软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(12):3062−3074 [doi: 10.13328/j.cnki.jos.004817]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于代码库和特征匹配的函数名称推荐方法*

高 原[1,2], 刘 辉[1,3], 樊孝忠[1], 牛振东[1]

[1](北京理工大学 计算机学院,北京 100081)

[2](第二炮兵装备研究院,北京 100085)

[3](高可信软件技术教育部重点实验室(北京大学),北京 100871)

通讯作者: 刘辉, E-mail: liuhui08@bit.edu.cn

摘 要: 函数名称质量的高低,对于理解和维护程序非常重要.然而对于软件开发人员,尤其是母语非英语的软件开发人员,为函数选取高质量的名称比较困难.为此,提出一种函数名称推荐方法.首先,基于开源软件创建函数库;然后,对于某个需要推荐名称的函数 *f*,从函数库中检索与其相似的函数.对检索返回的相似函数用自然语言处理工具对函数名进行解析并获取标注词条,然后,从相应的函数体中提取特征代码并与相应的标注词条建立关联.基于此关联关系以及函数 *f* 的特征,自动推荐合适的函数名.该方法在开源项的 1 430 个函数中进行了初步验证,结果表明:有 22.7%的推荐结果与原函数名完全一致,有 57.9%的推荐结果与原函数名关键词一致或基本一致.

关键词: 函数名称;推荐;特征选择;算法;自然语言处理

中图法分类号: TP311

中文引用格式: 高原,刘辉,樊孝忠,牛振东.基于代码库和特征匹配的函数名称推荐方法.软件学报,2015,26(12):3062−3074. http://www.jos.org.cn/1000-9825/4817.htm

英文引用格式: Gao Y, Liu H, Fan XZ, Niu ZD. Method name recommendation based on source code depository and feature matching. Ruan Jian Xue Bao/Journal of Software, 2015,26(12):3062−3074 (in Chinese). http://www.jos.org.cn/1000-9825/4817.htm

Method Name Recommendation Based on Source Code Depository and Feature Matching

GAO Yuan[1,2], LIU Hui[1,3], FAN Xiao-Zhong[1], NIU Zhen-Dong[1]

[1](School of Computer Science and Technology, Beijing Institute of Technology, Beijing 100081, China)

[2](The Second Artillery Equipment Research Institute, Beijing 100085, China)

[3](Key Laboratory of High Confidence Software Technologies of the Ministry of Education (Peking University), Beijing 100871, China)

Abstract: Quality of method names is critical for the readability and maintainability of program. However, it is difficult for software engineers, especially non-English speaking, inexperienced engineers, to propose high quality method names. To address this issue, this paper proposes an approach to recommend method names. First, a method corpus is constructed from open source applications. For a given method *f* to be named, similar methods are retrieved from the method corpus. Names of these retrieved methods are divided into phrases, and features of these methods are extracted as well. A mapping between these phrases and features is also created to derive a list of candidate phrases and features for the method to be named. These phrases are finally constructed into candidate method names. The

* 基金项目: 国家自然科学基金(61272169, 61472034, 61003065, 61371194); 国家重点基础研究发展计划(973)(2013CB329303); 教育部新世纪优秀人才支持计划(NCET-13-0041); 北京高等学校青年英才计划(YETP1183); 新闻出版重大科技工程项目(GAPP_ZDKJ_ BQ/01)

Foundation item: National Natural Science Foundation of China (61272169, 61472034, 61003065, 61371194); National Program on Key Basic Research Project of China (973 Program) (2013CB329303); Program for New Century Excellent Talents in University of Ministry of Education of China (NCET-13-0041); Beijing Higher Education Young Elite Teacher Project (YETP1183); Major Scientific and Technological Projects of Press and Publication, China (GAPP_ZDKJ_ BQ/01)

收稿时间: 2013-12-03; 定稿时间: 2015-01-08

proposed approach is evaluated on 1 430 methods in open source applications. Evaluation results suggest that 22.7 percent of recommended method names are the same as original ones, and 57.9 percent has the same or almost the same keywords as original ones.
Key words: method name; recommendation; feature selection; algorithm; natural language processing

软件维护成本约占软件总成本的 70%[1].软件代码难以理解,是导致高成本的主要原因.研究表明:在软件维护过程中,维护人员要将超过 50%的精力放在对软件代码的理解上[2].因此,提高程序的可读性和可理解性,是有效降低软件维护成本的途径之一.

影响软件可读性和可理解性的主要原因是软件实体(类、函数、变量)名称难以理解.软件代码中的 70%由实体名称组成[3].好的软件实体名称能够如实地反映该实体所代表的内容,能够使程序人员望文知义而不需深入实体内部或查找关联的文档.目前,由于代码只能使用相应的编程语言按照编程规范编写,供机器识别而不能完全按照人的思维模式进行描述,这就需要程序人员花费精力去实现该转换过程.如果可以直接从有意义的实体名称而非繁杂的代码或关联的文档获取程序的信息,工作效率就会得到有效的提高,从而降低软件维护成本.实体名称的重要性已经得到证实.Deissenboeck 和 Pizka[3]提出,合适的实体名称对于程序的理解非常重要.Caprile 和 Tonella[4]通过分析函数名称的结构发现,实体名称是获取理解程序实体信息的重要资源之一.Butler[5]通过统计数据证实:无论在类级别还是在函数级别,低质量的实体名称与低质量的程序代码有直接的关系.

虽然研究证明了实体名称质量的重要性,但在实际的应用过程中软件实体命名的质量并不高,尤其是对函数的命名.通过对开源项目中函数的分析发现,许多函数的命名并不合适.例如:在开源项目 Piccolo(http://sourceforge.net/projects/piccolo/)(版本 1.0.4)中,函数 *valpush*()不能直接反映出该函数要实现的功能,若改函数名为 *createArrayForString*(),会使该函数的功能一目了然.在项目 TeXlipse(http://sourceforge.net/projects/texlipse/)(版本 1.4.1)中,名为 *determineSourceFile*()的函数功能是返回选取的资源文件,那么使用 *getSelected SourceFile*()作为函数名称则更为合适;同样,在项目 itext (http://sourceforge.net/projects/itext/)(版本 5.06)中,函数 *forBits*()的功能是获取当前模式类别,使用 *getModeTypeByBits*()名称更为合适.类似此种情况在其他项目中还有很多,函数名称与功能不匹配会直接影响对软件代码的理解[6,7].

导致函数功能与名称不匹配的主要原因有两个:第一,开发人员迫于开发时间的压力只注重如何实现函数的功能而不愿花更多的时间为函数选择合适的名称;第二,程序的开发主要基于英语实现,开发人员的母语并非英语,缺乏选取合适英语词汇为函数命名的经验.一项网上调查显示:49%的人认为,给实体命名是一件很棘手的事情[8].此前我们的研究[9]也发现:程序人员实施的重构操作中,最为频繁的是针对不合适命名的操作.为了改善函数命名不合适的状况,研究者提出了一些方法,如:Kuhn[10]提出针对如何建立名称推荐系统应采取的策略进行了讨论;Deissenboeck 和 Pizka[3]通过概念与名称间的映射模型,为简明和一致性命名定义了更为精准的规则;Host 和 Ostvold[11]抽取并总结出一套常用的动词词汇,辅助用户选择函数名称中合适的动词等.这些研究主要着重于:(1) 从理论上讨论如何改善命名现状;(2) 从规则入手对命名进行约束;(3) 部分解决不合理选用词汇命名的问题.已有的研究都不能直接为程序人员提供命名帮助,只能间接为程序人员提供指导或需要程序人员间接地实现,同样需要程序人员花费较高的代价.

为此,本文提出了一种能够根据函数功能自动生成与之相匹配的名称方法,为函数命名提供直接的帮助.该方法一方面降低了程序人员开发的代价,提高了函数命名质量;另一方面间接提高了软件代码的可读性和可理解性,从而降低软件维护的代价.该方法已经在开源项目上进行了验证,结果表明:有 22.7%的推荐结果与原函数名完全一致,有 57.9%的推荐结果与原函数名关键词一致或基本一致.

本文第 1 节介绍相关的技术背景.第 2 节对提出的方法进行详细描述.第 3 节通过实验对提出的方法进行验证并对结果进行分析.第 4 节讨论影响实验的各种因素及所提方法的局限性.第 5 节总结相关的研究工作.第 6 节给出结论.

1 技术背景

1.1 鹅卵石编码算法

鹅卵石编码(shingles encoding)[12]技术主要应用于信息检索,该技术的特点是:如果两段文本差别较小,则生成的 shingle 变化也较小.因此与传统文本匹配技术相比,该技术具有更好的鲁棒性.

本文中,Shingles Encoding (http://alexnl.3vkj.net/AlgorthmAndExampleOfShinglesEncoding.doc)算法用于计算每个函数的指纹,并度量两段代码的相似度.算法中的两个参数:W 表示创建 shingle 的窗口大小,决定每个 shingle 有几个 token 组成;S 表示结果集的容量,代表 shingle 描述空间的上边界.当给定一文本,算法按照滑动窗口 W 查找该文本所有的子序列,并为每个子序列计算 shingle,从子序列中选取 S 个 shingle 放入结果集中.用 U 代表 shingle 集合的长度,可以用公式(1)计算 S:

$$S_{\min}(U)=\begin{cases}U\text{中的}s\text{个最小元素}, & \text{if } |U|\geqslant S\\ |U|, & \text{其他}\end{cases} \tag{1}$$

1.2 词性标注

词性标注是自然语言处理(NLP)领域的一项基础性研究,是语料库建设的重要组成部分.Stanford PoS Tagger(http://nlp.stanford.edu/software/stanford-postagger-full-2011-09-14.tgz)[13,14]是一款词性标注工具,使用宾州树库标注集对词组中的单词词性进行标注,如名词、动词、副词等.它是基于对数线性模型用 Java 实现的一款词性标注工具,并广泛应用于当前自然语言处理领域.在所提的方法中,该工具用于对分割后的函数名称词组进行词性标注.

1.3 χ^2统计方法

特征选取是从原始属性集合中选取最具有代表性属性子集的活动,我们使用χ^2 统计(CHI)[15,16]进行特征代码选取.该方法假设在特征项 t 与类别 c 之间的非独立关系类似于具有一维自由度的χ^2 分布,可用于衡量特征 t 和类别 c 之间的统计相关性.用 N 表示文本的总数,c 表示某一特定类别,t 表示某一特定特征,A 表示包含特征 t 且属于类别 c 的文本频数,B 表示包含特征 t 但不属于类别 c 的文本频数,C 表示属于类别 c 但不包含特征 t 的文本频数,D 表示既不属于 c 也不包含特征 t 的文本频数,则χ^2 值可以由公式(2)计算:

$$\chi^2(t,c)=\frac{N(AD-BC)^2}{(A+C)(B+D)(A+B)(C+D)} \tag{2}$$

χ^2 统计值越高,说明特征 t 与类别 c 之间的相关性就越大.当特征 t 与类别 c 不相关时,则有$\chi^2(t,c)=0$.

2 名称推荐方法

2.1 方法概述

如图 1 所示,名称推荐方法包括 4 个主要部分:检索相似函数、函数名称解析、特征代码抽取和名称推荐.

待推荐函数的函数体作为所提方法的输入,最终获取推荐的名称列表.首先,从函数库中检索与输入函数相似的函数.然后,将检索出的相似函数分为两部分处理:使用自然语言处理工具对函数名称进行语法解析并对相应的词条进行标注,从检索出的函数代码中抽取特征代码并在词条与特征代码之间建立映射关系.最后,从输入函数中检索相应特征代码,根据特征代码选定词条,并按照规则重组后推荐给用户.在后面的章节中,将逐一介绍所提方法的各个部分.

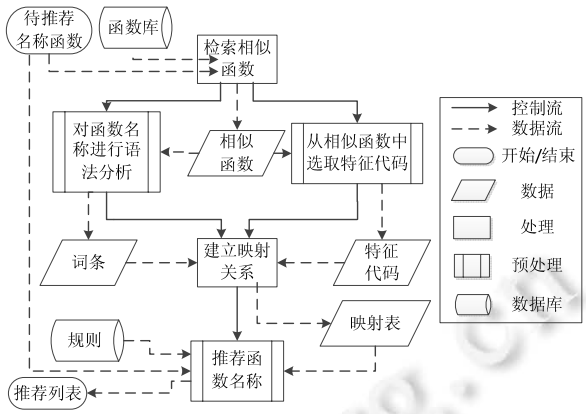

Fig.1 Flowchart of the approach

图 1 方法流程图

2.2 检索相似函数

在此过程中,我们从函数库中检索与输入函数相似的函数,并进行解析.具体的步骤如下:

- 将输入函数和函数库中的函数解析成抽象语法树(AST);
- 将抽象语法树的节点序列化为字符串;
- 使用 shingle encoding 算法对字符串进行标记并生成 shingle (shingle 通过 MD5(http://tools.ietf.org/html/rfc1321)算法得出(http://alexnl.3vkj.net/AlgorthmAndExampleOfShinglesEncoding.doc));
- 使用 Jaccard 系数对节点进行相似度度量;
- 将相似节点封装成节点对用以进行语义分析.

我们用 T_{m1} 和 T_{m2} 分别表示两个相似的 AST 方法节点,用 $S(T_{m1})$和 $S(T_{m2})$分别表示根据两种方法节点计算出的无重复 shingle 的集合.Jaccard 系数可以通过公式(3)计算:

$$Sim(T_{m1},T_{m2})=\left|\frac{S(T_{m1})\cap S(T_{m2})}{S(T_{m1})\cup S(T_{m2})}\right| \tag{3}$$

选取相似函数主要的算法如图 2 所示.检索过程结束之后,与输入函数相似的函数被置入集合 S_m 中.

```
1   Input: input method(m), method corpus (M);
2   Output: A set of similar methods (Sm).
3   Procedure: MethodMatching
4       For m, mi in M
5           Compare m, mi;
6           Find Matched Test Method Pairs (m, mi);
7           Sm.add(mi);
8       End
```

Fig.2 Algorithm for selecting similar methods

图 2 相似函数选取算法

2.3 函数名称解析

函数名称本身包含有价值的信息,好的函数名称可以直接反映函数的功能.函数名称由单个或多个单词按照相应的语法组成,来表达函数所要执行的动作或返回的信息.一些编程语言如 java,其命名规范要求命名函数时,首单词小写其他单词首字母大写.因此,我们采用驼峰拼写分隔符将函数名称分割成词组.例如,函数名 createFileBySelectedString 经过分割后成为 create File By Selected String.在分割的过程中,函数名称中会有一些非字母的符号,如数字、$等.研究表明,这种情况较少[11]且对于描述函数功能没有指导意义.因此,我们在分割的过程中将这类符号从词组中去除.另外,我们建立一个专用词词典(http://alexnl.3vkj.net/dictionary.rar)用于记录

专用词或计算机场景下的特定词条,用于提高分割的正确性.原函数名称包含的一些特定类别或缩略词等有助于识别,例如 FileList,KeyEvent 和 XML 用于描述特定事物或实体.

Stanford PoS Tagger 可对分割后词组中的单词进行词性标注,这些标记表明了该单词在词组中的词性,如名词、动词、副词等.为了提高标注的准确性,我们将一些技术领域的专有词,如“refactoring”,“hashcode”,“TDD”等补充到 WordNet 词典中,用于帮助在标注过程中识别一些未知词条.

2.4 特征代码抽取

函数的特征指函数中一部分特有的代码,这些代码能够体现该函数的特有行为并用以区分其他函数.在此过程中,我们对检索得到的相似函数进行语义分析,并针对不同的函数抽取特征代码.

我们以程序语句为单位将函数 f 进行分解,并假设函数中的每条语句都是该函数的一段特征代码,则函数名称分解出的词条与每条程序语句之间具有一定的相关性.用 c_m 代表函数名称中经过标注的词条,用 t_n 代表每段特征代码,则每个函数 f_i 可以表示为$(c_1,c_2,\ldots,c_m)\{t_1,t_2,\ldots,t_n\}$.不同的词条 c_i 和特征代码 t_i 存在于不同的函数中,每个词条 c_i 都有不同的特征代码 t_i 与之相对应,相同的 t_i 可能同时对应于不同的词条 c_i.词条 $c_i(i\in m)$和特征代码 $t_i(i\in n)$之间的相关性可以通过公式(2)计算得出,计算得出的数值越高,词条与特征代码的相关性越高.设定相应的阈值,当该值高于阈值时保留相应的特征代码,反之则丢弃.

特征选择的整个过程分为 3 个步骤:(1) 分解函数,以$(c_1,c_2,\ldots,c_m)\{t_1,t_2,\ldots,t_n\}$的方式表示;(2) 比对检索得出的函数,若 t_i 或者与 t_i 相似的代码多次出现在不同的函数中,则将该代码 t_i 作为候选的特征代码对象,并用二元组 $f(e,f_e)$表示.其中,e 代表函数间共同的或相似的代码 t_i,f_e 代表该代码出现的频次;(3) 在标注的词条与候选的特征代码对象之间建立关联.我们用四元组 $r(t,w,f_r,f)$来表示该关联关系,t 代表词条的标签,f 代表特征代码对象, w 代表词条与特征代码之间关系的权重,f_r 代表词条与特征同时出现的频次.其中,w 的值可以通过公式(2)得出, f_r 的值可以通过统计得出.最终,在词条与特征对象之间建立多对多的关系.

图 3 举例说明了从两个函数中选取特征代码的过程,图 3 中左右两个函数都是针对数组进行的操作.首先对函数进行分解,用 $t_i(i$ 表示语句的行号)表示函数中的特征代码,则两个函数可以分别表示为(create,Array)$\{t_4,t_7,t_8,t_{11}\}$和(expand,Array)$\{t'_3,t'_4,t'_6,t'_7\}$.在比对这两个函数的过程中我们发现,它们都包含相关数组创建(t_7 和 t'_4)、数组复制(t_8 和 t'_6)及数组返回(t_{11} 和 t'_7)的相同或相似代码,因此将这 3 条语句分别作为候选的特征代码.这些特征代码与函数名称分解出的词条(create,expand,Array)相关联.当函数中出现该特征代码时,可以选用这些对应的词条来组成函数名称.

```
1  private String[] createArray(string[] array, int
2                               desiredSize){
3    if (array==null) {
4      return new String[desiredSize];
5    }
6    if (array.length<desiredSize) {
7      String[] newArray=new String[desiredSize];
8      System.arraycopy(array,0,newArray,0,
9                       array.length);
10   }
11   return newArray;
12 }
```

```
String[] expandArray(String[] array, int
                     increment){
  int oldSize=array.length;
  String[] newArray=new String[array.length+
                               increment];
  System.arraycopy(array0,newArray,0,oldSize);
  return newArray;
}
```

Fig.3 An example of feature selection

图 3 特征代码选取举例

2.5 名称推荐

如图 4 所示,通过将候选特征代码对象与输入函数比对,依据特征代码对象与词条的对应关系选出用于组成函数名称的词条,然后按照规则对词条进行重组推荐给用户.

候选的词条将被分为名词和动词两类,并非所有的词条都适合用来组成函数名称.我们根据相应的规则从候选词条中选取单词并组成动词+名词的函数名称形式.词条选取的优先原则主要依据 3 方面因素:

(1) 特征对象出现的频率.输入函数中可能包含多个特征对象,优先选取频次高的特征对象对应的词条;

(2) 特征对象与词条关系的权重.特征对象与词条之间存在多对多的关系,对于单个特征对象对应的多个词条来说,权重高表明特征对象与该词条之间的关联程度更高;

(3) 词条的语义相关度.在选择词条时,我们主要考虑前两种因素,但如果通过前两种因素比较得出的结果相同,考虑到输入函数原始名称具有一定的借鉴意义,我们将候选词条与输入函数初始名称分解出的词条相比较,采用意义相近的词条.

WordNet Similarity(http://www.d.umn.edu/~tpederse/similarity.html)是一款开源工具包,用于度量 WordNet 中词条的相似度和相关度,使用该工具来度量词条之间的语义距离.但在实验中,需要考虑第(3)种因素的情况非常少.整个词条筛选的过程完毕,筛选后的动词和名词被组合成动词+名词的形式推荐给用户.

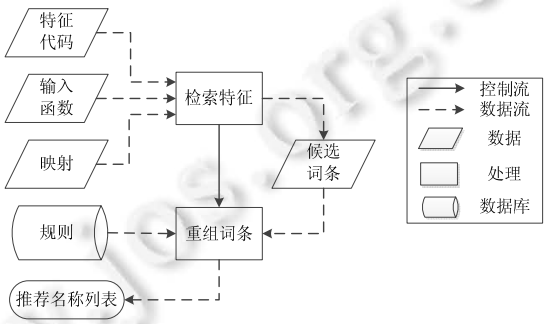

Fig.4 Overview of recommendation process

图 4 名称推荐示意图

3 实验验证

我们将所提方法应用在开源项目上,以验证该方法的有效性.首先,选取合适的开源项目用以建立函数库;其次,选取开源的项目作为验证对象,并从中分离出待验证的函数;然后,将所提方法应用于待验证的函数推荐函数名称;最后对推荐结果的可用性进行评估,以验证所提出方法的有效性.

3.1 实验准备

为了验证所提方法,我们选取了两个中等规模的开源项目作为验证对象 jEdit (http://alexnl.3vkj.net/TextEditor1.2.rar)(版本 4.3.2)和 text_editor(http://www.jedit.org/)(版本 1.2).text_editor 是一款功能强大的文本编辑组件,我们从中分离出 255 个类包含 295 个函数用于实验;jEdit 是一款成熟的文本编辑工具,我们从中分离出 323 个类包含 1 135 个函数用于实验.每个项目的具体信息见表 1,统计所得出的数据是通过工具 LineCount (版本 3.7)(Linecount.http://liangs.autodebug.com/myfiles/linecount3.7.rar)获取的.从项目的介绍可知,这两个项目虽然属于相同类型,但由不同的开发者完成.项目的规模跨度较大(从 14.5KLOC~29.1KLOC).基于这些项目,可以为获取真实的实验数据提供有效的保证.

Table 1 Basic information of evaluation projects

表 1 待测项目基本信息

项目名称	版本	类数量(个)	函数数量(个)	代码行数(LOC)
jEdit	4.3.2	323	1 135	29 051
text_editor	1.2	255	295	14 554

基于开源项目建立函数库,为输入函数推荐名称.我们选取了 55 个开源的项目(http://alexnl.3vkj.net/BasicInformationOfExperimentProjects.doc),并用这些项目构建函数库.首先,通过句法分析将这些项目中的类解析成为抽象语法树(AST),抽象语法树的节点包含了实体(包、类、函数等)的相关信息,我们只关注其中与函数

相关的信息,并将函数抽取出来组建函数库.对于其中可能存在重复函数的情况,则使用 CCFinderX (http://www.ccfinder.net/ccfinderx.html)对重复的函数进行检测并去除.最终建成的函数库中包含 53.3 万个函数.

函数名称推荐的结果(http://alexnl.3vkj.net/RecommendedMethodname.xlsx)可能出现如下几种情况:

(1) 与原函数名完全一致:推荐的函数名称与原函数名称相同;

(2) 与原函数名关键词一致:推荐的函数名称与原函数名称具有相同的关键词条,但其附加词或关键词的顺序不一致;

(3) 与原函数名关键词基本一致:推荐的函数名称与原函数名称的关键词条大体相同,仅有一个词条不一致;

(4) 与原函数名中的关键词相差较大:至少有两个词条不一致(如果函数名仅有一个词条,则推荐的函数名称与原函数名无共同关键词).通过对函数实体进行分析,判断新旧名称的优劣,继而分为两类:优于原函数名以及略于原函数名;

(5) 无推荐名称:通过该方法没有找到与函数相匹配的名称.

分类中所描述的关键词指的是名称中主要的动词和名词,该动词表示函数所要执行的动作,名词表示函数动作执行或被执行的主体.实验中仅对第(4)类情况进行人工分析,对前 3 类情况没有进一步的人工分析.其原因在于如下两个方面:

- 首先,函数名称的优劣比较非常困难.因为本实验采用知名开源软件作为实验对象,所以参与实验的人员并非相关软件的作者,对目标代码并不熟悉.要比较函数名称的优劣就必须完全理解相关代码的功能和结构,具有相当的难度,人工判定的准确性也难以保证;
- 其次,本实验涉及数千个函数,如果全部进行人工分析,则需要大量的人力和时间.而第(4)种情况的数量不多(约 8.95%),所以便于人工分析判定.

3.2 实验过程

协助参与本次实验的是有一定开发经验的在校学生,如图 5 所示,针对每个待验证的函数,按照以下步骤进行实验:

- 使用选定的开源项目构建函数库;
- 依次将所提方法应用于每个待验证的函数,为每个函数推荐名称;
- 对推荐的结果进行评估和分类;
- 统计各分类结果并计算该方法的成功率.

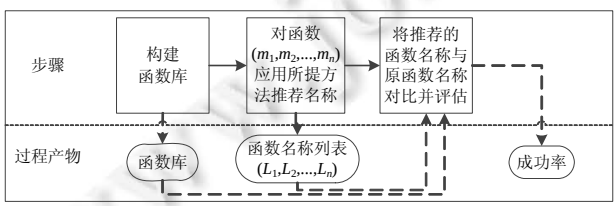

Fig.5 Evaluation process

图 5 实验过程

3.3 实验结果与分析

3.3.1 所提方法的可用性

将 jEdit 中的 1 135 个函数和 text_editor 中的 295 个函数分别作为输入来验证所提方法,实验结果见表 2.

从表 2 中可以看到:对于 jEdit 项目中的 1 135 个函数,应用所提方法所得到的推荐结果中有 23.74%与原函数名完全一致,38.59%与函数名称关键词一致;对于 text_editor 项目中的 295 个函数,得到的推荐结果中有

18.98%与原函数名完全一致,39.66%与函数名称关键词一致.对于人工分析的结果中,两个项目中可替代原函数名称的分别占 3.35%和 5.08%.鉴于第(2)类和第(3)类推荐的结果中出现了可用于命名的关键词,均有助于改善对函数的命名,加之通过人工分析所得出的优于原函数名称的部分,超过 80%的推荐结果认为是有效的.两个项目中,有 14.71%~19.33%的推荐结果没有达到我们的期望值.

我们分析了对应关键词基本一致部分的函数,通过比对检索得出的相似函数发现:它们所实现的功能基本相似,但是实现功能的方式发生了变化.某些函数的功能是通过委托函数来实现,即函数的某些子功能通过调用其他函数实现.在进行特征代码匹配的过程中,该部分会被遗漏,其对应的词条也无法进入最终的候选词条列表中,因此,在推荐函数名称时无法使用合适的词条对函数进行准确的描述.例如,在类 fileSaveService.java 中的函数 *saveFile*(),其功能是当文件不存在时创建并写入当前字符串,否则将当前字符串附加在文件尾.经过检索后,得到 6 个函数分别是 *saveToFile*(),*writeToFile*(),*save*(),*appendToFile*(),*writeFile*()和 *writeDocument*(),它们所实现的功能与 *saveFile*()类似.抽取的特征代码包含两部分内容:一是对文件检查,二是写文件.在将特征代码与函数 *saveFile*()匹配时只保留了写文件部分的特征代码,因为对于文件检查部分函数 *saveFile*()是通过调用另外一个函数 *checkFileStatus*()来实现的.因此,在名称推荐时不能准确地使用相应词条.该问题最有效的解决方法是通过内联函数,即,用被调用函数替代函数调用语句.但也会由此引入新的问题,如破坏函数的封装性、降低程序的灵活性、造成过长函数过大类等代码坏味.

Table 2 Evaluation results of two projects

表 2 实验结果

项目名称	实验结果		函数个数	所占百分比(%)
jEdit	完全一致		269	23.70
	关键词一致		438	38.59
	关键词基本一致		223	19.65
	差别较大	优于	38	3.35
		略于	55	4.85
	无结果		112	9.86
text_editor	完全一致		56	18.98
	关键词一致		117	39.66
	关键词基本一致		50	16.95
	差别较大	优于	15	5.08
		略于	20	6.79
	无结果		37	12.54

我们同时也对无返回结果和经人工分析略于原函数名称的函数进行了分析,造成该结果的原因有两个:

第一,待验证函数比较独特,在进行检索时没有找到与之相匹配的函数.由于这个原因,在 jEdit 项目中有 112 个函数未得到合适的推荐名称,占总数的 9.86%;在 text_editor 项目中有 37 个函数未得到合适的推荐名称,占总数的 12.54%;

第二,Stanford PoS Tagger 对函数名称的错误解析和标注导致词条错误分类,造成推荐名称不准确.在多数情况下,该工具对词组的解析和标注是正确的,但也有例外.例如,对于函数名 actionPerformed,分割后形成的词组为 action Performed.Stanford PoS Tagger 对其解析和标注的结果为 action/NN 和 Performed/NN,名词词组.Performed 实际上是对 action 的状态描述,因此应该标注为 Performed/VBD 或 Performed/JJ.但是如果该名称改为 performAction,则 Stanford PoS Tagger 可以对其进行正确解析和标注.

对词条的错误分类,造成推荐的函数名称不适用.由此原因,在 jEdit 项目中有 55 个函数未得到合适的推荐名称,占总数的 4.85%;在 text_editor 项目中有 20 个函数未得到合适的推荐名称,占总数的 6.79%.

3.3.2 实验阈值的选取

实验中,需要确定检索相似函数及特征代码选取时用到的相似度阈值.检索相似函数阈值决定了从函数库中匹配到的函数的数量,该值越大,从函数库中匹配到的函数越少;反之,则匹配到的函数越多.匹配到的函数的

数量会影响整体方法执行的效率.特征代码选取阈值则影响抽取的特征代码与标注的词条之间的关联关系,该值越小,获取的候选标注词条越多;反之,则获取的候选词条越少.这两个值都会直接影响到推荐结果的好坏,从而影响方法的准确性.下面我们针对每一单独阈值的变化进行分析.

首先,检索相似函数阈值会同时影响方法执行的效率和方法的准确性.因此,在选取该阈值时应同时考虑两方面的因素.我们通过实验对该值进行调整,分别得出该阈值与方法执行时间的关系(如图 6 所示)和该阈值与方法准确度之间的关系(如图 7 所示).

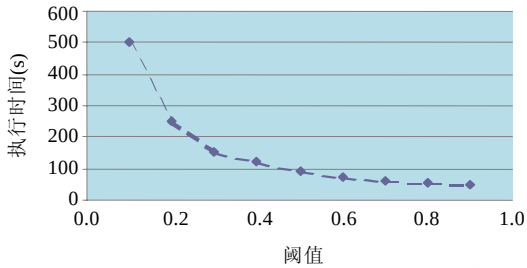

Fig.6 Relationship between threshold and execution time

图 6 阈值与执行时间的关系图

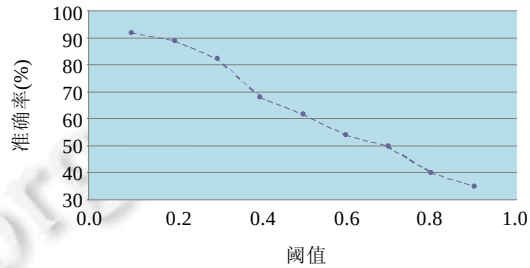

Fig.7 Relationship between threshold and accuracy

图 7 阈值与准确率的关系图

从图 6 中可以看到:当阈值增大时,方法的执行时间逐步下降,但下降的幅度逐步变缓;当阈值较小时,通过相似度匹配得到函数数量较大,用于分析函数名称和进行特征代码选取的工作量增大,需要较多的时间进行处理.图 6 中,当阈值接近 0 时,处理所需的时间超出了正常可容忍的范围.

从图 7 中可以看到:当阈值增大时,方法的准确度在不断下降.阈值不断增大,导致从函数库中匹配到的函数减少,一些包含相应特征代码的函数被过滤.在进行特征代码选取时,特征代码的缺失会导致相应候选词条的缺失,能够用于推荐合适名称的词条不在候选词条列表中,导致方法推荐名称失效.

综合图 6、图 7,当阈值大于 0.3 时,方法执行的时间减少速度降低;阈值从 0.3 变化到 0.4 时,方法的准确率陡然降低.为了更好地在方法执行时间和方法准确率之间获取平衡,选取 0.3 作为检索相似函数的阈值.

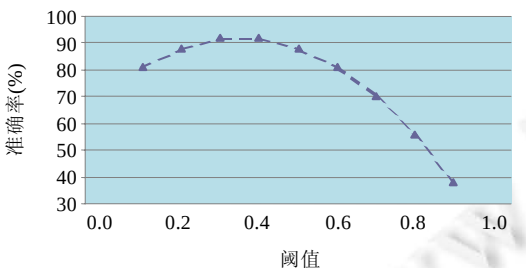

Fig.8 Relationship between threshold and accuracy time

图 8 特征代码选取阈值与准确率的关系图

其次,特征代码选取阈值,该阈值主要确定是否将特征代码保留并与标注词条相关联.通过实验,我们获取了该阈值与方法准确度之间的关系(如图 8 所示).可以看到:当阈值小于 0.3 时,方法的准确度在不断增加;当阈值大于 0.4 时,方法的准确度在不断降低.当阈值过小时,筛选得出的部分特征代码与标注词条之间的关联性较小,导致一些冗余或无用的词条保留在候选词条列表中,从而降低方法的准确度;当阈值过大时,一些可用于组成合适名称的词条被过滤,词条的缺失造成方法准确度的降低.综合两方面的因素,我们选取 0.35 作为特征代码选取阈值.

4 讨 论

4.1 实验人员

本次协助进行实验的人员是具有一定开发经验的在校学生,他们在校期间一直从事软件工程方面的研究并参与软件项目的质量评估,具有相应的基础.在实验的过程中,实验数据的收集主要通过所提方法获取,最大程度上减少了人为的主观因素.他们经过多年系统的英语培训,熟练掌握英语语法及词汇的应用,这样有助于对

实验结果中函数名称的好坏进行判断.

4.2 项目的选取

开发人员各自的习惯不同,从而使得开发的项目带有个人特征(程序结构、命名方式和特定函数等).为了降低这方面的影响,实验中用于构建函数库的项目以及待验证项目都是从 SourceForge 上选取的开源项目.这些项目分属不同领域,并由不同的开发者完成,代码覆盖各种规模.基于这些项目,可为获取真实的实验数据提供有效的保证.

4.3 自然语言处理工具

我们综合比对了一些标注工具[17,18],Stanford PoS Tagger 标注的准确度,尤其是对未知词条标注的准确度相对较高.因此,在实验中选取该工具来分析函数名称词组的语法结构并进行标注.它是建立在斯坦福语料库基础上的,但由于许多应用于技术领域或专有的词条并未收录入该词库,在进行词性标注时会将一些词条标注为未知,从而降低标注的准确性.为此,我们将一些应用于技术领域或专有的词条与 WordNet[19]中的词条合并建立新的词典(http://alexnl.3vkj.net/dictionary.rar),用于改善函数名称分割效果和词组词性标注的准确性.专家和学者也提出了一些方法有助于改进分析和标注的效果,如:Abebe 和 Tonella[20]提出通过组成名称的词汇构建句子,然后对句子进行依存分析可以提高标注的准确性.在此基础上,Binkley 等人[21]使用 4 种模式对 Stanford PoS Tagger 进行测试,并根据这 4 种模式提出了改进命名结构的 4 条规则;Binkley 等人[22]还针对类名提出了一种新的 Stanford PoS Tagger 模型.实验结果表明,该模型能够有效提高标注的准确性.

4.4 特征选择算法

特征选取通过机器学习和统计方法选取一些相关特征的子集以建立适用的学习模型,通常在文本分类研究领域用以提高准确度.主流的算法仍是一些传统的算法,如信息增益(IG)[23,24]、χ^2 统计(CHI)[15,16]、文本频次(DF)[25,26]和互信息(MI)[27,28]等.鉴于χ^2 统计方法比其他方法更为有效[28,29],我们使用该方法进行特征代码选取.

4.5 名称结构

所提的方法中,采用 Stanford PoS Tagger 对函数名称进行解析和标注.通过使用宾州树库中的标记,区分名称词组中每个词条的语法角色,如名词、动词和副词等.所提方法推荐的函数名称采用了名词+动词的结构,主要是由于使用该结构形式比较简单但能够清楚地对函数进行描述.研究[30]显示,动词+名词/形容词/副词这种结构在总的名称结构中的比例超过 70%.

4.6 方法改进

此次实验中,我们通过人工方式选取类型相同的开源项目构建函数库.一方面可以保证最大程度地获取对同种类型函数的支持,以便对所提方法进行有效验证;另一方面,既定的函数库有利于对实验过程进行有效的分析.然而在实际的应用过程中,这种方法会造成人力成本的增加,并且与开发人员的习惯不符.目前,随着网络技术的发展,在网络上存在大量各种类型的开源项目.因此,借助网络技术直接构建函数库并提供函数名称推荐支持,可以有效地节约人力成本.另外,实验结果显示,所提方法在推荐名称时所用的时间较长.经分析,方法运行时间的瓶颈主要在于代码特征的搜索和匹配.因此,改进搜索及匹配的算法能有效地提高方法的运行效率.

5 相关工作

目前,许多研究都证实了命名质量与代码的可读性和软件质量有密不可分的关系.一些学者从实证研究出发对该问题进行了论证.Caprile 和 Tonella[4]通过词法、句法和语义结构对 10 个 C 语言程序中的函数名称进行了分析,他们将名称分解成单词并使用语法找出词之间的语义关系.实验证明:好的函数名称能够促进程序理解,利于程序的演化和维护.Butler 等人[8]分别对 8 个开源项目中的标识符名称质量和源代码进行了评估,并使用χ^2 和费舍尔抽取测试对两者之间的独立性进行了计算,发现低质量的标识符名称会使源代码更为复杂,使其难以理解和维护.随着受控实验在计算机科学领域的流行,一些研究者通过受控实验的方式对该问题进行了研究.

Lawrie 等人[31]将函数命名的方式分为 3 种,即:完整单词、缩略语和单字母.然后,通过大规模的问卷调查形式对数据进行收集.通过对统计数据的分析得出结论:源代码中使用包含字典词汇的标识符比使用缩写或单个字母的标识符更容易理解.Blinman 等人[6]通过分组对比实验的方式对框架接口命名形式对应用程序理解的影响进行了研究,结果证明,使用具有描述性的接口名称比使用非描述性接口名称更能有效地帮助开发人员理解程序.另外,Butler[5]综合自己前期的研究和近年来关于命名质量对代码质量影响的研究,认为低质量的标识符名称是致使低质量代码的直接原因,从而导致了代码可阅读性差.

从以上的研究看出,目前亟需相关的方法或工具来支持高质量的标识符命名.一些研究者着重从命名规则入手进行研究.Deissenboeck 和 Pizka[3]认为,命名规则对于加强程序一致性和指导将概念转换为名称非常必要.因此,他们基于概念和名称的映射关系创建了一套模型,可以用于创建清晰的标识符名称.他们在该模型中引入名称词典,用以保证实体名称在软件生命周期内保持一致.林秋申等人[32]从提高 C/C++语言的可维护性、可理解性和健壮性的角度讨论了如何制定标识符的命名规范问题.曹娜[33]通过研究代码质量与代码整洁度的关系,总结出一套如何保持代码整洁的方法,其中也包括了如何进行有效的命名.运思婧[34]从标识符词性组成的规则角度出发,制定了符合 Java,C,C++使用情况的软件标识符词性规则,为评价标识符质量提供了新的思路和方法.还有一些研究者从实际应用出发提出了一些方法.Host 和 Ostvold[11]结合 java 函数命名规范,针对大量应用程序进行了研究.他们着重于构成函数名称首单词的动词,认为动词是构成名称结构的重要组成部分.因此,他们抽取并总结出一套常用的动词词汇,对这些动词的应用场景进行相应的描述,有助于用户在函数命名时进行选择.文献[35]在此基础上提出一种函数命名缺陷的检测方法,从数据流和控制流中抽取的相关属性来检测名称与实体内容的符合性,从而确定命名是否存在缺陷.另外,Kuhn[10]提出一套推荐系统用于帮助软件开发人员在编码过程中对实体命名,并就如何建立该系统及在建立该系统应采取的策略进行了讨论.他的研究思路对我们所提出的方法给予了一定启发.

还有一些研究将命名技术应用于改善软件维护.Rajlich 和 Wilde[36]认为,标识符名称是从程序中获取概念信息和理解程序最主要的资源.他们提出提出基于名称概念识别的方法用于概念定位,该方法可以避免由于开发人员单纯依靠经验对实体进行命名从而使得软件在不断演化中失去其原有的意义.Hill[37]提出一种新的模型,同时使用文本和结构信息来改善软件搜索和软件开发工具.该模型通过有效地查找和理解程序代码,能够有效地降低软件维护的成本.Thies 和 Roth[38]提出一种方法,通过对 java 程序中变量赋值的分析保持名称的一致性,该方法解决了不同人员同时进行程序开发时对程序实体命名不一致的问题.

虽然已有研究对于解决实体命名方面做了大量的工作,但尚未有能够直接为用户提供命名支持的方法和工具.一些研究着重于对命名规则进行研究[3,32−34],仅能间接地为程序人员命名提供参考.Host 和 Ostvold[11]总结出的常用动词列表只能部分地解决用户不能合理选用词汇命名的问题.Caprile 和 Tonella[4]针对定义的用于命名的语法结构就如何应用进行了展望,没有提出具体应用.Deissenboeck 和 Pizka[3]提出的模型则需要专家手工在标识符与域概念之间建立映射关系.

6 结 论

代码难以理解是造成软件维护成本高的一个主要原因,为实体选用易于理解的名称有助于降低软件的维护成本.然而,已有的研究工作未有能向用户提供直接命名支持的方法和工具.为此,本文提出了一种能够根据函数功能自动生成与其相匹配的名称的方法,为函数命名提供直接的帮助.该方法一方面降低了程序人员开发的代价,提高了函数命名质量;另一方面间接提高了软件代码的可读性和可理解性,从而降低软件维护的代价.该方法基于开源软件创建函数库,对于某个需要推荐名字的函数 f,从函数库中检索与其相似的函数.对检索返回的相似函数用自然语言处理工具对函数名进行解析,并获取标注词条.然后,从相应的函数体中提取特征代码并与相应的标注词条建立关联.基于此关联关系以及函数 f 的特征,自动推荐合适的函数名.我们从开源项目中选取了 1 430 个函数对所提方法进行验证,结果表明:有 22.7%的推荐结果与原函数名完全一致,有 57.9%的推荐结果与原函数名关键词一致或基本一致.

References:

[1] Boehm B, Basili VR. Software defect reduction top 10 list. Computer, 2001,34(1):135−137. [doi: 10.1109/2.962984]

[2] Von Mayrhauser A, Vans AM. Program understanding behavior during debugging of large scale software. In: Proc. of the Papers Presented at the 7th Workshop on Empirical Studies of Programmers. New York: ACM Press, 1997. 157−179. [doi: 10.1145/266399.266414]

[3] Deissenboeck F, Pizka M. Concise and consistent naming. Software Quality Journal, 2006,14:261−282. [doi: 10.1007/s11219-006-9219-1]

[4] Caprile C, Tonella P. Nomen est omen: Analyzing the language of function identifiers. In: Proc. of the 6th Working Conf. on Reverse Engineering. 1999. 112−122. [doi: 10.1109/WCRE.1999.806952]

[5] Butler S. The effect of identifier naming on source code readability and quality. In: Proc. of the Doctoral Symp. for ESEC/FSE on Doctoral Symp. (ESEC/FSE Doctoral Symp. 2009). New York: ACM Press, 2009. 33−34. [doi: 10.1145/1595782.1595796]

[6] Blinman S, Cockburn A. Program comprehension: Investigating the effects of naming style and documentation. In: Proc. of the 6th Australasian Conf. on User Interface (AUIC 2005), Vol.40. Darlinghurst: Australian Computer Society, Inc., 2005. 73−78.

[7] Høst EW. Understanding programmer language. In: Proc. of the Companion to the 22nd ACM SIGPLAN Conf. on Object-Oriented Programming Systems and Applications Companion (OOPSLA 2007). New York: ACM Press, 2007. 943−944. [doi: 10.1145/1297846.1297957]

[8] Butler S, Wermelinger M, Yu Y, Sharp H. Exploring the influence of identifier names on code quality: An empirical study. In: Proc. of the 2010 14th European Conf. on Software Maintenance and Reengineering (CSMR 2010). Washington: IEEE Computer Society, 2010. 156−165. [doi: 10.1109/CSMR.2010.27]

[9] Gao Y, Liu H, Fan XZ, Niu ZD, Shao WZ. Research on resolution sequence of bad smells. Ruan Jian Xue Bao/Journal of Software, 2012,23(8):1965−1977 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4152.htm [doi: 10.3724/SP.J.1001.2012.04152]

[10] Kuhn A. On recommending meaningful names in source and UML. In: Proc. of the 2nd Int'l Workshop on Recommendation Systems for Software Engineering (RSSE 2010). New York: ACM Press, 2010. 50−51. [doi: 10.1145/1808920.1808932]

[11] Host E, Ostvold B. The programmer's lexicon. Vol.i: The verbs. In: Proc. of the 7th IEEE Int'l Working Conf. on Source Code Analysis and Manipulation (SCAM 2007). 2007. 193−202. [doi: 10.1109/SCAM.2007.18]

[12] Broder A. On the resemblance and containment of documents. In: Proc. of the Compression and Complexity of Sequences 1997 (SEQUENCES'97). Washington: IEEE Computer Society, 1997. 21. [doi: 10.1109/SEQUEN.1997.666900]

[13] Toutanova K, Klein D, Manning CD, Singer Y. Feature-Rich part-of-speech tagging with a cyclic dependency network. In: Proc. of the 2003 Conf. of the North American Chapter of the Association for Computational Linguistics on Human Language Technology (NAACL 2003), Vol.1. Stroudsburg, 2003. 173−180. [doi: 10.3115/1073445.1073478]

[14] Toutanova K, Manning CD. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In: Proc. of the 2000 Joint SIGDAT Conf. on Empirical Methods in Natural Language Processing and Very Large Corpora: Held in Conjunction with the 38th Annual Meeting of the Association for Computational Linguistics (EMNLP 2000), Vol.13. Stroudsburg, 2000. 63−70. [doi: 10.3115/1117794.1117802]

[15] Forman G. An extensive empirical study of feature selection metrics for text classification. Journal of Machine Learning Research, 2003,3: 1289−1305.

[16] Pei YB, Liu XX. Study on improved CHI for feature selection in Chinese text categorization. Computer Engineering and Applications, 2011,47(4):128−130,194 (in Chinese with English abstract). [doi: 10.3778/j.issn.1002-8331.2011.04.035]

[17] Asmussen J. Survey of pos taggers. Technical Report, DK-CLARIN, 2011.

[18] Eugenie Giesbrecht SE. Is part-of-speech (pos) tagging—A solved task? an evaluation of pos taggers for the Web as corpus. In: Proc. of the 5th Web as Corpus Workshop (WAC5). Donostia-San Sebastin, 2009. 27−35.

[19] Wordnet. http://wordnet.princeton.edu/wordnet/

[20] Abebe S, Tonella P. Natural language parsing of program element names for concept extraction. In: Proc. of the 2010 IEEE 18th Int'l Conf. on Program Comprehension (ICPC). 2010. 156−159. [doi: 10.1109/ICPC.2010.29]

[21] Binkley D, Hearn M, Lawrie D. Improving identifier informativeness using part of speech information. In: Proc. of the 8th Working Conf. on Mining Software Repositories (MSR 2011). New York: ACM Press, 2011. 203−206. [doi: 10.1145/1985441.1985471]

[22] Butler S, Wermelinger M, Yu Y, Sharp H. Mining java class naming conventions. In: Proc. of the 2011 27th IEEE Int'l Conf. on Software Maintenance (ICSM). 2011. 93−102. [doi: 10.1109/ICSM.2011.6080776]

[23] Hu Y. Text feature selection method based on the information gain. Computer & Digital Engineering, 2013,(3):460−462 (in Chinese with English abstract). [doi: 10.3969/j.issn.1672-9722.2013.03.039]

[24] Ren YG, Yang RJ, Yin MF, Ma MW. Information-Gain-Based text feature selection method. Computer Science, 2012,(11):127−130 (in Chinese with English abstract). [doi: 10.3969/j.issn.1002-137X.2012.11.029]

[25] Liu L, Kang J, Yu J, Wang Z. A comparative study on unsupervised feature selection methods for text clustering. In: Proc. of the 2005 IEEE Int'l Conf. on Natural Language Processing and Knowledge Engineering (IEEE NLP-KE 2005). 2005. 597−601. [doi: 10.1109/NLPKE.2005.1598807]

[26] Fan DH, Wang ZH, Chen JH, Xu HY. Improved feature selection algorithm based on DF algorithm for text clustering. Journal of Gansu Lianhe University (Natural Sciences), 2012,(1):51−54 (in Chinese with English abstract). [doi: 10.3969/j.issn.1672-691X.2012.01.014]

[27] Wang G, Lochovsky FH. Feature selection with conditional mutual information maximin in text categorization. In: Proc. of the 13th ACM Int'l Conf. on Information and Knowledge Management (CIKM 2004). New York: ACM Press, 2004. 342−349. [doi: 10.1145/1031171.1031241]

[28] Yang Y, Pedersen JO. A comparative study on feature selection in text categorization. In: Proc. of the 14th Int'l Conf. on Machine Learning (ICML'97). San Francisco: Morgan Kaufmann Publishers, 1997. 412−420.

[29] Liu T, Liu S, Chen Z. An evaluation on feature selection for text clustering. In: Proc. of the ICML. 2003. 488−495.

[30] Høst EW, Østvold BM. The Java programmer's phrase book. In: Proc. of the 1st Int'l Conf. on Software Language Engineering (SLE 2008). Springer-Verlag, 2008. [doi: 10.1007/978-3-642-00434-6_20]

[31] Lawrie D, Morrell C, Feild H, Binkley D. What's in a name? A study of identifiers. In: Proc. of the 14th IEEE Int'l Conf. on Program Comprehension. Washington: IEEE Computer Society, 2006. 3−12. [doi: 10.1109/ICPC.2006.51]

[32] Lin QS, Xie GD. On programming style and robustness in C/C++. Journal of Putian University, 2002,(3):40−44 (in Chinese with English abstract). [doi: 10.3969/j.issn.1672-4143.2002.03.011]

[33] Cao N. Research on Code Tidiness and Quality. Software Guide, 201,(10):38−40 (in Chinese with English abstract).

[34] Jing YS. An evalution approach of identifier quality based on lexical rules [MS. Thesis]. Harbin: Harbin Institute of Technology, 2011 (in Chinese).

[35] Høst EW, Østvold BM. Debugging method names. In: Proc. of the 23rd European Conf. on Object-Oriented Programming. ser. (ECOOP 2009). Berlin, Heidelberg: Springer-Verlag, 2009. 294−317. [doi: 10.1007/978-3-642-03013-0_14]

[36] Rajlich V, Wilde N. The role of concepts in program comprehension. In: Proc. of the IWPC 2002. 2002. 271−278.

[37] Hill E. Integrating natural language and program structure information to improve software search and exploration [Ph.D. Thesis]. Newark, 2010.

[38] Thies A, Roth C. Recommending rename refactorings. In: Proc. of the 2nd Int'l Workshop on Recommendation Systems for Software Engineering (RSSE 2010). New York: ACM Press, 2010. 1−5. [doi: 10.1145/1808920.1808921]

附中文参考文献:

[9] 高原,刘辉,樊孝忠,牛振东,邵维忠.代码坏味的处理顺序.软件学报,2012,23(8):1965−1977. http://www.jos.org.cn/1000-9825/4152.htm [doi: 10.3724/SP.J.1001.2012.04152]

[16] 裴英博,刘晓霞.文本分类中改进型 CHI 特征选择方法的研究.计算机工程与应用,2011,47(4):128−130,194. [doi: 10.3778/j.issn.1002-8331.2011.04.035]

[23] 胡颖.基于信息增益的文本特征选择方法.计算机与数字工程,2013,(3):460−462. [doi: 10.3969/j.issn.1672-9722.2013.03.039]

[24] 任永功,杨荣杰,尹明飞,马明威.基于信息增益的文本特征选择方法.计算机科学,2012,(11):127−130. [doi: 10.3969/j.issn.1002-137X.2012.11.029]

[26] 樊东辉,王治和,陈建华,许虎寅.基于 DF 算法改进的文本聚类特征选择算法.甘肃联合大学学报(自然科学版),2012,(1):51−54. [doi: 10.3969/j.issn.1672-691X.2012.01.014]

[32] 林秋申,谢国栋.C/C++的编程风格与强壮性的探讨.莆田学院学报,2002,(3):40−44. [doi: 10.3969/j.issn.1672-4143.2002.03.011]

[33] 曹娜.代码整洁与代码质量研究.软件导刊,2013,(10):38−40.

[34] 运思婧.基于词性规则的软件标识符质量评价方法[硕士学位论文].哈尔滨:哈尔滨工业大学,2011.

高原(1978－),男,陕西汉中人,博士,工程师,主要研究领域为软件重构,软件测试,软件测评.

樊孝忠(1948－),男,教授,博士生导师,主要研究领域为自然语言处理.

刘辉(1978－),男,博士,副教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,软件演化与维护.

牛振东(1968－),男,博士,教授,博士生导师,主要研究领域为海量数字资源管理,智能信息检索,脑功能连接模型,智能教育技术,软件体系结构.