功能需求到测试用例的可追溯性研究

翟宇鹏 洪 玫 杨秋辉
(四川大学计算机学院 成都 610065)

摘 要 软件开发主要由需求收集、设计、实现、测试和维护等几个阶段构成,其中维护始终是主要开销。在维护阶段,开发人员必须理解程序才能定位缺陷,而系统的需求和源代码、测试用例等之间的可追溯性连接能帮助减少开发人员在理解程序时的工作量。研究现有的功能定位技术和可追溯性方法,通过对现有方法的改进,提出了一种集成动态执行信息和IR技术的方案来建立需求到测试用例的可追溯性连接,用以辅助开发人员在维护阶段的工作。

关键词 软件开发,软件维护,需求可追溯性,功能定位
中图法分类号 TP301 **文献标识码** A

Research on Traceability of Functional Requirements to Test Case

ZHAI Yu-peng HONG Mei YANG Qiu-hui
(College of Computer Science, Sichuan University, Chengdu 610065, China)

Abstract Software development mainly comprises of requirements gathering, design, development, testing and maintenance. Maintenance is the main cost in the lifetime of software. During maintenance, developers have to understand program in order to locate the defect. The traceability links among requirements, source code and test cases can effectively help developers to understand the program. In this paper, the existing feature location methods and traceability methods were analyzed. Based these existing methods, an improved method which integrates dynamic execution information and information retrieval was proposed. This method can assist developers during the maintenance by establishing traceability links between requirements and test cases.

Keywords Software development, Software maintenance, Requirements traceability, Feature location

1 引言

软件系统开发是以需求为驱动的,因此开发出的系统人工产品都与需求有连接关系[1]。需求可追溯性的目的就是为了找到这种连接。需求可追溯性被定义为通过定义和维持需求与软件结构文档、源代码、测试用例等人工产品之间的关系,并以前向和后向的方式来描述和追踪需求的生命周期。需求可追溯性提供了优化需求,估计改变源代码的影响,验证、测试和理解系统以及找到可重用组件等诸多好处。最近的研究定量地证明了需求可追溯性带来的好处,当需求可追溯性可靠时,开发人员非常依赖需求可追溯性:当需求发生变化时,可以追溯到与变化相关的测试用例,做相应的测试;当测试用例发现缺陷时,可以追溯到可能的需求错误。因此功能需求到测试用例的可追溯性极大地提升了开发人员在进行软件维护时工作的质量和效率。

本文给出了一种通过需求与源代码以及源代码和测试的可追溯性关系,建立起需求和测试用例的可追溯性关系的方法,以辅助维护人员在维护阶段的工作。

2 相关工作

可追溯性方法、功能定位与本文的研究十分相关。此处将对与本文相关的研究工作做一个综述。

2.1 可追溯性方法

可追溯性方法可以被分为3种主要的分支:动态方法、静态方法和动态与静态结合的方法。动态方法[2]通过收集和分析执行轨迹[3]来识别软件系统中哪一个正在被执行的方法对应着某一特定的场景;但是动态方法无法区别出那些重叠的场景,因为一个单独的方法可能参与到多个场景当中。静态可追溯性方法[4-6]使用源代码结构和文本信息来恢复高层次和低层次人工产品间的可追溯性连接。在过去十年的科学文献中,静态可追溯性方法已经得到了大量的关注。

一系列的研究表明虽然静态方法不需要可执行的软件系统,但是其所获得结果的精度却不如集成了静态和动态技术方法的结果。Poshyvanyk[7],Dit[8]和Eaddy[9]联合了静态和动态信息,提出了集成的可追溯性方法。Poshyvanyk联合了基于场景的事件概率排名和使用潜在语义索引(Latenet Semantic Indexing,LSI)的信息检索技术(Information Retrieval,IR)来实现功能定位。他们的实验研究表明,联合了静态和动态信息的方法能获得比单一使用IR技术更好的执行效果。Eaddy提出了一种减少依赖分析的新技术,该技术通过联合已有的技术极大地提高了概念定位的准确度。这些作者开发出了一款集成了IR技术、执行轨迹和剔除依赖分析的概念功能定位工具CERBERUS。利用集成方法获得的实验结果表明,利用动态数据有助于提高静态可追溯性方法的精确度。但是集成方法也同样存在着动态方法所具有的缺陷,即当软件存在缺陷时收集执行轨迹很困难。本文选择了集成的方法来提高精确度,但当软件系统不可执行时,采用静态的方

翟宇鹏(1992-),男,硕士,主要研究方向为嵌入式软件开发与测试,E-mail:zhaiyp1992@126.com;洪 玫(1963-),女,教授,主要研究方向为软件分析与测试、软件形式化验证、软件自动化测试;杨秋辉(1970-),女,副教授,主要研究方向为软件测试、经验软件工程、数据库系统及其应用。

法来解决收集执行轨迹困难的问题，使得本文提出的方法的可用性更加广泛。

2.2　功能定位技术

动态功能定位依赖于系统运行时信息的收集，动态分析在理解程序领域已经有丰富的研究，而动态功能定位是该领域的一个部分。Wong[10] 提出了一种基于执行切片的技术，该技术能出色地得到多种粒度，如语句、基本块等。该技术需要输入一些测试用例训练特定的功能，而另一些测试用例则不用于训练，能够区别出哪些代码是唯一对应某一个功能和哪些代码是对应多个功能。相比于动态功能定位，静态功能定位不需要可执行的软件系统。Petrenko 和 Rajlich[11] 提出了一种称为结合依赖索引的信息检索（Dependency Search combined with Information Retrieval，DepIR）的技术。该技术在用户探索文本检索的结果时，利用静态代码的依赖关系来指导用户。Ali[12] 提出了一种通过使用结构关系来提高基于文本检索的功能定位技术。集成的动静结合的功能定位方法在精度上较单纯的动态、静态方法高。单个跟踪和信息检索（Single Trace and Information Retrieval，SITIR）通过使用收集场景的执行信息，利用 IR 技术返回出现在执行信息中的方法排名结果。该技术极大地减少了搜索空间并产生了更好的结果。Bogdan Dit[13] 提出了一种集成了信息检索技术和连接分析算法的方法，利用静态和动态的融合信息来提高功能定位。

3　功能需求到测试用例的可追溯性方法

为了建立需求到测试用例的可追溯性连接，本文方法是：首先通过功能定位方法建立起需求到源代码的可追溯性连接；其次建立源代码到测试用例的可追溯性连接；最后通过需求与源代码以及源代码与测试用例间的可追溯性连接间接地建立起需求到测试用例的可追溯性连接。

3.1　整体流程设计

本文的研究重点是如何建立需求到测试的可追溯性连接，并获得较高的查全率和查准率。针对这一问题，本文提出了解决方案，基本思路如图 1 所示。该方案的核心是通过功能定位技术找到与功能性需求高度相关的源代码（图中步骤（1）－（4）），然后建立源代码与测试的可追溯性连接（图中步骤（5）），从而间接地建立起需求到测试的可追溯性连接（图中步骤（6））。

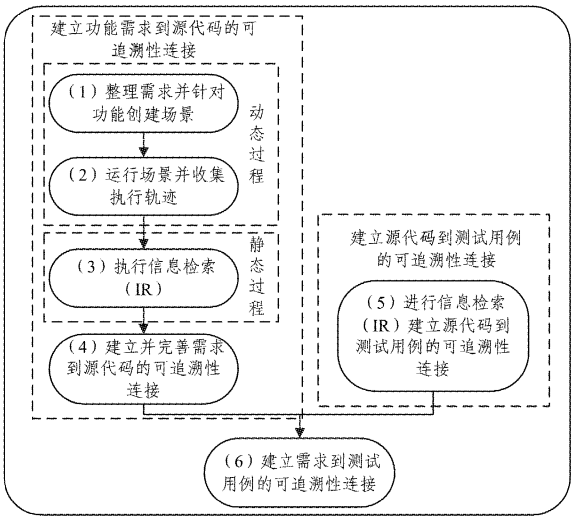

图 1　建立需求到测试的可追溯性连接方案的基本思路

（1）收集整理出所测试系统的需求以及需求所对应的功能（feature），这个步骤主要从系统的规格说明书等一系列文档中整理出来，然后根据这些功能性需求所对应的功能创建场景（scenario）。功能性需求是一个系统或系统的组件必须实现的可执行的功能。场景是一系列步骤的集合，执行这一系列的步骤来运行系统所包含的某一个特定功能。

（2）执行场景并收集执行轨迹。每一个执行轨迹中包含了若干个执行单元，如类和方法等。

（3）利用 IR 技术，从上一步骤收集到的执行轨迹中，识别出相应的功能性需求所对应的具体的方法或者类。

（4）建立并完善功能性需求到源代码的可追溯性连接关系。

（5）执行 IR 技术建立源代码到测试的可追溯性连接。该步骤将上一步骤中得到的功能性需求与源代码的可追溯性连接中的源代码部分作为查询语句，建立起源代码到测试的可追溯性连接。

（6）建立需求到测试的可追溯性连接关系。利用需求到源代码和源代码到测试的可追溯性连接关系，间接地建立起需求到测试的可追溯性连接。

3.2　建立功能需求到源代码的可追溯性连接

功能需求到源代码的可追溯性连接就是功能定位的过程，整个功能定位的过程是一个动静（动是指执行场景并收集执行信息，静是指 IR 技术的处理）结合的过程。

3.2.1　功能定位的动态过程

（1）创建场景

动态功能定位方法必须执行软件系统，为了在这个执行过程中调用某一个特定功能所对应的方法，必须针对该功能创建场景。本文中所涉及到的功能是指功能性需求所对应的功能。场景是针对某一个功能的一系列操作过程，它可以通过测试脚本创建而成。如果该系统带有 GUI 图形界面，那么点击图形界面的一系列按钮的步骤也是一个场景。因此，在带有图形界面的软件系统中，进行功能定位的操作时便可以不用测试脚本建立场景。

（2）执行场景并收集执行轨迹

针对某一个特定的功能创建好场景后，便可以在系统上执行这些场景。在执行场景的过程中同时收集执行轨迹。这些执行轨迹中包含的执行单元是该场景在执行的过程中所调用到的类或方法。本文使用了基于 Java Platform Debugger Architecture（JPDA）的轨迹追踪工具 MUTT[14] 来收集执行轨迹，基于 JPDA-based 收集轨迹的方法允许手动控制开始和停止。JPDA 内部有 3 层：1）Java 虚拟机接口（JVMTI），它在 Java 虚拟机上工作；2）Java 调试连线协议（JDWP），它是一个标准的通信协议；3）Java 调试接口（JDI），它给编程人员提供了一个高层次的 Java 语言接口。

3.2.2　功能定位的静态过程

功能定位的静态过程使用了信息检索技术（IR 技术），其过程如图 2 所示。

（1）创建语料库

使用预定义的粒度级别（类或方法等）来分割源代码，并从源代码中抽取出文档，也即在建成的语料库中，每一个类或方法都会有一个与自身相应的文档存在。

(2)语料库的标准化

该步骤将会把原文档分成一个一个单独的单词,去除标点符号、停词、大小写转换。停词是一种语言中最普遍的单词,因为它没有特殊的意义或者可识别度,所以在大多数情况下都不会将它作为查询的关键词。因此在创建索引时,这类停词会被去除掉,以减少索引的大小。英文单词如 the,a,this 等词就是停词。每一种语言的分词组件都会有一个停词集合,比如本文中使用的 Lucene 的分词组件中就有相应的停词集合。

(3)语料库索引

一个文本搜索引擎被用来索引这个语料库。虽然不同的文本检索引擎的工作方式是不同的,但是大部分的检索引擎都会创建一个与语料库中的文档相关联的数字索引,而后利用索引来度量文档之间的相似性。

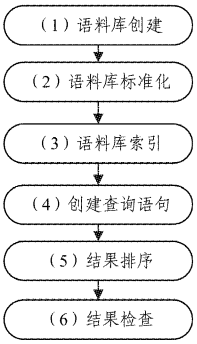

图 2 功能定位的静态过程

文中使用了向量空间模型(VSM)这一最常用的文本检索技术,该技术也是针对文本检索提出的。在 VSM 中,每一个文档都代表了一个词的多重集合,该集合与词的顺序、语义结构无关。VSM 不依赖于预定义的词汇表或语法,它可以很轻松地应用到任何类型的语料库中。在 VSM 模型中,每一个文档都与一个真正的值向量相关联。另外一种模型是潜在语义索引——LSI,它同样也可应用于恢复或建立需求到代码的可追溯性连接。Abadi[19] 通过实验证明,在任何种类的软件人工产品(源代码、测试等)的可追溯性恢复领域中,VSM 能比 LSI 提供更好的结果。同时,VSM 模型的实现最为简单,不需要特定的参数设置。

(4)创建查询语句

查询语句的作用相当于通过 Google 的搜索引擎检索需要的信息时所输入的若干关键字。一般来说,查询语句的创建是根据需求信息和通过对代码的理解来建立的。查询语句创建得越准确,查询的结果也就相应地越准确。大多数的文本检索引擎并不依赖于预定义的词汇表和语法,即使是错误的查询语句也可以用来查询。在查询过程中,查询语句也会进行标准化,然后被转化成与文档一样的向量。如果创建的查询语句在查询后并不能得到需要的结果,则表明该查询语句创建得不合理,需要重新创建。

(5)根据文档和查询语句的相关性对结果进行排序

查询语句被映射成与文档具有相同形式的向量,然后与语料库中的文档进行相似性比较。本文中使用的相似性度量方法是一种常用的余弦相似性度量准则。至此便可以把所有搜索出的文档向量和查询向量一起建立到一个 N 维向量空间中,维度由词的数目决定,每一个词就是一个维度。两个向量之间的夹角越小,则这两个向量越相似,从而可得到查询语句与一系列文档相似度的结果相似度结果按降序的形式排列。

(6)检查结果

检查结果也即对上一步返回的结果进行判断,判断是否有与查询语句高度相似的方法,进而得到需求与源代码的可追溯性连接。利用上一步骤中相似度高的结果,通过分析源代码可得出查询语句与哪一个方法或类关联,从而建立起需求到源代码的可追溯性连接。如果没有找到与需求相关的源码,则说明构建的查询语句不合理,重新构建查询语句,重复以上步骤。该步骤的检查工作目前没有较好的自动化方法,多采用人工操作方式,需要检查人员具有一定的知识和经验。

3.2.3 结合动静态过程进行功能定位

在动态功能定位中将会收集到场景所对应功能的执行轨迹信息,这些轨迹信息对应于软件被调用的方法,因此能知道到底是哪些源代码实现了这些功能。然后再利用基于 IR 技术得到的数据过滤掉一些执行轨迹,这样便可以抽取出与该功能最相关的源代码。文献[15]最早提出了这一思想。这样不再是针对整个项目或者软件系统的方法进行排序,同时这种排序结果与特定功能是最为相关的。

通过针对同一项目对比使用不同的定位方法定位到的方法数的大致比例结果如图 3 所示。可以看出,单纯使用动态功能定位得到的方法数目是很多的,基于 IR 技术也即静态功能定位方法得到的相关方法数目较少,而集成了动静结合的方法只对执行过的方法进行排序,这样使得查询后得到的方法数目极大地减少了,即起到了明显的过滤作用。

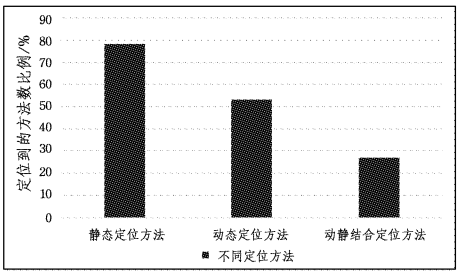

图 3 动静结合过滤对比

3.3 建立源代码到测试用例的可追溯性连接

本节建立源代码到测试用例之间的可追溯性连接,给出了一种基于 IR 技术的静态方法。基于 IR 技术建立源代码到需求之间的可追溯性连接方法与 3.2.2 节中所阐述的功能定位的静态过程是相同的。

首先对测试脚本建立索引空间,创建索引的过程与3.2.2节所阐述的过程完全一样,此处不再赘述。接着创建查询语句,由于创建的查询语句是针对测试用例中的方法,因此与 3.2.2 节中讲述的针对功能需求的查询语句有所不同。

已经有了需求到源代码的可追溯性连接,也就得到了相应需求所对应的源代码,这里的源代码是指具体的方法,包括完整的类名等。需求所对应的方法名和类名已知,便可以用该类名和方法名的组合作为查询语句,这个查询语句是对需求最为准确的描述。在 Java 编写的测试脚本中,如果按照命名约定编写测试脚本,那么就会出现相应的方法名和类名,也

即可以查询到对应的测试脚本，从而可以建立起源代码到测试之间的可追溯性连接。

作为一种补充，可以使用执行测试用例并收集执行轨迹的方式建立源代码到测试的可追溯性连接。但该方法需要执行所有的测试用例，收集到的执行轨迹很多，需要逐一与执行轨迹进行匹配，工作量较大，但这相比人工地从测试脚本中进行分析，找出与源代码对应的测试更容易。其最大的缺点就是需要测试脚本可运行。因此本文采用基于 IR 技术的方法。

4　实验验证

本节在第 3 节提出的方案的基础之上实现了一个工具原型，使用实际的开源项目对提出的方案的性能如可行性进行了验证。

4.1　可行性实验

4.1.1　实验对象

对文中提出的方案进行可行性分析实验。实验选取了开源项目 weka3.6 版本作为实验的测试项目，该工具是数据挖掘领域中知名的工具，使用 Java 语言编写，并且具有完备的测试用例集，因此选取它作为实验验证对象。

4.1.2　实验步骤

(1)创建场景

场景的创建是根据需求文档整理出相应的功能进行，本文针对 weka 的贝叶斯算法创建场景。

(2)执行场景并收集执行轨迹

首先创建好场景后运行场景，并收集执行轨迹，图 4 是执行贝叶斯算法后收集到的动态执行轨迹的效果图，该步骤实际上也是实现功能定位的动态过程部分。然后可以在轨迹中查询相应的方法，如“navie bayes”，利用返回的结果分析源代码，最终确定与贝叶斯算法相关的方法，建立起需求到源代码的可追溯性连接。

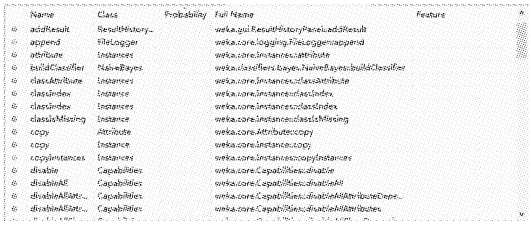

图 4　收集到的动态轨迹

(3)完善需求到源代码的可追溯性连接

重复前两个步骤，针对不同的功能创建执行场景并进行查询，从而建立起不同需求到源代码的可追溯性连接。然后通过利用方法之间的调用关系来完善可追溯性连接。

(4)建立需求到测试用例的可追溯性连接

接下来建立源代码到测试用例的可追溯性连接，然后间接地建立起需求到测试的可追溯性连接。在步骤(3)得到的可追溯性连接中，利用方法名和类名作为查询语句并执行查询。当然测试脚本所对应的索引空间和步骤(3)中一样，也由 Lucene 创建好。步骤(4)的执行结果如图 5 所示。通过人工分析测试脚本便可最终建立起贝叶斯算法与其对应的测试脚本间的可追溯性连接。重复步骤(4)便可建立起不同需求到测试用例之间的可追溯性连接。

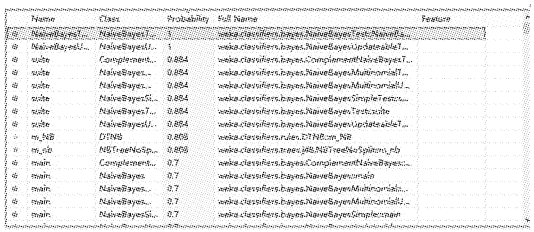

图 5　源代码到测试用例的连接结果

4.1.3　实验分析

本节利用开源软件 weka 验证了本文提出的基于动态执行信息和 IR 技术建立需求到测试的可追溯性连接方法的可行性。本节展示了如何使用本文提出的方案建立需求到测试的可追溯性连接的过程，下一节将对本文提出的建立需求到测试的可追溯性连接方案的性能进行分析。

4.2　性能实验

4.2.1　实验对象

本实验采用本文提出的方法与原有的功能定位技术 SITIR 以及可追溯性方法 TF-IDF 进行对比实验分析。选取的实验对象 Jedit 是一款流行的使用 Java 语言编写的文本编辑器。使用的 Jedit4.3 版本一共有 109(KLOC)代码，有 6400 个的方法，483 个类，选取了其中的 30 个功能。为了衡量性能指标，本实验采用 Precision 和 Recall 作为性能的评价标准。

4.2.2　实验步骤

实验分成两部分，第一部分为本文提出的功能定位技术与原有功能定位技术 SITIR 的比较，实验步骤如下：

(1)整理出两个方法需要使用的功能和查询语句；

(2)编译调试源代码，利用本文提出的方法和 SITIR 方法对 Jedit 建立需求到源代码的可追溯性连接，并记录实验结果；

(3)编译调试源代码，利用 SITIR 对 Jedit 建立需求到源代码的可追溯性连接；

(4)统计实验结果，分别计算两种方法的 Precision 和 Recall；

(5)根据两个实验结果完成对比分析。

第二部分为本文提出的可追溯性方法与原有可追溯性方法的比较，实验步骤如下：

(1)从需求文档中整理出 TF-IDF 所需要的功能和查询语句；

(2)执行 TF-IDF 方法，逐一输入查询语句，根据返回的结果，以一定的阈值逐一分析源代码，建立相应的需求到测试的可追溯性连接，并记录实验结果；

(3)利用第一部分实验得到的需求到源代码的可追溯性连接关系中的方法名和类名的组合作为查询语句，采用第 3 节中的基于 IR 技术的方法建立需求到测试之间的可追溯性连接，并记录实验结果；

(4)统计实验结果，分别计算两种方法的 Precision 和 Recall；

(5)根据两种方法得到的实验计算结果完成对比分析。

在本实验中，两种方法在分析结果时均涉及了相似性阈值的设定问题，阈值大小的设定将直接影响 Precision 和 Recall 的大小。通过分析多次实验中返回的结果，结合现有文

献对阈值的设定，对 Jedit 被测系统采用了 3 组阈值 0.60，0.65 和 0.70。

通过图 6 可以看出，本文给出的功能定位方法在 Recall 上平均有 10%左右的提高，在 Precision 上平均有 4%左右的提高，整体效果优于 SITIR 方法。

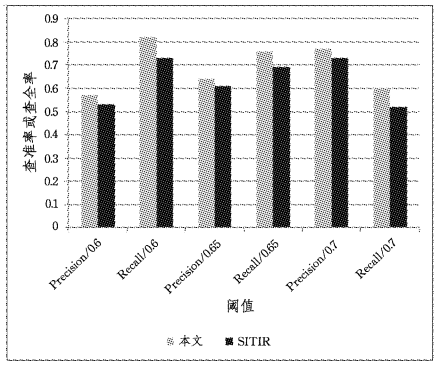

图 6 功能定位技术数据统计图

通过图 7 可以得出，本文提出的可追溯性方法在 Precision 和 Recall 方面较 TF-IDF 方法均有较大提高。这主要是由于 TF-IDF 方法针对整个项目的所有人工产品建立索引空间，在进行查询后会返回所有相似的结果，这些结果中存在大量的不相关的结果和冗余结果。而本文所提方法得到了更精确的需求到源代码的可追溯性连接，然后利用需求到源代码的可追溯性连接中的方法名和类名作为查询语句，使得查询结果更为准确。同时由于本文利用动态执行信息过滤掉了使用 IR 技术查询返回的无关的执行信息，相当于只对被调用的方法进行排序，从而使得本文的方法更具优势。

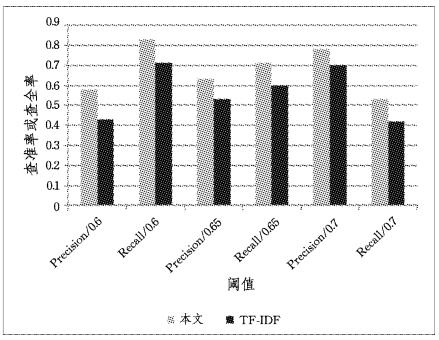

图 7 可追溯性方法数据统计图

结束语 本文以建立需求到测试的可追溯性连接这一问题作为研究对象，对现有的需求可追溯性方法、功能定位方法进行了研究，结合已有的研究工作提出了一种基于动态执行信息和信息检索技术(IR)的建立需求到测试的可追溯性连接方案。

本文提出的建立需求到测试的可追溯性连接方案具有以下特点：1)提出了一个完整的建立需求到测试的可追溯性连接方案；2)提高了建立需求到测试的可追溯性连接的精度。本文所提建立方案虽然是针对 Java 语言，但是可以很容易地扩展到其他语言中。

在面向对象的编程中，由于开发人员不好的编程习惯等原因会造成代码存在耦合性问题，其中一个类或方法可能会涉及到多个需求，这种一对多的情况可能会导致本文提出的方法所建立的可追溯性连接不够完整。该问题将作为论文后续工作继续研究。

参 考 文 献

[1] DAGENAIS B, OSSHER H, BELLAMY R K E, et al. Moving into a new software project landscape[C]// Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering. ACM, 2010: 275-284.

[2] LIU D, MARCUS A, POSHYVANYK D, et al. Feature location via information retrieval based filtering of a single scenario execution trace[C]// Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering. ACM, 2007: 234-243.

[3] WILDE N, CASEY C. Early field experience with the software reconnaissance technique for program comprehension[C]// International Conference on Software Maintenance. IEEE, 1996: 312-318.

[4] ABADI A, NISENSON M, SIMIONOVICI Y. A Traceability Technique for Specifications[C]// ICPC. 2008: 103-112.

[5] ANTONIOL G, CANFORA G, CASAZZA G, et al. Recovering traceability links between code and documentation[J]. IEEE Transactions on Software Engineering, 2002, 28(10): 970-983.

[6] MARCUS A, MALETIC J I. Recovering documentation-to-source-code traceability links using latent semantic indexing[C]// 25th International Conference on Software Engineering. IEEE, 2003: 125-135.

[7] POSHYVANYK D, GUÉHÉNEUC Y G, MARCUS A, et al. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval[J]. IEEE Transactions on Software Engineering, 2007, 33(6): 420-432.

[8] DIT B, REVELLE M, GETHERS M, et al. Feature location in source code: a taxonomy and survey[J]. Journal of Software: Evolution and Process, 2013, 25(1): 53-95.

[9] EADDY M, AHO A V, ANTONIOL G, et al. Cerberus: Tracing requirements to source code using information retrieval, dynamic analysis, and program analysis[C]// The 16th IEEE International Conference on Program Comprehension(ICPC 2008). IEEE, 2008: 53-62.

[10] WONG W E, GOKHALE S S, HORGAN J R, et al. Locating program features using execution slices[C]// 1999 IEEE Symposium on Application-Specific Systems and Software Engineering and Technology(ASSET'99). IEEE, 1999: 194-203.

[11] PETRENKO M, RAJLICH V. Concept location using program dependencies and information retrieval (DepIR)[J]. Information and Software Technology, 2013, 55(4): 651-659.

[12] ALI N, SABANÉ A, GUÉHÉNEUC Y G, et al. Improving bug location using binary class relationships[C]// 2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation (SCAM). IEEE, 2012: 174-183.

[13] DIT B, REVELLE M, POSHYVANYK D. Integrating information retrieval, execution and link analysis algorithms to improve feature location in software[J]. Empirical Software Engineering, 2013, 18(2): 277-309.

[14] http://sourceforge.net/projects/muttracer.

[15] LIU D, MARCUS A, POSHYVANYK D, et al. Feature location via information retrieval based filtering of a single scenario execution trace[C]// Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering. ACM, 2007: 234-243.