，直到用户取得满意的结果。然而后续工作 [32] 发现在做需求追踪时用Rochio反馈处理方法只有在前几次迭代中会提高准确率和完全率。

[14]表明标准Rochio反馈处理方法对信息检索方法生成排序表的优化并不明显，尤其是当查询语句和文档语料质量比较好，使得信息检索的结果质量比较高时，Rochio 反馈处理方法对信息检索的结果几乎起不到任何优化作用，对于其它软件制品之间做可追踪性（例如用例和UML图之间）使用Rochio 方法不仅不能提升排序表的效果反而可能会让效果并得更差。 [30] 对在 [13] 的基础上对不同查询语句进行了分类，与 [13] 使用固定的参数α 、β 、γ 不同， [30]根据分类采用不同的参数并取得一定的效果。根据 [33]，使用用户反馈（relevance feedback）信息提高对查询语句的检索效果要基于两个前提条件：

1. Rocchio算法基于的假设是：查询语句与目标文档相比，词项比较少。
2. 与查询语句相关的文档之间相似度比较高，假如对查询和文档进行分析，查询和相关文档应该能聚到同一个类中（聚类假设）。

[29]认为在需求追踪环境下以上两个前提假设均不成立，有些查询语句（需求的文本文档）词项比目标文档（代码文档）要多。因此像之前工作 [13, 14, 32] 那样不考虑前提条件是否满足直接使用用户反馈。并且，聚类假设也不成立，根据相关文献 [34, 35]，需求实体和代码实体往往使用不同的词典，需求实体往往使用问题领域词项，代码实体则使用技术领域词项和一些缩写同义词等。综上 [29]提出一种方法，只有查询语句的词项小于等于相关文档，并且用户判断的文档中与查询语句相关的文档比不相关的文档多时才对用户反馈使用Rocchio算法。详见算法 1：

与上述工作从通过用户对候选线索的反馈结果优化排序不同， [36]基于n-gram模型对查询语句进行分析，对不同的词项采取不同的权重调整策略。（增大，减小，不变）。

与上述迭代式方法不同， [4, 37]先让用户判断一部分候选追踪线索，作为方法的训练集。具体来 说， [4]使用一组被判断过的追踪线索（训练集）作为贝叶斯分类器的输入，通过对测试集进行学习来提高基于概率模型的信息检索方法的准确率。 [38]提出了一种使用嵌入词项和RNN技术的神经网络结构来生成软件制品之间相关性的方法。与基于信息检索的方法类似，该方法也会分析查询语句和目标文档的文本相似度，将用户已经判断过的追踪线索作为训练集作为方法的输入。相对于传统的基于VSM 和LSI模型的信息检索方法， [38] 方法得到的结果分别提高了41%和32%。但是该方法需要45%的测试集和10%的用户已经判断的追踪线索对通过训练集得到的分类模型进行优化。即用户需要判断所以候选追踪线索的55%。这在实践中会耗费大量的用户精力。

2.4　结合代码依赖关系

在基于信息检索方法对需求到代码的可追踪性生成过程中，往往因为词汇失配问题（需求往往使用领域相关词汇，代码往往使用缩写，同义词，专业术语等）使得生成追踪列表的精度（准确率，召回率）有限。本小节主要介绍使用代码依赖信息对信息检索得到的排序表进行优化，弥补词汇失配问题的一些相关工作。

对于基于信息检索方法生成的候选追踪线索列表， [11]利用代码依赖关系对列表重排序。我们的之前工作 [39] 对代码依赖进行了细分，直接依赖（类之间的继承、使用和方法之间的调用）、数据依赖（类之间的数据共享），并且认为直接依赖表达的是软件的控制流，数据依赖表达的是软件的数据流。 [39]表明对于需求到代码的可追踪性生成，代码元素直接的直接依赖和数据依赖起到互补的作用，两者相结合要比只使用其中一种取得更好的效果。后续工作 [12]

Algorithm 1: Adaptive Relevance Feedback (ARF)

```
List ← initial ranked list of candidate links
Classified ← ∅;
forall the artefact i do
    TP_i ← ∅;
    FP_i ← ∅;
while not (stoppingcriterion) do
    Get the link (s,t) on top of List
    The user classifies (s,t)
    Classified ← Classified ∪ {(s,t)}
    if (s,t) is correct then
        TP_s ← TP_s ∪ {t}
        TP_t ← TP_t ∪ {s}
    else
        FP_s ← FP_s ∪ {t}
        FP_t ← FP_t ∪ {s}
    Let V_s be the set of terms in s
    Let V_t be the set of terms in t
    if |V_s| ≤ |V_t| and |TP_s| ≥ |FP_s| then
        apply the standard Rocchio to s
    else
        if |V_t| ≤ |V_s| and |TP_t| ≥ |FP_t| then
            apply the standard Rocchio to t
    Recompute the ranked List of links
    List ← List − Classified
```

对代码依赖之间的紧密度进行了量化。对于代码之间的直接依赖，该工作认为两个类交互越越多并且和两个类和其它类交互的越少，则这两个类之间的代码依赖紧密度越高，公式如下：

$$Closeness_{DC} = \frac{2N}{WeightInDegree_{Sink} + WeightOutDegree_{Source}} \tag{2.9}$$

其中N表示类$Sink$和$Source$之间方法调用和类使用的次数，$WeightInDegree_{Sink}$表

示类$Sink$被调用或使用的次数，$WeightOutDegree_{Source}$表示类$Source$调用或使用其它类的次数。$Closeness_{DC}$位于区间[0,1]，该值越大表明两者关系越紧密。[12]基于信息检索技术并结合代码依赖紧密度分析做需求和代码之间的可追踪性并确的一定成果。 [12]的方法是对于给定查询语句，对检索结果进行排序。排在第一位的文档往往是有效的。根据其它文档与排在第一位文档的代码依赖紧密度，调整它们在排序表的位置。该工作包含两个步骤，首先设定代码依赖紧密度阈值，并由与排名第一的文档紧密度大于阈值的文档组成一个初始地域，详见算法 2，修改域内文档（这里是类）与查询语句（需求）的紧密度。

Algorithm 2: Establishing Initial Region

```
1  initialRegion ← ∅;
2  prunedGraph ← CDCGraph.setPruning(Threshold_DC, Threshold_CD);
3  topLink ← candidateList.next();
4  while prunedGraph.hasNoNeighbors(topLink.class) do
5      initialRegion.add(topLink.class);
6      topLink ← candidateList.next();
7  initialRegion.add(topLink.class);
8  initialRegion.topIRValue ← topLink.IRValue;
9  reachedClasses ← ∅;
10 reachedClasses.add(prunedGraph.getTransitiveCallers(topLink.class));
11 reachedClasses.add(prunedGraph.getTransitiveCallees(topLink.class));
12 reachedClasses.add(prunedGraph.getNeighborsByData(topLink.class));
13 foreach linkincandidateList do
14     if reachedClasses.contained(link.class) then
15         link.IRValue ← initialRegion.topIRValue;
16         initialRegion.add(link.class);
17 candidateList.reorderByIRValue();
```

此外对于域外类，可能存在域内类到域外类的调用或者被调用关系，或者域外类和域内某些类存在共享数据。通过算法 3改变域外相关类与需求的相似度。

Algorithm 3: Re-rank Links outside Initial Region

```
1  topIRValue ← initialRegion.topIRValue;
2  foreach (linkincandidateList) do
3      if !initialRegion.contains(link.class) then
4          foreach cininitialRegion do
5              pathList ← findValidPaths(link.class,c);
6              gMean ← 0;
7              foreach pathinpathList do
8                  gMean ← max(GeometricMean(Closeness_DC(path)),gMean);
9              link.IRValue ← link.IRValue + gMean ×
               (topIRValue − link.IRValue);
10             if hasDataDependencies(c, link.IRValue) then
11                 link.IRValue ← link.IRValue + (topIRValue-link.IRValue) ×
                   Closeness_CD(c, link.class) ;
12             if link.IRValue > topIRValue then
13                 link.IRValue ← topIRValue;
14 candidateList.reorderByIRValue();
```

2.5　结合用户反馈和代码结构信息

[15] 认为对于使用代码结构信息来调整候选追踪列表的方法 [11]，其最好的效果和信息检索方法得到的候选追踪线索排序表有很大关系。 [11]当某个文档和查询语句相似度比较大时，此时增大与文档存在结构依赖的其他文档与查询语句的相似度。但是在文档文本质量比较差的情况下。如果与查询语句相似度比较大的文档本身和查询语句就没有相关性，此时利用代码依赖就会将错误扩大，对信息检索方法得到的候选追踪线索排序表造成污染。 [15]提出，每次将排序表顶端的追踪线索交由用户（软件工程师）判断，只要用户判断给追踪线索有效的时候才提高与该追踪线索具有代码依赖的其它追踪线索的相似度值。与 [13, 14, 29] 不同， [15]不会改变词项的权重，因此不需要每次重新计算相似度。该方法相似步骤见算法 4

其中δ为相似度奖励系数，用来调节相关追踪线索相似度的变化程度。对于不同的追踪列表，该变量应该取不同的值，考虑到当候选列表的值分布比较集中时，该值稍微大一点就会使得相关候选追踪线索相似度值调整之后的排序会

Algorithm 4: User-Driven Combination of Structural and Textual Information —UD-CSTI

```
while not (stopping criterion) do
    Get the link (s, c_j) on top of List
    The user classifies (s, c_j)
    if s, c_j is correct then
        forall the c_t ∈ C do
            if (c_j, c_t) ∈ E then
                Sim(s, c_t) ← Sim(s, c_t) + δ × Sim(s, c_t)
    Reorder List
    Hide links already classified
```

发生很大的变化；当候选列表的值分布比较发散是，该值过小可能使得相关线索虽然小相似度值增大，但是对其再排序表的位置确影响不大。该方法提出自适应的δ。

$$\delta = median\{v_i, ..., v_n\} \tag{2.10}$$

其中$v_i = (max_i - min_i)/2$，v_i与第i个查询文档对应目标文档的相似度最大值和最小值。

2.6　需求追踪数据集（RTM）构造

[40]需求追踪领域面临的一个挑战是，当研究者想验证自己方法有效性时，需要搭建基线方法的实验运行环境，这个过程会耗费大量精力，并且有些可追踪数据集往往不公开。 [41]认为用于方法验证的可追踪性数据集比较少，研究者为了验证自己方法有效性往往需要自己做数据集，这个过程很耗时间。[41]公布了一个针对一个灌溉系统的数据集，包括了改系统各种软件制品之间的追踪关系。同时通过用同样的方法对该数据集和领域内常用的数据集进行处理，比较它们precision/recall图像的相似性，验证该数据集的有效性。 [42]提出一个概率模型，以帮助用户建立可靠的数据集并公开了他们采用这种方法构造的JDK1.5的数据集。

上述方法均为纯人工构造数据集的方法，敏捷开发领域，项目往往采用一些任务管理工具（issue tracker）来管理需求。用代码版本控制工具（例如CVS、SVN、GIT）来管理代码，图 2.1 主要由两部分组成：需求管理系统

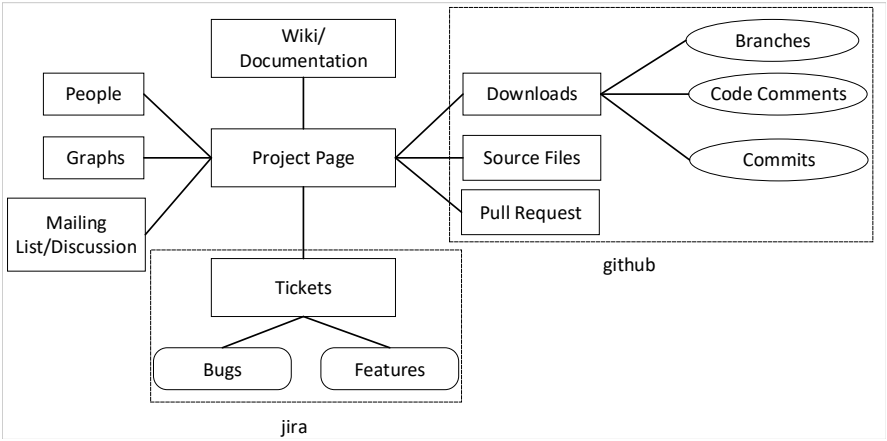

图 2.1: 开源软件结构图

（这里以jira为例），代码版本控制系统（这里以github为例）。用户在需求管理系统提交软件相关的bug或者新的需求（new feature）。研发人员会对用户所提交的issue进行评审以决定是拒绝接受该issue还是完成该issue。如果是后者接下来用户会完成与issue相关的代码（可能是完成某个需求也可能是修复某个bug）。然后用户会将更改的代码提交到github，并在commit 信息中表述该新增或修改代码的功能。很自然的这里issue 和提交的代码就形成了追踪关系。 [43]对需求管理系统中的new feature文本进行了分类，使得我们对new feature能有更好的利用。

2.7　本章小结

本章介绍了过时需求自动检测的相关技术，并分析了现有方法的不足。同时，本章介绍了在需求可追踪性、概念定位等相关领域，已有工作如何结合代码依赖关系与信息检索以提高原有方法的精度。

第三章　结合代码紧密度分析和用户反馈的软件可追踪生成方法

3.1　引言

在软件开发的过程中，会生成很多软件制品，这些软件制品之间的追踪线索是一项宝贵的资源。例如需求制品和源代码制品之间的追踪线索能够提供高层软件实体需求和底层软件实体源代码之间的追踪关系，进而帮助维护人员理解程序，并作为讨论功能更改的基础。然而人工的获取软件制品之间的追踪线索需要耗费大量时间和精力并且容易出错。软件工程师需要阅读并理解不同的软件制品，才能判断不同的软件实体之间是否存在追踪关系。为了解决这个问题领域内出现了一些半自动化的方法生成软件可追踪数据。

目前，领域内需求到代码元素追踪线索生成的主流方法是信息检索方法[31]，该方法通过计算需求制品和代码制品之间的文本相似度并按照相似度值自大到小排序，进而生成需求到代码元素的候选追踪列表，用户自上而下扫描候选追踪列表并判断候选追踪线索的有效性。由于不同软件制品之间有可能采用不同的词库，此种情况下会出现大量单词失配（Vocabulary Mismatch）问题严重影响生成候选追踪列表的精度。为了解决这个问题，一方面研究人员从不同的角度成功地提出了增强策略，比如更先进的词法分析 [20, 21] 和代码依赖性分析 [11, 12];另一方面，研究者认为用户对候选链接的反馈结果可以被用来提高候选列表精度。最近几年有大量的工作 [13–15] 使用用户反馈信息来提高信息索引方法的精度。 [13]要求用户迭代式判断候选线索有效性，并根据用户反馈修改查询语句中单词的权重。后续工作 [14]证明该方法对信息检索方法提升的精度很有限。 [15]提出结合代码依赖和用户反馈信息提高基于信息检索方法（IR）的精度，但是，要实现此方法的最佳性能，用户必须验证所有候选链接。这在实践中是不可行的。

为了解决上述问题，我们在信息检索方法的基础上，结合了代码依赖紧密度分析和用户反馈信息。紧密度是对类之间的直接（方法调用，类使用和继承）和间接（类之间的数据共享）代码依赖紧密程度的量化。通过设定紧密度阈值我们可以将功能相关的类划分到同一个代码域中，即每个代码域代表系统功能的一个方面。然后，我们的方法使用IR技术来生成需求和源代码（类）之间的候选追踪线索排序表，然后根据代码域和用户反馈对候选追踪线索排序表分两步进行优化：（1）我们要求用户只验证每个代码域中与给定需求相似度值最大

的类对应候选线索有效性。（2）我们根据用户对候选追踪线索的验证结果调整该链接中代码元素所在域的其它类对应候选线索相似度值，以及相关域外类对应候选线索相似度值。最终所有候选线索都会根据信息检索技术、代码依赖紧密度分析和用户反馈进行重排序。

本章中，我们提出一种结合用户反馈和代码依赖的软件可追踪生成方法，用户只需判断少量候选线索的有效性，明显提高了候选追踪线索列表的精度。

3.2 背景

在本节，我们将说明本工作的研究问题以及研究动机，并给出我们的问题定义。

3.2.1 研究问题

如图 3.1，在软件开发过程中会衍生出大量的软件制品，例如需求文档、设计文档、测试集合和代码等。不同软件制品之间的追踪关系有利于软件后期的维护和演化。其中需求到代码的可追踪性是其中最具有代表性也最具有挑战性的一类软件追踪性。其原因在于以下两点：（1）需求是软件系统中唯一记录了人对于完整系统功能的理解与期望的软件制品 [44]，而代码则是当前软件系统运行时行为的唯一真实反映，因此两者之间的关联关系是软件开发与维护人员最关注的；（2）需求到代码之间需要经过漫长的开发流程以完成语义上的转换。因此相对于任意其它两类软件制品之间的关联关系，需求和代码之间存在的语义偏差是最严重的。

需求到代码可追踪性建立的最大困难在于，当需求和代码元素比较多时，人工的建立可追踪性会耗费大量的时间和精力，当前基于信息检索的自动化技术存在精度低的问题。我们需要研究如下两个问题：

1. 如何尽可能减少用户参与。
2. 如何提高生成追踪线索的精度，当生成追踪线索精度比较低时，往往会使用户对工具失去信息。影响自动化工具可用性。

3.2.2 研究现状

目前，领域内主要关注于需求到代码可追踪性生成问题，主流方法基于信息检索以计算代码文本和需求文本的相似度，我们将在本小节对已有工作做简要阐述。

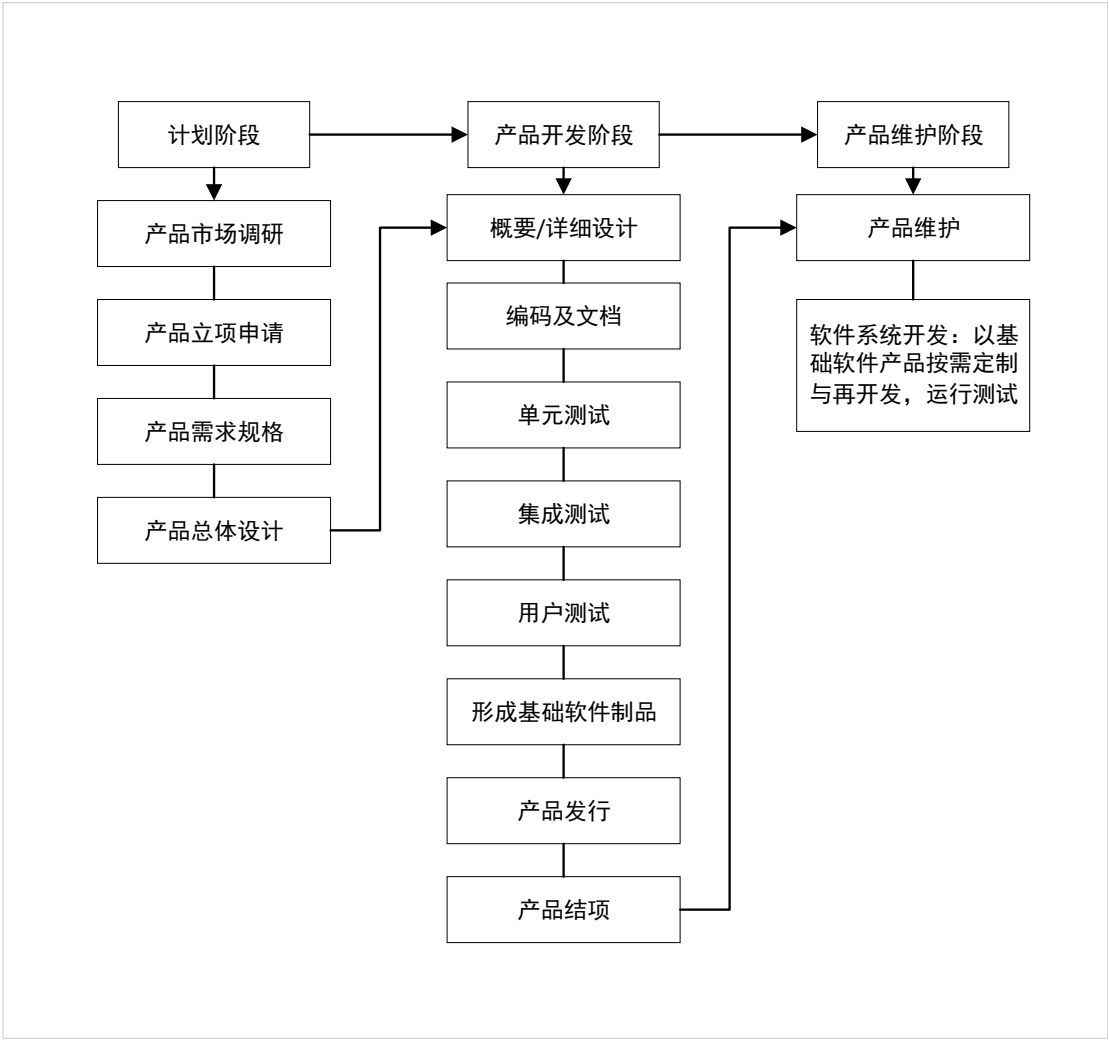

图 3.1: 软件开发流程

当前领域内最流行的需求到代码可追踪生成方法是信息检索方法，该方法通过计算需求和代码之间的文本相似度生成一个候选追踪线索排序表。该方法认为相似度越大的追踪线索越可能表明两者描述同一概念，即两者存在实现关系。需求文本和代码文本的规范化处理与相似度计算，是全自动的（候选追踪线索的确认依然需要人工参与），所以信息检索方法具有人工参与少、自动化程度高、易于实现并使用等特点。然而“词汇表失配”（vocabulary mismatch）问题 [21]极大影响了信息检索方法的准确性，这是因为不同的软件制品可能采用不同的术语表，进而可能出现不同的关键词（例：way与road都表示”道路“）表示同一个概念,或用同一个关键词表示不同的概念（例：time既指”时间“又指“次数”）的现象。针对这个问题，当前主流的研究思路是从文本预处理 [45]、信息检索模型 [19]等词法分析的角度出发来优化信息检索方法。同时近

表 3.1: 需求和类追踪关系

class	req
UpdateCodesListAction	UC15
UpdateNDCodesListAction	UC15
DrugCodesDAO	UC15
editDrugCodes_jsp	UC15
editNDCodes_jsp	UC15
DCBeanValidator	UC15
NDCodesDAO	UC15

年来有大量工作 [13–15] 利用用户对候选追踪线索的判断信息对候选列表进行优化。Panichella et al. 提出一种结合用户反馈和代码依赖的可追踪生成方法，但是为了达到该方法的最佳效果需要用户自顶向下判断整个候选列表。

3.2.3 研究动机

在本小节，我们以医疗管理系统iTrust为例，阐述我们的研究动机。

3.2.3.1 结合代码依赖紧密度分析和用户反馈的软件可追踪生成方法示例

iTrust是一个医疗管理系统，采用java语言。该系统包括需求和代码等制品。如前所述，鉴于需求到代码可追踪数据的重要意义。我们需要建立需求与实现需求之间的追踪关系。如表3.1所示为需求与实现该需求源代码之间的关联关系。以UC15为例，内容如图 3.2：

表 3.2为基于IR方法得到的UC15和代码元素对应追踪线索列表的子表，该表按照相似度值递减的顺序排序， *Is Trace*项的‘x’表示两者存在追踪关系，观察排序表的结果知有些有效的追踪线索（例如UC15 和NDCodesDAO）文本相似度值并不高，这是NDCodesDAO是一个与数据库操作相关类，用于维护数据库中的 *national drug code*(美国使用的药品标识符),以 *DAO*结尾的类往往与数据库操作有关，但是该单词并未出现在UC15文本中，这就导致两者文本相似度不高。实际上NDC是 *national drug code*的缩写，NDCodesDAO和UC15有追踪关系。图 3.3为表中涉及几个类对应的代码依赖图，带箭头的直线为直接依赖，虚线为数据依赖。从图中可以看出类

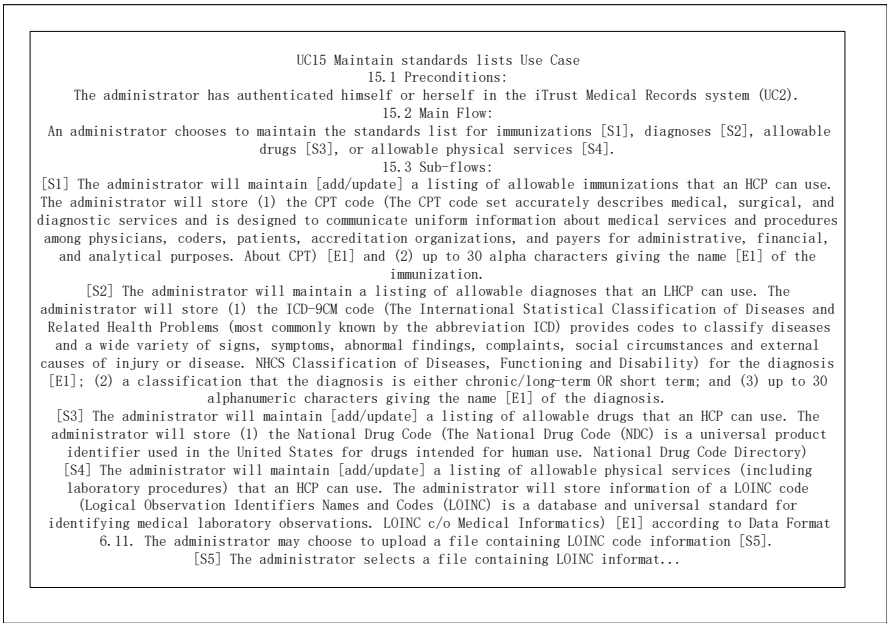
UC15 Maintain standards lists Use Case
15.1 Preconditions:
The administrator has authenticated himself or herself in the iTrust Medical Records system (UC2).
15.2 Main Flow:
An administrator chooses to maintain the standards list for immunizations [S1], diagnoses [S2], allowable drugs [S3], or allowable physical services [S4].
15.3 Sub-flows:
[S1] The administrator will maintain [add/update] a listing of allowable immunizations that an HCP can use. The administrator will store (1) the CPT code (The CPT code set accurately describes medical, surgical, and diagnostic services and is designed to communicate uniform information about medical services and procedures among physicians, coders, patients, accreditation organizations, and payers for administrative, financial, and analytical purposes. About CPT) [E1] and (2) up to 30 alpha characters giving the name [E1] of the immunization.
[S2] The administrator will maintain a listing of allowable diagnoses that an LHCP can use. The administrator will store (1) the ICD-9CM code (The International Statistical Classification of Diseases and Related Health Problems (most commonly known by the abbreviation ICD) provides codes to classify diseases and a wide variety of signs, symptoms, abnormal findings, complaints, social circumstances and external causes of injury or disease. NHCS Classification of Diseases, Functioning and Disability) for the diagnosis [E1]; (2) a classification that the diagnosis is either chronic/long-term OR short term; and (3) up to 30 alphanumeric characters giving the name [E1] of the diagnosis.
[S3] The administrator will maintain [add/update] a listing of allowable drugs that an HCP can use. The administrator will store (1) the National Drug Code (The National Drug Code (NDC) is a universal product identifier used in the United States for drugs intended for human use. National Drug Code Directory)
[S4] The administrator will maintain [add/update] a listing of allowable physical services (including laboratory procedures) that an HCP can use. The administrator will store information of a LOINC code (Logical Observation Identifiers Names and Codes (LOINC) is a database and universal standard for identifying medical laboratory observations. LOINC c/o Medical Informatics) [E1] according to Data Format 6.11. The administrator may choose to upload a file containing LOINC code information [S5].
[S5] The administrator selects a file containing LOINC informat...

图 3.2: 需求文本UC15的内容

类NDCodesDAO和类UpdateCodesListAction存在数据依赖和调用依赖，与editNDCodes_jsp存在调用依赖。在我们的方法中用户判断完这两个类与UC15相关性之后会提升NDCodesDAO与UC15的相似度值从而弥补单词失配带来的消极影响。

3.2.4　问题定义

问题定义. 给定一个需求集合R，该需求集合包含 d 个需求文本，同时给定代码集合（按粒度可分为包、类、函数，后两者为可追踪性中常用的代码粒度）C，则需求到代码的可追踪性就是在$R*C$上判定任意两个r和c之

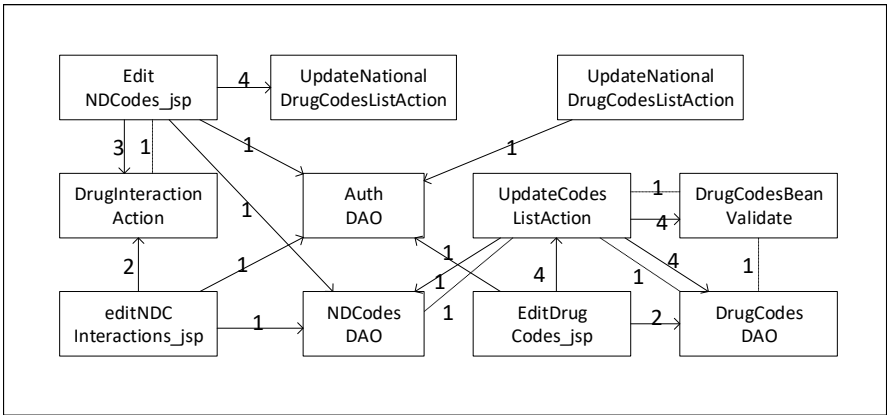

图 3.3: 代码依赖图

表 3.2: IR候选追踪线索排序表

class	req	score	isTrace
UpdateCodesListAction	UC15	0.3524	X
UpdateNDCodesListAction	UC15	0.3124	X
DrugCodesDAO	UC15	0.2418	X
editDrugCodes_jsp	UC15	0.2112	X
editNDCInteractions_jsp	UC15	0.1816	
editNDCodes_jsp	UC15	0.1238	X
DCBeanValidator	UC15	0.1045	X
DrugInteractionAction	UC15	0.0953	
AuthDAO	UC15	0.0682	
NDCodesDAO	UC15	0.0487	X
viewResult_jsp	UC15	0.0031	

间是否存在功能上相互关联。对于一个给定的需求r和给定的代码元素c，我们用布尔函数$isTrace(r,c)$来表示两者之间的关联关系，若c在代码中实现了（部分）r的功能，则$isTrace(r,c)$='trace'(相关)。否则$isTrace$='no-trace'(不相关)。由r、c、$isTrace(r,c)$三者组成的一个三元组t就代表了r和c之间的一条追踪线索。则R到C的可追踪性就是生成和维护两者之间的追踪线索集合$T = \{t = \{r, c, isTrace(r,c) || r \in R \cap c \in C\}$。问题是：

相对于传统方法，结合代码依赖紧密度分析和用户反馈生成的需求到代码的追踪数据精度是否有明显提升？

3.3 方法概述

在本节，我们将介绍结合用户反馈和代码依赖紧密度分析的软件可追踪生成方法。我们把软件开发过程中产生的需求制品和代码制品作为方法的输入，将候选追踪线索排序表作为方法的输出。方法包含四个步骤（1）获得目标软件存在的代码依赖；（2）计算代码依赖紧密度，设置紧密度阈值生成候选代码依赖域；（3）基于信息检索方法，产生需求到代码的初始候选列表。（4）根据用户反馈和代码依赖域优化调整初始候选列表。

如图 3.4 所示，在步骤一中，基于jvmti构造代码依赖捕获工具，运行目标系统并插桩代码依赖捕获工具得到运行时刻目标系统的函数调用和数据访问等数据，并由此建立方法之间的调用依赖和数据依赖关系。将方法之间的调用依

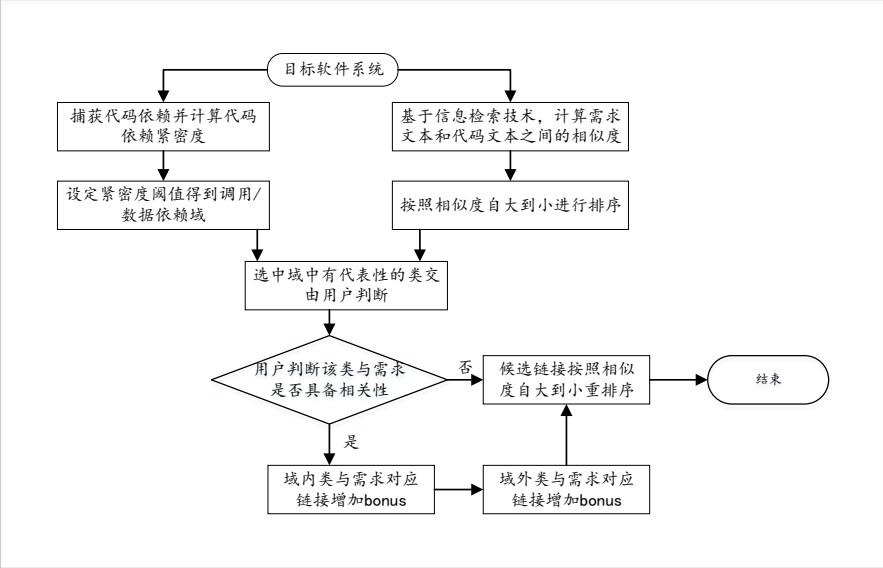

图 3.4: 基于用户反馈和代码依赖紧密度分析的软件可追踪生成方法流程

赖和数据依赖转换成类之间的调用依赖和数据依赖；在步骤二中，计算调用依赖和数据依赖紧密度，设置紧密度阈值，当两个类之间代码依赖紧密度大于阈值时两者在同一个代码域中，同一个域中的类功能比较紧密；在步骤三中，基于信息检索方法，计算代码文本和需求文本之间的相似度，并按照文本相似度从大到小的顺序生成候选追踪线索排序表；在步骤四中，对未判断的代码域，根据个域内类与需求相似度最大值，对未判断代码域按照相似度值自大到小的顺序重排序。对于排名第一的代码域，取其域内域需求相似度最大的类交由用户判断。如果该类与需求具备相关性，则调整域内类和相关域外类与需求的相似度值。

3.4　代码依赖关系捕获

3.4.1　类之间的代码依赖关系

我们考虑类之间的四种依赖关系，类的调用依赖、类继承、类使用和类的数据依赖。C_a到C_b存在调用依赖说明至少存在一个C_a 的方法调用了C_b的方法。C_a使用C_b表名，C_b是C_a的一个成员变量。两个类存在数据依赖表明两个类中存在访问同一个数据的方法，本文只关注在内存中的数据。与前三种代码依赖相比类之间的数据依赖捕获难度更大。然而 [12, 39]表明对于可追踪性数据生成任务，数据依赖与前三种代码依赖互补，相辅相成共同提高软件可追踪生成方法的精度。

3.4.2　获取代码依赖

我们使用代码依赖捕获工具 [39]来捕获上述提到的代码依赖。该工具基于jvmti接口在运行时刻获得目标系统中的函数调用和数据访问信息。我们基于如下原因选取该工具

1. 虽然能够捕获函数调用的动态分析工具很多（例如著名的The Eclipse Test & Performance Tools Platform，TPTP；或Java Plug-in Framework，JPF），但是这些工具都不能捕获函数之间的数据共享。

2. 基于java虚拟机的运行时检测，这个工具可以捕获实际执行的代码依赖关系并正确处理多态。由于该工具是在系统运行时刻对系统实际行为的观察与记录，因此不会产生非正确的代码依赖关系。

3. 该工具可以在一次系统运行中同时获得以上四种代码依赖，由于所执行测试集合的不完备性，动态分析无法保证捕获的代码依赖关系的完整性，虽然这一问题无法避免，但是我们认为部分确实的代码依赖关系在其缺失程度大致相同的情况下，并不会影响最终的实验结果，其原因在于我们用一个整合的工具在运行相同测试集合的时候同时捕获函数调用和数据依赖。

根据捕获的方法级代码依赖，我们可以得出对应的四种类级别代码依赖

1. 类之间调用依赖关系从方法调用依赖关系中抽象出来，调用边的权值为发生不同方法调用的次数。

2. 类之间数据依赖关系从方法数据依赖关系中抽象出来，数据依赖边的权值为两个类之间共享数据类型的个数。

3. 类之间使用关系从方法之间数据依赖关系中抽象出来。

4. 类之间继承关系从方法调用中得到，子类在调用自己的构造方法之前会去调用其父类的构造方法。

3.4.3　组织代码依赖关系

参考典型的基于信息检索使用代码依赖的方法 [12, 15]，我们把类之间的调用、继承和使用合并为一种类型的代码依赖即直接代码依赖，把类之间的数据共享看作另外一种代码依赖即数据依赖。前三种代码依赖归为同一种类型是因为它们结构类型，有明显的方向性，类之间的数据依赖与之

不同，表示两个类共享某种或某几种数据类型，没有方向的概念。同时，类的调用依赖、类之间的继承和使用在很大程度上是重叠的。我们采用两种不同的方法计算这两种代码依赖（直接代码依赖和间接代码依赖）的紧密度。如图 3.3，数据依赖由没有箭头的虚线表示，虚线上的值为虚线连接的两个类之间共享数据类型的数量；带有箭头的直线为直接代码依赖，箭头的方向代表调用发生的方向，直线上的值代表两个类之间发生的调用次数。例如：UpdateCodesListAction 和DrugCodesDAO 之间存在直接代码依赖，它包括两个方法调用和两个类使用。DrugCodesDAO 和DCBeanValidator 存在数据依赖，在运行UpdateCodesListAction的构造函数期间，DrugCodesDAO生成一个DrugCodesBean对象，并将其传递给DCValidator的 *validate*方法。因为以上三个类共享数据类型DrugCodesBean。

3.5　计算代码依赖紧密度并生成代码依赖域

在本步骤中，我们将按照 [12]中的方法计算直接代码依赖和间接代码依赖的紧密度。

3.5.1　计算直接代码依赖紧密度

对于类source和类sink之间的直接代码依赖，直觉上，如果两个类发生的方法调用或类使用次数比较多，则这两个类的关系比较紧密。另一个需要考虑的因素是source调用了多少个类，sink被多少个类调用过即source的出度（out-degree）和sink的入度（in-degree）。sink的入度较小意味着类sink更专注于为类source提供服务，而类source的出度较小意味着类source 更加依赖于类sink所提供的服务。直接代码依赖计算公式如下：

$$Closeness_e = \frac{2N}{WeightOutDegree_{e.source} + WeightInDegree_{e.target}} \tag{3.1}$$

其中 N代表不同的函数调用与类引用的数量。在统计 *source*的出度和 *sink*的入度时，我们将每条直接调用依赖关系上的函数调用与类引用的数量作为出入度的权值。$WeightOutDegree_{e.source}$和$WeightInDegree_{e.sink}$ 分别表示两个加权的度量。$Closeness_{DC}$的取值范围是0到1的闭区间。例如如图 3.3，editDrugCodes_jsp与UpdateCodesListActio间的直接代码依赖紧密度为2*4/（（1+4+2）+4）=0.62，而editDrugCodes_jsp和AuthDAO之间的直接代码依赖紧密度为2*1/（（1+4+2）+（1+1+1+1））=0.18，这表明与AuthDAO相比，editDrugCodes_jsp与UpdateCodesListAction 关系更紧密。

表 3.3: 共享数据类型

Data Type	Occurrences	idtf Value
DrugCodesBean	12	2.6013
NDCodesDAO	143	1.5252
Java.lang.String	1118	0.6321

3.5.2　计算数据依赖紧密度

当两个类访问同一个对象时意味着这两个类存在数据依赖，如表 3.3，展示了图 3.3 中的共享数据类型，其中“Occurrences”列代表一个数据类型在iTrust系统的所有类数据依赖关系中出现的次数。

观察表 3.3可知，String类型的出现次数远大于NDCode 和DrugCodesBean。由于 *String*类型普遍存在于各代码依赖中，所以当两个类共享数据类型*String*时，不能说明这两个类关系比较密切。因此在计算类之间数据依赖紧密度时要考虑所共享数据类型的重要程度。我们引入逆向数据类型频率（idtf）对数据类型重要性进行量化，公式如下：

$$idtf = log\frac{N}{n_{dt}} \tag{3.2}$$

其中N代表所有类数据依赖的总数，而n_{dt}代表一个给定数据类型在所有数据依赖关系中出现的次数。表中三个数据类型的idtf值也同样显示在该表中（iTrust系统中N的值为4792）。

根据*idtf*值，我们设定*idtf*的阈值$Threshold_{idtf}$来过滤*idtf*值小于阈值的数据类型。如果两个类之间共享数据类型的*idtf*值均小于阈值，则忽略两者存在的数据依赖。对这些数据类型以及相应的数据依赖关系的忽略是必要的，否则，即使这些数据依赖关系的紧密度值较小，但由于其能够将大量无关的类关联起来，因此这些数据依赖仍然会对我们的算法产生不利的影响 [12]。如果将$Threshold_{idtf}$设置为1.4.则图中*editNDCodes_jsp*和*DrugInteractionAction*之间的数据依赖（仅基于*String*）就会被忽略。

与信息检索技术中的idf概念 [31]类似，idtf值实际上反映的是一个数据类型在整个代码的全局范围内被共享的程度。idtf 值越高意味着该数据类型被两个类所独占的可能性就越大，也就意味着两个类之间更加紧密的交互。此外，对于类C_i和C_j 之间的一条数据依赖，这条数据依赖中所有数据类型的数量与C_i和C_j所共享的全部数据类型的数量（各自通过其它数据依赖）之间的比值也能够反映这两个类在“本地”的数据共享程度，这个比值越高意味着交互越

紧密。

由此，我们为类数据依赖定义紧密度$Closeness_{CD}$如下：

$$Closeness_{CD} = \frac{\sum_{x\in\{DT_i\cap DT_j\}} idtf(x)}{\sum_{y\in\{DT_i\cup DT_j\}} idtf(y)} \tag{3.3}$$

其中，$idtf(x)$表示数据类型x的$idtf$值，DT_i与DT_j分别表示两个类所有数据依赖边的数据类型。$Closeness_{CD}$的取值范围是0到1之间。如图 3.3的类UpdateCodeListAction和类DrugCodesDAO共享数据类型DrugCodesBean，其紧密度为2.60/（2.60+1.53）=0.63，而类UpdateCodeListAction和类NDCodesDAO共享数据类型NDCode，两者之间紧密度为1.53/（1.53+2.60）=0.37。结果显示相比于类NDCodesDAO，类UpdateCodeListAction与类DrugCodesDAO的关系更紧密。图 3.5为图 3.3对应的代码依赖紧密度图。

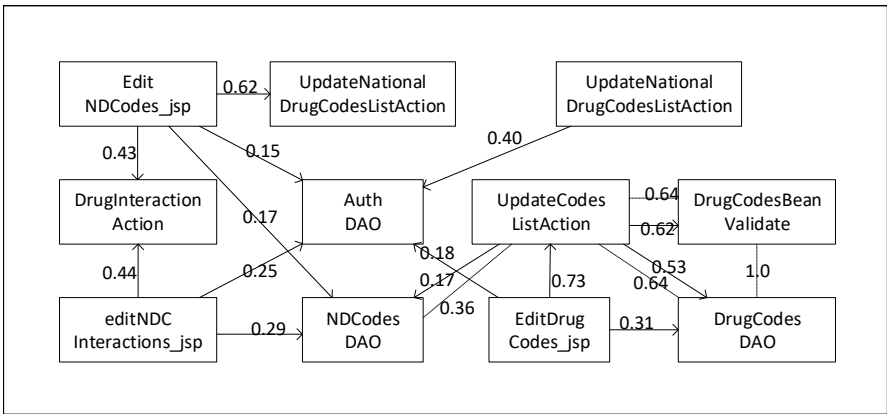

图 3.5: 代码依赖紧密度图

3.5.3　建立CDCGraph

我们定义图$CDCGraph$为一个有序对$G =< V, E >$，其中V是代码中类的集合，并以其类名来标识。进一步的，$CDCGraph$的顶点之间有两种类型的边。在两个顶点之间，E_{CD}代表一条直接调用边而E_{DC}代表一条数据依赖边。$CDCGraph$同时在每条代码依赖关系上标注其计算得到的紧密度值。从图建立的CDCGraph如图 3.3所示。

3.6　基于信息检索方法生成候选追踪线索列表

我们利用信息检索技术在需求和类之间生成候选线索列表，这部分共包含四个步骤：

1. 创建文档库：对于代码中的每个类，我们抽取一个包含了这个类的注释、类名、方法名以及类成员名的文档。对于需求制品中的每条需求，我们抽取一个包含了其题目与内容的文档（对于有结构的需求用例我们抽取其前置条件、主要流程以及分支流程，对于无结构的需求我们直接引入所有文本信息）。

2. 标准化文档库：所有类和需求的文档都以信息检索领域内通用的标准化手段进行预处理，包括标识符拆分、骆驼分词、停用词过滤、词形还原和词根获取。

3. 索引文档库与文本相似度计算：我们使用tf-idf [31]对文档库中的每个关键词进行索引，并使用三个主流的信息检索模型来计算文档之间的相似度，即空间向量模型（VSM） [3]，隐式语义检索（LSI） [19]，以及基于概率的Jensen-Shannon模型（JS） [24]。

4. 建立候选线索列表：在生成的候选追踪线索列表中，我们按照每条候选追踪线索相似度值自大到小顺序排序。

表 **??**展示了一部分在iTrust系统上生成的候选追踪线索列表。该列表通过VSM模型和需求UC15（标题为 *Maintain standards lists*）与图中十一个类建立了候选追踪线索，该列表由相似度值降序排列。”Is Trace“列中的”X“表示该类与UC15之间确实存在功能上的关联关系。

3.7　结合紧密度分析和用户反馈对候选列表重排序

接下来对于信息检索方法产生的候选追踪线索排序表，我们通过结合用户反馈信息和代码依赖紧密度分析来优化排序。 [46]表明一个给定需求一般在代码的某块联通域内被实现，而不是随机散步在代码各处，这也是我们前面设置代码依赖紧密度阈值生成代码域的依据。我们提出一个包含四个步骤的候选追踪线索重排序算法。第一，对于未代码域，根据各域内类与需求相似度最大值，对未判断代码域按照相似度值自大到小排序。对于排名第一的代码域，取其域内与需求相似度最大的类交由用户判断，如果该类与需求具备相关性则执行步骤二，否则直接跳至步骤四。第二，调整域内类对应候选线索相似度值。第三，调整域外类对应候选线索相似度值。第四，按照相似度值从大到小的顺序对所有候选线索重排序。由用户决定是否需要继续判断候选代码域，如果用户继续判断则重复步骤一，否则算法流程结束。

3.8　结合紧密度分析和用户反馈对候选列表重排序

我们的增强策略受到以下三个发现的启发，（1）候选列表中相似度最大的候选线索往往是有效的 [20]。一个给定需求一般在代码的某块联通域内被实现，而不是随机散步在代码各处，这也是我们前面设置代码依赖紧密度阈值生成代码域的依据 [46]。（3）通过事先验证候选追踪线索得到的用户反馈和代码紧密度分析可以改善基于信息检索的软件可追踪生成方法的精度 [15]。因此，首先我们通过设置代码依赖紧密度阈值，在代码依赖图中去掉紧密度值小于紧密度阈值的边，形成一些联通子图，我们称这些联通子图为候选代码域。基于这些代码域我们提出了一种包含三个步骤的迭代式候选列表重排序算法。第一，对于未代码域，根据各域内类与需求相似度最大值，对未判断代码域按照相似度值自大到小排序。对于排名第一的代码域，取其域内与需求相似度最大的类交由用户判断，如果该类与需求具备相关性则执行步骤二，否则直接跳至步骤四。第二，调整域内类对应候选线索相似度值。第三，在同一轮迭代中调整域外类对应候选线索相似度值。当用户停止判断未判断域与需求相关性时，整个算法结束。

3.8.1　建立候选代码域

前文已经提到我们的方法涉及两种代码，直接代码依赖（由方法调用、类继承和类使用组成）、数据依赖。我们为两种代码依赖分别设定阈值$Threshold_{DC}$和$Threshold_{CD}$来对代码依赖图*CDCGraph*进行裁剪。去掉小于紧密度阈值的边形成一些联通子图即代码依赖域。我们认为每个代码域中的各个类相互紧密协作完成同一个需求。对于任意一个给定需求，我们取每个代码域中与需求相似度值最大的类作为该域的代表类。对于这些代表类按照相似度值从大到小排序，用户自上而下判断其中一些代表类与需求相关性。用户通过判断代表类与需求的相关性得出其所在代码域与需求的相关性。如图 3.6，我们假设$Threshold_{DC}$ 值为0.6，$Threshold_{CD}$值为0.6，我们会得到两个代码域：代码域1：*UpdateCodesListction*、 *DrugCodesDAO*、 *editDrugCodes_jsp* 和 *DCBeanValidator*；代码域2：*editNDCodes_jsp*和 *UpdateNDCodesListAction*。根据候选代码域的排序，用户将先验证Region1的代表类然后验证Region2的代表类。

3.8.2　调整域内类对应候选线索相似度值

我们考虑两个因素给予域内非代表类对应候选追踪线索相似度奖励。（1）代码域的大小，即代码域包含类的个数。我们认为代码域中类的个数越少，类

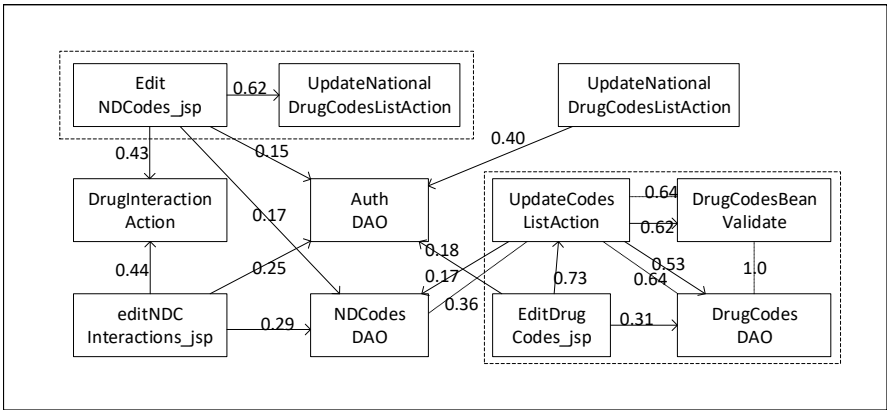

图 3.6: 候选代码域

之间的关系越紧密。（2）域内其它类与代表类C_{req}的交互紧密度程度。针对两种不同的代码依赖我们采用两种方式来量化两个类之间的交互程度

1. 直接代码依赖，我们在代码依赖图中寻找域内类C_{in} 到代表类C_{req}的路径，该路径需要满足一直调用或者一直被调用的关系。

2. 数据依赖只考虑两个类之间的直接数据依赖，不考虑传递性。

域内类对应候选线索相似度根据以上两种依赖关系可以拿到两次奖励，公式如下：

$$Bonus_{DC} = \prod_{x \in PATH} Closeness_{DC}(x) \tag{3.4}$$

公式解释：域内其它类（C_{in}）到域代表类（C_{req}）可能存在多条代码依赖路径，$PATH$为紧密度乘积最大的一条。$Closeness_{DC}(x)$ 为直接代码依赖x的紧密度值。综合考虑域大小和域内类与代表类交互程度，域内类对应候选追踪线索相似度值更新公式为：

$$IR_{in} = (IR_{current} + \frac{IR_{top}}{RegionSize})(1 + Bonus_{DC} + Closeness_{CD}(x)) \tag{3.5}$$

公式解释：$IR_{current}$是域内类C_{in}对应候选追踪线索当前的相似度值，IR_{top}为域内类对应候选追踪线索相似度最大值，即域内代表类（represent class）对应候选追踪线索相似度值。$RegionSize$为被判断代码域的大小，即代码域域内类的个数。

3.8.3　调整域外类对应候选线索相似度值

接下来我们对代码域域外的类对应候选追踪线索相似度值给予奖励，奖励的标准是这些类与域内代表类（C_{rep}）的交互程度。与前一步骤类似，我们针对不同的代码依赖关系采用不同的奖励方法。从直接代码依赖的角度出发，从一个代码域外的类C_{out}出发我们尝试找到一个通往被判断域内代表类C_{rep}的路径。与域内类到域内代表类之间的路径一样，域外类到域内代表类的一个合法路劲应该满足要求：这一路径是单向的，即C_{out}传递性到C_{rep} 或者C_{rep}传递性到C_{out}；域外类对应候选追踪线索相似度值更新公式为：

$$IR_{out} = IR_{current} + IR_{top}(Bonus_{DC} + Closeness_{CD}(x)) \tag{3.6}$$

其中$IR_{current}$为域外类C_{out}对应候选追踪线索当前的相似度值，IR_{top}所有类与给定需求相似度的最大值。在这一步骤中有可能所有域外类对应候选线索相似度都能得到奖励。我们希望给予与域代表类交互紧密的域外类更多的奖励以提升它们在列表中的排序。

3.8.4　结束判断代码域与需求相关性

当用户没有意愿继续候选代码域与需求相关性，或者已经没有未被判断的候选代码域时，此时算法流程结束，否则进入下一轮迭代。在每一轮迭代中，有些候选追踪线索的相似度值会被更新，候选追踪列表排序会发生变化。然而，为了避免给不相关类对应追踪线索相似度太大的奖励，我们要求追踪线索相似度更新之后的值不超过候选线索相似度最大值。并且我们保证我们的算法是稳定排序。算法1.描述了我们的算法。

3.9　案例分析

本小节以实验系统iTrust为例，详细阐述本文方法的执行过程。

3.9.1　实验系统介绍

iTrust是一个医疗管理软件，采用java语言编写。该系统包括uc和src目录，其中uc目录里面是一组用于描述需求的文本文件，src目录为该项目的代码文件，iTrust系统具体信息如下表所示：

Algorithm 5: recovery requirement traceability through user feedback and code dependency closeness analyse

Input: req,$List$,$Threshold_{DC}$,$Threshold_{CD}$

Output: $List$

```
1  irValue_top ← getTopRankedIRValue(List,req);
2  regions ← CDCGraph.prune(Threshold_DC,Threshold_CD);
3  while (notstoppingcriterion) do
4      Region_top ← regions.getTopRegion(list,req);
5      class_req ← topRegion.getRepresentativeClass(req);
6      The user verifies the link(req,class_rep)
7      if (req,class_rep is a relevant trace) then
8          foreach link in List do
9              bonus_DC ← getMaxBonusByDC(link.class,class_rep);
10             bonus_CD ← getBonusByCD(link.class,class_rep);
11             if Region_top contains link.class then
12                 link.value ← (link.value+irValue_top/Region_top.size()) (1+bonus_DC+bonus_CD);
13             else
14                 link.value ← link.value+irValue_top*(bonus_DC+bonus_CD);
15             link.value ← min(link.value,irValue_top);
16     Hide Region_top from regions
17     Hide verified links and reorder List
```

#	iTrust
Version	13.0
Programming Language	Java
KLoC	43
Executed classes	138
Evaluted requirements	34
Average number of classes implementing a requirement	8 (1-17)
Direct code dependencies	274
Class data dependencies	4792
Relevant Traces in RTM	255

3.9.2　为iTrust系统建立可追踪性过程

接下来结合iTrust系统，详细阐述我们算法的每一个步骤。

3.9.2.1　获得代码依赖

使用标准JDK提供的JVMTI接口，如图所示构造代码依赖捕获工具。运行iTrust系统，通过-agentpath参数将图2所示代码依赖捕获工具插桩到运行iTrust的虚拟机中。代码依赖捕获工具监控虚拟机中发生的函数进入、退出，域变量访问、修改等事件，当事件被触发时在对应回调函数中将数据访问记录保存到本地数据库中。最后通过对数据库进行分析得到函数之间的代码依赖，进而得到类之间的代码依赖。iTrust代码依赖详情见下表：

Call Relation Size	274
Data Relation Size	4792
Usage Relation Size	119
Direct Relation Size	274
Call & Data Relation Overlap	140
Usage & Data Relation Overlap	119
Direct & Data Relation Overlap	140

下表为本例子中涉及到的数据依赖

Source	Data Type	Target
NDCodesDAO	NDCode	UpdateCodesListAction
UpdateCodesListAction	LOINCBean	DrugCodesBeanValidator
UpdateCodesListAction	LOINCBean	DrugCodesDAO
DrugCodesBeanValidator	LOINCBean	DrugCodesDAO

3.9.2.2　计算代码依赖紧密度

如上方法所述，计算类之间数据依赖之前需要先计算各数据类型的*idtf*值，用例中涉及数据类型的*idtf*值如下表：N=4844

#	Data Type	occurrences	Idtf Value
1	LOINCBean	9	2.7310
2	NDCode	147	1.5179
3	Java.lang.String	1118	0.6368

根据上述方法中的紧密度公式，计算类之间的代码依赖紧密度，代码依赖紧密度图如图所示，设置直接代码依赖阈值0.6，设置数据依赖紧密度阈值0.6，紧密度高于阈值的类之间会形成代码域。如图，虚线框表示代码域，图中有两个代码域。

3.9.2.3　基于信息检索方法计算需求和代码之间的文本相似度

对需求uc和代码src中的文本，进行文本预处理。处理过程包括移除停用词，词形还原和词干提取等操作。特别的，对于从代码中抽出的文本首先需要根据骆驼命名法等变量规则进行分词。例如函数名initializeDrugCodes，首先根据驼峰规则划分为initialize、drug 和codes，然后根据进行词形还原和词干提取，分别得到init、drug 和code。对于预处理后的需求文本和代码文本，利用空间向量模型计算文本之间的相似度，下表为计算后的文本相似度从大到小排序表。最后一列isTrace表示该代码元素和需求之间是否具有相关性。X表示两者之间存在相关性。

class	req	score	isTrace
UpdateCodesListAction	UC15	0.3524	X
UpdateNDCodesListAction	UC15	0.3124	X
DrugCodesDAO	UC15	0.2418	X
editDrugCodes_jsp	UC15	0.2112	X
editNDCInteractions_jsp	UC15	0.1816	
editNDCodes_jsp	UC15	0.1238	X
DCBeanValidator	UC15	0.1045	X
DrugInteractionAction	UC15	0.0953	
AuthDAO	UC15	0.0682	
NDCodesDAO	UC15	0.0487	X
viewResult_jsp	UC15	0.0031	

3.9.2.4　对代码域进行重排序

对代码域进行排序，如图代码域，根据前一步得到的相似度值，按照每个域中的最大值自大到小进行排序。先将第一个域交由用户判断即判断UpdateCodesListAction与UC15的相关性。由表知，两者具备相关性（实际运用中由用户判断代表类和UC15相关性）。对于UpdateCodesListAction 所在的域，根据上述算法需要调整其它域内类对应候选追踪线索的相似度值。对于域外相关类，根据代码依赖图知，只有类NDCodesDAO与域内代表

类UpdateCodesListAction存在代码依赖，并且是同时存在数据依赖和直接代码依赖，因此需要对这个类对应候选追踪线索相似度值更新。同理，如果用户愿意继续判断，对由类editNDCodes_jsp和UpdateNDCodesListAction组成的代码域进行判断，并根据用户判断结果调整相关追踪线索相似度值。在例子中使用我们的算法，用户判断完两个类之后，根据判断结果都候选追踪线索重排序的结果如下表：

class	req	score	isTrace
UpdateCodesListAction	UC15	0.3524	X
editDrugCodes_jsp	UC15	0.3524	X
DCBeanValidator	UC15	0.3524	X
DrugCodesDAO	UC15	0.3524	X
editNDCodes_jsp	UC15	0.3524	X
UpdateNDCodesListAction	UC15	0.3124	X
NDCodesDAO	UC15	0.2954	X
editNDCInteractions_jsp	UC15	0.2468	
DrugInteractionAction	UC15	0.1845	
AuthDAO	UC15	0.1816	
viewResult	UC15	0.0031	

由表中可以看出，我们把与给定需求相关的类提高了排序表的前面。

3.10　本章小结

在本章中，我们提出一种基于代码依赖关系的过时需求自动检测与更新推荐方法，提高了过时需求检测时的精度，并推荐与过时需求更新相关的变更代码元素，以辅助维护人员完成对需求的更新。

第四章　数据组织及方法验证

在本章中，我们介绍了结合代码依赖紧密度分析和用户反馈软件可追踪生成方法的实验设计与分析。我们在四个实验系统（iTrust,Maven,Pig,Infinispan）下设计实验，验证了方法的有效性。同时，我们详细阐述了需求和代码依赖数据收集的过程。

4.1　数据组织

本节主要介绍不同实验系统代码依赖的获取过程和对于后两个实验系统（Pig，Maven）数据集的构造过程。

4.1.1　代码依赖捕获

本小节首先对我们使用的代码依赖捕获工具进行简要介绍，然后详细介绍不同类型的软件系统代码依赖的捕获方式。

如前文所述代码依赖捕获工具结构图如图4.1，使用标准JDK提供的JVMTI接口，监听Java虚拟机中产生的4 个JVMTI 事件，分别为类成员读取事件、类成员修改事件、函数进入事件以及函数返回事件；注册这4个事件的回调函数，在回调函数中将事件引起的函数进入、返回记录和数据访问记录保存到本地数据库中。我们通过jvm参数-agentpath将该工具插桩到运行目标测试集的虚拟机中。

对于应用软件，如实验中用到的iTrust系统。该系统能独立运行，我们运行该系统同时插桩代码依赖捕获工具，根据用户手册来运行该系统具有的各种功能从而捕获尽可能完整的代码依赖。但是对于一些支撑软件，像我们实验中的Pig，它是一个编译器本身不能独自运行。对于具备如下特点的软件系统：

1. 本身不能单独运行或者搭建其运行环境工作量比较大。

2. 具有丰富的测试集合。

我们提出通过运行其测试集合得到系统的代码依赖子集，然后合并代码依赖子集得到代码依赖集，如图 4.1 接下来我们将介绍用maven或ant构建工具管理的开源软件通过运行测试集得到代码依赖的过程。接下来我们将分别介绍实验系统Infinispan、Maven和Pig代码依赖的捕获过程。

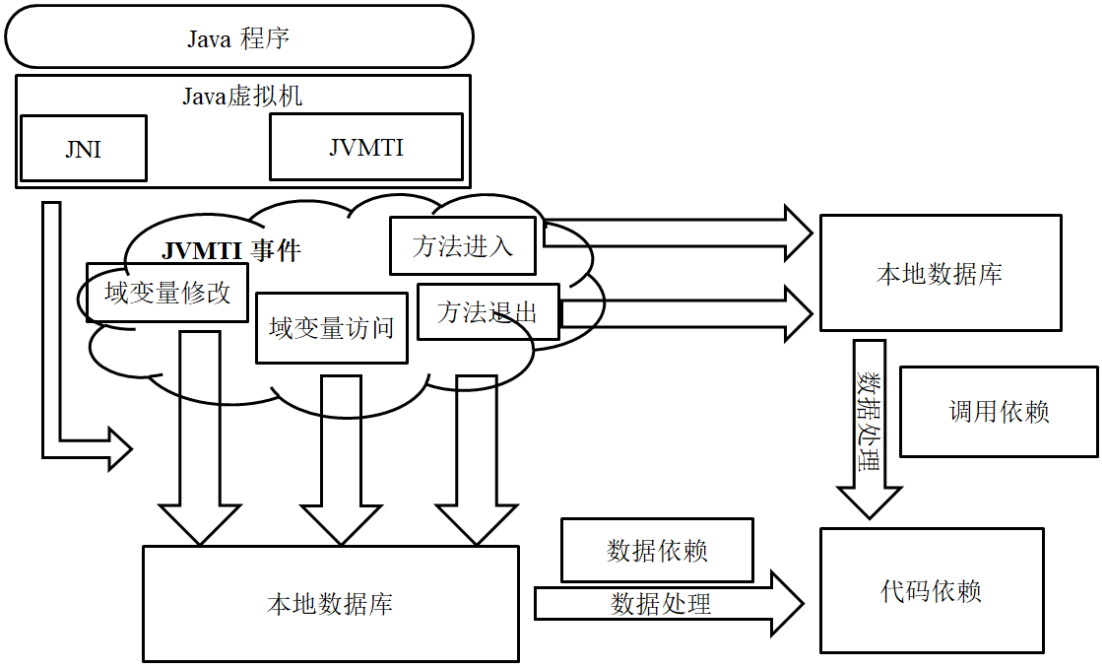

图 4.1: 代码依赖工具捕获结构图

4.1.2　Infinispan

Infinispan由构建工具maven管理，Infinispan结构如图包含源代码和测试集。简要介绍Maven：Maven是一个跨平台的项目管理工具，本身是一个Java开源项目。Maven主要服务于基于Java平台的项目构建、依赖管理和项目信息管理。Maven 帮助我们标准化构建过程，抽象了一个完整的构建生命周期模型，每个生命周期都由相应的插件来负责。本小节提到的测试阶段由surefire 插件负责。Maven本身并不是一个单元测试框架，Maven所做的只是在构建执行到特定生命周期阶段的时候，通过插件来执行JUnit或者TestNG测试用例，这一插件是maven-surefire-plugin，可以称之为测试运行器（Test Runner），它能很好的兼容JUnit3、JUnit4以及TestNG。我们通过调研发现通过maven管理的软件，其测试集由surefire插件运行。当执行*mvntest*命令时，surefire插件会运行所有以Test 结尾的类，我们将代码依赖捕获工具插桩到运行测试集的虚拟机中即可获得测试集对应的代码依赖捕获数据。需要说明的是：在Maven 的测试周期，默认情况下Maven会开启新的虚拟机进程用来运行测试集，而不是用运行Maven的虚拟机运行测试集。即我们要把工具插桩到运行测试集的虚拟机中。通过阅读maven-surefire-plugin（https://maven.apache.org/surefire/maven-surefire-plugin/）文档，我们通过配置surefire 的*argLine* 参数来完成工具插桩，xml见下表。这里我们还要着重阐述forkCount

1. forkCount 用来指定运行测试集的虚拟机进程数目，默认为1。如果该数字以C结尾，运行虚拟机的进程数目为这个数字乘以CPU内核数。如果该数字设置为0，则意味着不在启动新的虚拟机进程运行测试集，由主虚拟机进程（运行Maven的虚拟机）来执行测试集。

2. argLine jvm参数设置

基于以上内容我们有两者方法通过运行测试集捕获代码依赖：（1）配置pom文件里的maven-surefire-plugin插件，通过*argLine*将工具插桩到运行测试集的虚拟机中。（2）通过设置forkCount参数为0，用运行maven的虚拟机进程来运行测试集，此时在启动Maven时插桩代码依赖捕获工具即可。然后我们对代码依赖子图进行合并，例如我们在某两个测试用例分别得到类A调研类B，类B调用类C，由此经过合并处理我们就能得到类A调用类B调用类C这条完整的调用关系，对于数据依赖有类似的做法。本文通过这个方法获得Infinispan的代码依赖，由于同Infinispan一样，实验系统Maven也是由maven来管理，因此我们采取同样的方法获得其代码依赖。

4.1.3 Pig

Pig由构建工具Ant(Another Neat Tool)管理，最早用来构建著名的Tomcat，我们可以将Ant看成是一个Java版本的Make，Ant使用XML 定义构建脚本，相对于Makefile更加友好。Ant用junit框架来运行Pig的测试集。在运行Pig测试集的过程中为了捕获代码依赖我们需要将代码依赖捕获工具插桩到运行测试集的虚拟机中。这里介绍以下junit的两个重要参数，fork、jvmarg。

1. fork 当开启此参数（fork=”yes”）时，Ant会启动新的java虚拟机来执行测试集

2. jvmarg 当开启参数fork时，jvmarg用于向这个新开启的虚拟机传递jvm参数

基于以上内容，与被构建工具Maven管理的项目类似，我们有两种方法通过运行用Ant管理的实验系统Pig的测试集得到代码依赖：（1）配置文件build.xml中的junit，开启fork参数并通过jvmarg参数向新开启的虚拟机传递jvm参数（2）配置文件build.xml中的junit，不开启fork 参数（fork=”false”），此时在启动Ant时插桩代码依赖捕获工具即可。在我们的实验中，我们都是采用第一种方法插桩代码依赖捕获工具获取代码依赖子集，这是因为我们使用的代码依赖捕获工具是单进程的，如果采用第二个方法会出现几个进程同时向数据库写数据的同步问题。

4.1.4　RTM组织

在中已经提到，Pig、Infinispan和Maven均是我们通过对[47]的整理得出的数据集，[47]收集了这三个实验系统的Jira数据和git数据，Jira是Atlassian公司出品的项目与事务追踪工具，被广泛用于缺陷定义、客户服务、需求收集、流程审批、任务跟踪和敏捷管理等领域。如图 为实验系统Pig在Jira上的一个issue，

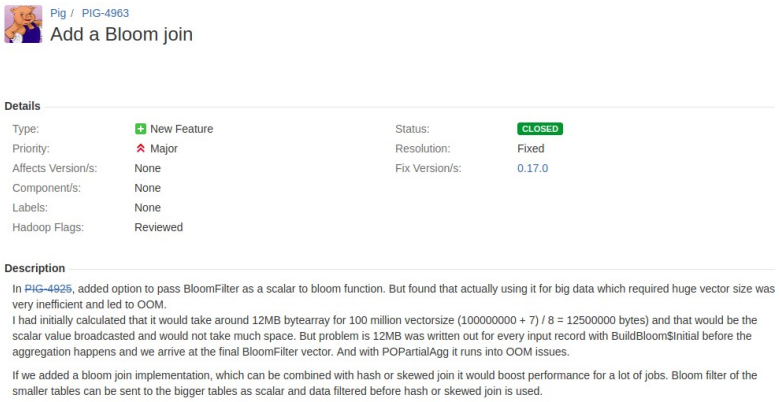

图 4.2: pig在jira上的一条issue信息

从图中可以看出改issue是要求增加新功能并在描述（description）里面说明了增加该新功能的原因。另外该issue还包含很多其他信息，例如该issue的类型、优先级、当前状态等。Git是一个开源的分布式版本控制系统，可以有效、高速的处理从很小到非常大的项目版本管理。git用于实验系统的代码管理和版本控制，记录了大量的代码提交（commit）信息。如图 为一次实验系统pig的一次commit，从标题可看出该commit是针对Pig-4963即4.2，同时也可以看到该commit引起的代码变更（我们这里只看.java结尾的文件，因为我们的对源代码的研究粒度是java类）。这里变更的文件有PigConfiguration.java、MRCompiler、PigCombiner.java等。我们能很容易的推断出需求issue_pig_4963与以上java类存在追踪关系。这也是我们为Pig、Maven和Infinispan建立RTM数据集的思路。以下为 [47]对每个项目整理的数据，该数据由8张数据表组成，如下表 4.1所示。

本文主要使用表，issue、issue_link、change_set和code_change来合成RTM数据集，如下图所示 4.4。issue表中是各种issue集合，包括bug、任务（task）、

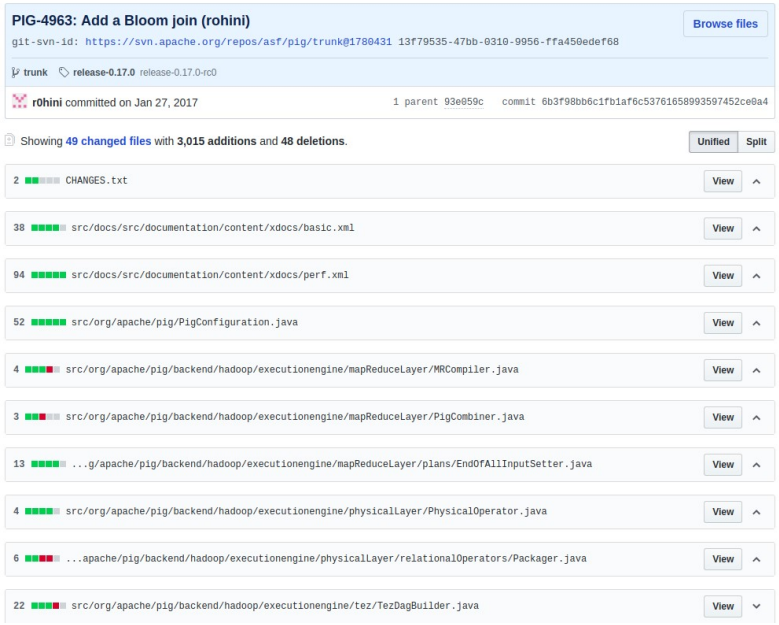

图 4.3: pig在git上针对issue的commit

需求（New Feature）等信息。我们主要使用New Feature 类型的issue，通过观察知其description字段是对这个issue的描述，我们把它看成是一个需求文本。change_set表描述的与issue对应的代码提交，而code_change描述的是一次代码提交引起的代码变更。因此通过issue、change_set 和code_change我们就可以得到从需求issue 到代码变更code_change的追踪关系。此外，issue_link描述的是需求issue之间的关系，包括重复、从属、依赖等关系。我们对于存在特定关系的issue进行合并，最终得到需求issue到代码变更所在类的追踪关系。issue筛选规则如下：

1. 忽略带有关键词testing或者testcase的issue，因为这些issue往往是和测试集有关系，不是软件系统的功能性需求。

2. issue必须已经被完成并且有与之关联的code_change，只有这样我们才能生成需求（issue）到代码的追踪关系。

3. 我们只选择优先级为“Major”和“Critical”的issue，因为我们认为这种issue是功能性需求的概率更大。

4. 我们只选择类型为“New Feature”（或Feature Request）的issue，我们认为这种需求粒度比较解决传统的需求（例如iTrust的需求），类型为bug的issue往往粒度太小，类型为task的issue又往往粒度太大。

表 4.1: 开源系统数据整理

table	description
issue	记录每个issue的具体信息，标识符、描述信息、类型、时间戳等
issue_link	记录issue之间的关系，例如从属、依赖、重复等
issue_component	记录改issue所属的模块
issue_fix_version	记录改issue被完成的版本
chang_set	一个代码提交的相关信息，代码提交者、时间戳、描述代码提交的文本
chang_set_link	大部分的代码提交都是为了完成一个issue，该表记录代码提交和issue之间的对
code_chang	一次代码提交引起的代码文本变更，例如增加或减少了哪些代码
project	项目的元数据，包含JIRA和git仓库的信息

此外，当issue之间的关系为从属（part-of）、重复（duplicate）、替代（supersede）时，我们会将几个issue进行合并。按上述方法我们通过整理得到了Maven、Pig和Infinispan的RTM数据集。需要说明的是，可能会存在这样一种情况：为了完成需求C，源代码里增加了类Func，在类Func里与用到了Origin类，但是Origin类并未发生代码变更。此时我们的方法只能得出需求C和代码Func存在追踪关系，会把代码Origin漏掉。根据 [41]与其他已有系统实验图像进行对比，发现图像类似。（未完... 证明我们RTM的有效性）

4.2　结合代码依赖紧密度分析和用户反馈的软件可追踪生成方法

在此小节，我们通过实验以验证结合代码依赖紧密度分析和用户反馈的软件可追踪方法的有效性。接下来，将具体阐述我们的实验设置及实验结果与分析。

4.2.1　实验目标与评价指标

我们的实验目的是分析结合用户反馈和代码紧密度分析能否改善自动化可追踪生成场景下的混合方法，为了达到这一目的，我们需要回答如下研究问题。RQ: 在需求到代码的可追踪性生成场景下我们的方法能否优于基线方法？为了回答这个问题，我们选择了以下三个基线方法：（1）只利用信息检索技术的纯信息检索方法（简称为*IR-ONLY*）；（2）基于信息检索方法，引入代码紧密度分析的方法（简称为*TRICE*[12]）；（3）基于信息检索方法，考虑代码结构信息和用户反馈信息的先驱方法（简称为*UD-CSTI*[15]）。我们将我们的方法命名为*CLUSTER*，在对比*CLUSTER*和三个基线方法时，我们依次使用三种不同的主

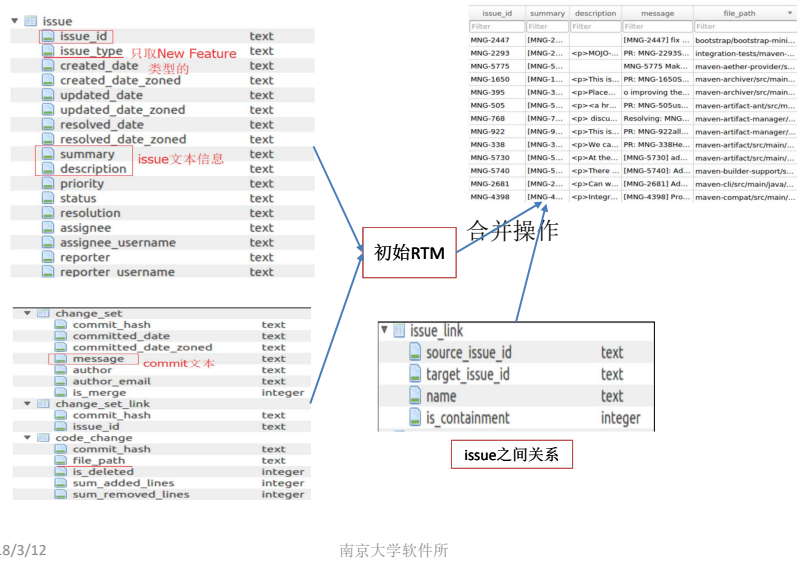

图 4.4: rtm整理过程

流模型（VSM、LSI和JS）。

RQ的评价指标：为了验证我们的*CLUSTER*方法的有效性，我们首先引入两个重要的度量，精确度（precision）和查全率（recall）。对应公式如下：

$$precision = \frac{|relevant \cap retrieved|}{|retrieved|}\% \tag{4.1}$$

$$recall = \frac{|relevant \cap retrieved|}{|relevant|}\% \tag{4.2}$$

参数解释：其中*relevant*是正确的候选追踪线索（在Golden RTM中）集合。*retrieved*表示软件可追踪生成方法所返回的追踪线索集合。由于自动化可追踪生成方法所返回的是一个排序列表，即按照文本相似度值从高到低排序。所以一种常用的比较IR 方法的方式是在不同的查全率下比较不同方法之间的精确度，可以由一条*Precision-Recall*曲线展示。为了进一步衡量不同自动化软件可追踪生成方法返回结果的整体质量，我们选用了领域内另外两个常用指标：AP（Average Precision）与MAP（Mean Average Precision）。其中*AP*用于度量全部查

询（需求）所检索相关文档的排序质量，计算公式如下：

$$AP = \frac{\sum_{r=1}^{N} (Precision(r) \times isRelevant(r))}{|RelevantDocuments|} \tag{4.3}$$

参数解释：r表示被查询实体（类）在排序表中的位置，N表示候选追踪线索的总数。$Precision(r)$表示对于前r个候选追踪线索的准确率。$isRelevant()$为一个二值函数，如果这条候选线索有效则返回1，否则返回0。此外，MAP用于描述不同查询（需求）所检索的相关文档（类）AP的平均值。计算公式如下：

$$MAP = \frac{\sum_{q=1}^{Q} AP(q)}{q} \tag{4.4}$$

参数解释：q表示一次查询而Q表示查询的总数。为了更全面的验证方法有效性，我们同时使用AP和MAP。

4.2.2　阈值设置

我们需要校准四个阈值，$Threshold_{idtf}$、$Threshold_{DC}$、$Threshold_{CD}$和LSI的k值。根据之前的[12, 39]案例分析，我们设置$Threshold_{idtf}$的值为1.4，用这个阈值来忽略普遍出现的数据类型。对于$Threshold_{DC}$和$Threshold_{CD}$的设置，我们首先使用3σ 标准分别去除$Closeness_{DC}$和$Closeness_{CD}$集合中的异常值。我们这里的异常值定义为：比集合标准差σ 高三倍或者低三倍的紧密度值。然后通过min-max标准化我们将剩余的代码依赖紧密度归一化到[0,1]区间。之前被过滤掉的高异常值此时设置为1，类似的之前被过滤的低异常值被设置为0。然后我们根据上节给出的度量指标，在[0,1] 区间选取一组能让所有实验系统在不同IR模型下（共十二种情况）综合表现最好的阈值。本文中设置$Threshold_{DC}$为0.4，$Threshold_{CD}$为0.8。在代码依赖图中，通过暂时移除紧密度小于紧密度阈值的边，剩下的一些联通子图我们称其为代码域。我们各实验系统的代码域个数如下表所示：

#	the number of code regions
iTrust	36
Maven	21
Pig	8
Infinispan	37

对于代码依赖紧密度，经过min-max标准化处理之后的值，仅用于设定阈值生成代码依赖域。对于算法5我们还是用原始的代码依赖紧密度值。对

表 4.2: 实验系统的相关细节

	iTrust	Maven	Pig	Infinispan
版本	13.0	3.5.2	0.17.0	9.2.0
编程语言	Java	Java	Java	Java
千行代码（KLoC）	43	101	365	521
代码（类）	138	94	236	388
需求（用例）	34	36	68	237
调用依赖	274	182	1998	1777
数据依赖	4792	1164	5405	6076
追踪关系	255	155	356	1515

于LSI方法中的K值，我们发现当k为90时iTrust和Maven的Precision/Recall表现最好，而对于Pig和Infinispan，k=200时Precision/Recall表现最好，从表4.2中可以看出iTrust和Maven数据集中需求的个数比较接近（94和138），Pig和Infinispan的需求比较接近（236和237）我们认为这是这两组系统在LSI模型下最优K值出现差异的原因。

4.2.3 实验系统

我们的实验部分是基于四个现实世界、来自不同领域的软件系统。iTrust（在线医疗档案管理）、Maven（构建管理工具）、Pig（编程语言）、Infinispan（数据库）。iTrust系统是在这个领域内被广泛使用的数据集，其RTM由系统开发与维护人员提供。但是该系统提供的高质量RTM是方法粒度的，即表达的是需求和方法之间的追踪关系。而本文的方法是基于类粒度的，因此在iTrust中，我们将需求和方法之间的追踪关系转化为需求和类之间的追踪关系。我们的转换规则是：若需求和方法之间有追踪关系，则我们认为需求和这个需求所在的类有追踪关系。其它三个实验系统是 [47]最新公布的数据集，这三个开源系统用Jira作为项目管理工具，记录了整个软件开发过程的相关信息（例如在一个时间点要求新加一个功能）。针对Jira中提出的任务或问题等会有相应的pull request进行解决，这样这两者之间就拥有了追踪关系[47]将jira包含的信息和github上相应的pull request信息进行了爬取。我们对这个信息进行了整理形成了系统对应的RTM数据集。对于数据整理的详细过程将在下面章节介绍。代表4.2列举了这四个系统的基本信息。

4.2.4　研究问题

我们的实验目的是分析结合用户反馈和代码紧密度分析能否改善自动化可追踪生成场景下的混合方法，为了达到这一目的，我们需要回答如下研究问题。RQ: 在需求到代码的可追踪性生成场景下我们的方法能否优于基线方法？为了回答这个问题，我们选择了以下三个基线方法：（1）只利用信息检索技术的纯信息检索方法（简称为*IR-ONLY*）；（2）基于信息检索方法，引入代码紧密度分析的方法（简称为*TRICE*[12]）；（3）基于信息检索方法，考虑代码结构信息和用户反馈信息的先驱方法（简称为*UD-CSTI*[15]）。我们将我们的方法命名为*CLUSTER*，在对比*CLUSTER*和三个基线方法时，我们依次使用三种不同的主流模型（VSM、LSI和JS）。同时本方法的另一个目标是尽量减少用户参与，即使用尽量少的用户反馈。所以，对于基线方法UD-CSTI，用户需要判断所有的追踪线索（我们方法中的用户判断环节通过RTM来模拟，即我们假设用户判断全部都是正确的）。而实验中我们的方法里需要用户判断的追踪线索不超过全部追踪线索的3.5%。即对于给定需求，iTrust中需要判断四个类，Maven中需要判断三个类，Pig中需要判断8个类，Infinispan中需要判断?个类与给定需求的相关性。基于以上设置，通过比较UD-CSTI和CLUSTER来判断在我们的方法中是否少量的用户反馈就能起到明显的效果。除了4.2.1中提到的指标，我们还引入了统计显著性测试来判断CLUSTER是否显著比基线方法好。通过调研[12, 48]中所设计的显著性测试。对于候选追踪列表，我们选择在每个有效追踪线索位置的F-measure作为我们测试的单独依赖变量。选择F-measure的原因是我们想分析相比于基线方法，CLUTSRE方法能否同时提升查全率和准确率。F-measure计算公式如下：

$$F = \frac{2}{\frac{1}{R} + \frac{1}{P}} \tag{4.5}$$

参数解释：P表示准确率，R表示查全率，F是P和R的调和平均。从公式中可以看出准确率和查全率越大，F-measure的值越高。由于F-measure 在两个不同系统之间是成对存在的，因此我们使用Wilcoxon rank sum测试[49]验证以下零假设：H_0:CLUSTER和基线方法性能没有区别。我们采用$p-value$显著性差异水平0.05作为衡量检验结果的标准。

4.2.5　实验结果与分析

根据 4.2.1提出的度量指标，表 4.3展示了4个实验系统的实验结果。我们比较了CLUSTER和基线方法在这四个实验系统上的表现（*AP*，*MAP*与显著

性检验的p-value值）。在36个实验结果对比中，有30个实验结果CLUSTER的F-measure值明显优于基线方法（p-value<0.05）。这表明在绝大多数情况下，相比基线方法CLUSTER 提高了候选追踪列表的准确率和查全率。此外，对于实验系统iTrust和Maven，CLUSTER方法的AP和MAP几乎全部优于基线方法。只有在Maven-VSM和Maven-LSI 上CLUSTER方法的MAP值略逊于TRICE。然而在Pig系统上CLUSTER方法只有在VSM模型下AP值优于UD-CSTI，其他情况下效果均不如基线方法UD-CSTI。但是从表 4.3中可以看出两者差距比较小（差异小于0.5）。并且如前文所述CLUSTER 方法只需要用户判断3.5%的追踪线索，而UD-CSTI方法需要用户验证所有的跟踪线索。图 4.5展示并对比了12种实验组合下的precision-recall曲线。

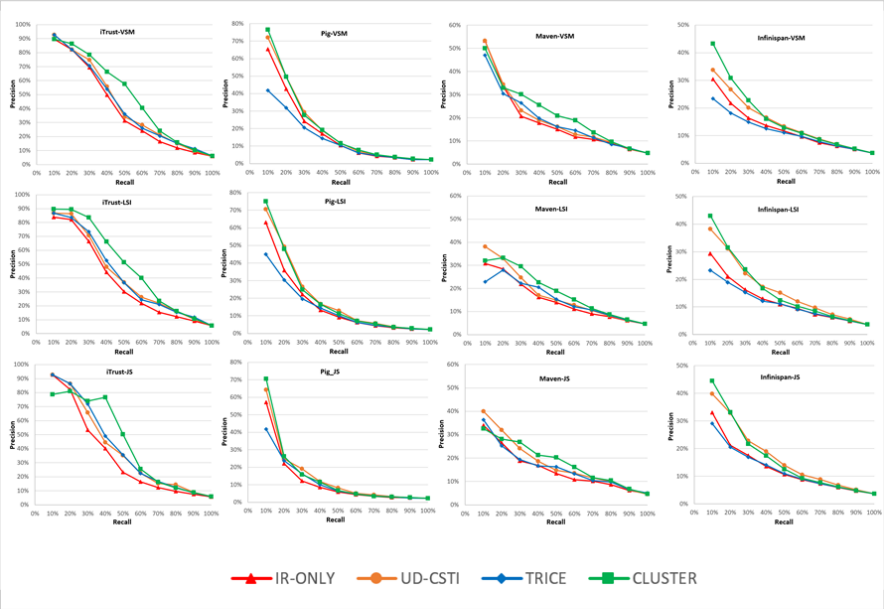

图 4.5: precision-recall图像

观察表 4.3和图 4.5，我们会发现TRICE方法的性能高度依赖通过纯信息检索方法生成候选追踪列表的质量。即TRICE方法依赖需求文本和代码文本的语料质量。在文本语料质量比较好的iTrust系统上，TRICE和UD-CSTI效果差异不大。但是在信息检索效果的AP和MAP都比较低的Pig和Maven系统，TRICE方法明显不如UD-CSTI。在Pig-VSM和Pig-LSI，TRICE方法的性能甚至不如纯信息检索方法。正如 **??**讨论的，没有用户反馈，TRICE方法只能把与给定需求相似度最大的类作为紧密度分析的输入，给与这个类代码依赖紧密度比较大的类对应追踪线索奖励。这种方法比较保守，然而也使得无法进一步提高基于信息检索方法的性能。例如在前面用例中，对于CLUSTER方法，用户需要判断两个

表 4.3: 实验系统的评价指标及Wilcoxon秩和检验结果

		VSM			LSI			JS		
		AP	MAP	p-value	AP	MAP	p-value	AP	MAP	p-value
iTrust	IR-Only	42.55	56.55	<0.01	41.21	55.73	<0.01	38.28	55.99	<0.01
	UD-CSTI	45.75	59.08	<0.01	45.42	59.35	<0.01	43.26	62.99	0.10
	TRICE	45.80	59.05	<0.01	45.18	58.76	<0.01	45.29	61.25	0.16
	CLUSTER	52.10	63.28	-	51.60	63.21	-	46.56	63.29	-
Maven	IR-Only	20.66	38.59	<0.01	17.21	42.37	<0.01	19.45	40.70	<0.01
	UD-CSTI	21.68	39.42	<0.01	18.52	43.28	<0.01	22.12	42.10	<0.01
	TRICE	21.68	41.58	<0.01	16.85	46.17	<0.01	19.30	41.69	<0.01
	CLUSTER	25.44	41.38	-	21.11	45.47	-	22.32	43.57	-
Pig	IR-Only	21.98	42.36	<0.01	19.88	42.25	<0.01	15.61	35.56	0.02
	UD-CSTI	24.19	43.49	0.49	22.40	43.67	0.68	18.90	37.82	0.27
	TRICE	16.88	41.91	<0.01	15.85	40.72	<0.01	13.83	36.79	0.18
	CLUSTER	24.64	43.32	-	22.30	43.19	-	18.88	37.57	-
Infinispan	IR-Only	14.47	22.87	<0.01	14.15	22.98	<0.01	14.98	22.59	<0.01
	UD-CSTI	16.59	24.14	<0.01	17.95	24.76	<0.01	19.57	24.56	<0.01
	TRICE	12.79	20.92	<0.01	12.96	21.84	<0.01	13.70	21.28	<0.01
	CLUSTER	18.47	23.82	-	18.61	23.91	-	19.86	23.24	-

域的代表类与需求相关性。如果相关会通过两个类向外扩散给与其依赖关系紧密的类对应候选线索奖励。此外，与给定需求相似度最大的类对应的追踪线索如果是无效的，在TRICE方法中由于没有用户反馈，按照TRICE方法会给与这个类紧密度比较大的类对应的追踪线索奖励。即此时这个错误会被放大最终使得被”优化之后的候选列表”准确率和查全率更低。在CLUSTER和TRICE方法中由于用户反馈的存在就避免了这种情况。实际上，我们观察到在9个实验中CLUSTER和UD-CSTI的效果均明显优于IR-ONLY，即便是在语料质量比较好的iTrust系统，情况也是如此。这表明在建立需求到代码的可追踪性时，用户反馈是一项重要的资源。

然而，使用用户反馈意味着用户需要付出额外的努力。正如前文提到的UD-CSTI方法为了达到其最优效果需要用户判断所有的追踪线索，这是不实际的。而我们的方法CLUSTER只需要用户判断3.5%的追踪线索就能达到与UD-CSTI类似的效果甚至优于UD-CSTI。这在基本不影响候选追踪线索列表

表 4.4: 考虑用户判断遇到的不相关追踪线索

		Recall(20%)		Recall(40%)		Recall(60%)		Recall(80%)	
		Precision	FP	Precision	FP	Precision	FP	Precision	FP
iTrust	VSM	-46.59%	+81	-0.24%	+1	+16.81%	-261	+3.93%	-428
	LSI	-46.34%	+80	+6.40%	-29	+21.64%	-349	+3.93%	-403
	JS	-46.59%	+81	+7.73%	-41	+11.48%	-381	+2.52%	-441
Maven	VSM	-6.79%	+24	+8.57%	-113	+6.78%	-287	+0.52%	-74
	LSI	-0.51%	+2	+7.12%	-117	+4.51%	-242	+1.01%	-186
	JS	+1.94%	-8	+8.94%	-127	+7.39%	-351	+1.68%	-231
Pig	VSM	-30.50%	+426	-4.72%	+319	+0.85%	-434	+0.38%	-856
	LSI	-24.04%	+410	-1.53%	+142	+0.01%	-4	+0.21%	-511
	JS	-11.70%	+367	+0.29%	-55	+0.09%	-93	+0.23%	-777
Infinispan	VSM	-5.12%	+309	+0.85%	-185	+1.06%	-677	+0.66%	-1369
	LSI	-4.85%	+309	+1.76%	-401	+0.43%	-301	+0.32%	-697
	JS	-4.32%	+264	+1.42%	-301	+0.34%	-273	+0.27%	-634

效果的前提下大大减少了用户需要付出的努力。为了进一步比较CLUSTER方法的成本效益（成本是指用户做出判断花费的精力，效益是用户判断后候选追踪列表准确率和查全率的提高），我们比较不同查全率对应的精度和遇到的无效追踪线索数量，如表 4.4

从表中可以看出当查全率小于20%时，由于我们的方法是先让用户判断一些具有代表性的候选追踪线索，这些追踪线索未必是最可能有效的。因此刚开始我们相对于IR-ONLY算法会遇到更多的无效追踪线索，此时我们的准确率要比IR-ONLY低。在查全率大于20%时，我们前期的用户投入开始发挥作用，如表所示在查全率为60%时，iTrust-LSI方法的准确率提升了21.64%，这也说明我们的方法检索出的无效追踪线索更少。从表中可以得出用户在建立153个正确追踪线索的过程中，CLUSTER方法检索出的无效追踪线索比UD-CSTI少349个。当查全率在50%和80%之间时，CLUSTER方法相对于IR-ONLY的优势尤为明显。值得注意的是我们把用户判断的追踪线索全部排在了候选追踪列表的前面，这也是造成当查全率比较低时CLUSTER方法引入大量无效追踪线索的原因。

根据实验结果，我们还有一点额外的观察。（1）对于同一个实验系统在不同的IR模型下，CLUSTER方法的表现差异很大。这是因为CLUSTER方法主要是通过给追踪线索奖励来实现重排序，如前所述（公式）奖励值依赖于IR值，

不同的IR模型对于IR值的计算有不同的方式。（2）我们尝试过要求用户判断所有代码域与给定需求相关性，该做法对pig和Maven两个实验系统的性能提升不大。这可能是因为这两个系统的需求粒度太小。表 **??**也验证了这一点，iTrust一个需求平均由8个类完成，而Maven和Pig的一个需求分别平均由4，5个类完成。在将来的工作中，我们计划文本聚类的方法 [50]增大需求粒度。

4.3　本章小结

在本章中，我们介绍了结合代码依赖关系紧密度分析和用户反馈的软件可追踪生成方法。我们在四个实验系统（iTrust、Maven、Pig 和Infinispan）下设计实验，验证了方法的有效性。同时，我们介绍了通过实验surefire或junit运行开源系统测试集获得代码依赖的方法和我们根据 [47]公布的数据整理实验系统对应RTM的过程。

第五章　软件可追踪生成工具的设计与实现

目前，针对结合用户反馈和代码依赖的软件可追踪问题，领域内还没有面向用户的工具。在本章中，我们介绍了软件可追踪生成工具的应用场景，并详细阐述了工具的设计与实现，该工具集成了我们结合用户反馈和代码依赖紧密度分析的软件可追踪生成方法。此外，我们结合一个案例说明工具的使用流程。

5.1　应用场景

在软件开发过程中，会衍生出各种与软件利益相关者对应的软件制品，如图 5.1如需求、代码和测试集等。这些制品之间存在追踪关系，例如某个需求由哪些代码来实现，某个测试用例用来对哪些代码进行测试。通过各软件制品之间的追踪数据，软件利益相关者可以实时了解项目进度。但是在实际的敏捷软件开发中，手工建立软件制品之间的追踪关系耗时费力，容易出错。并且由于软件迭代速度比较快，人员流动性比较大，这都使得建立维护软件制品之间的追踪关系很困难。我们将介绍结合用户反馈和代码依赖的软件可追踪生成的过程。

软件可追踪生成工具结合代码依赖紧密度分析和用户反馈两部分信息，对通过信息检索方法形成的候选追踪线索排序表进行优化调整，图 5.1 中展示了软件可追踪生成辅助工具在软件可追踪建立场景中的作用效果。我们将从用户视角说明软件可追踪生成辅助工具的益处。

1. 当用户需要软件系统需求到代码的追踪数据时，可以通过我们的辅助工具将软件按的需求和代码导入。一方面工具会使用信息检索技术计算需求和代码之间的文本相似度并按照相似度自大到小生成候选列表；另一方面，工具会运行系统的测试集并插桩代码依赖捕获工具得到软件运行过程中方法进入、退出和数据访问记录，并通过处理该数据得到软件的代码依赖。接下来会进入用户交互界面，工具推荐候选线索交由用户判断该候选线索的有效性。为了降低用户判断的成本，工具会展示与当前类代码依赖紧密度比较大的其它类，并显示它们的调用拓扑结构，同时用户可查看需求和类的文本信息。用户可以根据以上信息对追踪线索进行判断或者跳过该追踪线索的判断。工具会根据用户的反馈结果更新候选线索列表的排序。

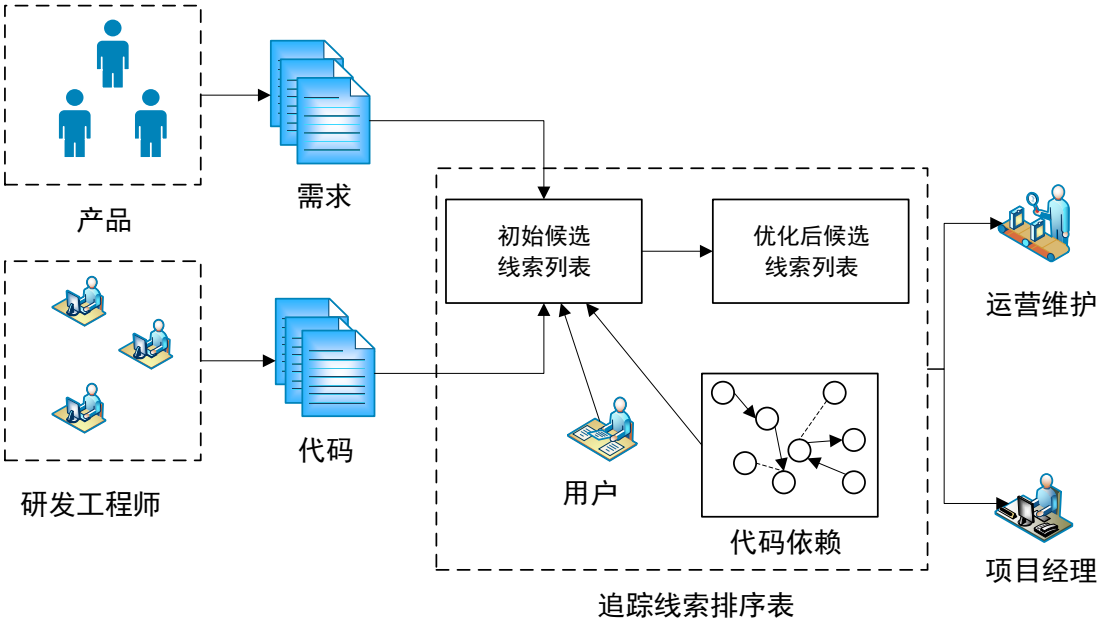

图 5.1: 结合用户反馈和代码依赖紧密度分析的软件可追踪生成技术应用场景

当用户不想再继续判断时，工具会向用户提供追踪线索排序表。一般情况下，对于每一个需求用户判断3.5% 左右的候选追踪线索就就能使整个排序列表的准确率、完全率大幅度提升。即通过我们的工具，用户只需付出较少精力即获得高质量的候选追踪列表。

利用软件可追踪生成辅助工具，工具会对系统做代码紧密度分析，并推荐少量候选追踪线索交由用户判断，然后根据用户判断的结果调整候选追踪列表，使得用户得到高质量的候选追踪列表。在用户判断过程中，用户可以借助工具查看目标类和目标需求的文本信息，目标类和目标需求的文本相似度，与目标类代码依赖关系紧密的其它类，以及它们之间调用关系的拓扑结构。用户通过这些信息可以更快更准确的判断候选线索有效性，从而提升候选追踪线索列表的质量。

5.2　工具的体系结构

图 5.2 展示了需求到代码可追踪生成辅助工具的体系结构，其中包括数据准备、信息检索、代码依赖捕获、用户交互以及候选线索列表优化五个模块。在本小节中，我们将对以上五个模块的设计与实现进行说明。

1. 数据准备模块：软件可追踪生成辅助工具的输入是需求文本和代码文本，这里的代码数据包含针对项目的测试集。输出是需求到代码的追踪线索列

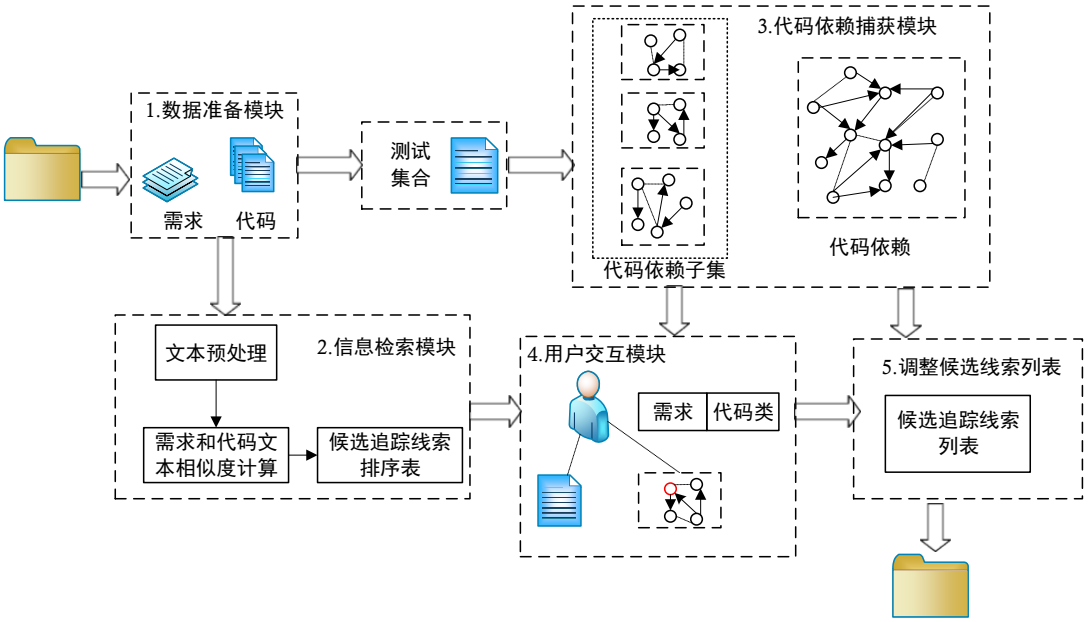

图 5.2: 软件可追踪生成辅助工具的系统体系结构图

表。数据准备模块主要是提供数据导入接口。通过左上角File菜单可以将需求和代码导入工具，如图...，数据导入模块是其它所有模块的基础，在代码依赖捕获模块，工具会通过数据准备模块中导入的测试代码集合得到代码依赖；在用户交互模块，工具会展示需求和代码的文本内容；在检索模块，工具对导入的需求和代码做文本相似度计算并生成候选追踪列表。

2. 信息检索模块：该模块是对数据准备模块中导入的需求和代码数据进行处理。对需求文本进行文本预处理，包括移除停用词、词形还原和词干提取等操作。对于代码文本，首先需要根据命名规则进行分词，然后与需求文本进行同样的文本预处理；接下来基于信息检索技术，计算需求文本和代码文本集合之间的文本相似度。我们实现了VSM、LSI与JS这三个被广泛使用的信息检索模型，以VSM（向量空间模型）为例，将需求文本和代码文本用高维向量q、r表示，向量中每个维度w对应一个单词的权重，权重w可用TF-IDF公式计算，对于q、r两个高维向量，两者组成一条候选追踪线索，可利用余弦距离计算向量之间的余弦相似度，我们将其定义为这条追踪线索的相似度值。然后我们根据追踪线索的相似度值，对候选追踪线索按照相似度值从大到小的顺序排序，形成候选追踪列表。

3. 代码依赖捕获模块：该模块主要负责捕获代码依赖关系，这里的代码依赖包括直接依赖（类之间的调用、使用和继承）和数据依赖（类之间的

数据共享）。虽然像TPTP（The Eclipse Test & Performance Tools Platform）和JPF（Java Plug-in Framework）等工具都能在软件系统运行过程中捕获系统中存在的方法之间的调用依赖。但是根据我们的调研，并没有一个现成的动态分析工具可以捕获方法之间的数据依赖。因此我们基于JVM提供的接口，注册函数调用数据访问等事件，并在其回调函数中存储方法进入、退出和数据访问记录，最终对该数据进一步处理得到代码依赖关系。根据项目使用的构建工具（Maven、Ant）我们采用不同的脚本运行测试集，并在此过程中通过不同的方式插桩我们的代码依赖捕获工具，得到各测试用例对应的代码依赖子集。然后我们对代码依赖子集进行合并得到代码依赖。

4. 用户交互模块：该模块首先会对代码依赖模块捕获的代码依赖进行紧密度分析，并通过代码依赖紧密度阈值生成代码域。对于每个需求，根据各域内类与需求相似度最大值，对用户为判断域按照相似度值自大到小的顺序重排序。对于排名第一的代码域，取其域内与需求相似度值最大的类交由用户判断与需求的相关性，如图用户界面所示。一方面在对话框左栏用户可以选择相关、不相关、跳过该判断和结束整个判断过程，右栏是当前需求的文本内容。另一方面用户可以点击帮助查看与该类代码依赖关系紧密的其它类，以及它们之间的拓扑结构，如图所示类依赖拓扑图，通过点击类，用户可以查看该类的文本内容。工具会根据用户的判断结果调整候选追踪列表，继续向用户推荐下一个需要用户判断的类或者生成最终结果。

5. 候选追踪列表优化模块：该模块会根据对代码依赖紧密度分析得到的代码域，和在用户交互模块中得到的用户反馈信息对候选追踪列表进行调整。当用户判断给定的候选追踪线索具有相关性时，对于该候选线索中的类，工具会通过不同的方式提升与该类在一个域中的其它类，和在域外但是和该类存在之间或数据依赖的类对应候选追踪线索的相似度值。该过程会迭代多次直到用户交互模块中，用户选择结束判断为止。此时工具会将候选追踪列表数据持久化到磁盘中，供软件利益相关者使用。

5.3 案例介绍

在本小节，我们将结合一个案例说明了工具的使用方式。

在iTrust系统开发过程中会衍生出包括需求和代码制品在内的各种软件制品。对于软件利益相关者而言，需求到代码的可追踪性具有重要意义。软件测试人员可以更好的对软件进行测试，项目负责人可以分析软件的完成情况更好

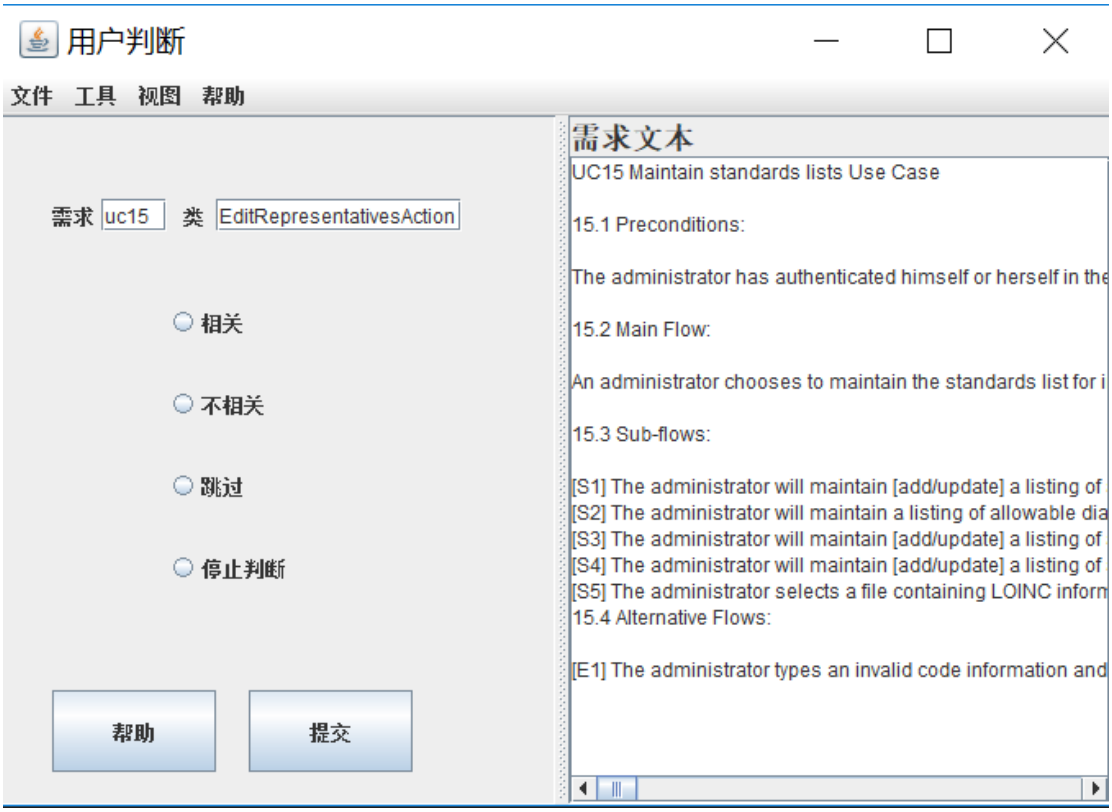

图 5.3: 用户交互界面效果图

的做决策。但是敏捷软件开发过程中并不会去专门维护需求到代码的可追踪性。当用户需要得到该数据时可借助于我们的辅助工具。首先点击菜单栏的文件选项，导入需求和代码（整个iTrust系统）。然后用户点击工具栏执行选项，此时工具会收集iTrust系统的代码依赖，同时基于信息索引技术形成初始候选追踪线索列表。接下来会进入用户交互界面，如图5.3所示，左栏显示了一条追踪线索，*uc15*到*Class*。接下来是四个选项，前两个选项相关，不相关要求用户判断这两条线索的相关性。如果用户不确定该追踪线索可以选择第三个选项跳过，此时工具会推荐新的候选追踪线索交由用户判断，如果用户不想继续判断，此时可选择最后一个选项停止判断。右栏显示该候选追踪显示的需求文本信息。如果在判断过程中用户需要更多与该追踪线索相关的信息，可以选择下方的帮助按钮，此时会进入图5.4 界面，

该界面分为两栏，左侧展示与类class之间紧密度大于紧密度阈值的其它类，并展示其拓扑结构。如图所示，每个矩形代表一个类，如果两个类是直接依赖关系则用带箭头的曲线表示，如果是数据依赖则用不带箭头的虚线表示。对于直接依赖边上的权值代表两个类之间发生的调用次数，对于数据依赖表示两个

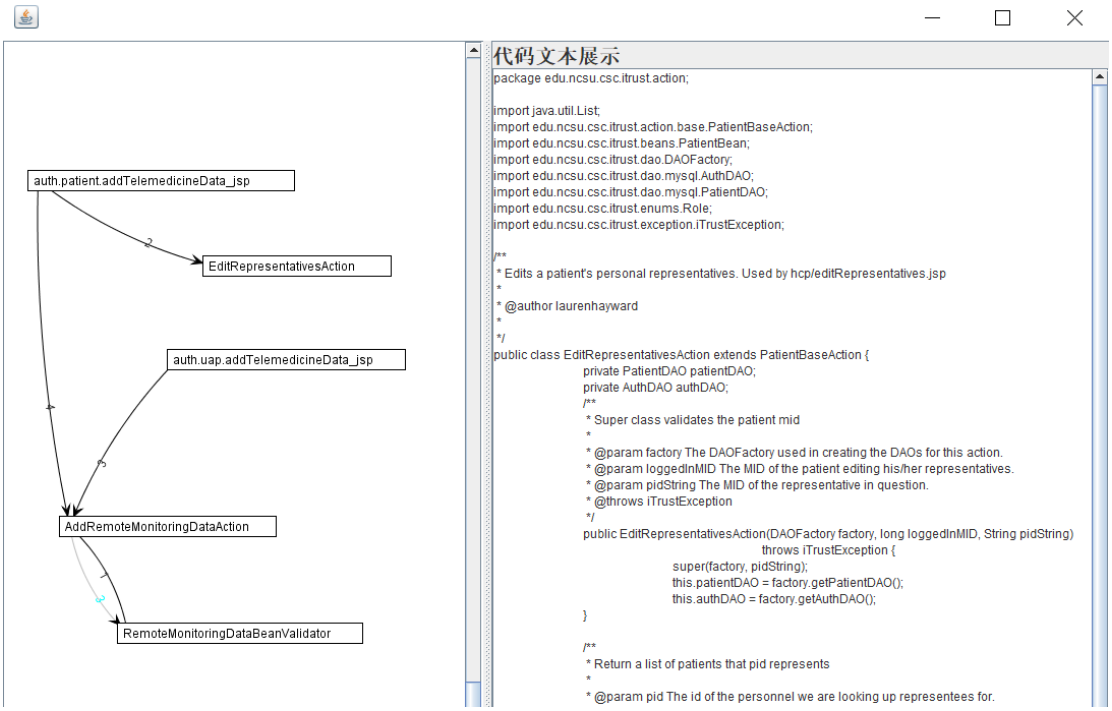

图 5.4: 代码依赖结构展示界面效果图

类共享的数据类型个数。直接代码依赖紧密度阈值默认为0.4，数据依赖紧密度阈值默认为0.8。用户在可以在图左下角自由设定紧密度阈值然后点击*更新*按钮生成新的拓扑结构。为了提高用户体验，用户可以根据需要拖到图中各矩形，用户可以使用*保存布局*按钮保存当前的布局，用户也可以使用*恢复布局*按钮使图行按照上一次存储好的布局展示。右栏用来展示代码文本内容。用户在左栏点击特定的类，右栏就会展示该类的文本内容。用户可以根据代码依赖信息和类与需求的文本信息做出更高质量的判断。

当用户选择结束判断时，会进入图5.5，该界面以表格的形式更直观的向用户展示追踪线索列表数据，追踪线索按照相似度值从大到小排列。点击左上方的*导出*按钮可以导出为pdf格式，方便保存。供以后软件利益相关者查阅。

5.4　本章小结

在本章中，我们介绍了需求到代码可追踪生成辅助工具的设计实现与应用场景，并辅以一个具体案例解释了工具的使用流程。需求到代码可追踪生成辅助工具能减少用户参与并且提高追踪列表的准确率和完全率。

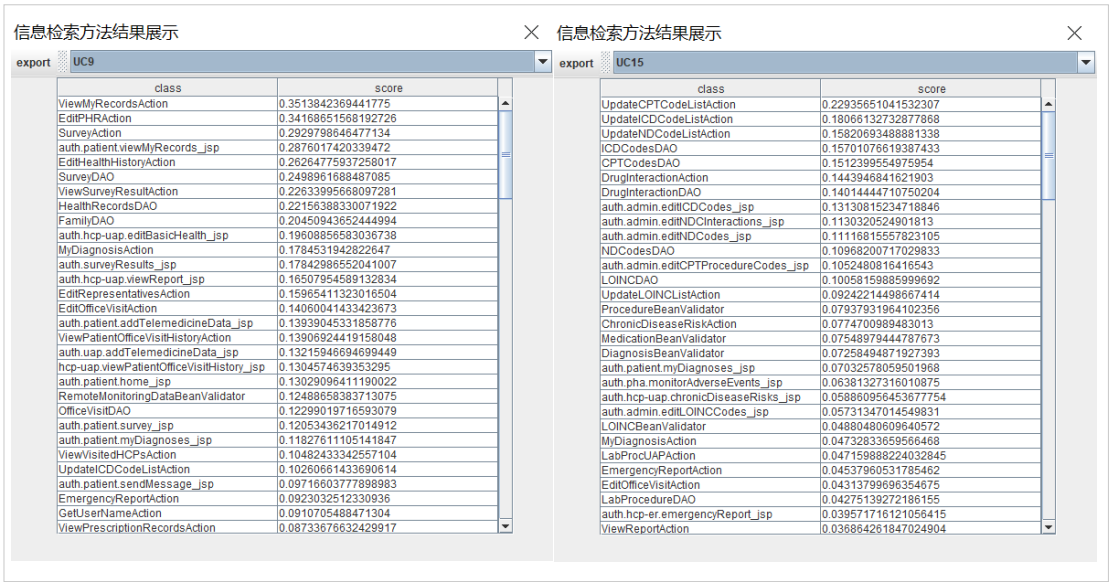

图 5.5: 需求到代码追踪线索列表展示界面效果图

第六章　总结与展望

6.1　工作总结

当前软件系统越来越庞大，软件开发流程迭代速度越来越快。这使得开发和维护人员很难建立和维护需求到代码的可追踪性，针对这种情况，我们提出了一套结合用户反馈和代码依赖的需求到代码可追踪生成方法并取得了一定效果。本文工作主要包括以下几点：

1. 我们提出了一种结合用户反馈和代码依赖紧密度分析的需求到代码的软件可追踪生成方法。通过代码紧密度分析，设置紧密度阈值将功能紧密的类绑定到同一个代码域中。在代码域中选择一个有代表性的交由用户判断域给定需求的相关性。然后根据用户判断结果（用户反馈）调整与该类关系紧密的域内其它类以及域外相关类对应候选线索的相似度值。从而在减少用户参与的情况下，提高追踪线索列表的准确率和完全率。

2. 我们在四个研究案例下设计实验，验证了上述方法的有效性。并且，我们阐述了通过对开源系统在github和jira上的数据处理，得到开源系统RTM的过程。

3. 我们设计并实现了一个需求到代码的可追踪生成辅助工具，并集成了结合代码依赖紧密度分析和用户反馈的需求到代码可追踪生成方法，以辅助用户得到高质量的追踪数据。

6.2　研究展望

在未来，我们的工作主要有以下两个方向可以进行探索：

1. 考虑用户反馈出错的可能性：当前方法假设用户的反馈信息一定是正确的。但是在实际环境中，很多软件系统比较复杂，并且并不是所有的用户都对软件系统很熟悉。因此，在这种情况下，对于有些候选追踪线索，用户可能会反馈错误的相关信息。以当前方法为基础，我们可以建立引入概率模型使得即使少部分用户反馈错误，我们的方法能够同样有效。

2. 为开源软件建立更高质量的可追踪数据集（RTM）：当前方法主要利用开源软件的功能请求信息（feature request）和对应的代码提交信息（code

commit）来建立需求到代码的追踪关系，接下来可以利用开源软件的其它信息，代码提交者，评论信息等进一步提高可追踪数据集的质量。

致　　谢

论文写在二零一八年的春天，在南大的生活也将划下句点。在南大的三年研究生学习，让我在学识、做事以及做人三个方面都收获颇多。回顾这三年的成长，点点滴滴都离不开老师、朋友以及家长的帮助。在这离别的时刻，我想对这些年一路陪伴以及帮助我的人表示感谢。

首先感谢我的导师，胡昊老师。在研究生的三年中，胡老师无论是在学习上，还是在生活上都给予了我悉心的指导与关怀。胡老师在平时既是一个前辈，在人生规划、职场生活方面给了我很多宝贵的建议；又是一位益友，在一起讨论科研问题，探讨项目经验，让我在学术以及项目上面都得到很多的启发。从胡老师那里学得的，不仅是做事，更是做人，让我终身受益。

感谢匡宏宇师兄，在科研和生活上给了我很大的帮助。和匡宏宇师兄合作期间，他严谨的学术风格以及科研素养让我受益颇多，并且我论文的选题也离不开师兄的帮助与教导。祝愿师兄科研顺利，家庭美满。

感谢一起奋斗的同门，聂佳、梁阳、戚可生、龚宇豪、单苏苏、朴琪、王勇、蔡欣辰，感谢大家三年的帮助与陪伴，希望大家以后都能有好的发展，科研之余也要多多锻炼和玩耍，科研的效率会更高。

感谢软件所的各位老师，无论是在硕士讨论班还是课堂，老师们的言传身教能够给我很多的启发与思考。同样感谢同学们在硕士讨论班的分享，让我对除自己研究方向之外的领域有了更多的了解。

最后我要向我的家人致以最衷心的感谢和最诚挚的敬意。

我很爱充满青春活力的校园生活，也感激在南大与你们的相遇。

各位，江湖再见!

简历与科研成果

基本情况 张宗飞，男，汉族，1993 年 1 月出生，山东省费县人。

教育背景

2015.9～2018.6 南京大学计算机科学与技术系 硕士

20011.9～2015.6 长安大学信息学院计算机科学与技术 本科

攻读硕士学位期间申请的专利

1. 匡宏宇，胡昊，吕建，**张宗飞**，“一种结合用户反馈和代码依赖的软件可追踪生成方法”，申请号：201810184034.9，已受理。

参考文献

[1] Coest: Center of excellence for software traceability. http://www.CoEST.org/.

[2] Xiaofan Chen and John C. Grundy. Improving automated documentation to code traceability by combining retrieval techniques. In *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), Lawrence, KS, USA, November 6-10, 2011*, pages 223–232, 2011.

[3] Giuliano Antoniol, Gerardo Canfora, Gerardo Casazza, Andrea De Lucia, and Ettore Merlo. Recovering traceability links between code and documentation. *IEEE Transactions on Software Engineering*, 28(10):970–983, 2002.

[4] G. Casazza G. Antoniol and A. Cimitile. Traceability recovery by modelling programmer behaviour.

[5] Patrick Mäder Patrick Rempel. Preventing defects: The impact of requirements traceability completeness on software quality. *IEEE Trans. Software Eng.*, 43(8):777–797, 2017.

[6] Patrick Mäder and Alexander Egyed. Assessing the effect of requirements traceability for software maintenance. In *28th IEEE International Conference on Software Maintenance, ICSM 2012, Trento, Italy, September 23-28, 2012*, pages 171–180, 2012.

[7] Jane Cleland-Huang, Orlena Gotel, Jane Huffman Hayes, Patrick Mäder, and Andrea Zisman. Software traceability: trends and future directions. In *Proceedings of the on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014*, pages 55–69, 2014.

[8] Jane Huffman Hayes Michael Vierhauser Giuliano Antoniol, Jane Cleland-Huang. Grand challenges of traceability: The next ten years. *CoRR*, abs/1710.03129, 2017.

[9] Balasubramaniam Ramesh and Matthias Jarke. Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*, 27(1):58–93, 2001.

[10] Orlena CZ Gotel and Anthony CW Finkelstein. The study of methods for language model based positive and negative relevance feedback in information retrieval. In *Proceedings of the First International Conference on Requirements Engineering*, pages 94–101. IEEE, 1994.

[11] Collin McMillan, Denys Poshyvanyk, and Meghan Revelle. Combining textual and structural analysis of software artifacts for traceability link recovery. In *ICSE Workshop on Traceability in Emerging Forms of Software Engineering (TEFSE)*, pages 41–48. IEEE, 2009.

[12] Hao Hu Patrick Rempel Jian Lu Alexander Egyed Patrick Mäder Hongyu Kuang, Jia Nie. Analyzing closeness of code dependencies for improving ir-based traceability recovery. In *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24, 2017*, pages 68–78, 2017.

[13] Jane Huffman Hayes, Alex Dekhtyar, and Senthil Karthikeyan Sundaram. Advancing candidate link generation for requirements tracing: The study of methods. *IEEE Transactions on Software Engineering*, 32(1):4–19, 2006.

[14] Paola Sgueglia Andrea De Lucia, Rocco Oliveto. Incremental approach and user feedbacks: a silver bullet for traceability recovery. In *22nd IEEE International Conference on Software Maintenance (ICSM2006), 24-27 September 2006, Philadelphia, Pennsylvania, USA*, pages 299–309, 2006.

[15] Annibale Panichella, Collin McMillan, Evan Moritz, Davide Palmieri, Rocco Oliveto, Denys Poshyvanyk, and Andrea De Lucia. When and how using structural information to improve ir-based traceability recovery. In *17th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 199–208. IEEE, 2013.

[16] Jane Cleland-Huang, Carl K Chang, and Mark Christensen. Event-based traceability for managing evolutionary change. *IEEE Transactions on Software Engineering*, 29(9):796–810, 2003.

[17] Patrick Mäder and Orlena Gotel. Towards automated traceability maintenance. *Journal of Systems and Software*, 85(10):2205–2227, 2012.

[18] Jane Huffman Hayes, Alex Dekhtyar, Senthil Karthikeyan Sundaram, E Ashlee Holbrook, Sravanthi Vadlamudi, and Alain April. Requirements tracing on target (retro): improving software maintenance through traceability recovery. *Innovations in Systems and Software Engineering*, 3(3):193–202, 2007.

[19] Andrian Marcus and Jonathan I Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proceedings of the 25th International Conference on Software Engineering*, pages 125–135. IEEE, 2003.

[20] Jane Cleland-Huang, Raffaella Settimi, Chuan Duan, and Xuchang Zou. Utilizing supporting evidence to improve dynamic requirements traceability. In *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, pages 135–144. IEEE, 2005.

[21] Denys Poshyvanyk Andrea De Lucia Malcom Gethers, Rocco Oliveto. On integrating orthogonal information retrieval methods to improve traceability recovery. In *IEEE 27th International Conference on Software Maintenance, ICSM 2011, Williamsburg, VA, USA, September 25-30, 2011*, pages 133–142, 2011.

[22] Jane Cleland-Huang, Adam Czauderna, Marek Gibiec, and John Emenecker. A machine learning approach for tracing regulatory codes to product specific requirements. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 155–164. ACM, 2010.

[23] Marc Eaddy, Alfred V Aho, Giuliano Antoniol, and Yann-Gaël Guéhéneuc. Cerberus: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis. In *The 16th IEEE International Conference on Program Comprehension (ICPC)*, pages 53–62. IEEE, 2008.

[24] Aharon Abadi, Mordechai Nisenson, and Yahalomit Simionovici. A traceability technique for specifications. In *The 16th IEEE International Conference on Program Comprehension*, pages 103–112. IEEE, 2008.

[25] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391, 1990.

[26] Susan T Dumais. Improving the retrieval of information from external sources. *Behavior Research Methods, Instruments, & Computers*, 23(2):229–236, 1991.

[27] Gerard Salton and Michael J McGill. Introduction to modern information retrieval. 1986.

[28] Thomas K. Landauer George W. Furnas Richard A. Harshman Scott C. Deerwester, Susan T. Dumais. Indexing by latent semantic analysis. *JASIS*, 41(6):391–407, 1990.

[29] Andy Zaidman Annibale Panichella, Andrea De Lucia. Adaptive user feedback for ir-based traceability recovery. In *8th IEEE/ACM International Symposium on Software and Systems Traceability, SST 2015, Florence, Italy, May 17, 2015*, pages 15–21, 2015.

[30] Wenjing Zhang and Junyi Wang. The study of methods for language model based positive and negative relevance feedback in information retrieval. In *Fourth International Symposium on Information Science and Engineering*, pages 39–43, 2012.

[31] Berthier A. Ribeiro-Neto Ricardo A. Baeza-Yates. *Modern Information Retrieval*. ACM Press / Addison-Wesley, 1999.

[32] Yin Li Ye Yang Qing Wang Lingjun Kong, Juan Li. A requirement traceability refinement method based on relevance feedback. In *Proceedings of the 21st International Conference on Software Engineering & Knowledge Engineering (SEKE'2009), Boston, Massachusetts, USA, July 1-3, 2009*, pages 37–42, 2009.

[33] Christopher D Manning, Prabhakar Raghavan, and Hinrich Sch眉tze. Introduction to information retrieval. *Journal of the American Society for Information Science & Technology*, 43(3):824–825, 2008.

[34] Christopher Morrell Dawn J. Lawrie, David W. Binkley. Normalizing source code vocabulary. In *17th Working Conference on Reverse Engineering, WCRE 2010, 13-16 October 2010, Beverly, MA, USA*, pages 3–12, 2010.

[35] David W. Binkley Dawn J. Lawrie. Expanding identifiers to normalize source code vocabulary. In *IEEE 27th International Conference on Software Maintenance, ICSM 2011, Williamsburg, VA, USA, September 25-30, 2011*, pages 113–122, 2011.

[36] Nan Niu Li Da Xu Jing-Ru C. Cheng Zhendong Niu Wentao Wang, Arushi Gupta. Automatically tracing dependability requirements via term-based relevance feedback. *IEEE Trans. Industrial Informatics*, 14(1):342–349, 2018.

[37] Giuliano Antoniol Massimiliano Di Penta, Sara Gradara. Traceability recovery in RAD software systems. In *10th International Workshop on Program Comprehension (IWPC 2002), 27-29 June 2002, Paris, France*, pages 207–216, 2002.

[38] Jin Guo, Jinghui Cheng, and Jane Cleland-Huang. Semantically enhanced software traceability using deep learning techniques. In *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*, pages 3–14, 2017.

[39] Hongyu Kuang, Patrick Mäder, Hao Hu, Achraf Ghabi, LiGuo Huang, Jian Lü, and Alexander Egyed. Can method data dependencies support the assessment of traceability between requirements and source code? *Journal of Software: Evolution and Process*, 27(11):838–866, 2015.

[40] Alex Dekhtyar Olly Gotel Jane Huffman Hayes Ed Keenan Greg Leach Jane Cleland-Huang, Adam Czauderna, Yonghee Shin Andrea Zisman Giuliano Antoniol Brian Berenbach Jonathan I. Maletic, Denys Poshyvanyk, Alexander Egyed, and Patrick Mäder. Grand challenges, benchmarks, and tracelab: developing infrastructure for the software traceability research community. pages 17–23, 2011.

[41] Cédric Jeanneret Martin Glinz Eya Ben Charrada, David Caspar. Towards a benchmark for traceability. In *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution, EVOL/IWPSE 2011, Szeged, Hungary, September 5-6, 2011.*, pages 21–30, 2011.

[42] John Grundy Robert Amor Xiaofan Chen, John G. Hosking. Development of robust traceability benchmarks. In *22nd Australian Conference on Software Engineering (ASWEC 2013), 4-7 June 2013, Melbourne, Victoria, Australia*, pages 145–154, 2013.

[43] Qing Wang Shoubin Li Barry W. Boehm Lin Shi, Celia Chen. Understanding feature requests by leveraging fuzzy method and linguistic analysis. In *Proceedings*

of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017, pages 440–450, 2017.

[44] Jane Cleland-Huang. Are requirements alive and kicking? *IEEE Software*, 30(3):13–15, 2013.

[45] Andrea Zisman Jane Cleland-Huang, Olly Gotel, editor. *Software and Systems Traceability*. Springer, 2012.

[46] Benedikt Burgstaller and Alexander Egyed. Understanding where requirements are implemented. In *IEEE International Conference on Software Maintenance (ICSM)*, pages 1–5. IEEE, 2010.

[47] Patrick Mäder Michael Rath, Patrick Rempel. The ilmseven dataset. In *25th IEEE International Requirements Engineering Conference, RE 2017, Lisbon, Portugal, September 4-8, 2017*, pages 516–519, 2017.

[48] Nasir Ali, Zohreh Sharafi, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. An empirical study on the importance of source code entities for requirements traceability. *Empirical Software Engineering*, 20(2):442–478, 2015.

[49] David J. Groggel. Practical nonparametric statistics. *Technometrics*, 42(3):317–318, 2000.

[50] Adelina Ciurumelea Sebastiano Panichella Harald C. Gall Filomena Ferrucci Andrea De Lucia Fabio Palomba, Pasquale Salza. Recommending and localizing change requests for mobile apps based on user reviews. In *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017*, pages 106–117, 2017.