软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.006087]

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

基于信息检索的缺陷定位：问题、进展与挑战*

郭肇强, 周慧聪, 刘释然, 李言辉, 陈 林, 周毓明, 徐宝文

(计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)
(南京大学 计算机科学与技术系,江苏 南京 210023)
通讯作者: 李言辉, 周毓明, E-mail: {yanhuili, zhouyuming}@nju.edu.cn

摘 要: 缺陷的存在会影响软件系统的正常使用甚至带来重大危害.为了帮助开发者尽快找到并修复这些缺陷,研究者提出了基于信息检索的缺陷定位方法.这类方法将缺陷定位视为一个检索任务,它为每个缺陷报告生成一份按照程序实体与缺陷相关度降序排序的列表.开发者可以根据列表顺序来审查代码从而降低审查成本并加速缺陷定位的进程.近年来,该领域的研究工作十分活跃,在改良定位方法和完善评价体系方面取得了较大进展.与此同时,为了能够在实践中更好地应用这类方法,该领域的研究工作仍面临着一些亟待解决的挑战.本文对近年来国内外学者在该领域的研究成果进行系统性的总结.首先,描述了基于信息检索的缺陷定位方法的研究问题.然后,分别从模型改良和模型评估两方面陈述了相关的研究进展,并对具体的理论和技术途径进行梳理.接着,简要介绍了缺陷定位的其他相关技术.最后,总结了目前该领域研究过程中面临的挑战并给出建议的研究方向.

关键词: 信息检索;缺陷定位;软件维护;缺陷报告
中图法分类号: TP311

中文引用格式: 郭肇强,周慧聪,刘释然,李言辉,陈林,周毓明,徐宝文.基于信息检索的缺陷定位: 问题、进展与挑战.软件学报. http://www.jos.org.cn/1000-9825/6087.htm

英文引用格式: Guo ZQ, Zhou HC, Liu SR, Li YH, Chen L, Zhou YM, Xu BW. Information retrieval based bug localization: research problem, Progress, and Challenges. Ruan Jian Xue Bao/Journal of Software, 2020 (in Chinese). http://www.jos.org.cn/1000-9825/6087.htm

Information Retrieval Based Bug Localization: Research Problem, Progress, and Challenges

GUO Zhao-Qiang, ZHOU Hui-Cong, LIU Shi-Ran, LI Yan-Hui, CHEN Lin, ZHOU Yu-Ming, XU Bao-Wen

(State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)
(Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China)

Abstract: Bugs can affect the normal usage of a software system or even bring huge damages. In order to facilitate find and fix bugs as soon as possible, researchers have proposed information retrieval based bug localization techniques. This kind of techniques regards bug localization as a task of text retrieval. Specifically, for a given bug report, a rank list of code entities in a descending order is provided according to relevance score between code entity and the bug. Developers can select entities in the rank from top to bottom, which helps reduce the review cost and accelerate the process of bug localization. In recent years, a great progress has been made in information retrieval based bug localization techniques. Nevertheless, it is still challenging to apply them in practice. This survey offers a systematic survey of recent research achievements in information retrieval based bug localization techniques. First, we introduce the research problem in information retrieval based bug localization. Then, we describe the current main research work in detail. After that, we discuss related techniques. Finally, we summarize the opportunities and challenges in this field and outline the research directions in the future.

Key words: information retrieval; bug localization; software maintenance; bug reports

软件缺陷(software bug)是伴随软件产品整个生命周期的产物.特别对于大型软件,由于其复杂的结构和众

* 基金项目: 国家重点研发计划项目(2018YFB1003901),国家自然科学基金(61772259, 61872177)
收稿时间: 2020-01-07; 修改时间: 2020-03-10; 采用时间: 2020-05-18; jos 在线出版时间: 2020-06-08

多的开发者,缺陷不可避免地在开发过程中被引入.软件缺陷的存在极大地降低了软件质量并且增加了软件的维护成本,严重的缺陷可能会给企业造成经济损失甚至对人的生命安全造成威胁.为了提升软件的质量,软件产品要在不断更新换代或者版本更迭中对已经发现的缺陷进行清除.从历史数据来看,对缺陷的清除工作消耗了接近一半的软件开发和维护成本[1].因此在学术界有大量的研究者致力于缺陷相关(包括但不限于缺陷预测[2][3]、缺陷定位[4][5]和缺陷修复[6]等)的研究.其中,非常重要并且耗费时间和精力的一个任务是对缺陷进行定位,即找出产生该缺陷的软件实体(例如源文件).

当软件中的缺陷被发现时,根据缺陷产生时的软件状况,用户或者软件系统会记录缺陷相关的信息并且生成一份半结构化的缺陷报告上传到缺陷追踪系统上[7].软件维护人员对缺陷的修复工作始于阅读缺陷报告,在理解缺陷的基础上,他们从代码库中查找与缺陷报告描述内容相关的代码实体进行审查直至找出产生该缺陷的位置.值得注意的是,一个在实际开发项目通常包含成百上千甚至更多的源文件,手工从其中找出与缺陷相关的那极小部分代码是十分困难的.为此,开发人员急需借助于缺陷自动定位工具来辅助他们查找产生缺陷位置.

为了帮助开发人员快速找到产生缺陷的位置,研究者们提出了多种自动化的缺陷定位(bug localization,简称 BL)解决方法.现有的解决方法可以根据是否需要执行测试用例划分为动态缺陷定位方法和静态缺陷定位方法.其中,动态缺陷定位方法要运行被测程序来收集程序执行的动态信息(包括执行路径和结果等),根据这些信息来定位缺陷语句在代码中的可疑位置.这类方法的主流思路被称为基于程序频谱的缺陷定位[4].由于直接运行目标程序,这类方法通常可以在较细的粒度上对缺陷进行定位,但是需要消耗执行程序的时间和资源成本.因此这类方法有助于对代码进行调试.而静态缺陷定位方法不需要直接运行被测程序,这类方法主要分析代码和缺陷报告中的静态信息(包括文本,代码实体等),然后计算每个代码实体(源文件)和报告中的静态信息的相似度分值,最后根据得分将可疑的代码实体推荐给开发者.静态方法的优势在于使用简单,成本低廉(不需要运行代码).同时它的不足是定位粒度较粗糙,通常在文件或者方法级别对缺陷进行定位.可以看出,这类方法适用于帮助开发者快速找到缺陷报告所描述的问题的相关位置.

近年来,静态缺陷定位研究工作的一类主流方法是基于信息检索的缺陷定位(information retrieval based bug localization,简称 IRBL).该领域目前的研究重点在于改良 IRBL 方法和完善 IRBL 的评价体系两方面.截止 2020 年 3 月,已有 20 多种(仍在增长)不同 IRBL 方法和多种不同的评价指标被提出.从现有研究结果来看,IRBL 方法在实验评估中已经取得了较高的性能.然而,由于现实中不同项目代码的结构和风格存在差异性,并且收到的缺陷报告内容质量参差不齐,目前的 IRBL 方法仍然存在局限性,在现实的实践应用中仍然面临一些挑战:

(1) 方法所用的特征由人工设计,对目标项目依赖高,在其他项目上迁移性能较差;
(2) 目前这些方法仅在实验评估中取得较好性能分数,但是由于各自实验设置的差异性,开发者无法准确判断哪一种方法最具有实用性.因此这些方法在生产实践中的表现结果未知;
(3) 现有方法仅仅给开发者推荐出与缺陷有关的代码文件列表,没有包含任何有助于定位的提示信息.

值得注意的是,目前学术界已有多份综述性的工作[4,8-12]对缺陷定位领域的技术研究进行总结.这些工作对多种特定类型的缺陷定位技术进行了整理和归纳,包括基于程序频谱的缺陷定位技术[4]、基于程序变异的缺陷定位技术[12]等.但是据我们所知,还没有一项工作对 IRBL 这种特定类型的缺陷定位方向进行总结(截止 2020 年 3 月).由于对 IRBL 研究的论文体量已经较为庞大并且是一类热门的研究方向,为了弥补这一空白同时方便后续相关研究工作的进行,本文首次对近 10 年来 IRBL 的研究历史进行回顾,对现有的成果进行分类介绍和总结,并在此基础上指出该领域面临的挑战.具体结构如下.

本文在第 1 节首先概述主要的研究问题;其次在第 2 节说明本综述的文献检索方式和文献汇总信息;然后分别在第 3 节和第 4 节从模型改良和模型评估两个方面介绍近年的研究进展;接着在第 5 节分类别地介绍缺陷定位领域的其他研究方法;在第 6 节分析 IRBL 领域的研究所面临的挑战;最后在第 7 节总结本文的内容.

1 研究问题

1.1 IRBL概要介绍

信息检索技术早期是被用来在代码中进行概念/特征定位(不同文献中的术语略有差别,concept location[13][14][15], feature location[16][17][18]或者 concern localization[19][20][21][22])任务,其目的是从代码库中找出与特定功能相关的代码.后来演变为使用信息检索技术对缺陷进行定位(IRBL),并成为目前主流的静态缺陷定位研究方法.具体来说,程序中的特征是指能够被开发者感知或者看见的功能,其中那些想要被抛弃并且被移除的“功能”便可以看作是一种特殊的特征(即,缺陷).特征定位是指从项目中找出具有指定特征的那部分源代码.因此,对于要在程序中抛弃并移除的“功能”的定位便逐步变成现有的基于信息检索的缺陷定位.

IRBL 方法关注的对象是缺陷报告和软件源码.从技术角度看,IRBL 方法将缺陷定位任务看作是进行信息检索(IR)的过程.Rao 等人[23]在 2011 年总结了信息检索领域的 IR 术语与软件工程领域在缺陷定位时所用术语的对照关系如表 1 所示.据此,IRBL 任务执行过程可以描述如下：IRBL 模型将缺陷报告看作查询(query),将报告中的文本进行分词处理组成查询语句.同时,将项目源码看作文档库(documents),对每份源文件进行预处理,构造出一个语料库.当收到缺陷报告时,从报告中构建查询语句并在文档库中检索(retrieval),根据查询语句与每个文档的相似度通过索引(index)将所有文档降序排列后反馈给开发者.在结果列表中,包含缺陷的文件(buggy files)尽可能排列在靠前的位置.开发者按照列表顺序对代码进行审查可以在花费较少的工作量的情况下找到有缺陷的源文件从而加速缺陷定位的进程.

Table 1 Parallels between IR terminology and the more traditional SE terminology

表 1 IR 术语与传统软件工程术语的对照关系

IR 术语	软件工程术语
文档 (Document)	代码库中的源文件
查询 (Query)	缺陷报告和/或它的文本描述
词汇 (Terms)	标识符名称
检索 (Retrieval)	缺陷定位
索引 (Index)	源代码库

IRBL 方法能够获得众多研究者和软件开发者的关注主要得益于这类方法在应用时具有以下两点优势.

(1) **外部依赖少**[24][25][26][27]**.** 相比于动态定位,技术静态技术不需要执行测试用例来触发缺陷并且不需要工作在可运行的目标软件系统中,只需要从源代码和缺陷报告中收集信息即可进行定位.因此,它们具备使用简单的特点从而可以应用于软件开发或维护过程的任何阶段.

(2) **计算成本低**[24][27] **.** IRBL 方法对源码文档的排序的主要依据是缺陷报告与文档之间的文本相似度,这部分分值的计算对现在的机器来说是十分简单,通常可以快速给出排序结果.因而这类方法具备反馈迅速的特点.所以在应用时的用户体验良好.而动态方法通常需要执行目标程序,实际响应时间和计算成本取决于目标程序的复杂程度.

1.2 IRBL方法流程

本小节对 IRBL 方法的流程进行细致描述.综合该领域的多个优秀的研究工作[24][25][28][29][30][31][32],IRBL 方法的流程主要包含建立语料库、构建索引、抽取特征、构建查询以及检索排序这五个基本步骤,如图 1 所示.

(1) **建立语料库.** 对代码库中的每个源代码文件的文本进行分词.同时对分好的词汇进行预处理,主要包括去除关键词(例如, “int”、“public”等)、去除停用词(例如,“a”、“the”等)、词根还原(例如,“delegating”还原为“delegate”)、拆分复合词(例如,“TypeDeclaration”分为“type”和“declaration”)等.处理完成之后,每个源文件对应一组词汇,所有源文件构成了整个项目的语料库;

(2) **构建索引.** 为语料库中的所有源代码文件构建索引,模型可以在后续步骤中根据索引信息找到指定文件并对这些文件根据相似度得分进行排序;

(3) **抽取特征.** 从目标项目中抽取有利于缺陷定位的特征信息(例如,版本历史信息、堆栈信息等).这些特

征信息会在第(5)步中集成到检索模型用于优化排序结果;

(4) **构建查询.** IRBL 方法将缺陷报告当作一条查询语句.使用步骤(1)中的方法对缺陷报告中的标题和描述进行数据预处理可以获得该缺陷报告对应的一组词汇;

(5) **检索和排序.** 使用一种 IR 模型(例如, VSM[28]、LSI[33]、SUM[23]、LDA[34]等)进行建模并利用(3)中的特征优化模型,对建立好索引的语料库计算查询语句和每个源代码文件之间的相似程度(例如,文本相似度),然后按照相似度得分高低对这些源代码文件进行降序排序.

经过上述五个步骤,IRBL 模型会对每份缺陷报告输出一份排序列表.需要说明的是,除非代码版本更迭导致源文件内容更新,否则步骤(1)和步骤(2)通常只需要执行一次,而步骤(3)、(4)和(5)在每次收到新的缺陷报告后都需要执行.

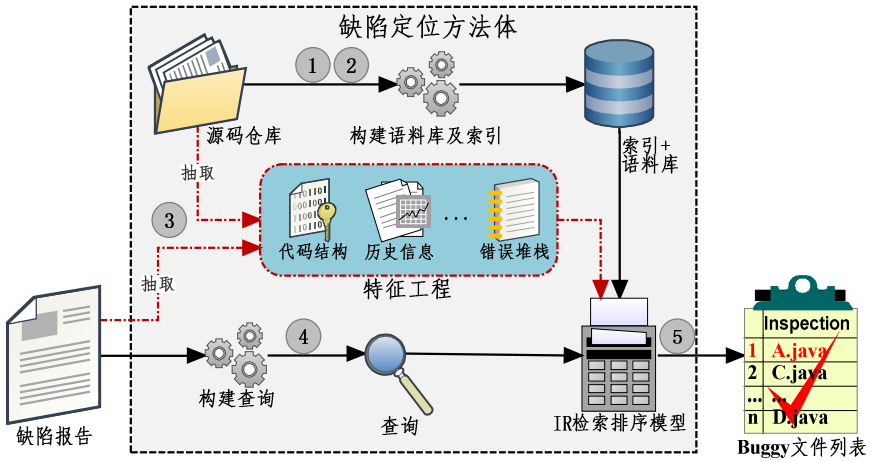

Fig.1 Overview of IR-based bug localization

图 1 基于信息检索的缺陷定位方法流程概览

1.3 IRBL研究内容

目前 IRBL 领域的研究重点是模型改良和模型评估,以下分别从两方面列出 IRBL 研究面临的主要挑战并和对应的研究内容.具体内容将在第 3 节和第 4 节详细介绍.

1.3.1 模型改良(见第 3 节)

IR 模型最初是为检索非结构化地自然语言文本设计的,因此适合处理纯自然语言文本的应用.对于缺陷定位应用来说,不论是软件库中的代码文件还是软件平台收到的缺陷报告,其中都包含着丰富的结构信息.当 IR 模型被直接应用到具有特定结构化信息的软件代码库检索时会遇到来自以下两方面挑战.

(1) **有用特征使用不完全.** 对源代码来说,有大量的可以使用的附加特征(例如,提交历史[32]、代码结构[24][25][30][31]等)可以辅助定位缺陷.对缺陷报告来说,其中包含了标题(Title)和描述(Description)两部分内容,每部分中包含的词汇的权重是有差别的[35].

(2) **无用信息过滤不充分.** 源代码中的关键字是一种干扰词汇[24][28],例如关键字“public”、“class”和“int”等等.这类词汇对于文本语义没有任何帮助并且几乎会出现在所有的代码文件中.代码中的部分注释组成另一类干扰词汇,例如和代码文件的许可证注释(License Comments)通常和文档语义无关.这类词汇会影响检索模型的效果.

上述挑战使得 IRRL 模型的定位性能低下.为了解决上述挑战,IRBL 领域研究的一个重点在于模型改良.即不断对模型进行修改以提高其定位准确性.可行的探索方向是挑选较好的 IR 模型;从各个角度分析代码和缺陷报告的特征信息并将其集成到 IR 模型中.现有研究人员已经挖掘出包括版本历史、代码结构、堆栈踪迹等多种重要的特征信息(见图 1 中红色点划线标明的特征工程路径);提升缺陷报告的质量;使用深度学习技术自动提

取特征等.

1.3.2 模型评估(见第 4 节)

IRBL 方法的输出结果是一个排序列表,因此在对这类方法进行性能评估时使用的是常用的 3 个排序指标 Top@K、MRR、MAP.这些指标在评估纯排序结果时是客观公平和有效的.然而在实际使用过程中是否有效不仅仅取决于排序结果,还与待排序的对象有关.在评估模型时主要有以下两方面挑战.

(1) **评估指标不全面.** 单纯的排序指标并不能得到全面的评价结果.例如,对于缺陷定位来说,开发人员要花费工作量在排序结果中依次审查代码文件[36][37].

(2) **测试项目不充足.** 要取得可靠的评估结果,必须要在大量的项目上对模型进行评估.仅在个别项目上的评价结果在更多项目上测试其泛化性能.

(3) **实验过程不统一.** 为了对比不同方法的性能,需要在统一的实验配置下进行实验.然而在不同研究者的实验设置中存在的差异使得研究者很难得到真实客观的对比结果.

上述挑战使得 IRBL 模型的良好表现只停留在实验阶段而无法在现实项目中被广泛应用.为了解决上述挑战,该领域的另一个研究重点是更加科学全面地评估 IRBL 模型,即使用更加合理的评估指标[36]、更加丰富的数据集[26][38][39]和统一的实验设置对模型进行评估.

2 文献检索

2.1 文献筛选与检索

IRBL 方法一直是缺陷定位领域的热点研究问题.近年来大量的研究工作集中在该领域并不断地对定位方法和评价体系进行改善.为了对该问题进行系统的分析、总结和比较,本文的参考文献力图覆盖近 10 年来与 IRBL 相关的主要研究工作,同时包含缺陷定位领域的其他相关技术的主要文献.具体来说,本文使用以下的步骤来进行相关文献的检索和筛选:

(1) **检索目标:** 谷歌学术搜索引擎(Google scholar)、DBLP Computer Science Bibliography、ACM Digital Library、IEEE Xplore Digital Library 以及 Springer Link Online Library 等论文数据库.检索内容主要包括软件工程-系统软件-程序设计语言方向的国际一流会议(ESEC/FSE、ICSE、ASE 等)和一流期刊(TSE、EMSE、IST 等)以及其他重要的会议和期刊如 MSR、SIGIR、WCRE 等录用的文章;

(2) **检索关键词:** 包括“bug localization”、“locating bugs”、“information retrieval”、和“fault localization”等以及相近的主题关键词如“text retrieval”;

(3) **检索起止时间:** 2000-2019 年之间的文献.需要说明的是,2010 年以前关于 IRBL 的相关研究相对较少,所以重点关注 2010 年及其之后的文献;

(4) **文献审查:** 首先对检索出来的文献进行人工筛选,去除不符合研究主题的文献;然后检查选中论文中的参考文献列表以防遗漏掉未检索到的文献;最后在 2.2 节中对所选文献进行汇总.

2.2 历史文献的汇总

经过文献的检索与筛选,本文收集了基于信息检索的缺陷定位领域中近 10 年来的学术论文共 82 篇.这些研究论文将会按照种类分别在第 3 节和第 4 节进行详细介绍.对于其他种类的缺陷定位相关技术,我们在第 5 节中对其进行分类和介绍.

表 2 列出了基于信息检索技术的相关文献的详细列表.可以看出,在 CCF 划分的软件工程领域 A 类会议上有 15 篇,期刊上有 6 篇.这表明 IRBL 研究是软件工程领域一个比较活跃的研究方向.图 2 展示了 10 年来 IRBL 领域的研究论文发表的数量分布情况.从图中可以看出,该领域的逐年论文发表数量呈现出一个波动上升的趋势.这说明对 IRBL 这个领域正越来越受到研究者的重视.在 2019 年,相关论文数量有所下降但仍然保持着较高的活跃度.

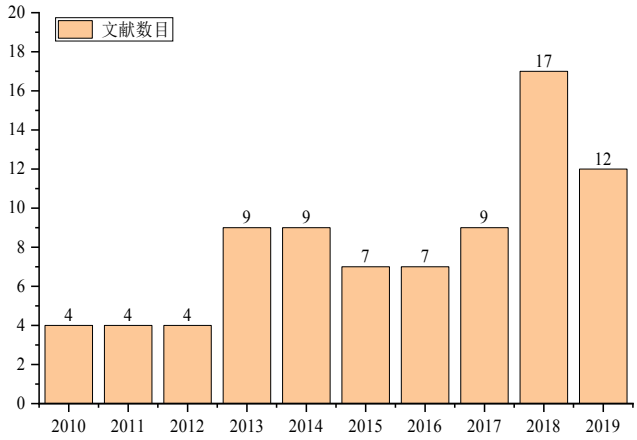

Fig.2 Illustration of literature by year

图 2 10 年来相关文献的年代分布

Table 2 List of related literature on IR-based bug localization researches

表 2 基于信息检索的缺陷定位研究的相关文献列表

类型	级别	出版物缩写	文章数	引用列表
国际会议	CCF A 类	ICSE	5	[28,40,41,42,43]
		FSE	5	[35,44,45,46,47]
		ISSTA	1	[39]
		ASE	4	[24,32,48,49]
	CCF B 类	ICSME	7	[25,26,30,50,51,52,53]
		ICPC	4	[31,54,55,56]
		WCRE	3	[57,58,59]
	CCF C 类	MSR	5	[23,60,61,62,63]
		QRS	1	[36]
		APSEC	2	[64,65]
国际期刊	CCF A 类	TSE	6	[66,67,68,69,70,71]
	CCF B 类	EMSE	2	[72,73]
		IST	6	[34,74,75,76,77,78]
		ASE	1	[79]
		JSEP	2	[80,81]
		SoSyM	1	[82]
		RE	1	[83]
	CCF C 类	SQJ	1	[84]
其他会议期刊	-	-	25	[29,33, 38, 79,85,86,87,88,89,90,91,92,93, 94,95,96,97,98,99, 100,101,102,103,104,105]
合计	-	-	82	-

3 IRBL 模型的改良

对 IRBL 模型的改良研究在不同研究阶段有不同的侧重点.图 3 展示了基于信息检索的缺陷定位方法从 2010 年以来的模型改良趋势和侧重点.同时,我们在表 3 中列出了现有的代表性的 IRBL 模型汇总比较信息.根据对现有研究的汇总,我们得出 IRBL 领域的研究侧重点主要包含以下四个方面.

(1) **更换检索模型.** 早在 2010 年及以前,由于 IR 技术刚刚被应用到缺陷定位领域,研究者的研究重点是尝试使用不同的 IR 模型对缺陷定位任务进行建模来找出该领域比较合适的 IR 模型(例如 VSM、LDA、SUM、LSI 等);

(2) **使用特征分析.** 约从 2011 年开始,研究者们发现单纯使用 IR 模型定位缺陷的性能低下很难满足实际的应用需求.研究者们发现除了文本相似度之外,软件中的其他特征也可以帮助定位缺陷.所以,这一阶

段的研究者们主要集中在挖掘软件项目中与缺陷有关的特有特征(例如,项目版本历史、缺陷堆栈等),并结合使用这些特征和 IR 模型来提升缺陷定位在测试项目上的性能;

(3) **进行查询重构.** 作为查询语句的缺陷报告是由不同的人员提交的,所以质量参差不齐.这些低质量的查询导致很多 IRBL 模型表现较差,因而在 2013 年后出现一些研究者着重于对查询进行重构以改善现有 IRBL 模型的查询准确性.

(4) 应用深度学习. 随着深度学习技术的发展,从 2017 年起,不少研究者开始尝试使用深度学习模型(如,卷积神经网络 CNN)对特征进行自动提取然后结合信息检索技术对缺陷进行定位.

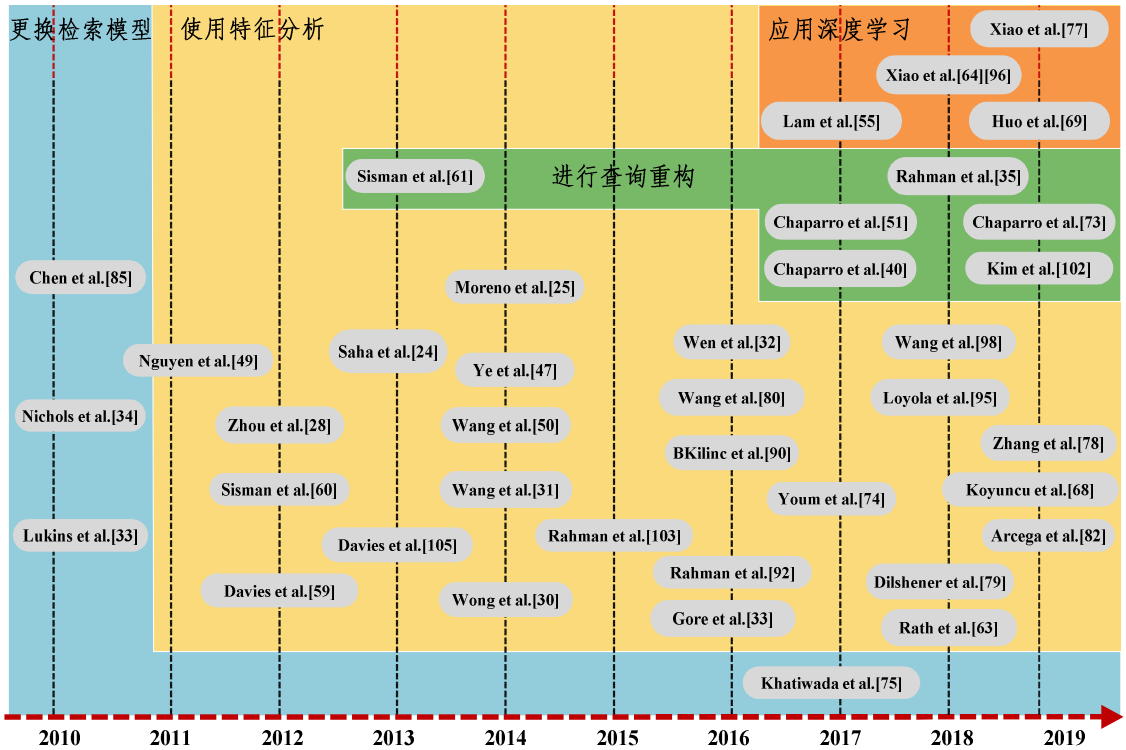

Fig.3 Overview of IR-based bug localization

图 3 基于信息检索的缺陷定位方法的模型改良趋势

3.1 更换检索模型

在信息检索领域有很多模型,其中最常见的包括向量空间模型[106](Vector Space Model,简称 VSM)、潜在语义索引模型[107] (Latent Semantic Index,简称 LSI)、隐狄利克雷分布模型[108](Latent Dirichlet Allocation,简称 LDA)等等.经过多年研究,IRBL 领域的方法中所用的检索模型已经涵盖了上述所有以及其他更多的 IR 模型.

3.1.1 VSM 模型

VSM 是一个十分简单的向量模型,它将自然语言文档库转化为一个字词-文件矩阵.矩阵的每一行代表一个单词在不同文件中的权重值(可以通过 TF.IDF[109]来表示),每一列代表一个文档所对应的特征向量.在字词矩阵的基础上,我们可以很容易地通过向量距离计算来衡量不同文档之间的相似性.因此,在 IRBL 领域现有的方法中,VSM 和它的变体是使用最广泛的 IR 模型.在现有的研究中,大约有超过半数的 IRBL 方法[25][28][30][31][32][35][47] [50][63][68][74][79][80][90][92]都使用 VSM 模型来进行建模.以下是该模型的相关应用.

2009 年,Gay 等人[110]提出一个使用相关反馈(relevance feedback)机制的方法来提升概念定位的结果,在他们的方法中最先明确使用 VSM 模型来进行检索.2012 年,Zhou 等人[28]指出传统 VSM 模型对于更倾向于将长度较短的文档排在前面,而已有研究表明长度较大的文档包含缺陷的可能更高.基于这点认识,他们对 VSM 模

型进行改进提出了 rVSM 模型,其中将文档长度作为一个权重参数.后续有多个研究工作(Wong 等人[30]、Rahman 等人[103]、Youm[74]等)均是在 rVSM 模型之上构建新的 IRBL 方法.2014 年,Wang 等人[50]设计了一个基于搜索的引擎 VSMcomposite,该引擎使用遗传算法(GA)将各种不同的 VSM 变体(即,向量元素的计算方式不同)模型进行组合得到新的近似最优组合模型来进行缺陷定位.

3.1.2 LSI 模型

LSI 是一种实用的主题模型[107][111].它是在传统的 VSM 模型基础上进行改良后得到的.LSI 利用奇异值分解(Singular Value Decomposition,简称 SVD)将词汇和文本映射到一个新的空间来表示文档矩阵之间的潜在语义关系.它的优势是可以解决词语中出现的同义词和多义词对结果的影响,这些词汇是无法被 VSM 模型识别处理的.然而,它的缺点是它需要进行高阶矩阵运算来计算查询字段和每篇文档的相似度,所以它的运行速度慢于 VSM 模型.其次,在使用 SVD 时需要假设数据的分布是正态分布,然而类似词频的统计数据往往不能符合这个条件.经过对文献的整理,我们发现在 IRBL 领域 LSI 是最早被用来进行特征/缺陷定位的 IR 模型,并且只活跃在研究早期(2010 年以前).近年来,几乎已经没有方法使用 LSI 作为 IR 模型来进行缺陷定位.以下是该模型的相关应用.

2004 年,Marcus 等人[13]最先将 LSI 模型应用到概念定位领域中.2006 年,Poshyvanyk 等人[16]将特征定位视为一个决策问题,他们使用 LSI 模型和基于场景的事件概率排序两种技术来强化定位结果.随后,他们在 2007 年又在 LSI 模型基础上提出了 PROMESIR 模型[18]进行特征定位.同年,Liu 等人[17]将 LSI 模型和代码执行踪迹以及代码中的注释和标识符信息相结合来提升定位性能.考虑到现有技术没有使用到软件仓库的动态特性,Rao 等人[87]在 2015 年提出了一个可以随着数据演化增量更新 LSA 模型参数的定位框架.

3.1.3 LDA 模型

LDA 是一个生成式的统计主题模型,同时也是一种典型的词袋模型.它将一篇文档看成是词语之间没有先后顺序和上下文关联的一组词语集合,文档中的这些词语属于多个主题并且每个词语都由其中一个主题生成.LDA 模型在模块化和可扩展方面强于 LSI 模型.与此同时,它在基于主题模型的信息检索应用上十分有效.在 IRBL 领域里,LDA 模型也是在早期研究中就被应用到 IRBL 领域中,并且在近年来仍有部分研究关注于改良 LDA 模型.以下是该模型的相关应用.

2008 年,Lukins 等人[27]就明确提出使用 LDA 模型检索源代码进行缺陷定位.在 2010 年,他们再次使用 LDA 模型来构建 IRBL 方法[3].他们得出结论,基于 LDA 的缺陷定位方法的定位准确度与目标项目尺寸或者源代码的稳定性没有显著关联.因此,LDA 有广泛的应用前景.在 2011 年时,Nguyen 等人[49]提出基于 LDA 主题模型的新方法 BugScout 使得缺陷定位效果有明显提升.2018 年,Wang 等人[98]在他们的方法 STMLocator 中使用了 LDA 模型并且利用代码库的历史信息来进行有监督的学习进一步发掘了 LDA 模型的性能.

3.1.4 其他 IR 模型

除了上述 3 种主流的 IR 模型之外,还有其他一些模型也被某些研究者用来进行缺陷定位.

2010 年,通过聚集和挖掘被相同位置共同引用的历史缺陷报告,Chen 等人[85][112]提出一个有效的使用协同位置收缩(co-location shrinkage,简称 CS)技术的支持向量机模型(CS-SVM)的来检索潜在的缺陷位置.2011 年,Rao 等人[23]使用平滑一元模型(Smoothed Unigram Model,简称 SUM)进行缺陷定位工作.在他们的实验结果中,SUM 的缺陷定位性能比 VSM、LSI 和 LDA 都要优秀.2012 年,Sisman 等人[60]使用 DFR 偏离随机性(Divergence from Randomness,简称 DFR)模型进行缺陷定位.该模型会根据文档特征概率与纯非判别随机分布的差异来评估文档对查询的恰当性.

除了手动构建 IR 模型,还有一些研究使用了现成的信息检索工具.例如,Saha 等人[24]在他们的方法 BLUiR 中使用了 Indri[113], Kilinc 等人[90]和 Rahman 等人[35]则使用 Apache Luence 作为他们方法的检索模型.

3.2 使用特征分析

直接应用 IR 模型进行检索很难得到令人满意的结果,因此需要进一步进行特征分析来改良 IRBL 模型.代码库和缺陷报告都不是由纯自然语言组成的文本,它们具有一些领域相关的结构化信息和特征.对这些特征进

行挖掘可以在一定程度上提升检索结果的准确性.综合现有文献的研究,我们发现对代码库和缺陷报告提取的特征主要包括版本历史、相似报告、代码结构、堆栈踪迹以及其他一些不常用的特征.本小节从提取特征的角度来介绍现有研究如何通过对特征进行细致地挖掘来提升定位模型的性能.

Table 3 Summary of representative IR-based bug localization models

表 3 基于信息检索的缺陷定位代表性模型汇总

年份	方法名称	定位级别**	IR 模型	使用特征					引用
				版本历史	相似报告	代码结构	堆栈踪迹	其他特征	
2010	LDA	M	LDA						Lukins 等人[34]
	LSI	M	LSI						Nichols 等人[33]
	CLS	F	SVM		●				Chen 等人[85]
2011	BugScout	F	LDA		●				Nguyen 等人[49]
2012	BugLocator	F	rVSM		●				Zhou 等人[28]
	TFIDF-DHbPd	F	DFR	●					Sisman 等人[60]
2013	BLUiR	F	Indri		●	●			Saha 等人[24]
2014	Lobster	Cl	VSM			●	●		Moreno 等人[25]
	VSMcomposite	F	VSM	●	●	●			Wang 等人[50]
	BRTracer	F	rVSM		●		●	文本分段	Wong 等人[30]
	AmaLgam	F	Mixed	●	●	●			Wang 等人[31]
	LtR	F	VSM	●	●	●			Ye 等人[47]
2015	Rahman*	F	rVSM	●	●			文件名	Rahman 等人[103]
2016	BugCatcher	F	Lucene			●			Kilinc 等人[90]
	AmaLgam+	F	Mixed	●	●	●	●	报告者	Wang 等人[80]
	MBuM	M	mVSM						Rahman 等人[92]
	Locus	F/Ch	VSM	●		●			Wen 等人[32]
2017	BLIA	File	rVSM	●	●	●	●	文本分段	Youm 等人[74]
	Khatiwada*	File	PMI/NGD						Khatiwada 等人[75]
2018	Loyola*	Ch	Word2vec	●					Loyola 等人[95]
	STMLocator	F	LDA	●			●		Wang 等人[98]
	TraceScore	F	VSM	●	●	●			Rath 等人[63]
	BLZZARD	F	Lucene				●		Rahman 等人[35]
	ConCodeSe	F	VSM			●	●	文件名	Dilshener 等人[79]
2019	FineLocator	M	Word2vec	●		●			Zhang 等人[78]
	D&C	F	rVSM	●	●	●	●		Koyuncu 等人[68]

●表示该方法用到了某特征

*这些方法在原文中没有被命名,因此用第一作者命名

**定位级别缩写 F:File(文件), M:Method(方法), Cl:Class(类), Ch:Change(变更)

3.2.1 版本历史

软件项目在演化过程中会通过版本更迭来实现对项目的更新和完善(例如,修复缺陷、添加新功能等).每一次变更都会向代码库中添加丰富的历史信息(例如,日志信息、文件变动信息等).与此同时,多数缺陷正是在代码变动的过程中因为开发人员主观疏忽或者是客观设计不合理而被引入到项目中,所以分析版本这些历史信息有利于为缺陷定位或者预测任务带来一些提示性的帮助.在现有的 IRBL 方法中,很多方法已经将版本历史信息作为一个重要的特征组件来提升方法的性能.版本历史特征的应用示例过程如图 4 所示.

2010 年,Chen 等人[85][112]利用历史缺陷报告和历史被修复的模块信息为新收到的缺陷寻找可能的位置,其中特别利用了缺陷报告之间的共位置关系.在他们的方法中,如果两个缺陷报告的修复模块相同,那么认为这两个缺陷报告有共位置关系.

2012 年,Sisman 等人[60]挖掘版本历史时应用到以下两点先验知识,1)在短期内被多次提交的修改很可能会包含缺陷;2)历史上包含缺陷的文件在之后版本中可能仍然有缺陷.其中,先验 1)也被应用在 Rahman 等人[103]的方法中.相比前两个方法,Wang 等人[31]和 Youm 等人[74]考虑到新发现的缺陷通常是由最新的提交引入而非远古提交.因此,在他们的方法中只考虑近期的版本历史而完全丢弃距提交的新缺陷报告超过 k 天的版本历史信息.

2013 年,Tantithamthavorn 等人[29]挖掘代码文件之间的共同变更历史信息调整现有方法 BugLocator[28]的结

果.挖掘步骤如下:首先找出先前所有被修复的文件并根据其所属的变更构建一个共同变更矩阵,矩阵中的元素代表两个文件之间的共同变更一致性;然后,对 BugLocator 生成的列表从共同变更矩阵寻找一致性较高的文件提升它们的排名.相比之下,共同变更历史信息可以对原方法的结果产生显著提升.

2016 年,Wen 等人[32]在产生缺陷之前版本历史中把每个变更块(change hunk,包括变更行的内容,变更日志等)索引为一个文档并从这个文档构建自然语言和代码实体两个语料库.此外,提取每个源文件的缺陷修复历史后计算一个加强得分并将其用作指示源文件包含缺陷可疑度的另一个特征.他们的定位模型不仅可以在文件级别定位缺陷同时首次提出在变更级别定位缺陷,即定位引入缺陷的变更.这可以帮助开发人员更容易找出产生缺陷的原因.

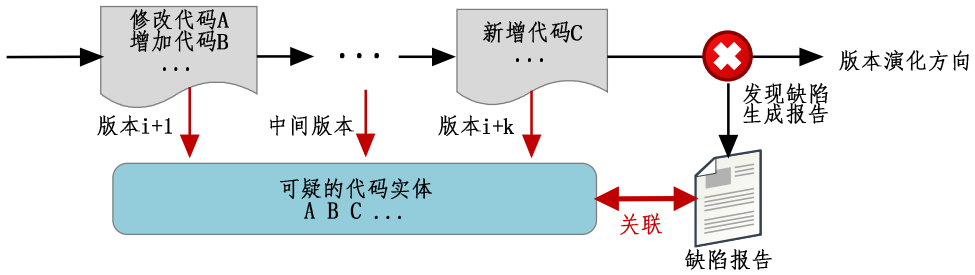

Fig.4 Application sample of features: version history

图 4 特征应用示例: 版本历史

3.2.2 相似报告

相似缺陷报告是另一类重要的辅助特征.一个直观的理解是若两个缺陷报告在描述上具有较高的相似度,那么这两个报告所对应的缺陷文件有很大概率有重叠的部分甚至完全一致.在这个基础上,可以先收集已经定位到产生缺陷位置(例如,源文件)的相似缺陷报告,适当增加这些位置的可疑度分值可以在一定程度上提升 IRBL 结果的性能.相似报告特征的应用示例过程如图 5 所示.

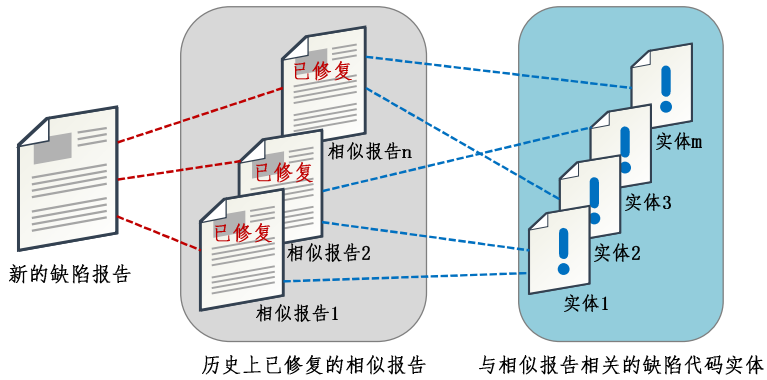

Fig.5 Application sample of features: similar bug reports

图 5 特征应用示例: 相似报告

2012 年,Zhou 等人[28]在他们的方法 BugLocator 中使用到了相似缺陷报告特征.对于新收到缺陷报告,他们首先在 rVSM 模型的基础上计算出该缺陷与每个源文件的文本相似度分值;然后,计算该报告与历史上已经解决的缺陷报告之间的相似度;接着,对历史上有缺陷的文件根据其对应的历史缺陷报告进行加权得到这部分文件的历史特征分数;最后结合两部分的分数将所有文件倒序排列作为最终的推荐结果.

Zhou 等人使用相似缺陷报告特征的方法在后续多个研究工作中被应用.Saha 等人[24]在 BLUiR 变体中、Wong[30]等人在 BRTracer 变体中、Wang 等人在 AmaLgam[31]和改良版 AmaLgam+[80]中都以同样的方式使用到

了相似缺陷报告信息.2017 年,Youm 等人[74]在 BugLocator 的基础上用到了历史缺陷报告中的评论信息(Comment)来强化定位方法.

3.2.3 堆栈踪迹

堆栈踪迹是缺陷产生时代码执行的异常信息.这些信息是高度结构化的,其中包含了异常的类型(例如,空指针异常和地址超界等)和出错代码所执行的路径信息(例如,代码行、类名、方法名).因为缺陷所在的位置很可能在于这个出错路径上,所以堆栈信息对于开发人员手动进行缺陷定位是十分重要的.而在研究定位工具时,也可以通过提取堆栈中的信息(例如,出错的类名)进一步强化工具的定位准确度.堆栈踪迹特征的应用示例过程如图 6 所示.

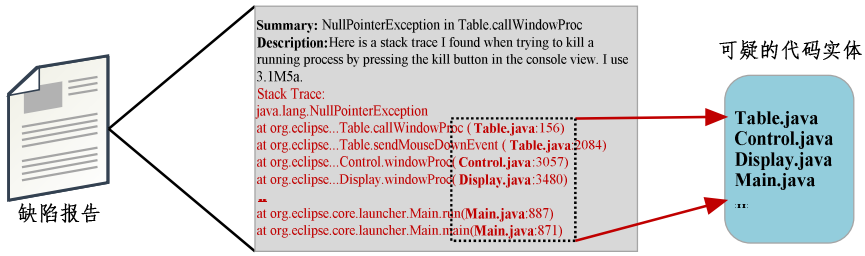

Fig.6 Application sample of features: stack trace

图 6 特征应用示例: 堆栈踪迹

2014 年,Moreno 等人[25]考虑将堆栈踪迹信息应用到他们的方法 Lobster 中.他们使用缺陷报告中的堆栈踪迹和软件源码中的程序依赖图来寻找在结构上相似的代码元素.具体说,对于给定的堆栈踪迹和代码元素,他们之间的结构相似性被定义为堆栈踪迹中元素和代码中元素的最小距离.同年,Wong 等人[30]使用正则表达式从缺陷报告的堆栈中提取所有的文件名以及对应的方法.在得到一组可疑的文件集合后,将上述文件对应方法中直接使用到的类所对应的文件也加入到可疑文件集合,最后通过提高对这些文件在结果中的排名来改善定位结果. 2017 年,Youm 等人[74]在他们的方法 BLIA 中集成了 Wong 等人对堆栈踪迹的处理方法.

2016 年,Wang 等人[80]在他们的改良方法 AmaLgam+使用了堆栈踪迹特征.他们提出一个直观假设,若某个类的引用距离堆栈顶部越近(出错位置),那么这个引用所在的文件越可能包含该缺陷.因此他们设计一个堆栈分数来度量堆栈中出现的每个文件的可疑程度.对每缺陷报告中的堆栈,他们首先按序提取出所有文件名并进行去重处理,然后使用每个文件排名的倒数作为该文件与缺陷报告在堆栈上的可疑程度.

2018 年,Rahman 等人[35]利用堆栈踪迹来重新查询.他们从堆栈踪迹中提取代码实体(类名和方法名).根据执行顺序构造出权重图并在图上面应用 PageRank 算法计算出每个代码实体的权重来重新构造查询语句.

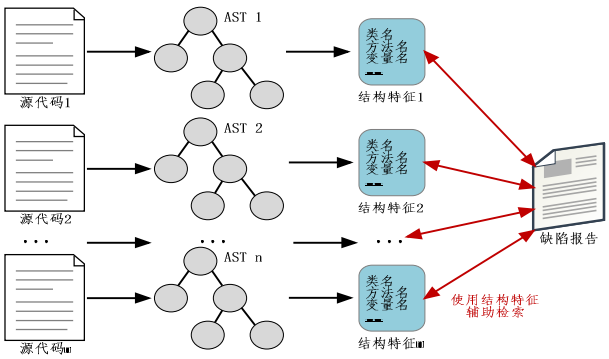

Fig.7 Application sample of features: code structure

图 7 特征应用示例: 代码结构

3.2.4 代码结构

项目中的源代码是一种结构化文本,它的内容是由各种不同的代码实体(例如类名、方法名等)组成.直接将它们看作自然语言文本处理会丢失其中的结构化信息会导致定位结果准确率低下.合理利用这些结构特征可以帮助进一步提升自动工具的准确性.代码结构特征的应用示例过程如图 7 所示.

2013 年,Saha 等人[24]在设计模型时首先考虑到代码结构特征.对于代码库来说,他们使用 Eclipse JDT 解析源代码的 AST 树并提取其中的四种代码实体(类名、方法名、变量名和注释)信息;对于缺陷报告来说,他们分别使用标题和描述构建两种查询.上述代码实体和查询共有八种不同的组合方式,他们分别计算每种组合的分数然后将所有组合的分数相加作为某个源文件的最终分数.最后他们依据该分数向开发者推荐有缺陷的文件.

2016 年,Kilinc 等人[90]提出 BugCatcher 方法,该方法首先使用基础的检索方法对源代码进行检索获得一个排序结果.然后,从代码中提取类名、方法名和注释分别建立索引并且根据这三类信息对首次结果进行重排.最后使用虽小范围技术和重新索引计算最终结果.同年,Wen 等人[32]提出 Locus 方法.该方法对自然语言和代码实体分别构建语料库和查询语句,将两个查询结果组合起来输出变更块(change hunk)的排序并在此基础上选出可以的文件或者变更.

2018 年,Dilshener 等人[79]提出一种不需要历史信息仅使用结构和堆栈信息的缺陷定位方法.同年,Swe 等人[99]将代码结构细分为类名、方法名和变量名分别处理来避免代码文件过大对结果带来的影响.Rath 等人[97]研究了缺陷报告中的结构信息对 IRBL 方法的影响.他们的结果表明堆栈踪迹会倾向于降低缺陷定位的性能并且需要额外的处理.

3.2.5 其他特征

除了上述四种被大量使用的特征之外,还有一些现有研究[30][74][79][80][103]挖掘了源代码和缺陷报告中的其他隐藏的对缺陷定位有利特征.这些工作也在一定的场景下改进了 IRBL 方法并提升了它们的定位性能.

2014 年,Wong 等人[30]考虑到缺陷通常发生在源文件中的小部分代码里,而某些源文件的尺寸很大会严重降低 VSM 模型的准确性.因此,他们将每个的源文件按照某一阈值(例如,100 行代码)分为若干大小相等片段,并在此基础上计算每个片段与缺陷报告的相似性.然后用得分最高的片段代表该文件.分段操作消除了文件大小对定位模型的影响.该特征在 2017 年也被 Youm 等人[74]应用在他们的方法 BLIA 中.

在某些缺陷报告中会记录代码库里出现的源文件名或者方法名等代码实体信息,这些信息往往与缺陷关联密切.因此,Rahman 等人[103](在 2015 年)和 Dilshener[79](在 2018 年)在他们的定位方法中提高了这些出现在缺陷报告中的代码实体所对应的源文件或者源方法的在最终排序列表中的优先级来获取更加可靠的结果.

2016 年,Wang 等人[80]提出了使用缺陷报告中的报告者信息来提升定位性能的方法 AmaLgam+.他们的直观理解是,一个报告者很可能去报告与相同/相似的代码组件有关的缺陷.因此,当受到新的缺陷报告时,查看该报告者之前提交的缺陷报告和对应的包含缺陷的文件对于新缺陷的定位有帮助.

3.3 进行查询重构

在一定程度上来说,由于大量研究已经对代码库和缺陷报告中的相关特征进行了比较充分的挖掘,很难再提取新的特征来改良 IRBL 方法.研究者从改善查询的角度提出了查询重构(Query Reformulation,简称 QR)的策略来提升性能.这类策略的一个优势是独立于 IRBL 方法本身,可以作为一个预处理步骤集成到现有的 IRBL 方法中,因此它的可移植性较好.查询重构的应用场景如图 8 所示.开发者首先使用自动方法构建初始查询并获取一个结果列表.当他们在排名靠前的代码中没有找到缺陷时就请求进行查询重构重新获取推荐结果.

需要重构的对象是用来构建查询语句的缺陷报告,因此,对缺陷报告的组成部分及特征有全面的理解十分重要.在实际使用过程中,开发者也是需要根据初始查询的反馈结果来决定是否需要使用查询重构策略.因此对缺陷报告的分析是进行查询重构的前序工作.根据研究的前后承接关系,我们将这部分相关内容分为缺陷报告分析和查询重构策略两个部分来介绍.

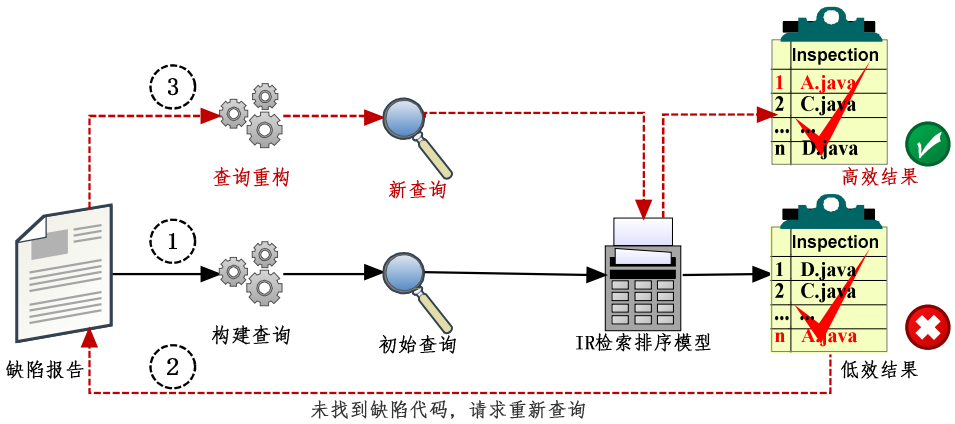

Fig.8 The scenario of query reformulation

图 8 查询重构的应用场景

3.3.1 缺陷报告分析

2008 年,Bettenburg 等人[70][114]对现实项目中(例如,Apache)的开发者和用户进行一项调研.他们发现用户提交的报告内容与开发者希望收到的信息之间不匹配.开发者想要得到的信息是复现步骤、错误堆栈和测试用例等,然而用户很难提供这些信息.为了解决这个问题,他们提出一个原型工具 CUEZILLA 来测量缺陷报告的质量并且能够向用户推荐填写缺陷的信息来提升报告质量.

2014 年,Kochhar 等人[48]总结了会影响缺陷定位有效性的潜在偏好:1)被错误分类的缺陷报告.被标记为缺陷的问题报告并非总是由缺陷引起,也包含了文档更新、代码重构等;2)已经可定位的缺陷报告.缺陷报告中可能已经指出了有缺陷的程序文件,没有必要对这些报告使用缺陷定位工具;3)不正确的真实值.在修复缺陷的更新提交中被修改的文件中,可能并不是所有的文件都与这个缺陷有关.他们的实验结果表明偏好 2)对 IRBL 有效性有显著影响,因此应当在后续研究中被纠正.

2018 年,Mills 等人[53]对缺陷报告进行调研,他们发现缺陷报告中的大多数词汇的确可以用来构建有效查询语句,从而使得在没有辅助查询扩展的情况下可以成功进行缺陷定位.在此基础上,他们设计了一种遗传算法(GA)来分析带有和不带有定位提示(localization hint, 例如,方法名)的缺陷报告文本以获得一个接近最优的查询,该查询提高了缺陷报告在 IRBL 应用中的潜力.然而在同年,Lawrie 等人[52]研究了缺陷报告在 IRBL 中的价值.他们指出 GA 算法存在作弊(cheats)问题,因为 GA 算法根据查询的性能评估值来产生优质的查询.然而在现实场景中一个查询的性能评估值是未知的,因此不具备实用性.

2019 年,Rath 等人[84]着重分析了缺陷报告中结构化信息对 IRBL 方法的影响.他们发现缺陷报告中的堆栈踪迹信息更倾向于对缺陷定位结果产生负面影响,因此对包含这类信息的报告需要特别处理.与此同时,相比于只包含自然语言的缺陷报告,那些包含源代码信息的缺陷报告不仅可以提升 BL 算法的性能而且有助于开发人员更快找到缺陷位置.

3.3.2 查询重构策略

2013 年,Sisman[61]首先将查询重构策略应用到缺陷定位领域中.在不需要用户提供额外的输入信息的情况下,他们的 QR 框架的工作原理是先进行一次初始查询得到相关的软件制品(artifact,例如,文件)排名列表,然后从排名靠前的制品中抽取附加的词汇来扩充查询语句.

2017 年,Chaparro 等人在[40][46]中将缺陷报告的内容详细地划分为 OB(Observed Behavior)、EB(Expected Behavior)、S2R(Step to Reproduce)3 个部分,他们使用启发式的规则识别和检测这三部分内容并发现多数缺陷报告中缺少 EB 和 S2R 这两部分内容.接着,他们在[51]中进行对比实验发现使用 OB 替代整个报告的内容来构造查询语句是一种简单有效的方法来重构低质量的缺陷报告.在此基础上,Chaparro 等人[73]在 2019 年了扩大了

实验对象和数据集,发现使用 OB 和报告标题的组合内容替代整份缺陷报告来定位能够达到更好的效果.

2018 年,Rahman 等人[35][42]提出使用一种上下文感知的查询重构方法 BLZZARD.他们的重构方法对缺陷报告中的两类重要信息堆栈踪迹和程序元素(例如,类名、方法名等)分别构造踪迹图和文本图模型对这两部分中的词汇进行建模.在构造好的图模型基础上,使用 PageRank 算法对图中的词汇进行加权.并根据权重来选取加入查询语句的词汇.

2019 年,Kim 等人[102]提出目前最新的自动查询重构方法.他们的方法包含三个组件:1)缺陷报告扩展;2)候选词提取;3)查询扩展.首先,他们使用缺陷报告中的附件信息(attachment)扩展缺陷报告.然后,在提取候选词时通过将情感词汇加入到停用词表来将它们移除并且使用伪相关反馈的方式从源码中检索文件并移除噪音文件.最后,他们对检索到文件中的词汇加权并选择权重较高的部分来扩展查询语句.

3.4 应用深度学习

随着深度学习技术的发展,近年来研究者们将深度学习技术引入缺陷定位领域.相较于基于信息检索的缺陷定位只考虑了缺陷报告和源代码文本上的特征,基于深度学习的缺陷定位方法可以提取缺陷报告和源代码文件更深层和更高维的抽象特征.这类方法可以减少人为设计特征的工作量,同时也提高了缺陷定位的准确率.

2017 年,Lam 等人[55]将深度神经网络(DNN)与信息检索技术中的 rVSM 相结合进行缺陷定位.首先,他们使用 rVSM 收集缺陷报告和源代码文件之间文本相似性的特征.然后,他们用 DNN 来学习将缺陷报告中的术语并将它们与源代码中潜在不同的代码字段关联起来.实验证明,深度神经网络和信息检索技术可以很好地相互补充,能够实现比单个模型更高的缺陷定位精度.

2018 年,Xiao 等人[64]提出一个深度学习方法在字符级别解释缺陷报告并使用语言模型来进行分析.该方法类似于机器翻译,首先将缺陷报告和源代码文件用字符表示.然后将结果传入到 CNN 模型中进行卷积操作.最后把 CNN 的结果应用在基于 RNN 的编码器-解码器中进行缺陷定位.

同年,Xiao 等人[96]使用卷积神经网络和级联森林对语义信息和结构特征建模.首先采用具有多个过滤器的卷积神经网络和具有多粒度扫描的随机森林集合,从缺陷报告和源文件中提取词向量的语义和结构特征.随后从级联森林进一步提取更深层次的特征,并观察缺陷报告与源代码文件之间的相关关系.

2019 年,Xiao 等人[77]提出使用词嵌入和强化的卷积神经网络来提升缺陷定位性能.他们使用词嵌入来表示缺陷报告和源文件中保留语义信息的单词,并使用不同的CNN模型来提取它们的特征.Polisetty等人[101]研究了基于深度学习的缺陷定位模型是否能够满足参与人员的需求.他们的结果表明,虽然深度学习模型比经典的机器学习模型表现得更好,但它们只能部分满足研究者实验中设定的采用标准.Huo 等人[69]提出了一种跨项目缺陷定位的深度迁移学习方法.使用他们所提出 TRANP-CNN 方法可以从源项目中提取出可转移的语义特征,充分利用目标项目中的标记数据进行有效的跨项目缺陷定位.

3.5 小结

本节从更换检索模型、使用特征分析、进行查询重构和应用深度学习四个方面详细介绍了在改良 IRBL 模型近年的研究进展.下面简述主要发现.

(1) 相比于其他 IR 模型,VSM 作为一种简单的 IR 模型能够在 IRBL 的研究中获得较好的性能.因此,目前超过半数的 IRBL 方法都使用 VSM 或者 VSM 的变体作为他们方法中 IR 模型.

(2) 从代码库和缺陷报告中挖掘出的特征里,对提升缺陷定位方法性能十分有效的是版本历史、相似报告、代码结构和堆栈踪迹这四种特征并在大量研究中被以不同的方式集成到他们的 IRBL 方法中.

(3) 查询重构是一种独立于具体定位方法的优化策略.因此,它可以作为一个预处理步骤集成在任何现有的 IRBL 方法中,具有很好的可移植性.

(4) 将深度学习模型应用到基于信息检索的缺陷定位中可以自动地从代码和缺陷报告中提取特征.最新的研究显示了深度学习模型在缺陷定位应用中的十分有效.

4 IRBL 模型的评估

本节介绍 IRBL 模型评估方向的研究进展,根据对现有文献的分类,主要从模型比较、评价指标和实验数据三个部分展开介绍.

4.1 模型比较

研究者在提出新的 IRBL 方法时会设计实验来验证新方法的有效性.然而通常的情况是,不同研究者在设计实验时不能够完全确保实验的一致性(即,与基准方法使用的实验设置是否一致,是否正确复现了基准方法)与合理性(即,实验设置是否存在问题).这些因素可能会使得实验结果不能准确显示新方法的真实性能,从而误导后续的研究工作.

为了获得比较真实客观的实验结果,一些研究工作致力于设置公平的实验对已有定位方法或者 IR 模型的效果进行比较和评估的实证研究,我们简称为模型比较研究.这类研究的好处在于以下三个方面[89],1)对研究者,帮助他们理解现有模型的优缺点和真实效果并在此基础上研究更有效的定位方法;2)对缺陷定位编程人员,帮助他们理解如何更好的使用现有方法来达到理想的定位结果;3)对缺陷报告者来说,帮助他们在提交缺陷报告时填写对定位缺陷最有用的信息来加强缺陷定位的成功.

4.1.1 对不同参数性能的比较

2011 年,Thomas 等人[66]在超过 8000 个缺陷报告上实证调查了分类器参数(总共 3172 种参数设置)和不同分类器组合对缺陷定位的影响.他们主要得出两个结论,1)分类器的参数设置对性能有显著的影响;2)使用不同分类器组合的结果优于使用任意单独的分类器.

2018 年,Tantithamthavorn 等人[76]研究了 IR 分类器的配置参数对方法级别缺陷定位的性能和工作量的影响.他们的主要结论如下,1)即使在分类结果的排序性能相似的情况下,不同的参数对分类结果的工作量有十分显著的影响;2)在方法级别表现较好的参数设置可以应用到文件级别,反之亦然.他们最后强调在评估方法时应当考虑审查结果所花费工作量.

4.1.2 对不同模型性能的比较

2011 年,Wang 等人[19]在一个大型的 Linux 内核数据集上比较了 10 种不同 IR 技术的关注点定位(concern localization)方法.他们的结果显示,1)简单的 IR 模型(如,VSM 和 SUM)比复杂的模型(如,LDA)定位效果要好;2)相同的 IR 技术在不同系统上处理不同应用时的表现有差异;3)基于 IR 的关注点定位技术在大型软件系统中和小型软件系统中一样有效;4)现有的 IR 技术在处理软件语料库时效果差于处理自然语言的语料库.

2015 年 Alduailij 等人[89]使用统计推断比较三种文本模型(VSM、LSI、LDA)在方法级别缺陷定位时的性能.他们得出结论是 VSM 是比较好的模型.接着,他们研究了额外参数对 VSM 性能的影响,包括方法长度、查询长度、方法的文档注释以及缺陷报告中提及的产品名称和组件名称.他们发现 VSM 与大多数被测的参数正相关.

2018 年,Shi 等人[94]对混合缺陷定位方法(Hybrid Bug Localization,即同时使用 IR 相似度和附加特征的方法)进行研究.他们比较了 8 种不同的 LtR 技术在使用 4 种归一化方法时的性能表现.他们发现 LtR 的表现好于最新的 BLUiR[24]和 AmaLgam 方法[31].

4.1.3 对不同实验方案的比较

一些研究者指出在进行模型比较的实验验证过程中存在不合理的设置.这些不合理的设置会影响实验结果的准确性.2018 年,Kim 等人[41]发现现有研究的实验数据集中包含一些 non-buggy 文件(例如,测试文件),他们指出将这些文件包含在数据集中会影响现有技术的可靠性,所以在以后实验方案中应当被去掉.根据他们在排除测试文件的数据集上的实验结果, 被多数研究者作为基准方法的 BLUiR[24]的实际定位效果并没有原文实验中表现得那么好. Kim 等人[41]指出的不合理的实验方案在得到认同,例如,Lee 等人[39]在他们的实证研究中排除了数据集中的测试文件.

4.2 评价指标

评价指标是度量方法有效性的重要依据.针对不同模型的输出结果,研究者使用不用的类型的指标对结果进行评价.本节从排序性能、分类性能和工作量三个角度介绍 IRBL 领域的评价指标.

4.2.1 评估排序性能的指标

目前大多数 IRBL 领域的研究都将缺陷定位视为一个排序任务,对于给定的缺陷报告,定位方法会输出一个根据其包含缺陷可能性从大到小排序的文件列表.因此,大量研究使用排序指标来评价 IRBL 方法的性能.表 4 对排序性能评估指标的计算方法和使用情况进行了汇总.这些指标的具体含义如下:

(1) Top@K.表示对缺陷定位方法生成的推荐列表中前 k 个文件进行审查时缺陷定位方法定位成功的概率.特别地,给定一个待定位的缺陷报告,如果缺陷定位方法生成的推荐列表的前 k 个源代码文件中至少有一个与给定的缺陷报告相关,则认为定位成功;

(2) MRR (Mean Reciprocal Rank).测量的是缺陷定位方法定位到的首个与缺陷报告相关的源代码文件在排序列表中的位置;

(3) MAP (Mean Average Precision).测量的是缺陷定位方法定位到的所有与缺陷报告相关的源代码文件在排序列表中的位置;

(4) E (Effectiveness).表示在找到一个与缺陷报告相关的文件之前最少需要被开发者审查的源文件数目.

前三个指标的值越大说明包含缺陷的文件在结果列表的位置越靠前,即方法的排序性能越好.而 Effectiveness 的值越小方法的性能越好.从表 4 可以看出,其中 Top@K、MRR 和 MAP 是应用最多的三种指标,分别在 40、38 和 46 篇文献中被用来评估 IRBL 方法.

Table 4 Summary of evaluation metrics for IRBL models in terms of ranking performance

表 4 评价 IRBL 模型的排序性能指标汇总

评估指标	计算公式	说明	使用列表
Top@K (TopK 召回率)	$Top@K = \frac{\lvert R_k \rvert}{n}$	n 表示评估过程中使用的 bug 报告的总数; $\lvert R_k \rvert$表示缺陷定位方法进行 TopK 推荐时, n个缺陷报告中定位成功的缺陷报告的数量.	[24][26][28][29][30][31][32][34][35][38][42][45][47][49][50][54][55][63][65][66][67][68][69][73][74][75][77][78][80][84][90][92][94][95][97][98][99][100][102] [112]
MRR (首位倒排均值)	$MRR = \frac{1}{n}\sum_{j=1}^{n}\frac{1}{rank_j}$	$rank_j$表示第j个缺陷报告所对应的列表中第一个与缺陷报告相关的源码的排名.	[24][25][26][28][30][31][32][35][39][41][47][50][54][55][63][65][68][69][73][74][75][77][78][79][80][84][89][90][92][93][94][95][96][97][98][99][100] [102]
MAP (平均准确率均值)	$MAP = \frac{1}{n}\sum_{j=1}^{n} AvgP_j$ $AvgP_j = \frac{1}{\lvert K_j \rvert}\sum_{k\in K_j}\{Prec@k\}$ $Prec(k) = \frac{\sum_{i=1}^{k} IsRelevant(k)}{k}$	对第j个报告有一个推荐列表l; $Prec@k$表示l的前k个文件中与缺陷报告相关的源码文件的比例; $IsRelevant(k)$为 1 表示列表l中第k个源代码文件与缺陷报告相关,否则无关; K_j表示列表l中所有与缺陷报告相关的源代码文件位置集合.	[23][24][25][26][28][30][31][32][35][38][39][41][45][47][48][50][54][55][56][60][61][63][65][67][68][69][73][74][75][77][78][79][80][84][87][89][90][92][93][94][95][96][97][99][100][102]
E (有效性)	$E(q) = \min(\forall_{r\in R_q} rank(r))$	R_q表示查询所有与缺陷相关的文件的集合; $rank(r)$表示r在列表中的排名.	[18][25][27][34][35][53][59][98][104] [105]

4.2.2 评估分类性能的指标

在信息检索领域,另一类重要的指标是用来评价一个方法的分类性能.然而由于 IRBL 的输出结果是一个序列,分类指标不能被直接用于对这些方法进行评价.现有部分研究[60][61][62][86]的做法是设置一个排名阈值将该序列分成两种类型,即在排名之前的文件包含缺陷,反之不包含缺陷.

从二分类的角度来看,IRBL 方法的定位结果总共可以划分为以下四种可能情况.

- 对于一个被定位为包含该缺陷的文件,它的确包含该缺陷,这种结果属于 TP(True Positive);
- 对于一个被定位为包含该缺陷的文件,它并不包含该缺陷,这种结果属于 FP(False Positive);
- 对于一个被定位为没有该缺陷的文件,它的确包含该缺陷,这种结果属于 FN(True Negative);

- 对于一个被定位为没有该缺陷的文件,它并不包含该缺陷,这种结果属于 TN(True Negative);

现有的二分类评价指标是基于上述四种情况的统计技术来计算的.表 5 对这些分类性能评估指标的计算方法和使用情况进行了汇总.这些指标的具体含义如下:

(1) Precision (精度). 在预测为有缺陷的文件中,有多少比例的文件真正与该缺陷相关;
(2) Recall (召回率). 在所有与缺陷相关的文件中,有多少比例的文件被定位为与该缺陷相关;
(3) F1-score (调和平均值). 精度和召回率的调和平均值,可以更加客观地评价分类结果;
(4) Accuracy(准确率)[85] 在所有分类结果中,正确分类的文件(包括正确分为与缺陷相关和正确分为与缺陷无关的文件)占所有文件的比例;
(5) MCC (Matthews Correlation Coefficient,马修斯相关系数)[82] .表示的是真实类别和预测类别的二元分类之间的相关系数.

这些指标的计算与设定的排名阈值有关,即在排名阈值之前的文件被定为包含某缺陷,反之不包含该缺陷.因为在没有阈值的情况下,任何结果是没有意义的.例如,精度都为1.0而精度都为$1/k$(k是项目中所有文件的数目).在这些指标中使用最多的是精度和召回率,它们都在 11 篇文献中被使用过.

Table 5 Summary of evaluation metrics for IRBL models in terms of classification performance

表 5 评价 IRBL 模型的分类性能指标汇总

评估指标	计算公式	使用列表
Precision (精度)	$Precision = \frac{TP}{TP+FP}$	[13][39][60][61][62][71][72][75][82][86][87]
Recall (召回率)	$Recall = \frac{TP}{TP+FN}$	[13][29][39][60][61][62][71][72][75][82][86][87]
F1 (调和平均值)	$F1 = \frac{2\times Precision\times Recall}{Precision+Recall}$	[72][86]
Accuracy (准确率)	$Accuracy = \frac{\mathrm{TP}+TN}{TP+FN+FP+TN}$	[29][85]
MCC (Matthews 相关系数)	$MCC = \frac{TP\times TN-FP\times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$	[82]

4.2.3 评估工作量的指标

在缺陷定位领域,研究人员发现上述性能评价指标不能完全满足 IRBL 方法在实际应用中的评估需求.由于开发人员需要花费有限的工作量(时间和资源)来对 IRBL 方法推荐的结果进行人工审查.在此期间,能够以较小工作量的找到缺陷所在位置才能够说明一个方法的实际有效性.因此,研究者们从工作量感知(Effort Aware)的角度来对结果进行评价.

Table 6 Summary of evaluation metrics for IRBL models in terms of effort-aware performance

表 6 评价 IRBL 模型的工作量性能指标汇总

评估指标	计算公式	说明	使用列表
Effort@K (TopK 工作量)	$Effort@k=\frac{\sum_{j=1}^{n}\sum_{i=1}^{k}LOC(f_{ij})}{\lvert R\rvert}$	f_{ij}代表第j个缺陷报告的推荐列表中第i个源文件; $LOC(f_{ij})$表示其对应的代码行;n代表缺陷报告的数目; $\lvert R\rvert$个报告对应的列表中前k个源文件汇总至少有一个包含该缺陷.	[36][37]
MAE (平均工作量均值)	$MAE = \frac{1}{n}\sum_{j=1}^{n}AvgE_j$ $AvgE_j=\frac{1}{\lvert P_j\rvert}\sum_{p\in P_j}\{AvgE@p\}$ $AvgE@p=\sum_{i=1}^{p}\frac{LOC(f_{ij})}{p}$	给定n个缺陷报告,对第j个报告有一个推荐列表l. $AvgE_j$表示对报告j,找出所有相关文件平均所需花费的工作量; P_j表示推荐列表l中所有与缺陷报告相关的源文件位置的集合.	[36][37]
MFE (首位工作量均值)	$MFE=\frac{1}{n}\sum_{j=1}^{n}\sum_{i=1}^{q_j}LOC(f_{ij})$	q_j表示第j个缺陷报告的源代码文件推荐列表中第一个与缺陷报告相关的源代码文件的位置.	[36][37]

表 6 中对工作量性能评估指标的计算方法和使用情况进行了汇总.其中大部分评估指标是 Zhao 等人[36][37]在 2015 年首先提出的.他们使用代码行数来衡量一个文件在审查时所需花费的工作量.其具体含义如下:

(1) Effort@K.由 Top@K 派生而来,衡量了缺陷定位方法推荐对列表中前 K 个源代码文件进行审查时,开发人员找到与缺陷报告相关的源代码文件平均需要花费多少工作量.

(2) MAE(Mean Average Effort).由 MAP 派生而来,表示从推荐列表中找出所有与缺陷报告相关的源代码文件时平均所需花费的工作量.

(3) MFE(Mean First Effort).由 MRR 派生而来,表示从源代码文件推荐列表中找到第一个与缺陷报告相关的源代码文件平均所需要的工作量.它的值越小表示在工作量感知下方法的性能越好.

4.3 实验数据

经过十几年的发展,目前 IRBL 领域已有较多研究者收集并且公开了它们的实验数据集.这对于继续在缺陷定位领域进行研究十分有利.表 7 列出了 IRBL 领域的公开可用数据集的汇总信息.包括数据集提供者、所使用的项目数和提供的缺陷报告数目等.

Table 7 Summary of experiments datasets for IR-based bug localization models

表 7 基于信息检索的缺陷定位模型的数据集汇总

项目语言	定位级别	提供者	项目数	报告数	使用列表
Java	File	Dallmeier 等人[115]	1	223	[23][45][54][60][115]
		Zhou 等人[28]	4	3479	[24][28][29][30][31][32][41][50][64][65][72][74][75][79][86][80][90][91][93][94][99]
		Thomas 等人[66]	2	6716	[66][76]
		Ye 等人[47]	6	22747	[36][47][55][79][95][96]
		Le 等人[45]	3	111	[45][54]
		Le 等人[72]	1	341	[72]
		Wang 等人[98]	3	14121	[98]
		Rahman 等人[35]	6	5139	[35]
		Mills 等人[53]	13	620	[52][53][73]
		Lee 等人[39]	46	9459	[39][73]
		Zhang 等人[100]	10	2279	[100]
	File and Change	Wen 等人[32]	3	347	[32][95]
		Rath 等人[83]	7	25283	[97][83]
	Method	Zhang 等人[78]	5	531	[78]
C++	File	Thomas 等人[66]	1	1368	[66][76]
C	File	Saha[26]	6	7716	[26]
C#	File	Garnier 等人[38]	20	878	[38]

目前 IRBL 领域的数据集的收集和使用过程中仍存在一些问题需要得到重视.

(1) **项目语言偏向于 Java.**目前,绝大多数研究者提供的数据集中的实验项目是由 Java 开发的,其他语言开发的数据集相对较少并且没有被大量引用.例如 Saha 等人[26]收集的 C 语言项目和对 Garnier 等人[38]收集的 C#语言项目都仅在各自的研究中被使用.Java 语言项目受到关注这得益于 Java 语言的流行和相关项目的良好维护.现有研究可以在一定程度上说明目前的方法在对 Java 项目进行缺陷定位时是有效的.然而,不同的语言在进行开发时有各自的特点,仅仅在一种语言的项目上实验的结果不能直接应用到其他语言开发的项目之上.其有效性还需要进一步的实验证实.

(2) **数据集质量各异.** 尽管数据集的收集过程都是按照 Dallmeier 等人[115]提供的过程进行收集.但是具体操作过程会有一些差异从而导致不同研究者收集的数据集存在一定到偏差,在这些数据上进行实验得到的结果可能会对分析带来误差.因此,为了得到客观公正的结果,需要花费更多的工作来审查数据集的质量来确保其准确性.

(3) **实验验证不充分.** 虽然已有较多的数据集,但是从统计信息来看,大多数研究者还是集中于使用个别

数据集(例如,Zhou 等人[28]和 Ye 等人[47])来验证他们的方法.一方面说明现有的多个数据集还未受到研究者的重视,其数据集的可靠性有待于进一步被验证;另一方面表明现有 IRBL 方法仅在个别数据集上获得好的性能,其泛化能力仍有待更多实验数据的证实.

4.4 小结

本节从模型比较、评价指标和实验数据三个方面详细介绍了在近年来对 IRBL 方法进行评估的研究进展.下面简述主要的发现.

(1) 在实际定位中要选择合适的 IRBL 方法.同时应当注意到参数配置对结果的有效性的影响比较大.因此在应用具体 IRBL 方法时应当选择适当的参数;

(2) 现有研究主要是评价 IRBL 方法的排序性能和分类性能,多数研究并没有从工作量感知的角度来评价他们的方法,导致现有的研究方法很难满足实际应用的需求;

(3) 目前,已有不同研究者提供的规模较大的数据集供后人使用.但是仍存在一些问题,例如数据集偏向于 Java 编写的项目.此外,目前大多数研究仍只在少数数据集上测试它们的方法,无法确保其具有良好的泛化性能.

5 缺陷定位的相关研究工作

缺陷定位是一个涉及层面十分广阔的领域.除了可以使用信息检索技术实现软件缺陷定位,还可以通过其他多种途径定位软件中的缺陷.例如基于程序频谱的缺陷定位、基于程序变异的缺陷定位、基于多模态的缺陷定位等.本节中对其他类型缺陷定位的相关工作进行简要介绍.

5.1 基于程序频谱的缺陷定位

基于程序频谱的缺陷定位(Spectrum-based fault localization,简称 SFL)是一类重要的动态定位技术.该类方法在进定位缺陷时需要执行大量的测试用例,然后通过对测试用例的程序频谱应用评估度量来定位错误语句.与基于信息检索的方法相类似的,该方法根据度量公式计算程序组件的分数,并对它们进行排序.基于程序频谱的定位效果依赖于程序频谱的构造方式和度量指标.以下是部分相关研究工作.

2009 年,Rui 等人[116]对早期的用于分析程序频谱的特定相似系数的性能进行了评估.实验结果表明,用于分析程序谱的特定相似系数的优越性能在很大程度上与实验的设计无关.此外,对于低质量的缺陷观察和有限数量的测试用例,基于程序频谱的缺陷定位已经获得了近乎最佳的准确性.实验也证明了基于程序频谱的缺陷定位可以有效地应用于嵌入式软件开发的环境中.

2010 年,Lee 等人[117]提出了一种利用频率加权函数进行缺陷定位的方法.该方法考虑了每个测试用例执行的程序语句的频率执行计数来进一步提高缺陷定位的性能.

2015 年,Naish 等人[88]发现现有的遗传规划方法由于评估度量的巨大搜索空间变得缓慢而不可靠.因此他们提出了一个“hyperbolic”度量的限定类,其中包含少量的数值参数,由此遗传规划算法可以在大量的程序频谱上可靠地发现有效度量从而使程序组件排序更准确.

2019 年,Ribeiro 等人[118]使用数据流频谱评估了十个基于程序频谱方法的排名指标的有效性和效率.该实验证明使用数据流频谱进行分析比使用控制流频谱分析效果更好,最高有 50%的缺陷位于排序列表的前 15 位.同年,Ma 等人[119]提出了一个统一的系统研究框架来评估和比较单缺陷和多缺陷情况下的基于程序频谱的缺陷定位方法.该框架实现了一个通用向量模型来深入理解各种基于程序频谱的缺陷定位方法.

5.2 基于变异分析的缺陷定位

基于程序变异的缺陷定位(Mutation-based fault localization,简称 MFL)是一种使用变异算子模拟人为错误来检测未知错误的定位方法[12][120-124].根据变异体与被测程序的执行结果,利用公式来推算代码语句的出错可能性.通常来说,这类方法在语句级别生成变异体,从而能够比较准确地定位出程序中有缺陷的语句.同时,由于需对大量程序变异体执行测试用例集,其执行开销较大.以下是部分相关研究工作.

2015 年, Papadakis 等人[125]首次提出了基于变异分析的缺陷定位的概念.他们提出的方法利用各个变异体间的相似性来检测缺陷.首先,使用变异算子为待测程序生成变异体.然后,对原程序和变异体执行测试用例并获取执行结果.接着,对执行结果构建特征计算变异体包含缺陷的风险分数.最后,根据相同位置的变异体与原程序语句风险分数来预测可能包含缺陷语句的位置.

同年,为了能够很好地处理真实世界中多语言程序的缺陷定位问题,Hong 等人[120]提出了一种基于变异的缺陷定位技术.该技术以目标程序的多语言源代码和一组测试用例作为输入生成一个按照包含缺陷风险值排序的语句列表.为了计算每个语句的风险值,该技术首先通过系统地改变每个语句来生成原程序的不同变体.接着根据语句发生特定变异后测试结果变化情况来计算风险值.值得注意的是,为了提高对多语言程序缺陷定位的准确性,他们提出了适用于多语言程序的变异算子.

由于 MFL 技术的计算开销较大,Liu 等人[121][122]在 2017 年提出了两种变异约简策略来降低 MFL 技术的计算成本.一种是面向语句的变异约简策略[121],该策略首先用全套变异算子在失败测试用例覆盖的每个语句上产生变异以确保能够使用完整类别的变异算子.然后利用每个语句上的变异体抽样策略选择使用特定百分比的变异体.另一种是动态变异执行策略[122],该策略会确定优先执行的变异体和测试用例首先执行,避免进行无用的执行来降低计算开销.

2018 年,Oliveira 等人[123]提出一种面向失败测试的变异运行策略 FTMES 来提高基于变异的缺陷定位的有效性并且降低其所需的计算开销.FTMES 只使用一组失败的测试用例来执行变异,并且通过使用覆盖率数据替换终止信息来避免执行已通过的测试用例.

5.3 基于多种模态的缺陷定位

很多方法只考虑了某一种类型的缺陷定位技术.比如基于信息检索的方法只考虑了缺陷报告中的文本信息及静态特征;基于程序频谱的方法只考虑动态的程序频谱特征.很多研究者发现,同时结合两种或以上类型的缺陷定位技术能够发挥出各自的优势从而能够获得更好的缺陷定位效果.以下是部分相关研究工作.

2015 年,Le 等人[45]将多模态信息检索与程序频谱组合进行缺陷定位并且自适应地创建了一个特定于某个缺陷的模型,以将特定的缺陷映射到它可能的位置.该方法同时考虑了缺陷报告中的文本描述和程序的频谱信息.最后结合两者的分析结果获得可疑的术语或单词,并且以此来判断程序组件包含缺陷的可疑程度.

2017 年,Dao 等人[54]研究了代码运行信息对基于信息检索的缺陷定位的帮助.作者在文中比较了 3 种目前流行的基于信息检索的缺陷定位方法(BugLocator[28], BLUiR[24], AmaLgam[31])和 3 种代码运行信息(程序覆盖、程序切片和程序频谱).实验证明代码的运行信息会很大改进基于信息检索的缺陷定位方法.

2019 年,Hoang 等人[67]也发现了只考虑缺陷报告和只考虑执行跟踪的方法的限制.他们提出了一种多模态缺陷定位方法,通过联合使用缺陷报告和执行跟踪来解决现有方法的存在的问题.

5.4 基于新型应用的缺陷定位

目前大多的 IRBL 方法在进行缺陷定位时并没有细致考虑到缺陷的类型,他们的研究是针对所有类型的缺陷而言.在现实应用中进行缺陷定位要面临不同的应用场景,此时直接使用这些方法可能无法得到预期效果.因此有部分研究开始在更加细分类别的缺陷上研究定位方法.

对软件崩溃的定位.软件崩溃开始研究一种特殊的缺陷,这种缺陷经常发生在用户使用软件的过程中,它导致软件无法正常工作而出现崩溃界面因而严重影响用户体验.这类缺陷通常会由软件会后台自动收集相应的崩溃信息提交给后台服务器.Wu 等人通过挖掘这些崩溃信息提出了 CrashLocator[126]和 ChangeLocator[127]分别在方法级别和变更级别定位这种缺陷.在 2020 年,Guo 等人[128]在 ChangeLocator 的基础上对崩溃变更的特征进行选择,他们发现过滤后的特征能够用来构建更好的崩溃变更定位模型.

对 Android 应用缺陷的定位. Android 应用与开源的桌面软件有所不同,它们运行在手机操作系统之上,这类应用的历史缺陷报告较少,并且没有充足的详细描述信息.因此,很难将现有的 IRBL 方法直接用于对 Android 应用进行缺陷定位. 2019 年,Zhang 等人[100]基于提交信息提出了一种适用于 Android 应用的缺陷定位方法.

5.5 小结

本节从程序频谱、变异分析、多模态和新型应用四个方面简要介绍了缺陷定位领域中其他相关研究.下面简述主要的发现.

(1) 程序频谱是主流的动态缺陷定位方法.其相关研究一直是缺陷定位领域的另一个热点;
(2) 变异分析是一种有效的动态缺陷定位方法.目前多数研究主要致力于降低这类方法的计算成本;
(3) 多模态的缺陷定位方法可以结合多种定位技术的优点,近年来也逐渐受到研究者的重视;
(4) 缺陷定位的应用已经开始逐步细分到研究具体类型的缺陷,这种趋势有利于更精细化地将缺陷定位技术应用到实践中.

6 挑战

从现有文献来看,IRBL 技术的相关研究已经取得很好的成效,最新的定位方法已经能够将定位性能提升到一个较高的层面.但是我们应当注意到,现有的研究技术在工业界中并没有被广泛应用[39],因为它还难以满足实际软件开发和维护过程的需求,本节从局限性、泛化性、准确性和实用性四个角度来介绍 IRBL 领域的研究面临着以下挑战.

(1) **局限性.** 目前对 IRBL 方法的性能评估是基于该方法在一组缺陷报告上测试的平均性能值(例如 MRR).这类评估指标可以在整体上对某个方法进行评价.但是有一点需要注意,在整体上具有较高的评估分数不代表该方法在每个具体缺陷报告的推荐结果也令人满意[73].对任何 IRBL 方法来说,是不可能对所有的缺陷报告都给出最优排序结果的.在应用一种方法时可能会遇到以下情况:对某些(可能仅仅一小部分)缺陷报告进行推荐时,该方法会将这些缺陷报告对应的源文件排在十分靠后的位置.对这样的推荐结果,开发者按照列表顺序从头审查将会花费巨大的工作量.尽管这种推荐结果出现的次数较少,但是由于开发者不知道何时会遇到这种推荐,他们会倾向于不使用该方法.上述情况属于一个 IRBL 方法的应用局限性.能够准确分析每个 IRBL 方法的局限性十分关键,这可以在使用时规避上述情况的发生.现有研究一个问题是局限性分析较少,导致开发者不能够在恰当的场景下使用这些 IRBL 方法从而这些方法的最大优势不能被充分利用,同时在使用时不能及时避免其劣势;
(2) **泛化性.** 影响泛化性的因素有两点: 1)测试数据集稀少.虽然现在已有多个公开的数据集,最近两年提出的一部分 IRBL 方法[41][64][65][79][94][99]仍仅仅在很少的测试项目(Zhou 等人[28]在 2012 年提供的 4 个项目)上进行了验证.特别地,这些项目数据大多是是由 Java 语言开发的,因此多数现有的实验结果是适用于 Java 项目的[35],但是在其他类型的项目上的性能表现未知,即 IRBL 方法的泛化性能未知.由于存在这种不确定性,目前的方法很难受到开发人员的信任,并且他们也很难挑选出优秀的方法.因此,许多工作[25][31][80]指出仍然需要在更多的项目上进行实验来测试 IRBL 方法的性能. 2)人工设计特征.现有的IRBL方法的特征选取和挖掘都是人工根据某一部分的缺陷数据进行设计的,用这种方式构建出的方法可以在可以一些数据上达到较好的效果.然而,遇到具有不同特征的数据,这些方法的定位能力便会降低从而就表现为泛化性低下[68];
(3) **准确性.** 准确的实验数据集是保证定位方法质量的重要依据.现有的数据集虽然都是按照 Dallmeier 等人[115]提供的方法进行收集的,但是不同的研究者在具体操作过程中会有细节上的差别[28][35][39][115]从而导致不同数据集的准确性受到影响.其中一些重要的细节信息需要验证,例如,与缺陷报告进行匹配的源代码的版本,是否为被修复前的版本[47];数据集中明显不包含缺陷的部分(例如,测试文件)应当被排除在推荐列表之外[39]等等.这些不完全统一的处理细节会严重影响方法测试结果的准确性甚至产生互相矛盾的比较结果[66];
(4) **实用性.** 目前的 IRBL 方法仅活跃在实验研究阶段,它们并没有在工程实践中大量应用[39].除了准确性和泛化性较低的因素之外,更重要的原因是这些定位方法(如,BugLocator[28]、BRTracer[30]、Locus[32]等)只能够提供给开发者建议性的缺陷文件列表,无法为开发人员修复缺陷提供更多的指导信息.在

生产实践中,开发人员需要完全依靠自己对代码的理解来手动检查,所以目前缺陷定位技术所提供的帮助很小,我们分析这是导致它们的实用性很难满足开发者的需要的另一个重要原因.

针对缺陷定位技术面临的问题挑战,我们从结果分析、实验验证和修复指导总结了以下几方面可能将会成为进一步深入研究的重要方向.

(1) **结果分析.** 其中一个研究方向是在缺陷定位框架中应当具有侦测机制,对缺陷报告的特征进行检测并进行分类,根据检测结果,缺陷定位框架应当决定是使用他们提出IRBL方法进行定位还是放弃对该报告的定位.这一过程的实现有利于开发者充分发挥 IRBL 方法在其擅长处理的缺陷报告中进行推荐的能力并且避免他们给出低效的推荐结果.这种机制的建立需要对IRBL方法的定位结果进行分析,通过收集较差的推荐列表所对应的的缺陷报告,分析这些缺陷报告的特征,在应用时遇到具备这类特征的缺陷报告就停止使用IRBL方法而是进行人工处理或者借助其他技术来帮助定位.除此之外,需要增多对 IRBL 方法进行工作量感知[36][37]方面的性能评估分析;

(2) **实验验证.** 针对泛化性和准确性的挑战问题,一个研究方向是增加科学准确对比实验研究目前方法的有效性.主要包括两个方面, 1) 收集全面精准的数据集.目前的研究所使用的数据集极其不统一,所以他们得出的结论具有片面性,不能代表其真实的性能.为基于信息检索的缺陷定位研究收集更多的准确数据集,需要涵盖各种语言的书写的项目以及各个领域的项目用来验证方法的有效性和泛化性. 2) 设置科学的实验流程和设置.目前的研究所用的实验设置比较混乱,这也会影响实验结果的准确性.因此需要在科学统一的流程和设置下进行实验.实验设计要符合现实开发的场景以确保其合理性;

(3) **修复指导.** 为了提高定位方法的实用性能,一个重要的研究方向便是对缺陷定位的结果提供修复指导提示信息.对于给定的推荐结果列表,开发人员只能按照顺序从头开始审查,这还是需要大量花费精力来理解缺陷.如果IRBL方法能够指出其中列表中的文件中哪些特征是与缺陷报告描述内容相符,这可以在很大程度上帮助开发人员快速理解缺陷并顺利进行后续的修复工作.

7 结束语

近年来,基于信息检索的缺陷定位技术(IRBL)由于其具有外部依赖少、计算成本低的优势,成为了缺陷定位领域的研究热点.本文围绕 IRBL 方法的模型改良和模型评估两大方面对当前的研究工作进行了梳理和总结.基于当前的研究进展,本文总结了该领域面临的主要挑战并指出了未来可能的研究方向.主要工作总结如下: (1) 针对模型改良,本文从更换检索模型、使用特征分析和进行查询重构三个维度进行归类和阐述; (2) 针对模型评估,本文从模型比较、评价指标和实验数据三个维度总结该领域的评估现状和存在的问题; (3) 本文从局限性、泛化性、准确性和实用性四个角度总结了当前 IRBL 领域研究面临的问题挑战并针对性地指出了未来的研究方向.

致谢 感谢各位审稿专家提出的宝贵意见.

References:

[1] Pressman RS. Software are Engineering: A Practitioner's Approach. 7th ed., New York: McGraw-Hill, 2010. 437-443. [doi: 10.1002/3527600434.eap439]

[2] Wang Q, Wu SJ, Li MS. Software defect prediction. Ruan Jian Xue Bao/Journal of Software, 2008,19(7):1565−1580 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/19/1565.htm [doi: 10.3724/SP.J.1001.2008.01565]

[3] Chen X, Gu Q, Liu WS, Liu SL, Ni C. Survey of static software defect prediction. Ruan Jian Xue Bao/Journal of Software, 2016,27(1):1−25 (in Chinese). http://www.jos.org.cn/1000-9825/4923.htm [doi: 10.13328/j.cnki.jos.004923]

[4] Chen X, Ju XL, Wen WZ, Gu Q. Review of dynamic fault localization approaches based on program spectrum. Ruan Jian Xue Bao/ Journal of Software, 2015,26(2):390−412 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4708.htm [doi: 10.13328/j.cnki.jos.004708]

[5] Zhang W, Li ZQ, Du YH, Yang Y. Fine-grained software bug location approach at method level. Ruan Jian Xue Bao/Journal of Software, 2019,30(2):195−210 (in Chinese). http://www.jos.org.cn/1000-9825/5565.htm [doi: 10.13328/j.cnki.jos.005565]

[6] Xuan JF, Ren ZL, Wang ZY, Xie XY, Jiang H. Progress on approaches to automatic program repair. Ruan Jian Xue Bao/Journal of Software, 2016,27(4):771−784 (in Chinese). http://www.jos.org.cn/1000-9825/4972.htm [doi: 10.13328/j.cnki.jos.004972]

[7] Serrano N, Ciordia I. Bugzilla, Itracker, and other bug trackers. IEEE Software, 2005,22(2):11-13.

[8] Yu K, Lin MX. Advances in automatic fault localization techniques. Chinese Journal of Computers, 2011,34(8):1411−1423 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.01411]

[9] Wang KC, Wang TT, Su XH, Ma PJ. Key scientific issues and state art of automatic software fault localization. Chinese Journal of Computers, 2015,38(11):2262-2278. [doi: 10.11897/SP.J.1016.2015.02262]

[10] Cao HL, Jiang SJ, Ju XL. Survey of software fault localization. Computer Science, 2014,41(2):1-7. [doi: 10.3969/j.issn.1002-137X.2014.02.001]

[11] Ju XL, Jiang SJ, Zhang YM, Dong GW. Advances in fault localization techniques. Journal of Frontiers of Computer Science and Technology, 2012,6(6):481-494. [doi: 10.3778/j.issn.1673-9418.2012.06.001]

[12] Chi Y, Su XH, Wang TT. A survey on mutation-based fault localization. Intelligent Computer and Applications, 2017,7(5):157-162. [doi: 10.3969/j.issn.2095-2163.2017.05.044]

[13] Marcus A, Sergeyev A, Rajlich V, Maletic JI. An information retrieval approach to concept location in source code. In: Proc. of the Working Conf. on Reverse Engineering. 2004. 214-223. [doi: 10.1109/WCRE.2004.10]

[14] Marcus A, Rajlich V, Buchta J, Petrenko M, Sergeyev A. Static techniques for concept location in object-oriented code. In: Proc. of the Int'l Workshop on Program Comprehension. 2005. 33-42. [doi: 10.1109/WPC.2005.33]

[15] Poshyvanyk D, Marcus A. Combining formal concept analysis with information retrieval for concept location in source code. In: Proc. Int'l Conf. on Program Comprehension. 2007. 37-48. [doi: 10.1109/ICPC.2007.13]

[16] Poshyvanyk D, Guéhéneuc YG, Marcus A, Antoniol G, Rajlich V. Combining probabilistic ranking and latent semantic indexing for feature identification. In: Proc. Int'l Conf. on Program Comprehension. 2006. 137-148. [doi: 10.1109/ICPC.2006.17]

[17] Liu DP, Marcus A, Poshyvanyk D, Rajlich V. Feature location via information retrieval based filtering of a single scenario execution trace. In: Proc. Int'l Conf. on Automated Software Engineering. 2007. 234-243. [doi: 10.1145/1321631.1321667]

[18] Poshyvanyk D, Guéhéneuc YG, Marcus A, Antoniol G, Rajlich V. Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval. IEEE Trans. Software Eng., 2007,33(6):420-432. [doi: 10.1109/TSE.2007.1016]

[19] Wang SW, Lo D, Xing ZC, Jiang LX. Concern localization using information retrieval: An empirical study on Linux kernel. In: Proc. of the Working Conf. on Reverse Engineering. 2011. 92-96. [doi: 10.1109/WCRE.2011.72]

[20] Zhang Y, Lo D, Xia X, Scannielo G, Le TDB, Sun JL. Fusing multi-abstraction vector space models for concern localization. Empirical Software Engineering, 2018, 23(4): 2279-2322. [doi: 10.1007/s10664-017-9585-2]

[21] Hill E, Rao S, Kak AC. On the use of stemming for concern location and bug localization in Java. In: Proc. of the Working Conf. on Source Code Analysis and Manipulation. 2012. 184-193. [doi: 10.1109/SCAM.2012.29]

[22] Le TDB, Wang SW, Lo D. Multi-abstraction concern localization. In: Proc. Int'l Conf. on Software Maintenance. 2013. 364-367. [doi: 10.1109/ICSM.2013.48]

[23] Rao S, Kak, AC. Retrieval from software libraries for bug localization: A comparative study of generic and composite text models. In: Proc. of the Working Conf. on Mining Software Repositories. 2011. 43-52. [doi: 10.1145/1985441.1985451]

[24] Saha RK, Lease M, Khurshid S, Perry DE. Improving bug localization using structured information retrieval. In: Proc. Int'l Conf. on Automated Software Engineering. 2013. 345-355. [doi: 10.1109/ASE.2013.6693093]

[25] Moreno L, Treadway JJ, Marcus A, Shen WW. On the use of stack traces to improve text retrieval-based bug localization. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution. 2014. 151-160. [doi: 10.1109/ICSME.2014.37]

[26] Saha RK, Lawall J, Khurshid S, Perry DE. On the effectiveness of information retrieval based bug localization for C programs. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution. 2014. 161-170. [doi: 10.1109/ICSME.2014.38]

[27] Lukins SK, Kraft NA, Etzkorn LH. Source code retrieval for bug localization using latent dirichlet allocation. In: Proc. of the Working Conf. on Reverse Engineering. 2008. 155-164. [doi: 10.1109/WCRE.2008.33]

[28] Zhou J, Zhang HY, Lo D. Where should the bug be fixed? More accurate information retrieval-based bug localization based on bug report. In: Proc. of the Int'l Conf. on Software Engineering. 2012. 14-24. [doi: 10.1109/ICSE.2012.6227210]

[29] Tantithamthavorn C, Ihara A, Matsumoto K. Using co-change histories to improve bug localization performance. In: Proc. of the Int'l Conf. on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing. 2013. 543-548. [doi: 10.1109/SNPD.2013.92]

[30] Wong CP, Xiong YF, Zhang HY, Hao D, Zhang L, Mei H. Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution. 2014. 181-190. [doi: 10.1109/ICSME.2014.40]

[31] Wang SW, Lo D. Version history, similar report, and structure: putting them together for improved bug localization. In: Proc. Int'l Conf. on Program Comprehension. 2014. 53-63. [doi: 10.1145/2597008.2597148]

[32] Wen M, Wu RX, Cheung SC. Locus: Locating bugs from software changes. In: Proc. Int'l Conf. on Automated Software Engineering. 2016. 262-273. [doi: 10.1145/2970276.2970359]

[33] Nichols BD. Augmented bug localization using past bug information. ACM Southeast Regional Conference. 2010. 61. [doi: 10.1145/1900008.1900090]

[34] Lukins SK, Kraft NA, Etzkorn LH. Bug localization using latent dirichlet allocation. Information and Software Technology., 2010,52(9): 972-990. [doi: 10.1016/j.infsof.2010.04.002]

[35] Rahman MM, Roy CK. Improving IR-based bug localization with context-aware query reformulation. In: Proc. of the Int'l Symp. on Foundations of Software Engineering. 2018. 621-632. [doi: 10.1145/3236024.3236065]

[36] Zhao F, Tang YM, Yang YB, Lu HM, Zhou YM, Xu BW. Is learning-to-rank cost-effective in recommending relevant files for bug localization? In: Proc. of the Int'l Conf. on Software Quality, Reliability and Security. 2015. 298-303. [doi: 10.1109/QRS.2015.49]

[37] Zhao F. Localizing relevant source code files for bug report: An effort-aware effectiveness evaluation M.Sc. Thesis. Nanjing, Jiangsu: Nanjing University, 2016.

[38] Garnier M, Garcia A. On the evaluation of structured information retrieval-based bug localization on 20 C# projects. In: Brazilian Symp. on Software Engineering. 2016. 123-132. [doi: 10.1145/2973839.2973853]

[39] Lee J, Kim D, Bissyandé TF, Jung W, Traon YL. Bench4BL: Reproducibility study on the performance of IR-based bug localization. In: Proc. of the Int'l Symp. on Software Testing and Analysis. 2018:61-72. [doi: 10.1145/3213846.3213856]

[40] Chaparro O. Improving bug reporting, duplicate detection, and localization. In: Proc. of the Int'l Conf. on Software Engineering. 2017. 421-424. [doi: 10.1109/ICSE-C.2017.27]

[41] Kim M, Lee E. Are information retrieval-based bug localization techniques trustworthy? In: Proc. of the Int'l Conf. on Software Engineering. 2018. 248-249. [doi: 10.1145/3183440.3194954]

[42] Rahman MM, Roy CK. Improving bug localization with report quality dynamics and query reformulation. In: Proc. of the Int'l Conf. on Software Engineering. 2018. 348-349. [doi: 10.1145/3183440.3195003]

[43] Herzig K, Just S, Zeller A. It's not a bug, it's a feature: How misclassification impacts bug prediction. In: Proc. of the Int'l Conf. on Software Engineering. 2013. 392-401. [doi: 10.1109/ICSE.2013.6606585]

[44] Thung F, Le TDB, Kochhar PS, Lo D. BugLocalizer: Integrated tool support for bug localization. In: Proc. of the Joint Meeting of the European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. 2014. 767-770. [doi: 10.1145/2635868.2661678]

[45] Le TDB, Oentaryo RJ, Lo D. Information retrieval and spectrum based bug localization: better together. In: Proc. of the Joint Meeting of the European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. 2015. 579-590. [doi: 10.1145/2786805.2786880]

[46] Chaparro O, Lu J, Zampetti F, Moreno L. Detecting missing information in bug descriptions. In: Proc. of the Int'l Symp. on Foundations of Software Engineering. 2017. 396-407. [doi: 10.1145/3106237.3106285]

[47] Ye X, Bunescu R, Liu C. Learning to rank revelant files for bug reports using domain knowledge. In: Proc. of the Joint Meeting of the European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. 2014. 689-699. [doi: 10.1145/2635868.2635874]

[48] Kochhar PS, Tian Y, Lo D. Potential biases in bug localization: do they matter? In: Proc. of the Int'l Conf. on Automated Software Engineering. 2014. 803-814. [doi: 10.1145/2642937.2642997]

[49] Nguyen AT, Nguyen TT, SI-Kofahi J, Nguyen HV, Nguyen TN. A topic-based approach for narrowing the search space of buggy files from a bug report. In: Proc. of the Int'l Conf. on Automated Software Engineering. 2011. 263-272. [doi: 10.1109/ASE.2011.6100062]

[50] Wang SW, Lo D, Lawall J. Compositional vector space models for improved bug localization. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution. 2014. 171-180. [doi: 10.1109/ICSME.2014.39]

[51] Chaparro O, Florez JM, Marcus A. Using observed behavior to reformulate queries during text retrieval-based bug Localization. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution. 2017. 376-387. [doi: 10.1109/ICSME.2017.100]

[52] Lawrie DJ, Binkley DW. On the value of bug reports for retrieval-based bug localization. In: Proc. of the Int'l Conf. on Software Maintenance and Evolution. 2018. 524-528. [doi: 10.1109/ICSME.2018.00048]

[53] Mills C, Pantiuchina J, Parra E, Bavota G, Haiduc S. Are bug reports enough for text retrieval-based bug localization? In: Proc. of the Int'l Conf. on Software Maintenance and Evolution. 2018. 381-392. [doi: 10.1109/ICSME.2018.00046]

[54] Dao T, Zhang LM, Meng N. How does execution information help with information-retrieval based bug localization? In: Proc. Int'l Conf. on Program Comprehension. 2017. 241-250. [doi: 10.1109/ICPC.2017.29]

[55] Lam AN, Nguyen AT, Nguyen HA, Nguyen TN. Bug localization with combination of deep learning and information retrieval. In: Proc. Int'l Conf. on Program Comprehension. 2017. 218-229. [doi: 10.1109/ICPC.2017.24]

[56] Takahashi A, Sae-Lim N, Hayashi S, Saeki M. A preliminary study on using code smells to improve bug localization. In: Proc. Int'l Conf. on Program Comprehension. 2018. 324-327. [doi: 10.1145/3196321.3196361]

[57] Beard MD. Extending bug localization using information retrieval and code clone location techniques. In: Proc. of the Working Conf. on Reverse Engineering. 2011. 425-428. [doi: 10.1109/WCRE.2011.61]

[58] Beard MD, Kraft NA, Etzkorn LH, Lukins SK. Measuring the accuracy of information retrieval based bug localization techniques. In: Proc. of the Working Conf. on Reverse Engineering. 2011. 124-128. [doi: 10.1109/WCRE.2011.23]

[59] Davies S, Roper M, Wood M. Using bug report similarity to enhance bug localisation. In: Proc. of the Working Conf. on Reverse Engineering. 2012. 125-134. [doi: 10.1109/WCRE.2012.22]

[60] Sisman B, Kak AC. Incorporating version histories in information retrieval based bug localization. In: Proc. of the Working Conf. on Mining Software Repositories. 2012. 50-59. [doi: 10.1109/MSR.2012.6224299]

[61] Sisman B, Kak AC. Assisting code search with automatic query reformulation for bug localization. In: Proc. of the Working Conf. on Mining Software Repositories. 2013. 309-318. [doi: 10.1109/MSR.2013.6624044]

[62] Wang SH, Khomh F, Zou Y. Improving bug localization using correlations in crash reports. In: Proc. of the Working Conf. on Mining Software Repositories. 2013. 247-256. [doi: 10.1109/MSR.2013.6624036]

[63] Rath M, Lo D, Mäder P. Analyzing requirements and traceability information to improve bug localization. In: Proc. of the Working Conf. on Mining Software Repositories. 2018. 442-453. [doi: 10.1145/3196398.3196415]

[64] Xiao Y, Keung J. Improving bug localization with character-level convolutional neural network and recurrent neural network. In: Asia-Pacific Software Engineering Conference. 2018. 703-704. [doi: 10.1109/APSEC.2018.00097]

[65] Youm KC, Ahn J, Kim J, Lee E. Bug localization based on code changes histories and bug reports. Asia-Pacific Software Engineering Conference. 2015. 190-197. [doi: 10.1109/APSEC.2015.23]

[66] Thomas SW, Nagappan M, Blostein D, Hassan AE. The impact of classifier configuration and classifier combination on bug localization. IEEE Trans. Software Eng., 2013,39(10):1427-1443. [doi: 10.1109/TSE.2013.27]

[67] Hoang T, Oentaryo RJ, Le TDB, Lo D. Network-clustered multi-modal bug localization. IEEE Trans. Software Eng., 2019,45(10):1002-1023. [doi: 10.1109/TSE.2018.2810892]

[68] Koyuncu A, Bissyande TF, Kim DS, Liu K, Klein J, Monperrus M, Traon YL. D&C: A divide-and-conquer approach to IR-based bug localization. IEEE Trans. on Software Eng., 2019. [doi: http://arxiv.org/abs/1902.02703]

[69] Huo X, Thung F, Li M, Lo D, Shi ST. Deep transfer bug localization. IEEE Trans. Software Eng., 2019:1-12. [doi: 10.1109/TSE.2019.2920771]

[70] Zimmermann T, Premraj R, Bettenburg N, Just S, Schröter A, Weiss C. What makes a good bug report? IEEE Trans. Software Eng., 2010,36(5):618-643. [doi: 10.1109/TSE.2010.63]

[71] Kim DS, Tao YD, Kim SH, Zeller A. Where should we fix this bug? A two-phase recommendation model. IEEE Trans. Software Eng., 2013,39(11):1597-1610. [doi: 10.1109/TSE.2013.24]

[72] Le TDB, Thung F, Lo D. Will this localization tool be effective for this bug? Mitigating the impact of unreliability of information retrieval based bug localization tools. Empirical Software Engineering, 2017,22(4): 2237-2279. [doi: 10.1007/s10664-016-9484-y]

[73] Chaparro O, Florez JM, Marcus A. Using bug descriptions to reformulate queries during text-retrieval-based bug localization. Empirical Software Engineering, 2019,25(4):2947-3007. [doi: 10.1007/s10664-018-9672-z]

[74] Youm KC, Ahn J, Lee E. Improved bug localization based on code change histories and bug reports. Information and Software Technology, 2017,82:177-192. [doi: 10.1016/j.infsof.2016.11.002]

[75] Khatiwada S, Tushev M, Mahmoud A. Just enough semantics: An information theoretic approach for IR-based software bug localization. Information and Software Technology, 2018,93:45-57. [doi: 10.1016/j.infsof.2017.08.012]

[76] Tantithamthavorn C, Abebe SL, Hassan AE, Ihara A, Matsumoto K. The impact of IR-based classifier configuration on the performance and the effort of method-level bug localization. Information and Software Technology, 2018,102:160-174. [doi: 10.1016/j.infsof.2018.06.001]

[77] Xiao Y, Keung J, Bennin KE, Mi Q. Improving bug localization with word embedding and enhanced convolutional neural networks. Information and Software Technology, 2019,105:17-29. [doi: 10.1016/j.infsof.2018.08.002]

[78] Zhang W, Li ZQ, Qing Wang, Juan Li. FineLocator: A novel approach to method-level fine-grained bug localization by query expansion. Information and Software Technology, 2019,110:121-135. [doi: 10.1016/j.infsof.2019.03.001]

[79] Dilshener T, Wermelinger M, Yu YJ. Locating bugs without looking back. Automated Software Engineering, 2018,25(3):383-434. [doi: 10.1007/s10515-017-0226-1]

[80] Wang SW, Lo D. AmaLgam+: Composing rich information sources for accurate bug localization. J. Softw. Evol. Process., 2016, 28(10): 921-942. [doi: 10.1002/smr.1801]

[81] Dit B, Revelle M, Gethers M, Poshyvanyk D. Feature location in source code: a taxonomy and survey. J. Softw. Evol. Process., 2013:53-95. [doi: 10.1002/smr.567]

[82] Arcega L, Font J, Haugen Ø, Cetina C. An approach for bug localization in models using two levels: model and metamodel. Software and Systems Modeling, 2019,18(6):3551-3576. [doi: 10.1007/s10270-019-00727-y]

[83] Rath M, Rempel P, Mader P. The IlmSeven dataset. Requirement Engineering. 2017:516-519. [doi: 10.1109/RE.2017.18]

[84] Rath M, Mäder P. Structured information in bug report descriptions - influence on IR-based bug localization and developers. Software Quality Journal, 2019,27(3):1315-1337. [doi: 10.1007/s11219-019-09445-6]

[85] Chen IX, Li CH, Yang CZ. Mining co-location relationships among bug reports to localize fault-prone modules. IEICE Trans. Inf. Syst., 2011,93-D(5):1154-1161. [doi: 10.1587/transinf.E93.D.1154]

[86] Le TDB, Thung F, Lo D. Predicting effectiveness of IR-Based bug localization techniques. In: Proc. of the Int'l Symp. on Software Reliability Engineering. 2014. 335-345. [doi: 10.1109/ISSRE.2014.39]

[87] Rao S, Medeiros H, Kak AC. Comparing incremental latent semantic analysis algorithms for efficient retrieval from software libraries for bug localization. ACM SIGSOFT Software Engineering Notes, 2015,40(1):1-8. [doi: 10.1145/2693208.2693222]

[88] Naish L, Neelofar, Kotagiri Ramamohanarao K. Multiple bug spectral fault localization using genetic programming. In: Proc. of Australasian Software Engineering Conference. 2015. 11-17. [doi: 10.1109/ASWEC.2015.12]

[89] Alduailij M, Al-Duailej M. Performance evaluation of information retrieval models in bug localization on the method level. In: Proc. of the Int'l Conf. on Collaboration Technologies and Systems. 2015. 305-313. [doi: 10.1109/CTS.2015.7210439]

[90] Kilinç D, Yücalar F, Borandag E, Aslan E. Multi-level reranking approach for bug localization. Expert Systems, 2016,33(3): 286-294. [doi: 10.1111/exsy.12150]

[91] Shi ZD, Keung JW, Bennin KE, Limsettho N, Song QB. A strategy to determine when to stop using automatic bug localization. In: Proc. of the Int'l Computer Software and Applications Conference. 2016. 185-190. [doi: 10.1109/COMPSAC.2016.39]

[92] Rahman S, Rahman MM, Sakib K. An improved method level bug localization approach using minimized code space. In: Proc. of the Int'l Conf. on Evaluation of Novel Approaches to Software Engineering. (Selected Papers) 2016. 179-200. [doi: 10.1007/978-3-319-56390-9_9]

[93] Gore A, Choubey SD, Gangrade K. Improved bug localization technique using hybrid information retrieval model. In: Proc. of the Int'l Conf. on Distributed Computing and Internet Technology. 2016. 127-131. [doi: 10.1007/978-3-319-28034-9_16]

[94] Shi ZD, Keung J, Bennin KE, Zhang ZJ. Comparing learning to rank techniques in hybrid bug localization. Appl. Soft Comput, 2019,62:636-648. [doi: 10.1016/j.asoc.2017.10.048]

[95] Loyola P, Gajananan K, Satoh F. Bug localization by learning to rank and represent bug inducing changes. In: Proc. of Int'l Conf. on Information and Knowledge Management. 2018. 657-665. [doi: 10.1145/3269206.3271811]

[96] Xiao Y, Keung J, Mi Q, Bennin KE. Bug localization with semantic and structural features using convolutional neural network and cascade forest. In: Proc. of Int'l Conf. on Evaluation and Assessment in Software Engineering. 2018. 101-111. [doi: 10.1145/3210459.3210469]

[97] Rath M, Mäder P. Influence of structured information in bug report descriptions on IR-Based bug localization. In: Euromicro Conference on Software Engineering and Advanced Applications. 2018. 26-32. [doi: 10.1109/SEAA.2018.00014]

[98] Wang YJ, Y Yuan, Tong HH, Huo X, Li M, Xu F, Lu J. Bug localization via supervised topic modeling. In: Proc. of Int'l Conf. on Data Mining. 2018. 607-616. [doi: 10.1109/ICDM.2018.00076]

[99] Swe KEE, Oo HM. Bug localization approach using source code structure with different structure fields. In: Proc. of Int'l Conf. on Software Engineering Research, Management and Applications. 2018. 159-164. [doi: 10.1109/SERA.2018.8477206]

[100] Zhang T, Hu WJ, Luo XP, Ma XB. A commit messages-based bug localization for android applications. International Journal of Software Engineering and Knowledge Engineering, 2019,29(4):457-487. [doi: 10.1142/S0218194019500207]

[101] Polisetty S, Miranskyy AV, Basar A. On usefulness of the deep-learning-based bug localization models to practitioners. In: Proc. of the Int'l Workshop on Predictor Models in Software Engineering. 2019. 16-25. [doi: 10.1145/3345629.3345632]

[102] Kim M, Lee E. A novel approach to automatic query reformulation for IR-based bug localization. In: Proc. of Sym. on Applied Computing. 2019. 1752-1759. [doi: 10.1145/3297280.3297451]

[103] Ranman S, Ganguly KK, Sakib K. An improved bug localization using structured information retrieval and version history. In: Proc. of Int'l Conf. on Computer and Information Technology. 2015. 190-195. [doi: 10.1109/ICCITechn.2015.7488066]

[104] Shao P, Atkison T, Kraft NA, Smith RK. Combining lexical and structural information for static bug localisation. International Journal of Computer Applications in Technology, 2012, 44(1),61-71. [doi: 10.1504/IJCAT.2012.048208]

[105] Davies S, Roper M. Bug localisation through diverse sources of information. In: Proc. of the Int'l Symp. on Software Reliability Engineering. 2013. 126-131. [doi: 10.1109/ISSREW.2013.6688891]

[106] Salton G, McGill M. Introduction to Modern Information Retrieval. NewYork: McGraw-Hill, 1983. [doi: 10.1080/00048623.2010.10721488]

[107] Deerwester S, Dumais ST, Furnas GW, Landauer TK, Harshman R. Indexing by latent semantic analysis. J. Am. Soc. Inf. Sci., 1990, 41(6):391-407. [doi: 10.1002/(SICI)1097-4571(199009)41:6%3C391::AID-ASI1%3E3.0.CO;2-9]

[108] Blei D, Ng AY, Jordan M. Latent dirichlet allocation. Journal of Machine Learning Research, 2003,3(4/5):993-1022. [doi: 10.1162/jmlr.2003.3.4-5.993]

[109] Manning CD, Raghavan P, Schu¨tze H. Introduction to Information Retrieval. NewYork: Cambridge University Press, 2008. [doi: 10.1002/asi.v61:4]

[110] Gay G, Haiduc S, Marcus A, Menzies T. On the use of relevance feedback in IR-based concept location. In: Proc. Int'l Conf. on Software Maintenance. 2009. 351-360. [doi: 10.1109/ICSM.2009.5306315]

[111] Hofmann T. Probabilistic latent semantic indexing. In: Proc. Int'l Conf. on Research and Development in Information Retrieval, Berkeley, CA, USA, August 1999. 50–57. [doi: 10.1145/312624.312649]

[112] Chen IX, Jaygarl H, Yang CZ, Wu PJ. Information retrieval on bug locations by learning co-located bug report clusters. In: Proc. of annual international ACM SIGIR conference on Research and development in information retrieval. 2008. 801-802. [doi: 10.1145/1390334.1390511]

[113] Strohman T, Metzler D, Turtle H, Croft W B. Indri: A language model-based search engine for complex queries. ICIA. 2005. 2-6.

[114] Bettenburg N, Just S, Schröter A. What makes a good bug report? In: Proc. of the Joint Meeting of the European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. 2008. 308-318. [doi: 10.1145/1453101.1453146]

[115] Dallmeier V, Zimmermann T. Extraction of bug localization benchmarks from history. In: Proc. Int'l Conf. on Automated Software Engineering. 2007. 433-436. [doi: 10.1145/1321631.1321702]

[116] Abreu R, Zoeteweij P, Golsteijn R, Gemund AJC. A practical evaluation of spectrum-based fault localization. J. Syst. Softw., 2009,82(11):1780-1792. [doi: 10.1016/j.jss.2009.06.035]

[117] Lee HJ, Naish L, Ramamohanarao K. Effective software bug localization using spectral frequency weighting function. In: Proc. of the Int'l Computer Software and Applications Conference. 2010. 218-227. [doi: 10.1109/COMPSAC.2010.26]

[118] Ribeiro HL, Araujo RPA, Chaim ML, Souza HA, Kon F. Evaluating data-flow coverage in spectrum-based fault localization. In: Proc. of the Int'l Symp. on Empirical Software Engineering and Measurement. 2019. 1-11. [doi: 10.1109/ESEM.2019.8870182]

[119] Ma CY, Nie CY, Chao WC. A vector table medel-based systematic analysis of spectral fault localization techniques. Software Quality Journal, 2019,27(1):43-78. [doi: 10.1007/s11219-018-9402-1]

[120] Hong S, Lee B, Kwak T, Jeon Y, Ko B, Kim Y, Kim M, . Mutation-based fault localization for real-world multilingual programs. In: Proc. Int'l Conf. on Automated Software Engineering. 2015:464-475. [doi: 10.1109/ASE.2015.14]

[121] Liu Y, Li Z, Wang LX, Hu ZW, Zhao RL. Statement-oriented mutant reduction strategy for mutation based fault localization. In: Proc. of the Int'l Conf. on Software Quality, Reliability and Security. 2017. 126-137. [doi: 10.1109/QRS.2017.23]

[122] Liu Y, Li Z, Zhao RL, Gong P. An optimal mutation execution strategy for cost reduction of mutation-based fault localization. Information Sciences, 2018,422:572-596. [doi: 10.1016/j.ins.2017.09.006]

[123] Oliveira AAL, Camilo-Junior CG, Freitas ENA, Vincenzi AMR. FTMES: A failed-test-oriented Mutant Execution Strategy for Mutation-based Fault Localization. In: Proc. of the Int'l Symp. on Software Reliability Engineering. 2018. 155-165. [doi: 10.1109/ISSRE.2018.00026]

[124] He T, Wang XM, Zhou XC, Li WJ, Zhang ZY, Cheung SC. A software fault localization technique based on program mutations. Chinese Journal of Computers, 2013,36(11):2236−2244 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2013.02236]

[125] Papadakis M, Traon YL. Metallaxis-FL: mutation-based fault localization. Softw. Test., Verif. Reliab. 2015,25(5-7): 605-628. [doi: 10.1002/stvr.1509]

[126] Wu RX, Zhang HY, Cheung SC, Kim SH. CrashLocator: Locating crashing faults based on crash stacks. In: Proc. of the Int'l Symp. on Software Testing and Analysis. 2014. 204-214. [doi: 10.1145/2610384.2610386]

[127] Wu RX, Wen M, Cheung SC, Zhang HY. ChangeLocator: Locate crash-inducing changes based on crash reports. Empirical Software Engineering, 2018,23(5):2866-2900. [doi: 10.1007/s10664-017-9567-4]

[128] Guo ZQ, Li YH, Ma WWY, Zhou YM, Lu HM, Chen L, Xu BW. Boosting crash-inducing change localization with rank-performance-based feature subset selection. Empirical Software Engineering, 2020,25(3):1905-1950. [doi: 10.1007/s10664-020-09802-1]

附中文参考文献:

[2] 王青,伍书剑,李明树.软件缺陷预测技术.软件学,2008,19(7):1565-1580. http://www.jos.org.cn/1000-9825/19/1565.htm [doi: 10.3724/SP.J.1001.2008.01565]

[3] 陈翔,顾庆,刘望舒,刘树龙,倪超.静态软件缺陷预测方法研究.软件学报,2016,27(1):1−25. http://www.jos.org. cn/1000-9825/4923.htm [doi: 10.13328/j.cnki.jos.004923]

[4] 陈翔,鞠小林,文万志,顾庆.基于程序频谱的动态缺陷定位方法研究.软件学报,2015,26(2):390−412. http://www.jos.org.cn/1000-9825/4708.htm [doi: 10.13328/j.cnki.jos.004708]

[5] 张文,李自强,杜宇航,杨叶.方法级别的细粒度软件缺陷定位方法.软件学报,2019,30(2):195−210. http://www. jos.org.cn/1000-9825/5565.htm [doi: 10.13328/j.cnki.jos.005565]

[6] 玄跻峰,任志磊,王子元,谢晓园,江贺.自动程序修复方法研究进展.软件学报,2016,27(4):771−784. http://www.jos.org.cn/1000-9825/4972.htm [doi: 10.13328/j.cnki.jos.004972]

[8] 虞凯,林梦香.自动化软件错误定位技术研究进展.计算机学报,2011,34(08):1411－1422. [doi: 10.3724/SP.J.1016.2011.01411]

[9] 王克朝,王甜甜,苏小红,马培军. 软件错误自动定位关键科学问题及研究进展. 计算机学报, 2015,38(11):2262-2278. [doi: 10.11897/SP.J.1016.2015.02262]

[10] 曹鹤玲,姜淑娟,鞠小林. 软件错误定位研究综述. 计算机科学, 2014,41(2):1-7. [doi: 10.3969/j.issn.1002-137X.2014.02.001]

[11] 鞠小林,姜淑娟,张艳梅,董国伟. 软件故障定位技术进展. 计算机科学与探索, 2012,6(6):481-494. [doi: 10.3778/j.issn.1673-9418.2012.06.001]

[12] 迟洋,苏小红,王甜甜. 基于变异的软件错误定位方法研究综述. 智能计算机与应用, 2017,7(5):157-162. [doi: 10.3969/j.issn.2095-2163.2017.05.044]

[37] 赵斐. Bug 报告的相关源代码文件定位: 一个工作量感知的有效性评价. 硕士学位论文. 江苏南京:南京大学, 2016.

[124] 贺韬,王欣明,周晓聪,李文军,张震宇,张成志.一种基于程序变异的软件错误定位技术.计算机学报,2013,36(11):2236−2244. [doi: 10.3724/SP.J.1016.2013.02236]