我们在进行语料库分析时也需要除去不包含实际含义的功能词(例如“the”“is”)及应用十分普遍的词汇词(例如“want”).

词性还原(lemmatization)和词干提取(stemming)是词形规范化的两类重要方式.词形还原把一个任何形式的语言词汇还原为一般形式(能表达完整语义).词干提取抽取词的词干或词根形式(不一定能够表达完整语义).我们使用了 NLTK Python[15]工具包中的 WordNetLemmatizer 和 SnowballStemmer 来完成词性还原和词干提取.

3) 去除同一个代码对应的描述中重复的描述

4) 统计每个方法名的逆向文本频率(IDF 值)

IDF 值是信息检索领域用来衡量一个词普遍重要性的度量,这里,我们统计 API 方法名的 IDF,使用如下公式计算:

$$IDF=\log\left(\frac{\text{总的代码片段数}}{\text{该}API\text{方法名出现的代码片段数}}\right).$$

通过统计 API 方法名的 IDF 值,我们可以回答哪些 API 提供了代码的核心功能而不是通用简单功能这个问题.表 1 展示了 IDF 值最低的前 10 个 API 方法名,包含了最常被调用的方法例如 println 和 toString,这些常用的函数对于区分不同代码片段的功能起到的作用小.

Table 1 Top-10 methods with lowest IDF

表 1 IDF 值最低的前 10 个 API 方法名

API 方法名	IDF 值	API 方法名	IDF 值
println	1.76	run	3.07
add	2.09	equals	3.15
main	2.21	put	3.24
get	2.50	close	3.30
toString	2.76	printStackTrace	3.32

5) 建立每个方法名对应的描述短语集合

对于每一个 API 方法名,我们计算与其相关的所有描述短语的 TF-IDF 值,以度量 API 方法名和描述短语的相关性,这样就能回答哪个 API 和哪个描述短语相关性更高的问题.在信息检索领域,词频(TF 值)指某一个给定的词语在该文件中出现的频率,我们采用了这个思想,关键是将某一方法名相关的描述当成一个文件,这里统计某一个给定描述短语在包含某方法名的代码片段对应的描述中出现的频率.每一个描述短语的 IDF 值由总文件数(总方法名数)除以包含该描述短语的文件数(方法名数),再将得到的商取对数.

4 代码描述增强

互联网上的许多开源代码缺乏注释,或注释中不能有效体现其包含的功能特点.针对这一不足,DERECS 利用从语料库中提取出的描述与方法调用和代码结构对应关系特征,分析代码中调用的方法调用及代码结构,将方法调用相关描述和代码结构相关的描述信息与缺乏注释等描述的代码片段关联起来,实现对代码片段增强描述的效果.这样就使得更多的代码片段可以被索引到,从而提高代码搜索结果的质量.这一步仍然以函数为粒度进行描述增强.

4.1 基于方法调用描述增强

方法调用描述增强主要包括两方面内容:

1) 通过 Eclipse Jdt 分析代码,抽取出调用的方法签名(完全限定名+参数类型列表),如果该方法签名与 API 文档中某一方法签名匹配,则将 API 文档中方法签名对应的自然语言功能描述注入到代码片段的描述中.方法签名匹配规则是:方法的完全限定名必须一致;方法参数类型列表尽量匹配,允许不一致.

2) 通过方法名匹配语料库提取出的方法名,在一定阈值内注入描述短语.阈值包括:

① 方法名的最小 IDF 值(MinMIDF)

IDF 太低的方法名广泛被使用,对于区分不同功能的代码片段能力差,如 println 和 toString;另一方面,这些

方法一般功能简单,方法名能够准确描述其功能,无需多余的描述.故对于 IDF 值太低的方法名不进行描述增强.

② 描述短语的最小 TF-IDF 值(MinDTFIDF)

TF-IDF 太低的描述短语要么是 TF 值低,即描述短语与方法名成对出现的概率低、关系弱,要么是 IDF 值低,属于广泛被使用的描述短语,不能准确描述和区分函数的功能.故对于 TF-IDF 的描述短语,不对某一方法名增加 TF-IDF 值低的描述短语.

③ 当前函数中的方法调用集合与代码片段方法集合的最小相似度(MinMListSim)

设置这个阈值的直觉是:来源于越相似的代码对应的描述短语,与当前待处理代码片段的相关性越大,相关性太低的描述会干扰查询结果,故需要设置该阈值.这里,评估代码的相似性基于代码中调用方法名集合的相似性,两个方法集合 M_1,M_2 的相似度参考了自然语言领域计算两段文本语义相似性的 MCS 方法[16],计算公式如下:

$$Sim(M_1,M_2)=\frac{1}{2}\left(\frac{\sum_{m\in(M_1)}\max Sim(m,M_2)\times idf(m)}{\sum_{m\in(M_1)}idf(m)}+\frac{\sum_{m\in(M_2)}\max Sim(m,M_1)\times idf(m)}{\sum_{m\in(M_2)}idf(m)}\right).$$

其中,

- $\max Sim(m,M_2)$为方法名 m 与方法集合 M 中各个方法名的文本相似性最大值.其中,计算两个方法名文本相似性 $sim(m,m')$需将方法名按照驼峰切分成序列,并基于最长公共子序列算法计算序列的相似度;
- $idf(m)$为方法名 m 的逆向文本频率,降低常用方法权重.

例如,当前需要增加描述的函数方法调用集合 M_1={getRequest,executeRequest},对于方法名 getRequest,SO 中有两代码片段方法集合 M_2={getRequest,executeRequest},M_{23}={getRequest,getQueryParam}包含了该方法名,分别描述短语集为[‘get url’,‘post servic’,‘send http’],[‘retrieve params’,‘get http method’].在增强描述时,首先根据 getRequest 对应的描述集合[‘get url’,‘get http method’,‘post servic’,‘send http’,‘retrieve params’]中每个描述短语的 TF-TDF 值过滤掉与 getRequest 不太相关的描述短语(例如‘retrieve params’),但由于有些方法名用法较多,描述对应的种类也有很多,需要进一步根据当前函数的方法调用集合筛选出与当前更相关的描述,通过计算相似度,M_2 和当前函数集合更接近,来自 M_2 的描述‘get url’优先增加到该函数的描述集合.

由此可以看出:如果两个代码片段调用相同的 API 方法集合,通过语料注入的描述短语内容应该一致.为了提高效率,DERECS 先对需要增强描述的代码进行聚类,具有相同方法调用集合的代码片段只进行一次基于方法调用描述增强.

4.2 基于代码结构描述增强

代码的结构特征包括词法序列、语法树等代码抽象表示形式.代码克隆检测技术常利用代码的结构特征来查找相似的代码片段.一组代码片段互为相似代码(克隆),其中一个代码片段拥有较好的描述,就可以用来增强该克隆组的代码片段的描述.

基于代码结构的描述增强利用了 SWCE 和 AutoComment[17]中使用到的基于词法记号的代码克隆检测技术.基于词法记号的代码克隆检测技术对代码进行词法分析,得到词法记号序列,然后简单抽象(类型名、变量名、方法名等抽象为同一记号),得到参数化词法记号,并进行一定规整化,再进行公共子串算法(CCFinder[18])或频繁序列模式挖掘算法(CP-Miner[19])等找到重复出现的串,即代码克隆.SWCE 中将代码克隆组与每一个代码克隆实例的关键词关联.DERECS 采用 SIM[20]工具,采用 AutoComment 的设计原理,查找与代码片段克隆的开源代码片段,将代码片段的描述直接注入给与之结构相似(克隆)的代码片段中.

经过之前的步骤,DERECS 对于开源代码库的每一个代码片段构造一个对应的描述文档,该描述文档中包含 3 个域:(1) 基于代码描述语料库进行代码描述增强得到的描述;(2) 过滤后代码自身注释;(3) 代码中的方法调用名及其拆分形式.我们将描述文档命名为代码片段存储位置,集中存储所有代码片段的描述文档,将描述文档的集合作为下一步查询的对象.

5 代码搜索

在离线完成代码描述增强后,DERECS 利用 Lucene[21]进行在线查询,并使用人气度对结果进行重排序.这一步,我们使用了 Lucene 多域查询描述文档,对不同的域设置不同的查询权重,使得匹配代码本身注释和方法名更优先.当用户输入查询语句时,Lucene 检索与其相关性高的多个代码片段描述,获得候选代码片段集合.

对于候选代码片段集合,我们采用了人气度再对结果进行重排序.在许多代码搜索工作,如 PARSEWeb[12]、SNIFF[3]中都用到了人气度来对结果排序,其基本考虑是,越有人气(被多处调用或者多次出现的模式)的代码参考价值越高,但计算方式各有差异.我们这里考虑代码片段模式在语料库中频率.对于以方法调用为主实现的代码片段,具有相同方法调用集合的代码片段被聚到一类,一类模式在语料库中出现的频率作为排序的标准.

6 实验评估

为了验证 DERECS 的有效性,我们进行了两个实验:第 1 个实验,我们验证方法查询的准确率,并研究方法中各技术点对总体查询结果的影响;第 2 个实验,我们与现有在线代码搜索引擎作比较.本节首先介绍查询测试集及评估标准,然后介绍两个实验的详细方法,并对实验结果进行分析.

6.1 实验建立

这一节介绍用于评估方法有效性的查询测试集的来源、方法实现及参数选取、方法有效性的评估标准.我们利用了实际用户查询数据,并参考了现有代码搜索相关工作中用到的数据及评估方法.

6.1.1 查询测试集和对象

建立了一个用户真实的自然语言查询语句的查询测试集,见表 2 前两列(后 3 列为 3 种不同工具的实验结果).我们从 Stack Overflow 中 Java 相关最频繁的问题选取了 10 个作为查询测试集的一部分.另外,从已有相关工作 SWCE[2]、SNIFF[3]的实验数据集中获得了更多的 Java 相关典型查询语句.该查询语句集中的查询语句不是精心挑选的关键词集合,而是典型的自由形式的自然语言查询,涉及了多个领域的问题.使用该查询测试集能够客观、全面地评估代码搜索方法的有效性.

Table 2 Evaluation query set

表 2 实验查询测试集

No.	Query	DERECS	SNIFF	Krugle
1	Convert an InputStream to a String	1	2	NF
2	Iterate through a HashMap	3	5	2
3	How to upload files to server using JSP Servlet?	1	2	1
4	How to download and save a file from Internet using Java?	1	NF	NF
5	How to parse JSON in Java	1	1	1
6	How to round a number to n decimal places in Java	NF	NF	NF
7	Java string to date conversion	1	4	5
8	Connect Java to a MySQL database	1	2	1
9	How to create a file and write to a file in Java?	1	1	NF
10	How to test a class that has private methods, fields or inner classes?	1	NF	NF
11	Send a HTTP request via URLConnection[5]	1	2	2
12	Redirect Runtime *exec*() output with System[5]	4	5	NF
13	Get OS Level information such as memory[5]	NF	NF	NF
14	SSH Connection[5]	1	1	1
15	Download and save a file from network[5]	1	3	4
16	Generate a string-based MD5 hash value[5]	1	6	NF
17	Read the content of a HttpResponse object line by line[5]	1	1	3
18	read a line of text from a file[4]	1	3	1
19	return an audio clip from url[4]	2	1	NF
20	execute SQL query[4]	1	1	2

查询对象是从 Github 下载的开源项目中抽取的 1 121 109 个代码片段(见第 2.1 节).因为 StackOverflow 从中选取了查询语句,我们在实验时并没有将来源于的代码片段作为查询对象.

6.1.2 方法实现及参数选取

在方法描述增强的第 1 步,我们从 jdk 的 javadoc 中抽取方法签名(完全限定名及参数类型列表)和方法功能描述对,作为实验对比的 SNIFF 的实现代码也包含 jdk 的 API 文档.

此外,我们选取注入阈值的方法如下:使用少量的数据(分数最高的前 20 个项目),抽取方法调用集合,共 31 836 个代码片段.挖掘其中不同的方法调用集合模式,并过滤掉包含方法调用数目小于等于 1 的.没有调用别的方法或者只调用了一个方法的代码片段,不利于观察及调整相似度阈值,故不加入考虑.对于支持度大于 10 的方法调用集合模式(共 84 个),设置多组阈值,人工分析描述注入结果,选取较为合理的默认阈值:Min*MIDF*=5, Min*DTFIDF*=0.5,Min*MListSim*=0.3.

在进行代码搜索,使用 Lucene 多域查询时,代码自身注释和代码中 API 方法名及其拆分形式权重为 2,基于代码描述语料库增强描述权重 1.

6.1.3 评估标准

为了评估代码搜索方法对于单个查询的有效性,我们引入了最佳匹配排名(best hit rank),表示查询结果中与查询语句匹配的最前一个回答的排名.好的最佳匹配排名意味着只需要花费较少开销就能找到所需的代码片段.

为了评估代码搜索方法对于整个查询测试集的有效性,我们引入了 k 准确率(P_k).k 准确率只考虑代码搜索方法返回的前 k 个结果,计算方法为

$$P_k=\frac{\text{最佳匹配排名小于等于}k\text{的查询数}}{\text{总查询数}}.$$

6.2 实验结果

6.2.1 实验 1

实验 1 验证方法查询的准确率,并研究方法中各技术点对总体查询结果的影响.表 2 中,DERECS 这一列展示了我们的方法返回结果的最佳匹配排名,NF 表示在前 10 个结果中没有找到所需代码.

由以上结果可以计算得到:当 k=1 时的准确率为 P_1=0.75,当 k=5 时的准确率为 P_5=0.90,当 k=10 时的准确率为 P_{10}= 0.90.对于大部分的查询,DERECS 在前 5 个结果中都可以找到所需代码片段.

DERECS 能够有效查找到高度抽象的查询语句所描述的代码片段,例如查询 4,该查询语句来源于 Stack Overflow,描述了高层功能“test private methods”,我们的方法查找到的代码片段(图 5(a)所示,https://github.com/stormzhang/9GAG/blob/master/app/src/main/java/me/storm/ninegag/view/swipeback/SwipeBackActivityHelper.java(第 81 行~第 88 行))和原来 Stack Overflow 的被接收答案(如图 5(b)所示,http://stackoverflow.com/questions/34571)一样,包含 getDeclaredMethod,setAccessible,invoke 方法调用.虽然这些方法对应的 javadoc 描述和代码片段原本的注释都不含高层功能信息,通过基于方法调用的描述增强,我们给开源代码增加了来源于 Stack Overflow 的相关自然语言信息,扩大了查询范围,从而可以有效地查询到结果.

a
```
public void convertActivityFromTranslucent() {
    try {
        Method method=Activity.class.getDeclaredMethod("convertFromTranslucent",new Class[0]);
method.setAccessible(true);
method.invoke(mActivity,new Object[0]);
    } catch (Throwable t) {
    }
}
```

b
```
Method method=targetClass.getDeclaredMethod(methodName,argClasses);
method.setAccessible(true);
return method.invoke(targetObject,argObjects);
```

Fig.5 Code snippet examples of search results and SO accepted answer

图 5 搜索到的代码片段及 StackOverflow 被接收答案代码片段示例

但是,对于查询 6 和查询 13,我们的方法应用并没有得到查询结果,主要是由于阈值的设置,没有对相应代码片段增加 Stack Overflow 描述,仅基于代码自身注释和 API 文档描述是不能查到相应代码片段的.

6.2.2 实验 2

第 2 个实验,我们与现有的代码搜索工作 SNIFF 以及 Krugle 作比较.我们没有直接与 SWCE 等及查询测试集来源的其他工作直接比较的原因是输入形式差异太大,例如对于 SWCE,模式的关键词来源于其代码实例的关键词(代码当做文本词语处理)集合,缺乏对代码中方法调用的理解.用户需要查与 API 使用相关的代码片段时,需要用具体的 API 方法名作为关键词,例如搜索完成“successfully login and logout”这样功能的代码,需要输入“FtpClient”这样的关键词才能搜索到相关代码片段.

我们选择 SNIFF 是因为该工作对代码增加了 API 文档描述,扩大查询范围.与 API 文档中单一的描述不同,我们的方法增加了来源于更多用户的多样描述,可以消除更多用户查询和代码描述之间的不一致.

Krugle 是著名的开源代码搜索引擎,可支持自然语言形式的查询语句,将代码及注释统一当成文本处理.DERECS 对代码进行分析,并增加上相关描述,是在自然语言文本之间的查询.表 2 中,后两列分别是 SNIFF 和 Krugle 查询结果的最佳匹配排名,对于大部分的查询,我们的方法都要优于这两种方法.

SNIFF 的 k=1,5,10 时的准确率分别为 P_1=0.35,P_5=0.75,P_{10}=0.80,Krugle 的 k=1,5,10 时的准确率分别为 P_1=0.25,P_5=0.55,P_{10}=0.55.而在实验 1 中,我们得到 DERECS 的 k=1,5,10 时的准确率分别为 P_1=0.75,P_5=0.90,P_{10}=0.90.DERECS 的准确率明显优于 SNIFF 和 Krugle.

我们以 0.05 的检验水准对最佳匹配排名结果进行显著性检验,假设如下:

- H1(DERECS 的最佳匹配排名优于 SNIFF):有效零假设为$\mu^{DERECS}=\mu^{SNIFF}$,真零假设为$\mu^{DERECS}\geqslant\mu^{SNIFF}$,备择假设为$\mu^{DERECS}<\mu^{SNIFF}$;
- H2(DERECS 的最佳匹配排名优于 Krugle):有效零假设为$\mu^{DERECS}=\mu^{Krugle}$,真零假设为$\mu^{DERECS}\geqslant\mu^{Krugle}$,备择假设为$\mu^{DERECS}<\mu^{Krugle}$.

H1 和 H2 的备择假设即 DERECS 拥有更好的最佳匹配排名,用户能够更快地找到所需代码片段.

t-test 成对双样本检验中,对于未查询到结果的,为了方便数据分析,将其最佳匹配排名统一用 11 来代替,检验结果见表 3.

Table 3 t-test results

表 3 t-test 检验结果

假设	样本容量	方法	平均数	方差	自由度	统计显著性 p	T 统计量	T 双尾临界	判定
H1	20 20	DERECS SNIFF	2.37 4.32	9.91 14.89	18	1.49×10^{-2}	−2.69	2.10	拒绝
H2	20 20	DERECS Krugle	2.37 5.84	9.91 21.47	18	2.44×10^{-3}	−3.52	2.10	拒绝

表 3 中对于假设 H1,μ^{DERECS}=2.3,小于μ^{SNIFF}=4.32,且 $p=1.49\cdot10^{-2}<0.05$,具有统计显著性,故拒绝了假设 H1 的零假设,接受了 H1 备择假设,即,DERECS 返回结果的最佳匹配排名明显优于 SNIFF;对于假设 H1,μ^{DERECS}=2.3,小于μ^{Krugle}=5.84,且 $p=2.44\cdot10^{-3}<0.05$,具有统计显著性,故拒绝了假设 H2 的零假设,接受了 H2 备择假设,即,DERECS 返回结果的最佳匹配排名明显优于 Krugle.

7 相关工作

代码搜索是近年来软件工程的研究热点之一,研究者先后提出了许多不同的代码搜索方法.这些方法分别支持不同的输入形式,推荐不同粒度的代码.最常见的输入形式是自然语言查询语句,这与通用搜索引擎输入一致,还有一些代码搜索方法直接以源代码作为输入.这些方法针对不同类型的输入,推荐抽象的方法调用序列、代码片段、构件或应用等不同粒度的代码,下面简要地介绍相关工作.

Sourcerer[22]以查询语句为输入,查找功能的实现代码、现有代码片段的使用及包含特定属性及模式的代码;在代码实体完全限定名上使用 TF-IDF 技术,结合 right-most handside boosting 技术和图排序算法来识别流行

的类.

Xsnippet[23]以查询语句为输入,给对象实例化任务提供示例代码.该方法具有可变的查询范围,先通用查询覆盖更多相关代码,再专门查询找到更适合的代码示例.启发式排序规则包括代码片段长度排序、出现频率排序等.

Mica[24]以查询语句为输入,在普通搜索引擎的基础上,Mica为用户提供更多API使用的相关信息,返回相关的API方法类和域,根据频率相关性排序,并提供部分官方文档链接.

PARSEWeb[12]以查询语句为输入,帮助程序员解决知道对象类型但不知道怎样获得该对象的问题.PARSEWeb与Google Code Search Engine交互,通过对代码分析将代码片段抽象表示成方法调用序列的形式,然后对方法调用序列进行聚类及排序.

SNIFF 支持自由形式的查询语句输入,利用库的文档来补充代码中方法的相关英文描述,并通过基于类型的插入技术,从而获得与查询语句高度相关的代码片段.

Exemplar[25]以查询语句为输入,结合信息检索和程序分析技术,查询应用的描述、源代码及帮助文档.该方法关键字匹配API调用的描述信息,先联系到API调用,再联系到应用.

Portfolio[26]以查询语句为输入,检索到的函数能够解决查询语句中描述的多项任务,而不是任务中不同构件的代码示例.该方法利用函数调用图建立模型,模拟编程者浏览及关联其他函数行为.

Prefab[27]以查询语句为输入,为快速软件原型推荐所需的构件.通过模块抽取(从 SourceForge 中提取源代码)特征描述抽取(从Softpedia中提取文档)构建代码搜索引擎,推荐含有尽可能包含想要特征的模块.

PRIME[28]以部分代码作为输入,基于对象类型查找语义相关的代码片段.

SWCE以查询语句关键词为输入,为用户查找可行的代码示例.先对代码片段词法分析抽象,使用Type-2克隆检测工具定位相似代码片段.为了提高查询的质量,需要识别语料库中流行的解决方案,使用最大频繁项集挖掘算法.

DEEPAPI[8]使用深度学习循环神经网络的方法,对于给定的查询语句,生成API用法序列.

RACS[29]支持自由形式的查询语句,查找JavaScript框架代码示例.该方法考虑JavaScript框架代码特征,使用方法调用关系图来描述API调用之间的复杂关系;同时,从自然语言查询语句中也捕捉相对应的关系,提高了JavaScript框架代码搜索的准确率.

S6[30]以规格描述(包括关键字、类或方法签名、测试用例、约束、安全限制)作为输入,找到满足用户规格描述函数及类,避免了在重用代码时的大量修改工作.用户指定尽可能精确的规格描述,系统通过一套程序转换规则映射检索到的候选方法、类(例如对返回类型的转换).

Strathcona[31]从开发者正在编写的代码中自动抽取结构上下文(包括类、方法、域、继承关系、调用关系及实例化)作为查询语句,为基于框架的编程任务检索代码示例.示例代码库从现有使用该框架的已有应用中自动抽取.

Prospector[32]工具为程序员编写API客户端代码提供帮助.用户描述输入/输出类型,自动使用API方法签名及从大量示例客户端代码中挖掘到的jungloids来合成便于复用的查询结果.

CLAN[33]以Java应用作为输入,查找与给定应用相似的应用,基于相似应用的API调用也相似这一特征,建立API调用-包/类-应用链接,并扩展相关文本框架.

本文介绍的 DERECS 方法基于方法调用及代码片段的结构特征对代码进行描述增强的方法,减小了被搜索的代码与自然语言查询语句之间的差异,提高了搜索的准确性.

8 结束语

本文提出了一种基于增强描述的代码搜索方法 DERECS.基于开源项目、问答系统等构建了代码-描述语料库,并分析代码及自然语言描述,提取方法调用和代码结构相关特征值.该方法基于方法调用特征和代码结构特征增强描述,减小了被搜索的代码与自然语言查询语句之间的差异,扩大了搜索的范围.使用真实自然语言查

询语句作为测试基准,本文验证了DERECS的有效性.结果表明,DERECS的搜索效果显著优于SNIFF和Krugle.

References:

[1] http://code.openhub.net/

[2] http://www.krugle.com/

[3] Thummalapenta S, Xie T. Parseweb: A programmer assistant for reusing open source code on the Web. In: Proc. of the 22nd IEEE/ACM Int'l Conf. on Automated Software Engineering. ACM Press, 2007. 204−213. [doi: 10.1145/1321631.1321663]

[4] Chatterjee S, Juvekar S, Sen K. SNIFF: A search engine for java using free-form queries. In: Proc. of the Fundamental Approaches to Software Engineering. Berlin, Heidelberg: Springer-Verlag, 2009. 385−400. [doi: 10.1007/978-3-642-00593-0_26]

[5] Keivanloo I, Rilling J, Zou Y. Spotting working code examples. In: Proc. of the 36th Int'l Conf. on Software Engineering. ACM Press, 2014. 664−675. [doi: 10.1145/2568225.2568292]

[6] Lv F, Zhang H, Lou J, Wang SW, Zhang DM, Zhao JJ. CodeHow: Effective code search based on API understanding and extended Boolean model. In: Proc. of the 30th IEEE/ACM Int'l Conf. on Automated Software Engineering. ACM Press, 2015. 260−270. [doi: 10.1109/ASE.2015.42]

[7] Raghothaman M, Wei Y, Hamadi Y. SWIM: Synthesizing what I mean. Computer Science, 2016,3(1). [doi: 10.1145/2884781.2884808]

[8] Gu XD, Zhang HY, Zhang DM, Kim SH. Deep API learning. In: Proc. of the 24th ACM SIGSOFT Int'l Symp. on the Foundations of Software Engineering. ACM Press, 2016. 631−642. [doi: 10.1145/2950290.2950334]

[9] Stack exchange data dump. https://archive.org/details/stackexchange

[10] Stack exchange data explorer. http://data.stackexchange.com/

[11] Chen F, Kim S. Crowd debugging. In: Proc. of the 2015 10th Joint Meeting on Foundations of Software Engineering. ACM Press, 2015. 320−332. [doi: 10.1145/2786805.2786819]

[12] Treude C, Robillard MP. Augmenting API documentation with insights from stack overflow. In: Proc. of the 38th Int'l Conf. on Software Engineering. ACM Press, 2016. 392−403. [doi: 10.1145/2884781.2884800]

[13] http://eclipse.org/jdt

[14] http://nlp.stanford.edu/software/lex-parser.shtml

[15] http://www.nltk.org/

[16] Mihalcea R, Corley C, Strapparava C. Corpus-Based and knowledge-based measures of text semantic similarity. In: Proc. of the 21st National Conf. on Artificial Intelligence. AAAI, 2006. 775−780.

[17] Wong E, Yang J, Tan L. Autocomment: Mining question and answer sites for automatic comment generation. In: Proc. of the 28th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). IEEE, 2013. 562−567. [doi: 10.1109/ASE.2013.6693113]

[18] Kamiya T, Kusumoto S, Inoue K. CCFinder: A multilinguistic token-based code clone detection system for large scale source code. IEEE Trans. on Software Engineering, 2002,28(7):654−670. [doi: 10.1109/TSE.2002.1019480]

[19] Li Z, Lu S, Myagmar S, Zhou YY. CP-Miner: Finding copy-paste and related bugs in large-scale software code. IEEE Trans. on Software Engineering, 2006,32(3):176−192. [doi: 10.1109/TSE.2006.28]

[20] http://dickgrune.com/Programs/similarity_tester/

[21] http://lucene.apache.org

[22] Bajracharya S, Ngo T, Linstead E, Dou YM, Rigor P, Baldi P, Lopes C. Sourcerer: A search engine for open source code supporting structure-based search. In: Proc. of the Companion to the 21st ACM SIGPLAN Symp. on Object-Oriented Programming Systems, Languages, and Applications. ACM Press, 2006. 681−682. [doi: 10.1145/1176617.1176671]

[23] Sahavechaphan N, Claypool K. XSnippet: Mining for sample code. ACM SIGPLAN Notices, 2006,41(10):413−430. [doi: 10.1145/1167515.1167508]

[24] Stylos J, Myers BA. Mica: A Web-search tool for finding API components and examples. In: Proc. of the 23rd Int'l Conf. on Program. IEEE, 2015. [doi: 10.1109/VLHCC.2006.32]

[25] Grechanik M, Fu C, Xie Q, McMillan C, Poshyvanyk D, Cumby C. A search engine for finding highly relevant applications. In: Proc. of the 32nd ACM/IEEE Int'l Conf. on Software Engineering. ACM Press, 2010. 475−484. [doi: 10.1145/1806799.1806868]

[26] McMillan C, Grechanik M, Poshyvanyk D, Xie Q, Fu C. Portfolio: Finding relevant functions and their usage. In: Proc. of the 33rd Int'l Conf. on Software Engineering. ACM Press, 2011. 111−120. [doi: 10.1145/1985793.1985809]

[27] McMillan C, Grechanik M, Poshyvanyk D. Detecting similar software applications. In: Proc. of the 34th Int'l Conf. on Software Engineering. IEEE, 2012. 848−858. [doi: 10.1109/ICSE.2012.6227178]

[28] Mishne A, Shoham S, Yahav E. Typestate-Based semantic code search over partial programs. ACM SIGPLAN Notices, 2012, 47(10):997−1016. [doi: 10.1145/2398857.2384689]

[29] Li X, Wang ZR, Wang QX, Yan SM, Xie T, Mei H. Relationship-Aware code search for JavaScript frameworks. In: Proc. of the 24th ACM SIGSOFT Symp. on the Foundations of Software Engineering. ACM Press, 2016. 690−701. [doi: 10.1145/2950290.2950341]

[30] Reiss SP. Semantics-Based code search. In: Proc. of the 31st Int'l Conf. on Software Engineering. IEEE, 2009. 243−253. [doi: 10.1109/ICSE.2009.5070525]

[31] Holmes R, Walker RJ, Murphy GC. Strathcona example recommendation tool. ACM SIGSOFT Software Engineering Notes, 2005, 30(5):237−240. [doi: 10.1145/1095430.1081744]

[32] Mandelin D, Xu L, Bodík R, Kimelman D. Jungloid mining: Helping to navigate the API jungle. ACM SIGPLAN Notices, 2005, 40(6):48−61. [doi: 10.1145/1065010.1065018]

[33] McMillan C, Hariri N, Poshyvanyk D, Cleland-Huang J. Recommending source code for use in rapid software prototypes. In: Proc. of the 34th Int'l Conf. on Software Engineering. IEEE, 2012. 364−374. [doi: 10.1109/ICSE.2012.6227134]

黎宣(1989－),女,重庆人,博士生,主要研究领域为代码搜索.

王千祥(1970－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,系统软件.

金芝(1962－),女,博士,教授,博士生导师,CCF 会士,主要研究领域为需求工程,知识工程.