一种软件特征定位研究结果的评估方法

尹春林[1],王 炜[1,2],李 彤[1,2],蒋 巍[1]
(1. 云南大学 软件学院,云南 昆明 650091;
2. 云南省软件工程重点实验室,云南 昆明 650091)

摘 要:软件特征定位是软件演化得以成功实现的重要前提条件,且软件特征定位实验结果的评估标准直接关系到软件演化活动的适用范围。当前软件特征定位结果的评估方法通常采用实验检索结果的 10% ~15% 检索量来进行下一步的定位实验,会不可避免地造成查找范围过大且查准率过低的情况。同时,由于特征定位相关领域新技术的出现,急需一种新的特征定位研究结果的评估方法来提升特征定位的效率。通过多次实验的总结,结合波及效应分析及当前使用的一些评价实验结果的方法,认为每次定位出一个源代码文件,恰好该源文件也是特征相关的源代码文件为最理想的特征定位。为此,提出一种新的实验结果评估方法,将检索量从实验数据总数的 10% ~15% 降到 1 个实验数据。数据分析结果表明,由所提出的评估方法所获得的特征定位结果可以有效地缩小软件演化的范围并提高软件演化的效率。

关键词:软件特征定位;软件演化;波及效应分析;检索量;查准率
中图分类号:TP301 **文献标识码**:A **文章编号**:1673-629X(2017)07-0047-04
doi:10.3969/j.issn.1673-629X.2017.07.011

An Evaluation Method of Studying Results of Software Feature Location

YIN Chun-lin[1], WANG Wei[1,2], LI Tong[1,2], JIANG Wei[1]
(1. College of Software, Yunnan University, Kunming 650091, China;
2. Key Laboratory for Software Engineering of Yunnan Province, Kunming 650091, China)

Abstract: Software feature localization is an important prerequisite for the successful development of software evolution, and the evaluation criteria of software feature localization experiment results are directly related to the scope of software evolution. The evaluation method of current software features location results of experiments using the 10% ~15% retrieval of retrieval results are used for the next step experiment, which could inevitably cause the too large search range and low precision. Meanwhile due to the emergence of new technologies in the field of feature localization, a new evaluation method of research results is urgently necessary to improve the efficiency of feature location. Through the summary of many experiments, combined with the ripple effect analysis and the current use of some experimental results of the evaluation methods i. e. each time a source code file is located exactly that the source file is also a source code files related feature for the most ideal feature location. For this purpose, a new experiment result evaluation method has been proposed, which has reduced the retrieval amount from the total number of experimental data of 10% ~15% to 1. The experimental results show that the feature localization results obtained by the proposed evaluation method have effectively narrowed the scope of software evolution and improve the efficiency of software evolution.

Key words: software feature location; software evolution; ripple effect analysis; retrieval amount; precision rate

0 引 言

特征定位(Feature Location,即故障定位)[1]也称为概念定位(Concept Location)[2-3]或者软件侦测(Software Reconnaissance)[4],是程序理解领域一个重要的组成部分[5-6]。该研究旨在建立特征与源代码之间的映射关系,而特征(Features)[1,7]是指可被定义和评估的软件功能属性。特征定位是软件过程、软件维护以及波及效应分析[8-10]等多种软件活动研究的基

收稿日期:2016-08-09 **修回日期**:2016-11-16 **网络出版时间**:2017-06-05
基金项目:国家自然科学基金资助项目(61462092,61262024,61379032);云南省自然科学基金重点项目(2015FA014);云南省自然科学基金(2013FB008);云南省教育厅科学研究基金(2016YJS007)
作者简介:尹春林(1991-),男,硕士生,CCF 会员(200058963G),研究方向为软件过程、软件演化、特征定位、数据挖掘、数据可视化等;王炜,副教授,CCF 会员(E200035464M),研究方向为软件工程、软件演化、数据挖掘。

础。按分析方式的不同特征定位可以分为:静态特征定位、动态特征定位、基于文本的特征定位和混合特征定位。在软件维护/演化领域,没有任何一个维护/演化任务能够脱离特征定位的支持[1,11]。文献[12-13]指出,在长生命周期软件系统中,50% ~75% 的系统成本和花费用于软件维护,其中一半以上的工作量用于特征定位。对特征定位的研究最早可以追溯到 1992 年,Wilde 提出了最早的特征定位方法—软件侦测[4]。可见,软件特征定位起步晚,发展缓慢,而当前国内外的软件特征定位都还处于一个起步阶段。

软件特征定位的好坏直接影响到软件的维护范围和软件的演化过程,好的软件特征定位可以缩短软件演化的过程,增加软件的使用效率,维护的时间缩短了,软件使用的时间就增加了。然而特征定位研究结果的评估标准直接决定软件维护和软件过程的展开范围,好的软件特征定位研究结果的评价方法可以增加软件维护和软件演化等的效率。当前国内外对特征定位研究主要采用信息检索中的查准率、查全率和调和平均数这三个指标来衡量实验输出结果的质量[14],而查准率和查全率却存在逆反关系,如何找到它们之间的平衡点,目前只能靠经验解决这个问题。软件系统中一部分软件的演化通常会涉及到其他部分的变化,为此,通常需要确定演化活动所影响的范围,这就是波及效应分析。一个特征通常关联多个源代码,特征定位的目的就是找到所有与特征相关联的源代码,建立起它们之间的映射关系。但随着波及效应分析的出现,仅需要定位出一个或者部分特征相关源代码就可以将剩下的工作交由波及效应分析来进行,并不需要在特征定位阶段就找到所有与特征相关的源代码。同时,当前软件特征定位采用的实验研究结果评估方法评估范围过大,直接导致了之后的演化效率过低。所以当前的特征定位实验结果评价标准就出现了一些不足。针对这个问题,提出了一个新的评估方法和标准。

1 当前衡量标准

为了验证特征定位实验结果的有效性、客观性和综合性能,目前通常利用信息检索中的三个指标:查准率、查全率和调和平均数。三个指标分别如下:

定义 1:Recall 表示查全率,correct 代表与特征相关的代码总量,retrieved 表示使用特征定位方法检索出来的代码量,则查全率的定义可表示为:

$$\text{Recall} = \frac{|\text{correct} \cap \text{retrieved}|}{|\text{correct}|} \times 100\% \quad (1)$$

定义 2:Precision 表示查准率,定义为:

$$\text{Precision} = \frac{|\text{correct} \cap \text{retrieved}|}{|\text{retrieved}|} \times 100\% \quad (2)$$

从定义 1、定义 2 中不难发现,查准率和查全率之间存在逆反关系,即如果想要获得高的查全率就会牺牲查准率,反过来要获得高的查准率可能就意味着查全率的降低,一种极端的情况是:将一个系统中所有的代码集合全部输出为结果集,这样能获得 100% 的查全率,但查准率肯定会降到最低。为此,就使用查全率和查准率的调和平均数 F(F-measure)来度量特征定位技术的综合性能[14]。

定义 3:使用 F 表示查全率和查准率调和平均数,则有[5]:

$$F = 2 \times \frac{\text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}} \quad (3)$$

2 当前衡量标准的缺陷

如果使用信息检索中提供的衡量标准,检索量的多少影响着查准率的高低、查全率的高低及软件演化活动的波及范围。之前文本特征定位方法使用经验阈值 $\alpha = 0.075$ 作为判定文本与特征是否具有相关性的标准[15],但在实验中发现,在数据规模较大时,取 $\alpha = 0.075$ 效果并不理想。因此,目前通常使用文献[7]推荐的方法,取相似度最高的 10% ~15% 的源代码作为结果。但是这样获得的特征定位结果并没有缩小特征定位结果的定位范围,尤其是大的软件系统,这种定位结果对软件维护人员来说没什么意义。拿到这样的输出结果,程序员通常需要进行二次筛选或者更多次的筛选才能准确找到特征相关源代码。ArgoUML 和 jEdit[1] 是目前常用的两个基准实验数据集,如下所述:

ArgoUML:包含 Queries、GoldSets 和 Traces 三个文件夹。ShortDescription 是每一个更新的概括性短语描述,LongDescription 是每一个更新的详细描述,Queries 为软件版本更新的说明。GoldSets 是与特征实际相关的源代码,即在演化过程中是哪些类被修改了[16]。Traces 为特征相关的执行迹,记录了特征对应的测试用例在执行过程中搜集到的源代码调用信息。

jEdit:包含 6 400 多个 method,531 个 class,1.9 万行代码。同时,还有 272 个通过使用 SVN(Subversion)和 IST(Issue Tracking System)工具分析获得的特征。其中有 150 个特征用例搜集过执行迹,这些特征以 ID 作为标识,在 jEdit 更新日志中可查。每个特征对应以下三个数据文件:

(1)Queries:包含以“LongDescription[ID].txt”和“ShortDescription[ID].txt”命名的两个文件。内容为特征的文本描述,该数据为 jEdit 版本更新时的改动说明文本。ShortDescription 里的内容是概括性短语描述,LongDescription 里的内容是详细描述。

(2)Traces:以“trace[ID].txt”命名的文本文件。

内容为与特征相关的执行迹。

(3) GoldSet:内容为以“GoldSet[ID]. txt”命名的文件。记录了与特征实际相关的源代码。此数据可用于验证定位结果的正确性[17]。

在 ArgoUML 中使用 0. 20-0. 22 版本的源代码进行特征提取和处理,可得到 1 562 个类和 96 个包,而 jEdit 包含 531 个 class,6 400 多个 method,1. 9 万行代码。

表 1 是在 class 粒度做的比较,如果按文献[14]推荐的方法,取特征定位结果集中相似度最高的前 10% ~15% 作为输出结果,基准数据集 ArgoUML0 输出的数据个数为 156 ~234 个,而小一点的 jEdit 输出的个数也有 53 ~79 个。然而在两个基准数据中给出的真实的与特征相关的源代码是远远小于这个数字的,表 2 是 jEdit4. 3 中的 10 个样本数据。

表 1　数据集对比

系统	总类个数	检索率/%	检索后个数
jEdit4. 3	531	10 ~15	53 ~79
ArgoUML0. 20-0. 22	1 562	10 ~15	156 ~234

表 2　10 个数据集

序号	ID	GoldSet
1	950961	5
2	1538702	9
3	1578785	7
4	1593464	4
5	1638642	8
6	1658252	3
7	1676041	4
8	2696564	4
9	2777073	4
10	1574587	4

表 2 中给出了 10 个样本,一个“ID”对应一个特征,“GoldSet”为与对应特征实际相关的 class 个数。以 jEdit 为例,用表 2 中“GoldSet”个数最多的“1538702”特征,再结合表 1、表 2 可知,就算把相似度最高的前 10% 检索出的结果也有 53 个 class,但实际与特征相关的 class 只有 9 个,这样获得的查准率也仅仅为 17% 。然而可以认为这 17% 未必就是真正的查准率,特殊的情况见表 3(表 3 中用一个数据的路径代表一个数据)。

表 3 是用 jEdit 做特征定位的源代码数据集,按 10% 的检索率检索出的与特征“1538702”相似度最高前 53 个 class。假设查全率为 100% ,即查到了与特征“1538702”实际相关的所有 class(9 个),极端情况下出现这 9 个 class 是最后 9 个,也就是第 44 ~53 个类,而前 43 个 class 都是与特征无关的 class,这样所获得的查准率同样是 17% ,但这种情况对软件演化、软件维护很不利。比如说这个是一个 bug,想要找到该 bug 的源代码,这样定位出的结果去找 bug 相关源代码就得先检查完前 43 个 class,才能找到真正与 bug 实际相关的源代码。有经验的程序员可能还不需要找 43 个 class 就找到了 bug 相关的源代码。所以可以认为这 17% 的查准率是虚的,没有什么意义。

表 3　特殊实验数据

检索出的实验结果数据	标签名	相似度排序
F:\jedit4. 3source\jEdit\org\gjt\sp\jedit\bsh\BlockNameSpace. java	2	1
⋮	⋮	⋮
F:\jedit4. 3source\jEdit\org\gjt\sp\jedit\bsh\BSHBlock. java	10	52
F:\jedit4. 3source\jEdit\org\gjt\sp\jedit\bsh\BshMethod. java	7	53

3　新评估方法及实验结果分析

特征定位的目的是快速准确地找到特征与特征相关源代码之间的正确映射关系,通常一个特征关联一个以上的源代码,传统的特征定位方法认为好的特征定位方法能够定位到所有的特征相关源代码。目前很多学者认为只用评估实验检索出来的前 10% ~15% 结果中第一个与实际特征相关联的代码排在第几位就可以了。如表 3 是一种特征定位方法检索出来的与某个特征实际相关的 53 个代码,第一个与该特征实际相关的代码为第 5 个,而另一种方法检索出来的第一个与实际特征相关的代码为第 4 个,则认为后一种方法要比前一种方法好,即认为排名越靠前结果越好。在前面两种评价方法的基础上,结合波及效应分析,提出了新的评估方法。结合这两种评估思想与波及效应分析,可以认为特征定位最理想的情况是只定位出一个特征相关源代码,剩下的工作交给波及效应分析。波及效应分析通过利用可以定位出的这一个源代码就可以找到其他几个与特征真实相关的源代码。在这种情况下查准率的计算如下:

定义 4:Precision 表示查准率; Retrieved_i 表示实验检索出的与特征 i 相似度最高的那一个数据代码,如果 Retrieved_i 真的与特征 i 相关,则 $\text{Retrieved}_i = 1$,否则 $\text{Retrieved}_i = 0$; Correct_n 表示实验定位的特征个数,则有:

$$\text{Precision} = \frac{\sum_{i=1}^{n} \text{Retrieved}_i}{\text{Correct}_n} \times 100\% \qquad (4)$$

实验结果表明,使用提出的方法仅仅只是检索出一个特征相关的源代码,计算查全率和调和平均数就没有意义,所以就不进行定义。这样的评价方法本质上是将检索量从实验数据总数的 10% ~15% 缩减到 1 个,但却大大压缩了特征定位的成本,提高了软件演化的效率。

4 结束语

特征定位是软件过程及软件演化等活动得以顺利展开的保证。特征定位研究结果的质量影响着软件演化活动的效率,而特征定位研究结果的评估标准决定了软件演化活动的波及范围。结合波及效应分析及当前常用的评价标准,提出了一种新的实验评价方法,所提出的评估标准可以大大降低软件演化活动的工程量及工程范围,所采用的评价标准不但具有普遍意义,更具有实际意义。

参考文献:

[1] Dit B,Revelle M,Gethers M,et al. Feature location in source code:a taxonomy and survey[J]. Journal of Software Maintenance & Evolution Research & Practice,2013,25(1):53-95.

[2] Abebe S L,Alicante A,Corazza A,et al. Supporting concept location through identifier parsing and ontology extraction [J]. Journal of Systems and Software,2013,86(11):2919-2938.

[3] Scanniello G,Marcus A,Pascale D. Link analysis algorithms for static concept location:an empirical assessment[J]. Empirical Software Engineering,2015,20(6):1666-1720.

[4] Wilde N,Gomez J,Gust T,et al. Locating user functionality in old code[C]//Proceedings of international conference on software engineering. [s. l.]:IEEE,1992:200-205.

[5] Alhindawi N,Alsakran J,Rodan A,et al. A survey of concepts location enhancement for program comprehension and maintenance[J]. Journal of Software Engineering and Applications,2014,7(5):413-421.

[6] 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.

[7] Poshyvanyk D,Gueheneuc 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] Bohner S A. Impact analysis in the software change process:a year 2000 perspective[C]//International conference on software maintenance. [s. l.]:[s. n.],1996:42-51.

[9] Li B,Sun X,Leung H,et al. A survey of code-based change impact analysis techniques[J]. Software Testing Verification & Reliability,2013,23(8):613-646.

[10] 王 炜,李 彤,何 云,等. 一种软件演化活动波及效应混合分析方法[J]. 计算机研究与发展,2016,53(3):503-516.

[11] Rajlich V,Gosavi P. Incremental change in object-oriented programming[J]. IEEE Software,2004,21(4):62-69.

[12] Li T. An approach to modelling software evolution processes [M]. Berlin:Springer,2009.

[13] Seacord R C,Plakosh D,Lewis G A. Modernizing legacy systems:software technologies,engineering processes,and business practices[M]. [s. l.]:Addison-Wesley Professional,2003.

[14] 鞠小林,姜淑娟,张艳梅,等. 软件故障定位技术进展[J]. 计算机科学与探索,2012,6(6):481-494.

[15] Marcus A,Sergeyev A,Rajlich V,et al. An information retrieval approach to concept location in source code[C]//11th working conference on reverse engineering. [s. l.]:IEEE,2004:214-223.

[16] 韩俊明,王 炜,李 彤,等. 演化软件的特征定位方法[J]. 计算机科学与探索,2016,10(9):1201-1210.

[17] 何 云,王 炜,李 彤,等. 面向行为主题的软件特征定位方法[J]. 计算机科学与探索,2014,8(12):1452-1462.

(上接第 46 页)

ken matching based string similarity join[C]//27th international conference on data engineering. [s. l.]:IEEE,2011:458-469.

[11] 谢子光. 多核处理器核间通信技术研究[D]. 成都:电子科技大学,2009.

[12] 徐媛媛. 基于 MapReduce 的相似性连接研究[D]. 宁波:宁波大学,2014.

[13] 庞 俊,谷 峪,许 嘉,等. 相似性连接查询技术研究进展[J]. 计算机科学与探索,2013,7(1):1-13.

[14] Chaudhuri S,Ganti V,Kaushik R. A primitive operator for similarity joins in data cleaning[C]//Proceedings of the 22nd international conference on data engineering. [s. l.]:IEEE,2006:5.

[15] 蔡进国,郭 宏,李伟强,等. 多核多线程环境下的程序并行优化方法[J]. 现代计算机,2004(3):3-5.

[16] Jiang Y,Deng D,Wang J,et al. Efficient parallel partition-based algorithms for similarity search and join with edit distance constraints[C]//Proceedings of the joint EDBT/ICDT Workshops. [s. l.]:[s. n.],2013:341-348.

[17] 于 方. 多核平台下的多线程并行编程[J]. 阴山学刊:自然科学版,2010,24(3):33-36.

[18] 眭俊华,刘慧娜,王建鑫,等. 多核多线程技术综述[J]. 计算机应用,2013,33(S1):239-242.