软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(11):3983–4007 [doi: 10.13328/j.cnki.jos.006332]

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

Firefox 缺陷跟踪系统中的用户反馈*

王 燕[1,2], 吴化尧[1], 聂长海[1], 徐家喜[2], 尹 震[1], 钮鑫涛[1]

[1](计算机软件新技术国家重点实验室(南京大学), 江苏 南京 210023)

[2](南京晓庄学院 信息工程学院, 江苏 南京 211171)

通信作者: 王燕, E-mail: wangyan@njxzc.edu.cn

摘 要: 缺陷追踪是软件项目管理的一个重要环节, 是保证现代大规模开源软件开发顺利进行并持续提高软件质量的必要手段. 目前, 大部分开源软件都使用开放的缺陷跟踪系统进行软件缺陷的管理. 它允许用户向开发者提交系统故障(即 defect 类型缺陷)以及系统改进建议(即 enhancement 类型缺陷), 但是这些用户的反馈所起的作用尚未得到充分研究. 针对这一问题, 对 Firefox 的缺陷跟踪系统进行实证研究, 收集了 2018 年和 2019 年提交的 19 474 份 Firefox Desktop 以及 3 057 份 Firefox for Android 缺陷报告. 在此基础上, 对比分析了普通用户和核心开发者提交的缺陷在数量、严重性、组件分布、修复率、修复速度以及修复者上的差别, 并调查了缺陷报告的撰写质量与缺陷处理结果和修复时间的关系. 主要发现包括: (1) 当前缺陷追踪系统中普通用户人数众多, 但参与程度较浅, 86%的用户只提交过一个缺陷, 其中, 高严重等级的缺陷不超过 3%; (2) 普通用户提交的缺陷主要分布在和用户交互相关的 UI 组件上(例如地址栏、音频/视频等), 然而还有 43%的缺陷由于缺乏充分描述信息而难以准确地定位到具体的关联组件; (3) 在缺陷处理结果上, 由于查重系统以及缺陷填报系统在设计上过于简单, 致使普通用户提交的大量缺陷被处理为“无用”缺陷, 缺陷修复率低于 10%; (4) 在缺陷修复流程上, 由于普通用户难以准确、充分地描述缺陷, 导致系统对其重视程度不足, 普通用户提交缺陷的处理流程也比核心开发者提交的复杂, 平均需要多花至少 8 天的时间进行修复. 上述研究结果揭示了当前缺陷追踪系统在用户参与激励机制、缺陷自动查重以及缺陷报告填写智能辅助等方面的不足, 能够为缺陷跟踪系统开发者和管理者改进系统、提高普通用户对开源软件的贡献提供参考.

关键词: 群智化生态系统; 用户反馈; 缺陷跟踪; 缺陷修复; Firefox

中图法分类号: TP311

中文引用格式: 王燕, 吴化尧, 聂长海, 徐家喜, 尹震, 钮鑫涛. Firefox 缺陷跟踪系统中的用户反馈. 软件学报, 2022, 33(11): 3983–4007. http://www.jos.org.cn/1000-9825/6332.htm

英文引用格式: Wang Y, Wu HY, Nie CH, Xu JX, Yin Z, Niu XT. User Feedback in Firefox Bug Tracking System. Ruan Jian Xue Bao/Journal of Software, 2022, 33(11): 3983–4007 (in Chinese). http://www.jos.org. cn/1000-9825/6332.htm

User Feedback in Firefox Bug Tracking System

WANG Yan[1,2], WU Hua-Yao[1], NIE Chang-Hai[1], XU Jia-Xi[2], YIN Zhen[1], NIU Xin-Tao[1]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

[2](School of Information Engineering, Nanjing Xiaozhuang University, Nanjing 211171, China)

Abstract: Bug tracking systems are a vital part of software project management. It is a necessary means to ensure the smooth development of modern large-scale open source software and continuously improve software quality. Most open source software ecosystems currently use open bug tracking systems to manage software bugs. It allows users to submit system failures (called defect bugs)

* 基金项目: 国家重点研发计划(2018YFB1003800); 国家自然科学基金(61902174, 62072226); 江苏省自然科学基金(BK20190291)

收稿时间: 2020-09-26; 修改时间: 2020-11-03, 2021-01-28; 采用时间: 2021-03-05

and suggestions for system improvements (called enhancement bugs), but the role of feedback from these users has not been fully studied. Therefore, this work conducted an empirical study on the bug tracking system used by Firefox, and collected 19 474 and 3 057 bug reports submitted in 2018 and 2019 for Firefox Desktop and Firefox for Android, respectively. Based on this, it is compared and analyzed the differences between the number, severity, distribution on components, fixing rate, fixing efficiency and assignees of bugs submitted by ordinary users and core developers, and at the same time, the relationship between the quality of bug reports and the fixing rate and efficiency of bugs is investigated. The main findings are as follows. (1) There are a large number of ordinary users, but their participation is still superficial. 86% of ordinary users have only submitted one bug and no more than 3% of bugs are of high severity. (2) The bugs submitted by ordinary users mainly distributed on UI components related to user interaction (e.g., address bar, audio/video, etc.), but there are also 43% of bugs that are difficult to accurately locate due to lack of sufficient description information. (3) In terms of bug processing results, due to the simple design of the duplicate checking system and bug filling system, a large number of bugs are treated as “useless” ones, and the fixing rate is less than 10%. (4) In the bug fixing process, due to the difficulty of ordinary users to accurately and fully describe bugs, the system does not pay enough attention to them, thus the process of bugs submitted by ordinary users is more complicated than that of core developers, and it takes at least 8 more days on average to fix them. These results reveal the shortcomings of the current bug tracking system in terms of user participation incentive mechanism, automatic bug duplicate checking, and intelligent assistance in filling out bug reports, which can provide help for the system developers and managers to improve system and enhance the contributions of ordinary users to open source software.

Key words: collective intelligence ecosystem; user feedback; bug tracking; bug fixing; Firefox

现代软件系统在互联网的相互连接作用下, 正形成一种复杂的大规模软件生态, 群智化、生态化的软件开发方法正逐渐成为当前软件开发的主流选择[1]. 开源软件是一种基于互联网群体智能的软件开发实践, 具有任何动机的参与者只要遵守软件许可证规定的条款, 都可以平等地参与软件开发任务, 通过协同构造出高质量的复杂软件系统, 展现出远超过单一开发者或传统软件开发组织的智能行为. 由于参与者众多, 软件开发过程中产生的信息量巨大, 开源软件通常需要借助于各种信息管理系统进行管理, 缺陷跟踪系统就是其中一种. 它用于记录被发现的缺陷, 并对缺陷修复的整个生命周期进行管理.

软件缺陷伴随着软件开发, 未被发现的缺陷可能会造成巨大的经济甚至威胁人类生命安全[2]. 近年来, 由于软件的规模以及复杂度不断地提高, 程序中的缺陷数量也随之增加, 缺陷管理变得越来越具有挑战性, 通常还需要多人合作共同处理[3]. 为了协调和避免重复工作, 多数大型开源软件项目都利用开放的缺陷跟踪系统跟踪和管理发布缺陷, 它为身处不同地理位置的参与者提供了统一的交流合作平台, 在开源软件项目中起着重要作用[4]. 这里的开放性指的是开源软件的开发者和用户都可以发布其所发现的缺陷或者对已有的缺陷进行评论(即对缺陷报告描述的缺陷进行补充说明或者提供更多的细节信息). Anvik 等人[5]指出: 开放性缺陷跟踪系统有助于吸引众多用户报告软件中存在的各种缺陷, 并协同合作共同解决这些缺陷. Pagano 等人[6]的研究也表明, 用户的反馈能够有效地帮助开发者提高软件质量以及识别软件需要新增功能.

目前, 开放性缺陷跟踪系统吸引了大量研究者的关注, 相关研究涵盖对缺陷修复时间影响因素的分析, 以及如何提高缺陷报告质量、如何提高缺陷管理的自动化等方面. 例如, Sasso 等人[7]对 Mozilla 和 Apache 项目的 650 000 份缺陷报告进行分析, 找出报告中影响缺陷修复效率的字段以及报告者通常又容易提供哪些字段; Saha 等人[8]对 Eclipse 以及 Linux 相关项目中处理时间超过一年的缺陷进行调查, 分析哪些因素导致了这些缺陷处理时间过长; Rastkar 等人[9]提出了一种基于监督学习的方法来自动对缺陷报告中的信息进行概述, 使缺陷分类人员可以快速了解缺陷相关信息, 以提高他们的工作效率; Jeong 等人[10]提出一种基于马尔可夫链图形模型, 用来捕获缺陷修复者指派历史记录, 并从中挖掘出开发者关系网, 进而为缺陷修复指定合适人选; Xia 等人[11]提出了一种缺陷重开预测器, 该预测器使用了缺陷描述文本信息、评论文本信息以及包括缺陷报告者、修复者、修复时间、优先等级等元信息来预测缺陷是否会被重开.

虽然针对缺陷跟踪系统的研究众多, 但用户所提交的缺陷能否得到及时而有效的处理、它们对软件产品质量提升能起多大作用等问题尚未得到系统性地定量分析. 也就是开源软件的缺陷跟踪系统的“群智化”程度尚不清楚. 基于直接的观察, 张伟等人[1]认为: 目前, 开源软件只是基于群体智能软件开发的原始形态, 它还远远没有达到其所追求的完全去中心化的社会软件开发. 因此, 我们有必要对当前缺陷跟踪系统的用户参与

程度进行系统深入的调查, 并在此基础上分析当前缺陷跟踪系统面临的问题和挑战, 为缺陷管理的进一步改进提供方向.

具体地, 本文从缺陷的数量、严重程度、组件分布、修复率、修复速度以及修复者等各方面对比分析两类不同缺陷报告者, 即普通用户和核心开发者所提交的缺陷, 评估他们在缺陷系统中的作用和地位. 这里, 普通用户指的是在缺陷管理系统中没有特权的用户, 他们只能进行缺陷的提交和评论, 通常为软件产品的使用者; 而核心开发者除了能够进行缺陷的提交和评论外, 还能进行缺陷确认或者编辑, 他们通常为开源组织的核心人员. 在本文中, 在不引起歧义的情况下, 普通用户也简称为用户. 本文的研究问题如下.

- RQ1: 普通用户与核心开发者在参与人数、提交缺陷个数以及提交缺陷的严重等级方面是否存在明显区别?

除了服务核心开发者之外, 开放缺陷追踪系统的一个重要特点是其能允许数量众多的软件产品使用者来参与软件缺陷的发现及报告活动, 普通用户人数越多, 提交的缺陷数量越多, 表明他们参与的积极性越高. 因此, 我们的第 1 个研究问题(RQ1)关注普通用户和核心开发者在缺陷修复活动中的参与情况, 其中, 参与人数能反映当前缺陷追踪系统是否能有效吸引不同的参与者, 提交缺陷个数能反映不同参与者在缺陷提交上的贡献情况, 而提交缺陷的严重等级能反映不同参与者参与程度的深浅(用户参与程度越深, 那么发现严重程度高的软件缺陷的概率就越大).

- RQ2: 普通用户与核心开发者提交的缺陷分别集中在哪些组件上? 这些组件在分布上是否存在明显区别?

每个软件产品通常都由若干个组件(或者模块)构成, 每个组件完成一定的功能, 但它们的地位和作用不完全相同, 出错的概率也不同. 在第 2 个研究问题(RQ2)中, 我们将缺陷按照与其关联的组件进行分类, 调查哪些组件更容易出错. 同时, 进一步分析普通用户和核心开发者所提交的缺陷在组件分布上的差异, 以了解不同参与者对提高软件产品不同组件质量的贡献.

- RQ3: 普通用户与核心开发者提交的缺陷, 在缺陷处理结果上是否存在明显区别?

修复缺陷报告中提及的缺陷是缺陷追踪系统运行的重要目的, 不同参与者提交的缺陷报告往往会具有不同的缺陷处理结果. 目前, 大多数缺陷跟踪系统如 Bugzilla, JIRA 都包含了多种缺陷处理方案, 如 FIXED(已修复的)、INVALID(无效的)、DUPLICATE(重复的)等, 其中, 只有 FIXED 类型的缺陷才涉及源代码的修改,对软件质量提升有帮助, 其他类型的缺陷可认为是“无用”的. 因此, 我们的第 3 个研究问题(RQ3)关注不同参与者提交的缺陷在最终缺陷处理结果上的差异, 该研究问题还有助于了解不同参与者提交的缺陷对软件质量的最终提升能起多大作用.

- RQ4: 在缺陷报告上, 哪些因素会对缺陷处理结果产生影响?

在对缺陷处理结果进行分析的基础上(RQ3), 还有必要对造成该结果的相关因素进行分析, 从而为缺陷跟踪系统的改进提供参考. 因此, 我们的第 4 个研究问题(RQ4)将进一步分析缺陷报告中哪些因素可能会对缺陷处理结果产生影响. 具体地, 我们将从缺陷标题(对缺陷的简要概括)、缺陷描述(包括发生问题的软件产品版本、复现步骤、预期结果以及实际结果)以及缺陷报告携带的附件这 3 个方面对普通用户提交的缺陷报告进行分析, 识别出影响缺陷处理结果的因素, 以帮助普通用户提高缺陷报告的撰写质量.

- RQ5: 普通用户与核心开发者提交的缺陷, 在缺陷处理的优先级以及缺陷修复时间上是否存在明显区别?

缺陷追踪系统中通常存在大量待处理的缺陷, 限于资源约束, 其中必有一部分缺陷得不到及时处理, 这会导致不同缺陷具有不同的缺陷修复时间. 因此, 我们的第 5 个研究问题(RQ5)主要关注不同参与者提交的缺陷在修复速度上的差异, 包括系统是否会优先处理核心开发者提交的缺陷以及普通用户提交的缺陷是否需要更多的时间才能得到修复这两个方面.

- RQ6: 普通用户与核心开发者提交的缺陷, 在缺陷修复者的指定以及缺陷修复流程上是否存在明显差别?

缺陷除了在处理优先级上的差异外, 不同缺陷往往还会被分配给不同的缺陷修复者来处理, 并具有不同的缺陷修复流程. 因此, 我们的第 6 个研究问题(RQ6)从缺陷修复者和修复流程的角度来对不同参与者提交缺陷的缺陷修复速度进行进一步分析. 其中, 一个缺陷从提交到修复完成需要经历若干个处理环节, 这些环节可能受多种因素如缺陷报告质量、报告者类型、缺陷类型等影响, 并且不同缺陷所经历的处理环节以及每个环节需要的处理时间也都可能不同.

- RQ7: 在缺陷报告上, 哪些因素会对缺陷修复速度产生影响?

在对缺陷修复速度进行分析的基础上(RQ5 和 RQ6), 我们的最后一个研究问题(RQ7)进一步从缺陷报告入手来对影响缺陷修复速度的相关因素进行分析. 具体地, 我们将从缺陷标题长度、缺陷描述的准确和充分程度以及缺陷报告是否携带附件这 3 个角度进行分析, 调查缺陷报告的撰写质量与缺陷修复速度之间的关系

为了回答以上问题, 本文选取了 Mozilla 的两个广泛流行且有代表性的软件产品 Firefox Desktop 和 Firefox for Android 进行实证研究. 具体地, 我们首先从缺陷跟踪系统 Bugzilla 中自动抓取 2018 年和 2019 年提交的这两个产品的全部缺陷的基础数据、缺陷处理历史记录、对应的报告人和修复者以及缺陷标题和评论等相关数据, 然后对这些数据进行整理、统计和分析. 另外, 在 Bugzilla 系统上, 用户既可以报告软件使用过程中出现的问题, 也可以提出软件需要改进的建议, 前者称为 defect 类型缺陷, 将后者称为 enhancement 类型缺陷. 与 defect 类型缺陷相比, enhancement 类型缺陷的发现对用户参与程度要求更高. 也就是说, 只有用户充分了解软件产品已有功能或者特性并进行积极思考的条件下, 才能提出更好的改进建议. 本文将这两种缺陷分开讨论, 以考察用户的参与程度以及系统对他们的重视程度.

本文的主要发现总结如下:

(1) 在人员数量上, 普通用户占总人数 75%以上. 在缺陷数量上, 普通用户提交的 defect 类型缺陷平均能达到该类型缺陷总数的 44%; 但在 enhancement 类型缺陷上, 普通用户提交的只占该类型总数的 17%. 普通用户提交的两种产品的缺陷的严重性为高等级的都不足 3%. 虽然核心开发者提交的 Firefox Desktop 产品的缺陷的严重性为高等级的也不足 5%, 但在他们提交的 Firefox for Android 产品的 defect 类型缺陷中, 仍有 25%的缺陷严重性为高等级.

(2) 在普通用户提交的缺陷中, 不能准确定位组件的缺陷平均高达 43%; 而在所有能够确定关联组件的缺陷中, 地址栏“Address Bar”和音频/视频“Audio/Video”组件上的缺陷分别占据普通用户提交的 Firefox Desktop 和 Firefox for Android 产品缺陷的第一位. 与普通用户相比, 核心开发者提交的缺陷除了与用户交互相关的 UI 组件相关外, 还存在与大量非 UI 的组件相关的缺陷(例如消息系统(messing system)和度量系统(metric)).

(3) 在已关闭的缺陷中, 普通用户提交的缺陷 32%以上为重复的缺陷, 除此之外, 还有大量信息不全以及无效的缺陷, 缺陷总体修复率低于 10%; 而核心开发者提交的缺陷 50%以上都能得到修复.

(4) 普通用户提交的处理为重复的 DUPLICATE 缺陷的标题长度要明显短于已修复的 FIXED, 在中值上前者比后者少 7 个字符; 另外, 处理为信息不全的 INCOMPLETE 的缺陷 72%评论信息中出现“需要更多信息”、“缺陷无法复现”等字样, 而处理为 FIXED 的这一比例也达到 23%. 最后, 携带附件的缺陷修复率为 11%, 而不携带附件的修复为 7%.

(5) 总体上, 普通用户和核心开发者提交的缺陷处于开放状态的为 10%–20%, 但普通用户提交的 Firefox Desktop 产品的 enhancement 类型缺陷仍有 40%左右处于开放状态. 在修复时间上, 普通用户的缺陷修复时间总体上为核心开发者的 2–3 倍.

(6) 通常, 普通用户提交的缺陷的修复流程比核心开发者提交的复杂. 例如, 前者提交的缺陷平均 90%需要先经过系统管理员确认, 而后者需要被确认的不足 2%; 此外, 普通用户提交的 defect 类型缺陷的修复者经验值与核心用户的没有明显差异. 但在 enhancement 类型缺陷上, 前者的经验值要显著低于后者.

(7) 在普通用户提交的已修复的缺陷中, 缺陷标题长度与缺陷修复时间之间没有直接关系; 而由于缺陷

报告提供的信息不全导致出现无法复现的这类缺陷, 其修复时间要比不出现不可复现的多 15 天; 同时, 在缺陷提交时携带附件有助于提交缺陷修复效率, 在修复时间上较不带附件可以缩短 7 天.

本文工作的贡献点主要包括两个方面: 第一, 构建了一个详尽的缺陷跟踪系统数据集, 这些数据不仅包括缺陷基本信息, 还有缺陷的处理历史记录、相应的报告者和修复者信息以及缺陷标题和评论信息; 第二, 从缺陷的数量、严重程度、组件分布、修复率、修复速度以及修复者等多个方面, 系统性地研究了不同角色在缺陷追踪系统中的作用和地位. 本文的研究有助于缺陷跟踪系统开发者和管理者者更好地了解当前缺陷跟踪系统面临的问题; 同时, 我们也基于文中的发现给出一些改进建议, 能够为提高实际软件开发质量以及提升普通用户对开源软件的贡献提供参考. 特别地, 为了提高缺陷跟踪系统工作效率, 减轻相关人员工作压力, 需要改善当前系统自动组件定位和查重功能, 其中, 深度学习以及多种技术的组合将会是值得借鉴的方法; 而在提高普通用户提交缺陷报告的质量方面, 缺陷跟踪系统有必要增加缺陷报告的智能填报机制, 以有效地引导用户撰写缺陷报告, 并且允许用户在线编辑缺陷报告以及上传必要的附件信息, 从而提高缺陷修复率和降低缺陷修复时间.

本文第 1 节介绍缺陷跟踪系统 Bugzilla. 第 2 节给出具体的实验设计. 第 3 节为实验结果及分析. 第 4 节分析影响本文结果有效性因素. 第 5 节介绍相关工作. 最后一节总结本文工作并指出未来的研究方向.

1 缺陷跟踪系统

随着日益增多的软件缺陷, 现代软件特别是群智化软件的维护面临着越来越大的挑战. 为了协同合作以及避免重复劳动, 目前大多数开源软件项目都使用缺陷跟踪系统(bug tracking system)来对软件缺陷进行收集、跟踪和处理. 目前流行的缺陷跟踪系统有 Bugzilla, JIRA, ITracker, FogBugz 等. 另外, 一些项目托管平台, 如 Google Code, GitHub 以及 FreeCode, 也提供了内置的缺陷跟踪系统.

Mozilla 的所有项目都使用 Bugzilla 作为其官方缺陷跟踪系统并对外开放. 目前, Bugzilla 已经由 Mozilla 的工具转化为一个通用型的缺陷跟踪系统[12]. Bugzilla 标准工作流定义了缺陷生命周期中一系列状态及状态之间转换规则. 用户也可以根据自己内部处理需求进行工作流的定制. Rocha 等人[13]总结了 Mozilla 项目的缺陷处理流程, 本文在此基础上增加了缺陷处理时使用的解决方案, 具体如图 1 所示.

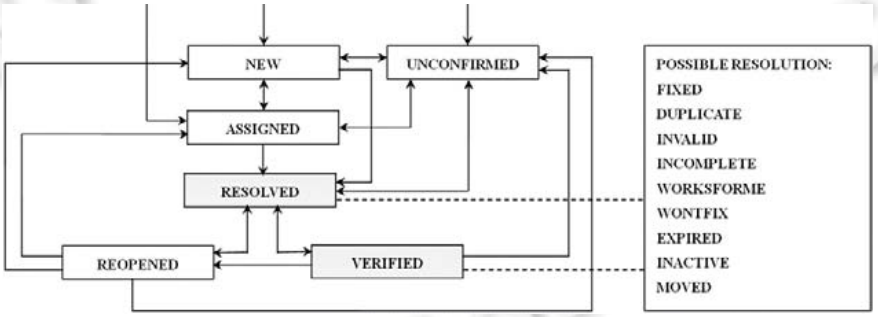


图 1 Mozilla 项目缺陷处理流程

一个典型的缺陷处理流程如下.

- 第 1 步, 用户提交缺陷, 缺陷进入 UNCONFIRMED 或者 NEW 状态.
- 第 2 步, 缺陷分配者为缺陷指定一位修复者, 缺陷进入 ASSIGNED 状态.
- 第 3 步, 修复者完成缺陷修复或者认为该缺陷不需要被修复, 缺陷进入 RESOLVED 状态.
- 第 4 步, 质量保证团队对缺陷处理结果进行验证: 如果验证通过, 则进入 VERIFIED 状态; 否则, 根据缺陷是否需求被重新确认, 分别进入 REOPENED 或 UNCONFIRMED 状态.

在以上 6 种状态中, RESOLVED 和 VERIFIED 状态被称为关闭状态, 代表缺陷已解决, 在图 1 中用灰色背景标出; 其他 4 种状态称为开放状态, 代表缺陷还未解决. 对于已解决的缺陷, 解决者还需要指明具体解决方

案. Mozilla 项目采用的解决方案主要有 8 种: FIXED(已修复的), DUPLICATE(重复的), INCOMPLETE(信息不全的), INVALID(无效的), WONTFIX(不予修复的), WORKSFORME(不能复现的), INACTIVE(不活跃的), MOVED(已被移除的). 在这 8 种缺陷中, 只有 FIXED 类型的缺陷涉及源码修改, 其他 7 种可认为是“无用的”, 它们对软件质量提升不起作用.

另外, 根据缺陷性质, Bugzilla 将缺陷分成 3 种类型: defect, enhancement 以及 task. defect 类型缺陷主要是指软件产品功能受损、安全漏洞、崩溃、死机、挂起等问题; enhancement 类型缺陷主要指新增功能需求、UI 界面及性能的改进等建议; 而 task 类型缺陷主要指重构、移除、替换以及其他相关工程任务.

2 实验设计

2.1 实验对象

本文选取 Mozilla 两个产品 Firefox Desktop 以及 Firefox for Android 作为研究对象. Firefox 是由 Mozilla 基金组织于 2002 年开发的一个可运行在多操作系统平台(如 Windows, Mac OS, Linux)的开源 Web 浏览器. 目前, 除了桌面浏览器外, Mozilla 自 2010 年起又先后发布多种移动端浏览器, 如“Firefox for Android”“Firefox for iOS”等. 如今, Firefox 已被分成 10 多个模块[14], 开发者们通过协同合并共同构建 Firefox 生态系统. 目前, Bugzilla 以及 Firefox 已成为软件工程领域众多工作的研究对象[15−18].

本文选取 Firefox Desktop 以及 Firefox for Android 作为研究对象, 主要基于以下 3 个原因.

- 第一, Mozilla 项目使用的缺陷跟踪系统 Bugzilla 是一个通用型系统, 具有一定的代表性, 众多开源和商业软件如 Apache, Linux, Eclipse, NASA 以及 Facebook 都使用它进行缺陷管理.
- 第二, Firefox 是一个非常成功的项目, 有着悠久的发展历史, Bugzilla 系统中存储了大量有关它的缺陷信息且可公开访问.
- 第三, Firefox Desktop 以及 Firefox for Android 来自不同领域: 一个是桌面应用, 另一个是移动端应用. 这有利于本文结果的推广.

需要说明的是, 在 Bugzilla 系统中, Firefox 桌面版名称直接为 Firefox, 本文称其为 Firefox Desktop, 以便区别与手机移动端 Firefox for Android.

2.2 数据数据集及数据收集方法

为了回答 RQ1–RQ7, 我们编写了 Python 自动化脚本, 从 Mozilla 的 Bugzilla 系统中爬取了 2018 年、2019 年提交的 19 474 份 Firefox Desktop 缺陷报告和 3 057 份 Firefox for Android 缺陷报告. 其中, 对于每份缺陷报告, 我们获取了包括缺陷的基本信息、历史处理记录、报告者、修复者以及标题和评论等在内的全部信息, 上述信息在 Mozilla 的 Bugzilla 系统中都是可以公开访问和获取的. 在 2020 年 6–9 月期间多次执行了缺陷报告爬取脚本(平均一次执行大约需 3 天), 同时编写代码自动对这些数据进行分析, 以确保我们收集的缺陷报告没有遗漏和错误. 为了方便相关研究人员复现并进一步扩展本文研究发现, 我们已将这些数据以及数据爬取脚本都上传至 GitHub (https://github.com/GIST-NJU/BUGS_FIREFOX_2018-2019). 为了能够比较准确地反映缺陷跟踪系统当前最新工作情况, 这里选取的是最近两年即 2018 年和 2019 年提交的缺陷; 我们不考虑 2020 年提交的缺陷, 主要考虑到这些缺陷在系统中存在的时间相对较短, 目前还处于不稳定状态, 对实验结果有效性会造成一定的影响.

下面, 我们对研究问题涉及的数据及其收集方法进行简要介绍.

- 缺陷基本信息

本文涉及的缺陷基本信息有缺陷类型、创建时间、修复完成时间、严重性、所属组件、现有状态以及解决方案等. 在上述有关缺陷的基本信息中, 除缺陷修复完成时间外, 其他信息均可由 Bugzilla 提供的 REST API 获取, 结果为 JSON 格式, 缺陷修复完成时间确定方法稍后再做说明. 图 2 为获取到的 1471532#缺陷的基本信息. 为了显示清晰, 图中剔除了一些无关数据.

- 缺陷状态转换过程

缺陷状态转换过程描述了缺陷处理过程中所经历的状态，它可从缺陷处理历史记录中提取. 具体地，将历史处理记录中的 what 字段限定为“Status”或者“Resolution”，即可提取其中的状态转换信息. 而缺陷历史处理记录可根据缺陷 id 从 Mozilla 的 Bugzilla 官网页面爬取. 图 3 为 1471532#缺陷的状态转换过程.

{"bugs":[{"id":1471532, "product":"Firefox","component":"General",
"type":"enhancement",
"severity":"normal",
"resolution":"FIXED",
"status":"RESOLVED",
"creation_time":"2018-06-27T09:18:26Z",
"cf_last_resolved":"2018-07-03T09:09:58Z",
"creator_detail":{"id":395101, "email":"choller@mozilla.com","nick":"decoder",
"name":"choller@mozilla.com","real_name":"Christian Holler (:decoder)" },
"assigned_to_detail":{"id":395101,"email":"choller@mozilla.com","nick":"decoder",
"name":"choller@mozilla.com","real_name":"Christian Holler (:decoder)"}}]}

图 2 缺陷 1471532 的基本数据

bugid=1471532

When	What	Removed	Added
2018/6/29	Status	ASSIGNED	RESOLVED
	Resolution	---	FIXED
2018/6/29	Status	RESOLVED	REOPENED
	Resolution	FIXED	---
2018/7/3	Status	REOPENED	RESOLVED
	Resolution	---	FIXED

图 3 缺陷 1471532 的状态转换

有了缺陷状态转换信息就可以来确定缺陷修复完成时间. 通常情况下，缺陷修复完成时间就是该缺陷的解决方案被修改为 FIXED 的时间，但需要注意的是，在修复过程中，缺陷可能会被多次修复. 例如, 1471532#的缺陷曾先后两次分别在“2018/6/29”和“2018/7/3”被修复(图 3 中用虚线框标出). 缺陷被多次修复的原因有很多，例如缺陷报告者对缺陷的描述不够准确或者不全面，或者缺陷处理者未能正确地找出缺陷发生的原因，再有，软件运行环境发生改变导致以前被修复的缺陷再次出现等. 在缺陷被多次修复的情况下，本文选取解决方案最后一次被修改为 FIXED 的时间作为缺陷修复完成的时间. 例如，上述 1471532#缺陷修复完成时间确定为“2018/7/3”.

需要注意的是，缺陷基本信息包含了一字段 cf_last_resolved，它仅表示缺陷状态最后一次被修改为 RESOLVED 的时间，而不能作为缺陷修复完成的时间，因为缺陷的解决还涉及解决方案，只有 FIXED 类型的解决方案才表示缺陷被修复. 这里以 1444845#缺陷为例进行说明. 从图 4 中可以看出，该缺陷状态最后一次被修改为 RESOLVED 是在“2018/7/3”,该时间就是 “cf_last_resolved”的值.但是此时缺陷的解决方案是 WONTFIX,也就是不进行缺陷修复,直到“2018/7/24”解决方案才为 FIXED.因此,该缺陷的修复时间应该为“2018/7/24”而不是“2018/7/3”. 综上所述,无论哪种情况,缺陷修复完成时间都可统一为解决方案最后一次被修改为 FIXED 的时间.这样缺陷创建时间与缺陷修复完成时间之间的差值能够较为准确地反映缺陷修复实际需要的时间.

- 报告者信息

本文涉及报告者的信息包括报告者名称以及报告者权限. 报告者名称用于识别报告者，从图 2 可以看出，缺陷基本信息提供了“creator_detail”字段，该字段为报告者详细信息，包括用户 id、邮箱名、昵称、用户名以及真实姓名等. 目前，大部分研究[17,19,20]都采用邮箱名来识别用户，该方法的有效性将在本文第 4 节进行讨论. 为了方便数据获取，本文采用用户 id 识别用户，它与邮箱具有一一对应关系. 报告者权限用于报告者分类，它将报告者分成两类: 普通用户和核心开发者. 普通用户无特殊权限，只能进行缺陷提交和评论；而核心开发者除了提交评论缺陷外，还可以进行缺陷确认编辑等工作. Mozilla 的 Bugzilla 仓库中存储了所有已注册的用户个人档案(profile)，个人档案提供了用户权限信息字段 Permissions，Permissions 共 3 种取值: “Can confirm bugs, can edit any bugs”, “Can confirm bugs”以及空值. 本文将权限取值为前两种的用户称为核心开发者，他们具有一定的缺陷操作特权，为 Mozilla 组织的核心人员；而后一种称为普通用户，无特权，他们通常为 Mozilla 产品的使用者. 图 5 为 id=395101 用户的档案，其中, Permissions 字段取值为“Can confirm bugs, can edit any bugs”，表明该用户为核心开发者. 用户个人档案可以根据用户 id 直接从 Mozilla 的 Bugzilla 官网页面进行爬取.

bugid=1444845			
When	What	Removed	Added
2018/3/15	Status	NEW	RESOLVED
	Resolution	---	DUPLICATE
2018/3/15	Status	RESOLVED	REOPENED
	Resolution	DUPLICATE	---
2018/7/3	Status	REOPENED	RESOLVED
	Resolution	---	WONTFIX
2018/7/24	Resolution	WONTFIX	FIXED

图 4　缺陷 1444845 的状态转换

图 5　用户 395101 的档案

- 修复者信息

本文涉及修复者的信息包括修复者名称以及修复者被指派修复任务的次数. 修复者名称用于识别修复者, 修复者被指派任务数用来衡量修复者经验值. 与缺陷报告者相同, 本文使用用户 id 作为缺陷修复者名称以识别缺陷修复者, 它可从缺陷基本信息的 assigned_to_detail 字段中获得. 另外, 无论是缺陷报告者还是缺陷修复者, 他们都是 Bugzilla 中已注册的用户, 因此修复者的个人档案也可以通过其 id 获取, 档案中的 Assigned to 字段就是该用户被指派修复任务的次数. 例如, 图 5 描述的用户共被指派修复任务 368 次.

- 标题和评论信息

标题是对提交的缺陷的简要概述, 评论是缺陷报告者和缺陷修复者以及其他参与者在缺陷处理过程中交流产生的信息. 评论信息的第 1 条是缺陷报告者对缺陷的详细描述, 包括产生缺陷的软件版本、缺陷复现步骤、实际结果以及预期结果等内容. 图 6 和图 7 分别为 1427924#缺陷的标题以及部分评论信息, 它们都可由 Bugzilla 提供的 REST API 获取.

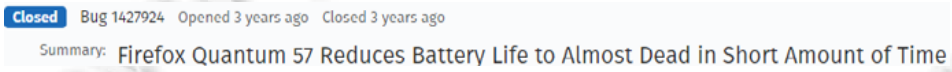


图 6　1427924 缺陷标题信息

图 7　1427924 缺陷评论信息

3 实验结果及分析

本节给出 RQ1–RQ7 在 Firefox 缺陷跟踪系统上的调查结果以及对结果的分析和讨论.

3.1 用户数量和缺陷数量以及缺陷严重程度(RQ1: 普通用户与核心开发者在参与人数、提交缺陷个数以及提交缺陷的严重等级方面是否存在明显区别?)

在收集的 19 474 份 Firefox Desktop 缺陷和 3 057 份 Firefox for Android 缺陷中, 我们将它们的缺陷报告者按照个人档案中的 Permission 字段取值分为核心开发者和普通用户两类(划分标准参见第 2.2 节). 表 1 给出了两种不同类型缺陷报告者数量.

表 1 缺陷报告者数量

产品	报告者类型	
	核心开发者	普通用户
Firefox Desktop	766	4 633
Firefox for Android	240	777

从表 1 可以看出: 无论是桌面版还是手机移动端, Firefox 缺陷报告者中的普通用户数量远远大于核心开发者数量. 具体地, 在桌面版软件产品中, 前者是后者的 6 倍左右. 相对于桌面版, Android 移动端系统规模要小些, 加上其开发时间较晚, 因此缺陷报告者数量总体上比桌面版要少些, 但普通用户仍占到总报告者的 75%. 这一结果表明: 无论是 Firefox Desktop 还是 Firefox for Android, 普通用户参与度都很高.

(1) 缺陷数量

图 8 为两种不同类型报告者提交的 defect, enhancement 以及 task 类型缺陷数量.

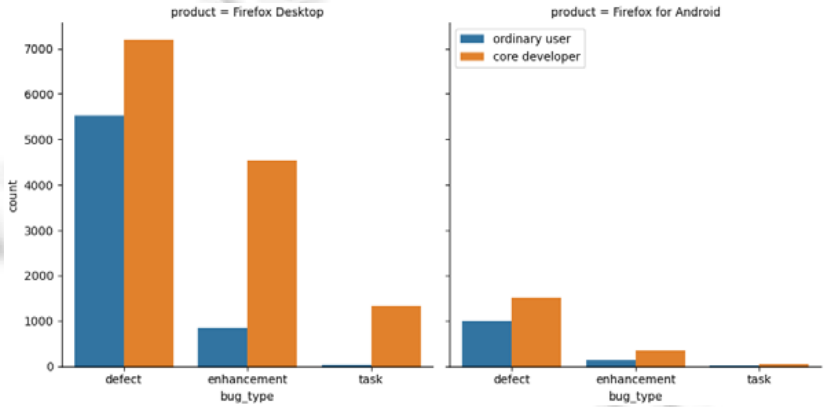


图 8 缺陷数量

从图 8 可以看出: 在两软件产品中, 无论是普通用户还是核心开发者, 他们提交的 defect, enhancement 以及 task 这 3 种类型缺陷数量都呈现出依次递减的趋势. 但普通用户提交的缺陷在这点上表现得更为明显. 例如, 普通用户提交的 Firefox Desktop 产品的 defect 类型的缺陷高达 5 000 多份, 但提交的 enhancement 类型缺陷不足 1 000 份, 这也与核心开发者提交的 4 000 多份 enhancement 类型的缺陷形成鲜明对比. 特别地, 在两产品中, 就 defect 类型缺陷而言, 普通用户提交的与核心开发者提交的缺陷各约占 50%(44% vs 56%), 但是在 enhancement 类型缺陷上, 前者提交的缺陷只占该类型缺陷总数的 17%. 这反映出大多数普通用户仅限于报告软件产品使用过程中遇到的异常问题, 而较少关注如何增强软件功能或者提高软件质量. 也就是说, 普通用户参与的程度还较浅. 进一步分析我们还发现: 无论是 Desktop 和 Android 产品, 86%左右的普通用户仅发布过一条缺陷, 只有 2%左右的普通用户才发布 5 条及以上的缺陷. 这说明绝大多数普通用户不是开源社区的长期贡献者. 由于两类报告者提交的 task 类型缺陷都很少, 特别是普通用户, 他们提交的只有数十个, 这与 task 类型缺陷涉及的系统重构等工程问题相关. 基于 task 类型缺陷数量较少不具备统计意义, 在接下来的研究问

题中, 在没有做特殊说明的情况下, 我们只涉及 defect 和 enhancement 类型缺陷.

(2) 缺陷严重程度

由图 8 可以看出: 普通用户提交了大量的缺陷, 特别是 defect 类型的. 接下来, 我们进一步调查这些缺陷的严重程度. 这里, 严重程度用来描述缺陷对软件产品产生的影响, 它是衡量缺陷重要性的一个指标.

在 Mozilla 的 Bugzilla 系统中, defect 类型缺陷的严重程度由高到低共分为 6 个等级: block, critical, major, normal, minor, trivial. 其中, block 等级最高, 该等级缺陷会阻碍系统开发和测试; 而 trivial 等级最低, 该等级缺陷通常是 UI 展示上的细微问题(如单词拼写错误或者文本未对齐), 基本上不影响软件功能的使用. enhancement 类型缺陷的严重程度除了以上 6 种等级外, 还有 enhancement 以及 N/A (not application)两种, 其中, N/A 表示目前已有的严重等级取值不适用, 它是 enhancement 类型缺陷严重程度的预留值.

在以上 8 种缺陷等级中, 通常将 block, critical 和 major 称为高严重等级, 其他几种称为低严重等级[21]. 表 2 给出了两种不同类型报告者发布的缺陷严重等级分布情况, 由于 defect 类型缺陷不存在 enhancement 及 N/A 等级严重程度, 在表 2 中用“–”表示.

表 2 缺陷严重性分布

产品名称	报告者类型	缺陷类型	严重程度(%)							
			高严重等级			低严重等级				
			blocker	critical	major	normal	minor	trivial	enhancement	N/A
Firefox Desktop	普通用户	defect	0.20	1.30	1.32	**95.55**	1.25	0.38	–	–
		enhancement	0.00	0.12	0.59	**96.79**	1.31	1.07	0.00	0.12
	核心开发者	defect	0.61	1.96	1.51	**93.22**	2.46	0.22	–	–
		enhancement	0.00	0.11	0.29	**98.68**	0.55	0.13	0.07	0.18
Firefox for Android	普通用户	defect	0.40	1.21	1.42	**96.56**	0.40	0.00	–	–
		enhancement	0.00	0.72	0.00	**99.28**	0.00	0.00	0.00	0.00
	核心开发者	defect	0.33	**23.36**	1.77	**74.08**	0.39	0.07	–	–
		enhancement	0.58	0.58	1.73	**95.95**	0.87	0.29	0.00	0.00

从表 2 可以看出:

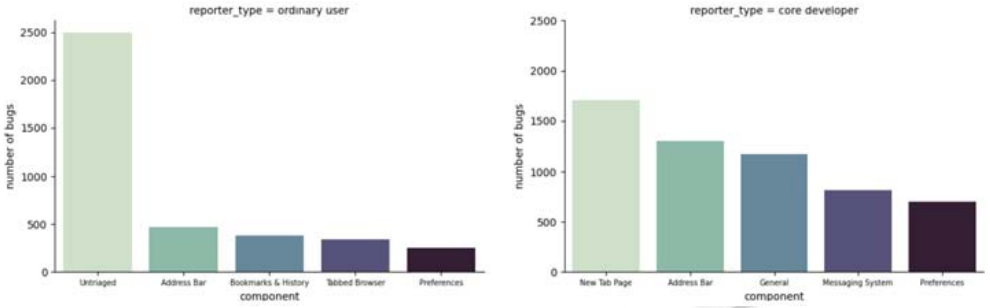
- 首先, 对于 Firefox Desktop, 普通用户和核心开发者提交的两种类型缺陷 90%以上都是 normal 级别(表中加粗表示), 高严重等级的缺陷不超过 5%. 这与 Firefox Desktop 是一个较成熟的软件产品、不易发现高严重缺陷相关. 即使如此, 两种不同类型报告者在高严重等级缺陷人均数量上还是存在一定差距. 例如, 普通用户提交的 defect 类型的高严重等级缺陷人均 0.03 个(156/4633), 而核心开发者提交的该类型的高严重等级的缺陷人均 0.38 个(294/766).
- 其次, 对于相对较新的 Firefox for Android 产品, 虽然普通用户和核心开发者提交的 enhancement 类型缺陷 95%以上都为低严重等级, 但在 defect 类型缺陷上, 两者差异明显. 具体而言, 普通用户提交的 defect 缺陷只有 3%左右为高严重等级, 但是核心开发者提交的高达 25%, 其中, critical 等级占 23%.

结论: 在 Firefox 缺陷跟踪系统中, 普通用户报告者人数众多, 且能够积极提交软件产品使用过程中遇到的 defect 类型缺陷, 但在 enhancement 类型缺陷上还存在不足, 所提交的 enhancement 类型缺陷的数量与核心开发者的差距较大. 另外, 无论在 Firefox Desktop 还是 Firefox for Android 中, 绝大多数(约 86%)普通用户都不是长期贡献者, 且他们提交的缺陷严重程度都较低, 这些都反映了普通用户参与程度较浅.

3.2 缺陷在组件上的分布(RQ2: 普通用户与核心开发者提交的缺陷分别集中在哪些组件上? 这些组件在分布上是否存在明显区别?)

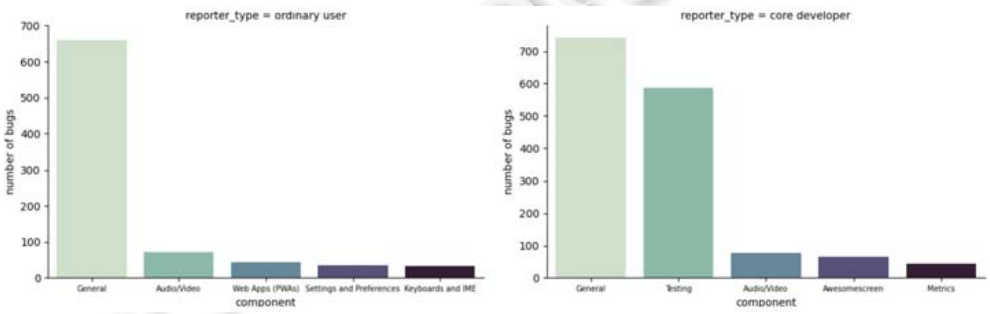
缺陷提交到系统后, 缺陷分类人员需要为每个缺陷指定一个与其关联的组件, 以方便缺陷定位. 组件的个数及类别与软件产品相关, Mozilla 的 Bugzilla 系统为 Firefox Desktop 产品定义了包括“about: logins” “Activity Stream: General”, …, “Web Payments UI”等在内的 52 个组件, 涉及安装、登录、迁移、地址栏、菜单、活动流、书签、分页、搜索、下载、安全、会话恢复以及消息系统等模块; 为 Firefox for Android 产品定义了包括“Activity Stream”“Add-on Manager”, …, “Web Apps (AWPs)”等在内的 39 个组件, 涉及安装、登录、插件管理、音频/视频、智能屏、键盘、文本输入、主题、工具栏、搜索、下载以及度量系统等若干模块.某些缺

陷由于报告者提供的信息不全无法确定其确定组件, 此时, 组件属性被指定为“Untriaged”; 还有些缺陷由于分类到目前已有的组件都不合适, 此时, 组件属性被指定为“General”. 图 9 和图 10 分别给出了 Firefox Desktop 和 Firefox for Android 两种产品缺陷数量排名前五的组件. 统计时, 本文将“Untriaged”和“General”也包括进来.



(a) 普通用户提交的缺陷 (b) 核心开发者提交的缺陷

图 9 Firefox Desktop 中缺陷数量排名前五的组件



(a) 普通用户提交的缺陷 (b) 核心开发者提交的缺陷

图 10 Firefox for Android 中缺陷数量排名前五的组件

从图 9 可以看出: 在 Firefox Desktop 中, 未能进行组件分类的缺陷远远高于其他类型的缺陷. 该类缺陷通常是由于报告者提供的信息不全引起的. 特别地, 在普通用户提交的缺陷中, 该类缺陷比例高达 39%, 这反映出普通用户在撰写缺陷报告方面的不足. 此外, 该产品的地址栏“Address Bar”组件以及与标签页有关的组件, 如新标签页“New Tab Page”、分页浏览器“Tabbed Browser”, 两类报告者都提交了大量相关缺陷, 同时, 这两类组件上的缺陷数量也是已确定组件的缺陷中最多的. 然而在一些非用户交互的 UI 组件上, 两类报告者存在较大差别. 例如, 与消息系统“Message System”有关的缺陷分别排在普通用户和核心开发者提交的缺陷中的第 34 位和第 4 位.

在 Firefox for Android 产品中(如图 10 所示), 被划为“General”的缺陷在普通用户提交的缺陷中占有绝对优势, 高达 58%. 同时, 39%核心开发者提交的该产品缺陷也被划分为“General”, 这要远远高于核心开发者在 Firefox Desktop 对应的 9%. 其中原因有待进一步调查. 不过, 在 Firefox for Android 产品中未定义“Untriaged”类型, 因此这里不排除这类缺陷中一部分是因为信息不全无法进行分类造成的. 同时, 从核心开发者提交的缺陷包含大量的“Testing”缺陷(约 30%)可以看出, Firefox for Android 产品还不够成熟, 因此组件定义也有可能待进一步完善.

进一步观察发现, 与用户交互的 UI 组件“Audio/Video”缺陷同时排在两类用户提交的 Firefox for Android 产品缺陷之前, 它是除了“General”和“Testing”之外, Firefox for Android 产品缺陷最多的组件. 此外, 与 Firefox Desktop 相似, 与普通用户相比, 核心用户仍能发现一定比例非 UI 组件上的缺陷. 例如, “Metric”上的缺陷位

列核心开发者提交的 Firefox for Android 产品缺陷的第 5 位, 但在普通用户提交的该产品缺陷中只排在第 22 位.

结论: 软件产品中与用户交互的 UI 组件较容易出错, 普通用户提交的缺陷主要集中在这些组件上, 而其他类型组件上的缺陷主要还是由核心开发者贡献. 同时, 由于提供的信息不充分, 普通用户提交的缺陷还存在大量的无法确定合适的组件的缺陷.

3.3 缺陷处理结果(RQ3: 普通用户与核心开发者提交的缺陷在处理结果上是否存在明显区别?)

缺陷跟踪系统的一个重要作用是进行缺陷修复, 但由于多种原因, 如被报告的缺陷已经存在或者报告者给出的信息不充分, 因此并不是每个被报告的缺陷都能得到修复. 本节将分析在已关闭的缺陷中得到修复的缺陷的比例, 即缺陷修复率. 为此, 我们提取缺陷基本信息中的解决方案 resolution 字段值. 表 3 为各种解决方案在已解决的缺陷中占比情况, 表中加粗部分为已修复的缺陷比例. 注意, 表3中的数据是采用各种解决方案的缺陷在已关闭的缺陷中的比例, 也就是它不包括还处于开放状态的缺陷.

表 3 缺陷解决方案

产品	报告者类型	缺陷类型	解决方案(%)			
			FIXED	DUPLICATE	INCOMPLETE	INVALID
Firefox Desktop	普通用户	defect	**8.46**	30.58	22.33	18.02
		enhancement	**9.61**	40.39	8.04	11.57
	核心开发者	defect	**51.37**	16.59	17.91	5.79
		enhancement	**83.49**	7.15	0.55	2.60
Firefox for Android	普通用户	defect	**8.48**	38.10	9.11	17.72
		enhancement	**7.32**	26.83	2.44	14.63
	核心开发者	defect	**30.07**	19.88	33.14	4.14
		enhancement	**75.20**	11.02	0.79	3.15

产品	报告者类型	缺陷类型	解决方案(%)			
			WONTFIX	WORKSFORM	INACTIVE	MOVED
Firefox Desktop	普通用户	defect	3.76	16.01	0.74	0.10
		enhancement	24.51	5.49	0.39	0.00
	核心开发者	defect	3.79	4.36	0.17	0.03
		enhancement	4.71	1.22	0.28	0.00
Firefox for Android	普通用户	defect	4.56	13.67	7.47	0.89
		enhancement	37.80	6.10	3.66	1.22
	核心开发者	defect	4.89	7.62	0.00	0.25
		enhancement	5.91	2.76	0.39	0.79

表 3 中数据表明, 普通用户发布的缺陷修复率远远低于核心开发者的: 前者不超过 10%; 而对于后者, 除了 Firefox for Android 的 defect 类型缺陷的 30%外, 其他的都在 50%以上甚至超过 80%.

进一步观察还可以发现: 在普通用户发布的缺陷中, DUPLICATE(重复的)缺陷比重很大, 在 30%左右; 另外还有大量的 INCOMPLETE(信息不全的), WONTFIX(不予修复的)缺陷, 可以最高可达 38%.

结论: 普通用户提交的缺陷中存在大量重复的、信息不全的、不予修复的缺陷, 这大大降低了普通用户提交的缺陷的修复率. 这些反映了普通用户提交的缺陷报告质量有待进一步加强.

3.4 缺陷报告与缺陷处理结果关系(RQ4: 在缺陷报告上, 哪些因素会对缺陷处理结果产生影响?)

从 RQ3 结果可以看出, 在普通用户提交的缺陷中存在大量“无用”缺陷, 如 DUPLICATE, INCOMPLETE, WONTFIX 等等, 本节将从缺陷报告的标题和描述以及是否携带附件等方面进行分析.

(1) 缺陷标题

RQ3 结果表明: 在普通用户提交的缺陷中, 处理为 DUPLICATE 类型的缺陷占有非常大的比值, 可高达 40%, 远高于核心开发者. 经调查, 在 Mozilla 产品的缺陷跟踪系统中, 用户发现缺陷后, 需要先进行查重, 查重界面如图 11 所示. 从图 11 可以看出, 查重的依据是缺陷的概述信息, 即缺陷标题.

下面我们将对比普通用户提交的处理为重复的 DUPLICATE 和其他解决方案如 FIXED 缺陷的标题长度, 以确定重查结果是否与缺陷概述详细程度有关, 结果如图 12 所示.

图 12 显示, DUPLICATE 的缺陷的标题长度要短于 FIXED 的. 具体而言, 两者的中位数分别为 59 和 66, 均值分别为 64.0 和 70.9. 采用双边 Mann-Whitney U 进行检验, 结果为 *p-value*=1.79e–8, 即在统计意义上, 两者具有明显差异.

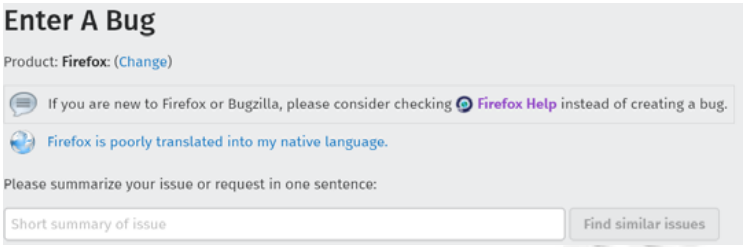


图 11 查重界面

由此可以得出: 缺陷标题描写详细程度对普通用户查重结果具有一定影响, 适当增加对缺陷的描述, 有利于提高查重准确率.

(2) 缺陷描述

在普通用户提交的缺陷中, 除了处理为 DUPLICATE 占有很大比例外, 还有一类 INCOMPLETE 缺陷也占有相当大的比例, 如在 Firefox Desktop 产品中, 普通用户提交的 defect 类型的缺陷中, 处理为 INCOMPLETE 约 22%. 如果减少这种缺陷, 普通用户提交的缺陷修复率将会大大提高.

INCOMPLETE 解决方案的缺陷是因为报告者对缺陷的描述不够详细, 导致缺陷修复者无法复现这些缺陷. 因此, 下面我们对缺陷的评论信息进行分析, 查看其是否包含“lack of information from the reporter”, “not reproducible”等关键词, 以确定报告者对缺陷描述的不够充分, 不能复现缺陷. 统计结果如图 13 所示.

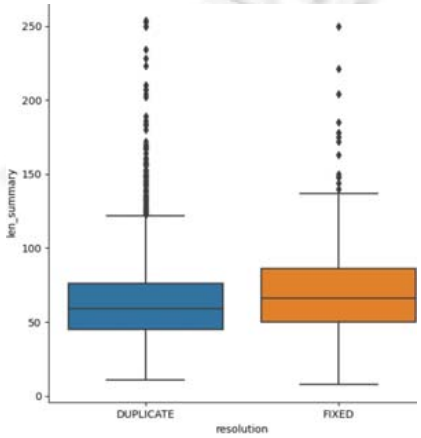


图 12 缺陷标题长度对比

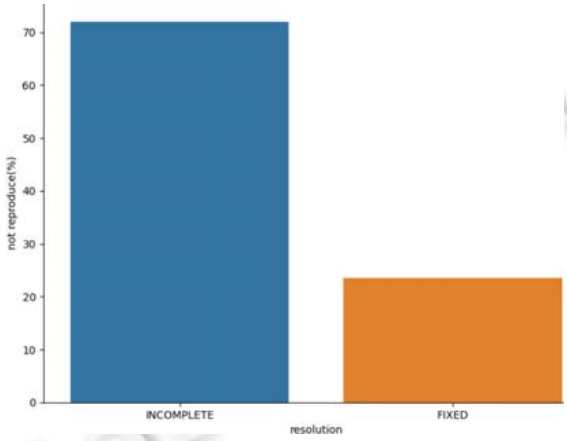


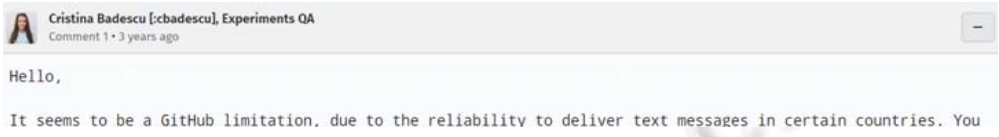
图 13 缺陷不能复现率对比

图 13 显示, 在普通用户提交的缺陷中, 处理为 INCOMPLETE 的缺陷, 其处理者曾出现不可复现该缺陷的约 72%, 即使处理为 FIXED 的缺陷, 这一数目也达到了 23%, 说明大量普通用户在缺陷的描述能力方面还有所欠缺, 提供的信息不够充分和准确, 导致缺陷处理者无法复现这些缺陷, 降低了缺陷修复率.

说明: 经初步调查, 我们从普通用户提交的缺陷评论信息中共选取了 25 个表达信息不充分导致缺陷不能复现的关键词, 如“not reproducible”“cannot reproduce”“unable to reproduce”“can you reproduce”等. 由于可能还存在其他表达方式, 因此图 13 中的结果可能与实际情况存在一定偏差, 但可以肯定的是, 实际结果应该比图 13 中的结果要大, 也就是在实际中, 普通用户提交的缺陷会有更多在其处理过程出现过信息不全、不能复现等情况. 在未来的工作中, 我们将利用机器学习的方法进行语义分析, 以提高结果准确率.

从表 3 可以看出: 除了 DUPLICATE, INCOMPLETE 以及 WORKSFORM 等无用缺陷外, 在普通用户提交的缺陷中, 还存在一大部分 INVALID 和 WONTFIX 等类型缺陷. 经调查, 这两类缺陷是主要由于用户不熟悉 Firefox, 对出现的问题定性错误, 或者不正确操作导致某些问题出现.

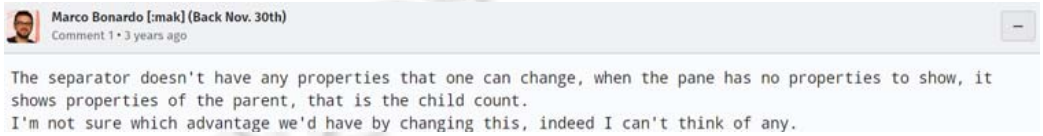
例如, 图 14 为两个 INVALID 缺陷示例, 其中, 图 14(a)为 1427613#的缺陷处理者回复信息, 该信息显示, 报告者所遇到的问题是由于 Github 平台引起的而非 Firefox 本身问题; 图 14(b)为 1430174#缺陷报告者根据该缺陷处理者建议操作后的回复信息, 该信息显示, 以前出现的问题消失了. 图 15 为两个 WONTFIX 类型缺陷示例, 它们的处理者认为: 报告者提交的问题并不是真正问题, Firefox 本身就该如此, 因此不需要修复.



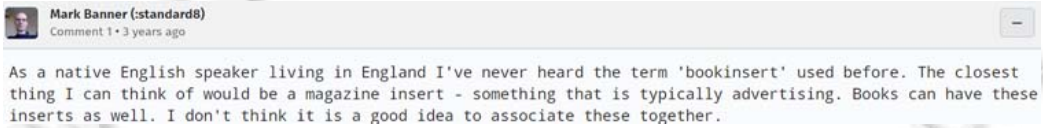
(a) 1427613 的缺陷评论信息

(b) 1430174 的缺陷评论信息

图 14 INVALID 缺陷示例



(a) 1433380 的缺陷评论信息



(b) 1433713 的缺陷评论信息

图 15 WONTFIX 缺陷示例

(3) 附件

在 Firefox 的缺陷跟踪系统中, 用户在提交缺陷时可以携带附件以辅助缺陷描述. 下面我们将调查携带附件是否有助于提高缺陷修复率, 具体方法为: 我们将普通用户提交的已处理完的缺陷按是否携带附件分成两类, 然后分别统计每类的缺陷修复率, 即处理为 FIXED 的在总体中占的比例. 统计结果为: 携带附件的缺陷修复率要略高于不带附件的, 两者分别为 11%和 7%.

结论: 无论是对缺陷的概述(即标题)还是描述, 普通用户都不能提供充分准确的信息, 导致了大量 DUPLICAT 以及 INCOMPLETE 等缺陷. 还有一部分普通用户由于对 Firefox 的不熟悉或者不正确操作, 造成 INVALID 和 WONTFIX 类型缺陷的出现. 与此同时, 携带附件能够帮助用户描述缺陷, 这是一种增加缺陷修复率的有效途径.

3.5 缺陷修复速度(RQ5: 普通用户与核心开发者提交的缺陷在缺陷处理的优先级以及缺陷修复时间上是否存在明显区别?)

Mozilla 的 Bugzilla 系统是个大型开放性缺陷跟踪系统, 每天面对大量提交的缺陷, 必然有一部分缺陷得不到及时处理. 为了了解缺陷是否被处理完, 我们从缺陷的基本信息中提取字段 status, 它反映了缺陷目前处于的状态, 其中, 处于 UNCONFIRMED, NEW, ASSIGNED 以及 REOPED 状态的缺陷表示该缺陷还未处理完,

它们仍处于开放状态; 而处于 RESOLVED 和 VERIFIED 状态的缺陷已处理完, 它们处于关闭状态. 图 16 给出了截止本文数据获取时间, 普通用户和核心开发者提交的缺陷处于开放状态的占比情况.

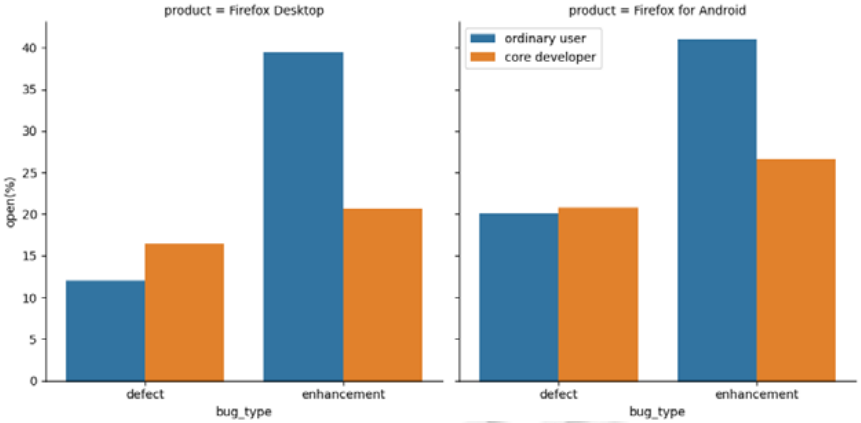


图 16 开放缺陷百分比对比

从图 16 容易看出: 在两产品中, 对于 defect 类型缺陷, 两种类型报告者提交的缺陷未解决率相差不大, 都不超过 20%, 甚至普通用户还要稍低些; 但是在 enhancement 类型缺陷上, 普通用户提交的缺陷未解决率高达 40%左右, 而核心开发者的在 20%–25%之间, 两者之间有一定差距. 这一定程度上反映了系统更偏向处理核心开发者发布的 enhancement 类型缺陷.

接下来, 我们进一步分析普通用户提交的已得到修复的缺陷需要的修复时间与核心开发者的是否有明显区别.

本文将缺陷的修复时间(bug fixing time)定义为

$$BFT=T_{fixed}-T_{submitted},$$

其中, T_{fixed} 为缺陷修复完成时间(计算方法参见第 2.2 节); $T_{submitted}$ 表示缺陷提交时间, 其取值为缺陷基本信息中的 creation_time 字段. 图 17 给出了缺陷修复时间箱线图, 图中的纵坐标为修复时间的对数.

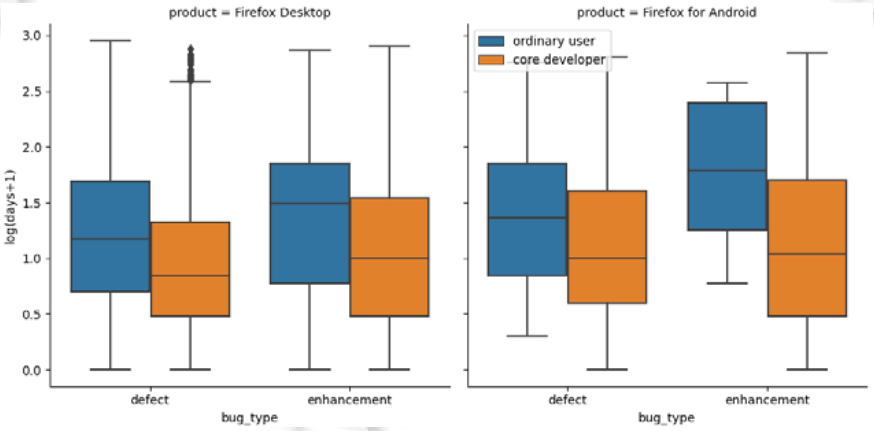


图 17 缺陷修复时间对比

从图 17 可以看出: 总体上, Firefox Desktop 中的缺陷修复时间要短于 Firefox for Android 中对应类型缺陷的修复时间. 这可能与前者开发早于后者、前者的缺陷修复者经验要高于后者的有一定关系. 另外, 在这两种产品中, 普通用户提交的缺陷比核心开发者提交的缺陷需要更长的修复时间, 这一发现与预期一致. 我们还

可以观察到: 相比 defect 类型, 普通用户和核心开发者两者在 enhancement 类型上的差距更大. 例如在 Firefox Desktop 中, 普通用户和核心开发者提交的 defect 类型缺陷需要的修复时间中位数分别为 14 天和 6 天, 两者相差 8 天; 而两者提交的 enhancement 类型缺陷的修复时间中位数分别为 30 天和 9 天, 两者之差增大到 21 天.

进一步地, Mann-Whitney U 检验结果(*p-value*<0.003)显示, 普通用户提交的缺陷的修复时间要明显大于核心开发者提交的缺陷的修复时间.

结论: 普通用户提交的 enhancement 类型缺陷未能得到系统充分重视, 仍处于开放状态比例很大(约 40%). 同时, 普通用户提交的缺陷修复时间要大于核心开发者提交的, 两者修复时间中值差在 8 天及以上.

3.6 缺陷修复者和修复流程(RQ6: 普通用户和核心开发者提交的缺陷在缺陷修复者的指定以及缺陷修复流程上是否存在明显差别?)

RQ5 结果表明: 普通用户和核心开发者提交的缺陷需要的时间存在一定的差距, 特别是在 enhancement 类型缺陷上, 两者的差距能够达到 10 天以上. 那么这两类缺陷的修复者是否也不同? 具体来说, 不同的类型报告者提交的缺陷其修复者的修复经验是否不同? RQ6 针对这一问题进行调查, 采用修复者被指派过的修复任务数来衡量修复者的经验值, 即用修复者个人档案中的 Assigned to 字段来表示修复者的经验.

需要说明的是, 一部分缺陷在指定具体修复者之前就完成了修复(即没经过 ASSIGNED 状态直接到达 RESOLVED 状态). 据观察, 这部分缺陷的修复者邮箱为“nobody@mozilla.org”, 用户 id 为 1. 在 RQ6 问题中, 我们排除这部分缺陷, 也就是只在已修复的缺陷中选取用户 id 不为 1 的作为研究对象, 提取其中每条缺陷修复者的被指派的任务数, 即 Assigned to 字段值进行分析. 修复者被指派任务数统计结果如图 18 所示.

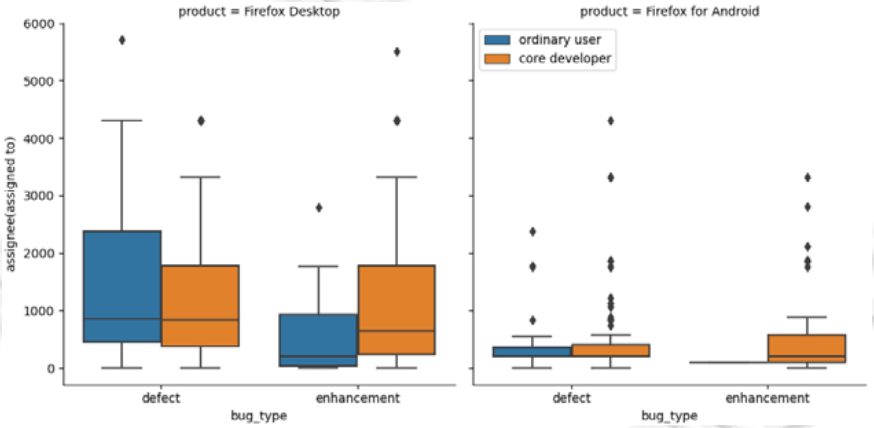


图 18 缺陷修复者对比

图 18 的纵坐标为修复者被指派过修复任务的次数. 从图 18 中直接观察: 在 defect 类型上, 两类报告者提交的缺陷的修复者被指派的任务数中值没有明显差别, 甚至普通用户的平均值要高于核心用户的. 但在 enhancement 类型上, 普通用户的要明显小于核心开发者的.

进一步从统计意义上确认两类报告者提交的缺陷的修复者被指派的任务数是否存在明显差异. Mann-Whitney U 的检验结果为: 对于 defect 类型缺陷, *p-value*>0.85, 因此普通用户提交的 defect 类型缺陷的修复者被指派任务数与核心开发者提交的没有明显区别; 对于 enhancement 类型缺陷, *p-value*=0.00057, 因此普通用户提交的 enhancement 类型缺陷的修复者被指派任务数明显小于核心开发者提交的.

上述结果表明: 在 enhancement 类型缺陷上, 系统更重视核心开发者提交的, 这与第 3.5 节中结论一致.

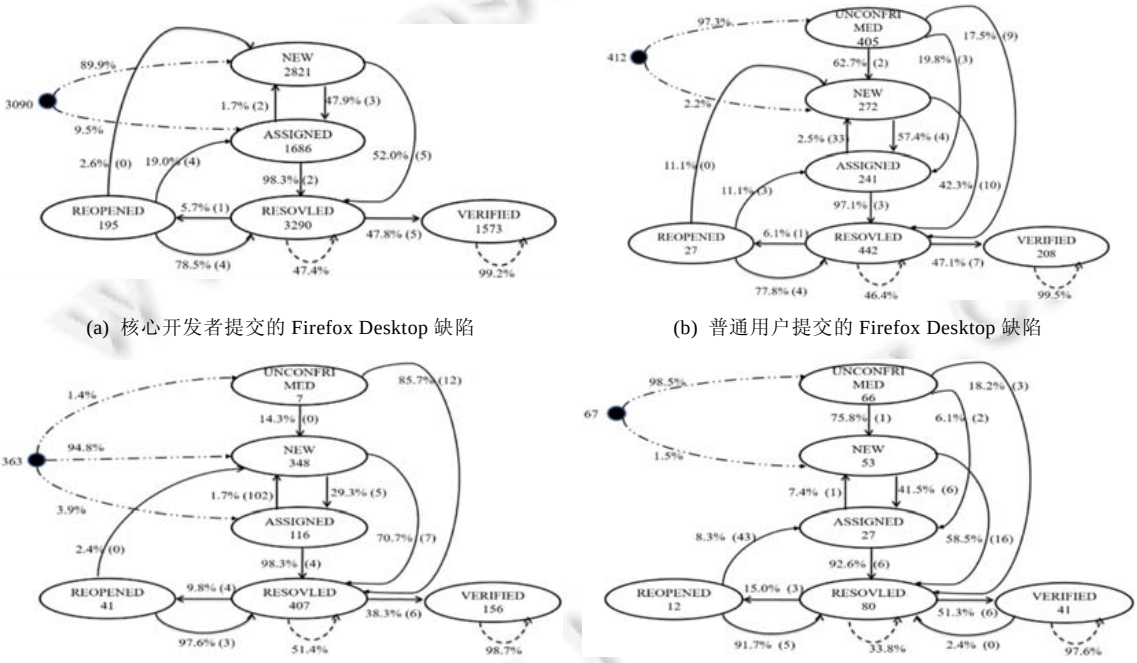
接下来对比分析普通用户和核心开发者提交的缺陷处理过程, 找出两者之间的差异.

本文从缺陷历史处理记录中提取已修复缺陷的状态转换信息, 并采用 BFG (bug flow graphs)方法进行可视化[13]. BFG 是一种用以描述缺陷处理过程中经历的状态以及状态之间转换的有向图, 其中, 图中节点代表

状态, 弧代表状态之间的转换, 弧的权值为 *P* (*MD*)形式, 其中, *P* 表示转换百分比, *MD* 表示转换时间的中位数. 为了了解哪些状态是缺陷处理过程中的主要状态, 本文在原 BFG 图中增加了每个状态被经历的次数(表示在状态下方). 为了显示清晰, 转换百分比小于 1%的转换未在图中标出.

在图 19 描述的 defect 类型缺陷的状态转换图中, 对比核心开发者和普通用户, 我们可以发现:

- 第一, 两种类型报告者提交的缺陷起始状态不同. 具体地, 核心开发者发布的缺陷 90%及以上直接从 NEW 状态开始, 而普通用户发布的缺陷 97%以上从 UNCONFIRMED 状态开始, 这反映了普通用户发布的缺陷在正式处理之前还需要经过系统管理人员的确认(即从 UNCONFIRMED 状态到 NEW 状态), 该过程通常需要 1–2 天时间.
- 第二, 两种类型报告者提交的缺陷状态之间转换的时间不同. 从 NEW 到 ASSIGNED 以及 ASSIGNED 到 RESOLVED, 普通用户提交的缺陷比核心开发者的转换时间多 1–2 天. 而由 NEW 直接到 RESOLVED, 两者之间的时间差增大到 5 天及以上.
- 第三, 普通用户发布的缺陷多次指定修复者和重新开放的概率均高于核心开发者提交的. 例如, 在 Firefox for Android 中, 普通用户提交的缺陷进入 ASSIGNED 状态后, 以 7.4%的概率返回到 NEW 状态等待重新分配修复者; 另外, 缺陷被解决后还有 15%的概率被重新开放需要重新修复. 这些结果进一步反映了, 普通用户提交的缺陷报告质量比较低. 这是因为缺陷报告对缺陷描述信息不全或者不够准确不仅会增加缺陷分配和修复的难度, 相应地增加修复的时间, 同时也会增加了错分和解决错误的概率.



(a) 核心开发者提交的 Firefox Desktop 缺陷

(b) 普通用户提交的 Firefox Desktop 缺陷

(c) 核心开发者提交的 Firefox for Android 缺陷

(d) 普通用户提交的 Firefox for Android 缺陷

图 19 defect 类型缺陷修复过程中的状态转换

在 enhancement 类型缺陷上, 普通用户提交的该类型缺陷大部分仍需要先经过管理人员的确认, 状态之间的转换时间总体上也大于核心开发者提交的, 缺陷的重开率也要高于后者. 但相比 defect 类型缺陷, 两种报告者提交的 enhancement 类型缺陷在状态转换时间上差距更大. 例如, 核心开发者发布的 enhancement 类型缺陷从 ASSIGNED 状态到 RESOLVED 状态需要 4 天时间, 而普通用户发布的则需要 19 天, 两者之差达到 15 天, 这远高于图 19(a)和图 19(b)两者之间对应的 defect 类型的 1 天时间差. 限于篇幅, 在此省略 enhancement

类型缺陷修复过程中的状态转换图.

结论: 在 defect 类型缺陷上, 系统为普通用户和核心开发者提交的缺陷指派的修复者经验值相当; 但在 enhancement 类型缺陷上, 相比普通用户, 核心开发者提交的更受重视, 其修复者被指派的修复经验值要明显大于普通用户的. 同时, 因为缺陷报告质量问题, 普通用户发布的缺陷修复流程更复杂, 而且相同的处理环节也需要更长的处理时间.

3.7 缺陷报告与缺陷修复速度关系(RQ7: 在缺陷报告上, 哪些因素会对缺陷修复速度产生影响?)

在本研究问题中, 我们继续从缺陷报告入手来对影响缺陷修复速度的相关因素进行分析. 具体地, 我们将调查缺陷标题长度、缺陷描述的准确性和充分性以及是否携带附件是否会影响普通用户提交的缺陷的修复时间.

(1) 缺陷标题

图 20 为缺陷修复时间与缺陷标题长度之间关系图. 图 20 显示: 无论是 defect 还是 enhancement 类型缺陷, 它们的修复时间和标题长度之间没有必然联系. 因为缺陷标题只是对缺陷的简要概述, 所以它对缺陷修复提供的帮助是有限的.

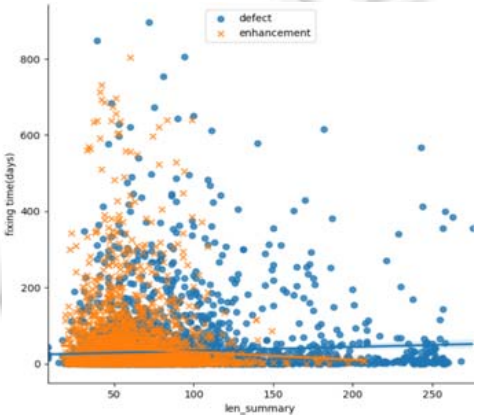


图 20 缺陷修复时间和标题长度之间的关系

(2) 缺陷描述

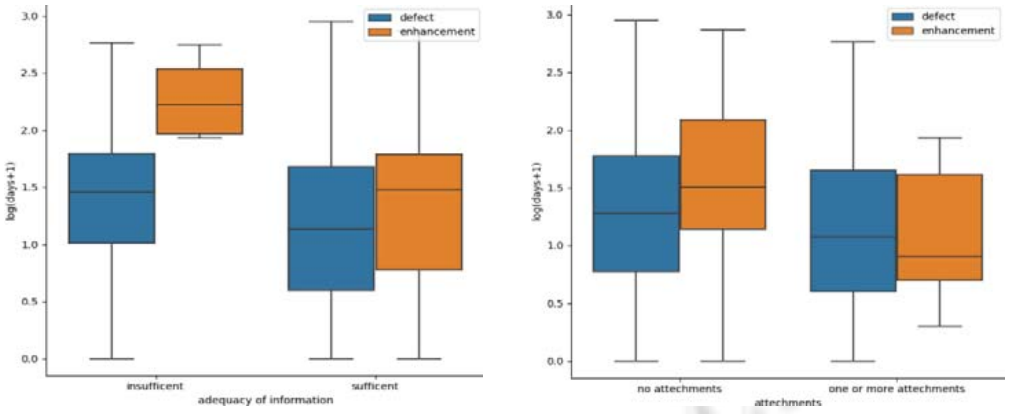
本文用缺陷评论中是否出现过需要更多信息以及缺陷不能复现等信息来衡量缺陷描述是否准确、充分. 具体地, 我们将缺陷按照评论信息中是否出现过不能复现该缺陷分成两类, 对比两类缺陷的修复时间, 来分析缺陷描述充分性对缺陷修复时间的影响. 结果如图 21(a)所示.

图 21(a)显示: 无论是 defecte 类型缺陷还是 enhancement 类型缺陷, 评论中出现过不能复现缺陷, 即缺陷描述不充分比不出现不能复现需要更长时间. 例如, 前者和后者的修复时间中值分别为 27.5 天和 12 天. defect 和 enhancement 类型缺陷的 Mann-Whitney U 检验结果(p-$value$<0.01)也表明, 是否出现过不能复现之间存在显著差异. 这其中原因很直接: 如果修复者不能复现缺陷, 则他/她需要再次甚至多次跟缺陷报告者进行交流. 显然, 在此这过程会存在一方等待另一方, 从而造成时间的浪费.

(3) 附件

Mozilla 缺陷跟踪系统允许用户以附件形式提供更多缺陷相关信息, 我们下面调查缺陷报告中是否携带附件对缺陷修复时间的影响, 结果如图 21(b)所示.

图 21(b)显示, 携带附件的缺陷比不携带附件的缺陷需要的修复时间更短些. 例如, 对于 defect 类型缺陷, 前者和后者的修复时间中位数分别为 11 天和 18 天. Mann-Whitney U 检验结果为 p-$value$<0.03, 表明是否携带附件在统计意义上存在差异. 也就是说, 附件可以帮助普通用户有效地缩短缺陷修复时间.



(a) 信息的充分性 (b) 是否携带附件

图 21 影响缺陷修复时间的因素

结论: 缺陷报告能否准确、充分地描述缺陷, 是影响缺陷修复速度的一个非常重要的因素; 描述不准确或者缺乏重要信息的缺陷, 其修复时间明显增多(约 15 天). 此外, 普通用户可以借助附件对缺陷报告进行补充, 在修复时间上可以降低 7 天左右.

3.8 结果讨论

本节围绕提高实际软件开发质量这一目标, 对第 3.1–3.7 节中的调查结果进行讨论, 分析当前缺陷跟踪系统存在的问题, 并为缺陷跟踪系统开发者和管理者改进系统、提高普通用户贡献度提供进一步的优化方向.

- 第一, 普通用户参与程度较浅.

RQ1 中的结果显示(见图 8), 普通用户提交的 defect 类型缺陷数量约占该类型缺陷总量 50%, 但是普通用户提交的 enhancement 类型缺陷只占对应类型缺陷总量 17%. 此外, RQ1 中的缺陷严重程度调查结果显示(见表 2): 核心开发者提交的 Firefox for Android 产品的 defect 类型缺陷约 25%为高严重等级; 然而, 普通用户提交的只有 3%为高严重等级. 这些都反映了目前大部分普通用户仅停留在报告软件产品使用过程中遇到的异常问题, 较少关注如何改进软件产品质量, 也较少进行深入的软件产品测试. 而且无论是 Firefox 的桌面版还是移动端, 86%左右的普通用户在本文调查的两年期间(2018 年、2019 年)仅提交一个缺陷. 进一步地, RQ2 的结果也表明(如图 9、图 10 所示), 普通用户很难发现非用户交互 UI 组件上的缺陷. 因此, 如何吸引数量众多的普通用户持续参与到缺陷追踪活动中, 以及如何激发每个用户深入参与缺陷的发现和修复等过程中并提供自己所拥有的潜在相关信息, 都是当前缺陷追踪和管理需要解决的关键问题. 这也是整个开源软件项目目前所面临的问题与挑战, 国内外大量学者正在就这些问题进行积极尝试和探索[1].

- 第二, 缺陷组件定位和自动查重功能有待进一步改进.

RQ2 和 RQ3 结果表明(如图 9、图 10、表 3 所示), 普通用户提交的缺陷中还有大量未确定组件(约 43%)以及重复的缺陷(约 32%). 这为缺陷分类人员工作带来了巨大的挑战. 与此同时, RQ4 的结果表明(如图 12 所示), 缺陷概述信息(即缺陷标题)对查重结果有着重要影响. 但对非专业测试人员的普通用户来说, 要求他们仅用一句话详细准确地描述缺陷难度不小. 因此, 查重时如何辅以其他信息, 如测试预期结果以及实际结果, 是值得缺陷跟踪系统开发人员探索的一个方向. 进一步地, 根据 RQ3 结果(见表 3), 即使 Firefox 核心开发者, 其提交的缺陷经查重后, 仍有大量处理为 DUPLICATE(最高到达 20%左右). 这说明, 当前缺陷跟踪系统的自动查重准确率或者召回率还有待进一步提高.

实际上, 目前缺陷自动查重方面的相关研究非常多, 这些研究也取得了很好的效果. 例如, Sun 等人[22]建议, 不仅可以对文本描述信息相似性进行度量, 还对缺陷所属产品、组件以及版本等非文本信息进行度量; Runeson 等人[23]利用向量空间模型和余弦相似度度量缺陷报告之间的文本相似性, 实验结果表明, 2/3 的重复

缺陷都能被检测到; Wang 等人[24]综合利用缺陷的自然语言描述和执行跟踪信息进行缺陷之间相似性度量, 召回率达到 67%–93%; Nguyen 等人[25]将文本相似性和主题相似性进行结合, 将查重准确率提高到 82%. 近年来, 深度学习技术也加入缺陷重复检测方法中. 例如, He 等人[26]提出了一种基于双通道卷积神经网络的重复缺陷报告检测方法. 他们的实验结果表明, 在开源项目 Open Office, Eclipse, Net Beans 等数据集上的分类准确性高达 95%.

然而在 Mozilla 的 Bugzilla 系统中, 目前普通用户提交的缺陷仍存在 30%左右的重复率. 是这些技术难以应用到实践中、还是 Bugzilla 不愿意采用这些技术? 或者已采用了这些技术但效果并未达到研究论文中汇报的水平? 这些问题及其背后的原因都是缺陷追踪系统领域值得进一步探索的研究方向. 如果能够有效降低缺陷重复率, 那么系统管理人员可以集中精力处理更多有用缺陷, 这对提高整个系统工作效率具有重要意义.

- 第三, 缺乏缺陷报告填写的智能辅助机制.

缺陷报告是缺陷修复的第一手材料, 它对缺陷修复的重要性不言而喻. RQ2 的结果表明(如图 9 和图 10 所示),普通用户提交的缺陷报告存在大量信息不充分, 难以组件定位. RQ4 和 RQ7 的结果也证实(如图 13 和图 21(a)所示), 用户对缺陷的描述是影响缺陷的处理结果和修复速度的一个非常重要的因素. 如果对缺陷的描述不够准确或者充分, 一方面缺陷修复人可能无法复现缺陷, 导致缺陷被处理为 INCOMPLETE 等无用的缺陷; 另一方面,对缺陷的描述不够准确或者充分, 则需要增加缺陷修复者和报告者之间的交流次数, 增加了缺陷的修复时间.

图 22 为当前 Mozilla 的缺陷跟踪系统的缺陷报告提交表单. 从图 22 可以看出, 复现步骤、实际结果以及预期结果的填写都只提供了简单的静态文本框. 这对非专业测试的用户来说要求非常高, 因为他们不清楚该如何有效地描述缺陷, 提供什么样的信息才更有价值, 才能更有利于缺陷的修复.

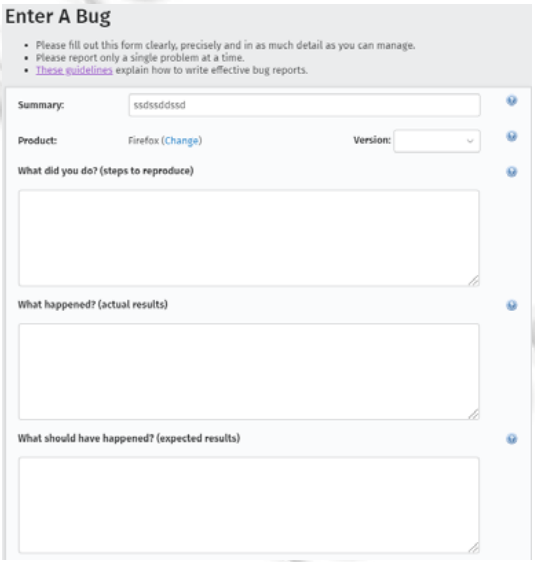


图 22 缺陷提交表单

为了提高缺陷报告质量, Just 等人[27]提出: 缺陷提交过程中的引导是必要的, 特别对于没有经验的普通用户, 这可以帮助他们收集更多信息. 同时, 这种引导还应该智能化, 即每一步设计的问题不是固定的, 而是根据缺陷提交者上一步回答问题情况而定. Lotufo 等人[28]还提出, 可以通过一定的游戏奖惩机制来激励普通用户提供质量更高的缺陷报告.

最近, Bhatia 等人[29]对问答平台 Stack Exchange (https://stackexchange.com/sites)使用的缺陷管理系统 Meta Stack Exchange (https://meta.stackexchange.com/)进行了调查, 他们研究了用户在线编辑缺陷报告是否能提高缺陷报告的质量. 在传统缺陷管理系统如 Bugzilla 中, 一旦报告者提交了缺陷报告后, 他只能通过一条条串行

评论的方式进行缺陷报告的修改或者补充说明. 这种方式无疑会增加缺陷修复者搜索有用信息的时间和难度. 与之不同, 在 Meta Stack Exchange 系统中, 报告者甚至经过授权的其他用户都可以对提交的缺陷报告进行包括增加信息、更正信息在内的在线编辑操作. 他们的研究结果表明: Meta Stack Exchange 中的 83%缺陷报告经过编辑, 其中 57%的缺陷报告质量得到了提升.

另外, RQ4 和 RQ7 的调查结果表明(如图 21(b)所示): 提交缺陷报告时附上附件不仅可以提高缺陷修复率, 还可以提高缺陷修复速度. 因为附件可以帮助用户对缺陷描述进行补充, 其中, 视频方式的附件对于不善于表达的用户来说是一种非常有效的缺陷方法. 但据我们调查, 当前缺陷跟踪系统没有对附件做任何要求, 完全由用户自己决定. 如果缺陷跟踪系统的开发者或者管理者对附件进行规范化或者给出一些建议, 这将有利于附件发挥更大的作用.

另外, 在普通用户提交的缺陷中, 还有很大一部分处理为 INVALID 和 WONTFIX, 特别是 enhancement 类型缺陷, WONTFIX 可高达 38%左右(见表 3). 在 RQ3 中, 我们调查得出, 这两类缺陷主要是因为用户不熟悉 Firefox 而造成的. 因此, 缺陷跟踪系统的开发者或者管理者有必要定期开展 Firefox 培训, 让用户更深入地了解 Firefox, 降低这些无效和不需要修复的缺陷, 让缺陷修复者有更多的时间去处理真正需要修复的缺陷, 提高缺陷跟踪系统的工作效率.

- 第四, 对普通用户提交的 enhancement 类型缺陷的关注度不够.

enhancement 类型缺陷是一种针对软件产品不足而提出的建议, 包括新增功能需求、UI 或者性能改进建议以及其他一些产品中面向用户部分的更改或者增强请求, 它在软件产品的演化中起着重要的作用. 例如, 对于 Firefox Desktop 产品, 核心开发者提交的缺陷中, 35%为 enhancement 类型缺陷.

除了开发者以外, 用户, 即软件产品的使用者, 他们对产品的改进意见或者需求能否得到满足, 也影响着产品在市场中的竞争力. 最近, Alsubaihin 等人[30]对应用商店开发人员在软件工程活动的各个方面进行了调查, 结果表明, 51%的开发者经常根据应用商店中用户的反馈进行软件完善性维护. 这些开发者不仅利用用户反馈来识别软件需要新增的功能, 而且还利用用户反馈进行软件功能的修改和删除. 另外, Pagano 等人[31]的研究进一步表明: 在应用商店中, 约 1/3 的用户反馈包括关于软件需求和用户体验等主题, 需求分析人员可以立即从这些反馈中获益, 可以了解应用程序的实际使用情况, 并评估软件功能的重要性, 以“民主”的方式扩展应用程序, 这有助于发布计划.

然而, RQ5 及 RQ6 的结果显示(如图 16–图 18 所示): 在 Firefox 缺陷跟踪系统中, 普通用户提交的 enhancement 类型缺陷未解决率高、修复时间长以及修复者经验值低, 这些都在一定程度上反映了当前缺陷跟踪系统未重视普通用户提交的 enhancement 缺陷. 这可能会导致开发者对用户的实际需求了解的不充分, 影响产品在市场上的竞争力; 另外, 如果长此以往, 用户提交缺陷的积极性也会受到一定影响, 出现恶性循环. 因此, 我们建议适当提高用户提交的这部分缺陷在系统中的地位.

4 影响结果有效性因素

本文对开源软件 Firefox 缺陷跟踪系统进行了实证研究, 根据收集的缺陷数据, 分析了用户提交的缺陷报告在缺陷跟踪系统中的作用以及系统对它们的处理特点. 然而, 实证性研究通常都存在影响结果有效性的因素, 本文也不例外. 下面我们从数据可靠性、外部有效性和内部有效性这 3 个方面进行分析.

- 第一, 数据可靠性. 本文研究使用的数据都可从 Mozilla 的缺陷跟踪系统 Bugzilla 中获取, 文中第 2 节详细描述了这些数据的收集方法. 然而如前文所述, 缺陷修复是个长期过程, 即使已经完成修复的缺陷也可能重新开放. 这样, 随着时间的推移, 本文中使用的数据可能会发生变化. 为了尽可能地减少数据变化带来的影响, 文中选用了 2018 年和 2019 年提交的缺陷. 截至我们的数据爬取时间, 这些缺陷至少历时半年, 大多数已处于一个较稳定状态, 因此文中的实验结果变化不大.
- 第二, 外部有效性. 本文选取了 Mozilla 项目的两个软件产品 Firefox Desktop 和 Firefox for Android 作为研究对象, 虽然文中的某些结果只对这两个产品有效, 但 Firefox 以及其缺陷跟踪 Bugzilla 都具有

一定的代表性, 它们也广泛应用于很多软件工程相关问题的研究. 因此, 本文中的某些结果虽然不能完全适用于其他开源软件, 但仍有很好的借鉴作用.

- 第三, 内部有效性. 本文采用用户 id 来标识用户. 虽然该方法较使用用户名、姓名、昵称等信息识别用户身份更有效, 但根据 Zhou 等人[19]的调查, Mozilla 项目使用的 Bugzilla 系统中仍存在多个用户使用同一个 email 地址或者同一个用户使用多个 email 地址等情况(本文用的用户 id 与其 email 具有一一对应关系), 不过这种情况只占少部分(10%以下). 因此, 文中的结果与实际情况可能存在一定偏差, 但总体不受影响.

5 相关工作

相关研究表明, 软件产品生命周期的 70%用于软件维护[32]. 源码修改、缺陷处理、迭代演化都是软件维护的主要任务. 目前, 大多数大型软件无论是开源的还是商业的, 如 Mozilla, Apache, Linux, Windows, 都采用缺陷跟踪系统来提高缺陷管理的效率[33]. 在缺陷跟踪系统中, 人们可以进行缺陷的报告、跟踪、讨论等等. 特别是开放性缺陷跟踪系统面向公众, 吸引了一大批参与者, 有利于提高软件整体质量. 然而, 随着参与者的增多, 一方面, 缺陷数量的增加使系统资源面临巨大压力; 另一方面, 无效的、重复的、信息不全的缺陷数量也在增加, 这严重浪费了系统的处理时间. 如何提高缺陷跟踪系统的处理效率, 学者们进行了大量的研究, 提出了各种解决方法. 这里仅给出与本文相关的主要研究工作, 我们将这些工作分为 3 类. 需要说明的是, 有些工作的内容与多种类别都相关, 我们按其主要内容来归类.

- 第 1 类, 分析影响缺陷修复效率的因素. 这类工作旨在找出与缺陷修复时间相关的因素, 以提高缺陷修复效率. Saha 等人[8]通过对 4 个 Eclipse 项目以及 Linux kernel, WineHQ, GDB 这 3 个项目中处理时间超过一年的缺陷分析发现: 长时间不指定修复者、缺陷的重要性没有得到充分认识, 都是缺陷得不到及时处理的重要原因. 同时, 他们指出, 仍有大量缺陷找不到具体被延迟处理的原因. Zhang 等人[21]对 Mylyn, Eclipse Platform 以及 Eclipse Plug-in Development Environment 这 3 个开源项目进行实证研究发现: 与其他操作系统相比, Linux 上的缺陷能够得到更及时的处理; 另外, 缺陷描述文本长度、评论信息长度、缺陷的严重性, 也会影响缺陷处理速度. 刘文杰等人[34]的实证调查进一步表明: 严重程度为中等级别的缺陷会被开发者优先解决; 严重程度高的缺陷因其解决难度较大, 导致解决时长较长; 而严重程度较低的缺陷受重视程度不高, 解决也需较长时间. Giger 等人[35]通过 Eclipse, Mozilla 以及 Gnome 项目实证调查缺陷报告中的字段与缺陷时间修复之间的关系, 他们指出, 缺陷修复者、报告者以及缺陷提交月份都会对缺陷修复时间产生重要影响. 他们还构建了预测模型, 预测提交的缺陷能否快速被修复.
- 第 2 类, 特征化各类缺陷. 这类工作主要通过实证研究分析某一类或者几类缺陷的特征. 例如, Joorabchi 等人[36]通过 Firefox, Eclipse, MediaWiki, Moodle, Firefox Android 以及 Industrial 这 6 个软件项目对 WORKSFORME 缺陷进行了实证研究, 统计出这类缺陷比例, 并分析它们产生的原因及在处理流程中的状态转换规律; Guo 等人[33]则分析了在 Windows Vista 以及 Windows 7 系统中什么类型的缺陷将会被修复, 同时, 他们还构建了一个统计模型用于预测新缺陷被修复的概率; 而 Zimmermann 等人[37]的研究则关注 Windows 中 REOPENED 缺陷的分析, 并给出缺陷报告中的不同特征对缺陷重开概率的影响; Rocha 等人[13]利用 BFG 图(bug flow graph), 对 Firefox 多种类型缺陷, 如 FIXED, DUPLICATE, INCOMPLETE 进行特征分析, 他们发现, DUPLICATE 缺陷处理时间的平均值以及中位数都最短, 分别为 28 天和 1 天, 而 INCOMPLETE 缺陷处理时间的平均值以及中位数都最长, 分别为 171 天和 161 天.
- 第 3 类, 探索开源项目中的潜在社会结构. 这类工作主要研究参与者个体行为以及个体之间的交互关系. Bissyande 等人[38]对 GitHub 上 100 000 个项目进行了调研, 他们发现: 单个开发者的项目很少会得到有用户的反馈, 但当开发者达到 5 人, 问题报告数急剧增加, 中位数可以达到 49 条. 他们还发现,

大部分项目开发者(约 67%)不参与该项目的缺陷报告, 但是大多数缺陷报告者(约 58%)对项目代码库有贡献. Zhou 等人[19]研究了参与者初始行为以及项目环境对参与者成为长期稳定项目贡献者之间关系, 结果表明, 以缺陷评论方式比缺陷报告方式进入项目成为长期贡献者的机会更大. 另外, 参与者报告的第 1 个缺陷能否得到修复也密切关系到他能否成为长期的贡献者. 然而, 项目的普及度却起着反作用. 也就是, 项目的用户数越大, 参与者成为长期贡献者的概率就越小. 而 Mockus 等人[20]从开发者参与度、核心团队规模、代码所有权、开发者代码生产能力、缺陷密度以及缺陷解决时间等 6 个方面对 Apacheh 和 Mozilla 项目进行实证研究, 他们还将这两个开源项目比 5 个商业软件进行对比.

与分析影响缺陷修复效率因素的相关研究不同, 本文还对比分析了普通用户和核心开发者提交的缺陷的修复流程, 找出了两者之间的差异, 并分析原因; 与特征化各类缺陷的相关研究主要关注缺陷本身的特点不同, 本文主要关注缺陷的报告者(即普通普通用户和核心开发者)的参与和影响; 最后, 与第三类已有相关工作研究对象主要为软件产品的开发者不同, 本文的研究对象为缺陷报告者. 总之, 本文首次调查分析了用户提交的缺陷在系统中的地位和作用, 从缺陷的数量、严重性、修复率、处理流程等多个方面对比了普通用户和核心开发者提交的缺陷; 同时, 本文还将 defect 和 enhancement 类型缺陷分开讨论, 为学者和开发者进一步改进开源软件缺陷跟踪系统提供帮助.

6 结论及未来研究方向

缺陷管理是软件生命周期中一个重要组成部分. 为了提高缺陷管理的质量和效率, 大型开源软件都采用开放性缺陷跟踪系统进行缺陷管理. 在开放性缺陷跟踪系统中, 任何人都可以进行缺陷的发布、评论甚至修复, 他们通过协同合作共同构造出高质量的复杂软件系统.

本文对开源软件 Firefox 的缺陷跟踪系统进行了实证研究, 收集了 2018 年和 2019 年发布的 22 531 个 Firefox Desktop 和 Firefox for Android 缺陷. 通过对比分析由软件产品使用者构成的普通用户和具有缺陷编辑权限的核心开发者所提交的缺陷在数量、严重程度、组件分布、修复率、修复速度以及修复者等方面的差异, 同时还调查了普通用户提交的缺陷报告质量与缺陷修复率和修复时间之间的关系, 我们发现:

(1) 由于缺乏有效的激励机制, 普通用户无论是在数量上还是严重程度和组件分布上, 都体现出参与程度较浅. 目前, 众多研究者正就这一问题进行积极探索.

(2) 缺陷报告重复性检测性能不理想, 导致重复缺陷报告大量存在, 降低了系统的工作效率. 多种检测性技术的适当组合以及深度学习技术, 是提高缺陷重复性检测率值得借鉴的方法.

(3) 先天性经验不足加上缺乏有效的缺陷报告填写智能辅助机制, 普通用户提交的报告信息不充分, 总体质量不高, 大量组件无法定位, 无用缺陷比例高, 缺陷修复比例低, 修复时间长. 对普通用户辅以智能引导以及允许他们在线编辑缺陷报告, 是提高缺陷报告质量值得尝试的方法. 另外, 本文调查结果显示: 缺陷提交时携带附件, 也是一种有效地提高缺陷修复率和修复速度的方法.

(4) 普通用户提交的软件新功能请求或者改进建议还未受到充分重视, 系统为普通用户和核心开发者两者提交的该类型缺陷指定的修复者具有明显的差异. 建议适当提高普通用户所提交的该类型缺陷在系统中的地位, 这对增加软件产品在市场上的竞争力有一定积极作用.

在下一步工作中, 为了更为全面地了解用户在开源软件的软件维护中的作用, 我们计划将缺陷管理系统与其他类型信息管理系统, 如版本控制系统进行结合, 综合衡量用户在缺陷的发布、缺陷评论以及缺陷修复等全过程中的作用和地位.

References:

[1] Zhang W, Mei H. Software development based on collective intelligence on the Internet: Feasibility, state-of-the-practice, and challenges (in Chinese with English abstract). Scientia Sinica Informationis, 2017, 47(12): 1601−1622. [doi: 10.1360/N112017-00117]

[2] Charette, RN. Why software fails [software failure]. IEEE Spectrum, 2005, 42(9): 42−49. [doi: 10.1109/MSPEC.2005.1502528]

[3] Xuan JF, Jiang H, Zhang HY, Ren ZL. Developer recommendation on bug commenting: A ranking approach for the developer crowd. Science China (Information Sciences), 2017, 60(7): 159−176. [doi: 10.1007/s11432-015-0582-8]

[4] Zhang T, Jiang H, Luo XP, Chan ATS. A literature review of research in bug resolution: Tasks, challenges and future directions. The Computer Journal, 2016, 59(5): 741−773. [doi: 10.1093/comjnl/bxv114]

[5] Anvik, J, Hiew L, Murphy CG. Coping with an open bug repository. In: Proc. of the OOPSLA Workshop on Eclipse Technology eXchange. 2005. 35−39. [doi: 10.1145/1117696.1117704]

[6] Pagano D, Bruegge B. User involvement in software evolution practice: A case study. In: Proc. of the 35th Int'l Conf. on Software Engineering. 2013. 953−962. [doi: 10.1109/ICSE.2013.6606645]

[7] Sasso TD, Mocci A, Lanza M. What makes a satisficing bug report? In: Proc. of the IEEE Int'l Conf. on Software Quality. 2016 164−174. [doi: 10.1109/QRS.2016.28]

[8] Saha RK, Khurshid S, Perry DE. Understanding the triaging and fixing processes of long lived bugs. Information & Software Technology, 2015, 65: 114−128. [doi: 10.1016/j.infsof.2015.03.002]

[9] Rastkar S, Murphy GC, Murray G. Automatic summarization of bug reports. IEEE Trans. on Software Engineering, 2014, 40(4): 366−380. [doi: 10.1109/TSE.2013.2297712]

[10] Jeong G, Kim S, Zimmermann T. Improving bug triage with bug tossing graphs. In: Proc. of the ESEC/FSE 2009. 2009. 111−120. [doi: 10.1145/1595696.1595715]

[11] Xia X, Lo D, Shihab E, Wang XY, Zhou B. Automatic, high accuracy prediction of reopened bugs. Automated Software Engineering, 2015, 22(1): 75−109. [doi: 10.1007/s10515-014-0162-2]

[12] 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. 2013. 133−143. [doi: 10.1109/CSMR.2013.23]

[13] Rocha H, de Oliveira G, Valente MT, Marques-Neto H. Characterizing bug workflows in Mozilla Firefox. In: Proc. of the 30th Brazilian Symp. on Software Engineering. 2016. 43−52. [doi: 10.1145/2973839.2973844]

[14] Zhang W, Li ZQ, Li B. Developer teams on bug resolution: An empirical study on Mozilla Firefox and Eclipse JDT projects. In: Proc. of the 24th Asia-Pacific Software Engineering Conf. Workshops. 2017. 32−40. [doi: 10.1109/APSECW.2017.10]

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

[16] Hong Q, Kim S, Cheung SC, Bird C. Understanding a developer social network and its evolution. In: Proc. of the 27th IEEE Int'l Conf. on Software Maintenance. 2011. 323−332. [doi: 10.1109/ICSM.2011.6080799]

[17] van Liere DW. How shallow is a bug? Why open source communities shorten the repair time of software defects. In: Proc. of the Int'l Conf. on Information Systems. 2009. 195−210. [doi: 10.2139/ssrn.1507233]

[18] Catolino G, Palomba F, Zaidman A, Ferrucci F. Not all bugs are the same: Understanding, characterizing, and classifying bug types. Journal of Systems and Software, 2019, 152: 165−181. [doi: 10.1016/j.jss.2019.03.002]

[19] Zhou M, Mockus A. Who will stay in the FLOSS community? Modeling participant's initial behavior. IEEE Trans. on Software Engineering, 2015, 41(1): 82−99. [doi: 10.1109/TSE.2014.2349496]

[20] Mockus A, Fielding RT, Herbsleb JD. Two case studies of open source software development: Apache and Mozilla. ACM Trans. on Software Engineering & Methodology, 2002, 11(3): 309−346. [doi: 10.1145/567793.567795]

[21] Zhang F, Khomh F, Zou Y, Hassan AE. An empirical study on factors impacting bug fixing time. In: Proc. of the 19th Working Conf. on Reverse Engineering. 2012. 225−234. [doi: 10.1109/WCRE.2012.32]

[22] Sun C, Lo D, Khoo SC, Jiang J. Towards more accurate retrieval of duplicate bug reports. In: Proc. of the 26th IEEE/ACM Int'l Conf. on Automated Software Engineering. 2011. 253−262. [doi: 10.1109/ASE.2011.6100061]

[23] 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. 2007. 499−510. [doi: 10.1109/ICSE.2007.32]

[24] Wang X, Zhang L, Xie T, Anvik J, Sun J. An approach to detecting duplicate bug reports using natural language and execution information. In: Proc. of the 30th Int'l Conf. on Software Engineering. 2008. 461−470. [doi: 10.1145/1368088.1368151]

[25] Nguyen AT, Nguyen TT, Nguyen TN, Lo D, Sun C. Duplicate bug report detection with a combination of information retrieval and topic modeling. In: Proc. of the Int'l Conf. on Automated Software Engineering. 2012. 70−79. [doi: 10.1145/2351676.2351687]

[26] He JJ, Xu L, Yan M, Xia X, Lei Y. Duplicate bug report detection using dual-channel convolutional neural networks. In: Proc. of the Int'l Conf. on Program Comprehension. 2020. 117−127. [doi: 10.1145/3387904.3389263]

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

[28] Lotufo R. Towards next generation bug tracking systems [Ph.D. Thesis]. Ontario: University of Waterloo, 2013.

[29] Bhatia A, Wang S, Asaduzzaman M, Hassan AE. A study of bug management using the stack exchange question and answering platform. IEEE Trans. on Software Engineering, 2022, 48(2): 502−518. [doi:10.1109/TSE.2020.2994006]

[30] Al-Subaihin AA, Sarro F, Black S, Capra L, Harman M. App store effects on software engineering practices. IEEE Trans. on Software Engineering, 2021, 47(2): 300−319. [doi: 10.1109/TSE.2019.2891715]

[31] Pagano D, Maalej W. User feedback in the appstore: An empirical study. In: Proc. of the 21st IEEE Int'l Requirements Engineering Conf. 2013. 125−134. [doi: 10.1109/RE.2013.6636712]

[32] Boehm BW, Basili VR. Software defect reduction top 10 list. Computer, 2001, 34(1): 135−137. [doi: 10.1109/2.962984]

[33] Guo PJ, Zimmermann T, Nagappan N, Murphy B. Characterizing and predicting which bugs get fixed: An empirical study of Microsoft windows. In: Proc. of the 32th Int'l Conf. on Software Engineering. 2010. 495−504. [doi: 10.1145/1806799.1806871]

[34] Liu WJ, Jiang H. Analysis of software bug reports severity feature. Computer Engineering and Applications, 2019, 55(14): 48−53, 208 (in Chinese with English abstract). [doi: 10.3778/j.issn.1002-8331.1810-0360]

[35] Giger E, Pinzger M, Gall HC. Predicting the fix time of bugs. In: Proc. of the 2nd Int'l Workshop on Recommendation Systems for Software Engineering. 2010. 52−56. [doi: 10.1145/1808920.1808933]

[36] 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. 2014. 62−71. [doi: 10.1145/2597073.2597098]

[37] Zimmermann T, Nagappan N, Guo PJ, Murphy B. Characterizing and predicting which bugs get reopened. In: Proc. of the 34th Int'l Conf. on Software Engineering. 2012. 1074−1083. [doi: 10.1109/ICSE.2012.6227112]

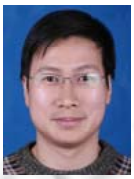
[38] Bissyande TF, Lo D, Jiang L, Reveillere L, Klein J, Traon YL. Got issues? Who cares about it? A large scale investigation of issue trackers from GitHub. In: Proc. of the 24th Int'l Symp. on Software Reliability Engineering. 2013. 188−197. [doi: 10.1109/ISSRE.2013.6698918]

附中文参考文献:

[1] 张伟, 梅宏. 基于互联网群体智能的软件开发: 可行性、现状与挑战. 中国科学: 信息科学, 2017, 47(12): 1601−1622. [doi: 10.1360/N112017-00117]

[34] 刘文杰, 江贺. 软件缺陷报告严重性属性分析. 计算机工程与应用, 2019, 55(14): 48−53, 208. [doi: 10.3778/j.issn.1002-8331.1810-0360]

王燕(1978－), 女, 博士生, 讲师, CCF 专业会员, 主要研究领域为软件测试和维护.



徐家喜(1972－), 男, 高级工程师, CCF 专业会员, 主要研究领域为软件测试.

吴化尧(1989－), 男, 博士, 助理研究员, CCF 专业会员, 主要研究领域为软件测试和维护.

尹震(1993－), 男, 硕士, 主要研究领域为软件测试, 云平台容错技术及评估.

聂长海(1971－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为软件工程.

钮鑫涛(1988－), 男, 博士, 助理研究员, CCF 专业会员, 主要研究领域为软件测试, 故障定位.