软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(7):2581−2598 [doi: 10.13328/j.cnki.jos.006312]

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

代码审查中代码变更恢复的经验研究*

王青叶[1], 万志远[1], 李善平[1], 夏 鑫[2]

[1](浙江大学 计算机科学与技术学院, 浙江 杭州 310007)

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

通信作者: 万志远, E-mail: wanzhiyuan@zju.edu.cn

摘 要: 代码审查是一种由其他开发者而非代码作者本人评审代码的形式. 在代码审查系统中, 开发者通过提交代码变更来修复软件缺陷或添加软件特性. 并非所有的代码变更都会被集成到代码库中, 部分代码变更会被拒收. 被拒收的代码变更有可能被恢复, 并继续接受审查, 提供代码贡献者改进代码变更的机会. 然而, 审查恢复过的代码变更需要花费更多的时间. 收集了 4 个开源项目中的 920 700 条代码变更, 采用主题分析方法识别出 11 类代码变更恢复的原因, 并定量分析被恢复的代码变更的特征. 主要发现包括: 1) 导致代码变更恢复的原因中, “提升改进”类型占比最大; 2) 不同项目之间, 代码变更被恢复的原因类别分布存在差异, 但并不显著; 3) 与从未恢复过的代码变更相比, 恢复的代码变更接收率低 10%, 评论数量平均多 1.9 倍, 审查所用时间平均多 5.8 倍; 4) 81%的恢复代码变更被接收, 19%的恢复代码变更被拒收.

关键词: 代码审查; 代码变更; 代码变更拒收; 代码变更恢复; 经验研究

中图法分类号: TP311

中文引用格式: 王青叶, 万志远, 李善平, 夏鑫. 代码审查中代码变更恢复的经验研究. 软件学报, 2022, 33(7): 2581–2598. http://www.jos.org.cn/1000-9825/6312.htm

英文引用格式: Wang QY, Wan ZY, Li SP, Xia X. Empirical Study on Restored Code Changes in Code Review. Ruan Jian Xue Bao/Journal of Software, 2022, 33(7): 2581−2598 (in Chinese). http://www.jos.org.cn/1000-9825/6312.htm

Empirical Study on Restored Code Changes in Code Review

WANG Qing-Ye[1], WAN Zhi-Yuan[1], LI Shan-Ping[1], XIA Xin[2]

[1](College of Computer Science and Technology, Zhejiang University, Hangzhou 310007, China)

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

Abstract: Code review is manual inspection of source code by developers other than the author. In a code review system, software developers submit code changes to fix software defects or add software features. Not all of the code changes will be integrated into the code base. Some of the code changes will be abandoned. The abandoned code changes could be restored for review, which allows contributors to further improve the code changes. However, it takes more time to review the restored code changes. This study collects 920 700 restored code changes from four open-source projects, investigates reasons for which code changes have been restored, uses thematic analysis method to identify 11 categories of reasons for restoring code changes, and quantitatively analyzes the characteristics of restored code changes. The main findings are as followings. (1) Among the reasons for which code changes are restored, the reason “improve and update” accounts for the largest proportion. (2) The distribution of reasons across the four projects are different, but the difference is not significant. (3) Compared with non-restored code changes, restored code changes have a 10% lower acceptance rate, 1.9 times more comments, and 5.8 times longer review time on average. (4) 81% of restored code changes has been accepted, while 19% of them has still been abandoned.

Key words: code review; code change; abandoned code change; restored code change; empirical study

* 基金项目: 国家重点研发计划(2020YFB1005400); 浙江省重点研发计划(2021C01014)
收稿时间: 2020-09-17; 修改时间: 2020-11-28, 2021-01-06; 采用时间: 2021-01-27; jos 在线出版时间: 2021-08-03

代码审查(code review)是一种由其他开发者而非代码作者本人评审代码的形式. 代码贡献者提交代码之后, 由其他开发者对此代码进行评审, 只有通过评审的代码才能最终合并到代码库当中. 代码审查有助于在软件开发早期有效探测并修复缺陷, 从而保证软件产品的质量, 减少测试和维护的代价[1−3]. 研究发现, 代码审查可发现并移除 30%−70%的软件缺陷[4−6]. 与未经审查的代码相比, 经过审查的代码引入缺陷的概率更低[7], 同时, 可阅读性更高[7]. 代码审查的概念诞生于 1976 年, Fagan 制定了一个高度结构化的代码审查流程, 该流程中, 代码审查参与者面对面对代码进行逐行审查. 随后, 诸多研究证实了代码审查的优势, 尤其在发现软件缺陷方面. 然而, 由于其审查流程繁琐、耗时和同步进行的特性, 阻碍其在实践中被广泛使用[8]. 近年来, 很多组织采用轻量级的代码审查实践[9], 提升代码审查效率.

代码变更(code change)是代码审查的对象. 代码变更是提交到代码审查系统中的一组源文件, 旨在修复缺陷或添加新功能. 代码变更包含一个用来识别该代码变更的编号(change-id), 通常由一个或多个提交(commit)组成. 提交记录代码贡献者对于一个或多个文件的更改. 一个代码变更可包含多个提交, 这些提交对该代码变更进行迭代改进. 当一个代码变更被接收时, 该代码变更中的最终提交将合并到代码库中.

代码审查流程中, 关键角色包括代码贡献者、项目管理维护者和代码审查者. 代码审查的流程通常包括如下步骤: (1) 代码贡献者提交一个代码变更到代码审查系统; (2) 代码审查者进行评审并给出改进建议; (3) 代码贡献者对代码变更进行改进完善后再次提交; (4) 审查者再次审查; (5) 项目管理维护者决定接收或拒收. 代码审查流程中, 并非所有提交的代码变更最终都能被接收, 有些代码变更会被拒收, 被拒收的代码变更可以被恢复.

代码变更拒收后被恢复(restored), 提供给代码贡献者改进代码变更的机会, 或引发一些新的问题. 研究发现, 在 Apache 项目中, 56%的代码变更被拒收[10]; 在 OpenAFS 和 FLAC 项目中, 60%的代码变更被拒收[11]. 文献[12]发现, 17%的代码变更被拒收. 代码变更被拒收通常导致代码贡献者社区贡献热情降低和声誉受损[13,14]. 部分拒收的代码变更会被恢复, 即代码审查流程中的参与者重新开放代码变更, 供代码审查者进行评审. 代码变更恢复时, 参与者通常会对恢复的原因进行说明. 文献[15]研究了代码变更拒收的原因. 文献[12]预测代码变更是否会被拒收, 该研究关注代码变更的拒收, 但未涉及代码变更的恢复. 文献[16,17]研究了重新开放(reopen)的缺陷(bug). 文献[18]研究了恢复的拉取请求(pull request). 然而, 这些研究并未关注代码变更恢复.

本文旨在通过对代码变更恢复进行经验研究, 为代码贡献者、代码审查平台提供参考和建议, 从而提高代码审查效率, 降低恢复代码变更的成本. 具体而言, 本文试图回答两个研究问题.

- RQ1: 在代码变更被拒收后, 哪些原因导致代码变更被恢复?
- RQ2: 被恢复的代码变更呈现何种特征?

回答这些研究问题, 可以帮助代码贡献者在代码变更拒收后, 提交更易于接收的代码变更; 可以帮助代码审查平台提供更有效的工具, 便利化代码审查, 优化资源配置.

为了回答上述研究问题, 本文选取了被广泛应用的 Gerrit 代码审查系统, 收集 Gerrit 代码审查系统中 4 个开源项目的 920 700 条代码变更(链接:https://pan.baidu.com/s/1Kro4hb0E7VMGo86eP8eXyw, 提取码:2sm0), 其中包含代码变更的很多基础信息, 例如创建时间和审查讨论信息. 本文将其共享, 以便研究者对其进行进一步的使用和研究. 本文利用主题分析方法对代码变更恢复的原因进行分析分类, 并使用威尔科克森符号秩检验(Wilcoxon signed-rank test)方法, 检验不同项目之间恢复原因的分布异同情况. 此外, 采用定量分析的方法, 理解恢复的代码变更的特征. 本文的主要发现包括: (1) 导致代码变更恢复的原因中, “提升改进”类型占比最大; (2) 不同项目之间, 代码变更被恢复的原因类别分布存在差异, 但并不显著; (3) 与从未恢复过的代码变更相比, 恢复的代码变更接收率低 10%, 评论数量平均多 1.9 倍, 审查所用时间平均多 5.8 倍; (4) 81%的恢复代码变更被接收, 19%的恢复代码变更被拒收.

本文的主要贡献可总结如下:

(1) 收集了 4 个开源项目中 920 700 条代码变更;

(2) 总结了 11 个代码变更被恢复的原因;

(3) 分析并提出了被恢复的代码变更的特征.

本文第 1 节介绍相关背景. 第 2 节介绍数据集与分析方法. 第 3 节讨论数据分析结果. 第 4 节介绍相关工作. 第 5 节总结全文和展望未来工作.

1 背 景

本节介绍代码审查中涉及的关键概念及基本流程, 以及 Gerrit 代码审查系统中的代码变更示例.

1.1 代码审查

代码审查与软件开发过程紧密集成, 被广泛地应用于私有和开源项目[19]. 代码审查与发布后的软件缺陷[20]、代码变更质量[9,21,22]、软件设计质量[23]有着紧密的关系. 此外, 代码审查可以让开发者之间知识共享, 促进交流[19]. Gerrit 是一种广受欢迎且广泛应用的代码审查工具, 用来记录基于版本控制系统 Git 的代码审查过程.

基于 Gerrit 的代码审查流程如图 1 所示, 包含如下步骤.

1) 提交代码变更: 代码贡献者提交代码变更到 Gerrit 代码审查系统.
2) 审查代码变更: 审查者审阅代码变更并进行反馈. 审查者既可以通过代码贡献者邀请, 又可以通过代码审查平台分配指定. 代码审查者通过在评论区添加评论反馈意见和建议.
3) 改进代码变更: 代码贡献者根据评论区审查者的反馈对代码变更进行改进, 之后再次提交以更新代码变更.
4) 关闭代码变更: 重复第 2 步和第 3 步, 直到代码变更满足要求, 项目管理维护者可以决定接收或拒收此代码变更.
5) 恢复代码变更: 在某些情况下, 代码变更拒收之后可以被恢复, 以继续接受审查. 这一步在代码审查过程中并不是必须的, 通常只有被恢复的代码变更才会有此过程.

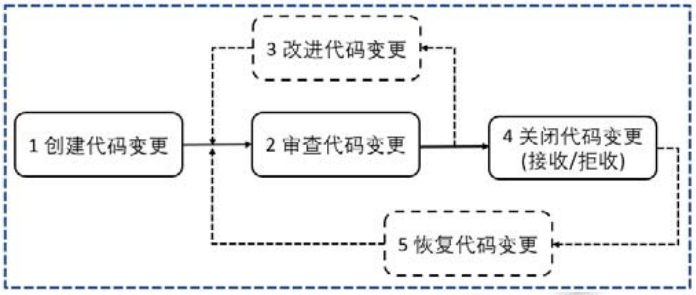

图 1 代码审查流程

1.2 Gerrit代码审查系统的代码变更示例

Gerrit 代码审查系统中, 一般代码变更包含如下信息.

- Change-Id: 代码变更的唯一标识.
- 题目: 代码变更的概括.
- 描述: 代码变更所涉及内容的简单描述.
- 评论: 在审查过程中, 审查者添加的文本反馈意见, 代码贡献者可以根据反馈意见改进代码变更.
- 状态: 代码变更经过审查后, 代码变更的状态一般包括拒收或合并, 具体如下:
  - 拒收(abandoned): 代码变更经过评审之后, 被拒绝合并到代码库中.
  - 合并(merged): 代码变更经过评审之后被接收, 可以合并到代码库中.

代码变更除以上主要信息外, 还包括审查者、最近更新时间和修改文件列表等信息. 以 OpenDev 项目中

#577425 代码变更为例, 图 2 展示了 Gerrit 代码审查系统中的该代码变更的题目、描述、Change-Id、审查者、评论和修改文件等信息.

图 2 Gerrit 中的代码变更示例(OpenDev#577425)

2 数据与分析方法

本文主要围绕以下两个研究问题展开.

- RQ1: 在代码变更被拒收后, 哪些原因导致代码变更被恢复? 研究代码变更的恢复原因类型以及这些原因类型在不同项目间的分布, 将帮助代码贡献者规避某些行为, 从而提交高质量的代码变更, 同时帮助代码审查平台提升审查效率.
- RQ2: 被恢复的代码变更呈现何种特征? 研究恢复代码变更的特征, 可以帮助代码审查平台优化资源配置, 有助于代码审查平台更好地决策.

2.1 数据收集

- 数据来源

本文选取 4 个广受欢迎的开源项目, 即 Eclipse, LibreOffice, OpenDev 和 Qt. 这 4 个开源项目拥有大量的活跃软件实践者参与项目开发与维护, 代码审查系统中包含大量的代码变更, 并以每天几十条甚至几百条代码变更的速度扩张. 此外, 这 4 个开源项目涵盖多种编程语言: Eclipse 使用 Java, Qt 使用 C++, OpenDev 使用 Python, LibreOffice 使用 C++和 Java.

- 收集数据

在代码审查的各个阶段, 代码变更在 Gerrit 系统中呈现不同状态. 完整的代码审查流程结束后, 各个代码变更的最终状态一般分为两种: 合并或拒收. 本文收集的原始数据包含 4 个项目中合并和拒收的代码变更. 本文收集数据的过程如图 3 中“数据收集”部分所示, 具体包含如下步骤.

1) 使用 Gerrit 系统提供的 REST APIs (https://gerrit.wikimedia.org/r/Documentation/rest-api.html), 收集 4 个开源项目中, 状态为合并和拒收的代码变更(截至 2020 年 5 月 19 日), 获取其元数据信息, 包括代码变更标题(“subject”属性的值)、创建时间(“created”属性的值)、更新时间(“update”属性的值, 即代码变更最终被拒收或合并的时间)、评论消息列表(“messages”属性的值, 该值包含一条及以上的评论消息). 每条评论消息包含元数据信息, 比如该条消息的创建日期(“date”属性的值)、消息内容(“message”属性的值). 如果消息内容中包含关键词“Restored\n”, 表明该代码变更被恢复, 该条消息的创建日期即为该代码变更的恢复时间. 在创建时间早于该条恢复消息的消息列表中, 包含关键词“Abandoned\n”且离该条恢复消息创建时间最近的那条消息, 其创建日期即为该代码变更恢复之前被拒收的时间(abandonedTime). 最终得到 920 700 条代码变更, 见表 1.

2) 将收集到的代码变更分为两类: “恢复”和“未恢复”, 该步骤得到的代码变更集合为数据集 1 (dataset 1), 见表 2.

3) 从每个项目“恢复”的代码变更集合中随机选取 200 条“合并”的代码变更和 200 条“拒收”的代码变更. 因此, 每个项目总共选取 400 条代码变更, 4 个项目最终得到 1 600 条“恢复”的代码变更. 该步骤得到的代码变更集合为数据集 2 (dataset 2), 见表 3.

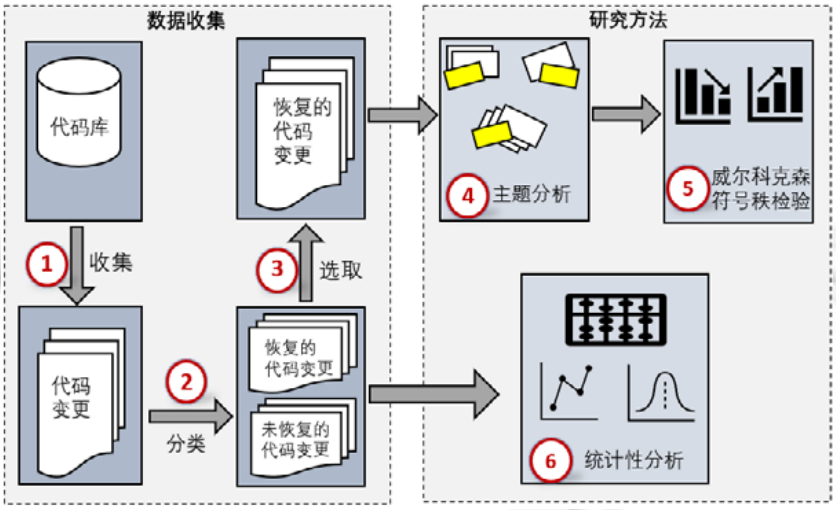

图 3 数据收集与分析方法框架

表 1 4 个开源项目的数据集

项目	是否恢复	代码变更数量	最终状态	代码变更数量	代码变更数量	代码变更总数量
Eelipse	是	232	合并	62 500	84 000	920 700
	否	62 268				
	是	218	拒收	21 500		
	否	21 282				
LibreOffice	是	396	合并	77 000	83 400	
	否	76 604				
	是	200	拒收	6 400		
	否	6 200				

表 1 4 个开源项目的数据集(续)

项目	是否恢复	代码变更数量	最终状态	代码变更数量	代码变更数量	代码变更总数量
Qt	是	890	合并	138 600	172 000	920 700
	否	137 710				
	是	457	拒收	33 400		
	否	32 943				
OpenDev	是	5 044	合并	558 700	581 300	
	否	553 656				
	是	672	拒收	22 600		
	否	21 928				

表 2 数据集 1

项目	未恢复过的代码变更数量	恢复的代码变更数量	总计
Eclipse	83 550	450	84 000
LibreOffice	82 804	596	83 400
OpenDev	575 584	5 716	581 300
Qt	170 653	1 347	172 000
总计	912 591	8 109	920 700

表 3 数据集 2

项目	Eclipse		LibreOffice		OpenDev		Qt	
状态	拒收曾被恢复	合并曾被恢复	拒收曾被恢复	合并曾被恢复	拒收曾被恢复	合并曾被恢复	拒收曾被恢复	合并曾被恢复
数量	200	200	200	200	200	200	200	200
总计	400		400		400		400	

2.2 研究方法

1) 主题分析

针对 RQ1, 采用主题分析方法(thematic analysis)[24]对数据集 2 (dataset 2)中的代码变更分析其恢复原因, 如图 3 中的步骤 4 所示. 具体而言, 主要包括以下步骤.

- 编码(coding): 对于每条代码变更, 仔细阅读其标题、描述信息和评论信息, 将描述代码变更恢复的原因的语句提取出来, 放在卡片上.
- 开放卡片分类(open card sorting): 将相似含义的卡片放在一起, 相似含义的卡片组成一个集合, 这样就得到不同的集合, 最后对这些集合进行命名. 本文第一作者和一个研究生各自独立地完成了数据的编码和集合命名. 本文使用 Fleiss Kappa[25]来衡量两个标记者之间的意见一致性, 两个标记者分类结果的 Kappa 值为 0.67. 该值表明, 两个标记者基本上同意对方的分类. 最终, 标记者对存在分歧的分类进行讨论并达成共识.

在分类的过程中, 发现有些代码变更被恢复时, 开发者并未说明其恢复的原因, 并且从代码变更的讨论区上下文也无从判断其原因. 此类型一般有两种情况: (1) 代码变更讨论区的内容太少, 无法确定其被恢复的原因; (2) 代码变更讨论区的内容较多, 但是并未对恢复的原因进行任何解释, 且根据上下文也无法判断其原因. 对于此类型的代码变更, 有些开发者或审查者也并不知道其为何恢复. 例如, 针对 Eclipse 项目中的代码变更#102024, 此代码变更由代码贡献者本人拒收, 之后被另一位开发者恢复. 值得注意的是, 代码变更拒收既可以由代码贡献者进行, 也可以由项目维护者进行. 而贡献者并不清楚开发者为何恢复此代码变更, 于是直接询问其恢复的原因. 具体如下所示:

*Eclipse # 102024: Hi Lars, not sure why you restored this change, but please be aware that it is not actual.

本文去除这 260 条(16.25%)未知因素导致恢复的代码变更, 对剩余的 1 340 条代码变更进行分类统计, 最终得到了 11 个恢复审查的原因类型, 见表 4.

表 4 代码变更恢复的类别

类别	描述
提升改进	代码变更已经提升改进或将要改进
重新活跃	代码变更因为长时间没有更新, 被代码审查系统清理拒收, 之后, 开发者由于某些因素继续此代码变更的工作
临时拒收	代码变更因为某些因素而被临时拒收, 待条件具备后, 恢复以继续审查
错误操作	代码变更因为开发者的失误而被错误拒收, 恢复以继续审查
拒绝拒收	代码变更被一个开发者拒收后, 另一个开发者不同意拒收, 将其恢复以继续审查
改变主意	代码变更被开发者拒收后, 由于某些因素, 开发者改变主意使其恢复以继续审查
解决分歧	代码变更因为合并分歧被拒收, 解决合并分歧问题后恢复以继续审查
更改分支	代码变更由一个分支移动到另一个分支
再次测试	代码变更恢复是为了再次进行某项测试
重新编译	代码变更提交之后, 编译不通过, 开发者拒收后将其恢复以重新编译
其他	代码变更因为其他原因而被恢复, 以继续接受审查

2) 威尔科克森符号秩检验

针对 RQ1, 使用威尔科克森符号秩检验方法, 分析不同项目间代码变更被恢复的原因分布差异, 如图 3 中的步骤 5 所示. 具体而言, 统计每个项目中, 每个原因类别的代码变更数量, 项目间进行原因类型分布的两两比较, 并计算 *p*-value 值.

3) 量化分析

针对 RQ2, 本文基于数据集 1 (dataset 1), 采用量化方法进行分析. 具体而言, 分别统计不同类型代码变更的评论数量、审查者数量, 并计算接收率、审查时间和拒收后恢复时间, 采用以下公式进行计算.

- 接收率

$$Acceptance\ Rate=acceptedChange/allStudiedChange \tag{1}$$

其中, *acceptedChange* 为被接收的代码变更数量, *allStudiedChange* 为所有的代码变更数量.

- 审查时间

$$reviewTime=outcomeTime-createdTime \tag{2}$$

其中, *createdTime* 为代码变更创建时间, *outcomeTime* 为代码变更最终拒收或其最终合并的时间. 判定最终拒收或合并的方法包含两种: (1) 对于每条代码变更, 在 Gerrit 的 Web UI 界面(图 2 所示)左上角显示的状态值(merged 或 abandoned), 即为截至查询时间点时该条代码变更的最终状态; (2) REST API 爬取的数据集中, 每个代码变更有一个状态值(status) (ABANDONED 或 MERGED), 即为截至查询时间点时, 该代码变更的最终状态. 本文采用第 2 种方法获取每条代码变更的状态值. Fan 等人[12]的研究工作也采用该方法取得代码变更的状态. 第 1 种方法需要分析每个代码变更的 Web UI 界面以获取代码变更状态, 这种方法工作量较大.

- 拒收后恢复时间

$$interval=restoredTime-abandonedTime \tag{3}$$

其中, *restoredTime* 为代码变更恢复的时间, *abandonedTime* 为代码变更恢复之前被拒收的时间. 判定代码变更在之前被拒收的方法包含两种: (1) 对于每条代码变更, 在其 Gerrit 的 Web UI 界面(如图 2 所示), 在代码变更的评论区(界面正下方)出现关键词“Abandoned”, 且该关键词单独占一行; (2) REST API 爬取的数据集中, 评论信息以关键词“Abandoned\n”开头, 表明开发者进行的拒收操作. 本文采用第 2 种方法获取每条代码变更的评论信息. 第 1 种方法需要分析每个代码变更的 Web UI 界面以获取评论区信息, 这种方法工作量较大.

在所有恢复的代码变更中, 约 8%的代码变更会经过多次(即大于 1 次)拒收和恢复. 由于多次恢复的数据量较少, 绝大部分(约 92%)恢复的代码变更都只经历一次恢复过程. 本文对多次恢复的代码变更, 只考虑其第 1 次拒收和第 1 次恢复的操作.

3 结果及分析

3.1 RQ1: 代码变更恢复原因

本文总结出 11 个类别的代码变更被恢复的原因, 每个类别的比例见表 5.

表 5 代码变更恢复的原因类别所占比例

类别	数量	比例(%)
提升改进	433	32.32
重新活跃	214	15.97
重新编译	130	9.70
再次测试	128	9.55
改变主意	121	9.03
拒绝拒收	74	5.52
临时拒收	61	4.55
更改分支	54	4.03
错误操作	48	3.58
解决分歧	13	0.97
其他	64	4.78

通过以上统计数据可知, 占比前 5 的类别依次是“提升改进”“重新活跃”“重新编译”“再次测试”“改变主意”, 而占比最小的是“解决分歧”. “提升改进”类型占比最大, 32.32%的代码变更因为开发者继续提升改进而被恢复; 其次, “重新活跃”的代码变更也较多, 代码变更如果长时间没有动态更新, 代码审查平台会自动地把此代码变更拒收, 之后, 开发者或审查者如果需要继续此代码变更, 可以随时恢复, 15.97%的代码变更因“重新活跃”而被恢复; 9.70%的代码变更因为编译问题, 它们没有通过自动集成测试, 故开发者拒收然后恢复,以重新进行自动集成测试; 9.55%的代码变更是由于需要再次测试而被恢复; 9.03%的代码变更是因为开发者改变了主意而被恢复; 只有 0.97%的代码变更是因为合并分歧问题而被恢复.

本文针对这 11 个类别分别做出详细的展示和说明如下.

(1) 提升改进

代码变更拒收之后, 有时因为此代码变更比较重要或尚有意义, 开发者对此代码变更做了进一步改进或将要改进提升, 此类代码变更就会被恢复以继续接受审查. 例如, 对于 Eclipse 项目中的代码变更#101472, 此代码变更因为并不是一个很好的实现而被拒收. 拒收一段时间之后, 开发者将其恢复, 因为开发者对其进行了重写改进. 最终, 该代码变更被合并.

此类型的数据在本文研究的所有分类代码变更中占比 32.32%, 典型的例子如下所示:

*Eclipse#90751: Restored. Just to push an update.

*LibreOffice#14507: Restored. Perhaps found a way to apply the Eike’s advice, working on it.

*LibreOffice#14889: Abandoned, messed up white spaces. Restored, cleaned the needless whitespace changes.

*OpenDev#722209: Abandoned, missing related bug and description. Restored, add related bug and description.

*OpenDev#720153: Restore, to update.

文献[15]在研究代码变更拒收的因素时, 发现有些代码变更因为不完整或不完善而被拒收. 这与本文的发现具有时间上的相关性, 即这些不完整或不完善的拒收代码变更在被“提升改进”后, 可能被恢复并再次接受审查.

(2) 重新活跃

代码变更因长时间没有更新, 有时是因为没有审查者进行审查, 或者审查者给出修改建议而代码贡献者长时间没有回复, 此类代码变更将被审查系统定时清理拒收. 之后, 如果代码贡献者或审查者有时间或有兴趣继续此代码变更, 那么此代码变更便会恢复以继续接受审查. 例如, 对于 Eclipse 项目中的代码变更#25821, 由于其超过 4 个月没有状态更新被拒收, 然而有开发者对此代码变更感兴趣不愿其拒收, 故代码变更被恢复. 有时代码变更因为缺少审查而被拒收, 而当审查者有时间审查该代码变更时, 代码变更便会恢复. 例如, Eclipse 项目中代码变更#45274, 代码贡献者上传代码变更后, 审查者并没有注意到此代码变更, 此代码变更超过 7 个月的时间都没有受到审查, 故被开发者拒收. 当审查者注意到此代码变更时, 对其进行恢复以继续接受审查. 针对 LibreOffice 项目中的代码变更#73417, 此代码变更在提交两个月内无状态更新, 故被代码审查

系统自动检测后拒收. 由于是自动拒收, 同时开发者认为此代码变更可能仍在审查过程中, 故恢复此代码变更. OpenDev项目也会定期检查没有动态更新的代码变更, 如果代码变更超过4周没有动态, 就会被自动拒收. 例如代码变更#704315, 由于代码变更提交后超过30天没有动态更新, 故被审查者拒收. 然而代码贡献者依然希望此代码变更合并, 故恢复此代码变更以继续接受审查. Qt 项目也会定期对代码变更进行活跃性检测, 如果长时间没有更新, 代码变更会被自动拒收. 例如代码变更#188610, 此代码变更长时间没有动态而被自动拒收后又重新恢复.

此类型的代码变更在本文研究的所有分类代码变更中占比 15.97%, 典型的例子如下所示:

*Eclipse#127969: Abandoning after 6 months or more of inactivity. If this change is still needed, please restore it. Restored, I am still waiting for someone to have a look at this.

*LibreOffice#38231: Restored. First of all, I sincerely apologize for not updating patch since a year. I was busy in my work.

*Opendev#708626: Abandoned, the magnum team is cleaning up the backlog of changes older than 30 days. Feel to restore your patch. Restored, this is a trivial fix, still valid.

*Qt#183802: Abandoned, automatic cleanup after prolonged inactivity. Restored, back to work on it.

对于长时间没有动态更新的代码变更, 代码审查系统会自动清理这些非活跃的代码变更, 以减少代码审查平台服务器的负荷(https://charm.cs.illinois.edu/gerrit/Documentation/user-change-cleanup.html#auto-abandon). 一旦有开发者对此类代码变更进行改进更新, 此类代码变更便可以恢复以继续接受审查.

(3) 重新编译

代码变更在提交之后, 代码审查系统会利用 CI (continuous integration, 持续集成)对其进行自动化地集成测试. 但有些代码变更在该过程中并不能编译通过, 于是开发者就拒收该代码变更, 之后恢复之以重新进行编译. 有些代码变更在多次编译失败后, 开发者拒收然后再恢复以重新编译. 例如 LibreOffice 项目中的代码变更#51494, 代码贡献者提交了 5 次均编译失败, 于是贡献者拒收代码变更之后恢复, 再次提交以重新编译.

此类型的代码变更在本文研究的所有分类代码变更中占比 9.70%, 典型的例子如下所示:

*Eclipse#33987: Restored. Try to pass build again.

*Eclipse#154324: Abandoned. Need to rerun the build. Restored. Build rerun.

*OpenDev#721606: Abandoned. Failing, let’s restart. Restored. Starting again.

*OpenDev#715324: Abandoned. Have to recheck due to molecule failure. Restored, restore to restart check/gate.

当代码变更出现编译错误时, 代码贡献者可以拒收代码变更, 然后恢复以重新进行编译. 文献[18]发现, 有些拉取请求被重新开放, 是因为其在编译时出现了错误. 这和本文的发现一致. 此外, 文献[26]发现: 针对编译和测试错误的缺陷, 相关解决方案常常被标记为“变通(workarounds)”. 变通是指当最明显的缺陷解决方案不可能达到时, 所采用的避免或解决缺陷的方案[26].

(4) 再次测试

即代码变更恢复为了再次进行某项测试. 大部分此类的代码变更会在标题上注明“测试”, 有些会在标题中注明“测试, 不要合并”. 例如在Eclipse项目中的代码变更#88029, 其标题就注明了“Test commit”, 即测试提交, 其描述为“Investigating why hudson builds are not being triggered. DO NOT MERGE”, 即调查为什么自动集成编译没有被触发, 并注明不要合并.

此类型的代码变更在本文研究的所有分类代码变更中占比 9.55%, 典型的例子如下所示:

*Eclipse#58374: Restored. Restore for test.

*Eclips#38490: Restored. Testing if “restore” triggers hudson build.

*LibreOffice#37593: DUMMY patch, please ignore. This patch, will not be merged, it is only here to allow testing of the automated gerrit interface. Please do not comment on this patch. Restored, test.

*Opendev#668028: Restored. Reopen for testing purposes.

*Qt#279364: CI test. Do not review, for test only.

有些代码变更被拒收是由于代码变更仅用作测试用途[18]. 大部分用作测试用途的代码变更最终被拒收, 这些代码变更通常会在标题中注明其测试用途, 不需要被审查和合并. 在小部分测试用途的代码变更中, 标题并未注明其是用于测试的, 仅恢复原因中包含“Test”等关键词.

(5) 改变主意

即代码变更被开发者拒收后, 由于某些因素开发者改变主意使其恢复以继续审查. 比如开发者之前感觉其已不重要或已无意义, 后来发现其依然重要, 于是将其恢复. 例如 Eclipse 项目中的代码变更#102439, 开发者拒收该代码变更, 是因为该代码变更实现的特征针对 Photon 系统. 之后, 开发者将其恢复, 并且明确解释了恢复原因: “我们改变了主意, 我们也将把它集成到 Oxygen 系统.”

此类型的代码变更在本文研究的所有分类代码变更中占比 9.03%, 典型的例子如下所示:

*Eclipse#110034: Abandoned, this is not required till the content is modified. Restored, this change is a required change.

*LibreOffice#93898: Let’s abandon since I don't measure the impact. Restored, let’s give it a chance. At least, it can be reverted.

*Opendev#714041: Abandoned. Abandon in favor of https://review.opendev.org/714409-easier, faster, cleaner. Restored. Since https://review.opendev.org/714409 was abandoned, restoring this one.

*Qt#196482: Abandoned. This isn’t necessary. Restored. Actually required.

关于开发者改变主意而恢复代码变更, 文献[18]中也有类似的发现, 即开发者改变主意, 重新开放了拒收的拉取请求.

(6) 拒绝拒收

即代码变更被一个开发者拒收后, 另一个开发者不同意拒收, 之后恢复以继续审查. 这种代码变更往往是两个开发者存在分歧意见. 例如 Eclipse 项目中的代码变更#92967, 开发者将其拒收, 因为该代码变更已被应用在另一分支上. 随后, 审查者将其恢复, 原因在于为了保持代码完整性, 需要将该代码变更在当前分支进行合并.

此类型的代码变更在本文研究的所有分类代码变更中占比 5.52%, 典型的例子如下所示:

*Eclipse#14949: Abandoned. This is already tracked on https://git.eclipse.org/r/#/c/14915/. Restored. No, it’s not. Did you look at it? This change simplifies the API considerably. It’s worth looking at.

*LibreOffice#63388: Restored. This is the better version.

*Opendev#678443: Abandoned. abandoning for now as we don’t need it I think. Restored, Cédric seems to think it’s a good idea, let’s keep it.

*Qt#206598: Why did you abandon? Please restore and I’ll review soon.

(7) 临时拒收

代码变更因为某些因素而被临时拒收, 待条件具备后恢复以继续审查. 例如 OpenDev 项目中的代码变更#706807, 该代码变更依赖于另一个尚未被合并的代码变更, 故被临时拒收, 以等待另一个代码变更合并. 在被拒收后一段时间, 依赖的代码变更被合并, 该代码变更也随之恢复.

此类型的代码变更在本文研究的所有分类代码变更中占比 4.55%, 典型的例子如下所示:

*LibreOffice#74337: It requires https://gerrit.libreOffice.org/#/c/74043/ before. Restored, required change is merged.

*Eclipse#68261: Abandoned. Res (Restore) it when you feel like it. Please only consider this patch after https://git.eclipse.org/r/#/c/61062/ is merged.

*OpenDev#679401: Abandoned. Temporarily abandoning. Will restore.

*OpenDev#669167: Abandoned. Holding off as we didn't get any positive feedback about backporting this downstream, I might reopen these once the change has been released in Train for a while.

*Qt#240519: Abandoned. What a waste of resources. I'll revive this here later once the usages of the 'duplicated' members have been removed due to the intended use of aspects.

在某些条件满足时, 临时拒收的代码变更会被恢复以继续接受审查. 对于临时拒收的代码变更, 其他开发者可不去改进或者审查, 从而减少了不必要的时间消耗.

(8) 更改分支

即代码变更由一个分支移动到另一个分支之后继续接受审查. 此类代码变更一般有两种情况: (1) 代码变更提交到错误分支上, 开发者拒收该代码变更, 之后恢复之以便将其移动到正确的分支上; (2) 代码变更在某个分支上被拒收, 但是可以尝试应用到其他分支上. 例如Qt项目中的代码变更#247598, 该代码变更因提交到错误的分支上而被拒收, 恢复的原因是为了将其移动到正确分支上面.

此类型的代码变更在本文研究的所有分类代码变更中占比 4.03%, 典型的例子如下所示:

*Eclipse#112506: Abandoned. Sorry, this is targeting the wrong branch.

*LibreOffice#68483: Abandoned. Wrong target branch. Restored. Resurrect, this should go into 6.1 too.

*Opendev#713978: Abandoned. Sent to wrong branch by mistake.

*Qt#288306: Abandoned. Wrong branch. Restored. Change destination moved from dev to 5.15. Was not meant for dev.

文献[18]与本文有相似的发现, 拉取请求重新开放的一个原因, 是提交到错误的分支上.

(9) 错误操作

即代码变更因为贡献者或审查者的失误操作而被错误拒收. 比如代码变更贡献者因为不熟悉 Gerrit 审查系统的一些操作规则而错误地拒收某些代码变更. 例如 Qt 项目中的代码变更#247805, 为了更新代码变更,代码贡献者拒收了该代码变更, 并提交了一个新的代码变更. 审查者建议该开发者不要使用先拒收、再创建新的代码变更的方式, 而应该对原有代码变更进行更新, 采用相同的 Change-Id.

此类型的代码变更在本文研究的所有分类代码变更中占比 3.58%, 典型的例子如下所示:

*Eclipse#80159: Restored. Abandoned wrong patch, this one is good for review.

*LibreOffice#73484: Restored, abandoned the wrong one, sorry!

*OpenDev#718835: Restored, seems like I accidentally abandoned this.

*Qt #250053: Abandoned, squashed. Restored, oops this is the squashed one now.

与本文的发现相似, 文献[15]中发现: 代码变更被拒收的一个重要原因, 在于开发者进行的一些错误操作, 比如错误地更新代码变更. 为降低操作错误率, 可提供检查清单, 供开发者检查并避免常见错误操作.

(10) 解决分歧

即代码变更因为合并分歧被拒收, 解决合并分歧问题后恢复以继续审查, 或恢复之后再去解决分歧. 例如 Eclipse 项目中的代码变更#87143, 因存在合并分歧被拒收. 当分歧解决后, 该代码变更被恢复并继续接受审查.

此类型的代码变更在本文研究的所有分类代码变更中占比 0.97%, 典型的例子如下所示:

*Eclipse#158130: Restoring and then to resolve conflicts.

*LibreOffice#55157: Abandoned, merge conflict. Restored, try to rebase it.

*LibreOffice#82693: Abandoned. seems to conflict with another tip. Will try this one again, after the other one is accepted. Restored, same as before -- but now without conflict.

*Qt#271890: Abandoned, conflicted commit.

与本文的发现相似, 文献[15]中发现, 代码变更拒收的原因包括代码变更合并分歧的问题. 文献[18]发现, 拉取请求被重新开放的一个原因是存在合并分歧的问题.

(11) 其他

代码变更因为其他因素而被恢复以继续接受审查. 例如, 针对 Eclipse 项目中的代码变更#103853, 因为此代码变更修复的缺陷是一个无效的缺陷, 故该代码变更被拒收. 一段时间后, 该缺陷被重新开放, 故该代码变更被恢复重新接收审查. 有些开发者不希望自己提交的代码变更拒收, 他们通过各种方法对自己的代码变更进行改进, 故他们会恢复拒收后的代码变更. 例如 LibreOffice 项目中的代码变更#14503, 开发者强烈希望该代码变更接收, 因此恢复了该代码变更. 此外, 部分代码变更被开发者拒收之后, 其他开发者希望继续此代码变更, 于是进行接管.

此类型的代码变更在本文研究的所有分类代码变更中占比 4.78%, 典型的例子如下所示:

*Eclipse#144160: Restored. Seems I cannot update a related change without restoring this one.

*Eclipse#110914: Abandon due to low priority. Restored. Restore due to increased priority.

*Qt#240521: Restored. Reuse for different change.

*Qt#172058: Restored. Jake will take over this change.

本文统计了每个项目中的代码变更恢复原因的分布图, 如图 4 所示. 代码变更恢复原因在每个项目中呈现不同分布: 在 LibreOffice 项目中, “重新活跃”类别代码变更占比很大; 在 OpenDev 中, “重新活跃”类别代码变更占比很小. 这可能是由于 OpenDev 总的代码变更数量多, 且 OpenDev 每天新增代码变更数量均值高于 LibreOffice 每日新增代码变更均值. OpenDev 项目参与者较多, 其社区活跃度更高, 存在较少不活跃代码变更被拒收的现象, 进而使得重新活跃而后恢复的代码变更较少. 值得注意的是, 在所有项目中, “提升改进”类别是所占比例较大的一个类别. 为了验证不同项目间代码变更恢复的原因分布差异是否具有统计学意义, 进一步使用威尔科克森符号秩检验方法. 结果发现: *p-value* 值均大于 0.05(最小值: 0.28, 最大值: 0.96, 中位数: 0.65), 这表明差异并不显著.

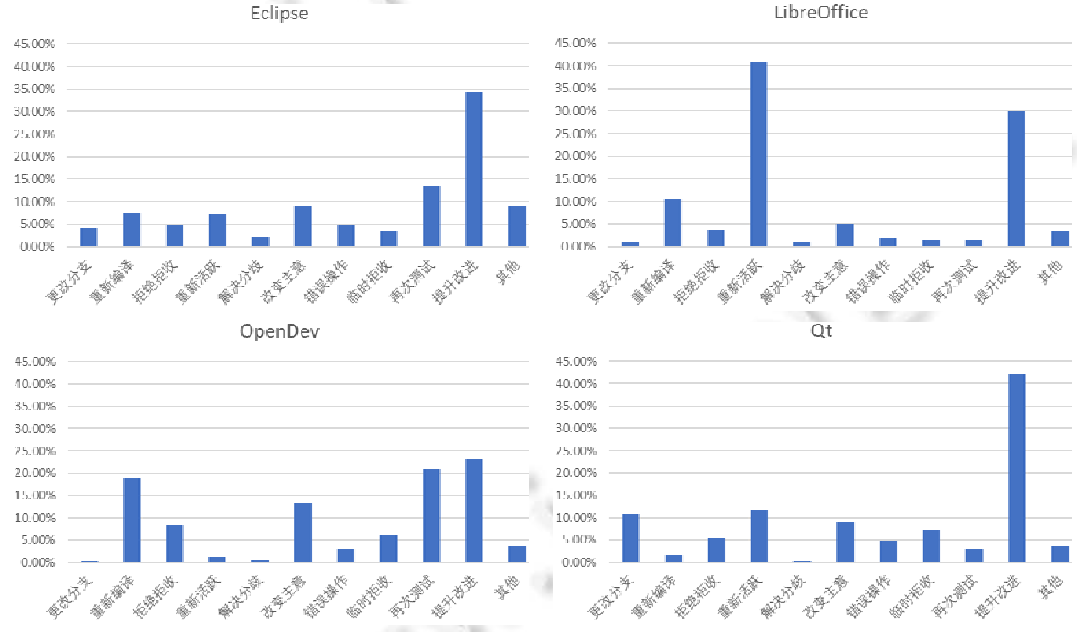

图 4 代码变更恢复的原因类别分布

3.2 RQ2: 恢复的代码变更的特征

3.2.1 代码变更接收率

表 6 展示了 4 个项目的恢复代码变更接收率以及未恢复代码变更的接收率. 表 6 可知, 对于 LibreOffice 项目, 未恢复过的代码变更接收率(92.51%)高于恢复的代码变更接收率(66.44%). 其他 3 个项目也有此规律.

由此可知, 相比未恢复过的代码变更, 恢复的代码变更接收率更低. 恢复的代码变更曾经被拒收过, 其恢复之后, 其接收的可能性低于从未恢复过的变更. 可能的原因是它曾经拒收过, 说明它曾经存在某些问题, 不符合项目标准, 那么它被恢复之后, 如果其存在的问题没有很好地解决, 就很有可能会再次被拒收.

表 6 恢复的代码变更及未恢复的代码变更的接收率

项目	未恢复过的代码变更的接收率(%)	恢复过的代码变更的接收率(%)
Eclipse	74.53	51.56
LibreOffice	92.51	66.44
Opendev	96.19	88.24
Qt	80.70	66.07
跨项目	90.98	80.92

表 7 展示了代码变更恢复之后的审查结果统计. 在 OpenDev 项目中, 88.24%的恢复后的代码变更被接收, 11.76%的代码变更被拒收. 在 Qt 和 LibreOffice 项目中, 约 66%的代码变更在恢复后被接收, 约 34%的代码变更恢复后依然被拒收. 在 Eclipse 项目中, 51.56%的代码变更在恢复后被接收, 48.44%的代码变更恢复之后依然被拒收. 从跨项目的角度来看, 约 81%的代码变更在拒收后被接收, 约 19%的代码变更在恢复后仍然被拒收. 由此可见, 代码变更恢复之后, 大多数代码变更被接收.

表 7 代码变更恢复后的审查结果统计

项目	代码变更恢复后的接收率(%)	代码变更恢复后的拒收率(%)
Eclipse	51.56	48.44
LibreOffice	66.44	33.56
Opendev	88.24	11.76
Qt	66.07	33.93
跨项目	80.92	19.08

3.2.2 代码变更评论数量

恢复的代码变更与未恢复的代码变更的评论数量平均值和中位数见表 8. 由表 8 可知, 相比从未恢复过的代码变更, 恢复的代码变更的评论数量的平均值和中位数更大. 在 Eclipse 项目中, 恢复的代码变更的评论数量平均值为 23.6, 而未恢复变更的评论数量平均值为 9.4, 前者远远大于后者, 恢复的代码变更的中位数(15.0)也高于未恢复代码变更评论数量的中位数(6.0). 其他 3 个项目也有此规律. 对于跨项目数据而言, 恢复的代码变更的评论数量平均值(51.0)约为未恢复代码变更评论数量平均值(17.6)的 2.9 倍, 前者中位数(24.0)是后者中位数(10.0)的 2.4 倍. 由此可见, 恢复的代码变更的评论数量更多. 可能的原因是, 代码变更提交后,经过一轮审查后被拒收或合并. 然而一些拒收后的代码变更可以恢复继续接受审查, 这就导致恢复的代码变更需要进一步的讨论和改进, 因此其评论数量更多.

表 8 代码变更评论数量的平均值和中位数

项目	代码变更评论数量平均值(单位: 条)		代码变更评论数量中位数(单位: 条)	
	恢复的代码变更	未恢复的代码变更	恢复的代码变更	未恢复的代码变更
Eclipse	23.6	9.4	15.0	6.0
LibreOffice	19.4	7.5	14.0	5.0
OpenDev	61.8	22.0	29.0	12.0
Qt	28.4	11.4	19.0	7.0
跨项目	51.0	17.6	24.0	10.0

3.2.3 代码变更审查者数量

4 个开源项目中, 代码变更审查者的统计信息见表 9. 相比从未恢复过的代码变更, 恢复过的代码变更审查者数量几乎无明显变化. 例如在 Eclispe 项目中, 恢复的代码变更与未恢复的代码变更的审查者数量的平均值和中位数都相等, 分别为 2.6 和 2.0. 其他几个项目类似, 其恢复的代码变更审查者数量的平均值和中位数与未恢复过的代码变更的平均值和中位数差别不显著. 对于跨项目数据而言, 恢复的代码变更的审查者数量中位数(4.0)和未恢复代码变更审查者数量中位数(4.0)相同, 二者平均值也相同. 由此可见, 尽管代码变更在拒收后又被恢复, 但是其总的审查者数量并没有明显变化. 此现象可能的原因是, 开源社区中熟悉某一模块

的开发者负责自己熟悉的业务, 这样能有效降低资源浪费, 保证开源项目的开发效率和产出能力.

表 9 恢复的和未恢复的代码变更审查者数量的平均值和中位数

项目	代码变更审查者数量平均值(单位: 个)		代码变更审查者数量中位数(单位: 个)	
	恢复的代码变更	未恢复的代码变更	恢复的代码变更	未恢复的代码变更
Eclipse	2.6	2.6	2.0	2.0
LibreOffice	2.9	2.8	3.0	3.0
OpenDev	6.2	6.4	4.0	5.0
Qt	3.5	3.9	3.0	4.0
跨项目	5.3	5.3	4.0	4.0

3.2.4 代码变更审查所用时间

本文分析统计了恢复的代码变更审查所用时间, 结果见表 10. 相比从未恢复的代码变更, 恢复的代码变更审查的时间更久. 例如在 Eclipse 项目中, 恢复的代码变更所用审查时间的平均值和中位数分别为 145.78 天、16.02 天, 未恢复的代码变更审查所用时间的平均值和中位数分别为 28.20 天、0.52 天. 对于跨项目数据而言, 恢复的代码变更的平均值(136.26)约为未恢复变更审查所用时间平均值(20.12)的 6.8 倍, 前者中位数(24.55)是后者中位数(1.66)的约 14 倍. 相比未恢复的代码变更, 恢复的代码变更的平均值和中位数都更大. 由此可见, 恢复的代码变更需要更久的审查时间. 这一现象可能的原因是, 约 16%的代码变更因为长时间没有更新而被系统拒收, 之后开发者恢复以继续此工作. 这类代码变更往往所用审查时间更长, 所以这可能也导致恢复代码变更的审查时间平均值远大于其中位数.

表 10 代码变更审查时间的平均值和中位数

项目	代码变更审查时间平均值(单位: 天)		代码变更审查时间中位数(单位: 天)	
	恢复的代码变更	未恢复的代码变更	恢复的代码变更	未恢复的代码变更
Eclipse	145.78	28.20	16.02	0.52
LibreOffice	83.02	4.10	21.61	0.47
OpenDev	106.70	18.66	23.45	2.48
Qt	410.23	36.49	110.66	1.11
跨项目	136.26	20.12	24.55	1.66

3.2.5 代码变更拒收后的恢复时间

代码变更拒收后的恢复时间的统计结果见表 11. 代码变更自拒收到恢复, 其时间间隔的平均值在 550 小时(约 23 天)–1 913 小时(约 80 天)之间, 但是其时间的中位数较小, 从 1 小时左右到 22 小时左右. 这说明一些代码变更在拒收后很快就被恢复, 而有一些代码变更在拒收后需要很长的一段时间才被恢复.

表 11 代码变更拒收后的恢复时间统计

项目	代码变更恢复时间平均值(单位: 小时)	代码变更恢复时间中位数(单位: 小时)
Eclipse	875.55	1.22
LibreOffice	639.60	14.04
OpenDev	550.64	13.54
Qt	1 913.37	22.36
跨项目	801.57	14.68

3.2.6 不同类别之间恢复的代码变更特征

表 12 展示了不同类别恢复的代码变更的接收/拒收率, 以及评论数量、审查者数量、审查所用时间和拒收后恢复时间的中位数及平均值. 对于接收率而言, “拒绝拒收”和“提升改进”的恢复代码变更排名前二, 分别为 67.6%和 66.7%. “拒绝拒收”的恢复代码变更接收率排名第一, 可能的原因是, 大量代码贡献者拒收了自己提交的代码变更, 审查者认为代码变更具价值, 希望继续对代码变更进行审查. “提升改进”的恢复代码变更接收率排名第二, 原因或是由于这些代码变更被进一步改进, 从而得到完善.

对于拒收率而言, “再次测试”的恢复代码变更拒收率最高, 为 95.3%. 原因是, 大部分此类代码变更仅用于测试任务. 4.7%的“再次测试”的恢复代码变更, 其主要功能并非测试, 因此这些恢复代码变更被接收.

对于评论数量而言, “更改分支”的恢复代码变更评论数量最少(中位数: 11.0, 平均数: 16.7). 对于审查者数量而言, “临时拒收”的恢复代码变更, 审查者数量最多(中位数: 3.0, 平均数: 4.4). 从审查所用时间来看, “重

新活跃”的恢复代码变更审查所用的时间最长(中位数: 162.8 天, 平均数: 329.0 天), 此类代码变更长久没有动态更新, 因此其审查过程较长. 从拒收后恢复时间来看, “重新编译”的恢复代码变更, 拒收后恢复时间最短(中位数: 0.03 小时, 平均数: 36.2 小时), “重新活跃”类别的恢复代码变更拒收后恢复的时间最长(中位数: 115.9 小时, 平均数: 1 398.6 小时).

表 12 不同类型恢复代码变更的特征

类别	接收率和拒收率(%)		评论数量 (单位: 个)		审查者数量 (单位: 个)		审查所用时间 (单位: 天)		拒收后恢复时间 (单位: 小时)	
	接收率	拒收率	中位数	平均值	中位数	平均值	中位数	平均值	中位数	平均值
更改分支	42.6	57.4	**11.0**	**16.7**	2.0	2.7	3.1	82.4	1.5	1 186.5
重新编译	54.6	45.4	18.5	24.8	2.0	3.2	4.0	13.1	**0.03**	**36.2**
拒绝拒收	**67.6**	32.4	17.5	20.3	3.0	3.6	13.0	78.8	28.5	527.1
重新活跃	43.0	57.0	16.0	20.8	4.0	3.8	**162.8**	**329.0**	**115.9**	**1 398.6**
解决分歧	23.1	76.9	13.0	16.7	3.0	2.8	0.9	16.6	1.4	44.2
改变主意	55.37	44.6	18.0	24.7	3.0	3.5	9.1	73.4	6.4	422.8
错误操作	56.3	43.8	14.5	18.7	3.0	3.1	6.8	43.6	0.2	18.8
其他	45.3	54.7	19.0	31.3	3.0	3.1	8.1	43.5	0.8	138.6
临时拒收	57.4	42.6	22.0	27.9	**3.0**	**4.4**	27.6	87.9	28.2	550.9
再次测试	**4.7**	**95.3**	13.0	22.9	2.0	2.8	12.5	59.3	21.9	529.6
提升改进	**66.7**	33.3	16.0	24.8	2.0	2.6	9.9	62.8	3.8	595.5

3.3 结果启示

本节讨论 RQ1 与 RQ2 的研究结果给代码贡献者和代码审查平台带来的启示.

- 代码贡献者

RQ1 对代码变更恢复的原因进行分析和总结, 进而得出一些参考性建议.

1) “提升改进”: 代码贡献者最好能够在代码变更早期尽量改进完善, 避免在拒收后重新工作.
2) “重新活跃”: 代码贡献者在提交变更之后, 可以定期查阅代码变更审查进度, 查看审查者是否提出建议需要改进, 以此避免代码变更被代码审查系统自动清理.
3) “错误操作”: 开发者在拒收代码变更时, 应仔细确认此代码变更是否需要拒收, 避免因错误操作或错误理解而错误地或意外地将代码变更拒收. 一个常见的错误操作是贡献者对代码变更进行更新时的不当操作. 具体而言, 代码贡献者改进其提交的代码变更, 一般采用两种方式: 一种方式是拒收此代码变更, 修改后再重新提交一个新的代码变更; 另一种方式是直接修改此代码变更, 之后使用 git commit -amend 命令对此代码变更进行更新. 第 1 种方式需要拒收后再次提交新的代码变更, 增加了代码审查的时间; 此外, 需要重新分配审查者, 从而增加了人力资源消耗. 因此建议代码贡献者正确使用 git commit -amend 命令, 以减少代码审查平台的工作负荷.

- 代码审查平台

RQ1 对代码变更恢复的原因进行分析和总结, 研究结果可以为代码审查平台的管理维护者提供一些参考建议.

1) “重新活跃”: 对于长时间无更新的代码变更, 代码审查平台可定期提醒相应的代码贡献者, 包括电子邮件形式, 以避免代码变更因长久无更新而被拒收.
2) “临时拒收”: 对于需临时拒收的代码变更, 可在代码审查平台设置一种专门的状态进行标记, 并定期检查这些代码变更是否满足恢复条件; 一旦恢复条件满足, 便可立即恢复此种状态的代码变更.

此外, 本文在研究 RQ1 时发现, 有些开发者在恢复代码变更时并未说明恢复原因. 针对此问题, 代码审查平台可以在开发者进行恢复操作时对开发者进行提醒, 并推荐其恢复原因.

RQ2 的结果表明, 不同类别之间的代码变更的接收率存在差异. 因此, 代码审查平台可以对恢复的代码变更是否被接收进行预测, 以便代码贡献者尽早确定是否继续改进此代码变更, 从而降低代码审查的工作负荷, 提升代码审查的效率.

3.4 有效性影响因素分析

- 内部有效性

为了避免分析代码中的缺陷，对代码逻辑进行仔细检查，同时对部分分析结果做了手工校验，保证结果的可靠性. 另外，代码变更恢复因素的分类过程是人工进行的，包括本文的第一作者和一个研究生. 他们并不是代码审查领域的专家，也不是 4 个项目的参与者. 分类过程由领域专家来进行，结果可能会不同. 此外，代码变更从被拒收到恢复需要经过一段时间，数据集中的一些代码变更有可能在未来被恢复. 具体而言，本文收集了 4 个开源项目中截至 2020 年 5 月 19 日的状态为合并和拒收的代码变更，其中一些代码变更在当时未被恢复过，其有可能在未来被恢复. 这种数据不确定性也是影响本文结果有效性的一个因素，为了提升结果的有效性，未来的研究工作可加入条件来截断数据，例如截至某个时间前创建的代码变更.

- 外部有效性

本文选取了 Gerrit 系统中 4 个主流的开源项目，Gerrit 和所选取的项目无法代表所有的代码审查系统和项目. 此外，本文研究的 4 个项目均为开源项目，本文结论并不一定适用于非开源项目. 未来可考虑分析不同的代码审查系统以及非开源项目来减少此威胁.

4 相关工作

本节简要介绍代码变更拒收相关研究，以及开源项目中缺陷和拉取请求被关闭后又被恢复的相关工作.

文献[27]调查了开源软件开发者的声誉如何影响提交代码变更的审查结果. 研究发现，核心开发者提交的代码变更可以更快地收到反馈，审查过程时间更短，更有可能被接收. 文献[28]调查了技术和非技术因素如何影响代码审查. 研究发现，非技术因素对代码审查结果有很大的影响. 文献[25]分析了 Apache 项目中的 2 603 个补丁，发现规模小、完整且独立的补丁更容易被接收. 文献[11]对两个开源项目进行经验研究，发现小的补丁(最多改动 4 行)更容易被接收，大的补丁不容易被接收. 文献[29]对给定的补丁，提出方法来预测是否它会被接收. 文献[12]研究了 4 个开源项目，对每条数据提取 34 个特征来预测代码变更是否被拒收. 文献[30]发现，与核心贡献者相比，非核心贡献者提交的补丁更可能被拒收. 文献[15]针对 4 个开源项目研究代码变更被拒收的原因，发现重复性代码变更是导致代码变更拒收最主要的原因. 此外，有些代码变更只是临时性的拒收，这些代码变更可以被恢复，并重新接收审查. 针对长时间没有动态和更新的代码变更，代码审查平台会定期检查，进行临时性拒收；当开发人员继续对此类代码变更改进时，可恢复此类代码，以便继续接受代码审查.

近年来，一些工作研究缺陷[16,17,31]、拉取请求[18,32]被关闭后又被恢复的情况. 文献[16,17]提取特征来构建模型，以预测软件缺陷是否会被重新打开. 文献[32]对 GitHub 中的拉取请求提取特征来构建模型，以预测拉取请求是否会被重新开放. 文献[31]研究了影响软件缺陷重新开放的因素. 文献[18]分析了 GitHub 中拉取请求被关闭后重新打开的原因，及其对代码审查的影响. 由于文献[16,17,32]是构建预测模型，文献[18,31]是经验研究，本文也是一项经验研究. 而文献[31]是对重新开放的缺陷进行的经验研究，文献[18]是对重新开放的拉取请求进行的经验研究. 本文研究对象是代码审查中的代码变更，它是开源社区的代码贡献而非缺陷，它与拉取请求不同但相似，故本文与文献[18]进行了对比分析. 文献[18]研究的是 GitHub 中的拉取请求，而本文研究的是 Gerrit 中的代码变更，两个工作的研究对象不同.

此外，文献[18]与本文发现的异同点包括:

- 首先，对于分析总结的类别，本文结果有 11 个类别，如表 4 所示；文献[18]结果有 16 个类别.
- 与文献[18]相比，本文发现了 4 个新类别(重新活跃、再次测试、拒绝拒收和临时拒收)；文献[18]中有 2 个类别是本文中没有的发现，即测试失败(test fail)与合并提交(squash commits).
- 此外，本文中其他 7 个类别与文献[18]的 13 个类别可对应起来: 编译问题对应“编译错误(compilation errors)”，改变主意对应“改变主意(change minds)”，更改分支对应“错误分支(wrong branch)”与“变基与更改分支(rebase and change branch)”，错误操作对应“意外关闭(closed accidentally)”，解决分歧对应

“分歧(conflict)”, 其他对应“其他(other)”, 提升改进对应“缺陷(bug)”“测试不足(insufficient tests)”“版本不兼容(incompatible version)”“平台不兼容(incompatible platforms)”“配置错误(configuration errors)”和“需要澄清(clarification needed)”.

- 此外, 重新开放的拉取请求与恢复的代码变更具有相似特征: 接收率更低、评论数量更多以及审查时间更久.

5 总结与展望

本文研究了代码变更被恢复的原因和被恢复的代码变更的特征. 收集了 Gerrit 系统中, 4 个开源项目 920 700 条代码变更, 采用主题分析方法识别出 11 类代码变更恢复的原因, 定量分析被恢复的代码变更的特征. 根据研究中的发现, 总结出给代码贡献者、代码审查平台的启示. 未来工作可考虑设计机器学习预测模型, 预测潜在的代码变更恢复, 并开发相应的自动化工具. 此外, 未来的研究工作可以进一步探究代码变更恢复后的状态特征, 比如恢复后被接收的原因或恢复后被再拒收的原因. 未来的研究工作可以探索 GitHub 和 Gerrit 提交代码贡献的不同方式如何影响代码贡献恢复.

References:

[1] Barnard J, Price A. Managing code inspection information. IEEE Software, 1994, 11(2): 59−69. [doi: 10.1109/52.268958]

[2] Brothers LR. Multimedia groupware for code inspection. In: Proc. of the Conf. Record on Discovering a New World of Communications (SUPERCOMM/ICC ’92). IEEE, 1992. 1076−1081. [doi: 10.1109/ICC.1992.268149]

[3] Mashayekhi V, Drake JM, Tsai WT, Riedl J. Distributed, collaborative software inspection. IEEE Software, 1993, 10(5): 66−75. [doi: 10.1109/52.232404]

[4] Panichella S, Arnaoudova V, Penta MD, Antoniol G. Would static analysis tools help developers with code reviews? In: Proc. of the 22nd IEEE Int’l Conf. on Software Analysis, Evolution, and Reengineering (SANER). IEEE, 2015. 161−170. [doi: 10.1109/SANER.2015.7081826]

[5] Floyd B, Santander T, Weimer W. Decoding the representation of code in the brain: An fMRI study of code review and expertise. In: Proc. of the 39th IEEE/ACM Int’l Conf. on Software Engineering (ICSE). IEEE, 2017. 175−186. [doi: 10.1109/ICSE. 2017.24]

[6] Jones C. Measuring defect potentials and defect removal efficiency. CrossTalk the Journal of Defense Software Engineering, 2008, 21(6): 11−13.

[7] Bavota G, Russo B. Four eyes are better than two: On the impact of code reviews on software quality. In: Proc. of the 2015 IEEE Int’l Conf. on Software Maintenance and Evolution (ICSME). IEEE, 2015. 81−90. [doi: 10.1109/ICSM.2015.7332454]

[8] Shull F, Seaman C. Inspecting the history of inspections: An example of evidence-based technology diffusion. IEEE Software, 2008, 25(1): 88−90. [doi: 10.1109/MS.2008.7]

[9] Bacchelli A, Bird C. Expectations, outcomes, and challenges of modern code review. In: Proc. of the 35th Int’l Conf. on Software Engineering (ICSE). IEEE, 2013. 712−721. [doi: 10.1109/ICSE.2013.6606617]

[10] Rigby PC, German DM. A preliminary examination of code review processes in open source projects. Technical Report, DCS-305-IR, University of Victoria, 2006.

[11] Weißgerber P, Neu D, Diehl S. Small patches get in! In: Proc. of the 2008 Int’l Working Conf. on Mining Software Repositories. ACM, 2008. 67−76. [doi: 10.1145/1370750.1370767]

[12] Fan YR, Xia X, Lo D, Li SP. Early prediction of merged code changes to prioritize reviewing tasks. Empirical Software Engineering, 2018, 23(6): 3346−3393. [doi: 10.1007/s10664-018-9602-0]

[13] Gousios G, Zaidman A, Storey MA, Deursen AV. Work practices and challenges in pull-based development: The integrator’s perspective. In: Proc. of the 37th IEEE/ACM IEEE Int’l Conf. on Software Engineering. IEEE, 2015. 358−368. [doi: 10.1109/ICSE. 2015.55]

[14] Steinmacher I, Pinto G, Wiese IS, Gerosa MA. Almost there: A study on quasi-contributors in open-source software projects. In: Proc. of the 40th IEEE/ACM Int’l Conf. on Software Engineering (ICSE). IEEE, 2018. 256−266. [doi: 10.1145/3180155. 3180208]

[15] Wang QY, Xia X, Lo D, Li SP. Why is my code change abandoned? Information and Software Technology, 2019, 110: 108−120. [doi: 10.1016/j.infsof.2019.02.007]

[16] Shihab E, Ihara A, Kamei Y, Ibrahim WM, Ohira M, Adams B, Hassan AE, Matsumoto K. Predicting re-opened bugs: A case study on the eclipse project. In: Proc. of the 17th Working Conf. on Reverse Engineering. IEEE, 2010. 249−258. [doi: 10.1109/WCRE. 2010.36]

[17] 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]

[18] Jiang J, Mohamed A, Zhang L. What are the characteristics of reopened pull requests? A case study on open source projects in GitHub. IEEE Access, 2019, 7: 102751−102761. [doi: 10.1109/ACCESS.2019.2928566]

[19] Rigby PC, Bird C. Convergent contemporary software peer review practices. In: Proc. of the 9th Joint Meeting on Foundations of Software Engineering. ACM, 2013. 202−212. [doi: 10.1145/2491411.2491444]

[20] McIntosh S, Kamei Y, Adams B, Hassan AE. The impact of code review coverage and code review participation on software quality: A case study of the QT, VTK, and ITK projects. In: Proc. of the 11th Working Conf. on Mining Software Repositories. ACM, 2014. 192−201. [doi: 10.1145/2597073.2597076]

[21] Tao YD, Han DG, Kim S. Writing acceptable patches: An empirical study of open source project patches. In: Proc. of the 2014 IEEE Int'l Conf. on Software Maintenance and Evolution. IEEE, 2014. 271−280. [doi: 10.1109/ICSME.2014.49]

[22] Tsay J, Dabbish L, Herbsleb J. Let's talk about it: Evaluating contributions through discussion in GitHub. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. ACM, 2014. 144−154. [doi: 10.1145/2635868.2635882]

[23] Morales R, McIntosh S, Khomh F. Do code review practices impact design quality? A case study of the QT, VTK, and ITK projects. In: Proc. of the 22nd IEEE Int'l Conf. on Software Analysis, Evolution, and Reengineering (SANER). IEEE, 2015. 171−180. [doi: 10.1109/SANER.2015.7081827]

[24] Braun V, Clarke V. Using thematic analysis in psychology. Qualitative Research in Psychology, 2006, 3(2): 77−101. [doi: 10.1191/1478088706qp063oa]

[25] Rigby PC, German DM, Storey MA. Open source software peer review practices. In: Proc. of the 30th ACM/IEEE Int'l Conf. on Software Engineering. IEEE, 2008. 541−550. [doi: 10.1145/1368088.1368162]

[26] Song DH, Zhong H, Jia L. The symptom, cause and repair of workaround. In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). IEEE, 2020. 1264−1266. [doi: 10.1145/3324884.3418910]

[27] Thongtanunam P, Tantithamthavorn C, Kula RG, Yoshida N, Iida H, Matsumoto K. Who should review my code? A file location-based code-reviewer recommendation approach for modern code review. In: Proc. of the 22nd IEEE Int'l Conf. on Software Analysis, Evolution, and Reengineering (SANER). IEEE, 2015. 141−150. [doi: 10.1109/SANER.2015.7081824]

[28] Baysal O, Kononenko O, Holmes R, Godfrey MW. Investigating technical and non-technical factors influencing modern code review. Empirical Software Engineering, 2016, 21(3): 932−959. [doi: 10.1007/s10664-015-9366-8]

[29] Jeong G, Kim S, Zimmermann T. Improving code review by predicting reviewers and acceptance of patches. Research on Software, 2009. http://rosaec.snu.ac.kr/publish/2009/techmemo/ROSAEC-2009-006.pdf

[30] Baysal O, Kononenko O, Holmes R, Godfrey MW. The secret life of patches: A Firefox case study. In: Proc. of the 19th Working Conf. on Reverse Engineering. IEEE, 2012. 447−455. [doi: 10.1109/WCRE.2012.54]

[31] Pan JK, Mao XG. An empirical study on interaction factors influencing bug reopenings. In: Proc. of the 21st Asia-Pacific Software Engineering Conf. IEEE, 2014. 39−42. [doi: 10.1109/APSEC.2014.90]

[32] Mohamed A, Zhang L, Jiang J, Ktob A. Predicting which pull requests will get reopened in GitHub. In: Proc. of the 25th Asia-Pacific Software Engineering Conf. (APSEC). IEEE, 2018. 375−385. [doi: 10.1109/APSEC.2018.00052]

王青叶(1989－), 女, 博士, CCF 专业会员, 主要研究领域为经验软件工程, 软件仓库挖掘.

李善平(1963－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为分布式计算, 软件工程, Linux 内核.

万志远(1984－), 女, 博士, 副教授, CCF 会员, CCF 专业会员, 主要研究领域为软件工程.

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