(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.

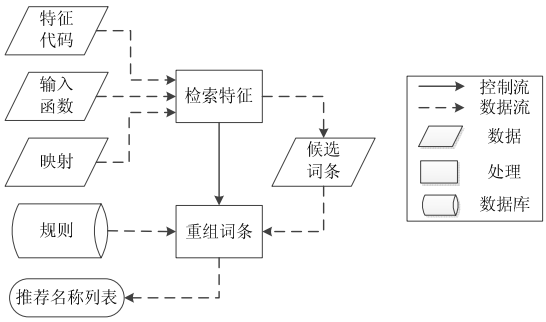

Fig.4 Overview of recommendation process

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.

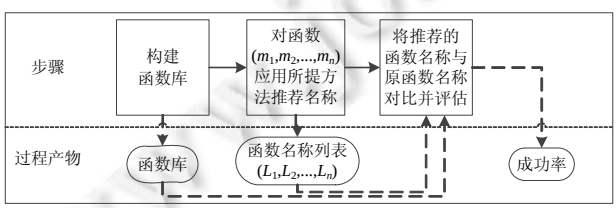

Fig.5 Evaluation process

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).

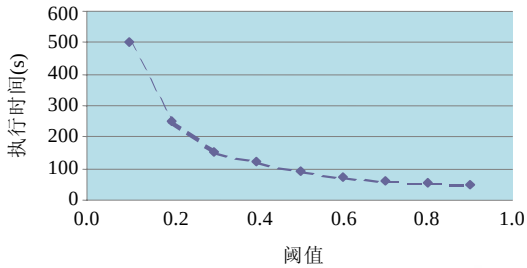

Fig.6 Relationship between threshold and execution time

图 6 阈值与执行时间的关系图

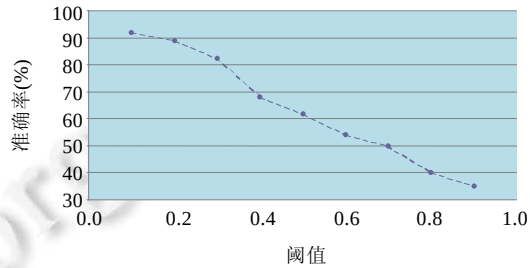

Fig.7 Relationship between threshold and accuracy

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.

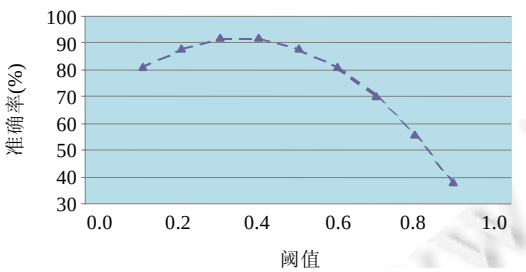

Fig.8 Relationship between threshold and accuracy time

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.