Mozilla Firefox 中群组与个人评审请求：速度与质量之间的权衡

Matej Kucera 格罗宁根
大学 荷兰格罗宁根
m
atej.kucera@yahoo.com

Marco Castelluccio
Mozilla 公司 伦敦，英国
m
castelluccio@mozilla.com

Daniel Feitosa 格罗
宁根大学 荷兰格罗宁根
d.feitosa@rug.nl

Ayushi Rastogi 格罗
宁根大学 荷兰格罗宁根
a.rastogi@rug.nl

arXiv:2601.01514v1 [cs.SE] 4 Jan 2026

摘要

将代码变更集成到软件代码库中的速度，也称为代码审查速度，是业界常用的一项指标，有助于提升吞吐量和开发者满意度。尽管以往研究探讨了影响审查速度的因素，但评审分配流程，尤其是“组审查请求”的作用仍不明确。在 Phabricator、GitHub 和 Bitbucket 等平台上，组审查请求会将代码变更分配给一个评审者群组，群组中的任何成员都可以进行审查；这不同于将代码变更分配给特定审查者的个人评审分配。借鉴管理科学领域的共享任务队列，本研究考察了组审查请求与个人审查请求对速度和质量的影响。

我们研究了 Mozilla Firefox 项目中约 66,000 项修订，结合统计建模与焦点小组讨论中从业者的观点。研究发现，团队审查与审查质量提升相关，表现为回归缺陷减少，而与审查速度的关联则微乎其微。其他被认为的益处包括工作分配更加均衡，以及为新审查者提供培训机会。

CCS 概念

• **软件及其工程 → 软件开发协作**；软件版本控制。

关键词

群组审查请求、代码审查速度、代码审查质量、软件工程、Mozilla、Firefox

ACM 参考格式：
Matej Kucera、Marco Castelluccio、Daniel Feitosa 和 Ayushi Rastogi。2026。《Mozilla Firefox 中群组与个人审查请求：速度与质量的权衡》。载于《*2026 IEEE/ACM 第48届国际软件工程会议（ICSE-SEIP '26）*》，*2026*年*4*月*12*日至*18*日，巴西里约热内卢。ACM，纽约，纽约州，美国，11页。
https://doi.org/10.1145/nnnnnnn.nnnnnnn

本作品采用知识共享署名 4.0 国际许可协议授权。*ICSE-SEIP '26*，巴西里约热内卢© 2026 版权所有：所有者/作者。ACM国际标准书号
978-x-xxxx-xxxx-x/YYYY/MM https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 引言

代码审查速度是软件开发行业的一项关键指标 [23, 51], ，也对开发者工作满意度有着重要影响 [14, 23, 28, 33, 52]。在开源软件（OSS）中，保持较高的审查速度尤为关键，因为漫长的延迟可能会打击贡献者的积极性 [32]。此外，长期未处理的补丁可能会遇到沟通障碍、与当前代码状态脱节等问题，进而导致复杂的合并冲突和集成问题 [33]。然而，优先追求速度可能会牺牲代码质量等其他目标 [51], ，进而损害审查流程。监控这些权衡至关重要，以确保在提升审查速度的同时，流程仍然有效。

加快代码审查有许多方法。一种方法是找出影响审查流程的因素，例如变更规模 [29] 和作者经验 [8, 22, 23]。基于这些见解，可以采取减少延迟的解决方案。例如，使用审查者推荐系统来缩短寻找审查者所需的时间 [2],，使用提醒来通知阻塞审查者 [33, 52], ，以及在接受后自动合并 [28]。旨在量化文献中影响速度的因素的解决方案，往往忽视审查者分配流程，而是关注审查流程本身及其结果；这些结果发生在审查者已经被分配之后。

一种审查者分配策略是组审查请求，即将代码变更分配给一个群组，而不是分配给单个审查者进行审查。GitHub、Phabricator、Bitbucket 和 Azure DevOps 等平台均提供此功能 [7, 17, 37, 42],，允许团队中的任何成员完成审查。根据传闻证据，我们观察到，即使此功能不可用或未被使用，团队也可能采取变通方式，例如在群聊中发布待处理的拉取请求，这表明该功能具有一定的相关性。

队列管理研究表明，共享任务队列（如群组审查请求）通过均衡分配工作并避免瓶颈，从而减少延迟，提高吞吐量 [21] 。然而，在实践中，实证研究表明，个人队列（如个人审查请求）通常表现更好，这归因于更强的责任感，以及较少出现社会惰化现象 [16, 53–55], ，即没有人主动承担责任。

本研究旨在探讨群组或个人评审分配流程如何影响代码审查。我们考察两个维度，即速度和质量，以确定

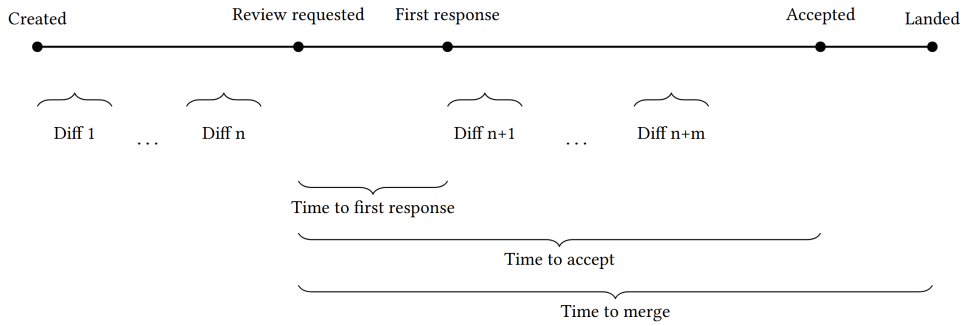

图1：一次修订的生命周期

速度的提升是否会以牺牲质量为代价，或反之亦然。因此，我们的研究问题是：

与个人请求相比，组代码审查如何影响代码审查速度和质量？

我们研究了 Mozilla 基金会的 Firefox 项目，该项目采用个人和团队代码审查 [42]。Firefox 是一款广泛使用的开源浏览器，每天服务数百万用户 [41],；自 2018 [12] 采用 Phabricator 以来，截至 2025年3月，该平台已记录 240,226 次修订。为考察审查者分配流程的影响，我们建立了统计模型，比较个人评审与团队审查对审查速度和质量的影响。我们还通过焦点小组讨论收集经验丰富的开发者的定性见解，以补充这项定量分析。

我们的研究发现，团队审查与回归缺陷发生可能性较低相关，这表明与个人审查请求相比，其软件质量更高。与此同时，软件质量上的这些差异并未伴随审查速度的显著变化，甚至可能完全没有变化；审查速度以接受时间衡量。此外，开发者认为，团队审查作为一种审查者分配流程还有其他优势，包括更均衡的工作量分配，以及培训新审查者的机会。

2 背景与相关工作

本节首先概述 Mozilla 的代码审查流程，随后回顾有关代码审查速度的现有文献，以及管理科学中与任务队列相关的研究。

2.1 Mozilla 的代码审查

Mozilla 使用多种工具来简化其代码评审流程。它通过自定义的问题跟踪平台 Bugzilla [40]跟踪错误。为了解决错误，作者会使用 Phabricator 代码审查平台提交补丁以供评审。

Phabricator 根据提交的差异创建修订 。差异 是作者提交到版本控制系统（VCS）的一组变更。作者可以通过提交新的差异来更新修订。Phabricator 的堆叠差异功能允许将大型代码变更拆分为多个较小的修订，这通常更容易审查。此类堆栈中的修订数量称为堆栈大小 [4]。其他开发者会通过人工或自动化的审查请求来审查修订，并由这些请求指定审查者 [42]。修订获接受后，即可纳入 （合并）到代码库中。

如图1所示，Mozilla 的典型代码审查遵循一系列结构化步骤：首先，作者编写并提交修订，随后指派一名或多名审查者，或请求其进行审查。之后，审查者会提供反馈或批准；获批的修订随后会合并到代码库中。在此过程中，还可能包括一些中间步骤，例如根据反馈更新修订、进行额外的审查迭代，以及运行自动化检查或验证。

Mozilla 代码评审工作流程的另一个常见特点是组审查请求。自2018年6月起，Mozilla 一直使用 Phabricator 作为其主要代码审查平台 [12], ，该平台支持手动和自动分配（群组）审查请求。作者可以将群组指定为某项修订的审查者，也可以通过预定义的“Herald 规则”自动分配群组；这些规则会根据修订的特征触发（例如，修订包含对指定文件路径的更改）。Mozilla 的大多数 Herald 规则都设置为分配团队审查 [43]。

Phabricator 中的代码评审流程会生成大量数据。Mozilla 维护着 bugbug，这是一个面向软件工程任务机器学习项目的开源平台 [10], ，其中包含机器学习模型、工具和数据集。bugbug 代码审查数据集包含原始 Phabricator 日志，包括修订事务和描述审查流程的元数据。

鉴于有群组和个人两种审查者分配流程，了解它们对代码审查流程的影响至关重要。哪种方法更有效？或者在什么条件下，一种方法会优于另一种？由于这两种审查形式在 Mozilla 都很常见，这一环境为通过实证证据探讨该问题提供了宝贵的机会。

2.2 代码审查速度

现有文献发现，许多因素都会影响代码审查速度 [2, 3, 5, 8, 22, 28, 29, 31, 33, 46, 50, 52, 56]。这些因素主要归属四类实体：作者、审查者、

修订和项目。表1列出了先前研究考察的因素，并按其所属实体分类。这份内容详尽但并非穷尽的清单，是通过文献检索和分析近期的一篇实证综述编制而成的 [30]。

过去已有多项研究调查了 Mozilla 的代码审查，最近的一项发表于 2016 [23, 24, 38, 46]。在引入 Phabricator 之前，Mozilla 使用 MozReview（ReviewBoard 的一个分支）作为代码审查平台。MozReview 不支持群组审查请求 [39]。自 2018 [12] 引入 Phabricator 以来，该流程发生了主要变化。据我们所知，自 2018年流程变更以来，尚未有研究对 Mozilla 的代码评审流程进行调查。

2.3 任务队列管理

任务队列管理是管理任务队列的过程，旨在确保任务高效、公平并按照正确的顺序完成 [21]。任务队列管理是一个复杂的问题，受到社会和情境因素的影响，并有两种配置：共享队列和个人队列 [55]。在共享队列中，任务会分配给一个工作人员池，池中的任何成员都可以领取任务。在个人队列中，任务会分配给单个个人。在代码审查的语境下，群组审查请求等同于共享队列，而个人请求等同于个人队列。

关于任务队列管理的文献提出了两种互补的观点，探讨共享任务队列如何影响负载均衡和效率。一种观点认为，在其他条件相同的情况下，共享任务队列能够降低劳动力成本，而不牺牲延迟 [21]。其基本思路是，潜在工作者越多，瓶颈和过重的个人工作负荷就越能得到缓解。然而，这种观点被认为只是理论上的，因为它没有考虑社会、环境及其他因素的影响。例如，共享任务队列可能会产生竞争效应：如果有报酬，工作者会与队列中的其他人竞争，以获得金钱、非物质或其他形式的报酬 [55]。不过，共享队列也可能因社会惰化而降低效率，即个人预料同事会弥补延迟，因而有意减少付出。

相比之下，个人队列与更强的任务所有权相关，同时也更容易出现负载分布不均和任务放弃。研究表明，当任务分配给工作者本人时，更强的所有权意识能够激励他们 [54]。然而，如果任务按照某种模式分配（例如依据资历和可见性），队列长度失衡就可能造成瓶颈并导致延迟。

Mozilla 的代码审查采用群组审查请求和个人审查者分配两种方式；然而，这些方式对审查速度和质量的影响尚不明确。借鉴任务队列管理的见解，我们发现分配策略可能通过多种方式影响审查速度。

具体到 Mozilla，我们还希望了解采用每种方式背后的原因。例如，当不确定谁最适合评审某项变更时，共享队列可能会很有用，但人们尚不清楚这类队列在代码审查这类高度依赖人工协作的流程中如何运作。

表1：已知会影响代码审查速度的因素

实体	指标
拉取请求	**代码行数（LOC）** [22, 29, 33, 56]
	受影响的模块 [22, 24]
	区域热度 [22, 56]
	文件类型 [33]
	审查者人数 [22, 52]
	死代码删除 [3]
	文件数量 [22, 24, 33, 52]
	提交次数 [30, 56]
	已分配审阅者 [52]
	测试 [29, 56]
	日期 [22, 33]
	创建于星期五 [29, 56]
	描述长度 [29, 56]
	CI 结果 [29, 56]
	评论 [22, 24, 29, 56]
	优先级 [24]
	严重程度 [24]
	任务类型 [22, 33]
项目	CI 延迟 [29, 56]
	项目年限 [29, 56]
	团队规模 [56]
	未完成修订数量 [24, 29, 56]
	平均补丁延迟 [33]
作者	与审查者的关系 [52]
	作者所属身份 [23, 29, 56]
	作者资历 [33]
	变更经验 [22, 23, 29]
	审查经验 [8]
	作者成功率 [56]
审查者	审阅者工作量
	审阅经验 [24]
	阻塞式评审 [33]

[1]粗体**加粗**指标是混杂因素

为弥补这些不足，我们的研究考察了审查分配实践如何影响代码审查的质量和速度。

3 研究设计

为了全面了解群组审查请求与个人审查请求相比，对代码审查速度和质量有何影响，我们设计了一项混合方法研究。首先，我们挖掘软件档案，建立统计模型，并控制混杂因素的影响，从速度和质量两个方面比较这两种评审分配流程。随后，我们在焦点小组中向一群经过筛选的资深开发者展示这些发现，以了解他们对相关观察结果的经验和看法。结合这两种方法，我们得以全面呈现选择不同评审分配流程时，在速度和质量方面所涉及的权衡。

以下小节将介绍我们的分析设计选择，首先说明用于回答

研究问题的数据处理方法。接下来，我们将详细说明用于衡量审查速度、质量和混杂因素的指标选择。最后，我们将介绍统计建模和焦点小组会议的详细信息，这两者共同用于回答我们的研究问题。本研究使用的所有源代码和数据集均可通过 Figshare 上的复现包获取 [26]。

3.1 数据集

为了分析 Mozilla 的典型代码评审工作流程，我们制定了一些纳入和排除标准。有关典型工作流程，请参阅第 2 节和图1。

3.1.1 纳入和排除标准。 我们研究了 Mozilla 最大的项目——Firefox 浏览器，该项目拥有丰富的开发历史。Firefox 项目的缺陷跟踪器 Bugzilla 被划分为 43 个团队，每个团队负责一个或多个产品，而每个产品又包含一个或多个组件。这些团队（群组）要么按功能组织（例如反跟踪、Cookie），要么按项目模块和组件组织（例如 Android、构建）。

我们从 Firefox 项目开始，收集了 240，226 个修订，这些修订随后通过 bugbug 提供的其他信息得到进一步丰富（见第 2 节）。我们选取了 Firefox 项目中所有在 2018 [12] 年 6 月正式采用 Phabricator 后创建的已发布修订。截止日期为 2025 年 3 月 8 日。我们排除了正式采用前近一年的数据，因为这些数据可能不符合典型的代码评审工作流程。

接下来，我们排除了已放弃或仍在进行中的修订，因为它们尚未完成代码评审流程。由于审查请求类型与修订状态之间不存在相关性，因此这不应影响分析。之后的排除旨在通过移除异常和边缘案例来提高数据质量。因此，与先前研究一样，[29], 我们排除了跳过审查流程的修订。这包括由作者审查的修订或未经审查的修订。我们还移除了生命周期时间戳不一致的修订（例如，某项修订被标记为已合并，但其创建时间却晚于合并时间）；这类情况是作者未遵循标准代码评审规范、手动发布修订所致。

此外，我们排除了在 Bugzilla 中没有关联缺陷报告的修订。根据一位参与研究设计的 Mozilla 资深研究人员的说法，关联缺陷报告表明任务具有相关性，也可能意味着其遵循了典型的代码评审流程，反之亦然。最后，我们还排除了缺少混杂属性（详情后文介绍）的修订，例如“问题类型”和“差异”。最终数据集包含 175，330 项修订，占 240，226 项修订的 70%，即全部数据的 70%。

3.1.2 伪装成个人请求的组请求。在进行合理性检查时，我们将 Phabricator 中随机抽取的代码审查数据与 bugbug 数据集中的原始、详尽对应数据以及 Phabricator 用户界面中的信息进行了人工比对，并发现了一处异常。在某些情况下，我们发现审查者在接受组审查请求时将其删除了。例如，请参阅修订 189553[1]。之后，我们在焦点小组中确定了这一行为的原因；然而，就数据分析而言，这些请求看起来像个人请求（因为组审查请求已被删除），占所有修订的 3%。

为提高数据质量，我们在根据 Phabricator 推断出的评审分配流程基础上，补充了来自 bugbug 数据集的其他信息，从而能够识别出这些已删除的审查请求。由此得到的数据更加准确，但规模也略小，因为 bugbug 仅包含过去两年的数据。我们的最终数据集由 66,318 个修订组成。为进行比较，我们还在包含 173,701 个修订的较大数据集上进行了实验，并观察到类似的结果（详情请参见复现包 [26]）。

3.2 指标

为了研究群组审查请求与个人审查请求对审查速度和质量的影响，我们从文献中选取了与之最接近、且可以使用 Firefox 项目现有代码审查数据进行测量的代理指标。同样，我们还从文献中确定了可测量的混杂因素，以及项目背景所特有的混杂因素。

3.2.1 代码审查速度。 在代码审查生命周期中，有三个关键事件：首次响应、接受和合并，它们反映了审查的速度。同样，也有三个常用指标用于衡量代码审查速度。首次响应时间衡量审查者首次响应之前所经历的时间 [28]。接受时间是某个修订被（所有）审查者接受之前所经历的时间 [28]。最后，合并时间是指该修订被合并到代码库之前所经历的时间 [28]。这三个指标与其在修订生命周期中的时间跨度之间的关系如图1所示。

我们选择接受时间作为审查速度的指标，因为它最能反映审查周期中可能受到群组审查请求影响的部分。相比之下，合并时间还包含与合并相关的其他延迟原因，而这些原因不受审查者影响。首次响应时间也可以作为另一个合理的候选指标，因为它反映了个体分配与群组分配后发生的首次活动，但这只是整个过程的一部分。我们没有采用这一指标，因为它无法展现全貌，而且接受时间与 Mozilla 更为相关。

我们将接受时间定义为从创建首次评审请求到接受修订的时间 [6]。由于 Phabricator 允许作者发布修订草稿并在请求评审前进行修改（见图1），我们调整了接受时间的定义，将起点设为创建首次评审请求，而非创建修订。在我们的数据中，所有修订中有 5% 的创建时间与首次评审请求时间相差 8 小时或更久。

3.2.2 群组审查请求。如前所述，审查请求分为两种：群组请求和个人请求。不过，单个修订可能关联多个异步创建的审查请求。我们根据修订最初关联的审查请求，将修订分为仅小组、混合和仅个人三类。这三类修订分别占 33.2%、4.2% 和 62.7%。

3.2.3 代码审查质量。 评估代码审查的质量很困难，因为评审旨在实现多个且往往难以量化的

[1]https://phabricator.services.mozilla.com/D189553

表2：上下文特定指标

实体	指标
修订	**首次审查请求类型**
	堆栈大小 [4]
	在周末创建
	导致回归
	修复回归
	Team
	问题评论数
作者	**作者所属身份**（员工、贡献者）

目标。在 Mozilla，错误数量被认为是一项重要指标；这些错误绕过了审查流程，因而会直接影响终端用户体验。回归缺陷被定义为由更改引入、导致原本正常运行的功能受到破坏的误差。Mozilla 会系统地监控已知的回归缺陷 [45]。

在我们的数据集中，我们根据每项修订在合并后是否被确认导致回归缺陷，对其进行了标注。我们采用这一指标作为代码审查质量的代理指标。在筛选后的数据集中，25.6% 的修订被确认导致了回归缺陷。但需要注意的是，近期修订造成的回归缺陷可能尚未被发现，这限制了该指标的完整性。这对个人审查请求和群组审查请求的影响应当相同；不过，这可能是一个偏差来源。

3.2.4 混杂因素。 我们控制已知会影响代码审查的因素，以及 Mozilla 特有的因素。参考相关文献（见表1），我们纳入了数据集中所有可测量的因素（以粗体突出显示）。此外，我们还纳入了 Mozilla 特有的因素（见表2），因为研究表明，项目年限和团队规模等背景因素 [57], 会影响审查速度。最后，我们结合 Phabricator 和 Mozilla文档 [4] 以及一位 Mozilla高级工程经理的经验，收集了 Phabricator 和 Bugzilla 生态系统的独特功能。

3.3 统计建模

为了构建具有代表性的统计模型，我们尝试了多种预处理步骤。我们注意到，接受时间和变更代码行数的分布高度偏斜；因此，我们针对这两个字段尝试了三个阈值，分别为 95%、99% 和不设阈值 [15] 。此外，2020年，Bugzilla 将严重性度量方法从描述性（例如“严重”“主要”）改为编码形式（例如“S1”“S4”）。因此，我们仅使用有效严重程度条目构建模型，或移除严重程度字段。最后，我们通过纳入或排除已删除的审查请求（见第3.1.2小节），对此进行了尝试。由此得到 36 个数据集（$3 \cdot 3 \cdot 2 \cdot 2$），涵盖 $175,459$ 至 $23,186$ 个修订（详情见复现包 [26]）。

接下来，我们构建多元线性回归模型，研究审查请求类型对审查速度的影响。选择线性回归模型，使我们能够在控制以下因素的影响的同时，找出审查请求类型与审查速度之间的关系。如果 p 值 < 0.05，我们就认为这种关系在统计上显著。对于新增和删除的代码行数等偏态功能，我们进行了对数转换，以便标准化并稳定方差、降低异方差性 [27]。

我们还使用方差膨胀因子（VIF） [35]评估了功能的多重共线性。所有 VIF 均低于 3，表明不存在多重共线性问题。最后，我们对所有分类变量进行了独热编码，将其表示为由 0 和 1 组成的二进制向量，这是线性回归所要求的。

我们使用调整后的 R^2 [27] 评估模型拟合度，并使用 Grömping 的方法 [18] 衡量每个功能的相对重要性。该方法通过计算移除每个变量时 R^2的下降幅度来衡量效应量，从而确定模型中由每个变量解释的百分比。

同样，为了研究审查类型对审查质量的影响，我们构建了逻辑回归模型 [48]。逻辑回归在因变量为布尔因变量时，具有与线性回归相同的优点。我们使用 McFadden 的伪 R^2 [36]评估模型拟合度。由于尚无计算逻辑回归模型中效应量的既定方法，因此我们不计算效应量。

3.4 焦点小组

我们与 Mozilla 开发者进行了一场成果介绍和焦点小组讨论，以补充定量分析。仅凭统计结果可能会忽略情境因素，尤其是在工业环境中，组织流程、文化实践和特定工具的使用行为都会影响结果。与从业者交流使我们能够验证研究发现，使结论符合实际工作实践，并纳入基于他们经验和专业知识的解读。参与者由一位高级工程经理通过内部渠道直接邀请，也通过群发邀请的方式招募。受邀者是根据其审查者专业知识和/或对代码评审流程的兴趣选出的。参与完全自愿，会议录音也已征得参与者的知情同意。

会议包括两个环节：首先用 15 分钟介绍研究结果，随后进行焦点小组讨论 [25]。介绍内容涵盖统计分析的结果，包括关于代码审查速度的线性回归模型、关于审查质量的逻辑回归模型，以及用于检验其他解释的相关性检验。为扩大参与范围，我们还发放了一份简短调查问卷，供无法参加讨论的人填写。我们对会议资料（录音和文字记录）以及调查回复进行了主题分析 [9]。

4 结果

我们的模型表现差异很大，调整后的 R^2 值介于 0.09 至 0.33 之间。在线性回归中，这属于弱至中等拟合度（R^2 值介于 0.25 至 0.50 [19] 之间）。我们将在第 6 节讨论这一点。表现最佳的模型（调整后的 R^2 为 0.33）排除了严重程度 特征，排除了含有已删除评审的修订，并排除了接受时间高于第 99 百分位数的修订。如前所述，该数据集包含 66,318 个修订。表现第二好的模型纳入了已删除的评审，因此数据集更大，包含 173,701 个修订，调整后的 R^2 值为 0.31。由于

表 3：用于推断代码审查速度的线性回归模型（系数单位为秒）

变量	系数（误差）	相对重要性
截距	314329 (19178)***	N/A
差异数量_	267091 (2955)***	46.72%
非作者评论数量_ _	65907 (3453)***	17.11%
作者评论数_ _	40203 (3387)***	15.44%
log(代码行数_源代码_代码_新增) + 0.5	57230 (3587)***	4.73%
堆栈大小_	75001 (2616)***	4.30%
审稿人数_ _	34880 (2413)***	4.18%
导致_回归	29509 (5406)***	1.47%
问题_类型_增强	30616 (6948)***	1.06%
作者成功率_ _	-7366 (2446)**	0.89%
修复_回归	-58744 (7444)***	0.85%
之前_接受_修订	-15240 (2511)***	0.84%
作者_过去_审阅_比例	-23449 (2807)***	0.58%
log(loc_源代码_代码_已删除) + 0.5	-16691 (2994)***	0.58%
是员工_	-23428 (5774)***	0.41%
创建于_星期五	52821 (5842)***	0.23%
包含测试_	-22026 (5525)***	0.20%
删除_仅	36577 (13337)**	0.18%
问题_类型_任务	-13278 (5752)*	0.08%
首次_评审_为_群组_混合	-4973 (11447)	0.06%
首次_评审_为_群组_仅小组	**-12533 (5345)***	**0.04%**
修改的文件_	-5720 (2223)*	0.03%
周末创建_	39228 (9711)***	0.02%
作者_过去_更改_比例	2907 (2917)	N/A
是重新格式化_	-475 (51526)	N/A
问题评论数_ _	673 (2569)	N/A

***p < 0.001, **p < 0.01, *p < 0.05

基于第 3 节中关于已删除评审的原因，后续所有分析均以表现最佳的模型为依据。

4.1 对代码评审速度的影响

表 3 展示了一个多元线性回归模型，用于解释接受时间（以秒为单位）与评审分配流程之间的关系。各项功能按相对重要性（效应量）排序。它们的关系受一系列根据文献推断出的混杂因素影响，除此之外则因项目而异。我们通过将独热编码分类变量以大写形式呈现，来对其进行视觉区分。例如，“增强” 问题_类型 称为“问题_类型_增强”。为简洁起见，尽管团队名称在 33 种情况中的 17 种情况下具有显著性，我们仍未在表 3 中列出团队名称。完整模型见复现包 [26]。

该模型的调整后 R^2 (0.33) 与文献中的其他类似模型相当 [56]。通常，拟合优度相对较低是意料之中的，因为代码审查是一个复杂的社会技术过程，而且许多因素（例如私人沟通）无法得到充分考虑。最后，表 3 中的「相对重要性」列报告了效应量，但有一些例外。由于算法运行时间复杂度为 $O(2^n)$，我们不报告作者_过往_变更_比率、是_重新格式化、问题_评论_数量以及团队_名称 的相对重要性。请注意，我们不报告统计上不显著的特征的相对重要性（见表 3）。

结果表明，模型中 79% 的部分可由差异数量以及作者评论和非作者评论的数量解释。以新增代码行数和堆栈大小衡量的修订规模，可解释模型的另外 9%。与以往研究类似 [33], ，审阅者人数很重要，相对重要性为 4%。包括评审分配流程在内的所有其他因素，相对重要性均低于 1。5%。关于评审分配流程，我们观察到，群组审查请求（以“first_评审_是_群组_仅小组”衡量）与接受时间略微缩短相关（效应量：0。04%），相比之下，个人请求则不是如此。

群组和个人审查请求与相似的代码评审速度相关。

4.2 对代码审查质量的影响

用于推断代码审查质量的模型，其 McFadden 伪 R^2 为 0.10 [36]。该模型的拟合度较弱至一般 [20, 36]，原因也相同：代码审查是一个复杂的社会技术问题。表4列出了模型系数和优势比，用以表示相对相关性。优势比为 2 意味着单位

表4：以逻辑回归模型推断回归作为代码质量指标

变量	系数（误差）	优势比
截距	-1.37 (0.08)***	0.25
堆栈大小_	0.50 (0.01)***	1.65
问题_类型_增强	0.40 (0.03)***	1.49
首次_评审_类型_仅小组	**-0.36 (0.02)*** **	**0.70**
仅删除_	-0.26 (0.07)***	0.77
周末创建_	-0.21 (0.04)***	0.81
log(代码行数_源代码_代码_新增) + 0.5	0.21 (0.02)***	1.23
是否为员工_	0.19 (0.03)***	1.21
差异数_	0.18 (0.01)***	1.20
修复_回归问题	-0.17 (0.03)***	0.84
创建于_星期五	-0.12 (0.03)***	0.89
是重新格式化_	-0.11 (0.22)	0.90
log(代码行数_源代码_代码_已删除) + 0.5	0.09 (0.01)***	1.10
作者_过往_评审_比例	0.07 (0.01)***	1.07
接受_所需_时间_秒	0.06 (0.01)***	1.06
作者_过去_变更_比例	-0.06 (0.01)***	0.94
之前_接受的_修订	0.06 (0.01)***	1.06
首次_评审_类型_混合	-0.06 (0.05)	0.94
问题_类型_任务	0.05 (0.03)	1.05
作者评论数_ _	0.04 (0.01)**	1.04
包含测试_	-0.04 (0.02)	0.96
修改的文件_	-0.01 (0.01)	0.99
审查者人数_ _	0.01 (0.01)	1.01
作者成功率_ _	0.00 (0.01)	1.00
非作者评论数量_ _	0.00 (0.01)	1.00

***p < 0.001，**p < 0.01

预测变量的增幅会使结果发生的优势翻倍。在表4中，功能按系数绝对值从高到低排列。为便于阅读，我们也从表4中移除了团队名称（详见复现包 [26]）。

该模型显示，删除_仅、首次审查类型、问题类型、堆栈大小和新增代码行数是与代码审查质量关系最密切的因素，代码审查质量以回归缺陷来衡量。其中一些关系符合预期。例如，增强功能通常比缺陷导致更多回归缺陷，而仅删除的修订会导致更少的回归缺陷。同样，新增代码行数和堆栈大小的系数为正，这表明大型代码变更更有可能导致回归缺陷。

最重要的是，我们发现，与“仅个人”审查请求相比，“仅小组”审查请求与较低的回归缺陷发生可能性相关，且具有统计显著性（ 0.001 0.7 ），优势比为 。这一见解假设开发者会针对相同类型的代码变更选择小组和个人审查请求，从而排除了审查分配流程选择中任何潜在的偏差。我们将在下文进一步探讨这一主张，寻找对数据中这一观察结果的合理替代解释，并在稍后结合开发者的经历进行分析。

4.2.1 其他解释。为进一步探究团队审查与回归缺陷之间的关系，我们试图找出可能影响这种关系的其他因素。我们怀疑，原因可能在于团队审查的运作方式有所不同（例如，审查者参与度提高），以及个体与情境因素（例如，经验丰富的开发者往往更倾向于团队审查）。下面我们探讨四种此类关系，其中两种是表1和表2中的混杂因素，它们与团队请求和个人请求相关，且效应量（相对）较大。

我们的数据并不服从正态分布；因此，我们选择非参数测试。对于连续变量与分类变量对，我们使用曼-惠特尼U检验 [34] 并采用邦费罗尼校正 [1] 来衡量统计显著性。此外，我们使用克利夫δ（CD）来衡量其效应，当数值小于 0.147 时，表明效应可以忽略不计 [11]。对于分类变量对，我们使用卡方检验 [47] 来衡量显著性，并使用克拉默V（CV）来衡量效应；当数值低于 0.10 时，表明没有效应 [13]。

审阅者人数 一种可能性是，群组审查请求会征集不止一名审查者。根据 Linus定律“只要有足够多双眼睛，所有错误都无关紧要” [49]，这意味着群组审查请求能够获得全面关注，从而减少回归缺陷。然而，我们的结果显示不存在统计差异（p值 = 1），群组审查请求和个人审查请求的平均已完成的审查数量均为 1.16。

尽管审查者人数没有差异，但选择审查者的机制有所不同。对于个人审查请求，审查者由代码作者选定。相比之下，在群组审查请求中，群组的一名或多名开发者会决定由谁来审查代码。

选择专家审查者的机制可以在减少群组审查请求中的回归缺陷方面发挥中介作用。

Herald 规则 我们之前的探索表明，Herald 规则会影响审查请求在个人与群组之间的分配。因此，我们调查了由 Herald 请求的评审与由修订作者请求的评审之间的差异。在我们的数据中，18.6% 的评审是由 Herald 规则请求的。

无论代码作者是否知道该向谁请求评审，都会调用 Herald 规则。我们的分析显示，这些规则明显偏向于群组审查请求（CV = 0.513）。此外，采用 Herald 规则与讨论增加以及问题类型的选择有关。我们观察到，采用 Herald 规则的分配会收到作者更多的评论（CD = 0.091）和审查者更多的评论（CD = 0.079），但与审查者请求机制（也称为首次审查类型）的选择之间的关系可以忽略不计。因此，群组请求中的更多讨论并不能解释这一观察结果。

同样，我们观察到，与缺陷相比，Herald 规则更常用于为问题类型的任务分配评审（CV = 0.084）。然而，首次审查类型与问题类型之间的效应量可以忽略不计（CV = 0.052），这表明问题类型无法解释为何群组审查请求的回归缺陷更少。

接下来，我们考察可能影响首次审查类型选择的代码作者特征。

作者所属身份 Mozilla文档建议首次贡献者向导师请求个人审查 [44]。这表明，个人审查请求与群组审查请求可能存在不同的提出者模式。进一步探究后，我们发现，员工更有可能请求团队审查（CV = 0.076）。此外，任务选择也有所不同：与员工相比，贡献者更有可能从事增强功能（CV = 0.092）。此外，Mozilla 的经验丰富的开发者指出，贡献者往往更喜欢开发令人兴奋的功能，而不是处理缺陷。

因此，我们考察作者所属身份（员工或非员工）与代码回归之间的关系，并假设作者所属身份在评审分配方式与回归之间起中介作用。然而，我们没有发现两者之间存在任何关系（效应量可以忽略不计：CV = 0.021），这表明群组审查请求中回归缺陷较少并非由作者所属身份所致。

作者专业能力 我们认为，所属组织并不能解释这一关系，因为无论所属组织如何，作者专业能力可能更为重要。正如所属组织与专业知识之间关系的效应量极小（CD= −0.019）所显示的那样，非员工既可能是专家，也可能不是专家，因此我们转而考察作者专业能力。

我们发现，以先前的修订成功率（已接受的修订所占百分比）衡量的专家，往往会提出更多个人请求（CD= 0.102）。然而，我们发现专业能力与速度或质量均无关。进一步检查还显示，群组与个人的已接受修订和已放弃的修订之比并无差异（p值 = 0.191）。

这种关系似乎不受开发者特征（所属组织和专业知识）、代码特征（任务与缺陷）的影响，也不受审阅者分配方法（自动与手动）或审查流程因素（审查者人数和讨论时长）的影响。这可能与审阅者选择有关，因为在组审查请求的情况下，自行选择的审阅者似乎比作者选择的审阅者更有效；不过，对这一主题的进一步研究超出了本研究的范围。另一种可能的解释是，很少采用团队审查的某些团队可能与更多的回归缺陷相关，反之亦然。

与个人请求相比，群组审查请求与更少的回归相关联，因此代码质量更高。

4.3 焦点小组与调查问卷

我们于2025年5月23日开展了一次焦点小组会议，通过开发者经验来补充我们从定量数据分析中获得的理解。焦点小组会议之前进行了时长15分钟的演示，总结相关工作；随后，我们通过现场提问以及调查问卷向无法停留超过15分钟的参与者提出了针对性问题。调查问卷中的问题可在复现包 [26]中找到。我们确定并邀请了九名Mozilla资深开发者（具有 10+ 年的经验），他们在Firefox的不同团队中工作，遍布全球。演示结束时，五名成员留下来参加了额外45分钟的讨论，其余四人通过调查问卷回答了问题。

我们向开发者提出的问题围绕五个关键主题展开。在引导和热身环节，我们征集了开发者关于组代码审查用途的经验，以及从数据中删除群组审查请求的原因。在前一种情况下，关于组代码审查用途的问题旨在收集此前尚不了解的原因；而后一问题则意在了解数据中观察到的差异究竟是有意为之，还是一种误差。之后的所有问题都围绕开发者对影响代码审查速度和代码质量的因素的经验与理解展开，并比较群组审查请求与个人审查请求之间的差异。

4.3.1 组审查请求的用途。 此前，我们观察到员工提出更多团队审查，而新人则会得到导师建议，向其请求评审。开发者补充了我们的理解：当作者不确定应向谁请求评审时，也会使用群组审查请求。例如，新入职的团队成员（同时也是员工）在不确定谁应当审查某项修订时，可以请求群组审查，因为了解哪些审查者最适合代码库的不同部分需要时间。

此外，几位开发者指出，出于负载均衡的考虑，他们更倾向于采用群组审查请求。他们用“因为这样可以分散工作量”和“为了实现负载均衡”等说法来表达群组审查请求的重要性。开发者认为，在大多数情况下，群组审查请求能够分散工作量；如果不采用群组审查请求，审查就会分配给同一批经验丰富的开发者，使他们负担过重，同时也会阻碍新团队成员积累知识。作为补充，他们指出，这种模式并不会妨碍专家审查严重变更，因为阻止群组评审（Herald 请求的大多数评审都会被阻止 [43]）通常被更多

人视为……此外，他们还指出，有时只有一位审查者具备作出回应的资格，这限制了组审查请求所能带来的益处。

4.3.2 删除组审查请求的原因。在进行数据检查时，我们发现大量组审查请求最终被删除（详情见第3节）。各团队的删除比例从13.9%到0.6%不等，占整个数据集的3%。然而，我们为提高数据准确性所做的尝试缩小了数据集的规模。我们询问开发者，删除是否属于误操作，或者是否另有原因。开发者回应称，关于组审查请求的做法因团队和时间而异。例如，一些团队会在某位成员选中组审查请求进行审查后将其删除，以避免其他成员在这上面浪费时间。另一些团队则不会进行任何修改，以确保其他团队成员也能看到该请求。

4.3.3 影响代码审查速度的因素。我们征询了开发者对影响审查速度的因素以及可采取措施的看法。总体而言，开发者对影响代码审查速度的因素的看法，与现有数据和既往研究相符。参与者认为，变更代码行数、讨论长度、作者经验、审阅者工作量以及作者与审查者的关系都会影响审查速度。具体从审查者的角度来看，开发者认为，清晰的摘要、较小的变更、较少的修改文件、对代码的熟悉程度以及对作者的信任，都能让审查更轻松、更快速。他们还指出，与因问题或更新遗漏而停滞在某一状态的评审相比，在需要作者和审查者交替回应的评审中，进展通常更快。

4.3.4 群组审查请求与更少的回归缺陷。最后，为探究群组审查请求与代码质量提升（以更少的回归缺陷衡量）之间关系的潜在解释，我们考察了作者所属身份和 Herald 规则的使用等因素如何影响这种关系。需要注意的是，焦点小组的参与者全部是正式员工。因此，他们的观点只能呈现部分情况。尽管如此，鉴于他们担任非正式员工贡献者的导师和审查者，可能对相关情况有所了解。

作者所属身份参与者认同文档中的建议，认为新人只应向自己的导师请求评审。他们还分享了一些可能影响观察到的回归缺陷数量的细节。为了支持表 4 中的观察结果（即正式员工提交的补丁包含更多回归缺陷），他们补充说，正式员工通常负责相对复杂的任务，而且截止期限紧迫。此外，与正式员工提交的补丁相比，开发者会更严格地评审非正式员工贡献者提交的补丁，以确保达到更高的质量标准。一位开发者补充道："你不知道贡献者会不会继续留下来"，因此，彻底评审的目的是防止未来出现回归缺陷。另一条调查回复再次印证了先前的观察，称正式员工"会很快修复回归缺陷"，而非正式员工则未必如此。还有一种观点认为，非正式员工贡献者或许比正式员工更注重避免回归缺陷，以便被视为优秀的贡献者。

Herald 规则最后，我们询问了参与者使用 Herald 规则的经验，因为这些规则与群组审查请求密切相关。所有参与者都表示，他们的团队都在使用 Herald。作为审查者，他们通常不会注意到评审请求是由 Herald 自动发起还是手动发起的，也认为两者在流程上没有区别。几位开发者解释说，他们的团队会配置 Herald，以强制执行阻塞式组审查请求。另一些开发者补充说，当他们不确定某些代码变更应由谁评审时，会依赖 Herald，并指出："Herald 规则有助于新人获得评审。"

从业者指出，尽管各团队的审查分配实践有所不同，群组审查请求有助于负载均衡、审查者技能发展以及更广泛的可见性。

此外，开发者通常认同用于解释代码审查速度和质量的模型，并经常针对意外的关系提供更多细节和解释（例如，正式员工的代码变更会导致更多回归缺陷）。

5 讨论

5.1 结果解读

在这项研究开始时，管理科学领域出现了两种相互对立的理论：一种认为，群组审查请求可以通过避免延迟和瓶颈来提高审查速度；另一种则认为，群组审查请求会因责任感降低和社会惰化而降低审查速度。我们的研究表明，群组审查请求的审查速度与个人请求相当。这与我们的预期相矛盾，但也可能指向多种合理的解释。或许两种力量同时发挥作用，彼此抵消了影响。另一种可能是，个人请求中的严重延迟抵消了群组审查请求中的观察结果，反之亦然。尽管速度保持相当，但我们随后对代码质量以及开发者对群组审查请求目的的看法进行了进一步探索，从而描绘出一幅全面的图景。

我们的研究表明，组审查请求与回归缺陷发生概率显著降低相关，而回归缺陷尤其是 Mozilla 和 Firefox 衡量审查质量的关键指标。此外，我们考察了几个可能起中介作用的因素，包括使用组审查请求的开发者的经验和所属组织、偏好组审查请求的情形，以及组审查请求与个人审查请求在运作方式上的差异。上述因素均无法解释为什么组审查请求导致的回归缺陷更少。

唯一合理的解释是这两种方式在审查者选择上的差异。个人请求假定代码作者知道哪位候选人适合审查，而组审查请求则依靠审查者根据兴趣、可用性和专业知识自我选择。要证实这种关系，还需要开展更多研究。

此外，参与焦点小组的开发者强调，在难以确定审查者时，组审查请求对于识别潜在审查者具有重要性和优先性，同时还能平衡工作量、促进审查者技能发展，并带来更广泛的可见性。

简而言之，群组审查请求的代码审查速度与个人请求相当，但由于减少了回归缺陷，因此与更高的代码质量相关。其他被感知到的益处包括改进评审任务管理，以及培养开展评审所需的技能。

5.2 对业界的建议

我们建议组织考虑采用群组审查请求，同时保留个人请求，尤其是在重点关注高效管理评审和提升质量的情况下。我们认为，我们研究得出的见解也适用于团队采用层级结构的大型类似项目。这种层级结构使得项目可以将补丁分配给某个群组进行审查，而该群组中的开发者很可能具备所需的专业知识并对此感兴趣。

5.3 对学术界的建议

从研究角度来看，仍有许多问题尚未解答。首先，与个人审查请求相比，群组审查请求对其他关键代码评审目标（如实用性、知识转移和代码的长期可维护性）有何不同影响？ [8]。这类研究有助于全面呈现群组审查请求的适用性及其局限。例如，我们的研究未考虑群组审查请求所需的成本；这类请求或许能提高审查质量，但开发者需要额外花时间确定由谁进行审查。

另一种可能是研究评审技能和评审速度随时间的变化。组审查鼓励更多人参与评审，因此评审速度可能会提高。又或者，通过团队评审积累经验的审查者，在进行个人评审时也会变得更快。一项有趣的变体研究是比较不同团队随时间推移在评审速度方面受到的影响：一些团队继续采用个人评审，而另一些团队则转为团队评审。

在组审查请求与个人审查请求中，评审负荷的分布有何不同？在 Mozilla 这样的组织中，少数审查者承担了数量不成比例的修订工作，从而造成流程瓶颈。受焦点小组中开发者经验的启发，探究群组审查请求是否能显著减轻这些关键人员的压力，以及如何做到这一点，将会很有意义。

最后，深入探究团队评审为何与较少的回归缺陷相关，以及团队评审与个人评审分别能发现哪些类型的回归缺陷，将会很有意义。

6 效度威胁

本研究容易受到各种效度威胁的影响，具体讨论如下。我们也介绍了为尽量降低风险而做出的选择。

构念效度 我们的模型使用代理指标来近似现实世界的特征。例如，我们使用“过往变更”作为代理指标来衡量作者经验。为尽量降低使用单一代理指标及其局限性所带来的风险，我们会使用多个代理指标来衡量同一特征。用于衡量作者经验的其他代理指标还包括此前已接受的修订、成功率和评审。

有时，由于缺少更准确的测量代理指标，我们的分析受到限制。例如，已有研究表明，新增和删除的代码行数不够准确，除非将变更的行数纳入考量 [29]。遗憾的是，可用的数据集不包含这类信息。同样，将 “首次评审请求”用作推断群组评审请求的代理指标也存在局限。例如，审查者重新分配的信息并不可用。

将回归缺陷作为代码审查质量的代理指标有其局限性。由于回归缺陷是在事后识别的，一些回归缺陷可能尚未被发现，尤其是近期的修订中。

内部效度 我们的研究发现可能存在局限，因为本研究使用的模型未能充分捕捉推动因变量变化的潜在因素。多元回归模型只能解释约三分之一的方差，R^2 值为 0.33；逻辑回归模型的解释力同样较低（McFadden 的伪 R^2 为 0.10）。尽管如此，这些数值与类似的既往研究一致 [56]。

可推广性 本研究依赖于单一来源的数据，即 Firefox 项目，这对外部效度构成了威胁。我们采用这种方法，是因为除 Firefox 以外的项目很少使用组审查请求。因此，研究结果未必能充分推广到其他项目或组织，还应在更多情境中加以验证。

7 结论

在本研究中，我们考察了 Mozilla Firefox 项目的 66,000 多次修订，以评估组审查请求与个人请求在影响代码审查流程方面的差异。通过统计建模和焦点小组讨论，我们发现，组审查请求与更高的审查质量相关，且速度没有明显差异。其他优势还包括更合理的工作量分配、促进审查者成长的机会，以及更高的可见性。总体而言，研究结果表明，组审查请求是提升审查流程和结果，同时保持速度的有效方式。

参考资料

[1] Hervé Abdi。2007年。多重比较的 Bonferonni 和 Šidák 校正。《测量与统计百科全书》3（2007年1月）。[2] Sumit Asthana、Rahul Kumar、Ranjita Bhagwan、Christian Bird、Chetan Bansal、Chandra Maddila、Sonu Mehta 和 B. Ashok。2019年。WhoDo：大规模自动推荐评审者。载于 *2019*年第*27*届 *ACM* 联合会议：欧洲软件工程大会和软件工程基础研讨会（*ESEC/FSE '19* 10 1145 3338906 3340449）。ACM。数字对象唯一标识符：./。[3] 2013 Alberto Bacchelli 和 Christian Bird。。现代代码审查的预期、结果与挑战。载于*2013*年第*35*届国际软件工程会议（*ICSE*）。 712 721 10 1109 2013 6606617 – 。数字对象唯一标识符：./ICSE.。Clau Batista。[4] 2022 。补丁栈。https://claubatista.github.io/post/01-15-2022-patch-stack/。 [访问日期：07-06-2025]。[5] Olga Baysal、Oleksii Kononenko、Reid Holmes 和 Michael W. Godfrey。2013年。非技术因素对代码审查的影响。载于*2013*年第*20*届逆向工程工作会议（*WCRE*）。 122 131 10 1109 2013 6671287 – 。数字对象唯一标识符：./WCRE.。[6] 2015 Christian Bird、Trevor Carnahan 和 Michaela Greiler。。从构建和部署代码评审分析平台中汲取的经验。载于*2015*年 *IEEE/ACM* 第*12*届软件仓库挖掘工作会议。IEEE。数字对象唯一标识符：10.1109/msr.2015.25[7] BitBucket。2024年。拉取请求的评审者群组。https://confluence.atlassian.com/bitbucketserver/reviewer-groups-for-pull-requests-1064084872.html。访问日期：2025-08-24。[8] Amiangshu Bosu、Michaela Greiler 和 Christian Bird。2015年。有效代码评审的特点：Microsoft 的一项实证研究。载于第*12*届软件仓库挖掘工作会议论文集

软件仓库挖掘工作会议（佛罗伦萨，意大利）（MSR '15）。IEEE出版社，146–156。[9] Virginia Braun 和 Victoria Clarke。2006年。在心理学中运用主题分析。《心理学中的定性研究》3，2（2006年1月），77–101。数字对象唯一标识符：10.1191/1478088706qp063oa [10] 2024 10 5281 4911345 [11] 1993 Marco Castelluccio。。bugbug。数字对象唯一标识符：. /ZENODO。Norman Cliff。。优势统计量：用于回答序数问题的序数分析。《心理学公报》114，3（1993年11月），494–509。数字对象唯一标识符：10.1037/0033-2909.114.3.494[12] Mark Côté。2018年。为 mozilla-central 启用 Phabricator 和 Lando。https://groups.google.com/g/mozilla.dev.platform/c/JKbDxHSaVfM/m/GuoQt4TlAQAJ。[访问日期：07-06-2025]。[13] Harald Cramér。1999年。《统计学数学方法》（PMS-9）。普林斯顿大学出版社。http://www.jstor.org/stable/j.ctt 1 9 4 bpm r Dror G. Feitelson、[14] Eitan Frachtenberg 和 Kent L. Beck。2013年。Facebook 的开发与部署。《IEEE互联网计算》17，4（2013年7月），8–17。数字对象唯一标识符：10.1109/mic.2013.25[15] David Freedman、Robert Pisani 和 Roger Purves。2007年。《统计学》（第4版）。W. W. Norton & Company。[16] Belal Dahiam Saif Ghaleb。2024年。社会惰化：理解、缓解并提升群体绩效。《国际科学多学科研究期刊》2, 9 2024 1321 1328 10 55927 2 9 10975（9月），–。数字对象唯一标识符：. / [17] 2025 ijsmr.v i 。GitHub。。关于代码所有者。https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/about-code-owners。访问日期：2025-08-24。[18] Ulrike Grömping。2006年。R 中线性回归的相对重要性：relaimpo 软件包。《统计软件期刊》17，1（2006年）。数字对象唯一标识符：10.18637/jss.v017.i01[19] Joe F. Hair、Christian M. Ringle 和 Marko Sarstedt。2011年。偏最小二乘结构方程模型：确实是一种灵丹妙药。《营销理论与实践期刊》19，2（2011年4月），139–152。数字对象唯一标识符： 10 2753 1069 6679190202 . /mtp - Giselmar A. J. Hemmert、Laura M. Schons、Jan Wieseke 和 Heiko Schimmelpfen-nig。2016年。逻辑回归中基于对数似然的伪R²：推导样本敏感基准。《社会学方法与研究》47，3（3月 2016 507 531 10 1177 0049124116638107），–。数字对象唯一标识符：. / [20] Daniel P. Heyman。书评：《排队系统》第一卷：Leonard Kleinrock 著《理论》。《网络》6, 2 1976 189 190 10 1002 3230060210（），–。数字对象唯一标识符：. /net。[21] 1976 Yujuan Jiang、Bram Adams 和 Daniel M. German。。我的补丁能通过吗？要等多久？Linux 内核案例研究。载于2013年第十届软件仓库挖掘工作会议（MSR）。IEEE。数字对象唯一标识符：10 1109 2013 6624016 . /msr. 。Oleksii Kononenko、Olga Baysal [22] 2013 和 Michael W. Godfrey。。代码审查质量：开发者如何看待。载于第38届国际软件工程会议论文集（ICSE '16 10 1145 2884781 2884840）。ACM。数字对象唯一标识符：. / 。Oleksii [23] 2016 Kononenko、Olga Baysal、Latifa Guerrouj、Yaxin Cao 和 Michael W. Godfrey。2015年。探究代码审查质量：人员和参与度是否重要？载于2015年 IEEE 软件维护与演化国际会议（ICSME）。111 120 10 1109 2015 7332457 – 。数字对象唯一标识符：. /ICSM. 。Jyrki Kontio、[24] Johanna Bragge 和 Laura Lehtola。。焦点小组法作为软件工程中的实证工具。Springer伦敦，93–116。数字对象唯一标识符：10.1007/978-1-84800-044-5 4[25] 2008 2025 Matej Kucera。。复现包。数字对象唯一标识符：10 6084 9 29532185 . /m 。Figshare。Arun K. Kuchibhotla、[26] Lawrence D. Brown、Andreas Buja 和 Junhui Cai。2019年。线性回归全解。数字对象唯一标识符：10 48550 1910 06386 . /ARXIV. 。Gunnar Kudrjavets、Aditya Kumar、[27] Nachiappan Nagappan 和 Ayushi Rastogi。2022年。挖掘代码审查数据以了解接受与合并之间的等待时间：一项实证分析。载于第19届国际软件仓库挖掘会议论文集（MSR '22）。ACM，579–590。数字对象唯一标识符：10.1145/3524842.3528432[28] Gunnar Kudrjavets、Nachiappan Nagappan 和 Ayushi Rastogi。2022年。小型代码变更是否更快合并？一项多语言实证研究。载于第19届国际软件仓库挖掘会议论文集（MSR '22 537 548 10 1145 3524842 3528448）。ACM，–。数字对象唯一标识符：. / 。Gunnar Kudrjavets 和 Ayushi Rastogi。[29] 2023 。代码审查速度对从业者重要吗？（2023 10 48550 2311 02489）。数字对象唯一标识符：. /ARXIV. 。Gunnar Kudrjavets、Jeff Thomas、Nachiappan Nagappan 和 Ayushi Rastogi。[30] 2022年。内核代码与非内核代码有何不同？BSD系列操作系统案例研究。载于2022年 IEEE 软件维护与演化国际会议（ICSME）。IEEE。数字对象唯一标识符：10 1109 55016 2022 00027 . /[31] icsme 。。Zhixing Li、Yue Yu、Tao Wang、Gang Yin、ShanShan Li 和 Huaimin Wang。2022年。你还在处理这个吗？关于拉取请求弃置的一项实证研究。《IEEE软件工程汇刊》48，6（2022年），2173–2188。数字对象唯一标识符：10.1109/TSE.2021.3053403[32] Chandra Maddila、Sai Surya Upadrasta、Chetan Bansal、Nachiappan Nagappan、Georgios Gousios 和 Arie van Deursen。2020年。Nudge：加速逾期拉取请求的完成。（2020 10 48550 2011 12468）。数字对象唯一标识符：. /ARXIV. 。H. B. [33] 1947 Mann 和 D. R. Whitney。。关于检验两个随机变量中一个是否在随机意义上大于另一个。《数学方法》《统计学》18, 1 1947 50 60 10 1214 1177730491（3月），–。数字对象唯一标识符：。/aoms/ [34] 2018 Katerina M. Marcoulides 和 Tenko Raykov。。回归模型中使用潜变量建模方法评估方差膨胀因子。《教育与心理测量》79，5（2018年12月），874–882。doi:10.1177/0013164418817803[35] Daniel McFadden。1974。定性选择行为的条件Logit分析。载于《计量经济学前沿》，Paul Zarembka（编）。Academic Press，纽约，105–142。[36] Microsoft。2025。分支策略和设置。https://learn.microsoft.com/en-us/ azure/devops/repos/git/branch-policies?view=azure-devops&tabs=browser。访问日期：2025-08-24。[37] Microsoft。2025。Bugzilla。https://bugzilla.mozilla.org/home。[访问日期：07-06-2025]。[38] Audris Mockus、Roy T. Fielding 和 James D. Herbsleb。2002。开源软件开发的两个案例研究：Apache 和 Mozilla。《ACM软件工程与方法学汇刊》11，3（2002年7月），309–346。doi:10.1145/ 567793.567795[39] Mozilla。2015。创建提交并提交审查请求。https://www.mozilla.org/en-US/about/governance/policies/commit/review/commits.html#specifying-reviewers-in-commit-messages。[访问日期：20-09-2025]。[40] Mozilla。2025。Bugzilla。https://bugzilla.mozilla.org/home。[访问日期：07-06-2025]。[41] Mozilla。2025。Firefox公共数据报告：用户活动仪表板。https: //data.firefox.com/dashboard/user-activity。访问日期：2025-06-24。[42] Mozilla。2025。获取评审；Firefox源代码文档。https: //firefox-source-docs.mozilla.org/contributing/reviews.html。[访问日期：07-06-2025]。[43] Mozilla。2025。Herald：所有规则。https://phabricator.services.mozilla.com/herald/query/all/。[访问日期：07-06-2025]。[44] Mozilla。2025。如何为 Firefox 贡献代码。https://firefox-source-docs.mozilla.org/setup/contributing_code.html#getting-your-code-reviewed。[访问日期：07-06-2025]。[45] Mozilla。2025。如何标记回归缺陷。https://firefox-source-docs.mozilla.org/bug-mgmt/processes/regressions.html。[访问日期：27-06-2025]。[46] Mehrdad Nurolahzade、Seyed Mehdi Nasehi、Shahedul Huq Khandkar 和 Shreya Rawal。2009。补丁审查在软件演化中的作用：对 Mozilla Firefox 的分析。载于软件演化原理（IWPSE）与软件演化（Evol）联合国际及年度 ERCIM 研讨会论文集（ESEC/FSE *09* 9 18 10 1145 1595808 1595813）。ACM，–。数字对象唯一标识符：。/。[47] 1900 Karl Pearson。。X. 关于相关变量系统中一组偏离最可能值的偏差标准：该系统是否可以合理地被认为是由随机抽样产生的。《伦敦、爱丁堡和都柏林哲学杂志与科学期刊》50，302（7月 1900 157 175 10 1080 14786440009463897），–。数字对象唯一标识符：。/ Chao-Ying Joanne Peng、[48] Kuk Lida Lee 和 Gary M. Ingersoll。2002。逻辑回归分析与报告简介。《教育研究杂志》96, 1 2002 3 14 10 1080 00220670209598786（9月），–。数字对象唯一标识符：。/ Eric [49] 1999 Raymond。。《大教堂与市集》。《知识、技术与政策》12，3（1999），23–49。[50] Caitlin Sadowski、Emma Söderberg、Luke Church、Michal Sipko 和 Alberto Bacchelli。2018。现代代码审查：Google 的案例研究。载于第40届国际软件工程会议：软件工程实践（ICSE '18）论文集。ACM，181–190。doi:10.1145/3183519.3183525[51] Tony Savor、Mitchell Douglas、Michael Gentili、Laurie Williams、Kent Beck 和 Michael Stumm。2016。Facebook 和 OANDA 的持续部署。载于2016年 IEEE/ACM 第38届国际软件工程会议伴随会议（ICSE-C）。21–30。[52] Qianhua Shan、David Sukhdeo、Qianying Huang、Seth Rogers、Lawrence Chen、Elise Paradis、Peter C. Rigby 和 Nachiappan Nagappan。2022。运用助推大规模加速代码审查。载于第30届 ACM 欧洲软件工程大会与软件工程基础研讨会联合会议（ESEC/FSE '22 472 482 10 1145 3540250 3549104）论文集。ACM，–。数字对象唯一标识符：。/。Masha Shunko、Julie Niederhoff 和 Yaroslav Rosokha。。人不是机器：排队设计对服务时间的行为影响。《管理科学》64, 1 2018 453 473 10 1287 2016 2610（1月），–。[53] 2018 数字对象唯一标识符：。/mnsc. 。Hummy Song、Anita L. Tucker 和 Karen L. Murrell。。队列合并的不经济性：对急诊科住院时长的实证调查。《管理科学》61，12（2015年12月），3032–3053。doi:10.1287/mnsc.2014. 2118[55] Jingqi Wang 和 Yong-Pin Zhou。2018。队列配置对服务时间的影响：来自超市的证据。《管理科学》64，7（2018年7月），3055–3075。doi:10.1287/mnsc.2017.2781[56] Yue Yu、Huaimin Wang、Vladimir Filkov、Premkumar Devanbu 和 Bogdan Vasilescu。2015。稍候：GitHub 上拉取请求评估延迟的决定因素。载于2015年 IEEE/ACM 第12届软件仓库挖掘工作会议。367–371。doi:10.1109/MSR.2015.42[57] Xunhui Zhang、Yue Yu、Tao Wang、Ayushi Rastogi 和 Huaimin Wang。2022。拉取请求延迟解析：实证概览。《实证软件工程》27 [54] 2015