软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(1):171−196 [doi: 10.13328/j.cnki.jos.006500]

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

缺陷报告质量研究综述*

邹卫琴[1], 张静宣[1], 张霄炜[2], 陈 林[2], 玄跻峰[3]

[1](南京航空航天大学 计算机科学与技术学院, 江苏 南京 211106)

[2](南京大学 计算机科学与技术学院, 江苏 南京 210023)

[3](武汉大学 计算机学院, 湖北 武汉 430072)

通信作者: 陈林, E-mail: lchen@nju.edu.cn

摘 要: 在软件开发和维护过程中, 缺陷修复人员通常根据由终端用户或者开发/测试者提交的缺陷报告来定位和修复缺陷. 因此, 缺陷报告本身的质量对修复人员能否快速准确定位并修复缺陷具有重要的作用. 围绕缺陷报告质量的刻画及改进, 研究人员开展了大量的研究工作, 但尚未进行系统性的归纳. 旨在对这些工作进行系统性地梳理, 展示该领域的研究现状并为未来的研究方向提供参考意见. 首先, 总结了已有缺陷报告存在的质量问题, 如关键信息缺失、信息错误等; 接着, 总结了对缺陷报告质量进行自动化建模的技术; 然后, 描述了一系列对缺陷报告质量进行改进的方法; 最后, 对未来研究可能面临的挑战和机遇进行了展望.

关键词: 软件质量保障; 缺陷定位与修复; 缺陷报告质量; 质量建模和改进

中图法分类号: TP311

中文引用格式: 邹卫琴, 张静宣, 张霄炜, 陈林, 玄跻峰. 缺陷报告质量研究综述. 软件学报, 2023, 34(1): 171–196. http://www.jos.org.cn/1000-9825/6500.htm

英文引用格式: Zou WQ, Zhang JX, Zhang XW, Chen L, Xuan JF. Survey of Research on Bug Report Quality. Ruan Jian Xue Bao/Journal of Software, 2023, 34(1): 171–196 (in Chinese). http://www.jos.org.cn/1000-9825/6500.htm

Survey of Research on Bug Report Quality

ZOU Wei-Qin[1], ZHANG Jing-Xuan[1], ZHANG Xiao-Wei[2], CHEN Lin[2], XUAN Ji-Feng[3]

[1](College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China)

[2](School of Computer Science and Technology, Nanjing University, Nanjing 210023, China)

[3](School of Computer Science, Wuhan University, Wuhan 430072, China)

Abstract: During the software development and maintenance process, bug fixers usually refer to bug reports submitted by end-users or developers/testers to locate and fix a bug. In this sense, the quality of the bug report largely determines whether the bug fixer could quickly and precisely locate the bug and further fix it. Researchers have done much work on characterizing, modeling, and improving the quality of bug reports. This study offers a systematic survey on existing work on bug report quality, with an attempt to understand the current state of research on this area as well as to open new avenues for future research work. Firstly, quality problems of bug reports reported by existing studies are summarized into a list, such as the missing of key information and errors in information items. Then, a series of work on automatically modeling bug report quality are presented. After that, those approaches are introduced that aim to improve bug report quality. Finally, the challenges and potential opportunities for research on bug report quality are discussed.

Key words: software quality assurance; bug localization and fixing; bug report quality; quality modeling and improving

在软件项目开发和维护过程中, 软件固有的复杂性使得软件缺陷不可避免[1]. 受软件规模的日益增加以及缺陷收集策略的变更 (如仅由项目内部人员报告缺陷到网上公开接收用户报告缺陷) 等因素的影响, 软件项目需要

* 基金项目: 国家自然科学基金 (62002161, 61902181, 61872177, 61872273); CCF-腾讯犀牛鸟基金 (RAGR20200106); 中国博士后科学基金 (2020M671489)
收稿时间: 2021-05-08; 修改时间: 2021-06-24, 2021-09-14; 采用时间: 2021-09-29; jos 在线出版时间: 2021-11-24
CNKI 网络首发时间: 2022-11-15

处理的软件缺陷数目越来越多[2]. 为了便于对软件缺陷进行跟踪和管理, 很多软件项目 (尤其是大型的开源软件项目如 Mozilla (www.mozilla.org) 通常会使用缺陷追踪系统如 Bugzilla (https://www.bugzilla.org/)、JIRA (http://www.atlassian.com/software/jira) 等来接收软件缺陷报告并记录软件缺陷在其生命过程中的各种状态[3,4]. 在使用或测试软件的过程中, 终端用户或开发测试人员可以就其遇到的软件问题向缺陷追踪系统提交缺陷报告, 在报告中记录与该缺陷相关的各种信息, 如问题描述、重现步骤、系统平台和软件版本等[5]. 然后, 缺陷修复人员通过分析缺陷报告中所提供的缺陷相关信息对缺陷进行定位并修复.

缺陷报告作为描述软件缺陷的载体, 其自身质量的高低对缺陷修复人员能否快速准确定位和修复缺陷具有重要影响. 然而, 在实践中, 由于如缺陷报告者经验不一、软件自身及其使用场景的复杂性和缺陷追踪系统功能的不完善等因素的影响, 缺陷修复人员所参考的缺陷报告其质量往往参差不齐[6,7]. 缺陷报告质量的差异影响缺陷定位的效率以及缺陷修复的及时性和准确性, 影响软件的质量以及软件项目的成败. 因此, 对缺陷报告自身质量的研究吸引了越来越多研究者投入到这个重要的研究中, 已成为了软件工程领域的研究热点.

围绕缺陷报告质量这一主题, 研究人员已开展了大量工作, 然而尚未对这些工作进行系统性梳理. 为了揭示该领域的核心问题和相关研究进展, 对相关工作进行综述研究则变得非常重要. 本文旨在通过系统性地调研回答缺陷报告质量这 3 方面的问题, 即已有研究 (1) 揭示了缺陷报告存在的哪些质量问题? (2) 提出了哪些缺陷报告质量度量的方法? (3) 采用了哪些策略来改进缺陷报告自身的质量?

为了回答上述问题, 我们首先通过关键词搜索得到与缺陷报告质量研究相关的文献. 然后, 我们对这些文献进行人工审查筛选出真实相关的文献. 接着, 我们进一步对这些真实相关文献的参考文献进行人工审查, 识别出搜索关键词未覆盖到的真实相关文献. 最终, 我们得到用于本研究的真实相关文献 120 篇. 为了回答缺陷报告质量的刻画、建模和改进这 3 个方面的研究问题, 我们对这 120 篇调研文献进行系统性的分析整理. 通过该过程, 我们了解了缺陷报告质量研究的现状, 明晰了现有研究工作的局限性和所面临的挑战, 并对该领域未来的研究方向进行了展望.

本文第 1 节介绍了缺陷报告质量研究的相关背景. 第 2 节描述了本综述的文献检索方法和相关文献汇总信息. 第 3 节描述了缺陷报告存在的相关质量问题. 第 4 节和第 5 节分别从质量建模和改进两个方面介绍了相关研究现状. 第 6 节分析了缺陷报告质量研究所面临的挑战和机遇. 第 7 节总结了本文内容.

1 背 景

1.1 缺陷追踪系统

软件系统本身的复杂性导致软件缺陷不可避免[1]. 随着软件系统的规模和复杂性逐渐增加, 缺陷的数目亦随之增加, 最初通过人工方式将缺陷记录在电子表格进行缺陷管理的传统方式已不能满足对数目庞大的软件缺陷的管理 (尤其是当多人需要对缺陷数据进行并发访问和提交时). 为了解决缺陷管理这一问题, 很多软件项目, 尤其是大型软件项目和组织开始使用缺陷追踪系统 (bug tracking system, BTS) 来收集和跟踪软件项目的缺陷[2].

缺陷追踪系统是一种供软件项目人员管理和跟踪软件缺陷的软件. 像大多数信息系统一样, 缺陷追踪系统主要管理的是软件缺陷数据, 其主要功能是供用户访问和提交软件缺陷信息. 用户登录系统后可以提交缺陷报告反馈其所遇到的软件问题或修改已经存在的缺陷报告信息, 同时其也可以通过系统的搜索功能搜索感兴趣的缺陷报告[4]. 不同的项目组织其所使用的缺陷追踪系统也可能不同, APTEST 公司在其网站 (http://www.aptest.com/bugtrack.html) 上列出了截止到 2015 年的 70 多种开源和商业缺陷追踪系统. 其中最主流的两种缺陷追踪系统为 Bugzilla 和 JIRA. Bugzilla 是一款开源免费的缺陷管理软件, JIRA 则是一款商用软件. 这两款软件被各种规模不同的开闭源软件和相关组织所使用, 典型的用户包括 Apache、IBM、Mozilla、NASA、Linux 等.

1.2 缺陷报告

在类似 Bugzilla 和 JIRA 的缺陷追踪系统中, 软件缺陷以缺陷报告的形式被记录. 用户在提交缺陷报告时, 通

常需提供各种信息, 如问题描述、复现步骤等. 这些信息可以帮助开发人员了解相应的软件缺陷并对其进行定位和修复. 图 1 展示了使用 Bugzilla 的 Eclipse 项目中一个典型的缺陷报告 (bug ID 为 13587) 片段. 该报告是一个半结构化的信息载体: 不但包括一些结构化的信息条目, 如当前缺陷所处的状态 (status)、所属产品和组件 (product 和 component)、优先级和严重程度 (importance 字段的两个组成部分)、报告该缺陷的人 (reported) 等, 还包括一些非结构化的自由文本信息, 如缺陷的问题概要描述 (summary)、问题详细描述 (description) 和围绕该缺陷的讨论 (comments).

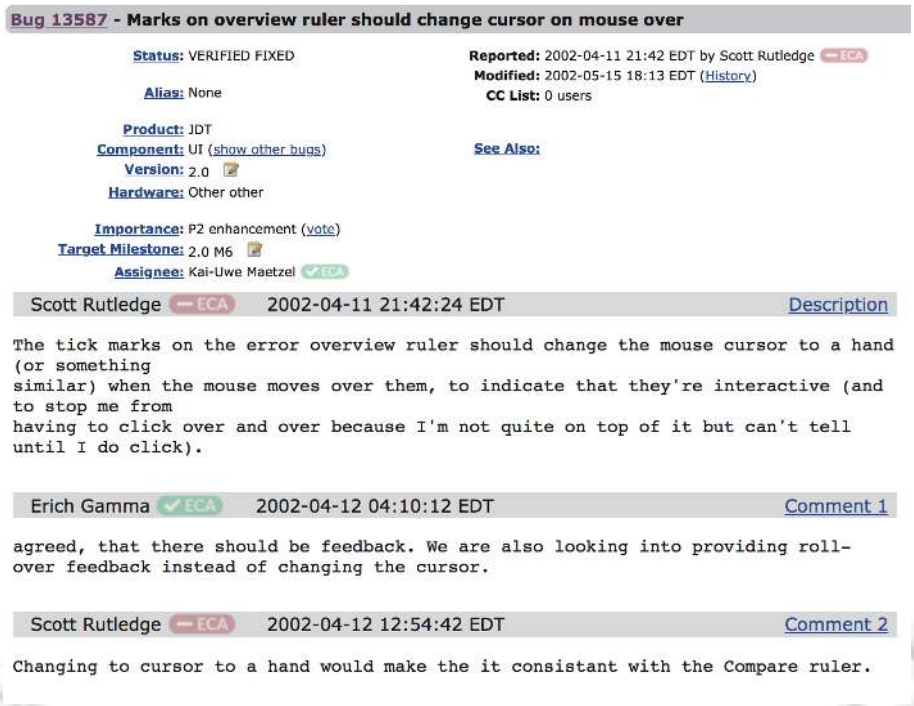

图 1 Eclipse 中 bug ID 为 13587 的缺陷报告片段

1.3 缺陷报告质量

为了更好地改进软件产品, 保障软件质量, 很多软件项目支持网上公开接受产品用户报告问题, 即允许任何用户向它们的缺陷追踪系统中提交缺陷报告. 然而, 普通用户不同于专业的软件开发或者测试人员 (实际上, 即使专业的开发测试者在报告缺陷时也可能出错[8]), 他/她们的经验往往不同, 加之系统本身的复杂性 (如有的软件产品组件繁多交互复杂, 导致缺陷报告者无法准确判断缺陷所属的产品组件) 和缺陷追踪系统某些功能方面的局限性 (如没有较好的信息补充和提示信息等), 导致用户所提交的缺陷报告质量参差不齐. Xia 等人发现大约 80% 的缺陷报告存在信息条目在后期缺陷处理的过程中被重新赋值的情况[9]. 此外, Zimmermann 等人发现, 开发者期望获得的缺陷信息条目和用户提供的缺陷信息条目存在不匹配的问题, 导致开发者在处理缺陷的过程中无法及时复现和处理缺陷, 延长缺陷的修复时间, 进而影响软件的质量[10].

由于缺陷修复对软件质量保障有重要作用以及缺陷报告质量对缺陷定位修复具有决定性影响, 缺陷报告质量的研究引起了广大研究人员的注意, 形成了一系列研究成果. 有的研究侧重于对缺陷报告存在的质量问题进行分析[6,9,11], 有的研究尝试对缺陷报告的质量进行自动化建模度量[10,12,13], 还有的研究则提出了一些策略和技术以对缺陷报告的质量进行改进[14−18], 即研究主题涉及到对缺陷报告质量的刻画、度量和改进. 然而, 针对已有的大量研究成果, 尚缺乏系统性的综述研究. 为了填补这一空白, 本研究尝试对这些工作进行系统性梳理. 通过梳理, 一方面能帮助明晰缺陷报告质量研究中的核心研究问题及其进展, 更重要的是, 能帮助发现已有研究所面临的问题和挑

战, 为将来的研究方向提供启发.

2 文献检索

2.1 文献检索与筛选

本文主要采取如下步骤抽取缺陷报告质量研究领域的相关文献.

(1) 检索策略: 为了对缺陷报告质量研究进行全面的总结, 我们采取了与 Chen 等人[19]、涂菲菲等人[20]的工作相同的检索策略. 即首先在 DBLP Computer Science Bibliography (https://dblp.uni-trier.de) 中使用关键词进行搜索, 随后人为检查搜索结果确定相关文献, 最后阅读相关文献的参考文献部分抽取剩余相关工作.

(2) 检索关键词: 由于本综述的研究主题为缺陷报告质量, 我们首先在 DBLP 中测试了关键词“bug report quality”“defect report quality”“test report quality”的搜索结果, 发现这些词只能检索出极小部分参考文献 (3 个检索关键词所检索出的文献数分别为 14, 2, 7). 为了扩大文献覆盖范围, 我们使用“bug report”“defect report”“issue report”“failure report”“test report”“problem report”“defect record”作为最终的检索关键词在 DBLP 上进行搜索.

(3) 文献审核: 在使用 (2) 中关键词进行搜索得到相关文献候选集后, 我们对其中每篇文献进行人工审核, 删除不符合研究主题的文献. 在本调研中, 我们只关注和保留由用户提交的缺陷报告存在的质量问题、质量度量和质量改进研究. 同时, 鉴于不同的人员、系统或组织对缺陷报告质量的理解或定义各不相同, 为了避免调研主题的过度分散及调研文献数目不可控的问题, 本文主要关注由软件开发维护者在实际处理缺陷报告的过程中理解、定位、修复缺陷时所提到 (或遇到) 的缺陷报告质量问题. 从真实场景反馈的问题出发, 研究其现状及进展, 为未来研究提供重要的实践评估标准. 更进一步地, 本文重点关注缺陷报告内部存在的质量问题, 冗余的缺陷报告相关研究不纳入本文的调研范围. 排除冗余缺陷报告相关研究主要出于如下考虑: 1) 本文重点关注缺陷报告本身质量问题, 作为报告了同一问题的冗余缺陷报告, 研究其中的一份, 有一定代表性. 2) 已有文献认为冗余不是一个大的质量问题. 据文献 [21] 对开发者的问卷调查发现, 尽管大多数开发者会碰到冗余缺陷报告, 但很少将其当做一个严肃的质量问题; 相反, 开发者指出冗余缺陷报告往往能提供额外的缺陷信息帮助更快的修复缺陷. 当前也确实存在一些通过融合冗余缺陷报告产生质量更高的缺陷报告提升软件任务的研究工作 (如文献 [18,21] 的工作等). 3) 目前对冗余缺陷报告的研究工作很多 (例如, 单独在 DBLP 上使用关键词“duplicate report”搜索所得相关论文数为 67), 适合开新论文对其单独研究.

(4) 文献补漏: 通过 (3) 中人工审核得到相关文献后, 我们进一步检查所得相关文献的参考文献部分, 以得到一些关键词未覆盖到的相关文献. 最终, 筛选出 120 篇文献用于本文的研究综述. 所调研文献列表可详见如下网址: https://github.com/SurfGitHub/BRQuality_JOS/blob/main/paperList_BRQuality.pdf

2.2 历史文献的汇总

通过文献检索和筛选, 最终获取到 120 篇文献用于本文研究, 文献发表时间为 2005–2020 年. 在这 120 篇论文中, 会议论文 79 篇, 期刊 41 篇. 其中, 有 10 篇重点在分析缺陷报告自身存在哪些质量问题; 有 19 篇主要对质量进行建模度量; 91 篇论文试图对缺陷报告的质量进行改进. 图 2 按年份展示了缺陷报告质量研究文献的数量分布. 从图 2 可以发现, 随着年份的增长文献数量波动上升, 在近 3 年的文献数量最多. 这说明缺陷报告质量的研究一直是稳定且活跃的研究领域, 在近些年越来越受到研究者的关注.

表 1 展示了相关文献在计算机主流会议和期刊 (仅展示了 CCF 推荐列表中相关会议和期刊) 发表的数量分布, 其中会议论文 39 篇, 包含 ICSE 论文 2 篇、FSE 论文 6 篇、ASE 论文 3 篇、SANER 论文 6 篇、ESEM 论文 4 篇、ICPC 论文 2 篇、ICSME 论文 3 篇、ISSRE 论文 1 篇、CSCW 论文 1 篇、EASE 论文 2 篇、MSR 论文 3 篇、QRS 论文 1 篇、ICST 论文 1 篇、ACML 论文 1 篇、APSEC 论文 1 篇、COMPSAC 论文 1 篇、SEKE 论文 1 篇; 期刊论文 15 篇, 包含 TSE 论文 2 篇、TOSEM 论文 1 篇、EMSE 论文 4 篇、JSEP 论文 1 篇、JSS 论文 2 篇、JCST 论文 1 篇、IST 论文 1 篇、IJSEKE 论文 3 篇等. 表 1 所列文献数达到了 54 篇, 总占比为 45%. 这个分布说明, 对缺陷报告质量的研究在国际主流会议和旗舰期刊中处于较活跃的状态, 是较热门的研究领域.

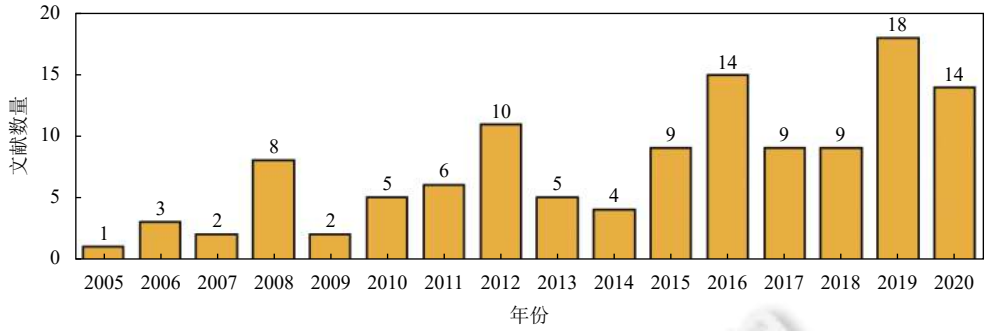

图 2 相关论文发表的年代分布

表 1 相关论文在计算机主流会议/期刊发表的数量分布情况

类型	出版物缩写	文章总数	发表年份 (文章数)	引用列表
会议	ICSE	2	2013 (1), 2019 (1)	[18,27]
	FSE	6	2008 (1), 2015 (2), 2017 (1), 2019 (1), 2020 (1)	[6,13,16,41,51,52]
	ASE	3	2007 (1), 2016 (1), 2019 (1)	[37,40,57]
	SANER (CSMR, WCRE)	6	2011 (1), 2012 (1), 2013 (2), 2014 (1), 2018 (1)	[9,12,30,69,82,128]
	ESEM	4	2011 (2), 2014 (1), 2016 (1)	[8,23,24,39]
	ICPC	2	2015 (1), 2017 (1)	[59,61]
	ICSME (ICSM)	3	2008 (1), 2013 (1), 2017 (1)	[77,85,94]
	ISSRE	1	2020 (1)	[32]
	CSCW	1	2010 (1)	[43]
	EASE	2	2016 (2)	[25,26]
	MSR	3	2008 (1), 2009 (1), 2012 (1)	[33,46,54]
	QRS	1	2016 (1)	[45]
	ICST	1	2016 (1)	[56]
	ACML	1	2019 (1)	[64]
	APSEC	1	2012 (1)	[87]
	COMPSEC	1	2014 (1)	[129]
	SEKE	1	2011 (1)	[101]
期刊	TSE	2	2010 (1), 2018 (1)	[10,28]
	TOSEM	1	2011 (1)	[63]
	EMSE	4	2015 (1), 2016 (1), 2020 (2)	[11,31,76,100]
	JSEP	1	2019 (1)	[53]
	JSS	2	2016 (1), 2020 (1)	[90,93]
	JCST	2	2012 (2)	[74]
	IST	1	2016 (1)	[67]
	IJSEKE	3	2018 (1), 2019 (2)	[89,98,99]

3 缺陷报告存在的质量问题

在 120 篇相关文献中, 有 10 篇文献 (文献 [6,8–11,22–26]) 旨在研究缺陷报告存在哪些质量问题. 在本节中, 我们首先对已有工作的具体研究方法进行概要性介绍, 然后总结罗列文献调研所得到的软件开发者和维护者提到或遇到的缺陷报告质量问题, 最后介绍存在质量问题的具体信息条目.

3.1 研究方法

在 10 篇关于缺陷报告存在的质量问题实证研究中, 研究人员采用的主要技术手段为问卷访谈的定性分析法和基于缺陷报告数据的定量分析方法, 调研范围包括开源社区和工业界, 所关注的缺陷报告大多为一般性的缺陷

报告 (generic bug report, 即不区分缺陷的具体类型), 表 2 展示了这些文献的具体信息. 现对相关工作进行概要介绍如下.

表 2 缺陷报告存在的主要质量问题相关研究

文献	缺陷类型	调研方法	调研对象/数据集	回复者/数据集规模	相关信息条目
Bettenburg等人[6,10] (2008)	一般性	问卷调查	Apache、Eclipse、Mozilla的开发者和缺陷报告者	466	产品、版本、复现步骤、堆栈信息、测试用例、期望/观测行为、截图等16个条目
Breu等人[22] (2009)	一般性	仓库挖掘	Mozilla和Eclipse的缺陷报告	600	复现步骤、构建号、操作系统、测试用例、程序输出、截图等
Laukkanen等人[23] (2011)	一般性	问卷调查	工业界的开发者 (5家公司)	74	文献[6]中条目+5个领域属性 (如应用配置、操作数据等)
Wang等人[8] (2011)	一般性	仓库挖掘	工业项目Qone	5 637	问题概要/详细描述、复现步骤、所属模块、严重程度、优先级、运行环境
Xia等人[9] (2014)	一般性	仓库挖掘	OpenOffice、NetBeans、Eclipse、Mozilla的缺陷报告	190 558	产品、组件、严重程度、优先级、操作系统、版本等8个条目
Davies等人[24] (2014)	一般性	仓库挖掘	Eclipse、Firefox、Apache HTTP、Facebook API的缺陷报告	1 600	观测/期望行为、堆栈信息、测试用例、复现步骤、截图、应用代码等10个条目
Garousi等人[25] (2016)	一般性	问卷调查	工业界的开发者 (一家公司)	38	同Bettenburg等人[6]
Yusop等人[26] (2016)	可用性	问卷调查	开源社区和工业界的开发者和缺陷报告者	147	问题概要描述、问题起因、修复方案、观测/期望行为、复现步骤、严重程度、产品、硬件等13个条目
Soltani等人[11] (2020)	一般性	访谈/问卷调查/仓库挖掘	访谈: 工业界的开发者(电商、ERP、汽车制造) 问卷: GitHub开发者 仓库挖掘: 250个GitHub项目的缺陷报告	受访者/问卷回复者: 35/305 缺陷报告: 835 381	复现步骤、堆栈信息、修复建议、用户信息、代码片段等7个条目

从表 2 可以发现, 在 10 篇相关文献中, 有 5 篇论文 (文献 [6,10,23,25,26]) 使用了问卷调研的方式, 有 4 篇文献 (文献 [8,9,22,24]) 使用了仓库挖掘的方式 (即分析缺陷报告仓库数据), 有 1 篇论文 (文献 [11]) 综合使用了问卷、访谈和仓库挖掘的方式. 问卷访谈的对象既有来自开源社区的开发者/报告者, 又有来自工业界的开发者/报告者. 开源社区的问卷对象主要来自典型开源项目如 Apache、Eclipse、Mozilla 和一些 GitHub 项目, 工业界的问卷对象基本局限在研究者所接触到的个别或少数几个公司的人员. 在问卷回复的规模上, 开源社区相较工业界回复数明显较多 (466 份[6,10]、305 份[11]、74 份[23]、38 份[25]). 在以仓库挖掘为主要研究手段的文献中, 缺陷报告同样主要取自几个典型的开源项目如 Mozilla、Eclipse、Openoffice、NetBeans (个别工作使用了 Facebook API、Qone 和 GitHub 项目数据), 各工作所分析的缺陷报告数据规模差异较大 (600, 1 600, 5 637, 190 588 和 835 381). 而在分析的具体缺陷报告类型上, 绝大多数 (9/10) 文献考虑的都是一般性的缺陷报告, 只有 Yusop 等人[26]重点关注了可用性缺陷报告 (usability bug report, 指影响用户体验的在用户接口处出现的缺陷) 存在的质量问题. 在一般性缺陷报告信息条目 (如复现步骤、观测/期望行为) 的基础上, Yusop 等人[26]额外分析了可用性相关信息条目 (如可用性原则 (usability principle)、UI 事件轨迹 (UI event trace)、问题起因 (cause of problem, 如问题出现时在做什么)) 等存在的质量问题.

值得注意的是, 在 10 篇研究文献中, Bettenburg 等人[6,10]最早展开了从软件开发者的角度去探究缺陷报告存

在哪些质量问题的研究. 该工作为缺陷报告存在的质量问题研究奠定了重要调研基础, 后续的研究大多基于他们的工作进行展开, 其主要实现方式为在工业项目上完全或部分重现[6,10]的实验[23,25], 或通过挖掘缺陷仓库数据对其所提质量问题进行定量分析与验证[9]. 鉴于此, 在第 3.2 节罗列相关缺陷报告质量问题时, 我们只对文献 [6,10] 中的工作进行详细介绍, 其他工作的细节在表 2 中展示, 不再一一详细介绍.

3.2 存在的质量问题

作为从开发者角度研究缺陷报告质量问题的开创性工作, Bettenburg 等人较为深入全面地分析了缺陷报告存在的质量问题[6,10]. 具体地, 针对缺陷报告的 16 个信息条目 (包括产品 (product)、组件 (component)、版本 (version)、严重程度 (severity)、硬件 (hardware)、操作系统 (operating system)、问题概要描述 (summary)、构建信息 (build information)、观测行为 (observed behavior)、期望行为 (expected behavior)、复现步骤 (steps to reproduce)、堆栈信息 (stack traces)、截图 (screenshots)、代码示例 (code examples)、错误信息 (error reports) 和测试用例 (test cases)), 作者向 Apache、Eclipse 和 Mozilla 的开发者和缺陷报告者发送了 2 226 份调查问卷 (收回 466 份). 在问卷中, 面向开发者, 作者询问了他们在修复缺陷时会使用到哪些信息、哪些信息对修复缺陷最有帮助、以及遇到的主要问题; 面向缺陷报告者, 作者主要询问了他们一般会提供哪些信息、哪些信息最难提供、以及认为哪些信息对开发者修复缺陷最有帮助. 通过调研, 作者发现缺陷修复者在修复缺陷时所用或认为重要的信息条目缺陷报告者未能充分或准确提供, 信息供需之间存在较大的鸿沟. 具体质量问题表现在以下几个方面:

① 信息不全或缺失 (文献 [6,8,10,11,23,25,26]).

② 信息错误, 包括整个信息条目 (如版本) 错误或条目内部 (如复现步骤) 存在错误 (文献 [6,8,9,10,22,23,25,26]).

③ 自由文本可阅读性差 (如太长)(文献 [6,10,25,26]).

④ 缺陷相关信息已提供, 但细节方面需要报告者进一步解释澄清 (文献 [8,22,26]).

⑤ 其他 (如无效、垃圾信息等)(文献 [6,8,10,26]).

结合上述所列质量问题、问题提及的参考文献和表 2 中对具体参考文献细节的提供, 我们可以发现, 无论在工业界还是开源社区, 所研究的是一般性缺陷报告还是特定类型的缺陷报告 (如可用性缺陷报告), 软件开发维护者所遇到的缺陷报告质量问题均有一定的共性, 其中, ① 信息不全或缺失以及 ② 信息错误是软件开发维护者所提到或遇到的最普遍 (严重) 的问题. 尽管已有工作在调研领域多样性和调研对象规模上存在一定的局限性, 上述质量问题依然能为我们了解软件产业中软件开发者和维护者处理缺陷时遇到的缺陷报告质量问题提供重要认知基础.

3.3 存在质量问题的具体信息条目

第 3.2 节从宏观上总结了缺陷报告存在的质量问题. 为了对缺陷报告质量问题有一个较为细致而全面地了解, 本节将各文献提到的存在质量问题的具体信息条目逐一介绍. 表 3 展示了各文献所提具有质量问题的具体信息条目.

表 3 各文献及其对应的存在质量问题的具体信息条目

文献	存在质量问题的信息条目
Bettenburg等人[6,10] (2008)	堆栈信息、测试用例、版本、观测/期望行为、代码示例、产品、操作系统、复现步骤
Breu等人[22] (2009)	复现步骤、构建号、操作系统、测试用例、示例、程序输出、截图、组件等
Laukkanen等人[23] (2011)	观测行为、复现步骤、用户输入、应用配置、堆栈信息、产品信息、操作数据等
Wang等人[8] (2011)	问题概要描述、问题详细描述、复现步骤、所属模块、严重程度、优先级、运行环境
Xia等人[9] (2014)	产品、组件、严重程度、优先级、操作系统、版本
Davies等人[24] (2014)	观测行为、复现步骤、堆栈信息、测试用例等
Garousi等人[25] (2016)	复现步骤、观测行为等
Yusop等人[26] (2016)	问题起因、可用性原则、录像、UI事件轨迹和问题概要等
Soltani等人[11] (2020)	崩溃描述、复现步骤、测试用例、堆栈信息、修复建议、用户内容

Bettenburg 等人[6,10]发现开发者定位修复缺陷时常用的条目为复现步骤、观测和期望行为、堆栈信息、测试用例、问题概要描述、截图、版本等; 然而, 除了复现步骤和观测/期望行为, 报告者对剩余常用条目的提供并不理想, 尤其是在堆栈信息、测试用例、代码示例、产品和操作系统这几个条目上. 开发者常遇到的问题包括复现步骤和测试用例有误、版本和观测/期望行为错误等. Garousi 等人[25]在一家土耳其的国防信息技术领域的公司重现了 Bettenburg 等人[6,10]的实验, 作者发现, 信息不全、复现步骤有误与观测行为不正确是最严重的几个问题, 亟待被解决.

Breu 等人[22]通过分析随机抽取来自 Eclipse 和 Mozilla 的 600 个缺陷报告评论中开发者的提问和回答, 同样发现缺少复现缺陷相关的信息是最主要的一类评论. 在理解缺陷和调试的过程中, 开发者经常需要请求缺陷报告者提供额外的一些信息 (如复现步骤、构建号、操作系统、测试用例、示例、程序输出和截图等) 以及对一些细节进行澄清说明. 此外, 缺陷报告也经常被指派到错误的组件 (即缺陷报告中组件条目值错误).

Laukkanen 等人[23]向 6 个工业软件开发组织的 147 人发放问卷, 分析了 18 个信息条目的有用性及存在的问题. 作者发现, 超过 50% 的问卷参与者提到观测行为、复现步骤、用户输入、应用配置、堆栈信息、产品信息、操作数据等经常存在信息错误或者缺失的问题.

Soltani 等人[11]通过访谈不同领域的开发者得到了一些对调式至关重要的信息条目. 然后, 他们对 GitHub 上最流行项目的维护者和流行项目的开发者发放问卷, 以验证访谈的结论. 作者发现崩溃描述、复现步骤或测试用例、堆栈信息对开发者调式问题非常重要; 其中复现步骤、堆栈信息、修复建议、用户内容对缺陷修复也有显著性影响. 然而, 通过对 GitHub 上 835 381 个缺陷报告进行分析, 他们发现超过 70% 的报告都不包含这些信息.

Davies 等人[24]检查了 4 个开源项目 (Eclipse、Apache HTTP、Firefox、Facebook API) 的 1 600 个缺陷报告的内容来分析缺陷报告的质量问题. 他们分析了 Bettenburg 等人[6]提到的 10 个最重要的指标, 包括观测行为、复现步骤、错误信息等. 作者发现, 只有观测行为和期望行为在超过一半的缺陷报告中出现 (大多数报告最多包含 3 个重要条目), 被开发者认为非常重要的条目如堆栈信息和测试用例则很少出现在报告中.

Xia 等人[9]对 4 个开源项目 (OpenOffice, NetBeans, Eclipse 和 Mozilla) 的 190 558 个缺陷报告进行分析, 统计了 8 个信息条目的重新赋值情况, 包括产品、组件、严重程度、优先级、操作系统、版本、修复者和状态, 4 个项目中 8 个条目被重新赋值的占比分别为 9.76%–31.40%、14.13%–64.04%、0%–9.19%、10.50%–16.34%、4.55%–12.34%、6.04%–12.34%、51.04%–74.73%、8.14%–26.54%.

Wang 等人[8]通过分析 Qone 软件的数据发现, 超过 35% 的测试人员都提交过无效的缺陷报告, 其中无效的缺陷报告占比为 26%; 同时, 超过 82% 的测试人员基本都提交过描述不够清楚的缺陷报告, 其中描述不清的缺陷报告占比为 23%. 在其所分析的 7 个信息条目中, 作者发现模块 ID、问题详细描述和概要描述、软件运行环境和操作步骤修改最频繁.

Yusop 等人[26]发现用户在可用性缺陷报告中极少提供与可用性相关的信息. 此外, 报告者认为较难提供的信息包括问题起因、可用性原则、录像、UI 事件轨迹和问题概要, 其中报告者认为最难提供的问题起因在修复可用性缺陷时帮助最大.

3.4 小 结

从软件开发维护者的角度来看, 当前缺陷报告主要存在的质量问题为信息条目缺失和错误, 宏观表现为缺陷修复者在修复缺陷时所用或认为重要的信息条目缺陷报告者未能充分或准确提供, 信息供需之间存在较大的鸿沟; 这些重要信息条目包括复现步骤、堆栈信息、测试用例、观测/期望行为等.

4 缺陷报告质量自动化度量

关于缺陷报告质量的自动化度量, 当前研究依据度量方式主要可分为两大类: 一类是通过抽取缺陷报告相关的各种指标 (如堆栈示例信息、报告者经验等) 对缺陷报告或者具体信息条目的质量进行直接度量 (14 篇, 文献 [6,10,12,13,27–36]), 度量的结果为具体的质量等级 (如好、中、差) 或者数值评分 (不具体划分等级, 分值越高

被认为质量越好). 另一类则通过预判其他指标来间接度量一个缺陷报告的质量 (5 篇, 文献 [37–41]), 这些指标可看作是质量的代理指标, 其被相关研究人员认为与质量有较密切关系, 包括缺陷报告的处理代价、揭示真实问题的可能性. 关于对质量的间接度量这一部分, 我们只选取了那些作者明确将代理指标与缺陷报告质量联系在一起的文献, 其他未做显示说明只是单纯预测代理指标的工作不纳入本调研的文献范围. 表 4 概要展示了质量度量的相关工作, 现对其进行具体介绍如下.

表 4 缺陷报告质量自动化评估相关文献

评估类型	评估粒度	质量度量结果	关注/利用特征	依赖技术	文献
直接度量	整个缺陷报告	质量等级	列表枚举信息、截图、堆栈信息、补丁、代码示例的存在情况、关键词完整性、报告者声誉、文本语言学特征、领域特征等	传统机器学习、深度学习、统计	[6,10,12,27,28,29,30,31,32]
		无等级, 数值越高越好	文本主题的内聚性	LDA	[33]
	问题详细描述	分类等级	文本的可重现性、可观测性、确定性和聚焦性、代码示例和特殊关键词的存在情况等	传统机器学习、启发式规则	[34,35]
	复现步骤	分类等级	步骤的缺失和与交互的匹配程度	词嵌入、自然语言语法分析、动态执行、启发式规则等	[13]
	评论	无等级, 数值越高越好	评论的内聚性	LSA	[36]
间接度量	整个缺陷报告	处理代价	报告者声誉、文本可读性、日负载、早期所添评论和附件数等	传统机器学习	[37,38]
		对真实问题的揭露	文本相似性、情感、报告者经验、图片相似性、问题的严重性和多样性	传统机器学习、多目标优化、组合	[39,40,41]

4.1 直接质量度量

对缺陷报告的质量度量已有工作按照度量单元的粒度可细分为两类: 一类是对整个缺陷报告的质量进行度量, 另一类是对缺陷报告内部的具体信息条目如复现步骤进行质量度量.

整个缺陷报告的质量度量: 就现有文献而言, 对缺陷报告质量的直接度量主要以机器学习方法为主, 将其作为一个分类问题进行解决, 模型的输出为质量的具体等级[6,29]. 个别工作使用传统的基于信息检索的技术如隐含狄利克雷分布 (latent Dirichlet allocation, LDA) 等通过分析缺陷报告的文本内容来度量其质量, 输出结果为数值评分而非等级[33]. 在以机器学习为主的质量度量方面, 已有研究的出发点通常是从缺陷报告抽取各方面的特征, 包括文本内容特征、结构化条目信息特征、报告者特征等, 然后使用传统的机器学习方法如支持向量机 (support vector machine, SVM)、朴素贝叶斯 (Naïve Bayes, NB) 等进行模型训练和预测. 对于使用 LDA 技术进行质量评估的研究, 则主要依据报告文本内容主题的内聚性进行判断. 具体如下.

Bettenburg 等人开发了原型工具 CUEZILLA 来评估一个缺陷报告的质量[6,10]. 该工具为一个有监督的机器学习预测模型 (作者对比了包括支持向量机、线性回归等多种机器学习模型), 其训练实例特征的选取主要基于缺陷修复者希望用户提供的缺陷信息 (通过前期问卷调查获取), 包括是否包含列表枚举信息 (如+、−、1.、(1) 等)、截图、堆栈信息、补丁、代码示例、问题描述文本可读性和关键词完整性. 模型输出为缺陷报告质量的评分 (1–5 分, 分数越高表示缺陷报告质量越好) 或其评分所对应的等级 (如差、一般、好等). 在由开发者标记的 289 个缺陷报告 (来自 Eclipse、Apache、Mozilla) 数据集上测试, CUEZILLA 能达到 0.44 的准确率 (即预测等级和开发者标记等级一致).

Aversano 等人[29]重点分析了缺陷报告者的声誉对报告质量的影响, 并结合 Bettenburg 等人[6,10]所提的一些指标如堆栈信息和附件是否存在、关键词完整性、问题描述文本可读性等特征, 训练机器学习模型 (对比了 11 个机器学习算法), 实现了对缺陷报告质量的自动化评估, 评估等级为非常差到非常好 5 个等级. 在 150 个来自开源软

件 PrimeFaces 缺陷报告构成的数据集上 (每个缺陷报告由硕士生、研究人员和开发者 3 个人进行打分评级), 其所提方法能达到 0.77 左右的精确度 (precision).

Bhattacharya 等人[30]提出使用问题描述文本的长度 (单词数)、复现步骤、观测/期望行为及其他信息 (包括版本号、触发缺陷的输入等) 指标来度量手机应用的缺陷报告质量, 并重点比较了安全性 (security) 缺陷报告和非安全性 (non-security) 缺陷报告的质量高低. 作者认为缺陷数据集中相关指标提供的报告占比越高, 该数据集质量也相对越高.

Chen 等人提出了一套自动化度量众包测试报告质量的框架 TERQAF[12]. 该框架从 4 个方面来刻画软件质量, 包括形态学 (morphological)、词汇性 (lexical)、分析性 (analytical) 和相关性 (relational) 指标. 形态学指标包括测试报告问题描述文本的大小 (词句等数目)、文本可读性、标点符号的使用; 词汇性指标包括不明确词、指代词、指导词的使用情况; 分析性指标包括否定词、行为词、动作词和接口元素词的使用情况; 相关性指标包括列表枚举、操作环境和截图的提供情况. 在获取到每个缺陷报告这些指标的数值后, 使用阶步转变函数将每个数值指标转换成“好”或“坏”两个类型值. 如果一个缺陷报告有至少 60% 的指标值为“好”, 则该报告被认为具有高质量. 在其扩展版论文[31]中, 作者使用了逻辑回归分类器来预测一个众包测试报告的质量高低. 作者共使用了 5 个工业界项目的 936 个测试报告进行评估 (每个测试报告由 3 个开发者进行打分确定“好”“坏”的类别), 实验发现所提框架能达到 0.85 左右的精确度 (precision)、0.76 左右的召回率 (recall) 和 0.80 左右的 F 值 (F-measure).

Linstead 等人提出使用缺陷报告的内容内聚性 (coherence) 来评估缺陷报告的质量[33]. 具体地, 作者将每一个缺陷报告当做一个文档, 使用 LDA 主题模型对缺陷报告文档集建模得到每个缺陷报告的一系列主题及其概率分布, 然后计算这些主题的信息熵来衡量缺陷报告内容的内聚性, 内聚性越高, 表明该报告所涉主题越集中, 其对应的质量也越好. 在评估时, 作者主要通过人为检查 10 个内聚性高/低 (由所提方法计算所得) 的缺陷报告的真实情况来评判其方法的有效性.

无效缺陷报告在开闭源软件中具有一定的普遍性[8,42], 鉴于此, 一些研究人员试图通过构建预测模型将缺陷报告进行有效和无效的质量等级划分. 具体地, Zanetti 等人[27]通过利用报告者的交互活动构建报告者社交网络, 并抽取其社交网络属性 (如中心度) 构建 SVM 模型预测报告的有效性, 在 4 个开源软件 (包括 Eclipse、NetBeans、Firefox、Thunderbird) 的 700 000 多个缺陷报告上, 模型在有效缺陷报告上能达到 0.903 的精确度 (precision) 和 0.389 的召回率 (recall).

Fan 等人[28]在 Zanetti 等人[27]所提报告者社交网络属性基础上, 额外使用了其他 3 类特征, 包括报告者经验 (所提缺陷报告数和有效缺陷报告数)、报告完整性 (堆栈信息、复现步骤和补丁等的包含)、报告可读性、文本有效性 (问题概要和详细描述的有效性, 使用朴素贝叶斯方法预测获得). 最后, 利用所有的 33 个特征, 以随机森林为学习模型, 预测一个缺陷报告是否有效, 在 5 个开源软件 (包括 Eclipse、NetBeans、Mozilla、Firefox、Thunderbird) 的 560 697 个缺陷报告上, 模型的 AUC 和有效/无效类别的 $F1$ 值分别为 0.81, 0.74/0.67.

与 Fan 等人[28]和 Zanetti 等人[27]使用传统机器学习模型不同, He 等人[32]提出使用深度学习的方法来判断缺陷报告的有效性. 作者将缺陷报告的问题概要和问题描述文本使用词嵌入模型进行语义特征的抽取, 随后使用卷积神经网络进行学习预测, 并进一步对能帮助解释报告是否有效的成分进行了分析与识别, 在 5 个开源软件 (包括 Eclipse、NetBeans、Mozilla、Firefox、Thunderbird) 的 540 491 个缺陷报告上, 模型的 AUC 和有效/无效的 $F1$ 值分别为 0.85, 0.80/0.69.

具体信息条目的质量度量: 对缺陷报告具体信息条目的质量进行度量的工作有 4 个, 分别是对问题详细描述 (description)、评论 (comments) 和复现步骤进行质量度量, 它们的共同点是对各条目所对应的自由文本信息进行质量分析. 这 4 个工作所采用的技术路线和度量指标均不相同, 具体如下.

(1) 问题详细描述. Schugerl 等人[34]提出了 4 个维度及对应的 10 个指标来对缺陷报告的问题详细描述条目进行质量评估. 4 个维度为可重现性 (reproducibility, 对缺陷重现步骤或上下文的提供)、可观测性 (observability, 对所观测到的行为的描述清晰程度)、确定性 (certainty, 报告者对所报问题的猜想确定程度) 和聚焦性 (focus, 包含偏离主题 (如抱怨) 的文本程度). 10 个指标包括复现步骤/上下文、枚举的数目、代码片段的数目、单词数、拼写

错误等. 基于 10 个指标训练朴素贝叶斯模型来预测问题详细描述文本的质量, 质量级别包括非常差到非常好这 5 个等级. 作者使用了 ArgoUML 的 163 个缺陷报告 (其中由参与 ArgoUML 项目的 9 个硕士、博士研究生对报告进行质量评级) 作为数据集评估其度量技术, 在考虑差一 (off-by-one, 如“非常好”被预测为“好”) 的误差范围情况下, 其整体准确率能达到 0.86.

Bettenburg 等人[35]研发了原型工具 quZilla 来评估缺陷报告问题详细描述的质量, 在评估时, 他们主要考虑了问题描述文本的可读性 (如列举和空行的使用)、特定关键词的存在情况 (如 reproduce、screenshot 等)、代码示例的存在与否等特征, 对相关指标进行打分最后得到问题描述文本的质量等级, 即非常好到非常差这 5 个等级, 在 100 个 Eclipse 的缺陷报告构成的数据集上进行测试 (每个缺陷报告由参与问卷的开发者进行评分确定其真实质量等级), quZilla 能达到 0.42 的准确度.

(2) 评论. Dit 等人[36]认为缺陷报告中的评论信息应当具有较好的内聚性 (讨论应围绕缺陷本身展开), 内聚性低的评论文本将影响其可读性和可理解性. 为了度量评论的内聚性, 作者首先使用潜在语义分析 (latent semantic analysis, LSA) 算法计算出相邻的两个评论之间的余弦相似性, 然后, 对这些余弦相似值求平均作为最终缺陷报告整体评论的内聚性. 该平均值越高, 内聚性越好, 越能帮助理解软件缺陷. 在评估方法的有效性时, 作者首先随机挑选了所计算得到的 Eclipse 中各 5 个内聚性最高和最低的缺陷报告, 让一个有经验的计算机专业的毕业生 (使用 Eclipse 编程 5 年、Bugzilla 1 年) 查看内聚性值与评论真实情况的一致性, 验证了方法的有效性.

(3) 复现步骤. 考虑到复现步骤对重现缺陷的重要作用, Chaparro 等人[13]研发了 EULER 工具来自动分析安卓应用中与 GUI 交互对应的复现步骤的质量问题 (如模棱两可的步骤、使用意想不到的词汇、步骤缺失等). EULER 工具由步骤匹配、步骤执行和推理、随机应用探索和质量评估这 4 个技术部分组成. 该工具将复现步骤的质量等级分为 HQ (high quality, 复现步骤完全与 App 的交互匹配)、LQ-AS (low-quality-ambiguous step, 复现步骤与多个 GUI 组件或者交互匹配)、LQ-VM (low-quality-vocabulary mismatch, 复现步骤与任何 APP 交互都不匹配)、MS (missing step, 对重现缺陷必要的步骤未出现在报告中). 作者发现在随机筛选出的 24 个缺陷报告 (来自 6 个安卓应用) 的 88 个复现步骤中, 73% 的复现步骤其质量等级标签均能预测准确.

4.2 间接质量度量

现有的关于缺陷报告质量间接度量的工作均是面向整个缺陷报告粒度. 通过对相关文献进行梳理, 研究人员主要使用缺陷报告处理代价/结果和揭示问题的可能性来间接反映缺陷报告的质量. 所使用的技术方法包括基于机器学习的方法和基于多目标优化技术的判定方法. 相关文献所涉及的特征包括缺陷报告自身的一些特征 (如报告文本可读性和完整性)、缺陷报告者声誉和缺陷追踪系统的一些特征 (如日负载) 等.

处理代价/结果: 基于质量高的缺陷报告通常比质量低的缺陷报告更快得到处理这一假设, Hooimeijer 等人通过预判一个软件缺陷报告处理代价的高低 (即从提交到最后关闭所花时间是否会超过某个阈值) 来度量其质量[37]. 预判模型为线性回归模型, 模型使用的特征包括缺陷报告的严重程度、可读性、日负载 (即缺陷报告提交前后 24 小时内提交的缺陷数)、缺陷报告者的声誉、给定时间内严重程度的更改情况、评论的数目和附件数. 作者发现, 用户自报的严重程度和报告提交早期 (一天) 所增加的评论数和附件数对模型的性能有较大影响. 在 27 984 个 Mozilla 的缺陷报告上实验发现, 不同处理代价阈值对应的预测精确度在 0.60 左右, F 值为 0.72–0.76.

与 Hooimeijer 等人[37]的工作类似, Gromova 等人[38]认为缺陷报告的问题描述文本质量差会直接导致如报告被拒 (rejected) 或处理时间更长等后果. 基于此, 他们提出通过预测缺陷报告的处理方案 (被拒或不会被修复的概率) 和处理时间来帮助 QA 工程师评估缺陷报告的质量. 预测模型以问题描述文本为输入, 其中描述中的每个单词使用 TF-IDF 的形式进行加权表示, 通过结合使用特征选择和不平衡类处理策略, 运行支持向量机得到预测模型. 在 5 242 个 JBOSS 的缺陷报告数据集上, 预测准确率在 0.68–0.86, F 值为 0.71–0.88.

揭示真实问题的可能性: 这一类工作主要针对众包测试中由众包工人所提交的缺陷报告 (也叫测试报告). 这类报告通常比较短, 含有截图, 且在短时间内被大量提交 (含有较多噪音). 为了帮助开发者从大量含噪的报告中筛选有价值的测试报告, 一些研究人员将测试报告是否揭示了真正的软件问题作为衡量该报告的重要质量标准. Wang 等人提出了一种基于聚类的分类方法[39]. 作者首先对测试报告使用 K-means 方法对测试报告进行聚类

(使用基于文本、情感和报告者经验特征的余弦相似度寻找最近邻). 给定一个新的测试报告, 计算与其相似的前 C 个簇, 基于每个簇训练一个分类器来预测该测试报告是否揭示了真实的软件缺陷 (1 为是, 0 为否), 最后通过投票统计的方法确定测试报告最终的类型. 作者共使用了百度众包平台的 35 个项目的 15 095 个测试报告进行技术评估, 发现预测精确度能达到 0.89, 召回率能达到 0.97.

Feng 等人通过对测试报告进行优先级排序从而将质量更高更可能揭示软件缺陷的众包测试报告返回给开发人员[40]. 他们提出了一种集合图像理解的多目标优化排序算法, 即使用 SPM (spatial pyramid matching) 技术衡量截图之间的相似性, 使用自然语言处理技术 (包括分词、词性标注、关键词提取) 对问题描述文本进行分析并使用 Jaccard 距离衡量文本之间的相似性, 最后对这两种相似性进行整合得到平衡距离 (balanced distance) 从而完成对所有测试报告的排序. 作者共使用了 5 个工业项目的 600 多个测试报告和 2 500 多张图片进行测试, 评估指标采用了 APFD (average percentage of faults detected) 和 linear interpolation (average number of inspected test reports), 作者发现基于图片理解的技术通常都能提升测试报告排序的效果.

此外, Feng 等人[41]还提出了两种排序策略: 第 1 种是危险性 (risk) 策略, 即根据报告中的关键词来预测最可能揭示问题的测试报告将其返回给开发者进行审查; 第 2 种策略是多样性 (diversity) 策略, 即每次选择那些与已审查过的测试报告海明距离最大的报告进行审查; 最后将两种策略进行组合以达到同时向多样性和最早揭示软件问题方面进行搜索的目的. 作者使用了百度公司的 3 个产品的 757 个测试报告进行实验, 评估指标同样使用了 APFD 和 linear interpolation, 作者发现其所提方法在某些项目上几乎能获得理论上的最优解, 在所有场景下比无序或随机排序的测试报告结果都要好.

4.3 小 结

关于缺陷报告质量问题的度量, 从现有的研究成果可以发现: (1) 在缺陷报告粒度对质量的直接或间接度量, 大多数工作都是将该问题转换成分类问题, 使用传统的机器学习算法如支持向量机、朴素贝叶斯、回归模型等构建预测模型, 模型的输入特征主要包括基于原始信息条目内容所抽取的一些统计特征如信息条目的完整性、报告者经验/声誉、自由文本特性 (如内聚性和可读性) 等; (2) 在缺陷报告具体信息条目粒度, 当前研究较少, 主要集中在非结构化文本条目 (如问题详细描述) 的质量分析上; (3) 在对缺陷报告或具体信息条目质量度量技术进行评估时, 已有工作在评估方法、评估指标的使用乃至计算方式上都存在较大差异, 这在一定程度上限制了我们在统一的评估框架内对各技术的效果进行直接比较和进一步的深入分析.

5 缺陷报告质量改进

通过对相关文献进行深入分析, 我们发现当前的缺陷报告质量改进研究主要集中在 3 个方面: (1) 对现有的缺陷报告系统或机制进行改进; (2) 自动生成缺陷报告或增强现有缺陷报告; (3) 自动预测缺陷报告具体信息条目的准确性或准确值. 表 5 概要展示了相关文献, 下面对这些工作进行详细介绍.

5.1 缺陷报告系统/机制的改进

如前所述, 现代的软件开发项目通常使用缺陷追踪系统来收集和管理缺陷报告, 这些系统或工具在功能设计方面的不足在一定程度上影响了缺陷报告的质量. 因此, 有些研究人员[7,14,15,43−54]提出对缺陷报告系统或机制进行改进, 改进方向可概括为 3 个方面.

(1) 降低用户收集和提供信息的负担. 这体现在改变信息收集方式 (如由用户提供到自动收集、由用户填写到用户选择)、只收集必须的信息条目、因地制宜 (如报告界面依报告者经验、缺陷类型的变化而变化) 和关注用户隐私.

(2) 引导用户. 在用户填写缺陷报告时及时向用户反馈并提供改进建议.

(3) 引入质量过滤机制. 培养有经验的缺陷报告者和过滤无实质内容的信息.

5.1.1 降低用户收集和提供信息的负担

(1) 改变信息收集方式. Zimmermann 等人[7]建议缺陷追踪系统应努力减轻用户收集和提供缺陷信息的负担,

可采取的措施包括将一些相关的堆栈信息自动识别并添加到报告中、截屏留存观测行为信息等. Ko 等人[15]以 5 个开源项目近 20 万个缺陷报告的问题概要描述为数据来源, 从语言学的角度 (名词、动词等的使用场景及模式) 分析了人们通常是如何描述软件问题的. 作者发现缺陷报告的问题概要描述通常都包含了软件实体/行为、其某方面质量特性的不足及问题出现的执行上下文几方面; 95% 的名词短语指向的都是可见的软件实体、物理设备或者用户动作. 基于上述发现, 作者建议使用更结构化的缺陷报告模板来迎合用户描述软件问题的习惯, 如将问题概要描述分成实体、问题、执行上下文等几部分, 每部分可提供一些常见的候选词供用户选择. Just 等人[14]基于在 Apache、Eclipse、Mozilla 项目的问卷调研开发者在修复缺陷时所需信息和遇见的常见问题的调研结果, 指出新一代缺陷追踪系统应帮助用户自动收集和报告关键信息 (如复现步骤和测试用例等).

表 5 缺陷报告质量改进相关文献

类型	改进方向	具体策略	文献	成果类型
改进缺陷报告系统/机制	降低用户收集和提供信息的负担	① 改变信息收集方式 ② 精简信息条目 ③ 定制化报告界面 ④ 关注用户隐私 ⑤ 其他	[7,14,15,43–50]	建议为主, 原型工具较少
	引导用户	① 及时反馈 ② 及时推荐	[6,7,10,14,51–53]	
	引进质量控制机制	① 培养报告者 ② 内容过滤	[14,44,54]	
自动生成/增强缺陷报告	自动生成	① 生成整个报告 ② 生成单个条目 (如复现步骤)	[16,55–57]	原型工具技术框架
	自动扩充/增强	① 扩充额外信息 (如隐私) ② 增强已有条目 (如问题描述)	[18,58–62]	原型工具
自动预测缺陷报告具体条目	是否准确/具体值	① 预测多个条目 ② 预测单个条目	[17,63–102,103–129]	机器学习主导的技术框架

(2) 精简报告模板. Dal Sasso 等人[45]提出应该定义一个最小的缺陷报告内容基准, 该基准既不会给报告者带来较大负担, 同时又能提供缺陷修复所必需的信息. 作者收集了来自 Apache (使用 JIRA) 和 Mozilla (使用 Bugzilla) 的 65 万个缺陷报告, 分析了这些报告的所有信息条目对缺陷修复时间的影响, 其中报告者、被指派修复缺陷的开发者、问题概要描述、问题详细描述、报告处理截止时间 (due date) 的影响最大, 作者建议应充分提供这些条目. Herraiz 等人[46]提出 Eclipse 的缺陷报告模板要求报告者提供的信息条目太多, 有些条目的选项难以区分, 建议软件项目使用更简单的缺陷报告模板以减少用户在提交缺陷报告时的困惑. 在文中, 作者以 Eclipse 缺陷报告的严重程度和优先级条目为例, 通过分析发现严重程度等级只需要分成 3 个而不是 7 个, 优先级等级也只需分成 3 个而不是 5 个.

(3) 因地制宜. Just 等人[14]指出一个软件的不同用户的知识水平往往不相同, 有经验的缺陷报告者可能知道错误日志的存在从而会提供堆栈信息等, 而新手可能只会报告奇怪的难以复现的软件行为描述. 作者建议针对不同级别的报告者, 缺陷追踪系统应提供不同的缺陷报告操作界面. Nichols 等人[47,48]和 Yusop 等人[50]指出传统的缺陷追踪系统如 Bugzilla 都主要为报告软件功能问题为中心, 报告条目选项太多, 对提交可用性问题的用户而言并不友好. 作者认为缺陷追踪系统在支持可用性缺陷报告方面应改进其易用性, 更好地支持用户交互的描述和多类型附件的管理, 提供更多基于元数据的冗余缺陷检测. Breu 等人[43]发现开发者和用户在缺陷的不同生命周期对信息的需求也不相同. 开发者在前期的关注点集中在信息的缺失及信息细节方面 (对调式具有重要影响), 为此作者提倡缺陷追踪系统提供更简便的方式来提供截图、堆栈信息和复现步骤等关键信息. 一些开源项目的维护者向 GitHub 写了一封公开信[44], 提到在 GitHub 中用户提交的缺陷报告 (也叫问题单) 往往缺少如复现步骤或所测软件版本等关键信息, 希望 GitHub 能支持开发者定制问题单的信息条目, 提供能让用户将必要信息都报告在问题单里

的机制.

(4) 关注用户隐私. Nichols 等人[47,48]指出可用性缺陷 (usability bugs) 相对于功能性缺陷 (functional bugs) 的复杂性更高、主观性更强、需要更加广泛的用户参与度. 为了能让更多的用户参与可用性缺陷的报告, 作者认为缺陷追踪系统在收集系统数据时应注重保护用户的隐私. 为了解决因触发缺陷的输入涉及用户隐私导致输入信息未提供或报告未提交的问题, Castro 等人[49]提出基于原始输入产生新的较少暴露用户隐私的触发缺陷的输入 (依靠符号执行和 SMT 求解器技术), 同时向用户反馈当前报告暴露了多少用户信息从而让用户决定是否提交报告的方案, 达到促进用户提交包含触发缺陷输入的报告, 同时保护用户隐私的目的.

5.1.2 引导用户

Just 等人[14]认为新一代缺陷追踪系统在设计时应为用户提供上下文协助、提醒用户在报告缺陷时添加相关信息. Zimmermann 等人[7]建议及时对用户提交的信息进行反馈 (如是否完整正确), 同时启发用户和开发者收集哪些信息以及如何收集以提交高质量的报告. 他们开发了原型工具 CUEZILLA[6,10], 该工具能自动分析缺陷报告的质量, 并能为缺陷报告者提供如何改进缺陷报告质量使得缺陷能被更快更好修复的建议. Song 等人[51]研发了观测行为 (observed behavior, OB)、期望行为 (expected behavior, EB) 和复现步骤 (steps to reproduce, S2R) 的检测工具 BEE, 该工具能将问题详细描述中属于 OB、EB 和 S2R 的文本句子标识出来, 并且自动检测这些条目是否缺失. 依据 OB、EB 和 S2R 的标识和检测结果, 缺陷报告者可以对缺失的条目进行进一步地完善和补充. 目前该工具已与 GitHub 集成并提供使用接口. 类似地, Chaparro 等人[52]通过人工分析 2 912 个缺陷报告总结了 154 个 OB、EB 和 S2R 的出现模式, 并基于出现模式提出了 3 种自动检测缺陷报告是否包含 EB 和 S2R 的方法. 预测结果将帮助用户提交信息更完整的缺陷报告.

通过人工分析缺陷报告数据, Karim 等人[53]发现复现步骤、测试用例、代码示例、堆栈信息和期望行为等关键条目经常被用户忽略或未在报告中提供. 为了帮助用户提交质量更高的缺陷报告, 作者基于报告的问题概要描述信息, 训练了多个机器学习模型 (包括 NB、KNN、SVM 等), 预测用户是否需要在新提交的报告中额外提供这些条目.

5.1.3 引入质量过滤机制

为了解决缺陷报告质量低的问题, Lotofu 等人[54]提议向缺陷追踪系统引入 Stack Overflow 中利用声誉和奖励系统来鼓励可取行为的机制. 通过该机制, 让缺陷追踪系统一方面可以有效筛选有用贡献 (如质量更高的缺陷报告), 另一方面也能鼓励更多参与者更频繁参与到软件项目中, 对示范行为的正面反馈和参与者对更高的声誉和奖励 (如一些特权) 的追求将促使他们贡献质量更高的缺陷报告.

类似地, Just 等人[14]也建议对那些提交了良好缺陷报告 (如提交了 critical security bugs) 的报告者进行奖励, 同时, 将报告者的声誉集成进缺陷追踪系统以逐渐培养/识别出有经验的缺陷报告者. 作者提到有经验的缺陷报告者往往有较好的缺陷报告提交历史 (如所提交的缺陷大多被开发者认可并修复), 其所提交的缺陷报告也往往被认为质量更高, 进而被开发者更加认真地对待和处理. 在缺陷报告系统中有目的地对缺陷报告者进行引导培养并识别出有经验的开发者, 将帮助获得和筛选出质量更高的缺陷报告.

一些开源项目的维护者希望 GitHub 能对缺乏实质性的内容 (如点赞评论) 进行较好的管理[44]. 他们指出, 类似“+1”“thumbup”或“me too”等评论尽管能让维护者知道一个缺陷问题的普遍性, 但对维护者和其他订阅了该缺陷问题的其他人来说更像是一个垃圾消息. 他们建议 GitHub 的问题单能有一个投票机制, 将上述“+1”等信息进行投票整合, 而不是让其散落在缺陷报告中成为垃圾信息, 这也将更好地提醒软件项目人员进行相关处理.

5.2 自动生成或增强缺陷报告

第 5.1 节主要从改进已有缺陷报告系统/机制的角度描述了研究人员开展的一些典型工作 (以改进建议为主), 改进缺陷报告质量的另一个重要分支为通过特定的技术和方法自动生成整个缺陷报告或通过改进或添加关键信息条目来增强已有缺陷报告的质量[16,18,55−62]. 这部分的相关工作大多已有原型工具, 其中, 缺陷报告自动生成目前针对的主要是安卓应用或者众包测试应用场景, 而缺陷报告自动增强主要增强的信息条目为复现步骤、执行轨迹

和问题描述等部分. 现对这两方面的相关典型工作介绍如下.

5.2.1 自动生成缺陷报告

Moran 等人[16,55]开发了原型工具 FUSION 来帮助应用开发者和 beta 测试用户生成一个可操作的复现步骤完整的缺陷报告 (目前仅限于安卓应用中基于 GUI 的功能性缺陷). 具体地, 在软件被测试或发布前, FUSION 首先通过静动态分析技术收集 GUI 组件相关的信息 (如 actions、types) 和事件流 (event flow) 等. 在用户报告缺陷时, FUSION 可以提供相关界面向用户展示当前可以进行的操作、相对应的 GUI 组件及相关的截图. 用户基于 FUSION 界面的展示选择相应的操作, 基于用户的已有选择和交互, FUSION 进一步在线预测后续相关步骤并显示相关信息. 基于多次交互, 用户最终可以得到一个复现步骤完整的缺陷报告.

Moran 等人[56]还研发了针对安卓应用的 CRASHSCOPE 工具, 该工具首先通过系统化的输入机制 (按照不同静动态分析产生的策略) 来触发应用崩溃 (APP crash), 然后生成一个包含截图、详细重现步骤和异常堆栈信息的缺陷报告以及对应的可回放脚本.

针对众包测试中众包工人能较容易提供包含缺陷的截图但较难书写专业的缺陷报告这一问题, Yu[57]利用图像理解技术开发了原型工具 CroReg, 该工具能依据用户提交的截图直接生成测试报告. 具体地, 作者一方面使用 OCR 技术从截图中识别其中的文本, 另一方面使用 im2text 技术将截图直接转成文本, 最后通过计算二者的相似性生成测试报告.

5.2.2 自动增强缺陷报告

一些软件缺陷源自用户与软件产品 GUI 组件的一系列交互, 而用户通常难以在报告中准确提供全部交互操作, 导致所报告的缺陷难以复现. 为了解决这一问题, Herbold 等人[58]提出了一种可以集成到软件产品内部监控用户操作 (monitor user action) 的机制, 该机制可以将真实的用户操作轨迹进行记录并抽取到报告中, 作者还设计了一种基于用户的操作轨迹机制, 该机制可以将用户的 GUI 操作行为进行重放.

White 等人[59]研发了针对安卓应用的工具 CRASHDROID, 该工具能从崩溃报告 (crash report) 的函数调用栈中自动生成较易理解的复现步骤 (expressive S2R), 同时提供缺陷回放所需的内核事件跟踪轨迹 (a kernel event trace).

针对众包测试报告的高冗余、文字少、截图多的特点, Hao 等人[18]提出利用冗余的测试报告来生成内容更丰富的增强版缺陷报告. 作者研发了工具 CTRAS, 该工具首先根据报告中的描述文本和图片将冗余的缺陷报告进行分组, 选取组里面最有信息量的 (informative) 报告作为主报告 (master report), 随后将剩余冗余报告中对缺陷理解和修复有补充作用的信息融入到主报告中, 以此得到质量更高的众测报告.

为了解决一些缺陷报告的问题描述比较短所含信息有限的问题, Zhang 等人[61]提出为其添加额外文本句子以扩充其内容的方法. 待添加的句子取自历史报告中的问题描述内容, 作者首先计算了候选句子和当前待扩充报告的文本、主题、组件/产品等 6 个相似性特征的值, 通过对 6 个特征值进行加权求和得到最终的相似性值, 值最大的前 K 个句子用来扩充短文本报告.

依托模型校验技术和已有的安全方针信息, Weimer[62]则试图对一些违反安全方针 (safety policy violations) 的缺陷报告自动生成一个补丁, 该补丁揭示了修改代码方式, 以避免违反相关的安全方针.

Zamfir 等人[60]指出系统出现并行 bug 时的系统信息与出现在缺陷报告中的信息是存在鸿沟的, 为此, 作者提出了一个低开销的 DCop 系统, 该系统能收集部分与死锁 bug 出现非常相关的运行时信息 (如函数调用栈和持有的互斥锁) 并用其扩充已有的缺陷报告.

5.3 预测缺陷报告的具体条目

如第 3 节所述, 缺陷报告存在的一个主要质量问题是具体信息条目经常存在缺失或错误, 针对这一问题, 研究人员开展了一系列工作[17,63–129], 尝试通过预测信息条目的值或准确性来改善缺陷报告的质量, 且目前所预测的条目主要为结构化的信息条目如优先级、所属组件等 (这些条目的取值通常为预定义的离散分类值). 通过调研, 我们发现现有预测具体信息条目值的工作通常都是将预测任务视为一个机器学习分类任务 (极少数工作使用了其他

方法), 流程为以历史数据为依托, 通过预处理构造数据集, 抽取数据特征, 对比测试不同机器学习算法的预测效果, 同时解决模型训练过程中遇到的一些问题 (如类实例不平衡等). 而在被预测的信息条目中, 缺陷报告所属组件 (component)、严重程度 (severity) 和优先级 (priority) 是重点被预测的条目. 这在一定程度上是因为这些条目的值对缺陷报告的处理紧急程度和由谁 (团队) 来修复该缺陷起着较重要的作用, 同时这些条目也被发现经常会存在错误[9]. 考虑到相关的研究遵循类似的研究思路, 本文首先概括介绍关于组件、严重程度和优先级这 3 个条目的预测研究. 然后是对其他信息条目预测的工作介绍.

5.3.1 缺陷报告所属组件、优先级和严重程度预测

如前文所述, 对组件、优先级和严重程度的预测工作遵循类似的流程, 以机器学习方法应用为主 (个别使用了其他技术), 通过从缺陷报告抽取各种特征得到数据实例, 构建模型以完成预测任务. 在模型构建过程中, 针对特定环节存在的问题再进行处理, 改进模型的预测效果. 下面从特征抽取、技术应用和特定环节改进这 3 个方面进行介绍.

(1) 特征抽取

在预测组件、优先级和严重程度的任务中, 所抽取的特征通常来自缺陷报告本身, 但在特征使用上略有侧重. 在预测组件时, 大多数研究仅使用了缺陷报告的问题概要描述或/和问题详细描述[63−67], 另外一些工作则在问题概要描述/问题详细描述的基础上添加了其他信息包括历史缺陷报告的评论信息[68]、结构化信息 (如操作系统、平台)[69,70]、堆栈信息[71]. 在预测优先级时, 单独使用缺陷报告问题概要描述和问题详细描述特征条目的工作较少[72,73,105,106], 大多数对比或使用了多种特征提高优先级预测的效果, 这些特征包括缺陷报告文本信息 (问题概要描述和问题详细描述)、结构化信息、缺陷报告者信息、时间相关性信息、相似缺陷报告特征等[74−81,103,104,107]. 在预测严重程度时, 有较多研究所使用的信息条目仍仅为问题概要描述或/和问题详细描述[17,82,84−89], 其他典型工作则额外探究了堆栈信息[71]、产品组件信息[90,114]、情感信息[91,92]、系统设计知识[109]、技术问答网站如 Stack Overflow 上的相关讨论[93]在严重程度预测任务上的效用.

对于问题概要描述和问题详细描述这样的文本条目, 通常需要进行如分词、去停用词、词性还原或去词干等预处理步骤, 随后使用相关模型对其文本语义进行表征. 在已有调研文献中, 对组件进行预测的任务通常采取 TF-IDF 词特征向量和 LDA 主题分布来表征文本语义[63,64,66,67]; 对优先级进行预测的任务通常采取 TF/TF-IDF 特征向量、LDA 主题分布、词嵌入向量表示[72−74,76,77,80,81,104−107]; 对严重程度进行预测的任务采取的文本语义表征方式包括 TF/TF-IDF 特征向量、LDA 主题分布、N-gram 和词嵌入/文档向量[17,82,84−86,88,89,92,93,111,129].

(2) 技术应用

基于机器学习算法: 尽管对缺陷报告所属组件、优先级和严重程度的预测基本都是基于机器学习算法, 这 3 类在机器学习算法方面的应用现状却并不相同. 对组件的预测, 现有的工作基本侧重在对传统的机器学习方法的使用 (如 SVM、NB、KNN 等)[63,66−68,70,71]; 而优先级的预测任务则展示了由传统的机器学习方法 (如 NB、决策树) 到神经网络再到当前流行的深度学习技术 (如 CNN、DNN) 的使用轨迹[72−75,79,103−107]; 对严重程度的预测任务, 现有的研究仍然以传统机器学习为主[17,82,84−86,90−91,93,108−110,112,113,116−127], 也存在一些使用深度学习技术进行预测的研究[92,94,111,115].

基于其他技术: 除了主流的基于机器学习算法的预测模型, 也存在一些使用其他技术 (如信息检索技术) 进行具体条目的预测. Somasundaram 等人[66]比较了 3 种组件预测的分类算法, 即 SVM-LDA, SVM-TFIDF, LDA-KL, 三者使用的数据均为 Eclipse 缺陷报告的问题详细描述文本. 前两个的分类器是 SVM, 实例特征为问题详细描述对应的主题概率分布和单词的 TF-IDF 权重向量, LDA-KL 算法是基于新缺陷报告的主题概率分布与各组件下历史缺陷报告的平均主题概率分布的 KL 距离来进行组件分类. 作者发现 LDA-KL 方法与已有方法相比效果相当, 同时在类别不平衡时表现更稳定.

Yan 等人[67]提出对问题详细描述使用一种新的主题模型预测报告所属组件的方法, 即 DPLSA-JS (discriminative probability latent semantic analysis and Jensen-Shannon divergence), 作者在 5 个 OSS 项目上进行实验, 发现其效果优于前文[66]的 LDA-KL 和 SVM-LDA 方法.

Chawla 等人[83]提出一种基于模糊 (fuzzy) 相似性的方法来预测缺陷报告中的组件值. 具体地, 作者首先将历史缺陷报告按照组件分组并计算问题概要描述和详细描述中每个单词与每个组件的相关性 (即单词出现在某个组件的次数除以出现在所有组件的总次数). 对于一个新的缺陷报告, 抽取出问题概要描述和详细描述中的每个单词及其与各组件的相关性, 然后求平均相关性算出该缺陷报告属于某个组件的可能性, 与某个组件的相关性值越大被认为越可能属于该组件.

Tian 等人[128]使用基于信息检索的方法预测缺陷报告的严重程度. 该方法主要包括两个组件, 即相似度计算和类标签分配. 在相似度计算模块, 他们使用扩展的 BM25 算法计算两个缺陷报告的相似度. 在类标签分配模块, 通过计算得到最相近的 K 个缺陷报告的相似度及其加权和得到新缺陷报告的严重程度.

(3) 特定环节改进

在构建机器学习模型预测具体条目值的流程中, 由于数据集本身特点 (如类实例分布不均衡) 或者所用数据处理方法特性 (如向量空间模型表征产生的高维特征空间) 的影响, 使得训练所得到的预测模型有时不能取得较好的预测效果. 因此, 一些研究人员就模型构建的具体环节所存在的一些问题进行处理, 从而改进相应模型的性能. 通过对现有的文献调研发现, 研究人员主要针对实例特征空间庞大、用于训练的标签实例数不足、类实例分布不均衡、原始数据集存在噪音这些问题进行了处理. 具体介绍如下.

庞大的特征维度: 在所调研文献中, Iqbal 等人[95]关注和解决了优先级预测过程中, 使用 TF 或 TF-IDF 向量表征问题概要描述和详细描述时带来的特征维度急剧膨胀的问题, 并提出使用矩阵分解和主成分分析等技术对构造的特征进行约简和选择, 在约减的特征基础上综合应用聚类和传统机器学习技术 (包括 SVM、NB) 以获得更好的预测效果. 类似地, 在进行严重程度预测的过程中, 现有工作对缺陷报告的问题概要描述和详细描述使用 TF-IDF、N-gram 表征文本语义时同样会遭遇语义向量维数庞大影响模型预测效果的问题. 为了解决该问题, 研究人员主要使用了特征选择方法来达到降维和选择重要特征的目的, 通常使用的特征选择方法包括信息增益、卡方检验、相关系数等[85−89].

标签实例数不足的处理: Zhang 等人[96]认为现有的缺陷报告严重程度预测方法依赖于大量已标注的训练数据, 在缺少带标签的数据的情况下, 现有方法效果并不好. 为了解决这个问题, 他们提出了一种增量学习方法, 即使用主动学习技术标注没有标签的缺陷报告, 同时使用迁移学习来获得足够多的训练数据, 如此可以生成大量的有标签数据构建缺陷报告严重程度预测模型. 而在预测优先级任务上, 鉴于一些项目历史数据不足和项目领域的相似性, Sharma 等人[80,81]提出在跨项目的场景中进行缺陷报告优先级预测, 即基于一个项目的历史缺陷数据训练模型, 预测另一个项目的缺陷报告优先级.

不平衡类问题的处理: 一些研究人员[97−99]发现训练数据中不同严重程度 (即实例类别) 所包含的实例数分布不平衡, 导致模型预测效果不佳. 为了解决这一问题, Roy 等人[97]引入代价敏感学习技术, 使用代价矩阵给予不同的严重程度不同的权重. Guo 等人[98]和 Hamdy 等人[99]则提出使用过采样技术 SMOTE 来解决严重程度类实例数不平衡的问题. 类似地, 在组件预测的过程中, 考虑到不同组件的实例数不同且有的报告组件的值存在混合标签的情况 (如 platform 和 platform:xxx), 为了较真实准确地进行预测, Zhang 等人[64]使用多层神经网络来预测缺陷报告组件条目的值. 作者首先基于缺陷报告时效性 (recency, 从缺陷报告被用户提交到被用于模型训练的时间差) 对训练实例进行抽样解决类实例不平衡问题. 然后, 提出一个分层的训练框架, 即分别训练组件 (super-component, 如 platform) 和子组件 (sub-component, 如 platform:xxx) 分类器, 并对两个分类器的结果进行优化, 最终对缺陷报告的组件值进行预测.

数据集质量: Tian 等人[100]发现 51% 的重复缺陷报告尽管报告了相同的缺陷, 其所报告的严重程度却不相同, 因此使用人工标注的数据集来进行缺陷报告严重程度预测是不可靠的. 作者认为尽管认定缺陷报告严重程度是一个主观活动, 开发者在填写严重程度时也应该尽量统一标准、尽量准确; 而研究者在进行相关研究时, 也应该充分考虑缺陷报告严重程度不可靠的现象.

5.3.2 其他信息条目的预测

Wu 等人[101]研发了原型工具 BUGMINER, 该工具使用历史缺陷报告数据训练了一个线性 SVM 分类器, 该分

类器可以根据缺陷报告中其他信息条目的值预测某个缺失条目的值, 以此达到补全缺陷报告信息的目的. Xia 等人[102]提出了一种考虑了类不平衡问题的多标签学习算法 (imbalanced ML.KNN), 用于预测缺陷报告中一些信息条目 (包括组件、产品、优先级、严重程度等 8 个信息条目) 的值是否会被更改或者需要完善. 作者首先基于元特征 (meta-feature, 如组件、产品、平台)、文本特征 (问题概要描述和详细描述)、混合特征 (前两者) 训练 3 个 ML.KNN 分类器, 然后使用不同的权重将 3 个分类器的预测结果进行加权求和得到最终的预测结果.

5.4 其 他

除了上述工作, 有些实践者或研究者在缺陷报告质量保障方面亦给出了一些有建设性的洞见或者做了一些实践尝试. Tatham (知名开源免费软件开发者, 开发了如 Putty、SSH、term 终端等)[130]给缺陷报告提交者提供了一些如何有效报告缺陷的建议, 包括如何向开发者展示复现问题、记录错误消息中的数字、详细描述所见等. 为了提高缺陷报告评论的可读性, Dit 等人[131]提出当前评论应尽量回应该评论之前的评论且尽量只涉及到一个主题, 这样可以提高缺陷讨论的可读性. 为此, 作者开发了一套推荐系统, 为给定评论自动推荐相关的其他评论.

5.5 小 结

围绕缺陷报告质量改进问题, 我们通过对现有研究进行总结发现: (1) 在缺陷报告系统或机制改进方面, 现有研究主要集中在改进用户信息收集过程、及时引导用户和引入质量控制机制这 3 个方面 (以建议为主, 工具支持方面较缺乏); (2) 在缺陷报告自动生成或增强方面, 缺陷报告自动生成目前针对的主要是安卓应用或者众包测试应用场景, 缺陷报告自动增强主要增强的信息条目为复现步骤、执行轨迹和问题描述等部分, 目前已存在一些原型工具; (3) 在缺陷报告具体信息条目预测方面, 当前研究主要集中在对缺陷报告所属组件、严重程度和优先级条目准确值的预测上, 普遍使用的方法为基于缺陷报告数据 (包括利用 TF-IDF/LDA 等模型抽取问题概要/详细描述的语义特征和一些结构化信息条目等), 应用机器学习技术进行建模预测, 部分工作对数据集噪音、特征空间庞大和类不平衡问题进行了处理.

6 研究挑战与趋势

从现有文献来看, 围绕缺陷报告质量的刻画、度量和改进已经取得了一系列研究成果. 同时, 我们也应当意识到, 目前该研究还处于发展阶段, 仍存在一些亟待解决的问题和挑战, 如工具链支撑不足、应用场景单一、领域知识表达不足、基准数据集缺乏等, 对这些问题的探索和解决将成为未来的研究方向.

(1) 面向多领域多类型的缺陷报告质量问题分析

在调研缺陷报告存在哪些质量问题时, 已有研究的调研对象主要来自几个主流开源项目 (如 Eclipse、Mozilla 等) 和工业软件项目的开发者或缺陷报告数据, 分析的缺陷报告类型也基本是通常意义上的缺陷报告, 针对专门类型的缺陷报告 (如可用性缺陷报告) 研究非常少. 研究领域和缺陷报告类型的有限性在很大程度上影响了我们对缺陷报告质量问题的普遍性和严重性的认知. 在软件与各领域应用 (如工业制造) 进行融合发展的未来趋势下, 不同细分领域 (如航空航天) 所对应的不同缺陷类型 (如安全性缺陷) 的缺陷报告存在什么质量问题及差异性如何, 对软件质量的保障将变得尤为重要, 可能成为未来需要重点研究的主题.

为了有效开展此类研究工作, 研究人员一方面可以利用当前在开源项目或个别工业项目上的研究成果 (包括研究发现及其研究手段如问卷访谈或仓库挖掘等) 作为调研基础, 另一方面, 在缺陷数据集获取和质量分析过程中, 有必要进一步加强与工业界的沟通, 通过深度合作, 充分挖掘缺陷报告存在的真实质量问题, 促进研究成果在软件开发实践中的落地应用.

(2) 质量评估基准数据集的构建

自动化缺陷报告质量评估在发现质量问题和改进质量问题闭环中起着承上启下的重要作用. 只有真实准确地度量出缺陷报告的质量, 软件开发人员和用户才能有的放矢, 改善缺陷报告质量相关的整个过程. 在自动化评估缺陷报告质量这一环节, 尤其是针对缺陷报告质量的直接度量 (即对质量进行打分 (如 1–5 分) 或划分等级 (如好、中、差), 度量结果用于反映开发者对缺陷报告质量的判断, 即对他/她们理解、定位、修复缺陷的辅助程度), 拥有

一个每个缺陷报告的质量评估结果均真实可靠的黄金基准数据集 (a golden benchmark) 至关重要, 因为该基准数据集是评估并改进缺陷报告质量评估技术的前提.

然而, 通过文献调研, 我们发现, 已有研究通常使用的是工业界或人为标注的开源项目的缺陷报告数据集 (规模较小, 大多为几百个缺陷报告), 且基本不存在数据集共享的情况. 为了促进该研究领域的发展, 未来需要相关研究人员和软件项目人员共同努力构建并提供一个可以公开使用的、具有一定规模的、面向不同领域 (尤其是那些蓬勃发展的领域如智能系统) 的、用于质量评估的基准数据集. 在构建数据集的过程中, 缺陷报告真实质量的标注通常需要软件开发或维护人员的协作, 因此标注成本较高, 如何利用已有技术降低标注成本, 是构建数据集的潜在方向. 未来研究可能可以从以下两方面进行探索: 其一, 探索在数据标注环节引入基于专家知识引导的众包机制的可能性. 众包有助于解决多领域数据标注问题, 而将从各个领域总结的评判缺陷报告质量好坏的启发式规则当作专家知识, 可以引导经验相对缺乏的众包工人更好的标注数据. 其二, 探索基于少量标注数据的半监督或弱监督方法的潜在应用价值. 半监督和弱监督是一类较为经典的处理标记数据不足的解决方案, 其有望在只要求软件开发维护者标注少量缺陷报告的前提下, 对剩余未标注报告进行较好的质量标注.

(3) 质量评估指标体系的建立

在评估缺陷报告质量时, 现有研究通常使用的一些指标包括关键信息条目的存在情况、文本内容的可读性、报告者声誉等. 这些指标的选取大多来自研究人员自身的经验且在度量时较少考虑领域知识, 计算方式相对简单. 指标选取及度量的合理性并未在真实应用场景得到验证. 在现实生活中, 软件开发维护人员在判定缺陷报告质量时通常会考虑哪些方面, 这些方面具体应该用哪些指标进行度量 (相关方面及指标的收集可能需要使用如访谈问卷的定性方法), 各指标的相对重要性如何 (一些特征选择或相关性分析的统计方法可能可以适用), 以及如何整合这些指标 (如可以尝试使用一些参数调制的方法或者直接使用一些机器学习或深度学习的技术). 未来研究可能要从回答上述问题出发, 基于调研和已有研究成果建立合理的质量评估指标体系, 以确保质量度量结果的有效性和可参考性, 为后续质量的改进提供指导性意见.

(4) 可定制化的增强版缺陷报告系统

在缺陷报告质量改进方面, 现有研究通过如问卷调查等定性分析手段, 总结出了一系列改进报告质量的建设性意见, 其中, 针对缺陷报告系统或机制的改进是其中重要的研究内容. 围绕缺陷报告系统的改进, 软件开发者提到的主要问题是已有缺陷报告系统在特定类型的缺陷报告方面支持较差 (与该类型缺陷相关的关键信息很难收集和管理)、缺陷报告模板难以适配具体项目和应用场景、质量控制机制缺失等. 对于这些问题, 通过调研我们发现, 目前仍未有实质性的进展和突破, 很多改进建议也只停留在建议层面. 作为连接用户和软件项目的重要媒介以及管理软件项目问题的重要工具, 切实推进对缺陷报告系统功能的改进和完善具有重要的现实意义. 例如, 未来研究可以首先尝试收集不同用户场景下特定类型缺陷报告的信息条目需求, 基于此整理出领域内关键信息条目的共性需求, 并进一步着手研发相应的报告支持机制或系统, 感知用户的工作环境上下文, 自动提供定制化的报告模板.

(5) 面向关键信息条目收集的工具支撑

根据本文第 3 节关于缺陷报告存在的主要问题的介绍, 我们可以发现无论是在工业界还是开源社区, 面向的是通用性缺陷报告还是特定类型的缺陷报告, 对缺陷定位和修复最为有用的一些关键信息条目, 包括复现步骤、观测行为、测试用例、堆栈信息等, 普遍存在信息缺失或错误的问题[6,23,26]. 现有研究[10]已经表明, 这些信息的提供对用户来说并非易事. 因此, 针对这些关键信息条目研发专门的工具来辅助用户收集或自动为用户收集具有重要的意义.

通过文献调研我们发现, 一些研究人员在该方向上已做了一些探索性工作并研发了相关原型工具, 包括如识别并判断关键信息条目的缺失情况[51,52]或对其进行质量评估[36]、提醒/推荐用户添加这些信息[6]等. 然而, 这些工作只能起到有限的辅助性作用, 提交较准确完整的关键条目信息的主要工作仍需用户完成. 此外, 尽管存在一些自动化收集复现步骤、执行轨迹等条目的工具, 如文献 [58,59], 这些零星工具目前只针对如安卓应用等特定的场景, 离应用于真实的软件开发场景并真正发挥作用仍有较大距离. 如何针对各种应用场景, 针对不同类型的缺陷, 研发成熟的自动 (或有效辅助) 收集缺陷报告关键信息条目的工具, 在未来软件定义一切、软件无所不在的大背景下,

将变得越来越重要且越来越具有紧迫性.

(6) 基于领域知识融合和文本语义表征增强的缺陷条目预测技术

作为缺陷报告质量改进的一个重要方向, 缺陷报告具体条目准确值 (主要是预定义条目如组件、优先级等) 的预测在近些年获得了较多关注, 涌现出一批研究成果. 然而, 这些技术成果尚未在真实软件项目开发中得到较好应用, 其预测结果的准确性和可解释性有待进一步提高和完善. 例如, 在条目预测研究中, 基于机器学习的预测模型是最主要的一类解决方案, 其依赖的训练实例特征主要包括缺陷报告问题概要描述和详细描述、当前缺陷报告负载、报告者经验等.

通过文献分析, 我们发现现有研究在抽取相关实例特征时, 往往忽略了对领域知识的表达 (如不同项目对优先级评定所定义的规则、优先级和严重程度条目具有动态变化的特性等). 同时, 对最主要的特征来源, 即缺陷报告问题概要描述和详细描述, 已有文献大多使用传统的如基于 TF/TF-IDF 向量空间模型、N-gram 或 LDA 主题分布来表征其文本语义, 少数研究考虑了词嵌入技术来抽取语义特征. 这些模型忽略了对文本上下文语义的抽取或者未考虑缺陷报告中各软件实体所代表的领域知识, 导致对问题描述文本所表达的问题语义表征不足. 增强对领域知识的融合 (如通过挖掘项目说明、需求规约等) 以及对问题描述文本问题语义的抽取 (如通过构建缺陷知识图谱), 是未来改进该类技术的一个潜在研究方向.

7 结束语

缺陷报告质量问题一直是软件工程领域的研究热点. 本文围绕缺陷报告质量问题刻画、度量和改进这 3 大方面对当前的研究工作进行了系统性梳理和总结. 基于当前的研究进展, 总结了该领域面临的主要挑战并讨论了未来潜在的研究方向. 本文主要工作总结如下: (1) 整理了缺陷报告存在的主要质量问题; (2) 总结了缺陷报告质量度量的技术和方法; (3) 梳理了缺陷报告质量改进的各类策略和方法. (4) 最后, 分析了该领域研究存在的问题和挑战并展望了未来的研究方向.

References:

[1] Brooks Jr FP. The Mythical Man-month: Essays on Software Engineering (20th Anniversary Edition). Boston: Addison-Wesley Professional, 1995.

[2] Zou WQ, Lo D, Chen ZY, Xia X, Feng Y, Xu BW. How practitioners perceive automated bug report management techniques. IEEE Trans. on Software Engineering, 2020, 46(8): 836–862. [doi: 10.1109/TSE.2018.2870414]

[3] Joorabchi ME, Mirzaaghaei M, Mesbah A. Works for me! Characterizing non-reproducible bug reports. In: Proc. of the 11th Working Conf. on Mining Software Repositories. Hyderabad: ACM, 2014. 62–71. [doi: 10.1145/2597073.2597098]

[4] Serrano N, Ciordia I. Bugzilla, ITracker, and other bug trackers. IEEE Software, 2005, 22(2): 11–13. [doi: 10.1109/MS.2005.32]

[5] Xuan JF, Jiang H, Hu Y, Ren ZL, Zou WQ, Luo ZX, Wu XD. Towards effective bug triage with software data reduction techniques. IEEE Trans. on Knowledge and Data Engineering, 2015, 27(1): 264–280. [doi: 10.1109/TKDE.2014.2324590]

[6] Bettenburg N, Just S, Schröter A, Weiss C, Premraj R, Zimmermann T. What makes a good bug report? In: Proc. of the 16th Int'l Symp. on Foundations of Software Engineering. Atlanta Georgia: ACM, 2008. 308–318. [doi: 10.1145/1453101.1453146]

[7] Zimmermann T, Premraj R, Sillito J, Breu S. Improving bug tracking systems. In: Proc. of the 31st Int'l Conf. on Software Engineering—Companion volume. Vancouver: IEEE, 2009. 247–250. [doi: 10.1109/ICSE-COMPANION.2009.5070993]

[8] Wang DD, Wang Q, Yang Y, Li Q, Wang HT, Yuan F. "Is it really a defect?" An empirical study on measuring and improving the process of software defect reporting. In: Proc. of the 2011 Int'l Symp. on Empirical Software Engineering and Measurement. Banff: IEEE, 2011. 434–443. [doi: 10.1109/ESEM.2011.62]

[9] Xia X, Lo D, Wen M, Shihab E, Zhou B. An empirical study of bug report field reassignment. In: Proc. of the 2014 Software Evolution Week—IEEE Conf. on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE). Antwerp: IEEE, 2014. 174–183. [doi: 10.1109/CSMR-WCRE.2014.6747167]

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

[11] Soltani M, Hermans F, Bäck T. The significance of bug report elements. Empirical Software Engineering, 2020, 25(6): 5255–5294. [doi:

10.1007/s10664-020-09882-z]

[12] Chen X, Jiang H, Li XC, He TK, Chen ZY. Automated quality assessment for crowdsourced test reports of mobile applications. In: Proc. of the 25th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Campobasso: IEEE, 2018. 368–379. [doi: 10.1109/SANER.2018.8330224]

[13] Chaparro O, Bernal-Cárdenas C, Lu J, Moran K, Marcus A, Di Penta M, Poshyvanyk D, Ng V. Assessing the quality of the steps to reproduce in bug reports. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Tallinn: ACM, 2019. 86–96. [doi: 10.1145/3338906.3338947]

[14] Just S, Premraj R, Zimmermann T. Towards the next generation of bug tracking systems. In: Proc. of the 2008 IEEE Symp. on Visual Languages and Human-centric Computing. Herrsching: IEEE, 2008. 82–85. [doi: 10.1109/VLHCC.2008.4639063]

[15] Ko AJ, Myers BA, Chau DH. A linguistic analysis of how people describe software problems. In: Proc. of Visual Languages and Human-centric Computing (VL/HCC2006). Brighton: IEEE, 2006. 127–134. [doi: 10.1109/VLHCC.2006.3]

[16] Moran K, Linares-Vásquez M, Bernal-Cárdenas C, Poshyvanyk D. Auto-completing bug reports for Android applications. In: Proc. of the 10th Joint Meeting on Foundations of Software Engineering. Bergamo: ACM, 2015. 673–686. [doi: 10.1145/2786805.2786857]

[17] Yang G, Min K, Lee JW, Lee B. Applying topic modeling and similarity for predicting bug severity in cross projects. KSII Trans. on Internet and Information Systems, 2019, 13(3): 1583−1598. [doi: 10.3837/tiis.2019.03.026]

[18] Hao R, Feng Y, Jones JA, Li YY, Chen ZY. Ctras: Crowdsourced test report aggregation and summarization. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Montreal: IEEE, 2019. 900–911. [doi: 10.1109/ICSE.2019.00096]

[19] Chen JJ, Patra J, Pradel M, Xiong YF, Zhang HY, Hao D, Zhang L. A survey of compiler testing. ACM Computing Surveys, 2021, 53(1): 4. [doi: 10.1145/3363562]

[20] Tu FF, Zhou MH. Data quality problems in software development activity data. Ruan Jian Xue Bao/Journal of Software, 2019, 30(5): 1522−1531 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5727.htm [doi: 10.13328/j.cnki.jos.005727]

[21] Bettenburg N, Premraj R, Zimmermann T, Kim S. Duplicate bug reports considered harmful… really? In: Proc. of the 24th IEEE Int'l Conf. on Software Maintenance. Beijing: IEEE, 2008. 337–345. [doi: 10.1109/ICSM.2008.4658082]

[22] Breu S, Premraj R, Sillito J, Zimmermann T. Frequently asked questions in bug reports. Technical Report, Calgary: University of Calgary, 2009. [doi: 10.11575/PRISM/31259]

[23] Laukkanen EI, Mantyla MV. Survey reproduction of defect reporting in industrial software development. In: Proc. of the 2011 Int'l Symp. on Empirical Software Engineering and Measurement. Banff: IEEE, 2011. 197–206. [doi: 10.1109/ESEM.2011.28]

[24] Davies S, Roper M. What's in a bug report? In: Proc. of the 8th ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement. Torino: ACM, 2014. 1–10. [doi: 10.1145/2652524.2652541]

[25] Garousi V, Ergezer EG, Herkiloğlu K. Usage, usefulness and quality of defect reports: An industrial case study. In: Proc. of the 20th Int'l Conf. on Evaluation and Assessment in Software Engineering. Limerick: ACM, 2016. 1–6. [doi: 10.1145/2915970.2916009]

[26] Yusop NSM, Grundy J, Vasa R. Reporting usability defects: Do reporters report what software developers need? In: Proc. of the 20th Int'l Conf. on Evaluation and Assessment in Software Engineering. Limerick: ACM, 2016. 1–10. [doi: 10.1145/2915970.2915995]

[27] Zanetti MS, Scholtes I, Tessone CJ, Schweitzer F. Categorizing bugs with social networks: A case study on four open source software communities. In: Proc. of the 35th Int'l Conf. on Software Engineering (ICSE). San Francisco: IEEE, 2013. 1032–1041. [doi: 10.1109/ICSE.2013.6606653]

[28] Fan YR, 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. [doi: 10.1109/TSE.2018.2864217]

[29] Aversano L, Tedeschi E. Bug report quality evaluation considering the effect of submitter reputation. In: Proc. of the 11th Int'l Joint Conf. on Software Technologies. Lisbon: ICSOFT-EA, 2016. 194–201. [doi: 10.5220/0005982601940201]

[30] Bhattacharya P, Ulanova L, Neamtiu I, Koduru SC. An empirical analysis of bug reports and bug fixing in open source Android APPs. In: Proc. of the 17th European Conf. on Software Maintenance and Reengineering. Genova: IEEE, 2013. 133–143. [doi: 10.1109/CSMR.2013.23]

[31] Chen X, Jiang H, Li XC, Nie LM, Yu DJ, He TK, Chen ZY. A systemic framework for crowdsourced test report quality assessment. Empirical Software Engineering, 2020, 25(2): 1382–1418. [doi: 10.1007/s10664-019-09793-8]

[32] He JJ, Xu L, Fan YR, Xu Z, Yan M, Lei Y. Deep learning based valid bug reports determination and explanation. In: Proc. of the 31st IEEE Int'l Symp. on Software Reliability Engineering (ISSRE). Coimbra: IEEE, 2020. 184–194. [doi: 10.1109/ISSRE5003.2020.00026]

[33] Linstead E, Baldi P. Mining the coherence of GNOME bug reports with statistical topic models. In: Proc. of the 6th IEEE Int'l Working Conf. on Mining Software Repositories. Vancouver: IEEE, 2009. 99–102. [doi: 10.1109/MSR.2009.5069486]

[34] Schugerl P, Rilling J, Charland P. Mining bug repositories—A quality assessment. In: Proc. of the 2008 Int'l Conf. on Computational

Intelligence for Modelling Control & Automation. Vienna: IEEE, 2008. 1105–1110. [doi: 10.1109/CIMCA.2008.63]

[35] Bettenburg N, Just S, Schröter A, Weiß C, Premraj R, Zimmermann T. Quality of bug reports in eclipse. In: Proc. of the 2007 OOPSLA Workshop on Eclipse Technology EXchange. Montreal: ACM, 2007. 21–25. [doi: 10.1145/1328279.1328284]

[36] Dit B, Poshyvanyk D, Marcus A. Measuring the semantic similarity of comments in bug reports. In: Proc. of the 1st Int'l Workshop on Semantic Technologies in System Maintenance (STSM2008). STSM, 2008. 265–280

[37] Hooimeijer P, Weimer W. Modeling bug report quality. In: Proc. of the 22nd IEEE/ACM Int'l Conf. on Automated Software Engineering. Atlanta: ACM, 2007. 34–43. [doi: 10.1145/1321631.1321639]

[38] Gromova A, Itkin I, Pavlov S, Korovayev A. Raising the quality of bug reports by predicting software defect indicators. In: Proc. of the 19th IEEE Int'l Conf. on Software Quality, Reliability and Security Companion (QRS-C). Sofia: IEEE, 2019. 198–204. [doi: 10.1109/QRS-C.2019.00048]

[39] Wang JJ, Cui Q, Wang Q, Wang S. Towards effectively test report classification to assist crowdsourced testing. In: Proc. of the 10th ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement. Ciudad Real: ACM, 2016. 1–10. [doi: 10.1145/2961111.2962584]

[40] Feng Y, Jones JA, Chen ZY, Fang CR. Multi-objective test report prioritization using image understanding. In: Proc. of the 31st IEEE/ACM Int'l Conf. on Automated Software Engineering. Singapore: ACM, 2016. 202–213. [doi: 10.1145/2970276.2970367]

[41] Feng Y, Chen ZY, Jones JA, Fang CR, Xu BW. Test report prioritization to assist crowdsourced testing. In: Proc. of the 10th Joint Meeting on Foundations of Software Engineering. Bergamo: ACM, 2015. 225–236. [doi: 10.1145/2786805.2786862]

[42] Anvik J, Hiew L, Murphy GC. Coping with an open bug repository. In: Proc. of the OOPSLA Workshop on Eclipse Technology Exchange. San Diego: ACM, 2005. 35–39. [doi: 10.1145/1117696.1117704]

[43] Breu S, Premraj R, Sillito J, Zimmermann T. Information needs in bug reports: Improving cooperation between developers and users. In: Proc. of the 2010 ACM Conf. on Computer Supported Cooperative Work. Savannah: ACM, 2010. 301–310. [doi: 10.1145/1718918.1718973]

[44] GitHub Open Source Project Maintainers. An open letter to GitHub from the maintainers of open source projects. 2019. https://github.com/dear-github/dear-github

[45] Dal Sasso T, Mocci A, Lanza M. What makes a satisficing bug report? In: Proc. of the 2016 IEEE Int'l Conf. on Software Quality, Reliability and Security (QRS). Vienna: IEEE, 2016. 164–174. [doi: 10.1109/QRS.2016.28]

[46] Herraiz I, German DM, Gonzalez-Barahona JM, Robles G. Towards a simplification of the bug report form in Eclipse. In: Proc. of the 2008 Int'l Working Conf. on Mining Software Repositories. Leipzig: ACM, 2008. 145–148. [doi: 10.1145/1370750.1370786]

[47] Twidale MB, Nichols DM. Exploring usability discussions in open source development. In: Proc. of the 38th Annual Hawaii Int'l Conf. on System Sciences. Big Island: IEEE, 2005. 198c. [doi: 10.1109/HICSS.2005.266]

[48] Nichols DM, Twidale MB. Usability processes in open source projects. Software Process: Improvement and Practice, 2006, 11(2): 149–162. [doi: 10.1002/spip.256]

[49] Castro M, Costa M, Martin JP. Better bug reporting with better privacy. ACM SIGOPS Operating Systems Review, 2008, 42(2): 319–328. [doi: 10.1145/1353535.1346322]

[50] Yusop NSM, Grundy J, Vasa R. Reporting usability defects: Limitations of open source defect repositories and suggestions for improvement. In: Proc. of the 24th ASWEC Australasian Software Engineering Conf. Adelaide: ACM, 2015. 38–43. [doi: 10.1145/2811681.2811689]

[51] Song Y, Chaparro O. BEE: A tool for structuring and analyzing bug reports. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Online: ACM, 2020. 1551–1555. [doi: 10.1145/3368089.3417928]

[52] Chaparro O, Lu J, Zampetti F, Moreno L, Di Penta M, Marcus A, Bavota G, Ng V. Detecting missing information in bug descriptions. In: Proc. of the 11th Joint Meeting on Foundations of Software Engineering. Paderborn: ACM, 2017. 396–407. [doi: 10.1145/3106237.3106285]

[53] Karim R, Ihara A, Choi E, Iida H. Identifying and predicting key features to support bug reporting. Journal of Software: Evolution and Process, 2019, 31(12): e2184. [doi: 10.1002/smr.2184]

[54] Lotufo R, Passos L, Czarnecki K. Towards improving bug tracking systems with game mechanisms. In: Proc. of the 9th IEEE Working Conf. on Mining Software Repositories (MSR). Zurich: IEEE, 2012. 2–11. [doi: 10.1109/MSR.2012.6224293]

[55] Moran K, Poshyvanyk D. Fixing bug reporting for mobile and GUI-based applications. In: Proc. of the 38th Int'l Conf. on Software Engineering Companion. Austin: ACM, 2016. 831–834. [doi: 10.1145/2889160.2889269]

[56] Moran K, Linares-Vásquez M, Bernal-Cárdenas C, Vendome C, Poshyvanyk D. Automatically discovering, reporting and reproducing

Android application crashes. In: Proc. of the 2016 IEEE Int'l Conf. on Software Testing, Verification and Validation (ICST). Chicago: IEEE, 2016. 33–44. [doi: 10.1109/ICST.2016.34]

[57] Yu SC. Crowdsourced report generation via bug screenshot understanding. In: Proc. of the 34th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). San Diego: IEEE, 2019. 1277–1279. [doi: 10.1109/ASE.2019.00161]

[58] Herbold S, Grabowski J, Waack S, Bünting U. Improved bug reporting and reproduction through non-intrusive GUI usage monitoring and automated replaying. In: Proc. of the 4th IEEE Int'l Conf. on Software Testing, Verification and Validation Workshops. Berlin: IEEE, 2011. 232–241. [doi: 10.1109/ICSTW.2011.66]

[59] White M, Linares-Vásquez M, Johnson P, Bernal-Cárdenas C, Poshyvanyk D. Generating reproducible and replayable bug reports from Android application crashes. In: Proc. of the 23rd Int'l Conf. on Program Comprehension. Florence: IEEE, 2015. 48–59. [doi: 10.1109/ICPC.2015.14]

[60] Zamfir C, Candea G. Low-overhead bug fingerprinting for fast debugging. In: Proc. of the 1st Int'l Conf. on Runtime Verification. Julians: Springer, 2010. 460–468. [doi: 10.1007/978-3-642-16612-9_35]

[61] Zhang T, Chen JC, Jiang H, Luo XP, Xia X. Bug report enrichment with application of automated fixer recommendation. In: Proc. of the 25th Int'l Conf. on Program Comprehension. Buenos: ACM, 2017. 230–240. [doi: 10.1109/ICPC.2017.28]

[62] Weimer W. Patches as better bug reports. In: Proc. of the 5th Int'l Conf. on Generative Programming and Component Engineering. Portland: ACM, 2006. 181–190. [doi: 10.1145/1173706.1173734]

[63] Anvik J, Murphy GC. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. ACM Trans. on Software Engineering and Methodology, 2011, 20(3): 1–35. [doi: 10.1145/2000791.2000794]

[64] Zhang W, Challis C. Software component prediction for bug reports. In: Proc. of the 11th Asian Conf. on Machine Learning. Nagoya: PMLR, 2019. 806–821.

[65] Sureka A. Learning to classify bug reports into components. In: Proc. of the 50th Int'l Conf. on Objects, Models, Components, Patterns. Prague: Springer, 2012. 288–303. [doi: 10.1007/978-3-642-30561-0_20]

[66] Somasundaram K, Murphy GC. Automatic categorization of bug reports using latent Dirichlet allocation. In: Proc. of the 5th India Software Engineering Conf. Kanpur: ACM, 2012. 125–130. [doi: 10.1145/2134254.2134276]

[67] Yan M, Zhang XH, Yang D, Xu L, Kymer JD. A component recommender for bug reports using discriminative probability latent semantic analysis. Information and Software Technology, 2016, 73: 37–51. [doi: 10.1016/j.infsof.2016.01.005]

[68] Wang DQ, Zhang H, Liu R, Lin MX, Wu WJ. Predicting bugs' components via mining bug reports. Journal of Software, 2012, 7(5): 1149–1154.

[69] Lamkanfi A, Demeyer S. Predicting reassignments of bug reports—An exploratory investigation. In: Proc. of the 17th European Conf. on Software Maintenance and Reengineering. Genova: IEEE, 2013. 327–330. [doi: 10.1109/CSMR.2013.42]

[70] Sabor KK, Nayrolles M, Trabelsi A, Hamou-Lhadj A. An approach for predicting bug report fields using a neural network learning model. In: Proc. of the 2018 IEEE Int'l Symp. on Software Reliability Engineering Workshops (ISSREW). Memphis: IEEE, 2018. 232–236. [doi: 10.1109/ISSREW.2018.00011]

[71] Sabor KK, Hamou-Lhadj A, Trabelsi A, Hassine J. Predicting bug report fields using stack traces and categorical attributes. In: Proc. of the 29th Annual Int'l Conf. on Computer Science and Software Engineering. Markham: IBM, 2019. 224–233.

[72] Zhang W, Challis C. Automatic bug priority prediction using DNN based regression. In: Proc. of the 15th Int'l Conf. on Natural Computation, Fuzzy Systems and Knowledge Discovery. Kunming: Springer, 2019. 333–340. [doi: 10.1007/978-3-030-32456-8_36]

[73] Umer Q, Liu H, Illahi I. CNN-based automatic prioritization of bug reports. IEEE Trans. on Reliability, 2020, 69(4): 1341–1354. [doi: 10.1109/TR.2019.2959624]

[74] Kanwal J, Maqbool O. Bug prioritization to facilitate bug report triage. Journal of Computer Science and Technology, 2012, 27(2): 397–412. [doi: 10.1007/s11390-012-1230-3]

[75] Tran HM, Le ST, van Nguyen S, Ho PT. An analysis of software bug reports using machine learning techniques. SN Computer Science, 2020, 1: 4. [doi: 10.1007/s42979-019-0004-1]

[76] Tian Y, Lo D, Xia X, Sun CN. Automated prediction of bug report priority using multi-factor analysis. Empirical Software Engineering, 2015, 20(5): 1354–1383. [doi: 10.1007/s10664-014-9331-y]

[77] Tian Y, Lo D, Sun CN. Drone: Predicting priority of reported bugs by multi-factor analysis. In: Proc. of the 2013 IEEE Int'l Conf. on Software Maintenance. Eindhoven: IEEE, 2013. 200–209. [doi: 10.1109/ICSM.2013.31]

[78] Goyal N, Aggarwal N, Dutta M. A novel way of assigning software bug priority using supervised classification on clustered bugs data. In: El-Alfy ESM, Thampi AM, Takagi H, Piramuthu S, Hanne T, eds. Advances in Intelligent Informatics. Cham: Springer, 2015. 493–501. [doi: 10.1007/978-3-319-11218-3_44]

[79] Yu L, Tsai WT, Zhao W, Wu F. Predicting defect priority based on neural networks. In: Proc. of the 6th Int'l Conf. on Advanced Data Mining and Applications. Chongqing: Springer, 2010. 356–367. [doi: 10.1007/978-3-642-17313-4_35]

[80] Sharma M, Kumari M, Singh VB. Bug priority assessment in cross-project context using entropy-based measure. In: Proc. of the 2019 ICMLCI on Advances in Machine Learning and Computational Intelligence. Bugis: Springer, 2020. 113–128. [doi: 10.1007/978-981-15-5243-4_10]

[81] Sharma M, Bedi P, Chaturvedi KK, Singh VB. Predicting the priority of a reported bug using machine learning techniques and cross project validation. In: Proc. of the 12th Int'l Conf. on Intelligent Systems Design and Applications (ISDA). Kochi: IEEE, 2012. 539–545. [doi: 10.1109/ISDA.2012.6416595]

[82] Lamkanfi A, Demeyer S, Soetens QD, Verdonck T. Comparing mining algorithms for predicting the severity of a reported bug. In: Proc. of the 15th European Conf. on Software Maintenance and Reengineering. Oldenburg: IEEE, 2011. 249–258. [doi: 10.1109/CSMR.2011.31]

[83] Chawla I, Singh SK. Improving bug report quality by predicting correct component in bug reports. Int'l Journal of Computational Intelligence Studies, 2019, 8(1–2): 143–157.

[84] Kukkar A, Mohana R, Nayyar A, Kim J, Kang BG, Chilamkurti N. A novel deep-learning-based bug severity classification technique using convolutional neural networks and random forest with boosting. Sensors, 2019, 19(13): 2964. [doi: 10.3390/s19132964]

[85] Menzies T, Marcus A. Automated severity assessment of software defect reports. In: Proc. of the 2008 IEEE Int'l Conf. on Software Maintenance. Beijing: IEEE, 2008. 346–355. [doi: 10.1109/ICSM.2008.4658083]

[86] Sharma G, Sharma S, Gujral S. A novel way of assessing software bug severity using dictionary of critical terms. Procedia Computer Science, 2015, 70: 632–639. [doi: 10.1016/j.procs.2015.10.059]

[87] Yang CZ, Hou CC, Kao WC, Chen IX. An empirical study on improving severity prediction of defect reports using feature selection. In: Proc. of the 19th Asia-Pacific Software Engineering Conf. Hong Kong: IEEE, 2012. 240–249. [doi: 10.1109/APSEC.2012.144]

[88] Roy NKS, Rossi B. Towards an improvement of bug severity classification. In: Proc. of the 40th EUROMICRO Conf. on Software Engineering and Advanced Applications. Verona: IEEE, 2014. 269–276. [doi: 10.1109/SEAA.2014.51]

[89] Liu WJ, Wang SS, Chen X, Jiang H. Predicting the severity of bug reports based on feature selection. Int'l Journal of Software Engineering and Knowledge Engineering, 2018, 28(4): 537–558. [doi: 10.1142/S0218194018500158]

[90] Zhang T, Chen JC, Yang G, Lee B, Luo XP. Towards more accurate severity prediction and fixer recommendation of software bugs. Journal of Systems and Software, 2016, 117: 166–184. [doi: 10.1016/j.jss.2016.02.034]

[91] Yang G, Zhang T, Lee B. An emotion similarity based severity prediction of software bugs: A case study of open source projects. IEICE Trans. on Information and Systems, 2018, E101.D(8): 2015–2026. [doi: 10.1587/transinf.2017EDP7406]

[92] Ramay WY, Umer Q, Yin XC, Zhu C, Illahi I. Deep neural network-based severity prediction of bug reports. IEEE Access, 2019, 7: 46846–46857. [doi: 10.1109/ACCESS.2019.2909746]

[93] Tan YS, Xu SJ, Wang ZW, Zhang T, Xu Z, Luo XP. Bug severity prediction using question-and-answer pairs from stack overflow. Journal of Systems and Software, 2020, 165: 110567. [doi: 10.1016/j.jss.2020.110567]

[94] Han BZ, Li XH, Xing ZC, Liu HT, Feng ZY. Learning to predict severity of software vulnerability using only vulnerability description. In: Proc. of the 2017 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Shanghai: IEEE, 2017. 125–136. [doi: 10.1109/ICSME.2017.52]

[95] Iqbal S, Naseem R, Jan S, Alshmrany S, Yasar M, Ali A. Determining bug prioritization using feature reduction and clustering with classification. IEEE Access, 2020, 8: 215661–215678. [doi: 10.1109/ACCESS.2020.3035063]

[96] Zhang TL, Chen R, Yang X, Zhu HY. An uncertainty based incremental learning for identifying the severity of bug report. Int'l Journal of Machine Learning and Cybernetics, 2020, 11(1): 123–136. [doi: 10.1007/s13042-019-00961-2]

[97] Roy NKS, Rossi B. Cost-sensitive strategies for data imbalance in bug severity classification: Experimental results. In: Proc. of the 43rd Euromicro Conf. on Software Engineering and Advanced Applications (SEAA). Vienna: IEEE, 2017. 426–429. [doi: 10.1109/SEAA.2017.71]

[98] Guo SK, Chen R, Li H, Zhang TL, Liu YQ. Identify severity bug report with distribution imbalance by CR-SMOTE and ELM. Int'l Journal of Software Engineering and Knowledge Engineering, 2019, 29(2): 139–175. [doi: 10.1142/S0218194019500074]

[99] Hamdy A, El-Laithy A. SMOTE and feature selection for more effective bug severity prediction. Int'l Journal of Software Engineering and Knowledge Engineering, 2019, 29(6): 897–919. [doi: 10.1142/S0218194019500311]

[100] Tian Y, Ali N, Lo D, Hassan AE. On the unreliability of bug severity data. Empirical Software Engineering, 2016, 21(6): 2298–2323. [doi: 10.1007/s10664-015-9409-1]

[101] Wu LL, Xie BY, Kaiser GE, Passonneau R. BugMiner: Software reliability analysis via data mining of bug reports. In: Proc. of the 23rd

Int'l Conf. on Software Engineering & Knowledge Engineering (SEKE2011). Miami Beach: SEKE, 2011. 95–100. [doi: 10.7916/D8W95JCN]

[102] Xia X, Lo D, Shihab E, Wang XY. Automated bug report field reassignment and refinement prediction. IEEE Trans. on Reliability, 2016, 65(3): 1094–1113. [doi: 10.1109/TR.2015.2484074]

[103] Kumari M, Singh VB. An improved classifier based on entropy and deep learning for bug priority prediction. In: Proc. of the 18th Int'l Conf. on Intelligent Systems Design and Applications. Vellore: Springer, 2018. 571–580. [doi: 10.1007/978-3-030-16657-1_53]

[104] Alenezi M, Banitaan S. Bug reports prioritization: Which features and classifier to use? In: Proc. of the 12th Int'l Conf. on Machine Learning and Applications. Miami: IEEE, 2013. 112–116. [doi: 10.1109/ICMLA.2013.114]

[105] Umer Q, Liu H, Sultan Y. Emotion based automated priority prediction for bug reports. IEEE Access, 2018, 6: 35743–35752. [doi: 10.1109/ACCESS.2018.2850910]

[106] Gao GF, Li H, Chen R, Ge X, Guo SK. Identification of high priority bug reports via integration method. In: Proc. of the 6th CCF Conf. on Big Data. Xi'an: Springer, 2018. 336–349. [doi: 10.1007/978-981-13-2922-7_23]

[107] Kanwal J, Maqbool O. Managing open bug repositories through bug report prioritization using SVMs. In: Proc. of the 4th Int'l Conf. on Open-source Systems and Technologies. Lahore: ICOSST, 2010. 1–7.

[108] Yang G, Baek S, Lee JW, Lee B. Analyzing emotion words to predict severity of software bugs: A case study of open source projects. In: Proc. of the Symp. on Applied Computing. Marrakech: ACM, 2017. 1280–1287. [doi: 10.1145/3019612.3019788]

[109] Iliev M, Karasneh B, Chaudron MRV, Essenius E. Automated prediction of defect severity based on codifying design knowledge using ontologies. In: Proc. of the 1st Int'l Workshop on Realizing AI Synergies in Software Engineering (RAISE). Zurich: IEEE, 2012. 7–11. [doi: 10.1109/RAISE.2012.6227962]

[110] Sabor KK, Hamdaqa M, Hamou-Lhadj A. Automatic prediction of the severity of bugs using stack traces. In: Proc. of the 26th Annual Int'l Conf. on Computer Science and Software Engineering. Toronto: IBM, 2016. 96–105.

[111] Arokiam J, Bradbury JS. Automatically predicting bug severity early in the development process. In: Prof. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering: New Ideas and Emerging Results. Seoul: ACM, 2020. 17–20. [doi: 10.1145/3377816.3381738]

[112] Sharmin S, Aktar F, Ali AA, Khan MAH, Shoyaib M. BFSp: A feature selection method for bug severity classification. In: Proc. of the IEEE Region 10 Humanitarian Technology Conf. (R10-HTC). Dhaka: IEEE, 2017. 750–754. [doi: 10.1109/R10-HTC.2017.8289066]

[113] Singh VB, Misra S, Sharma M. Bug severity assessment in cross project context and identifying training candidates. Journal of Information & Knowledge Management, 2017, 16(1): 1750005. [doi: 10.1142/S0219649217500058]

[114] Alia SS, Haque N, Sharmin S, Khaled SM, Shoyaib M. Bug severity classification based on class-membership information. In: Proc. of the 7th Joint Int'l Conf. on Informatics, Electronics & Vision (ICIEV) and the 2nd Int'l Conf. on Imaging, Vision & Pattern Recognition (icIVPR). Kitakyushu: IEEE, 2018. 520–525. [doi: 10.1109/ICIEV.2018.8641032]

[115] Chauhan A, Kumar R. Bug severity classification using semantic feature with convolution neural network. In: Proc. of the 2019 ICCET on Computing in Engineering and Technology. Bugis: Springer, 2020. 327–335. [doi: 10.1007/978-981-32-9515-5_31]

[116] Nnamoko N, Cabrera-Diego LA, Campbell D, Korkontzelos Y. Bug severity prediction using a hierarchical one-vs.-remainder approach. In: Proc. of the 24th Int'l Conf. on Applications of Natural Language to Information Systems. Salford: Springer, 2019. 247–260. [doi: 10.1007/978-3-030-23281-8_20]

[117] Chaturvedi KK, Singh VB. Determining bug severity using machine learning techniques. In: Proc. of the 6th CSI Int'l Conf. on Software Engineering (CONSEG). Indore: IEEE, 2012. 1–6. [doi: 10.1109/CONSEG.2012.6349519]

[118] Kukkar A, Mohana R, Kumar Y. Does bug report summarization help in enhancing the accuracy of bug severity classification? Procedia Computer Science, 2020, 167: 1345–1353. [doi: 10.1016/j.procs.2020.03.345]

[119] Kumari M, Singh UK, Sharma M. Entropy based machine learning models for software bug severity assessment in cross project context. In: Proc. of the 20th Int'l Conf. on Computational Science and Its Applications. Cagliari: Springer, 2020. 939–953. [doi: 10.1007/978-3-030-58817-5_66]

[120] Choeikiwong T, Vateekul P. Improve accuracy of defect severity categorization using semi-supervised approach on imbalanced data sets. In: Proc. of the Int'l MultiConf. of Engineers and Computer Scientists. Hong Kong: IMECS, 2016. 1–5.

[121] Baarah A, Aloqaily A, Salah Z, Zamzeer M, Sallam M. Machine learning approaches for predicting the severity level of software bug reports in closed source projects. Int'l Journal of Advanced Computer Science and Applications, 2019, 10(8): 285–294. [doi: 10.14569/IJACSA.2019.0100836]

[122] Awad MA, ElNainay MY, Abougabal MS. Predicting bug severity using customized weighted majority voting algorithms. In: Proc. of the 2017 Japan-Africa Conf. on Electronics, Communications and Computers (JAC-ECC). Alexandria: IEEE, 2017. 170–175. [doi: 10.1109/JEC-ECC.2017.8305803]

[123] Zhang T, Yang G, Lee B, Chan ATS. Predicting severity of bug report by mining bug repository with concept profile. In: Proc. of the 30th Annual ACM Symp. on Applied Computing. Salamanca: ACM, 2015. 1553–1558. [doi: 10.1145/2695664.2695872]

[124] Pushpalatha MN, Mrunalini M. Predicting the severity of bug reports using classification algorithms. In: Proc. of the 2016 Int'l Conf. on Circuits, Controls, Communications and Computing (I4C). Bangalore: IEEE, 2016. 1–4. [doi: 10.1109/CIMCA.2016.8053276]

[125] Pushpalatha MN, Mrunalini M. Predicting the severity of closed source bug reports using ensemble methods. In: Proc. of the 2nd Int'l Conf. on SICA. Bugis: Springer, 2019. 589–597. [doi: 10.1007/978-981-13-1927-3_62]

[126] Jindal R, Malhotra R, Jain A. Prediction of defect severity by mining software project reports. Int'l Journal of System Assurance Engineering and Management, 2017, 8(2): 334–351. [doi: 10.1007/s13198-016-0438-y]

[127] Hamdy A, El-Laithy A. Semantic categorization of software bug repositories for severity assignment automation. In: Jarzabek S, Poniszewska-Marańda A, Madeyski L, eds. Integrating Research and Practice in Software Engineering. Cham: Springer, 2020. 15–30. [doi: 10.1007/978-3-030-26574-8_2]

[128] Tian Y, Lo D, Sun CN. Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. In: Proc. of the 19th Working Conf. on Reverse Engineering. Kingston: IEEE, 2012. 215–224. [doi: 10.1109/WCRE.2012.31]

[129] Yang G, Zhang T, Lee B. Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports. In: Proc. of the 38th IEEE Annual Computer Software and Applications Conf. Vasteras: IEEE, 2014. 97–106. [doi: 10.1109/COMPSAC.2014.16]

[130] Tatham S. How to report bugs effectively. 2008. https://www.chiark.greenend.org.uk/~sgtatham/bugs.html

[131] Dit B, Marcus A. Improving the readability of defect reports. In: Proc. of the 2008 Int'l Workshop on Recommendation Systems for Software Engineering. Atlanta: ACM, 2008. 47–49. [doi: 10.1145/1454247.1454265]

附中文参考文献:

[20] 涂菲菲, 周明辉. 软件开发活动数据的数据质量问题. 软件学报, 2019, 30(5): 1522−1531. http://www.jos.org.cn/1000-9825/5727.htm [doi: 10.13328/j.cnki.jos.005727]

邹卫琴(1988－), 女, 博士, 副教授, CCF 专业会员, 主要研究领域为缺陷定位, 软件仓库挖掘.

陈林(1979－), 男, 博士, 副教授, 博士生导师, CCF 高级会员, 主要研究领域为软件工程, 程序分析.

张静宣(1988－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为软件仓库挖掘, 智能软件工程.

玄跻峰(1984－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为软件分析与测试.

张霄炜(1996－), 女, 博士, 主要研究领域为机器学习, 软件质量分析.