基于自然语言的软件信息检索工具

叶 挺 陈秀招 邹艳珍 赵俊峰 谢 冰
(北京大学信息科学技术学院软件研究所 北京 100871)
(高可信软件技术教育部重点实验室 北京 100871)

摘 要 随着开源软件项目规模的增大,如何快速地学习、理解一个软件项目成为基于复用的软件开发活动中的一个重要环节。这些开源软件项目的源代码和文档集的数量都比较庞大,开发人员在学习过程中查找和阅读这些软件信息需要花费大量的时间和精力。为此,提出一种基于自然语言的软件信息检索方法,以帮助开发人员快速地检索并理解其需要的软件信息。基于该方法,设计并实现了NaLSiSe工具。NaLSiSe工具在中国计算机学会主办的第一届软件研究成果原型竞赛中荣获优秀奖。以Lucene为例,验证了该工具可以有效减少开发人员阅读源代码和文档的工作量,同时具备简洁的用户界面和友好的用户体验。

关键词 软件复用,软件项目,信息检索,自然语言提问
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.09.017

Design and Implementation of Natural Language Based Software Information Retrieval Tool

YE Ting CHEN Xiu-zhao ZOU Yan-zhen ZHAO Jun-feng XIE Bing
(Software Institute, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)
(Key Laboratory of High Confidence Software Technologies, Ministry of Education, Beijing 100871, China)

Abstract Open source software projects have become a kind of important sources in software reuse. When the source code and code related documents in a project are very large, it is time-consuming for end-users to find the right information (code or document) segments. We proposed a natural language based software information retrieval approach, which helps developers to get the software information they need more quickly and conveniently. Based on this approach, we designed and implemented a natural language based software information search engine (NaLSiSe). We took Lucene as an example to illustrate how NaLSiSe automatically analyses and organizes the related software information, constructs query in natural language based search engine so as to improve the precision.

Keywords Software reuse, Software project, Information retrieval, Natural language question

1 引言

随着软件复用技术的不断发展,开源软件项目逐渐成为一种非常重要的可复用软件资源,并得到软件开发者的重视和日益广泛的应用[1,2]。然而在实际应用中,我们发现开发人员在学习这些软件项目时却需要花费大量的时间和精力。一方面,目前互联网上虽然存在着大量的项目相关学习资料,包括官方文档、Wiki、学习手册等等,但是这些资料往往来源分散、组织复杂,不易于开发人员进行检索和阅读;另一方面,这些学习资料往往在内容上相互依赖或者相互补充,开发人员还需要通过整理来发现各种学习资料之间的关联关系,从而建立一个整体的软件信息框架。因此,为了帮助开发人员快速地检索并理解其需要的软件信息,需要提供一种基于自然语言的软件信息检索方法。一方面,需要提供一种方法来支持自然语言提问的检索方式;另一方面,需要建立代码和文档之间的关联,既结合问题的分析,又结合查询的重构,最终帮助开发人员得到更加准确的结果。

根据A. Marcus等人[3]的调研,在软件工程领域内,超过20种任务(如特征定位、代码搜索和复用、需求分析、逆向工程等)都会使用到信息检索技术。目前,在软件信息检索方面已经存在很多研究工作。譬如瑞士苏黎世大学的Abraham Bernstein等人[4]设计了Ginseng工具,它主要利用了一些自定义的语法规则将英文的自然语言查询问题转换为SPARQL检索语句,然后通过标准的SPARQL语句对本体知识库进行检索,来获取相关的答案。Brian de Alwis等人[5]开发了Ferret工具,利用Eclipse JDT和Java反射机制来获取软件

到稿日期:2013-12-25 返修日期:2014-01-16 本文受国家863计划:网构化软件生产、构造和复用技术与工具(2012AA011202),国家自然科学基金:软件构件自动标签及其应用技术研究(61103024)资助。

叶 挺(1991-),博士生,主要研究领域为软件工程、软件信息检索、数据挖掘技术等;陈秀招(1990-),硕士生,主要研究领域为软件复用、软件复用库系统和软件信息检索等;邹艳珍(1976-),博士,副教授,主要研究领域为软件复用、软件资源管理、数据挖掘技术等,E-mail:zouyz@sei.pku.edu.cn(通信作者);赵俊峰(1974-),博士,副教授,主要研究领域为软件工程、基于构件的软件开发、情境感知的软件系统、智慧城市等;谢 冰(1970-),博士,教授,博士生导师,主要研究领域为软件工程、软件复用、人工智能等。

代码的结构信息，并以形式化方法存储起来，以此支持开发人员完成一些概念性的查询。微软的 Andrew Begel 等人[6]设计开发了 Codebook 工具，通过挖掘代码库、文档库、缺陷库，将软件制品的各方面信息组织起来，试图为整个开发团队中的人员构造出一个完整的关联网络。Michael Würsch 等人[7]实现了一个 evolizer 工具集，他们利用语义网[8]组织软件制品中的各种数据，并以 Ginseng 接口完成对自然语言查询的支持。

上述研究工作为基于自然语言的软件信息检索奠定了基础，但仍存在以下不足：首先，在信息组织方面，没有注重组织代码类与类之间、类与文档之间的关联信息，并借助这些关联信息做一些简单的推理。而且在查询接口方面，也缺少对问题的深入分析，不方便用户进行提问与使用。

基于上述目标，本文提出了一种基于自然语言的信息检索方法，该方法通过关联软件项目的代码和文档，结合对两者的分析，不仅可以得到基本的结构信息和关联信息，并借助推理得到一些增量信息，而且在查询构造方面，通过自然语言的问题分析，还可以构造不同层次的查询；另外，在软件信息的检索方法方面，根据关联分析和相似度计算，也可以有效地保证检索的效果，最终可以帮助开发人员快速地得到相关答案。

本文第 2 节介绍了该方法的概览；第 3—5 节给出了该方法的具体设计；第 6 节对该方法中相关算法进行了实验与分析；最后进行了总结和展望。

2 基于自然语言的软件信息检索方法概述

为了提高软件信息检索的准确性，基于自然语言的检索工具需要解决 3 个核心问题，即如何分析并提取代码和文档中的信息；如何分析自然语言问题并构造查询；如何实现软件信息的检索并返回相关答案。为此，本文提出的基于自然语言的软件信息检索方法可分为 3 个层次（模块）：软件信息的获取和组织、自然语言提问的查询构造、软件信息检索与精化，如图 1 所示。

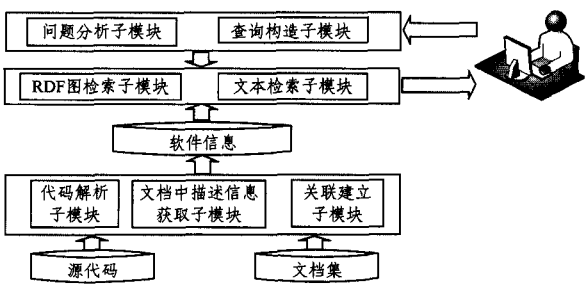

图 1 工具架构图

软件信息的获取和组织的主要工作是对项目的代码和文档进行预处理，从而获取代码的结构信息和代码与文档的关联信息，然后按照特定的格式组织起来。主要包括代码解析、文档中的描述信息获取、关联代码与文档这 3 个工作。

自然语言提问的查询构造主要包括两部分工作：1）从自然问题中提取三元组，并完成自然语言问题到 SPARQL 检索语句的转换，最终可以检索 RDF 图；2）从自然语言问题中提取关键词，并完成查询的重构，最终可以检索文本文档。这样，在完成软件信息的获取和组织之后，我们就可以对用户提出的自然语言问题进行分析，从而构造查询来检索软件信息。

软件信息的检索与精化主要是根据对问题的理解和分析，采用 RDF 图检索和文本检索相结合的方式加以实现。由于 RDF 图检索需要对三元组进行映射，存在着很多三元组不一致的情况，再加上 RDF 图中的信息本身就比较有限，因此还需要结合对文本的检索来获得最终的检索结果。

在下面的章节中，本文将从这 3 个模块出发分别介绍基于自然语言的软件信息检索工具设计与实现的详细过程。

3 软件信息的获取与组织

本文获取的信息主要包括 3 个方面，即代码结构信息、文档中的描述信息、代码与文档的关联信息，如图 2 所示。

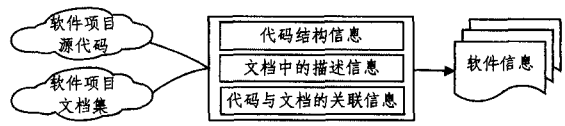

图 2 软件信息获取示意图

下面将逐个说明如何实现 3 个方面信息的获取。

3.1 代码结构信息

我们关注的代码结构信息主要有两点：1）代码中类的成员信息，包括方法和属性，这一类信息可以通过对代码进行简单解析得到；2）代码中类与类之间的关系信息，在该工具的场景下，类与类之间的关系主要定义为 3 种，即继承、调用和关联。

我们借助外在的代码解析工具 Eclipse JDT，首先获得源代码的抽象语法树，然后对各类信息进行分别存储，以完成最终的关系遍历。这里主要用到的是类信息和属性信息、方法信息，其中类信息的数据结构中包括了类所继承的父类、属性信息的顺序表、类所包含的方法信息的顺序表等；属性信息的数据结构中包括了属性名、所属的类，以及去除数组及常见容器结构后获得的简单类型等；方法信息的数据结构中包括了方法名、所属的类，以及方法调用信息的顺序表等。

得到基本信息之后，我们又借助了推理机制，通过自定义的传递规则获取类与类之间的间接调用信息。

3.2 文档中的描述信息

一般来说，对一个开源工具，可用的文档主要有 Javadoc、官方 Wiki，以及其他学习手册等。对于 Javadoc 来说，由于其关联关系是特定的，即每个类对应于一个 Javadoc 文档，因此不需要再进行关联建立；而对于 Wiki 和学习手册等，首先需要从其特定的格式（如 HTML 文件、WORD 文件等）中提取文本，然后进行后续处理。

从文档中提取到文本之后，要获取文档中的描述信息，这个过程可以大致分为两个步骤：首先需要将文档切分成句子，这是获取描述信息的首要前提，这个环节主要通过设定字符串的边界，如“\n”，“.”，“?”等来进行相应的切分；然后就是对一个句子进行词性标注、句法分析，并提取其事实三元组描述信息。

我们采用两种方法提取句子中的三元组，1）借助依存语法分析来提取三元组，2）通过最邻近名词集合来找到三元组。依存语法分析[9]是一种基于句子中词与词之间关系的形式语

法，它认为谓词是句子的中心，其他的语句成分依赖于谓词而构成句子。依存语法分析得到的词语之间的依存关系可以表示为一个三元组：

〈Relation，Governer，Dependent〉

其中，Relation 表示依存关系的名称，Governer 表示支配词，Dependent 表示从属词。我们根据依存关系三元组来提取问题的三元组，其核心思想为：若两个依存关系三元组中的支配词相同，且一个依存关系为 nsubj(nominal subject)，另一个依存关系为 dobj(direct object)或者 pobj(object of preposition)，则我们从中可以获取问题的一个核心三元组。

另外，我们还借助最邻近名词集合的方法来补充其三元组，这种方法的基本思想是通过找到句子中的所有名词，然后在两两邻近的名词之间找到其连接动词的方法来生成三元组。该方法生成的三元组中将存在一定的噪音，但是在后续过程中，该噪音并不会影响最终的效果。

3.3 代码与文档的关联信息

根据现有的研究工作，我们利用正则表达式和信息检索相结合的方式来构造代码与文档之间的关联。其方法框架如图 3 所示。

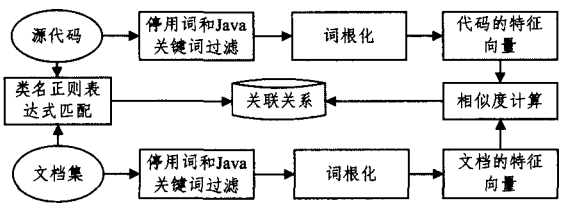

图 3 关联代码与文档的方法

从图 3 中可以看到，关联关系最终从两个方面来获取：1）类名的正则表达式匹配，2）文本相似度计算。正则表达式是一种强大而灵活的文本处理工具，使用正则表达式，我们能够以编程的方式来构造复杂的文本模式，然后对输入的字符串进行搜索。

4 自然语言提问的查询构造

在完成软件信息的获取和组织之后，我们将对用户提出的自然语言问题进行分析，从而构造查询来检索软件信息。这个环节主要包括两部分工作：1）从自然问题中提取三元组，并完成自然语言问题到 SPARQL 检索语句的转换，最终可以检索 RDF 图；2）从自然语言问题中提取关键词，并完成查询的重构，最终可以检索文本文档。

4.1 构造 SPARQL 检索 Query

首先需要对问题进行语法和句法分析，分析的目的是为了提取问题的主谓宾三元组，然后将其转换为 SPARQL 检索语句，并首先尝试从 RDF 图中检索答案。提取三元组的主要算法也是同前面介绍的文本描述信息获取一样，所以此处不再赘述。而转换部分，主要是考虑大小写和单复数的处理，并对准关键词和 RDF 图元素，最终通过 URI 标定来完成 SPARQL 检索语句的转换。

4.2 构造文本检索 Query

传统的文本检索方法为：从问题中提取出关键词并过滤掉停用词，然后直接构造查询。为了提高检索的准确率，在我们的技术框架中，文本检索的 Query 包括两个部分的构造，即三元组 Query、关键词 Query，三元组 Query 用于匹配前面抽取到的文档描述信息，而关键词 Query 用于检索一般的文档。

4.2.1 构造三元组 Query

三元组 Query 即问题中的三元组信息，前面我们介绍过，在进行 SPARQL 转换过程中，首先需要提取问题中的主谓宾三元组结构，提取到三元组后，一方面进行 SPARQL 转换，另一方面，直接保存该三元组信息，用于匹配文档描述信息。

4.2.2 构造关键词 Query

不同于传统的关键词 Query 构造方法，我们需要对查询进行重构，即首先对问题中的词条进行单复数扩展，然后对其进行相关词学习，并采用相关词推荐的方法来进行半自动的查询扩展。

使用相关词表来提升信息检索的效率是最直观的方法之一，因为与查询相关的文档可能采用不同的词来描述查询目标，这就导致相关文档与原始查询中的任何词都不匹配，而通过相关词表，我们就可以扩展查询，从而发现其相关文档。本文在借鉴相关研究工作的基础上，最终实现相关词学习的过程如下：

1. 对文本切分句子并过滤文本中的停用词；

2. 统计每个词的词频以及任意两词的共现情况，在一个句子中共同出现看作是一次共现；

3. 对一个词而言，另一个词与它的相关度界定为两词共现的次数除以该词的词频；

4. 当相关度大于某特定阈值，认为两词相关。

得到相关词表之后，我们通过过滤问题中的停用词来为余下的词半自动地推荐相关词列表，用户可以选择这些相关词加入其中，以完成查询的重构。

5 软件信息的检索与精化

在检索实现的过程中，根据对问题的理解和分析，我们首先采用 RDF 图进行检索，但由于 RDF 图检索需要对三元组进行映射，存在着很多三元组不一致的情况，再加上 RDF 图中的信息本身就比较有限，因此我们还需要结合对文本的检索来获得最终的检索结果。

5.1 RDF 图检索

本文主要使用 Jena 框架来进行语义数据的检索，Jena 框架通过使用 Java 类和变量来保持对语义 Web 的一致性处理，从而帮助开发人员方便地完成检索功能。此外，在 protégé 中，通过一些图形化的操作方式，我们也可以进行 SPARQL 检索。

5.2 文本检索

除了对 RDF 图的检索之外，我们还需要对文本进行检索。这个部分包括：基于三元组匹配的检索、基于文本相似度的检索和基于文档关联信息的推荐 3 种方式，下面将分别具体介绍。

5.2.1 基于三元组匹配的检索

我们提取到问题的三元组之后，首先将其映射到 RDF 图中，在这个过程中，只有代码结构类的问题才会完成映射，而其他问题往往不会映射到其中，所以我们还将问题三元组匹

配到文档中的描述信息三元组，若能够完成匹配，则说明文档中至少有一句话跟该问题描述的是同样的内容。

由于前面我们将文档描述信息三元组存储在文本文件中，因此这里的匹配就是简单的文本匹配，通过调用 JDK 中的文件操作 API 来完成基于三元组匹配的检索。

5.2.2 基于文本相似度的检索

若三元组匹配能够检索到相关文档，则结果相对比较准确，但是很多情况下，文档中并不是通过一句话来描述问题相关的信息，所以这种情况下，我们就需要传统的基于文本相似度的检索方法。

在进行检索之前，首先需要对文档进行索引，本文将输入文档数据以一种倒排索引(inverted index)的数据结构进行存储。在进行关键字快速查找时，这种数据结构能够有效地利用磁盘空间。倒排索引的核心是以关键字为中心，而不是以文档为中心，简言之，倒排索引并不是回答“这个文档包括哪些词”，而是回答“哪些文档包括这个词”。

根据上文介绍的过滤掉停用词、扩展单复数形式，以及推荐相关词，对这样一个输入的 Query，本文采用的相似度计算的公式如下：

$$\sum_{t \text{ in } q}(tf(t_d) * idf(t)^2) * queryNorm(q) \quad (1)$$

其中，对 Query 中的每一个 term，首先通过 TF-IDF 计算得到一个值，然后用 queryNorm(q)表示每个 Query 的归一化值，即每个 term 所占权重的平方和。

5.2.3 基于关联信息的推荐

本文采用的另一种检索相关文档的方法是基于关联信息的推荐，前面已经介绍了如何得到代码的结构信息和代码与文档的关联信息，基于这些信息，在检索的同时，考虑到这些关联关系，我们认为可以为用户推荐其隐含的相关文档，具体举例如下：

若检索结果中有文档 2，而文档 2 关联类 B，类 B 继承了类 A，而类 A 关联了文档 1，那么文档 1 也会被自动推荐出来，这样可以实现对单纯基于相似度检索的有效补充。基于关联信息的推荐方法如图 4 所示。

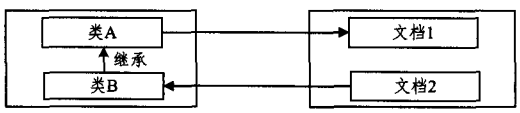

图 4 基于关联信息的推荐方法

6 实验与分析

本文实现的 NaLSiSe 是一个基于自然语言的软件信息检索工具，NaLSiSe 可以自动地获取一个项目的软件信息，并检索接口供用户检索软件信息。下面需要通过一个具体的项目来验证 NaLSiSe 工具的效果。

Lucene 项目是一个全文搜索引擎，于 2001 年作为高质量的 Java 产品加入 Apache 软件基金会。Lucene 项目源代码组织清晰，注释信息比较完备，且变量名都具有一定的语义，同时文档较为齐全，Javadoc、Wiki 以及相关开发手册《Lucene in Action》等都比较容易获取。这些特点都从不同程度上保证了项目的质量，所以本文在实验过程中选取 Lucene 项目作为实验项目，来验证工具的实现效果。

为了帮助开发人员更快地学习一个软件项目，首先需要获取该项目的软件信息。为此，我们在 RDF 图中预定义了 class、method、document 3 个类，并使用代码解析子模块来解析代码，得到代码的结构信息，使用代码与文档关联构造子模块来得到代码与文档的关联信息，并将这些信息都存储在 RDF 图中。

本文的软件信息检索模块用 3 个层次来检索软件信息，第一层检索是对 RDF 图的检索，涉及代码结构的信息，若能在 RDF 图中获取相关答案，则一定是比较准确的答案；第二层检索是对描述信息三元组的匹配，若能匹配到文本三元组，也应该是比较准确的答案，所以本文对这两个层次的检索结果都优先排序；然而，大多数情况下，用上述的两层检索都匹配不到相关信息，我们就只有借助一般的文本检索方法，即首先进行查询扩展，然后基于相似度计算得到其相关信息。

由于 RDF 图中检索到的答案比较准确，因此本文更多地关注在一般的文本检索环节，即语义分析和查询扩展是否可以有效地改进检索结果。

我们以 3 个问题为例，对文本检索结果进行验证和分析，3 个问题的基本信息如表 1 所列。其中，Q1 来源于 Lucene 的官方 FAQ，Q2 和 Q3 来源于 StackOverflow，而相关文档数量是我们人工标注的结果。下面将分别对每一个问题进行检索方法的验证和分析。

表 1 问题基本信息

问题	Q1：Where does lucene store the index?	What are the various ways of optimizing Lucene performance?	Why does the same search on different indexes return results with different scores?
问题来源	官方 FAQ	StackOverflow	StackOverflow
相关文档数量	8	5	10

在信息检索中，一般会以准确率(Precision)和召回率(Recall)来评价检索的效果，其定义如下：

$$\text{准确率} = \frac{\text{返回结果中相关的文档数目}}{\text{返回结果的数目}} \quad (2)$$

$$\text{召回率} = \frac{\text{返回结果中相关的文档数目}}{\text{所有相关文档的数目}} \quad (3)$$

对一个检索系统而言，每次检索的平均准确率 AvgP(Average Precision)是每次检索的准确率的平均值，整个系统的平均准确率 MAP(Mean Average Precision)是一个评价搜索引擎性能的常用指标。它指的是每次检索的平均准确率的平均值，具体计算公式如下所示：

$$AvgP_i = \int_{j=1}^{N} \frac{P(j) * pos(j)}{NumPos}$$
$$MAP_m = \frac{AvgP_i}{m} \quad (4)$$

其中，对每次检索来说，$P(j)$指的是在排名 j 以前的文档是检索的返回结果的情况下本次检索的准确率；而 $pos(j)$非 1 即 0，1 表示排名 j 的文档是本次查询的相关文档，而 0 表示该文档与本次查询不相关；$NumPos$ 指的是排名 j 以前的相关文档数量；MAP_m 表示 m 次查询的平均准确率的平均值。

本文工作的主要关注点在于是否可以在排名靠前的文档

中找到与用户输入问题相关的文本。对一个问题而言，检索结果中一般包括最佳答案和相关文档，最佳答案即通过该文档几乎可以直接回答该问题；相关文档即与该问题相关，但往往只能部分地回答该问题。

基于上述分析，本文首先用 MAP 来验证 NaLSiSe 检索相关文档的性能，从检索的平均准确率来看，具体结果如表 2 所列。

表 2　检索的平均准确率

	Q1	Q2	Q3	MAP
Lucene 搜索	0.38	0.10	0.16	0.21
加入语义分析	0.37	0.33	0.29	0.33
加入相关词推荐	0.37	0.39	0.29	0.35
加入关联文档推荐	0.41	0.39	0.37	0.39

从表中我们可以看到，Lucene 直接搜索得到的结果往往不好，其原因在于 Lucene 只是基于关键字进行搜索，当一个自然语言问题输入之后，反而会带来更多的噪音。当我们加入语义分析之后，检索效果的改进比较显著，首先是因为这一步我们过滤了停用词，并考虑扩展了单词的单复数形式，其次，若在问题中可以提取到完整的三元组，则基于三元组的匹配是比较准确的。

加入相关词推荐之后，效果改进的幅度相对较小，其原因在于：系统会自动推荐一个完整的相关词列表，当用户选择不同的相关词时，得到的检索结果会存在一些差异。本文提供这种方式的动机在于：当用户对系统给出的检索结果不满意时，可以通过加入不同的相关词来重构自己的查询，以得到另外的检索结果。

加入关联文档列表之后，效果仍然会有稳步的提升，其原因在于：本文基于类的继承关系、类与文档的关联关系建立了这些文档之间的关联，同时根据这种关联额外推荐了另外一个文档列表。系统这种方式额外给出了一些信息，所以对结果的优化是显著的。

结合表 2 中数据，我们得到不同情况下 MAP 的对比，如图 5 所示。

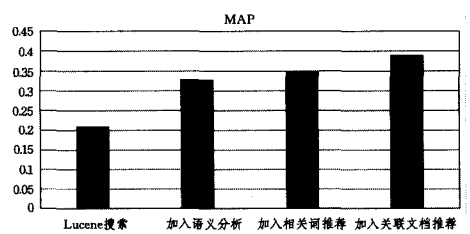

图 5　不同情况下 MAP 的对比

除了对平均准确率的分析之外，本文还通过最佳答案的排名情况来分析检索的结果，具体情况如表 3 所列。可以看到，加入语义分析或者相关词推荐以后，对于上述 3 个问题，在保证平均准确率有所提高的基础上，最佳答案的排名都有一定的改善。其中，Q1 的最佳检索结果从第二名上升到了第一名；Q2 与 Q3 的检索结果由没有找到分别改善为出现在检索结果的第四名和第六名。这表明本文所提出的检索精化方法在实践中能够有效帮助用户找到期望的结果，提高了检索效率。

表 3　最佳答案排名情况

	Q1	Q2	Q3
Lucene 搜索	Rank2	Not Exist	Not Exist
加入语义分析	Rank1	Rank4	Rank6
加入相关词推荐	Rank1	Rank4	Rank6
加入关联文档推荐	Rank1	Rank4	Rank6

结束语　帮助开发人员更好地学习和复用资源库中的软件项目是学术界和产业界都非常关注的一个问题。本文设计并实现了基于自然语言的软件信息检索工具 NaLSiSe。工具从功能上划分为软件信息的获取和组织、自然语言提问的查询构造、软件信息的检索和精化 3 个核心部分。本文以 Lucene 项目为例进行实验分析，通过 3 个问题来具体验证和分析了本文方法在不同程度上对传统检索的改进。实验表明，NaLSiSe 工具可以支持自然语言问题的查询，有效地完成软件信息的检索和自动优化，从而提高了软件信息检索的效率，可以帮助开发人员减少学习的时间成本。

在未来工作中，将对自然语言问题的分析、文本描述信息三元组的提取、细粒度的信息检索这 3 方面做进一步改进，探讨如何借助细粒度的分析、结合数据结构化的相关技术，强化问题理解与答案识别，从而从检索的粒度上优化检索过程，并最终实现一个软件项目的问答系统。

参 考 文 献

[1] Wang H, Wang C. Open source software adoption: A status report[J]. IEEE Software, 2001, 18: 90-96

[2] Madnmohan T R, De' R. Open source reuse in commercial firms [J]. IEEE Software, 2004, 21: 62-69

[3] Marcus A, Antoniol G. On the use of text retrieval techniques in software engineering[C]// Proceedings of IEEE/ACM International Conference on Software Engineering, Technical Briefing. 2012

[4] Bernstein A, Kaufmann E, Kaiser C. Querying the Semantic Web with Ginseng A Guided Input Natural Language Search Engine [C]// Proceedings of the 15th Workshop on Information Technology and Systems. 2005: 45-50

[5] de Alwis B, Murphy G C. Answering Conceptual Queries with Ferret[C] // Proceedings of IEEE/ACM International Conference on Software Engineering. 2008: 21-30

[6] Begel A, Phang K Y, Zimmermann T. Codebook: Discovering and Exploiting Relationships in Software Repositories[C]// Proceedings of IEEE/ACM International Conference on Software Engineering. 2010: 125-134

[7] Würsch M, Ghezzi G, Reif G, et al. Supporting Developers with Natural Language Queries[C]// Proceedings of IEEE/ACM International Conference on Software Engineering. 2010: 165-174

[8] Berners-Lee T, Hendler J, Lassila O. The Semantic Web [J]. Scientific American, 2001, 284(5): 34-43

[9] Nivre J. Dependency grammar and dependency parsing [R]. Technical Report MSI report 05133. 2005