我们是在加速还是减速？代码速度的时间维度

Gunnar Kudrjavets格罗宁根大学9712 CP 格罗宁根，荷兰 g.kudrjavets@rug.nl

Nachiappan Nagappan*Meta Platforms,Inc.*门洛帕克，加利福尼亚州 94025，美国 nnachi@meta.com

Ayushi Rastogi格罗宁根大学9712 CP 格罗宁根，荷兰 a.rastogi@rug.nl

arXiv:2303.04293v1 [cs.SE] 7 Mar 2023

摘要——**本文研究了各种代码审查周期的持续时间如何随项目生命周期发生变化。我们研究了四个开源软件（OSS）项目：Blender、FreeBSD、LLVM 和 Mozilla。我们挖掘并分析了 283,235 次代码审查的特征，这些审查平均涵盖七年的开发活动。我们的主要结论是，时间的流逝和项目规模都不会影响代码速度。我们发现：（a）对于 FreeBSD、LLVM 和 Mozilla，各种代码审查周期的持续时间（首次响应时间、接受时间和合并时间）要么缩短，要么保持不变；对于 Blender 则不存在明确的方向性趋势；（b）代码库规模的增加（每年 3–17%）并不伴随代码速度下降；（c）对于 FreeBSD、LLVM 和 Mozilla，合并时间的 30天移动中位数保持在一个固定范围内。这些发现不会因代码变更指标的波动而改变，例如提交数量或代码变更的不同作者数量。**

索引词——**代码审查、代码速度、开发者生产力**

I. 引言

软件行业的一个关键目标是快速开发、审查、集成和部署代码变更。软件行业致力于提高代码速度 [1], [2], [3] ，并采用各种工具或流程改进措施。持续集成（CI）和持续部署（CD）的各种实践形式 [4]已成为行业和开源软件（OSS）社区中大多数当前项目的默认做法。同样，轻量级的现代代码审查 [5], [6] 如今也已成为开展代码审查的事实标准流程。对于现有的非敏捷项目而言，从瀑布模型等方法转向 CI/CD，或从邮件列表转向现代代码协作工具，能够立即显著提升代码速度 [7], [8]。然而，这些代码速度提升能够持续多久，仍有待深入研究。

我们着重研究软件项目不断演进时，各个代码审查周期的时长变化方向，旨在确定代码审查速度会如何随时间变化。我们从四个不同的开源软件项目中挖掘代码审查数据：Blender [9], FreeBSD [10],LLVM [11], 和 Mozilla [12]。我们分析了 283,235 次代码审查，并评估不同代码审查周期（首次响应时间、接受时间和合并时间）随时间呈现的趋势。研究结果表明，随着项目演进，代码审查速度保持不变。

II. 背景与动机

对代码速度的理解和定义范围各不相同。一种通用定义是“从进行代码变更到将其交付给客户所需的时间” [13]。本文聚焦于一种与代码审查时长相关、可量化程度更高的指标。我们使用合并时间作为代理指标，以量化代码变更传播的速度。合并时间涵盖这样一段时间：从工程师发布一组已准备好接受代码审查的代码变更，到这些变更合入目标分支为止 [14]。

业界对于代码速度随时间变化的趋势，存在两种流行但相互矛盾的理论。第一种假设认为，由于系统规模和复杂性不断增加（Lehman 第二定律“复杂性不断增加” [15]），例如团队规模扩大导致沟通复杂性上升，代码速度会降低。第二种假设则认为，代码速度会随时间推移而提高。代码速度提高的原因在于，工程师会逐渐熟悉代码库，人际沟通会变得更加高效，而且工具基础设施也会不断改进。

Lehman 软件演化定律中的另一条定律是“持续增长” [15]。Lehman 第六定律指出：“[程]序的功能内容必须不断增加，才能在其生命周期内维持用户满意度” [16]。由于功能有所增加，我们可以合理地假设代码库的规模也会随之扩大。代码库的规模通常以源代码行数（SLOC）衡量。根据我们在行业中的经验，我们还观察到，随着项目需求不断增加，开发团队的规模往往会扩大，而非缩小。根据 Brooks 定律，“[向]一个进度落后的软件项目增加人手，只会让它更加落后” [17]。虽然开发新功能时，软件项目不一定会因此延误，但代码库和团队规模的扩大如何影响代码速度，目前尚不清楚。为进一步研究这一问题，我们提出以下研究问题：

研究问题： 项目的代码速度随时间呈现怎样的趋势？代码速度是增加、降低，还是保持不变？

表I：修正后的曼-肯德尔（MK）检验的p 值，以及森斜率（Theil-Sen估计量）的变化幅度。我们展示了不同代码审查周期的30天移动中位数和全时段中位数。具有统计显著性的曼-肯德尔p 值表明存在上升或下降的单调趋势。森斜率（以95%的置信水平呈现）用于确定每单位时间步长内的趋势幅度。时间步长为30天。代码审查周期的单位为小时。

	首次响应时间				接受时间				合并时间			
	30天中位数		全时段中位数		30天中位数		全时段中位数		30天中位数		全时段中位数	
Name	MK	森	MK	森	MK	森	MK	森	MK	森	MK	森
Blender	.231		.229		.715		.846		.609		.666	
FreeBSD	$< .001$	-0.003	$< .001$	-0.002	$< .001$	-0.011	$< .01$	-0.003	.193		$< .05$	0.006
LLVM	$< .001$	0.000	$< .001$	0.000	$< .001$	-0.005	$< .001$	-0.002	$< .001$	-0.009	$< .001$	-0.002
Mozilla	$< .001$	-0.001	$< .001$	-0.001	$< .01$	0.000	$< .001$	0.000	$< .01$	-0.001	.060	

III. 方法论

A. 数据选择

第二节指出，我们主要关注的代码速度指标是合并时间 [14]。我们还知道，Google [18] 和微软 [5] 的研究发现，首次响应时间和接受时间也是对开发者很重要的代码审查指标。

现有的代码审查数据集要么聚焦于 GitHub [19]，要么聚焦于 Gerrit [20], [21], [22]。默认情况下，Gerrit 会在变更获批后立即将其合并。这意味着，将接受时间和合并时间视为两个独立事件既无价值，也不合理。GitHub 于 2016 [23]新增了仅批准变更的功能。我们随机选择了 100 个 GitHub 项目，并检查了这些项目中的拉取请求。要产生有价值的数据，该功能的采用率和持续使用率仍需进一步提高。

Phabricator 是一种代码协作工具，它公开了一种数据模型，可正式跟踪各个代码审查周期 [24]。根据有关现有 Phabricator 项目的公开信息 [25], ，我们为四个拥有多年开发历史的主要开源软件项目挖掘了数据。这些项目是 Blender、FreeBSD、LLVM 和 Mozilla。我们的初始数据集包含 283,235 项代码审查。我们删除了未进行现代代码审查的记录。不符合要求的代码审查示例包括：不包含任何代码行的审查、作者批准了自己变更的审查，以及未经任何批准便提交的审查。经过筛选并应用一致性检查后，最终数据集包含 280,456 项代码审查。

B. 统计分析

首先，我们考察代码速度是否存在趋势。我们使用曼-肯德尔检验 [26], [27] 来确定是否存在单调上升或下降趋势。单调趋势 [28] 表示“变量随时间持续增加（或减少），但趋势不一定是线性的” [29]。曼-肯德尔检验的零假设（H_0）是不存在单调趋势。备择假设（H_1）是存在单调趋势。其次，如果存在统计显著的曼-肯德尔相关性，则我们计算森斜率（Theil-Sen 估计量）[30], [31] ，以评估趋势的幅度。森斜率表示每个单位时间步长的变化率。

如何处理自相关是时间序列分析面临的一项挑战。一种常用方法是对时间序列进行聚合，采用更粗略的时间粒度，例如按月或按年取样 [32]。另一种缓解方法是使用修正后的 Mann-Kendall 检验来调整自相关 [33], [34]。我们采用这两种方法，以降低在趋势并不存在时错误地得出趋势存在这一结论的可能性。

我们针对三个代码审查时间段进行了这些计算：首次响应时间、接受时间和合并时间。我们考察了30天滚动（移动）中位数和全时段中位数。全时段中位数是截至某一给定时间的特定指标的中位数。这两项指标都有助于评估特定变量的趋势 [35]。我们选择中位数而非均值，因为中位数对离群值的抵抗力更强。统计检验采用的显著性水平为.05 α 。

IV. 结果

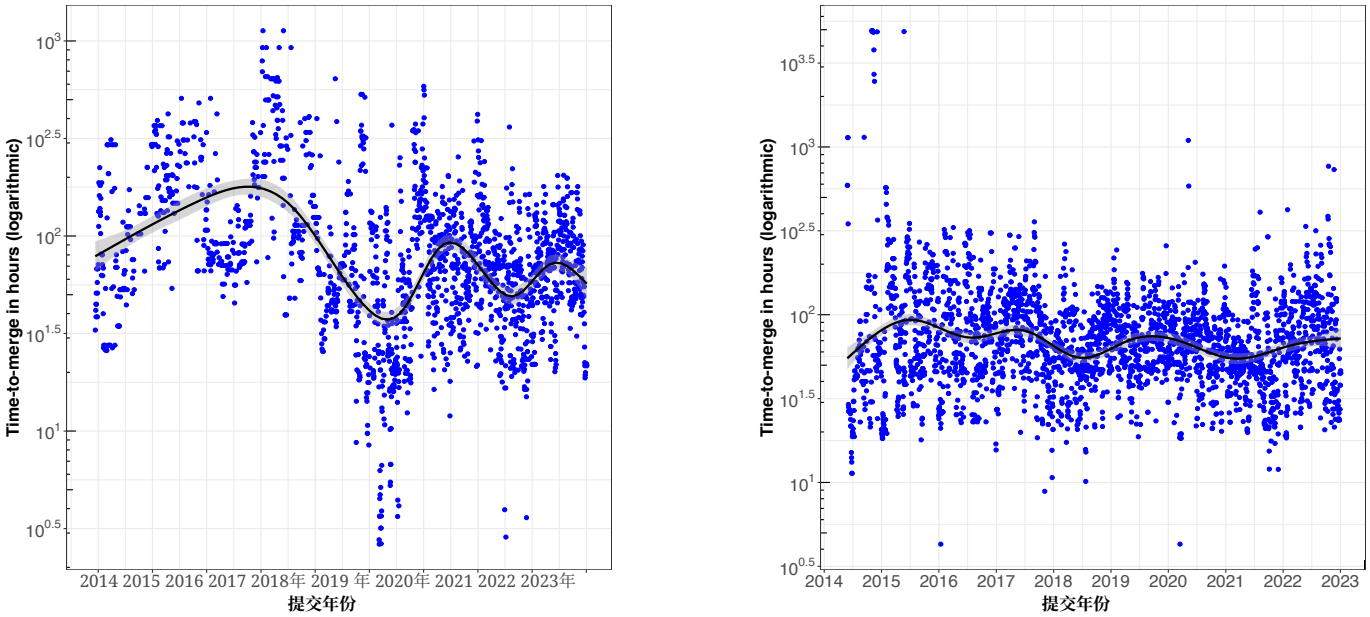
统计分析结果见表I。分析的主要观察结果如下：(a) 除 Blender 外，所有项目都存在具有统计显著性的单调趋势；(b) 尽管趋势具有统计显著性，但根据数值和可视化结果（见图1），其幅度很小；(c) 所有30天中位数斜率值均为负值或零，这表明代码速度略有提升。

使用 30 天滚动平均值或中位数是标准做法。根据具体情境，也会使用 90 天（季度）技术指标来检查趋势。我们还使用 90 天滚动中位数计算曼-肯德尔值和 Sen斜率，作为额外的数据点。由此得出的结论并未改变。

作为上述分析的补充，图1展示了 30 天滚动中位数合并时间的趋势可视化结果。根据目视观察，我们注意到，对于 FreeBSD、LLVM 和 Mozilla，大多数中位数都处于相对固定的范围内。

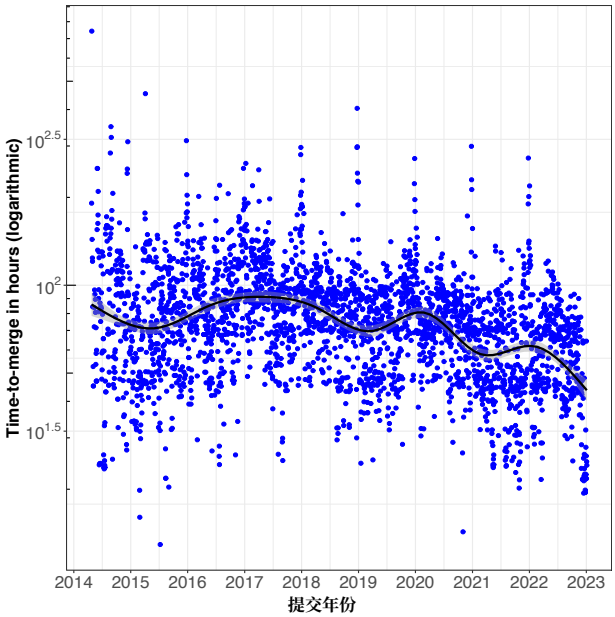
对于 Blender，结果中有一组明显的聚集，表明 2017 年至 2019 年间代码速度有所提升。

图 1：不同项目的合并时间 30天滚动中位数趋势。

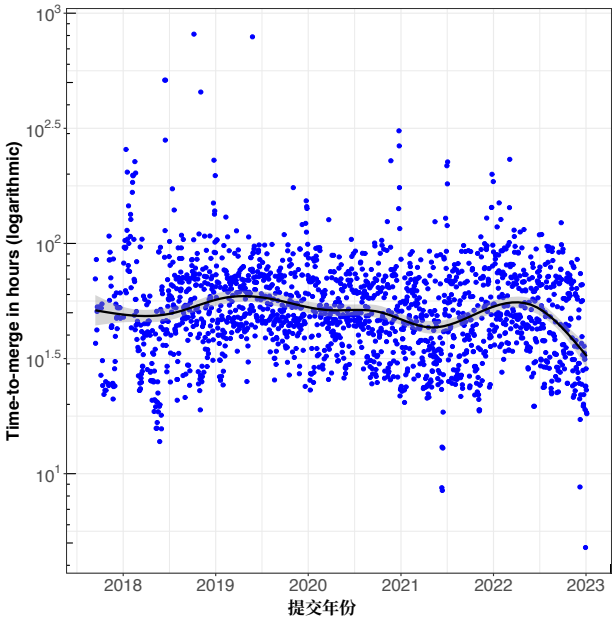


(a) Blender。

(b) FreeBSD。



(c) LLVM。



(d) Mozilla。

为进一步调查这一结果，我们检查了 2017 年至 2019 年间随机抽取的 50 条 Blender 代码审查记录。我们无法找到能够确切解释 30天滚动中位数数值下降的证据。根据公开信息（“Blender 现在有一支规模大得多的团队负责核心开发”），我们推测，工程师人数的突然增加可能导致代码速度暂时提升 [36]。由于“前所未有的 108 位新贡献者” [37]，这一趋势可能在 2020年恢复正常。

各种数据点和事件都可能影响项目的代码速度。只有少数这类潜在变量会被正式跟踪。关于功能开发、项目管理、组织挑战或业务优先级变化的决策，并非总会被记录下来并向公众公开。我们对所有混杂变量的了解有限。为了理解与项目开发相关的指标可能产生的影响，我们挖掘了能够获取的数据。研究结果见表 II。各项目唯一共同的指标是源代码行数持续增长。我们聚焦于可从客观特征中

源代码管理系统和缺陷数据库中挖掘的数据。我们考察源代码行数总量的年度变化（使用 scc [38] 计算）、提交数量、不同作者的数量以及不同提交者的数量。我们使用每个项目原始 Git 仓库的默认分支，或可用的 GitHub 只读镜像中的默认分支。

表II：各项目代码变更指标年度增减百分比的中位数。统计范围为图1所涵盖的代码审查周期。由于作者和提交者的角色可能不同，因此将二者区分开来。

项目	周期	SLOC	提交数量	不同作者	不同提交者
Blender	2014–2022	6.2%	−2.7%	13。5%	2.3%
FreeBSD	2014–2022	3.5%	−1.2%	0.5%	−4.3%
LLVM	2014–2022	17。4%	6.8%	18.5%	15。6%
Mozilla	2017–2022	11。1%	−4.8%	−6.9%	−13.8%

我们调查了与所分析项目相关的缺陷跟踪数据库内容。我们发现，与缺陷相关的指标并不能可靠地反映项目的工作负荷。我们没有发现有关缺陷数量及其对开发流程影响的正式规则。此外，我们也没有观察到开发者将分配给自己的缺陷作为主要工作项清单。

V. 讨论

我们的主要发现是，不同代码审查周期的趋势没有显著变化。无论提交次数、不同作者数量和不同提交者数量增加还是减少，这一趋势都适用于所有项目。所有项目的代码库规模年度变化中位数均增长了 3–17%。与此同时，不同代码审查周期的 30 天中位数和全时段中位数仅在小数点后第二位或第三位发生变化（见表I）。如果说有什么变化，那么首次响应时间、接受时间和合并时间的持续时长略有缩短。

主要发现令人惊讶。软件工程领域的传统观点认为，随着项目规模扩大、存在时间变长，沟通和流程都会变慢。我们观察到的结果可能有多种解释。一种可能是，尽管代码复杂度和团队规模持续增加，工程师在项目代码库中工作时也变得更加熟悉、高效。随着时间推移，开发者也会建立更好的人际关系，从而提高沟通效率。可以合理推测，开发基础设施也会有所改善。这些因素能够抵消处理复杂调试问题所耗费的时间，或弥补审查代码的理解速度。

另一种可能是，无论项目规模如何，受人性等因素影响，代码速度都会维持在一个特定范围内。工程师会在有空时审查代码。无论开发者对代码有多熟悉，或有其他并行优先事项，他们都会花固定的时间进行代码审查。在图1中，FreeBSD、LLVM 和 Mozilla 的“平稳性”或缺乏趋势清晰可见。Blender 项目的合并耗时则更为多变。这种差异可以用 Blender 是我们研究的所有项目中规模最小的项目来解释。例如，Blender 各年的提交中位数和不同作者人数分别为 8,589 和 89，而 LLVM 则分别为 29,555 和 592。

另一种解释与组织的自我纠正行为有关。我们研究的每个产品都有一个非正式或正式的核心团队。核心团队由项目中最活跃或资历最深的成员组成。根据我们对开源软件的经验，代码速度处于特定范围内，也可能是核心团队成员确保代码审查得到及时响应、待处理问题数量不断减少的结果。

VI. 有效性威胁

我们的研究面临特定类别的威胁。其中一种威胁与我们的研究结果在其他情境中的应用或外部有效性 [39]有关。我们研究的项目全部属于开源软件项目。开源软件开发中的激励结构不同于工业项目。此外，由于我们关注的是特定粒度的代码审查，项目选择范围仅限于使用 Phabricator 进行代码协作的项目。在目标环境中进行进一步复制研究之前，我们不建议将这些结果加以泛化。

另一种威胁与内部有效性有关。这类威胁涉及对结果的解释，以及是否能从数据中得出正确的结论。我们采用标准推荐的非参数统计方法来得出结论。滚动30日均值和中位数等指标在量化金融领域被广泛用于指示趋势 [35]。我们通过计算90日中位数和全时段中位数来 corroborateourfindings bycalculating90-day and all-time medians that indicate a similar trend。

VII. 结论与未来工作

我们研究了四个主要开源软件项目中的代码速度趋势：Blender、FreeBSD、LLVM 和 Mozilla。我们的分析基于283,235次代码审查，涵盖每个项目平均七年的开发活动。我们发现代码速度不会随时间推移而下降。虽然这些项目的代码库规模每年以中位数计增长3–17%，但代码速度要么保持不变，要么略有提升。

我们计划在 Gerrit 和 GitHub 等其他代码协作工具上复现我们的发现。虽然它们提供的粒度不及 Phabricator，但相关发现将有助于证实或推翻我们的论断。此外，我们观察到代码速度落在特定范围内，这一现象值得进一步研究。

VIII. 数据可用性

Phabricator 数据、用于执行统计分析的各种 R脚本以及相关 SQL查询可在 Figshare 上获取。[1]

[1]https://figshare.com/s/4558d92adc8d5d262bd6

参考文献

[1] C. Maddila、S. S. Upadrasta、C. Bansal、N. Nagappan、G. Gousios 和 A. v. Deursen，“Nudge：加速逾期拉取请求完成”，ACM 软件工程与方法学汇刊，2022年5月。 [在线]。可访问：https://doi.org/10 1145 3544791 。/ P. Riggs。（11月）加快速度，减少等待：改善 Meta 的代码审查时间。 [在线]。可访问：https://engineering.fb.com/2022/11/16/culture/meta-code-review-time-improving/ [3] L. Chen、P. C. Rigby 和 N. Nagappan，“理解为何无法建模代码审查所需时长：一项工业案例研究”，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论文集，系列：ESEC/FSE 2022。美国纽约州纽约市：计算机协会，2022年，第1314–1319页。 [在线]。可访问：https://doi.org/10 1145 3540250 3558945 。/ . M. Fowler。（5月）持续集成。 [在线]。可访问：https://martinfowler.com/articles/continuousIntegration.html [5] C. Bird、T. Carnahan 和 M. Greiler，“构建和部署代码审查分析平台的经验教训”，载于2015年 IEEE/ACM 第12届软件仓库挖掘工作会议（软件仓库挖掘）。美国加利福尼亚州洛斯阿拉米托斯：IEEE计算机学会，2015年5月，第191–201页。[在线]。可访问：https://doi.ieeecomputersociety.org/10.1109/MSR。2015.25[6] C. Sadowski、E. Söderberg、L. Church、M. Sipko 和 A. Bacchelli，“现代代码审查：Google案例研究”，载于第40届国际软件工程大会：实践中的软件工程，系列：ICSE-SEIP ’18。瑞典哥德堡：计算机协会，2018年5月，第181–190页。 [在线]。可访问：https://doi.org/10 1145 3183519 3183525 。/ 。M. M. Jha、R. M. F. Vilardell 和 J. Narayan，“扩展敏捷 Scrum 软件开发：通过缩短上市时间为平台开发提供敏捷性和质量”，载于2016年 IEEE 第11届全球软件工程国际会议（全球软件工程国际会议），2016年，第84–88页。 [在线]。可访问：https://doi.org/10.1109/ICGSE。2016.24[8]K. Kaur、M. Khurana 和 Manisha，“敏捷 Scrum 方法对上市时间和代码质量的影响——案例研究”，载于2021年第3届计算、通信、控制与网络进展国际会议（计算、通信、控制与网络进展国际会议），2021年，第1673–1678页。 [在线]。可访问：https://doi.org/10.1109/ICAC3N53548.2021。9725375[9] Blender基金会。（2023）Blender 的 Phabricator。 [在线]。可访问：https://developer.blender.org/ [10] FreeBSD基金会。（2023）FreeBSD 的 Phabricator。 [在线]。可访问：https://reviews.freebsd.org/ [11] LLVM基金会。（2023）LLVM 的 Phabricator。 [在线]。可访问：https://reviews.llvm.org/ [12] Mozilla基金会。（2023）Mozilla 的 Phabricator。 [在线]。可访问：https://phabricator.services.mozilla.com/ [13] 微软研究院。（2019年5月）第14届 IEEE/ACM 软件测试自动化国际研讨会。加拿大蒙特利尔。[在线]。可访问：https://www.microsoft.com/en-us/research/event/14th-ieee-acm-international-workshop-on-automation-of-software-test/ [14] D. Izquierdo-Cortazar、N. Sekitoleko、J. M. Gonzalez-Barahona 和 L. Kurth，“使用度量指标跟踪代码审查绩效”，载于第21届软件工程评估与评价国际会议论文集，系列：EASE’17。瑞典卡尔斯克鲁纳：计算机协会，2017年6月，第 214 223 [ ] 10 1145 3084226 3084247 –页。在线。可访问：https://doi.org/ 。/ 。M. Lehman，“程序、生命周期与软件演化规律”，IEEE会刊，第68卷，第9期，第1060–1076页，1980年。 [在线]。可访问：https://doi.org/10 1109 1980 11805 。/ PROC。/ 。M. M. Lehman，“软件工程、软件过程及其支持”，软件工程杂志，第6卷，第5期，第243–258页，1991年9月。 [ ] 10 1049 1991 0028 在线。可访问：https://doi.org/ 。/sej。/ 。F. P. Brooks, Jr，《人月神话：软件工程随笔》，第2版。美国马萨诸塞州波士顿：Addison Wesley，1995年8月。[18] Google。（2023）代码审查速度。 [在线]。可访问：https://google.github.io/eng-practices/review/reviewer/speed.html [19] G. Gousios，《GHTorrent 数据集与工具套件》，载于《第十届软件仓库挖掘工作会议论文集》，丛书 MSR ’13。美国加利福尼亚州旧金山：IEEE出版社，2013年5月，第233–236页。 [ ] 10 5555 2487085 2487132 在线。可访问：https://dl.acm.org/doi/。/。M. Mukadam、C. Bird 和 P. C. Rigby，《来自安卓的 Gerrit 软件代码审查数据》，载于《第十届软件仓库挖掘工作会议论文集》，丛书 MSR ’13。IEEE出版社，2013年，第 45 48 [ ] 10 1109 2013 6624002 –页。在线。可访问：https://doi.org/。/MSR。M. Paixao、J. Krinke、D. Han 和 M. Harman，《CROP：将代码审查与源代码变更关联起来》，载于《2018年IEEE/ACM第15届软件仓库挖掘国际会议（MSR）》，2018年，第46–49页。 [ ] 10 1145 3196398 3196466 在线。可访问：https://doi.org/。/。X. Yang、R. G. Kula、N. Yoshida 和 H. Iida，《挖掘现代代码审查仓库：人员、流程与产品数据集》，载于《第13届软件仓库挖掘国际会议论文集》，2016年，第460–463页。 [在线]。可访问：https://doi.org/10 1145 2901739 2903504 。/。C. Wanstrath。（9月）全新的 GitHub 宇宙：宣布推出新工具、论坛和功能。[在线]。可访问：https://github.blog/2016-09-14-a-whole-new-github-universe-announcing-new-tools-forums-and-features/ [24] Phacility。（2021年6月）Phacility——主页。 [在线]。可访问：https://www.phacility.com/ [25] Phabricator。（2021年1月）使用 Phabricator 的组织。 [在线]。可访问：https://secure.phabricator.com/w/usage/companies/ [26] H. B. Mann，《针对趋势的非参数检验》，《计量经济学》，第13卷第3期，第245–259页，1945年。 [在线]。可访问：https://doi.org/10 2307 1907187 。/ M. Kendall，《秩相关方法》，第4版。英格兰伦敦：Hodder Arnold，1976年1月。[28] R. M. Hirsch、J. R. Slack 和 R. A. Smith，《月度水质数据趋势分析技术》，美国地质调查局，技术报告，1981 [ ] 10 3133 81488 。在线。可访问：https://doi.org/。/ofr 太平洋西北国家实验室。（12月）单调趋势的曼-肯德尔检验。 [在线]。可访问：https://vsp.pnnl.gov/help/vsample/design_trend_mann_kendall.htm [30] P. K. Sen，《基于肯德尔τ的回归系数估计》，《美国统计学会杂志》，第63卷第324期，第1379–1389页，1968年。 [在线]。可访问：https://doi.org/10 1080 01621459 1968 10480934 。/。。R. O. Gilbert，《环境污染监测的统计方法》。美国田纳西州纳什维尔：约翰·威立父子出版公司，1987年2月。 [ ] 7037501 在线。可访问：https://www.osti.gov/biblio/ M. C. Coen、E. Andrews、A. Bigi、G. Martucci、G. Romanens、F. P. A. Vogt 和 L. Vuilleumier，《预白化方法、时间粒度和时间分段对曼-肯德尔趋势检测及相关 Sen斜率的影响》，《大气测量技术》，第13卷第12期，第6945–6964页，2020年12月。 [在线]。可访问：https://doi.org/10 5194 13 6945 2020 。/amt- - - K. H. Hamed 和 A. R. Rao，《针对自相关数据的修正曼-肯德尔趋势检验》，《水文学杂志》，第204卷第1–4期，第182–196页，1998年1月。 [在线]。可访问：https://doi.org/10.1016/s0022-1694(97)00125-x [34] S. Yue、P. Pilon、B. Phinney 和 G. Cavadias，《自相关对水文序列趋势检测能力的影响》，《水文过程》，第16卷第9期，第1807–1829页，2002年。 [在线]。可访问：https://doi.org/10 1002 1095 。/hyp。R. J. Hyndman，《移动平均》，载于《国际统计科学百科全书》，M. Lovric 编。柏林、海德堡：施普林格·柏林·海德堡，2011年，第866–869页。 [在线]。可访问：https://doi.org/10 1007 978 3 642 04898 2_380 。/ - - - - T. Roosendaal。（12月） Blender 2019 年亮点盘点。 [在线]。可访问：https://www.blender.org/press/blender-top-16-highlights-of-2019/[37] F. Siddi。（2021年7月）数字中的 Blender——2020年。 [在线]。可访问：https://www.blender.org/news/blender-by-the-numbers-2020/ [38] B. E. C. Boyter。（2022年3月）Sloc Cloc and Code（scc）。 [在线]。可访问：https://github.com/boyter/scc/ [39] F. Shull、J. Singer 和 D. I. K. Sjøberg，《高级实证软件工程指南》。英格兰伦敦：施普林格，2008年。