软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(6):2288−2311 [doi: 10.13328/j.cnki.jos.006367]

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

重复软件缺陷报告检测方法综述*

郑 炜[1,4,5], 王晓龙[1], 陈 翔[3,6], 夏 鑫[2], 廖慧玲[1], 刘程远[1], 孙瑞阳[1]

[1](西北工业大学 软件学院, 陕西 西安 710072)

[2](Faculty of Information Technology, Monash University, Melbourne, VIC 3800, Australia)

[3](南通大学 信息科学技术学院, 江苏 南通 226019)

[4](空天地海一体化大数据应用技术国家工程实验室(西北工业大学), 陕西 西安 710072)

[5](大数据存储与管理工业和信息化部重点实验室(西北工业大学), 陕西 西安 710072)

[6](信息安全国家重点实验室(中国科学院 信息工程研究所), 北京 100093)

通讯作者: 陈翔, E-mail: xchencs@ntu.edu.cn

摘 要: 软件缺陷在软件的开发和维护过程中是不可避免的, 软件缺陷报告是软件维护过程中重要的缺陷描述文档, 高质量的软件缺陷报告可以有效提高软件缺陷修复的效率. 然而, 由于存在许多开发人员、测试人员和用户与缺陷跟踪系统交互并提交软件缺陷报告, 同一个软件缺陷可能被不同的人员报告, 导致了大量重复的软件缺陷报告. 重复的软件缺陷报告势必加重人工检测重复缺陷报告的工作量, 并造成人力物力的浪费, 降低了软件缺陷修复的效率. 以系统文献调研的方式, 对近年来国内外学者在重复软件缺陷报告检测领域的研究工作进行了系统的分析. 主要从研究方法、数据集的选取、性能评价等方面具体分析总结, 并提出该领域在后续研究中存在的问题、挑战以及建议.

关键词: 缺陷报告; 重复检测; 深度学习; 自然语言处理; 信息检索

中图法分类号: TP311

中文引用格式: 郑炜, 王晓龙, 陈翔, 夏鑫, 廖慧玲, 刘程远, 孙瑞阳. 重复软件缺陷报告检测方法综述. 软件学报, 2022, 33(6): 2288–2311. http://www.jos.org.cn/1000-9825/6367.htm

英文引用格式: Zheng W, Wang XL, Chen X, Xia X, Liao HL, Liu CY, Sun RY. Systematic Literature Review of Duplicated Bug Report Detection Methods. Ruan Jian Xue Bao/Journal of Software, 2022, 33(6): 2288–2311 (in Chinese). http://www.jos.org.cn/1000-9825/6367.htm

Systematic Literature Review of Duplicated Bug Report Detection Methods

ZHENG Wei[1,4,5], WANG Xiao-Long[1], CHEN Xiang[3,6], XIA Xin[2], LIAO Hui-Ling[1], LIU Cheng-Yuan[1], SUN Rui-Yang[1]

[1](School of Software, Northwestern Polytechnical University, Xi'an 710072, China)

[2](Faculty of Information Technology, Monash University, Melbourne, VIC 3800, Australia)

[3](School of Information Science and Technology, Nantong University, Nantong 226019, China)

[4](National Engineering Laboratory for Integrated Aero-space-ground-ocean Big Data Application Technology (Northwestern Polytechnical University), Xi'an 710072, China)

[5](MIIT Key Laboratory of Big Data Storage and Management (Northwestern Polytechnical University), Xi'an 710072, China)

[6](State Key Laboratory of Information Security (Institute of Information Engineering, Chinese Academy of Sciences), Beijing 100093, China)

* 基金项目: 国家自然科学基金(61972317); 教育部重点实验室基金(GDSC202006); 信息安全国家重点实验室开放课题(2020-MS-07); 陕西省 2021 年重点研发项目(2021GY-041)

收稿时间: 2021-01-29; 修改时间: 2021-03-15; 采用时间: 2021-05-07; jos 在线出版时间: 2021-05-20

Abstract: Software bugs are inevitable in the process of software development and maintenance. Software bug reports are an important bug description documents in the software maintenance process. A high-quality software bug report can effectively improve the efficiency of software bug repair. Nevertheless, due to the existence of many developers, testers, and users interact with the bug tracking system and submit bug reports, the same bug may be reported by different parties, resulting in a large number of duplicate software bug reports. Duplicate software bug reports will inevitably increase the workload of manual detection of duplicate bug reports, cause waste of manpower and material resources, and reduce the efficiency of bug repair. This study systematically analyzes the research work of worldwide scholars in the field of duplicated detection of bug reports in recent years by means of literature research. It mainly analyzes and summarizes the research methods, data set selection, performance evaluation, etc, and puts forward the problems and challenges in the follow-up work in this field, and the correspondent's suggestions.

Key words: bug report; duplicate detection; deep learning; natural language processing; information retrieval

随着信息产业的快速迅猛发展、软件产业规模的不断扩大以及软件开发流程的愈加复杂, 软件缺陷无法避免[1]. 根据 IEEE 标准定义, 软件缺陷是指软件产品中存在的、使产品无法满足软件需求及其规格要求、需要修复的瑕疵和问题[2]. 在这样的背景下, 为了保障软件质量和提高开发效率, 就需要流程化和精细化管理软件项目开发的全过程. 为了对软件缺陷修复过程进行管理, 企业一般采用缺陷跟踪系统来存储、记录和管理软件缺陷报告[3]. 为了提升用户的体验, 同时也为了减少软件测试所需的费用, 许多大型软件允许用户直接提交软件缺陷报告. 然而, 这可能导致当出现某一软件缺陷(如功能缺陷、性能缺陷等)时, 会有多个用户提交同一缺陷的软件缺陷报告[4]. 这就导致了缺陷跟踪系统中存在大量重复的软件缺陷报告. 因此, 软件维护人员不得不额外处理这些重复的软件缺陷报告, 并浪费了大量的人力物力.

软件缺陷是计算机软件或程序中存在的某种破坏正常运行能力的问题、错误或者隐藏的功能缺陷, 软件缺陷的存在, 会导致软件产品在某种程度上不能满足用户的需要[5]. 当一个新的软件缺陷报告提交后, 相关的软件维护人员就必须对其进行分类标记, 确定这份缺陷报告描述的是已有的缺陷还是新的缺陷. Alipour 等人[6]指出: 重复软件缺陷报告会引发对应软件缺陷的重复解决, 并造成时间的浪费. 但 Nicolas 等人[7]认为: 在软件的开发和维护过程中, 直接关闭重复软件缺陷报告并丢弃其包含的信息并不合理. 因为他们发现: 一些重复软件缺陷报告提供的额外信息, 有助于更好地进行缺陷的定位和修复. 除此之外, 重复软件缺陷报告还有助于支持缺陷修复人员推荐等任务. Budhiraja 等人[8]指出: 检测重复软件缺陷报告是一项重要的任务, 其可以有效地避免将同一软件缺陷分配给不同的开发人员. 然而, 人工进行标记软件缺陷报告是否重复可能存在一定的误差, 当软件维护人员未检测出重复的软件缺陷报告时, 就会把软件缺陷重复分派给相应的开发人员. 对于大型的开源软件(例如 Firefox、Eclipse、Mozilla 等), 随着软件规模的不断扩大和版本的升级以及用户数量的不断提升, 缺陷跟踪系统中的软件缺陷报告的数量也在不断增加. 根据 Fan 等人[9]的统计, 他们发现: 部署在 Bugzilla 上的 Mozilla 项目有一段时间平均每天收到的软件缺陷报告有 300 个之多, 对于人工处理标记其中是否有重复的软件缺陷报告, 将消耗大量的人力物力. 2015 年, 根据 Zhang 等人[10]的统计, 他们发现: Eclipse 的缺陷跟踪系统中存在 45 688 个被标记为重复(duplicate)的软件缺陷报告. 目前, 我们根据 Eclipse (https://bugs.eclipse.org/bugs/)的缺陷跟踪系统统计到 57 245 个被标记为重复的软件缺陷报告. 在 5 年多的时间里, Eclipse 的缺陷跟踪系统中提交了 11 557 个重复软件缺陷报告, 平均每年增长 2 311 个重复软件缺陷报告. 显然, 重复软件缺陷报告仅仅依靠人工标记变得越来越困难. 因此, 对于可自动检测重复软件缺陷报告方法的研究很有必要.

过去已有大量的研究工作聚集在重复软件缺陷报告的自动检测方法上, 为了更好地理解重复检测方法面临的主要问题以及目前的研究进展, 对这一领域的研究进行系统的分析、总结和比较是十分必要的. 本文的参考文献尽力覆盖近 20 年来与重复软件缺陷报告检测方法相关的研究工作, 同时包含与重复软件缺陷报告检测方法研究的其他相关的主要文献. 通过系统性地研究相关的文献资料来尝试回答以下问题: (1) 调研文献主要实现了哪些研究方法以检测重复的软件缺陷报告? (2) 调研文献的研究方法的分类方式有哪些? (3) 不同的研究方法采用的评价指标有哪些? (4) 在已有的研究方法中, 数据集的来源以及数据规模如何? (5) 不同研究方法之间的效果和性能如何?

基于以上提出的问题，首先搜索了近 20 年来重复软件缺陷报告检测相关的研究工作，通过人工审查的方式，筛选出近 20 年来的 36 篇高质量论文作为研究对象；然后，以重复软件缺陷报告检测为研究主题，从研究方法、数据集和性能评价等方面提取研究文献数据进行总结分析．本文通过以上过程调研了重复软件缺陷报告检测领域的研究现状，提出了一些重要的发现，最后总结了该领域面临的问题与挑战．

图 1 展示了本文的综述过程．

- 首先，确定综述研究的搜索关键词以及文献检索来源，进行初步的文献检索工作；
- 接着，对检索得到的文献进行人工审查和评估，剔除与重复软件缺陷报告检测研究无关的文献；
- 随后，对筛选出来的文献进行归纳总结和分类，从重复软件缺陷报告检测研究的新技术、新理论和实证研究两大类展开深入的讨论．对于提出新技术、新理论的文献，本文根据文献采用的具体技术进行分类总结，总结出以下 3 种主要技术方法：自然语言处理(natural language processing, NLP)、机器学习(machine learning, ML)、深度学习(deep learning, DL)．对文献采用的数据集来源进行分析、比较和总结．此外，总结了此类文献常用的性能评价指标，从性能评价指标角度比较不同文献方法的有效性．

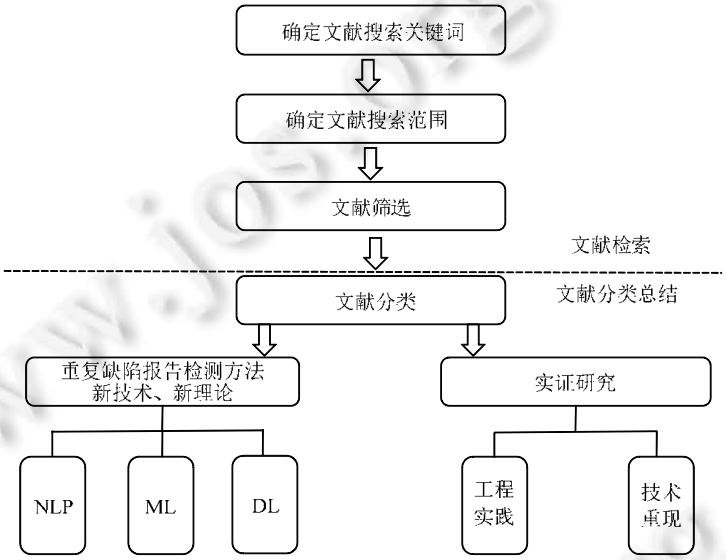

图 1　综述框架

本文采用以下流程来完成相关文献的检索和筛选．

- 首先定义文献的选取标准：该文献针对重复软件缺陷报告检测提出了新理论、新技术，或者该文献对重复软件缺陷报告检测方法的相关理论提供了实证研究支持，或者对其应用进行了实证研究讨论；
- 此外，检索到的文献必须是公开发表的期刊、会议、技术报告或书籍．

依据以上标准，本文通过以下 4 个步骤对文献进行检索和筛选．

(1) 检索来源：本文文献检索主要考虑了 DBLP Computer Science Bibliography、ACM Digital Library、IEEE Xplore Digital Library 论文搜索引擎和论文数据库；

(2) 检索关键字：文献检索的关键字包括 Duplicate Bug Report、Bug Report、Error Report、Similar Bug Report、Bug Report Duplicate Detection、Bug Report Analysis、Duplicate Detection、Software Bug Report、Duplicate Software Bug Report；

(3) 文献审查：通过人工审查的方式移除掉与综述主题无关的文献，并通过查阅相关论文的参考文献和相关研究人员发表的论文列表，以进一步识别出遗漏的文献；

(4) 文献评估：对于筛选出来的文献，我们遵循 Kitchenham 和 Charters[11]建议的 SLR 方法．采用设定问

题的方法和策略进行文献的质量评估, 以提高统计文献的可信度和严谨性. 对于每一个评审问题, 我们设置 3 个档位的分值进行具体的评估: 0 分值代表文献中没有对此问题的描述; 1 分值代表此文献除了对此问题有部分描述以外, 还有对其他研究方向的涉猎; 2 分值代表有明确的描述.

设置问题如下.

- QA1: 研究问题——重复软件缺陷报告的检测. 从文献标题、摘要、关键字以及简介中获取研究方向的信息, 是否有明确研究为重复软件缺陷报告的检测. 当文献对 QA1 问题描述为 0 分值时, 则该文献的总分值为 0, 不再进行其他问题的验证; 1 分值代表此文献除了对此问题有部分描述以外, 还对其他研究方向有所涉猎; 2 分值代表有明确的描述;
- QA2: 对于提出新技术、新理论的文献, 文中应当具有清晰的重复软件缺陷报告检测方法的具体描述. 对于实证研究类的文献, 应当明确指明重现的重复软件缺陷报告的检测方法是什么. 当 QA2 问题描述为 0 分值时, 表示在提出新技术、新理论的文献中没有针对重复软件缺陷报告检测的具体方法描述, 或实证研究文献中没有指明重现的重复软件缺陷报告检测方法; 1 分值表示在提出新方法的文献中方法描述不足; 2 分值表示在提出新方法的文献中有清晰的方法描述和逻辑, 或实证研究文献中明确指明了重现的重复软件缺陷报告检测方法;
- QA3: 是否有详细的实验设置和结果?
- QA4: 是否包含对实验结果的性能分析评价?

对于文献的评估工作, 我们邀请了华为西安研究所的 3 位从事相关领域的高级软件工程师来完成. 当出现分值评估不一样的情况时, 按照少数服从多数的原则进行最终的分值确定, 按照上述策略对检索得到的论文进行质量评估排序: 高(分值=8)、中(4≤分值≤7)、低(0≤分值≤3).

通过以上步骤进行文献的检索与筛选, 我们累计得到具有中高评分的 36 篇文献, 其中, 提出新方法的文献 34 篇, 实证研究类文献 2 篇, 并将这些文献纳入到综述的后续文献分析中, 见表 1.

表 1 最终的文献列表及对应质量评分

研究编号	对应参考文献编号	质量评分	研究编号	对应参考文献编号	质量评分
S1	[10]	7	S19	[6]	8
S2	[11]	8	S20	[27]	8
S3	[12]	8	S21	[28]	8
S4	[13]	8	S22	[29]	8
S5	[8]	4	S23	[30]	5
S6	[15]	8	S24	[31]	8
S7	[15]	4	S25	[32]	8
S8	[16]	8	S26	[33]	8
S9	[17]	8	S27	[34]	8
S10	[18]	6	S28	[35]	8
S11	[19]	8	S29	[36]	8
S12	[20]	6	S30	[36]	8
S13	[21]	8	S31	[37]	8
S14	[22]	8	S32	[38]	8
S15	[23]	4	S33	[39]	8
S16	[24]	6	S34	[40]	8
S17	[25]	8	S35	[41]	6
S18	[26]	6	S36	[42]	8

我们对调研文献的发表源进行了分析, 最终结果见表 2. 由表 2 我们知道: 大多数的文献来源于会议文献(共计 27 篇), 期刊文献较少(共计 9 篇). 其中包含 ICSE 论文 8 篇、IST 论文 4 篇、ASE 论文 3 篇、ISSRE 论文 3 篇、MSR 论文 3 篇、APSEC 论文 3 篇、SANER 论文 2 篇、QRS 论文 2 篇、ICPC 论文 1 篇、ICSME 论文 1 篇、TSE 论文 1 篇、JSS 论文 1 篇、IEEE ACCESS 论文 1 篇、ASWEC 论文 1 篇、IS 论文 1 篇、CSMR 论文 1 篇.

表 2 论文发表源统计

类别	CCF 级别	会议/期刊源	文献编号	数量
会议文献	A	ICSE	{S1,S3,S4,S5,S6,S27,S28,S31}	8
		ASE	{S7,S8,S9}	3
	B	SANER	{S10,S14}	2
		ICPC	{S11}	1
		ISSRE	{S22,S26,S29}	3
		ICSME	{S24}	1
	C	QRS	{S15,S20}	2
		MSR	{S16,S17,S18}	3
		APSEC	{S19,S21,S23}	3
	–	ASWEC	{S34}	1
		IS	{S35}	1
		CSMR	{S33}	1
期刊文献	A	TSE	{S32}	1
	B	IST	{S12,S13,S25,S30}	4
		JSS	{S2}	1
	–	IEEE ACCESS	{S36}	1

为了保证文献的权威性，筛选所得的文献中绝大部分(累计 32 篇)发表在 CCF 推荐的期刊和会议上，也有一些(累计 4 篇)发表在非 CCF 推荐的期刊和会议上. 表 2 展示了调研文献所属会议/期刊的分布情况，其中，CCF-A 级别文献 12 篇，CCF-B 级别文献 12 篇，CCF-C 级别文献 8 篇，非 CCF 推荐的文献 4 篇.

我们对每年发表的相关论文的数量进行统计，结果如图 2 所示. 其中，2010 年–2014 年、2016 年–2020 年文献发表数均在 3 篇及以上. 近 3 年来，平均每年发表数为 5 篇. 由数据分析可见：近年来，越来越多的研究人员聚焦在重复软件缺陷报告检测方法的研究上.

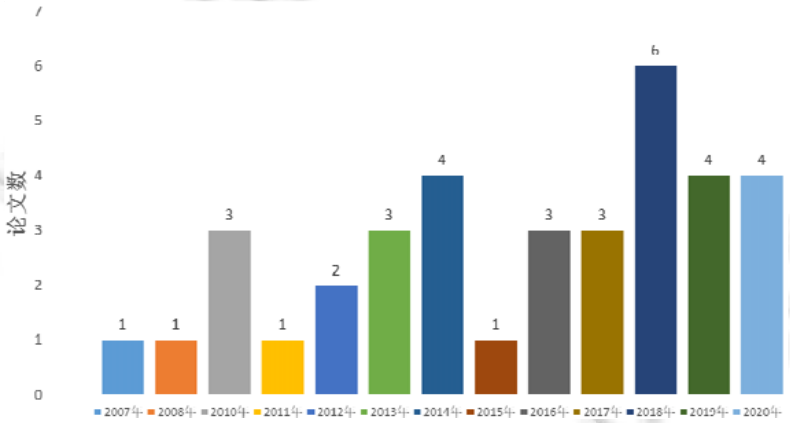

图 2 研究文献按年份的分布情况

本文第 1 节介绍重复软件缺陷报告的例子，然后总结使用自然语言处理方法检测重复软件缺陷报告的通用步骤. 第 2 节研究总结重复软件缺陷报告检测方法的分类方法以及调研文献中使用到的技术方法. 第 3 节讨论文献中使用到的数据集来源、规模和特征字段的选取. 第 4 节总结文献采用的性能评价指标及其有效性评价. 第 5 节对系统文献调研的有效性影响因素进行分析. 第 6 节对已有研究进行总结，提出该领域面临的问题和我们的建议. 第 7 节总结全文.

1 重复软件缺陷报告检测概况

本节介绍关于重复软件缺陷报告检测技术方法的预备知识. 其中，第 1.1 节介绍重复软件缺陷报告的例子，以辅助理解重复软件缺陷报告检测问题. 第 1.2 节介绍重复软件缺陷报告检测方法的通用步骤.

1.1 重复软件缺陷报告例子

表 3 给出了托管在 Bugzilla 上的 Eclipse 项目的一对重复软件缺陷报告(其缺陷报告编号分别是 481543 和 481645). 在收到编号为 481645 的软件缺陷报告时, 如果直接将其分派给缺陷修复的开发人员, 将会造成人力物力上的浪费. 此时应该进行的是: 从缺陷跟踪系统中检测此软件缺陷报告是否为重复软件缺陷报告, 如果为是, 则不需要分派给开发人员. 在节约人力物力的同时, 也提高了软件测试和维护的效率.

表 3 重复缺陷报告的示例

Tag 名称	内容	
Bug ID	481543	481545
Product	CFT	CFT
Component	General	General
Summary	Refactor client requests so that different versions can be supported for different server definition	Refactor client requests so that different versions can be supported for different server definition
Description	PWS/Diego requires changes to the underlying client to address issues like 503 on fetching application stats, as well as different mechanism to fetch files. However, this may not be compatible or applicable to other CF server definitions like BlueMix. On possibility is to define a request factory that is associated with a server definition that gets used by the server behaviour, and allows server definitions to use a default version or specify their own via	PWS/Diego requires changes to the underlying client to address issues like 503 on fetching application stats, as well as different mechanism to fetch files. However, this may not be compatible or applicable to other CF, server definitions like BlueMix. On possibility is to define a request factory that is associated with a server definition that gets used by the server behaviour, and allows server definitions to use a default version or specify their own via

1.2 基于自然语言处理检测重复软件缺陷报告的方法的通用步骤

图 3 展示了基于自然语言处理来检测重复软件缺陷报告的过程. 其输入包含新提交的软件缺陷报告, 以及基于缺陷跟踪系统提取的软件缺陷报告数据集. 基于自然语言处理的重复软件缺陷报告检测方法主要有数据预处理、文本表示、分类器的选择这 3 个阶段. 数据预处理是对软件缺陷报告的数据集进行语料库的构建, 通过一系列的文本预处理操作, 得到标准化的文本数据. 为了高效地检测重复软件缺陷报告, 就必须找到一种较好的文本表示方法, 这种文本表示不仅要能够真实地反映软件缺陷报告文档的内容(摘要、描述等), 还要有对于不同文档内容的区分能力. 分类器用来输出判别软件缺陷报告是否重复的概率, 因此针对新的软件缺陷报告, 可以得到缺陷跟踪系统与这一新的软件缺陷报告相似的软件缺陷报告的有序列表. 然后, 检测重复软件缺陷报告的工作人员根据重复软件缺陷报告的推荐列表来判别新的缺陷报告是否为重复软件缺陷报告.

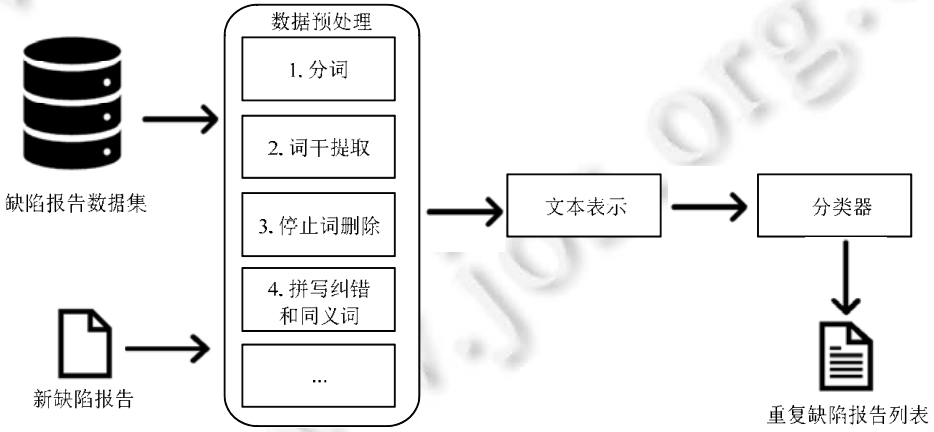

图 3 基于 NLP 的重复软件缺陷报告检测流程

2 研究方法

这一节调研不同文献采用的重复软件缺陷报告检测方法, 对其研究方法进行分类, 以研究不同的重复软

件缺陷报告检测方法的研究情况.

2.1 重复软件缺陷报告检测方法分类研究

为了更好地对文献进行系统化的分析和结构化研究, 我们在本小节进行重复软件缺陷报告检测方法分类方式的研究.

2.1.1 Lin 分类策略

2016 年, Lin 等人[13]将重复软件缺陷报告检测方法归类为以下 3 种策略(我们简称其为 Lin 策略).

(1) 基于排名的方法. 主要思想是: 将新的软件缺陷报告作为查询条件, 从现有的软件缺陷报告中构造可能重复的软件缺陷报告的排序列表;

(2) 基于二进制的方法. 主要思想是: 将软件缺陷报告作为输入, 输出为此软件缺陷报告的标记(重复或者不重复);

(3) 基于决策的方法. 输入为一对软件缺陷报告, 输出为确定它们是否是彼此的重复软件缺陷报告.

根据 Lin 策略, 我们将筛选出的文献进行了分类, 具体分类见表 4. 其中, 基于排名的方法的文献数量为 22 篇, 基于决策的方法的文献数量为 11 篇, 基于二进制的方法的文献数量为 3 篇. 由此可以得出, 大多数的研究方法偏好于基于排名的方法.

表 4 基于 Lin 策略对已有文献的分类结果

分类策略	研究编号	数量
排名方法	{S1,S2,S4,S5,S6,S7,S8,S10,S13,S15,S20,S21, S22,S23,S26,S27,S28,S29,S30,S31,S32,S33}	22
二进制的方法	{S12,S14,S24}	3
决策的方法	{S3,S9,S11,S16,S17,S18,S19,S25,S34,S35,S36}	11

2.1.2 Behzad 分类策略

2020 年, Behzad 等人[22]将重复软件缺陷报告检测方法归类为特征提取的策略(我们简称其为 Behzad 策略). Behzad 策略的主要思想是: 根据一对软件缺陷报告的特征进行分析比较. 特征是指软件缺陷报告的数据字段, 特征可以分为 5 类.

(1) 文本: 文本是利用自然语言处理和信息检索技术提取软件缺陷报告文本字段相似性的一种特征;

(2) 时态: 时态是一种特征, 它显示两个软件缺陷报告之间的间隔时间;

(3) 结构: 结构是一种基于运行时信息和软件缺陷报告中的堆栈跟踪计算的特征类型;

(4) 分类: 分类是一种使用等式比较字段分析两个软件缺陷报告相关性的特征类型;

(5) 主题和上下文: 主题和上下文是一种用于将软件缺陷报告的文本字段与包含独立内容的单词列表进行比较的特征.

根据 Behzad 策略, 我们将筛选出的文献进行了分类, 具体分类见表 5.

表 5 基于 Behzad 策略对已有文献的分类结果

分类策略					研究编号	数量
文本	时态	结构	分类	主题和上下文		
*	–	–	*	*	{S1,S3,S14,S17}	4
*	–	–	–	–	{S2,S4,S6,S7,S21,S22,S23,S24,S28,S30,S32}	11
*	–	*	–	–	{S5,S27}	2
*	–	–	–	*	{S8,S18,S19}	3
*	–	–	*	–	{S9,S15,S34,S35,S36}	5
*	*	*	–	–	{S10}	1
–	–	–	*	–	{S11,S26}	2
–	*	–	*	*	{S12}	1
–	–	*	–	–	{S13,S31,S33}	3
*	*	–	*	–	{S16,S25}	2
*	–	*	*	–	{S20}	1
–	–	–	–	*	{S29}	1

其中, 使用到基于文本特征的策略方法共计有 29 篇文献, 基于分类特征的策略方法共计 15 篇, 基于主题和上下文特征的策略方法共计 9 篇, 基于结构特征的策略方法共计 7 篇, 基于时态特征的策略方法共计 4 篇.

2.2 重复软件缺陷报告检测方法研究

在研究总结前人分类策略的基础上, 本文提出了一种基于技术研究的分类策略, 具体思想是: 根据研究文献中使用到的具体的技术手段进行分类. 对于研究的文献的分类可以分为 3 类.

(1) 基于自然语言技术(natural language processing, NLP)的方法;

(2) 基于机器学习(machine learning, ML)的方法;

(3) 基于深度学习(deep learning, DL)的方法.

从表 6 中我们可以知道: NLP 技术中, 基于 VSM 模型、BM25F 模型、TF-IDF 算法、词嵌入模型的重复软件缺陷报告检测方法使用次数高于其他研究方法的使用; ML 技术中, 基于 LDA 主题模型(一种非监督学习的机器学习方法)是使用频次最多的方法; DL 技术中, 基于 CNN、DNN 的方法是使用频次最多的, 但调研的文献也只有 4 篇使用到了 CNN 的方法, 使用到 DNN 方法的文献仅有 2 篇.

表 6 文献使用的研究方法及对应文献

技术方法所属分类	技术方法	文献编号	数量
NLP	词(文档)嵌入模型	{S3,S4,S19,S22,S23,S24,S29,S30}	8
	余弦相似性	{S20,S25,S30,S26,S34}	5
	TF-IDF 算法	{S2,S10,S22,S23,S30,S15,S33}	7
	VSM 模型	{S2,S5,S7,S15,S27,S28}	6
	BM25F 模型	{S2,S8,S9,S14,S17,S32}	6
	隐马尔可夫模型	{S13}	1
	N-gram 模型	{S21}	1
	word2vec	{S2}	1
	其他	{S31}	1
ML	主题模型 LDA	{S3,S8,S18,S35}	4
	支持向量机(support vector machine, SVM)	{S2,S6,S16}	3
	贝叶斯算法	{S35}	1
	聚类算法	{S34}	1
	其他	{S1}	1
DL	深度神经网络(deep neural network, DNN)	{S4,S29}	2
	卷积神经网络(convolutional neural network, CNN)	{S11,S24,S19,S36}	4
	长短期记忆网络(long short-term memory, LSTM)	{S24}	1

2.2.1 基于 NLP 技术的相关工作

重复软件缺陷报告的检测一直是软件维护中的关键问题之一, 其主要目的是为避免将同一缺陷再分配给不同的开发人员.

(1) 词嵌入模型的应用

Budhiraja 等人[8]在研究 Alipour[6]以及 Nguyen[18]提出的基于主题模型 LDA 检测重复软件缺陷报告的方法时发现, 主题模型 LDA 具有较好的召回率; 在研究 Yang[31]以及 Mikolov[43]提出的基于词嵌入模型的重复软件缺陷报告检测方法时发现其具有较好的精度. 为了同时获得主题模型 LDA 与词嵌入模型的优势, Budhiraja 等人[8]提出了一种基于主题模型 LDA 与词嵌入模型相结合的重复软件缺陷报告的检测方法(LWE).

Budhiraja 等人[14]提出了一种基于词嵌入模型与深度神经网络相结合的重复软件缺陷报告检测方法(DWEN), 通过训练的词嵌入模型, 将软件缺陷报告转换成一个向量, 再经过随机梯度下降的深度神经网络方法训练这些向量, 以此来检测一对软件缺陷报告是否为重复软件缺陷报告对.

词嵌入模型注重单词间的关系, 考虑单词出现的上下文; TF-IDF 方法则更注重的是不同文档在整个语料库中的关系. Yang 等人[31]认为, 这两种方法是相辅相成的. 基于此, 提出了一种基于词嵌入模型与 TF-IDF 方法相结合的方法, 通过计算软件缺陷报告的摘要、描述、组件以及产品的相似性分数, 生成重复软件缺陷报告列表. Hu 等人[32]在 Yang 等人[31]提出方法的基础上进行了优化, 增加了计算软件缺陷报告的摘要、描述作

为文档嵌入向量的相似性分数. 结果显示, 其召回率可以有效提高 7.89%–8.96%.

Wang 等人[39]提出了一种将屏幕截图信息与文本信息相结合的重复软件缺陷报告检测方法, 提取屏幕截图的图像结构特征和图像颜色特征, 文本的处理通过单词嵌入和 TF-IDF 方法进行处理. 以此计算屏幕截图相似性和文本相似性, 相似度分数主要依靠文本相似性分数判决软件缺陷报告是否为重复软件缺陷报告.

(2) VSM 模型、TF-IDF 算法、余弦相似度的应用

Sabor 等人[29]提出: 首先将函数调用的堆栈跟踪信息抽象为包名序列的特征提取方法, 将每个函数替换为包名序列, 以此减小特征向量的维度; 然后使用堆栈跟踪相似性和软件缺陷报告的组件、严重程度字段的线性组合的余弦相似度来度量新的软件缺陷报告和软件缺陷报告库中的软件缺陷报告的相似性, 根据软件缺陷报告对的余弦相似度的大小排列生成一组重复的余弦相似度缺陷报告的推荐列表. Banerjee 等人[34,35]提出了一种基于余弦相似度、时间窗口和文档因素相结合的方法, 时间窗口和文档因素被用来缩小原本巨大的搜索空间. Gopalan 等人[44]则提出了一种基于 INCLUS 聚类算法[45]与余弦相似度相结合的重复软件缺陷报告检测方法.

Chaparro 等人[20]基于 Lucene 技术方法提出了新的查询策略. Lucene 技术是将 TF-IDF 方法和信息检索中的布尔模型结合起来, 计算新的软件缺陷报告与现有报告之间的相似性的方法. Lerch 等人[42]仅基于软件缺陷报告中的堆栈跟踪信息来计算软件缺陷报告之间的相似性.

Runeson 等人[37]提出了基于矢量空间模型结合余弦相似度算法检测重复软件缺陷报告的方法. Huang 等人[25]提出了基于矢量空间模型结合 TF-IDF 算法的重复软件缺陷报告检测方法. Wang 等人[36]在 Runeson 提出的方法之上增加了执行信息的相似度检测, 其中, 执行信息使用矢量空间模型进行向量化表示, 并规定, 文本信息的相似度分数权重大于执行信息的相似度分数权重. Song 等人[15]基于矢量空间模型开发了 JDF 工具(一组集成在 Jazz 中的 Eclipse 插件), 使用自然语言和执行信息查找给定软件缺陷报告的潜在重复项.

设 VSM 中的两个向量: $D_1=D_1(w_{11},w_{12},\ldots,w_{1n})$, $D_2=D_2(w_{21},w_{22},\ldots,w_{2n})$表示软件缺陷报告 D_1 和 D_2, 图 4 中, $D_1=D_1(w_{11},w_{12},\ldots,w_{1n})$称为软件缺陷报告 D_1 的向量空间模型或向量表示.

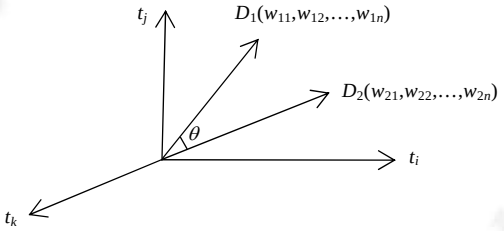

图 4 向量空间模型示意图

任意两个软件缺陷报告 D_1 和 D_2 之间的相似度系数 $Simcos(D_1,D_2)$代表了两个缺陷报告文本内容的相关程度, 则缺陷报告 D_1 和 D_2 的余弦相似度的计算公式如下:

$$Simcos(D_1,D_2)=\frac{\sum_{k=1}^{n} w_1k\times w_2k}{\sqrt{\sum_{k=1}^{n} w_1k\times w_1k\sum_{k=1}^{n} w_2k\times w_2k}} \tag{1}$$

Thung 等人[17]基于 Runeson 提出的方法实现了开源工具 DupFinder (https://github.com/smagsmu/dupfinder), 可以集成在缺陷跟踪系统(Bugzilla)中使用. DupFinder 从缺陷跟踪系统中出现的新的缺陷报告和历史的软件缺陷报告摘要和描述字段中提取文本, 使用向量空间模型来度量软件缺陷报告的相似性, 并根据这些报告与新的软件缺陷报告的余弦相似性, 向系统维护人员提供潜在的重复软件缺陷报告列表.

(3) BM25F 模型

BM25F 是结构化文档检索有效的文本相似性函数, 是专门为短文本查询设计的. 给定 N 个文档的语料库

D, 每个文档 d 由 K 字段组成, f 字段中的术语可以用 $d[f]$表示, $1\leqslant f\leqslant K$. Nd 是包含 t 项的文档数, wf 是字段 f 的权重, $occurrences(d[f],t)$是字段 f 中项 t 的出现次数, $lengthf$ 是 $d[f]$的大小, $average\_lengthf$ 是 D 中所有文档的 $d[f]$的平均大小, bf 是根据字段长度进行变化的参数, $0\leqslant bf\leqslant 1$.其计算公式如下:

$$BM25F(d,q)=\sum_{t\in d\cap q}\log\frac{N}{Nd}\times\frac{\dfrac{\sum_{f=1}^{k}wf\times occurrences(d[f],t)}{\dfrac{bf\times lengthf}{average\_lengthf}+1-bf}}{k_1+\dfrac{\sum_{f=1}^{k}wf\times occurrences(d[f],t)}{\dfrac{bf\times lengthf}{average\_lengthf}+1-bf}} \tag{2}$$

Sun 等人[19]对 BM25F 模型进行了优化扩展, 使其可以进行较长文本的查询. Alipour 等人[6]基于 Sun 等人[19]的研究方法进行了扩展, 考虑了上下文的语义处理. 相比较重复软件缺陷报告的相似度测量的精度提高了 4.52%左右. Rakha 等人[41]进行实证研究时发现, 以前使用 BM25F 模型进行重复软件缺陷报告检测工作的性能被高估了. Aggarwal 等人[24]基于 BM25F 模型提出了一种构建上下文特征的方法检测重复软件缺陷报告.

(4) 其他

Ebrahimi 等人[23]在进行软件缺陷报告重复检测研究工作时发现, 只有少数的研究方法考虑了执行信息(堆栈跟踪), 他们提出了一种利用堆栈跟踪和隐马尔可夫模型(HMM)自动检测重复软件缺陷报告的新方法. Sureka 等人[30]第一次提出了一种使用字符级 N-gram 模型进行文本相似性匹配的识别重复软件缺陷报告的方法. Dang 等人[40]提出了一种基于堆栈跟踪的新的相似性度量值, 称为位置依赖模型(PDM). PDM 根据两个调用堆栈上的函数信息、这些函数到顶部架构的距离以及匹配函数之间的偏移距离计算两个调用堆栈之间的相似性.

2.2.2 基于 ML 技术的相关工作

Nguyen 等人[18]提出了一种将主题模型 LDA 和 BM25F 模型相结合的 DBTM 综合模型, 它从语义上捕捉缺陷报告中的文本信息, 结合基于主题的特征结构制定软件缺陷报告之间的语义相似性度量. Klein 等人[27]在 Alipour 等人[6]研究的基础之上, 通过引入一系列基于软件缺陷报告主题分布的度量(度量两个主题文档分布之间的第 1 个共享主题)来扩展主题模型 LDA. Chen 等人[46]将 LDA 主题模型与贝叶斯算法相结合, 来检测重复软件缺陷报告.

Su 等人[12]基于 Adaptive Bug Search (ABS)(Oracle 开发的一种服务, 使用机器学习来查找给定输入缺陷的潜在重复缺陷), 利用产品和组件的关系实现重复软件缺陷报告的检测工作. Lin 等人[13]基于 Sun 等人[16]使用支持向量机提出的判别模型(SVM-54, 综合了缺陷报告 54 个文本结构的相关特征, 对不同的软件缺陷报告字段进行组合检测)的基础之上, 提出了一种增强 SVM-SBCTC 模型, 通过考虑 3 种不同类型的文本相关关系——基于聚类的相关性(使用 TF-IDF 算法)、基于 BM25F 的文档相关性和基于 Word2vec 的语义相关性, 进行重复软件缺陷报告的检测工作. Lazar 等人[26]提出了一种基于新的文本相似性特征和二进制分类的自动重复软件缺陷报告检测的改进方法, 将软件缺陷报告的文本字段串联在一起作为分类的基础, 而不是仅仅使用软件缺陷报告的标题和描述文本, 采用支持向量机方法对软件缺陷报告进行标记分类为重复或者非重复.

2.2.3 基于 DL 技术的相关工作

Akilan 等人[47]提出了一种基于卷积神经网络的重复软件缺陷报告检测方法, 他们设计的神经网络共包括 3 层, 分别是提取层(获取每个单词间的语义和语法关系)、相似性计算层(映射缺陷报告句子的相似性)、连接层(计算缺陷报告间的相似性). He 等人[21]提出了一种通过构建双通道的卷积神经网络(DC-CNN)的重复软件缺陷报告检测方法, 在该方法中, 每个软件缺陷报告都通过使用词嵌入模型转换为二维矩阵. Xie 等人[28]提出了一种基于卷积神经网络与词嵌入模型结合的重复软件缺陷报告检测方法, 该方法采用词嵌入作为输入层, 为卷积层提供特征向量; 然后, 使用卷积层和池化层构建任意两个软件缺陷报告的分布式向量表示; 之后,

可以得到输入的两个软件缺陷报告的相似度分数. Deshmukh 等人[33]提出了一种利用暹罗卷积神经网络和长短期记忆网络模型建立分类网络模型准确检测重复软件缺陷报告的方法, 该方法首先使用单词嵌入模型将自然语言软件缺陷报告文本的描述转换成数字表示, 然后用长短期记忆网络模型和卷积神经网络模型对这些数值表示进行编码, 然后计算这两个编码的相似性, 以获得两个软件缺陷报告的相似性.

Xiao 等人[38]提出了一种使用异构信息网络准确地检测语义相似度的重复软件缺陷报告检测的方法, 该方法将软件缺陷报告的语义关系嵌入到一个低维空间中, 由两个向量表示软件缺陷报告, 两个向量在空间中的距离越近, 表示两个软件缺陷报告的相似度越高.

2.3 小 结

本节首先识别调研文献中的研究方法并对其归类, 总结了基于 NLP 技术、ML 技术、DL 技术在重复软件缺陷报告检测方法应用的现状. 我们对 3 种技术在重复软件缺陷报告检测的表现进行总结分析. 基于 NLP 技术进行重复软件缺陷报告检测工作时需要依赖大量的数据集进行语言模型的训练, 语言模型的选择在很大程度上决定了方法的有效性. 基于 ML 技术需要依赖大量的训练数据, 需要人工进行标注训练集, 降低了训练效率; 再者, 基于 ML 技术在处理更高阶、更抽象的自然语言时只能够学习预先制定的规则, 而对于规则之外的语言特征无法进行有效的表示. 基于 DL 技术, 如卷积神经网络、循环神经网络等, 通过对生成的词向量进行学习, 自动地学习更高阶的自然语言. 总体来说, 基于 DL 技术与其他两种技术相比, 在重复软件缺陷报告检测上具有更好的性能.

3 数据源分析

本节主要讨论调研文献的研究对象, 包括数据集来源、数据集规模以及数据提取分析. 其中, 数据集来源主要为开源数据集. 在此, 我们提出一个判断数据集规模的准则, 即根据软件缺陷报告的数量级将数据集规模分为以下 3 类.

(1) 小型数据集(S): 软件缺陷报告数量<10K;

(2) 中型数据集(M): 10K≤软件缺陷报告数量<100K;

(3) 大型数据集(L): 软件缺陷报告数量>100K.

3.1 数据集来源及规模

在案例研究/实验中, 数据集是评估方法有效性的重要因素. 本文调研了已有研究的数据集使用情况, 包括数据集来源和数据集规模.

已有研究在重复软件缺陷报告检测方法的研究中, 使用的数据集来源通常为大型开源项目公开的数据集. 表 7 汇总了调研文献使用的数据集, 包括数据集来源和选取的数据集时间周期. 在 36 篇调研文献中: 文献 S5 和 S7 使用的数据集不明确; 文献 S1、S21、S23、S28、S31 和 S36 的数据集时间周期不明确, 这里不作讨论. 从表 7 可以看出: 已有研究使用的数据集仅限于在选定时间段中的软件缺陷报告数据, 在该时间段外的软件缺陷报告将被忽略. 而部分文献(如 S16、S8 等)选取的数据集时间差仅仅只有 1 年, 在这种情况下, 存在于该时间段以外的重复软件缺陷报告将检测不到, 可能会造成检测性能被高估. 文献 S4、S11、S24 使用的数据集由 Lazar 等人[48]生成, 该数据集包含自项目启动至 2013 年底以来提交的所有软件缺陷报告, 并提供所有重复软件缺陷对的信息. 他们选取的数据集时间周期长, 包含所有数据, 对性能评估的准确性影响较小[48].

我们对调研文献中使用的来自开源项目的数据集进行统计分析, 图 5 展示了在 36 篇调研文献使用的(不包括文献 S5 和 S7)开源数据集的主要来源及其使用频率. 由图 5 可知: 来自 Eclipse 项目的数据集是调研文献中研究人员使用最多的开源数据集, 其次是 OpenOffice、Mozilla、Firefox 等开源项目. 除了开源项目中的数据集以外, 已有研究也使用特定环境下的数据集. 例如: 文献 S15 在验证他们所提出的检测重复软件缺陷报告的方法过程中采用了软件测试竞赛的数据集, 包括 9 个考试题和 7 422 个缺陷报告; 文献 S31 使用微软公司的 5 种产品中的数据进行实验.

表 7 研究文献使用的数据集情况汇总

文献编号	数据集(格式: 来源名称 时间周期, 如 Eclipse 2001.10-2014.12)
S14	(1) Mozilla 2010, (2) Eclipse 2008, (3) OpenOffice 2008-2010, (4) Android 2007.11-2012.09
S16,S8,S32	(1) Mozilla 2010, (2) Eclipse 2008, (3) OpenOffice 2008-2010
S9	(1) Mozilla 2010, (2) Eclipse 2008, (3) OpenOffice 2008-2010, (4) Eclipse 2001-2007
S6	(1) Firefox 至 2007.06, (2) Eclipse 2008, (3) OpenOffice 2008
S2	(1) Apache 2009-2013, (2) ArgoUML 2000-2013, (3) SVN 2001-2013
S3	(1) Mozilla 2006-2013
S4	(1) Mozilla 1998.04-2014.12, (2) OpenOffice 2000.10-2014.12
S11,S24	(1) OpenOffice 2000.10-2014.12, (2) Eclipse 2001.10-2014.12, (3) NetBeans 1998.08-2014.12
S10	(1) Firefox 2010, (2) OpenOffice 2008-2010, (3) Eclipse 2001-2007,...(20)
S12	(1) Android 2008-2017, (2) Mozilla 2010-2013, (3) Eclipse 2008-2013, (4) OpenOffice 2008-2013
S13	(1) Firefox 至 2016 年, (2) GNOME 至 2016 年
S15	(1) 软件测试竞赛 2016-2018
S17,S18	(1) Android 2007.11-2012.09
S19	(1) Hadoop 2005.07-2018.12, (2) Hdfs 2006.04-2018.01, (3) Map Reduce 2006.03-2018.01, (4) Spark 2010.04-2018.10
S20	(1) Eclipse 2001.10-2015.02
S22	(1) Eclipse 2001.10-2013.02, (2) Mozilla 2001.09-2007.10
S25	(1) OpenOffice 至 2014.03, (2) Eclipse 至 2011.12, (3) Firefox 至 2012.06(每个数据集时间周期>10 年)
S26	(1) Firefox 启动至 2012
S27	(1) Eclipse 2004.06, (2) Firefox 2004.01-2001.04
S29	(1) Eclipse 2001.10-2018.09, (2) GCC 1999.08-2018.09, (3) LLVM 2003.10-2018.09, (4) Freedesktop 2003.01-2018.09, (5) GNOME 1999.02-2018.09, (6) KDE 1999.01-2018.09, (7) LibreOffice 2010.08-2018.09, (8) Linux kernel 2002.11-2018.09, (9) OpenOffice 2000.10-2018.09
S30	(1) Baidu CrowdTest crowdtesting platform 2017.06.01-2017.06.10
S33	(1) Eclipse 2001.10-2007.12
S34	(1) Eclipse 2008.1-2008.12, (2) Mozilla 2010.1-2010.12, (3) Open office 2010.1-2010.12
S35	(1) Apache 1998.5-2013.10, (2) Mozilla 2001.5-2013.10, (3) Eclipse 2001.7-2013.10

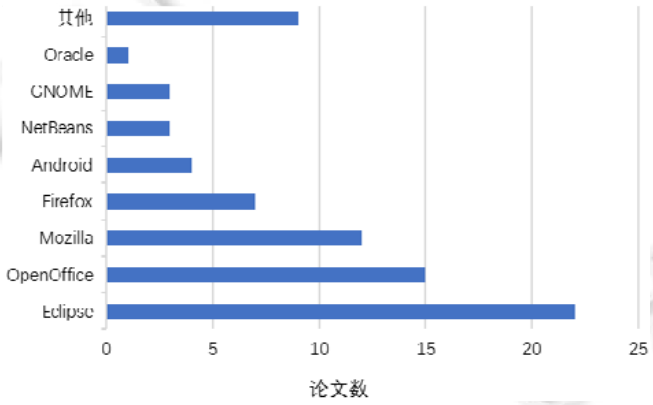

图 5 开源数据集的来源

在调研文献中, 研究人员在使用数据集进行验证实验时可能会使用来自不同开源项目的数据集, 即联合数据集. 我们对调研文献的数据集来源的个数进行统计分析, 其中有 2 篇文献(约占 6%)只提出了研究方法, 没有明确方法是否在数据集上进行验证, 如图 6 所示. 24 篇文献(约占 67%)选择两个及以上的数据集对重复软件缺陷报告检测方法进行实验评估. 其中, 文献 S10 使用来自 20 个开源项目的数据集, 涵盖了不同的软件类型, 从桌面应用程序(如 Eclipse)到框架/库(如 Struts 和 PDFBox), 也涉及各种领域(如浏览器(Firefox)、办公软件(OpenOffice)、分布式计算(Hadoop 和 Spark)等). 在此, 我们分析调研了文献数据集来源的软件项目类型, 软件项目分类如下.

- 文档管理系统: OpenOffice、LibreOffice、SVN;
- 开发工具: Eclipse、NetBeans、Groovy、Maven、ArgoUML、GCC、LLVM;
- 桌面应用程序: GNOME、Freedesktop、KDE;

- 浏览器: Firefox、Mozilla;
- 框架/库: Struts、PDFBox、MapReduce、Cassandra、Hive;
- 系统/组件: Oracle、Hadoop、Spark、Hdfs;
- 服务器: Apache;
- 操作系统: Android、Linux kernl.

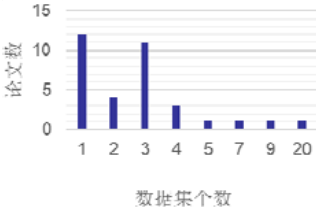

图 6 调研文献使用数据集个数情况

表 8 展示了 36 篇调研文献的数据集项目来源的分布情况, 可以看出: 已有的研究方法多数偏向于应用在文档管理系统、开发工具以及浏览器这些软件项目. 这 3 类软件类型拥有较多的用户数量, 因此提交的软件缺陷报告也更多. 例如, Mozilla 每天收到超过 350 份软件缺陷报告, 而 Eclipse 每天收到超过 80 份软件缺陷报告[34]. 已有的重复软件缺陷报告检测方法多数处于学术研究阶段, 它们并没有在实际项目开发实践中得到应用, 这是该领域研究人员值得关注的问题. 根据已有研究使用的数据集来看, 33%的研究仅在单个数据集上进行了验证. 因此, 从已有研究的应用场景调研情况来看, 该领域的已有研究成果与投入到实际项目开发实践还有一定的距离; 同时, 实证研究结论的一般性需要基于更多数据集来进行验证.

表 8 调研文献数据集软件项目类型分布情况

软件项目类型	文献编号
文档管理系统	{S2,S4,S6,S8,S9,S10,S11,S12,S14,S16,S24,S25,S29,S34,S36}
开发工具	{S6,S8,S9,S10,S11,S12,S14,S16,S20,S21,S22,S23,S24,S25,S27,S29,S33,S34,S35,S36}
桌面应用程序	{S13,S29}
浏览器	{S3,S4,S6,S8,S9,S10,S12,S13,S14,S16,S22,S25,S26,S27,S34,S35,S36}
框架/库	{S10,S19}
系统/组件	{S1,S10,S19}
服务器	{S2,S35,S36}
操作系统	{S12,S14,S17,S18,S29}

根据前面定义的数据集规模, 我们对调研文献使用的数据集规模进行统计分析, 结果如图 7 所示.

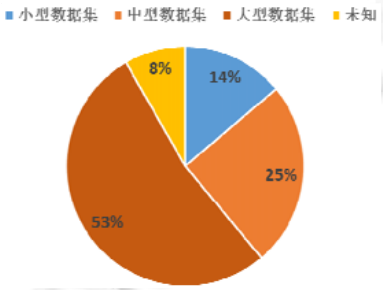

图 7 调研文献的数据集规模分布情况

调研文献中约 53%的研究使用大型数据集, 约 25%为中型数据集, 约 14%为小型数据集规模, 文献 S1、S15、S28 以及 S31 使用的数据集来自特定软件项目. 例如: 文献 S15 使用的是软件测试竞赛的数据集, 该研究与人工评审方法进行对比, 从时间和评估率两个方面证实该研究方法的优越性. 因此, 这些数据集具有特定的软件项目场景要求.

本节分析了调研文献中研究所使用的数据集来源和规模,下面简述主要发现.

(1) 调研文献中, 重复软件缺陷报告检测方法主要使用主流的缺陷跟踪系统中公开的数据进行实验评估. 在选取数据集时, 存在选取数据集时间周期过短而导致选取的软件缺陷报告不全面, 造成遗漏重复软件缺陷报告的问题;

(2) 数据集的大小会影响重复软件缺陷报告检测方法的准确性, 随着缺陷跟踪系统中软件缺陷报告的数量越来越多, 重复软件缺陷报告检测方法的精度会降低. 因此, 为了研究方法性能评估的可信赖性, 研究人员倾向于使用主流的大型开源项目的数据集, 或是多个数据集联合使用. 当然, 不同类型的研究方法选择的数据集规模也会存在不同的偏好, 如果研究方法在多种场景都适用, 那么选择多个开源项目的数据集进行评估更有利于性能评价;

(3) 调研文献中, 对于数据集选取都经过细致的比较与斟酌, 由此可见数据集对于研究方法性能评估的重要性. 仅使用单个数据集进行方法验证, 不能表明该方法在其他数据集上的性能, 无法证实其泛化能力.

3.2 数据提取分析

数据提取是指在重复软件缺陷报告检测方法的研究案例中, 研究者基于软件缺陷报告提取其中的描述文本和元数据字段的文本信息. 本节调研文献中研究采用的软件缺陷报告的字段, 我们称其为特征字段, 然后对其进行分类总结.

3.2.1 软件缺陷报告

软件缺陷报告是描述软件潜在问题的文档, 其主要目的是为开发人员提供缺陷的详细信息, 以便开发人员可以快速地完成对软件缺陷的严重程度的预测、分配对应的缺陷修复人员、定位和修复软件缺陷等软件活动, 又称为软件 Bug 报告、软件错误报告、软件故障报告等. 本文中使用的名称是软件缺陷报告. 一个典型的软件缺陷报告如图 8 所示.

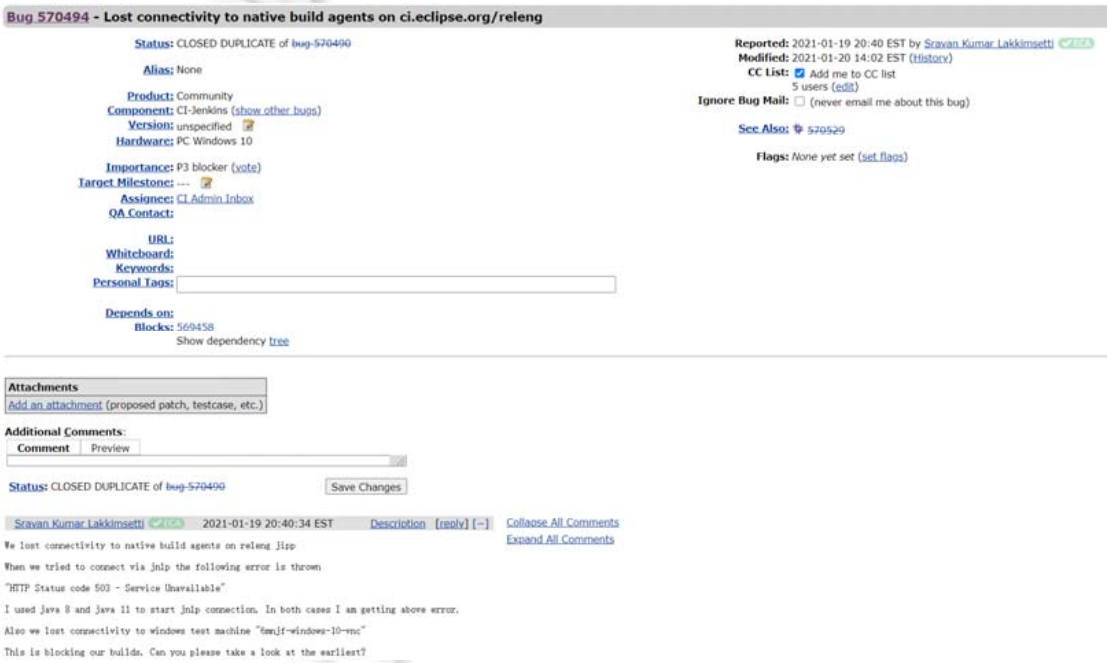

图 8 Eclipse 中 570494 软件缺陷报告示例

软件缺陷报告包含的内容可以分为描述文本和元数据两类, 其中, 描述文本包括标题、描述和评论. 其他的特征字段属于元数据. 如表 9 为软件缺陷报告主要的特征字段说明.

表 9 Eclipse 中软件缺陷报告的主要字段展示

特征	描述
Bug ID(软件缺陷编号)	软件缺陷报告的唯一表示, Bug 570494 表示 Eclipse 编号为 570494 的唯一软件缺陷报告
Summary(标题)	软件缺陷的简要描述
Description(描述)	软件缺陷的详细描述, 缺陷产生的详细信息
Status(状态)	软件缺陷目前所属的处理状态, 如: 打开、已修复、关闭等
Product(产品)	软件缺陷所属的产品
Component(模块)	软件缺陷所属的模块
Importance(重要性)	软件缺陷的重要程度, 包括优先级和严重程度两个方面
Hardware(硬件)	软件缺陷发生的硬件环境
Assignee(分配人员)	软件缺陷分配的维护人员
URL(链接地址)	软件缺陷相关的链接地址
Keywords(关键字)	软件缺陷的主要特点
Duplicates(重复缺陷报告)	重复软件缺陷报告的标识
Depends on(依赖)	软件缺陷依赖于其他缺陷的解决
Blocks(阻塞)	软件缺陷影响到的其他缺陷
Reported(报告)	软件缺陷的上报时间和上报人员
Attachments(附件)	软件缺陷相关的其他信息(如缺陷故障的截图、视频、执行信息等)
Comment(评论)	其他人员对于软件缺陷的看法或建议, 或是对于描述信息的补充

3.2.2 研究方法中采用的字段调研

本节调研各文献研究方法使用缺陷报告中哪些关键特征字段进行重复软件缺陷报告的检测工作, 探讨特征字段的选取对于重复软件缺陷报告的检测工作性能的影响.

文献 S1 方法中使用产品和组件的关系作为限制条件对潜在重复软件缺陷报告进行检测时获得了较好的性能, 但在增加版本作为限制条件的时候, 检测结果会错过部分重复软件缺陷报告. 文献 S3、S4、S6、S7、S8、S10、S21、S25、S35、S36 只考虑使用软件缺陷报告中的描述文本标题和描述部分. 文献 S5 使用软件缺陷报告的特征字段标题、描述以及执行信息作为检测重复软件缺陷报告的检测内容. 文献 S9、S17、S34 检测方法不仅检测摘要和描述字段中文本内容的相似性, 还包括产品、组件、版本、优先级等特征字段的相似性检测. 文献 S11、S22、S26 将产品和组件这两种软件缺陷报告的元数据与描述文本的标题和描述一起进行处理, 仍然以标题和描述为重复分类标准的主要依据, 可以提高重复软件缺陷报告分类的准确性. 文献 S23 同样是选取描述文本标题和描述以及元数据产品和组件 4 个特征字段作为重复检测标准的依据, 然而, S23 是将产品和组件作为一组特征计算该组的相似性分数, 我们认为, 产品和组件的组合特征能够很好地反映软件缺陷报告之间的结构关系.

文献 S14 使用缺陷报告中产品、组件、优先级和版本字段, 在提取新的软件缺陷报告的描述文本/元数据字段特征时, 以软件缺陷报告对的方式比较软件缺陷报告的相似性, 并生成每对检测的软件缺陷报告中每个基于字段的相似性分数. 文献 S15 提取软件缺陷报告中的缺陷编号、描述和标题作为重复检测的特征字段, 缺陷编号对于每个缺陷报告具有唯一的标识作用. 文献 S16 选取缺陷报告编号、标题、描述、产品、组件、类型、优先级、版本、打开日期作为重复软件缺陷报告检测分类特征的基础, 然后将描述文本标题和描述与其他的元数据串联在一起用于生成联合特征计算其相关性. 文献 S18 选取缺陷报告编号、打开日期、状态、标题、描述、组件、优先级和版本特征字段最为文本检测的重点, 与文献 S16 中字段的选取相比, 舍弃了版本字段. 文献 S18 认为, 软件缺陷报告中很少有指定版本字段, 因此在检测的过程中忽略版本字段特征的相似度检测. 文献 S24 在特征字段的选取工作中, 也将缺陷编号作为一个缺陷报告的唯一标识, 选取组件、标题、描述作为相似性检测工作主要的特征依据. 文献 S19 认为: 随着软件系统频繁更新版本, 同一应用程序的多个版本的软件系统可能会共存. 用户使用的版本更新并不是马上进行的, 在其他较低版本中可能会再次遇到较高版本中已解决的问题. 因此, S19 选取元数据组件、版本、创建时间和优先级等和描述文本串联起来, 将串联向量放入输出层进行逻辑回归分类判断是否为重复的软件缺陷报告. 文献 S32 选取缺陷报告编号、描述、标题、组件、优先级、类型、版本和报告日期字段特征作为重复软件缺陷表达检测的检测依据.

文献 S20 选取的特征字段与其他调研文献有较大的差别, 在特征字段的选取上并没有选取描述文本的字

段作为重复检测的依据, 而是选取堆栈跟踪信息与元数据组件和严重程度的线性组合测量一对软件缺陷报告的相似性, 因此方法存在一定的局限性, 对于不包含堆栈跟踪信息的软件缺陷报告将不能进行重复检测工作, 只能用来检测特定的包含堆栈跟踪信息的软件缺陷报告, 对于当前主流的缺陷跟踪系统不具备普遍的适用性. 而文献 S31、S33 仅仅选取了堆栈跟踪信息作为软件缺陷报告重复检测的依据, 与其他文献中特征字段的选取相比更加单一, 也不具备普遍的适用性.

文献 S27 选取堆栈跟踪信息以及描述文本标题和描述作为重复检测工作的主要依据, 并设置了一系列的策略, 在不同的情况下, 只选取堆栈跟踪信息的相似性、描述文本标题和描述字段的相似性或者是两者相似性之间的结合分数来判断两个缺陷报告的相似性. 文献 S28 指出: 53%的重复缺陷报告是在 20 天内提交的, 90%的重复缺陷报告是在 20–60 天之间提交的. 因此, 在重复检测工作的过程中选取软件缺陷报告开始时间字段可以在一定程度上缩小软件缺陷报告的查找空间, 文献 S28 也选取描述文本标题和描述特征字段作为重复检测工作的主要检测依据. 文献 S29 选取产品、组件、版本、严重性、优先级和标题, 选取的特征字段具有以下关系连接: 标题和组件、标题和版本、标题和优先级、标题和严重性以及产品和组件, 因此, 文献 S29 制定了这 5 个组合相似性特征来表示两个软件缺陷报告之间的相似性. 文献 S30 在考虑文本相似性的基础上增加了屏幕截图的相似性计算来表示两个软件缺陷报告之间的相似性.

3.2.3 字段值在软件缺陷报告生命周期内的变化

软件缺陷报告的字段值通常会随着状态的变化或时间的推进而更改[49], 例如: 当测试人员发现一个问题时, 软件缺陷报告的状态为 Unconfirmed(待确认的); 当确定该问题是一个缺陷后, 提交针对该缺陷的软件缺陷报告, 此时, 软件缺陷报告的状态更改为 New(新的); 软件缺陷解决后, 其软件缺陷报告状态更改为 Resolved(已解决的). 在软件缺陷报告的生命周期内, 软件缺陷报告的大部分字段(除了报告者、报告时间、描述字段)都是可更改的[50]. 对于调研文献研究方法中采用的字段, 如摘要、描述、产品以及组件等, 当提交软件缺陷报告的工作人员发现其相关字段值不准确或不完整时, 可以进行补充修改. 因此, 在特定时间检索的软件缺陷报告数据与其他时间检索到的软件缺陷报告数据可能有所不同.

3.2.4 小 结

通过以上调研文献特征字段选取的研究总结, 我们知道, 目前的研究方法主要依靠描述文本的特征字段标题和描述作为重复软件缺陷报告检测工作的检测重点. 通过元数据字段和描述文本字段的结合, 可以实现很好的相似性测量, 在检测工作中应该尽量全面地考虑有用的检测字段, 提高检测的准确性和可靠性. 执行信息(堆栈跟踪信息)也是一个检测重复软件缺陷报告标准很好的衡量指标. 在字段选取和方法策略的执行上, 应尽量考虑方法对于目前缺陷跟踪系统中重复软件缺陷报告检测的普适性和实用性, 从多角度衡量每个特征字段对于重复软件缺陷报告检测工作的作用.

4 性能评价

本节主要研究总结已有的研究文献中的性能评价指标, 对其分类并分析已有研究方法的性能和效果.

4.1 评价指标及分类

我们提取出各文献中的评价指标并对其进行分类, 评价指标是评价文献研究方法的性能和效率的关键特性, 我们据此得出不同的研究方法之间的性能差异.

(1) Recall at Top N

该指标又被称为 Accuracy@k、Hit@k、Prediction Accuracy 和 Top K Rank. $N_{\text{correct}}@k$ 是 top-k 推荐的正确识别的重复缺陷报告的数量, N_{total} 是总的缺陷报告的数量. Recall at Top N 度量值越高, 重复软件缺陷报告检测方法的技术性能越好. 其计算公式如下:

$$Recall@k = \frac{N_{correct}@k}{N_{total}} \tag{3}$$

(2) MAP (mean average precision)

MAP 具有良好的稳定性, 对于衡量评估排序技术性能, 在每个相关的重复软件缺陷报告被检索到之前, 该标量返回的是前 k 个重复软件缺陷报告文档集的精度值的平均值. 其计算公式如下:

$$MAP(L_{test}, L_{pred}) = \frac{1}{L_{test}} \sum_{i=1}^{L_{test}} \frac{1}{index_i} \tag{4}$$

其中, L_{test} 是测试集中的真实重复软件缺陷报告, L_{pred} 是预测重复软件缺陷报告的排序列表, $index_i$ 是第 1 次查询检索真实重复软件缺陷报告集的索引.

(3) F-measure

该指标是重复软件缺陷报告的查准率和查全率的调和平均数, 可以对查准率和查全率这两个指标进行有效的平衡. 其计算公式如下:

$$F\text{-}measure = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{5}$$

Recall 与 Precision 的定义见后文公式(9)、公式(10).

(4) kappa 系数

这是一种衡量分类精度的指标. P_0 即 Accuracy 准确率. P_e: 假设每一类的真实样本个数分别为 a_1, a_2, …, a_c, 而预测出来的每一类的样本个数分别为 b_1, b_2, …, b_c, 总样本个数为 n, 则 P_e 的计算公式如下:

$$P_e = \frac{a_1 \times b_1 + a_2 \times b_2 + ... + a_c \times b_c}{n \times n} \tag{6}$$

则其 kappa 系数计算公式如下:

$$k = \frac{P_0 - P_e}{1 - P_e} \tag{7}$$

(5) Accuracy(准确率)

其计算公式如下:

$$Accuracy = \frac{|TP| + |TN|}{|TP| + |TN| + |FN| + |FP|} \tag{8}$$

混淆矩阵		真实值	
		Positive	Negative
预测值	Positive	TP	FP
	Negative	FN	TN

TP (true positive): 表示的是正确识别出来的重复软件缺陷报告. TN (true negative): 表示正确识别的不是重复的软件缺陷报告. FN (false negative): 表示的是把不是重复的软件缺陷报告错误的识别成重复的软件缺陷报告. FP (false positive): 表示的是把重复的软件缺陷报告错误地识别为非重复的软件缺陷报告.

(6) Precision(精准率)

其计算公式如下:

$$Precision = \frac{|TP|}{|TP| + |FP|} \tag{9}$$

(7) Recall(召回率)

其计算公式如下:

$$Recall = \frac{|TP|}{|TP| + |FN|} \tag{10}$$

(8) MRR (mean reciprocal rank)

该指标返回的是一系列查询的倒数排名的平均, 其计算公式如下:

$$MRR = \frac{1}{Q} \sum_{i=1}^{Q} \frac{1}{rank_i} \tag{11}$$

$rank_i$ 是重复的软件缺陷报告的推荐列表数量.

(9) AUC (area under the curve)

ROC 曲线下方的面积. ROC 曲线的横坐标是 FP, 纵坐标是 TP, AUC 量化了 ROC 曲线表达的分类能力; 分类能力越好, AUC 越大, 输出概率越合理, 排序的结果也越合理.

我们对研究文献的性能评测指标的使用进行统计, 统计结果显示, 采用次数最多的评价指标是 Recall at Top *N*. 最终的结果见表 10, 从表中可以看出: 为了实验数据结果的准确性和可靠性, 调研的文献一般都采用多种评价指标来进行实验结果的评测.

表 10 调研文献评价指标及分类

评价指标	文献	数量
Recall at Top *N*	{S2,S3,S4,S6,S8,S9,S10,S13,S20,S21,S22,S23,S26,S27,S28,S30,S32,S33}	17
MAP	{S9,S10,S13,S22,S23,S30,S32}	7
F-measure	{S11,S12,S19,S29,S31,S34,S36}	7
kappa	{S14,S17,S18}	3
Accuracy	{S11,S12,S14,S15,S17,S18,S19,S29,S35,S36}	10
Precision	{S11,S12,S15,S16,S25,S29,S31,S34,S36}	9
Recall	{S1,S11,S12,S15,S16,S25,S29,S31,S34,S36}	10
MRR	{S10,S20,S22,S23,S30,S33}	6
AUC	{S11,S16,S17,S18,S25}	5
未知	{S5,S7}	2

4.2 调研文献检测的效果评价

本节我们对调研文献的检测方法的检测效果进行系统性的分析与总结. 我们以研究方法的实验结果的性能评价为依据, 分析文献实验方法的有效性. 我们将实验结果的性能评价指标分为 5 个度量值. 具体见表 11.

表 11 评价文献方法有效性的度量

性能评价指标值	性能等级
0.00–0.20	极低的有效性(low)
0.20–0.40	一般的有效性(fair)
0.40–0.60	中等的有效性(moderate)
0.60–0.80	较高的一致性(substantial)
0.80–1.00	高度的一致性(perfect)

我们依据文献中使用了相同的性能评价指标来比较不同文献提出的方法的有效性. 因为不同的方法采用不同的数据集, 以及数据集规模的大小不同都会造成实验结果的不一致, 我们这里统计的是调研文献中评价指标的数值为其最好情况下的平均值, 忽略不同数据集之间的差异. 这里, 我们仅依据 Recall at Top *N*、Accuracy、Recall 这 3 种使用频次最多的评价指标进行方法有效性的分析.

(1) 基于 Recall at Top *N* 调研文献方法性能等级情况

为方便统计分析比较不同方法之间的差异, 我们只统计 *N*=10 的情况, 见表 12. 其中, 文献 S2、S13、S27、S30 的方法在 *N*=10 的情况下, 相比其他方法有较好的性能效果. 然而, 不同 *N* 值的 Recall at Top *N* 指标值存在差异, 调研文献的方法一般情况为: *N* 值逐渐增大, 其 Recall at Top *N* 值也越大, 代表其方法性能越优越; 但是随着 *N* 值的不断增大, Recall at Top *N* 值会逐渐趋于饱和, 超过 *N* 值的阈值时, Recall at Top *N* 甚至会下降. 另外, 在 Recall at Top *N* 趋于饱和时的性能效果显示: 所有方法都可以达到较高的一致性(substantial)的效果等级, 大部分的文献方法可以达到高度的一致性(perfect)的效果等级. 如在 S3 文献中: 当 *N*=20 时, Recall at Top 20=0.558; 当 *N*=100 时, Recall at Top 100=0.723; 当 *N*=600 时, Recall at Top *N* 趋于饱和.

(2) 基于 Accuracy 调研文献方法性能等级情况

基于 Accuracy 评价指标调研文献方法性能等级情况见表 13. S11、S12、S14、S17、S18、S19、S29、S35、S36 文献方法的性能等级均达到了高度的一致性(perfect), 仅 S15 的性能等级为较高的一致性(substantial).

表 12 基于 Recall at Top *N* 评价文献方法性能等级表

Recall at Top *N* 评价指标值	文献编号	性能等级
0.91	S2	高度的一致性(perfect)
0.50	S3	中等的有效性(moderate)
0.70	S4	较高的一致性(substantial)
0.57	S6	中等的有效性(moderate)
0.82	S8	高度的一致性(perfect)
0.63	S9	较高的一致性(substantial)
0.64	S10	较高的一致性(substantial)
0.91	S13	高度的一致性(perfect)
0.85	S20	高度的一致性(perfect)
0.40	S21	中等的有效性(moderate)
0.54	S22	中等的有效性(moderate)
0.28	S23	一般的有效性(fair)
0.85	S26	高度的一致性(perfect)
0.92	S27	高度的一致性(perfect)
0.38	S28	一般的有效性(fair)
0.92	S30	高度的一致性(perfect)
0.62	S32	较高的一致性(substantial)
0.81	S33	高度的一致性(perfect)

表 13 基于 Accuracy 评价文献方法性能等级表

Accuracy 评价指标值	文献编号	性能等级
0.96	S11	高度的一致性(perfect)
0.97	S12	高度的一致性(perfect)
0.93	S14	高度的一致性(perfect)
0.61	S15	较高的一致性(substantial)
0.84	S17	高度的一致性(perfect)
0.95	S18	高度的一致性(perfect)
0.91	S19	高度的一致性(perfect)
0.94	S29	高度的一致性(perfect)
0.89	S35	高度的一致性(perfect)
0.97	S36	高度的一致性(perfect)

(3) 基于 Recall 调研文献方法性能等级情况

基于 Recall 评价指标调研文献方法性能等级情况见表 14. S11、S12、S16、S29、S31、S36 文献方法的性能等级均达到了高度的一致性(perfect). S1、S25、S34 文献方法的性能等级均达到了较高的一致(substantial), 仅 S15 文献方法的性能等级为中等的有效性(moderate).

表 14 基于 Recall 评价文献方法性能等级表

Recall 评价指标值	文献编号	性能等级
0.62	S1	较高的一致性(substantial)
0.94	S11	高度的一致性(perfect)
0.96	S12	高度的一致性(perfect)
0.43	S15	中等的有效性(moderate)
0.99	S16	高度的一致性(perfect)
0.74	S25	较高的一致性(substantial)
0.83	S29	高度的一致性(perfect)
0.96	S31	高度的一致性(perfect)
0.723	S34	较高的一致性(substantial)
0.97	S36	高度的一致性(perfect)

在分析总结文献的不同方法对于重复软件缺陷报告的检测效果的过程中, 我们发现, 基于卷积神经网络的方法相比较传统的 NLP 的方法和传统的主题模型的方法在效果上更加优越. 然而, 我们在文献的调研过程中发现, 基于深度学习方法检测重复软件缺陷报告的研究相对较少. 这可以是我们今后研究工作的一个方向.

5 有效性影响因素分析

本文对近 20 年来重复软件缺陷报告检测方法的研究进行了系统性的调研, 经过分析, 发现调研过程存在两方面的不足: (1) 调研文献不够全面; (2) 数据提取可能不够准确.

首先, 为了保证调研文献的系统性, 我们制定了研究的搜索策略, 包括搜索术语、检索来源、搜索过程等一系列过程. 在搜索的过程中, 我们根据收集的搜索术语, 通过人工搜索的方式对文献进行筛选. 因此, 检索关键词可能存在不能覆盖该领域所有相关文献的情况; 另外, 还存在人工操作的不确定性导致相关文献遗漏的情况.

其次, 我们在提取数据的过程中可能存在一些不准确的情况. 根据研究问题的制定, 我们从调研文献中提取相关的信息. 在该过程中, 我们几个合著作者一起讨论以确保数据的统一. 但一方面, 可能由于部分信息的缺失导致数据的不完整; 另一方面, 可能由于讨论交流不到位导致数据不够全面. 更具体地表现为: 验证方法没有充分地加以介绍、性能评价的对比没有直观的说明、研究方法的优势没有很好地解释等, 这些因素都可能影响数据提取的准确性.

6 后续研究面临的问题与挑战

目前, 针对重复软件缺陷报告检测问题的研究已进行了多年且取得了一定的进展. 但该问题仍然具有一定的研究价值和研究意义: 一方面, 现有研究方法检测重复软件缺陷报告的准确度和可靠性还有待提高, 这也是大多数研究成果未能成功运用到企业软件质量保障流程的原因之一; 另一方面, 随着深度学习的不断发展, 其涌现出的新技术和新思想可以引入到重复缺陷报告检测问题的研究上, 例如, 基于 Transformer 的双向深度语言模型 BERT. 本文通过系统文献调研的方式, 分析了重复软件缺陷报告检测方法的研究进展, 希望为该领域未来的工作提供一些有用的参考. 通过前面的分析与总结, 我们发现了该领域发展过程中面临的一些问题与挑战. 本节从深度学习的研究方法、研究方法的实用性以及数据源的局限性这 3 个角度来介绍重复软件缺陷报告检测领域研究所面临的问题与挑战, 并提出相关建议.

(1) 深度学习的研究方法

近年来, 由于深度学习技术的快速发展以及它在自然语言处理中的应用, 深度学习技术也逐渐应用到重复软件缺陷报告检测的研究中. Deshmukh 等人[33]于 2017 年首次使用深度学习技术进行重复软件缺陷报告检测的研究. Budhiraja 等人[14]于 2018 年将词嵌入模型与深度神经网络技术相结合应用到重复软件缺陷报告检测方法的研究中. Xie 等人[28]提出了使用卷积神经网络以实现自动化重复软件缺陷报告的检测方法. He 等人[21]于 2020 年提出了基于深度学习技术的重复软件缺陷报告检测方法, 通过实验, 其结果表明, 检测效果优于 Deshmukh 等人[33]和 Budhiraja 等人[14]的方法. 最先进的深度学习技术已被证明优于传统方法, 它将成为主流的技术服务于该领域未来的研究工作中. 但是如何利用该技术提高重复软件缺陷报告检测方法的性能, 是该领域面临的问题, 也是挑战.

对于未来深度学习在重复软件缺陷报告检测的研究方法, 建议首先对已有工作进行实证研究, 验证在开源和工业项目中的有效性. 在实际应用中验证方法的价值, 同时也能够发现需要改进的地方, 以提出新的研究方法. 其次, 把深度学习技术与现有其他技术进行融合, 譬如: 深度学习技术优化自然语言处理的文本分类方法结合主题模型进行建模, 也可能是一个值得研究的方向. 另一方面, 深度学习的方法需要大量的训练数据支撑对模型进行训练, 在语料库充足并且融合了多种数据来源时, 其检测效果和普适性要优于其他模型, 其优势是可以更加准确地表示文本语义关系, 对于文本的结构和顺序可以更加准确地表达. 就未来的重复软件缺陷报告检测方法的研究, 能否结合小样本学习的方法通过少量的样本对深度学习相关模型进行训练, 以达到目前深度学习方法应用的检测效果, 是一个值得探讨和研究的话题, 这将可以对于前期数据的准备和模型的训练工作节约大量的时间成本.

对于深度学习模型在深度方面的研究, 在调研文献中涉及使用到的单通道卷积神经网络和双通道卷积神经网络模型展现的结果显示, 层数结构越深、越复杂的模型在文本表示的表达上就会拥有更好的效果. 后续

工作可以尝试对深度学习模型的深度进行多层深度学习模型在重复软件缺陷报告检测问题上的探索, 这一方面仍然有很大的实验空间.

(2) 研究方法的实用性

重复软件缺陷报告检测的工作已研究多年, 已有工作提出了许多技术来解决它. 然而, 很少有研究成果应用到工程实践中. 大部分研究先提出自己的研究方法, 并在数据集上进行性能评估以证明该方法比已有方法性能更好, 但没有落实到实际开发环境中. Sun 等人[19]于 2011 年提出了基于信息检索的重复软件缺陷报告检测方法, 并计划将该方法集成到 Bugzilla 系统中. He 等人[21]于 2020 年提出使用双通道卷积神经网络(DC-CNN)进行重复软件缺陷报告检测的方法, 但只在 3 个开源项目的数据集上进行了验证, 缺少实证研究.

在实际开发环境中, 缺陷跟踪系统中多数数据集来自开源软件项目 Eclipse, 根据前面的数据提取分析也表明, 已有研究使用最多的数据集是 Eclipse. 研究人员选择不同的数据集进行性能评估结果也不同. Ebrahimi 等人[23]发现, 他们的研究方法在 GNOME 中比在 Firefox 中性能更好. 因此, 已有研究方法可能具有实际应用的局限性.

另一方面, 通过比较缺陷跟踪系统中所有现有的软件缺陷报告来确定新的软件缺陷报告是否重复需要花费很长时间. 使用启发式或元启发式的方法缩小搜索空间, 这对于实时应用和连续查询任务的应用尤其重要.

在提出研究方法的同时, 考虑把研究方法应用到实际开发环境中, 譬如把研究方法作为一个工具以便研究人员使用. 其次, 进行实证研究, 关注研究方法最适用的应用场景, 使研究方法发挥更大的价值.

目前, 主流的缺陷追踪系统(例如 Bugzilla)已增加了重复软件缺陷报告检测的功能. 在用户创建新的软件缺陷报告时, 会根据 Summary 字段中的信息实时地显示系统推荐的重复软件缺陷报告列表, 以避免用户提交重复软件缺陷报告. 但该功能与用户预期仍有一定的距离, 例如近 5 年来, Bugzilla 中的 Eclipse 项目平均每年仍然会出现 2 311 个重复软件缺陷报告.

(3) 数据源的局限性

准确的实验数据集是保证重复软件缺陷报告检测方法质量的重要依据. 在第 3.1 节通过分析调研文献的数据集来源与规模, 我们发现: 已有研究使用的数据集仅限于一定时间段内的软件缺陷报告, 在这个时间段外的软件缺陷报告将被忽略. 如果被测试的软件缺陷报告存在该时间段外的重复软件缺陷报告, 则不能被检测到. 此外, 大多数研究方法使用的不同数据集之间的实验彼此独立, 并不能完全认为这些方法是普遍适用的. 研究人员应尽量将不同来源的数据集组合在一起进行实验, 以形成更大的数据集集合, 增强方法对不同数据集之间的普遍适用性. 在进行数据预处理工作时, 寻找合适的停止词列表是其中关键的工作内容之一, 以此确保相似性计算的准确性, 保持软件缺陷报告的语义[25]. 调研文献中使用的数据集的大小不一, 随着缺陷跟踪系统中软件缺陷报告的不断增加, 重复检测的精度会降低[50]. 根据文献 S27 中的实验数据, 作者使用 Firefox 中的 77 份软件缺陷报告进行测试时, 获得了 93%的召回率; 当对 10%的 Firefox 数据使用同样方法测试时, 召回率下降到 53%. 这也要求研究人员在实际的实验中应保证测试数据集拥有足够大的数据量. 应在实验过程中考虑不同数据规模在实验中产生不同召回率的原因, 对实验作进一步的改进, 以确保在大当量的数据测试和实际的应用中, 重复检测精度不会出现大幅度的波动, 确保系统和方法性能的稳定性.

研究人员可以使用不同开源项目的公开数据集, 其次关注数据集的质量, 选取合适的实验数据集. 此外, 选取的数据集的时间周期应当尽可能地长, 尽可能地扩大单个开源项目数据集的范围, 降低数据集对性能评估准确性的影响. 正如 Rakha 等人[41]建议的, 研究人员可以尝试使用不忽略任何数据的评估.

(4) 特征字段的选取

在第 3.2 节的研究总结中, 我们已经知道了特征字段的选取对于重复软件缺陷报告的检测工作存在一定精度的影响. 缺陷跟踪系统中, 软件缺陷报告的质量并不能得到有效的保证, 缺陷跟踪系统中存在大量的字段不完整、关键特征字段缺失的软件缺陷报告. 在第 3.2 节的研究总结中我们已经提到: 在不同时间内, 软件缺陷报告的字段值可能有所变化. 因此, 选取的软件缺陷报告数据集是最初提交的软件缺陷报告还是在其生命周期内字段值进行过更改的软件缺陷报告, 这对于重复软件缺陷报告检测结果也会存在一定的影响. 因此,

在检测方法中设置相关缺失字段的检测是否对检测性能会有一定的影响, 都是值得探讨的问题. 对于字段的选取, 较好的做法是将描述文本和元数据字段结合起来对相似度进行加权, 得到最终的相似性分数. 描述文本标题和描述字段是研究人员普遍都会考虑的特征字段. 对于元数据的选取, 则没有一个统一的标准, 元数据字段对于重复软件缺陷报告检测工作的性能提升有多大作用? 哪些元数据字段的检测可以明显改善重复软件缺陷报告检测方法的精度? 都是未来值得我们进一步研究的课题.

7 总 结

本文以文献调研的形式分析了近 20 年的重复软件缺陷报告检测的研究进展, 从研究方法、数据集和性能评价这 3 个方面提取数据分析重复软件缺陷报告的研究现状. 其次, 从深度学习的研究方法、研究方法的实用性、数据源的局限性以及特征字段的选取这 4 个角度, 总结了当前重复软件缺陷报告检测领域研究的问题和挑战, 并提出建议. 虽然重复软件缺陷报告检测的已有研究工作已经积累了较多的研究成果, 但仍存在一些问题与挑战, 这也成为该领域未来的研究方向.

References:

[1] Moran K, Linares-Vásquez M, Bernal-Cárdenas C, Poshyvanyk D. FUSION: A tool for facilitating and augmenting android bug reporting. In: Proc. of the 38th Int'l Conf. on Software Engineering Companion (ICSE 2016). New York: Association for Computing Machinery, 2016. 609−612.

[2] IEEE Standard Classification for Software Anomalies. IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993), 2010. 1−23.

[3] Serrano N, Ciordia I. Bugzilla, ITracker, and other bug trackers. IEEE Software, 2005, 22(2): 11−13.

[4] Wang S, Chen TH, Hassan AE. How do users revise answers on technical Q&A Websites? A case study on stack overflow. IEEE Trans. on Software Engineering, 2020, 46(9): 1024−1038.

[5] Shah M, Pujara N. A review on software defects prediction methods. CoRR abs/2011.00998, 2020.

[6] Bettenburg N, Premraj R, Zimmermann T, Kim SH. Duplicate bug reports considered harmful … really? In: Proc. of the 2008 IEEE Int'l Conf. on Software Maintenance. Beijing, 2008. 337−345.

[7] Fan Y, Xia X, Lo D, Hassan AE. Chaff from the wheat: Characterizing and determining valid bug reports. IEEE Trans. on Software Engineering, 2020, 46(5):495−525.

[8] Zhang J, Wang X, Hao D, *et al*. A survey on bug-report analysis. Sci. China Inf. Sci., 2015, 58: 1−24.

[9] Kitchenham BA, Charters S. Guidelines for performing systematic literature reviews in software engineering. EBSE Technical Report, EBSE-2007-01. ICSE IEEE Computer Society, 2007. 1051−1052.

[10] Su E, Joshi S. Leveraging product relationships to generate candidate bugs for duplicate bug prediction. In: Proc. of the 40th Int'l Conf. on Software Engineering: Companion Proc. (ICSE 2018). New York: Association for Computing Machinery, 2018. 210−211.

[11] Lin MJ, Yang CZ, Lee CY, Chen CC. Enhancements for duplication detection in bug reports with manifold correlation features. The Journal of Systems & Software, 2016, 121.

[12] Budhiraja A, Reddy R, Shrivastava M. LWE: LDA refined word embeddings for duplicate bug report detection. In: Proc. of the 40th Int'l Conf. on Software Engineering: Companion Proc. (ICSE 2018). New York: Association for Computing Machinery, 2018. 165−166.

[13] Budhiraja A, Dutta K, Reddy R, Shrivastava M. DWEN: Deep word embedding network for duplicate bug report detection in software repositories. In: Proc. of the 40th Int'l Conf. on Software Engineering: Companion Proc. (ICSE 2018). New York: Association for Computing Machinery, 2018. 193−194.

[14] Song YK, Wang XY, Xie T, Zhang L, Mei H. JDF: Detecting duplicate bug reports in Jazz. In: Proc. of the 32nd ACM/IEEE Int'l Conf. on Software Engineering (ICSE 2010). Vol.2. New York: Association for Computing Machinery, 2010. 315−316.

[15] Sun CN, Lo D, Wang XY, Jiang J, Khoo SC. A discriminative model approach for accurate duplicate bug report retrieval. In: Proc. of the 32nd ACM/IEEE Int'l Conf. on Software Engineering (ICSE 2010). Vol.1. New York: Association for Computing Machinery, 2010. 45−54.

[16] Thung F, Kochhar PC, Lo D. DupFinder: Integrated tool support for duplicate bug report detection. In: Proc. of the 29th ACM/IEEE Int'l Conf. on Automated Software Engineering (ASE 2014). New York: Association for Computing Machinery, 2014. 871−874.

[17] Nguyen AT, Nguyen TT, Nguyen TN, Lo D, Sun CN. Duplicate bug report detection with a combination of information retrieval and topic modeling. In: Proc. of the 27th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE 2012). New York: Association for Computing Machinery, 2012. 70−79.

[18] Sun CN, Lo D, Khoo SC, Jiang J. Towards more accurate retrieval of duplicate bug reports. In: Proc. of the 2011 26th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE 2011). Lawrence, 2011. 253−262.

[19] Chaparro O, Florez JM, Singh U, Marcus A. Reformulating queries for duplicate bug report detection. In: Proc. of the 2019 IEEE 26th Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Hangzhou, 2019. 218−229.

[20] He JJ, Xu L, Yan M, Xia X, Lei Y. Duplicate bug report detection using dual-channel convolutional neural networks. In: Proc. of the 28th Int'l Conf. on Program Comprehension (ICPC 2020). New York: Association for Computing Machinery, 2020. 117−127.

[21] Neysiani BS, Babamir SM, Aritsugi M. Efficient feature extraction model for validation performance improvement of duplicate bug report detection in software bug triage systems. Information and Software Technology, 2020, 126.

[22] Ebrahimi N, Trabelsi A, Islam MS, Hamou-Lhadj A, Khanmohammadi K. An HMM-based approach for automatic detection and classification of duplicate bug reports. Information and Software Technology, 2019.

[23] Aggarwal K, Rutgers T, Timbers F, Hindle A, Greiner R, Stroulia E. Detecting duplicate bug reports with software engineering domain knowledge. In: Proc. of the 2015 IEEE 22nd Int'l Conf. on Software Analysis, Evolution, and Reengineering (SANER). Montreal, 2015. 211−220.

[24] Huang S, Chen L, Hui Z, Liu J, Yang S, Chen Q. A method of bug report quality detection based on vector space model. In: Proc. of the 2019 IEEE 19th Int'l Conf. on Software Quality, Reliability and Security Companion (QRS-C). Sofia, 2019. 510−511.

[25] Lazar A, Ritchey S, Sharif B. Improving the accuracy of duplicate bug report detection using textual similarity measures. In: Proc. of the 11th Working Conf. on Mining Software Repositories (MSR 2014). New York: Association for Computing Machinery, 2014. 308−311.

[26] Alipour A, Hindle A, Stroulia E. A contextual approach towards more accurate duplicate bug report detection. In: Proc. of the 10th Working Conf. on Mining Software Repositories (MSR). San Francisco, 2013. 183−192.

[27] Klein N, Corley CS, Kraft NA. New features for duplicate bug detection. In: Proc. of the 11th Working Conf. on Mining Software Repositories (MSR 2014). New York: Association for Computing Machinery, 2014. 324−327.

[28] Xie Q, Wen Z, Zhu J, Gao C, Zheng Z. Detecting duplicate bug reports with convolutional neural networks. In: Proc. of the 2018 25th Asia-Pacific Software Engineering Conf. (APSEC). Nara, 2018. 416−425.

[29] Sabor KK, Hamou-Lhadj A, Larsson A. DURFEX: A feature extraction technique for efficient detection of duplicate bug reports. In: Proc. of the 2017 IEEE Int'l Conf. on Software Quality, Reliability and Security (QRS). Prague, 2017. 240−250.

[30] Sureka A, Jalote P. Detecting duplicate bug report using character *N*-gram-based features. In: Proc. of the 2010 Asia Pacific Software Engineering Conf. Sydney, 2010. 366−374.

[31] Yang X, Lo D, Xia X, Bao L, Sun J. Combining word embedding with information retrieval to recommend similar bug reports. In: Proc. of the 2016 IEEE 27th Int'l Symp. on Software Reliability Engineering (ISSRE). Ottawa, 2016. 127−137.

[32] Hu D, *et al*. Recommending similar bug reports: A novel approach using document embedding model. In: Proc. of the 2018 25th Asia-Pacific Software Engineering Conf. (APSEC). Nara, 2018. 725−726.

[33] Deshmukh J, Annervaz KM, Podder S, Sengupta S, Dubash N. Towards accurate duplicate bug retrieval using deep learning techniques. In: Proc. of the 2017 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Shanghai, 2017. 115−124.

[34] Banerjee S, Syed Z, Helmick J, Culp M, Ryan K, Cukic B. Automated triaging of very large bug repositories. Information and Software Technology, 2017: 89.

[35] Banerjee S, Syed Z, Helmick J, Cukic B. A fusion approach for classifying duplicate problem reports. In: Proc. of the 2013 IEEE 24th Int'l Symp. on Software Reliability Engineering (ISSRE). Pasadena, 2013. 208−217. [doi: 10.1109/ISSRE.2013.6698920]

[36] Wang XY, Zhang L, Xie T, Anvik J, Sun JS. An approach to detecting duplicate bug reports using natural language and execution information. In: Proc. of the 30th Int'l Conf. on Software engineering (ICSE 2008). New York: Association for Computing Machinery, 2008. 461−470.

[37] Runeson P, Alexandersson M, Nyholm O. Detection of duplicate defect reports using natural language processing. In: Proc. of the 29th Int'l Conf. on Software Engineering (ICSE 2007). Minneapolis, 2007. 499−510.

[38] Xiao G, Du X, Sui Y, Yue T. HINDBR: Heterogeneous information network based duplicate bug report prediction. In: Proc. of the 2020 IEEE 31st Int'l Symp. on Software Reliability Engineering (ISSRE). Coimbra, 2020. 195−206.

[39] Wang JJ, Li MY, Wang S, Menzies T, Wang Q. Images don't lie: Duplicate crowdtesting reports detection with screenshot information. Information and Software Technology, 2019, 110.

[40] Dang Y, Wu R, Zhang H, Zhang D, Nobel P. ReBucket: A method for clustering duplicate crash reports based on call stack similarity. In: Proc. of the 2012 34th Int'l Conf. on Software Engineering (ICSE). Zurich, 2012. 1084−1093.

[41] Rakha MS, Bezemer C, Hassan AE. Revisiting the performance evaluation of automated approaches for the retrieval of duplicate issue reports. IEEE Trans. on Software Engineering, 2018, 44(12): 1245−1268.

[42] Lerch J, Mezini M. Finding duplicates of your yet unwritten bug report. In: Proc. of the 2013 17th European Conf. on Software Maintenance and Reengineering. Genova 2013. 69−78.

[43] Gopalan RP, Krishna A. Duplicate bug report detection using clustering. In: Proc. of the 2014 23rd Australian Software Engineering Conf. Milsons Point, 2014. 104−109.

[44] C. Jingliang, M. Zhe, S. Jun. A data-driven approach based on LDA for identifying duplicate bug report. In: Proc. of the 2016 IEEE 8th Int'l Conf. on Intelligent Systems (IS). Sofia, 2016. 686−691.

[45] Akilan T, Shah D, Patel N, Mehta R. Fast detection of duplicate bug reports using LDA-based topic modeling and classification. In: Proc. of the 2020 IEEE Int'l Conf. on Systems, Man, and Cybernetics (SMC). Toronto, 2020. 1622−1629.

[46] Lazar A, Ritchey S, Sharif B. Generating duplicate bug datasets. In: Proc. of the 11th Working Conf. on Mining Software Repositories (MSR 2014). New York: Association for Computing Machinery, 2014. 392−395.

[47] Mikolov T, *et al*. Efficient estimation of word representations in vector space. arXiv preprint arXiv: 1301.3781, 2013.

[48] Hiew L. Assisted detection of duplicate bug reports [MS. Thesis]. British Columbia: Department of Computer Science, the University of British Columbia, 2003.

[49] Li YR, Gopalan RP. Clustering high dimensional sparse transactional data with constraints. In: Proc. of the 2006 IEEE Int'l Conf. on Granular Computing. Atlanta, 2006. 692−695.

[50] Tu FF, Zhu JX, Zheng QM, Zhou MH. Be careful of when: an empirical study on time-related misuse of issue tracking data. In: Proc. of the 2018 26th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering (ESEC/FSE 2018). New York: Association for Computing Machinery, 2018. 307−318.

郑炜(1975－), 男, 博士, 副教授, CCF 高级会员, 主要研究领域为软件测试, 软件安全, 软件仓库挖掘.

王晓龙(1995－), 男, 硕士生, 主要研究领域为文本相似度检测, 重复缺陷报告检测.

陈翔(1980－), 男, 博士, 副教授, CCF 高级会员, 主要研究领域为软件缺陷预测, 软件缺陷定位, 回归测试, 组合测试.

夏鑫(1986－), 男, 博士, 讲师, 博士生导师, 主要研究领域为软件仓库挖掘, 经验软件工程.

廖慧玲(1996－), 女, 硕士生, 主要研究领域为软件缺陷定位.

刘程远(1994－), 男, 硕士生, 主要研究领域为缺陷报告的重复性识别/检测, 深度学习.

孙瑞阳(1994－), 男, 硕士生, 主要研究领域为软件缺陷预测.