采矿的承诺和危险

艾里尼·卡尔利亚姆瓦库

加拿大维多利亚大学

伊卡利亚姆@乌威契克

雷夫·辛格

加拿大维多利亚大学

LSinger @乌威契克

摘要

乔治·古西奥斯

加拿大代尔夫特理工大学

戈西奥斯@图德尔福特·nl

凯利·布林科

加拿大维多利亚大学

kblincoe@acm.org

丹尼尔·german⇤

加拿大维多利亚大学

东加利福尼亚

丹妮拉·达米安

加拿大维多利亚大学

丹尼拉德湖

1 .介绍

gitHub 拥有超过 1000 万个 Git 存储库，正成为互联网上软件工件的最重要来源之一。研究人员开始挖掘 GitHub 事件日志中存储的信息，试图了解用户如何利用该网站在软件上进行协作。然而，到目前为止，还没有研究描述 GitHub 提供的数据的质量和属性。我们记录了一项实证研究的结果，该研究旨在了解 GitHub 中存储库的特性，以及用户如何利用 GitHub 的主要特性——即提交、拉请求和问题。我们的结果表明，尽管 GitHub 是软件开发的丰富数据源，但出于研究目的挖掘 GitHub 应该考虑各种潜在的风险。例如，我们显示，大多数项目都是个人的，不活跃的；GitHub 也被用于免费存储和 Web 托管服务；几

乎 40 %的拉请求没有显示为合并，尽管它们是合并的。我们为软件工程师研究人员提供了一套关于如何在 GitHub 中处理数据的建议。

类别和主题描述符

D . 2.8 [软件工程] :管理——软件配置管理

一般条款

软件工程

关键词

挖掘软件储存库，git，GitHub，代码评论，偏见。

⇤corresponding 作家

MSR '14，5 月 31 日- 2014 年 6 月 1 日，印度海德拉巴，˘版权 2014 ACM 978-1-4503-2863-0/14/05...15.00 美元。

MSR'14，2014 年 5 月 31 日至 6 月 1 日，印度海得拉巴，版权所有 2014 ACM 978-1-4503-2863-0/14/05...15.00 美元 http://dx.doi.org/10.1145/2597073.2597074

gitHub 是一个建立在 Git 版本控制系统之上的协作代码托管站点。GitHub 引入了一种“分支&拉”模型，在这种模型中，开发人员创建自己的存储库副本，并在他们希望项目维护人员将他们的更改拉进主分支时提交拉请求。除了代码托管、协作代码审查和集成问题跟踪之外，GitHub 还集成了社交功能。用户可以通过“观看”项目和“跟踪”用户来订阅信息，从而获得关于这些项目和感兴趣用户的信息。用户也有可以填充标识信息并包含他们最近在网站中的活动的简档。

截至 2014 年 1 月，GitHub 已经发布了超过 1060 万个重新定位，是目前世界上最大的代码托管网站。它的流行性、集成的社交功能以及元数据通过可访问的 api 的可用性使得 GitHub 对软件工程研究人员非常有吸引力。现有的研究既有定性的[4、7、16、17、19]也有定量的[10、24、25、26]。定性研究的重点是开发人员如何利用 GitHub 的社交功能形成印象，并对其他开发人员和项目的活动得出结论，以评估成功、绩效和可能的合作机会。定量研究旨在系统地归档 GitHub 的公开数据，并利用这些数据调查 GitHub 环境中的开发实践和网络结构。

作为我们关于 GitHub [15 的合作研究的一部分，我们在 2013 年进行了一项探索性在线调查，以评估开发人员使用 GitHub 的原因，以及它如何支持他们与他人合作。通过分析调查数据，我们注意到 GitHub 存储库也用于严格意义上的软件开发以外的目的。许多受访者使用存储库来托管个人项目，没有任何合

作计划。这表明，将 GitHub 数据“照原样”用于软件工程研究可能存在看不见的重大风险。如果不小心首先确定数据符合研究目的，存储库内容和活动的多样性以及开发人员的意图可能会改变研究结论。

关于 SourceForge 挖掘的数据[14]，也注意到了公开挖掘的数据中存在误解的潜在风险。此外，伯德等人。[6]描述了与利用存储在十年内的信息相关的承诺

https://github.com/features

92

集中式版本控制系统。因此，我们制定了以下研究问题来解决这项研究:

RQ :挖掘 GitHub 用于软件工程研究的承诺和风险是什么?

我们强调在数据和容易被误解的数据区域中可以发现的偏差，与 GitHub data o↵ers.的承诺并列我们利用对 240 名 GitHub 用户进行调查所获得的洞察力来识别潜在的风险，并通过对正确数据集的定量分析以及对 434 份 GitHub 报告的手动检查来提供这些风险的证据。我们就如何最好地利用 GitHub 提供的数据向研究人员提供建议，并概述一些需要避免的分析风险。

2 .背景及相关工作

GitHub 上托管的许多项目都是公开的，因此任何有互联网连接的人都可以查看这些项目中的活动。可用的活动包括围绕问题的交流、请求和提交，包括评论和订阅信息。GitHub 上的大量公共数据使得研究人员能够轻松挖掘项目数据，并且已经创建了各种工具和数据集来帮助研究人员实现这一目标。

2.1 背景

基于网络的代码托管服务，如 GitHub，显然是软件工程研究人员感兴趣的。数据的丰富性和公开性简化了数据收集和处理的一些问题。无论如何，实际的困难仍然存在，可能会改变从数据中得出的结论。

SourceForge 是另一个代码托管站点；在 GitHub 广泛采用[8 号之前，它在流行性方面达到顶峰。Howison 和 Crowston [14]指出，在 Source-Forge 上托管的项目经常被放弃，他们的数据经常被从以前系统导入的数据污染。他们还发现，由于在 SourceForge 空间之外托管的项目数据，信息经常丢失。类似地，魏斯·[28]得出结论，并非所有 SourceForge 数据都被认为是完美的: SourceForge 中类别的名称经常会改变，项目会不断地启动和停止。通过将他的数据与 FLOSS-Mole 的数据进行比较，Weiss 强调指出，关于不活跃和不可访问的项目的信息完全缺失。同样，Rainer 和 Gale [21]对 SourceForge 数据质量进行了深入分析。他们注意到只有 1 %的 SourceForge 项目实际上是活跃的，正如他们的指标所显示的。作者建议谨慎使用 SourceForge 数据，并建议研究机构对来自 SourceForge 等门户网站的数据质量进行评估。因此，我们提出的发现强调了研究人员在从 GitHub 数据中得出结论时要记住的潜在风险。

最近的软件工程研究也强调了错误修复数据集的偏差。这些偏差会损害使用这些数据集的研究的有效性和普遍性。研究人员经常依赖错误之间的联系，并提交开源软件数据的集合，以前称为 OssMole

在提交日志中完成，但是链接的错误只代表固定错误总数的一小部分。伯德等人。[5]发现这组虫子是轮胎人口中有偏见的样本。巴赫曼等人。[3]发现错误跟踪系统本身的错误集可能有偏差，因为并非所有的错误都是通过这些系统报告的。阮等人。[18]发现类似的偏见甚至存在于采用严格指导方针和流程的商业项目中。最近，拉赫曼等人。[20]表明大样本量可以抵消 e↵ects 的偏见。在我们的工作中，我们展示了大型 GitHub 数据集之间存在偏见，并就如何避免这种偏见提供了建议。

2.2 相关工作

其他人以前也研究过 GitHub。在代码托管网站中引入社交功能引起了研究人员的特别关注。几项定性研究采访了 GitHub 用户，以更好地理解这些社交功能是如何被[4、7、16 使用的。调查结果表明，GitHub 用户对其他开发人员和项目的活动和潜力形成印象并得出结论。然后，用户将这些结论内化，以决定跟踪谁、跟踪什么，或者下一步在哪里做出贡献。这些社交特征带来的透明度也有助于团队保持对其成员活动的意识，并将其用于组织工作。Pham 等人。[19]调查了 GitHub 的社交功能所带来的开发者行为的更高可见度是否对开发者的测试行为有影响。通过互见和在线调查，他们强调了在贡献者中促进理想的测试文化的挑战，并提出了这样做的策略。

Tsay 等人。[26]对 5000 个项目进行了定量研究，以了解 GitHub 的社会特征如何促进项目成功。麦当劳和高金斯·[相互查看 GitHub 用户，以确定他们如何衡量项目的成功。他们的研究表明，项目成员认为 GitHub 的社会特征是增加贡献的驱动力。

更多的研究已经超出了 GitHub 的社交功能。Thung 等人。[25]建立了参与 100，000 个 GitHub 项目的开发者社交网络，以展示 GitHub 生态系统的社会结构。Tachteyev 等人。[24]通过检查 GitHub 档案中可用的自我报告的位置信息，查看 GitHub 开发人员的地理位置。Gousios 等人。[10]研究了在 GitHub 上拉请求是如何工作的。他们发现拉请求模式 o↵ers 快速转变，增加了社区参与的机会，减少了纳入捐款的时间。他们表明，相对较少的因素决定了合并请求的决定和处理请求的时间。他们还定性地研究了拉请求被拒的原因，发现技术原因只是少数。

其他研究集中在通过 GitHub api 更容易获得数据。正确的[项目提供了 GitHub api 数据的镜像。它通过监视和记录 GitHub 事件，并应用相关资源的递归依赖检索来获取数据。可以向 o✏ine.查询由莱特瑞项目提供的数据集自 2012 年开始收集数据以来，它提供了一个全面的数据集，目前正在努力检索所有数据

93

表 1 :我们研究中发现的危险总结。

危险描述

存储库不一定是一个项目。大多数项目的承诺很少。大多数项目都不活动。

IV 很大一部分存储库不是用于软件开发的。五、三分之二的项目(71.6 %的存储库)是个人项目。

VI 只有一小部分项目使用拉式请求。在那些使用它们的人中，它们的使用是非常倾斜的。VII 如果拉请求中的提交被返工(响应评论)，GitHub 只记录

提交是同行评审的结果，而不是最初的提交。VIII 大多数拉取请求显示为未合并，即使它们实际上已合并。九许多活跃的项目并没有在 GitHub 中进行所有的软件开发。

重要项目的可用历史记录。在 stan-dalone 模式下运行时，ghtorrent 还可以检索单独存储库的历史。GitHub 档案[13]提供了 GitHub 事件历史的数据集。它还通过监视 GitHub 时间线来获取数据。然而，它只包含自 2011 年开始数据收集以来的事件。此外，人们可以使用诸如 Gitmaner [27]这样的工具来提取特定存储库的事件历史。

GitHub 的普及和数据获取的便捷可能会看到更多针对 GitHub 项目和用户的研究。我们的工作提供的建议可以帮助研究人员避免与 GitHub 数据相关的常见陷阱。

3 .研究设计

本文中报告的详细分析是由我们自己对 GitHub 环境的研究激发出来的，目的是研究它是如何用于[合作的。在这项研究中，我们对 GitHub 用户进行了调查，然后进行了采访，以进一步探索我们的研究结果。调查参与者是从 GitHub 2013 年 5 月的公共活动流中选出的，他们选择了最近活跃的拥有公共电子邮件地址的用户。我们的调查是探索性的，开放式问题询问使用 GitHub 的原因，GitHub 如何支持协作、管理依赖关系和跟踪活动，以及 GitHub 的 e↵ect 开发流程。我们向 1000 名 GitHub 用户发送了调查，收到了 240 份回复(回复率为 24 %)。关于使用 GitHub 的目的，我们收到了一些不同寻常的回复。例如，响应者指出，他们使用 GitHub 用于代码托管或协同开发以外的目的。这些案例促使我们进一步分析 GitHub 存储库内容以及 GitHub 内部的协作，如第 4.2 和 4.3 节所述。

然后，我们对 GitHub 数据进行定量和定性分析，以识别和测量风险的程度和频率。我们的过程分为以下几个部分:

1 .项目元数据的定量分析。我们使用了 2014 年 1 月提供的[数据集。如第 1 节所述，右侧数据集是 GitHub 存储库、其用户及其事件(包括提交、问题和拉请求)的综合集合。为了评估项目中合并请求的真实比例，我们克隆了回复以进一步研究它们的日志和文件内容。

http://ghtorrent.org/downloads.html

2 .手动分析 434 个项目样本。当元数据的定量分析不够时，我们转向深入的手动分析。我们从光数据集中存在的 300 万个项目列表中随机抽取了 434 个项目样本(关于我们对项目的定义，参见第 4 节中的危险 I)。这个样本大小提供了 95 %的置信水平和 5 %的置信区间。

4 .结果

通过我们的混合方法研究，我们确定了分析 GitHub 数据时需要注意的九个细节。表 1 总结了风险。对于每一种危险，我们都提供定量证据来支持它，并在适当的情况下提供定性证据。本节详细描述了每一种危险，并提供了定量和/或定性证据来显示其严重性。我们还就如何避免每一种危险提供建议。

危险 I :储存库不一定是一个项目。

一般来说，拉请求作为分布式开发模型，尤其是由 GitHub 实现的拉请求，形成了一种在分布式软件开发上进行协作的新方法。在基于拉的开发模型中，潜在贡献者无法写入项目的主存储库。相反，这些贡献者分叉(克隆)存储库，并使他们的更改彼此独立。当一组更改准备提交给主存储库时，它们会创建一个拉请求，该请求指定一个本地分支与主存储库中的分支合并。然后，项目核心团队的一名成员(目标存储库的提交者)负责检查这些更改，并将它们拉向项目的主分支。如果更改被认为不令人满意(例如，作为代码审查的结果)，可能会请求更多的更改；在这种情况下，贡献者需要用新的提交更新他们的本地分支。

由于这种流行的开发模式，我们可以将存储库分为两种类型:基本存储库(不是分叉的存储库)和分叉存储库。分叉存储库中的活动独立于其关联的基本存储库进行记录。当提交并通过 GitHub 拉请求被拉进另一个存储库时，此提交不会出现在接收方存储库的历史记录中；它只出现在组件起源的存储库中。因此，独立于其分叉存储库来衡量一个回收库的活动将忽略作为单个项目一部分的所有回收库的活动。

例如，Ruby on Rails 项目有 8，327 个叉子(8，275 个叉子是直接从它的底部位置制造的，其余的都是叉子)。在 Rails git 日志中的 50k 条约定中，GitHub 报告只有 34k 条约定发生在 Rails 基本存储库中

(rails/rails)，其余的 16k 条约定源于其分支，另外还有 11k 条约定在分支中，但没有被正确传送到基本存储库中。例如，提交 fd1c515d4dac...在 Rails 的 git 存储库中找到。然而，根据 GitHub，它是在阿鲁纳格/铁轨上创建的，而不是在铁轨/铁轨上创建的。GitHub 记录了这个提交通过 id 为 13502 的拉请求被合并到一个/多个轨道中。

在本文的剩余部分中，为了正确地说明软件开发团队的所有活动，我们将基本存储库及其分支的所有活动聚合在一起。因此，我们使用术语“项目”来指代基本存储库及其分支，并继续使用术语“存储库”来表示 GitHub 存储库(基本存储库或分支)。

GitHub 的 680 万个公共存储库中，300 万个(44 %)是基本存储库，只有. 600 万个至少被分叉一次；因此，这些基本资料库代表了 300 万个 di↵erent 项目。对于至少有一个叉子的基本存储库，它们的叉子数量是高度倾斜的: 80 %的存储库只有一个叉子，最多有 3 个叉子的存储库占 94 %。然而，有一些存储库被严重分叉: 4111 个基本存储库被分叉至少 100 次。最分叉的报告是章鱼/汤匙刀，这是一个 GitHub 管理的存储库，供用户测试分叉是如何工作的。危险避免策略:要分析 GitHub 上托管的项目，必须考虑基本存储库和所有相关分叉存储库中的活动。

4.1 关于项目活动

GitHub 中的活动主要反映在提交中。因此，我们可以使用两个 di↵erent 代理来衡量一个项目的活动:通过提交的次数和提交的时间。

危险二:大多数项目几乎没有承诺。

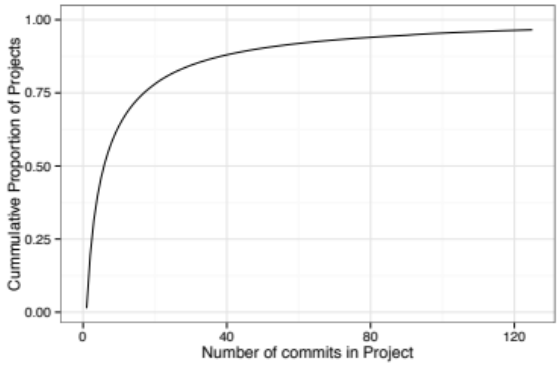
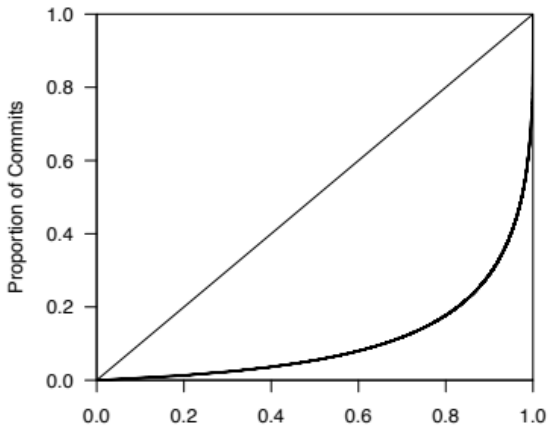


图 1 :具有给定提交次数的项目的累积比率。大多数项目很少提交。每个项目的提交次数中位数是 6，90 %的项目提交次数少于 50 次。

GitHub 回购位于 https://github.com/rails/rails.的 http://rubyonrails.org

洛伦兹曲线



项目比例

图 2: Lorenz 曲线显示，很少有项目占提交的大部分。

我们测量了每个项目的提交活动(也就是说，给定项目的所有存储库中所有提交的联合)。图 1 显示了累积分布，这种分布非常倾斜，提交的中值数量仅为 6，最大数量为 427，650。

尽管有大量项目几乎没有什么活动，但在 GitHub 中，最活跃的项目占了提交的大部分。这显示在图 2 中的 Lorenz 曲线中，该曲线描述了项目群体中提交的不平等。最活跃的 2.5 %项目的提交数量与其余 97.5 %的项目相同。

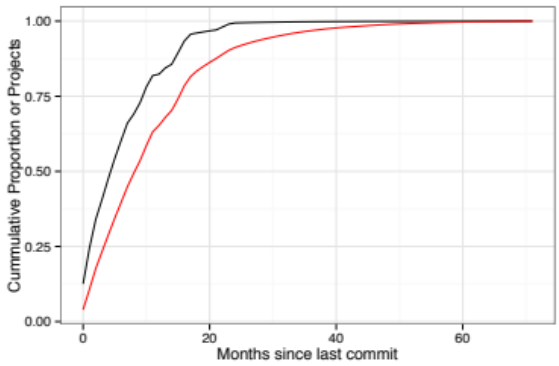


图 3 :自 2014 年 1 月 9 日以来的最近 n 个月中，活跃项目的累积比率。红线是过去 n 个月创建的项目的比例。在过去的六个月里，大约有 46 %的项目处于闲置状态。上个月只有 13 %的项目是活跃的，其中 1 / 3 是在此期间创建的。

危险三:大多数项目都不活动。

如果大多数项目很少提交，它们也很可能是不活动的。图 3 显示了过去 n 个月中有活动的项目的累积比例。例如，在过去 6 个月中(自 2013 年 7 月 9 日以来)，只有 54 %的项目是活跃的。然而，许多

在此期间创建了项目(GitHub 中所有项目的 34 %)。在 2013 年 7 月 9 日之前创建的 1，958，769 个项目中，只有 430，852 个(22 %)在过去 6 个月中至少有一项承诺。

我们也可以通过比较 GitHub 中第一个存储库的创建日期和项目最后一次提交的日期来衡量项目活动。这如图 4 所示。在这方面，项目活动的天数中位数是 9.9 天。32 %的项目只活跃了一天，这表明它们被用于测试或存档目的。只有 38 %的人活跃了一个多月。然而，活跃的项目仍然活跃: 25 %的项目至少有 100 天的活动。

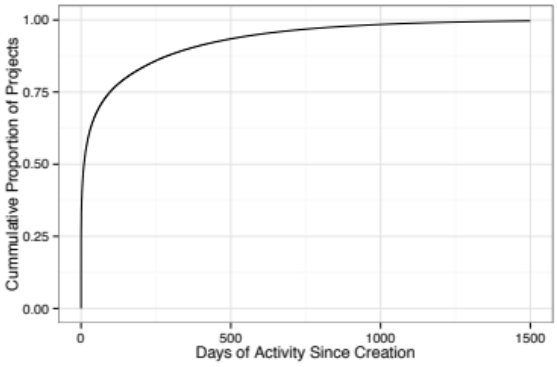


图 4 :自创建以来最近 n 天内具有活动能力的项目的累积比率。me-dian 的天数是 9.9 天，25 %的项目在 100 天或更长时间；只有 32 %的人在创建活动后不到一天就有了活动。

危险避免策略:为了识别活动项目，考虑最近提交和撤回请求的数量。

4.2 项目内容

危险四:很大一部分存储库不用于软件开发。

调查结果表明，GitHub 除了用于软件开发之外，还用于各种目的。我们的 240 名受访者中有 34 人(14 %)表示，他们使用 GitHub 存储库进行实验、托管他们的网站以及学术/班级项目。大约 10 %的应答者使用 GitHub 专门用于存储。

存储库的用途不能从项目元数据中可靠和自动地识别出来。我们使用 434 个随机选择的存储库来确定 GitHub 存储库是用于软件开发还是用于其他目的。我们审查了每一个回复的描述和文件，并给它分配了一个适当的标签来标记其内容，例如“软件库”或“类项目”。接下来，我们将标签分类为专用类别。如果存储库的内容是用于构建任何类型工具的文件，我们将它们定义为“软件开发”。这种类型的使用包括存储库、plu-gins、gems、框架、附加组件等。“实验”是一类包含示例、演示、示例、测试代码和教程示例的存储库。网站和博客被归类为“网络”，班级和研究项目被归类为“学术”。“存储”类别包括包含配置文件的存储库(包括“.”文件)或

表 2 :手动检查每种使用类型的存储库数量。这些类别是相互排斥的。

软件开发 275 (63.4 %)、实验 53 (12.2 %)、存储 36 (8.3 %)、学术 31 (7.1 %)、Web 25 (5.8%) 不再可访问 11 (2.5 %)、空 3 (0.7 %)

其他供个人使用的文件和文件，如演示幻灯片、简历等。出现 404 错误的存储库被标记为“不再可访问”。仅包含许可证文件、gitween 文件、自述文件或根本没有文件的存储库被归入“空”类别。表 2 显示了 434 个存储库的类别和分布情况。

尤其是，网络已经成为 GitHub 的一个重要用途。GitHub 允许 it 用户在其服务器上免费托管网站。使用此服务的存储库通常以其名义包括 github.io 或 github.io。有 73，745 个项目有这样的名字，这表明这

项免费服务很受欢迎。危险避免策略:当选择要分析的项目时，人们不应该仅仅依赖于他们的存储库中的文件类型来识别软件开发项目。研究人员应该查看描述和自述文件，以确保该项目符合他们的研究需求。

4.3 关于参与项目的用户

危险五:三分之二的项目(71.6 %的储存库)是个人项目。

我们的调查询问受访者，他们主要是为了与他人合作还是个人使用 GitHub。240 名受访者中有 90 人(38 %)回答说，他们主要是为了自己的项目使用 GitHub，而不是为了与他人合作。这种反应是调查 GitHub 项目中有多少协作和社会互动的动力因素。

在 git 中，提交记录了作者和提交者。提交者是有权访问存储库的人。在 GitHub 中，只有 2.9 %的提交者不是提交者。我们可以通过计算项目所有仓库中 di↵erent 委员会的数量来评估项目是否是个人项目。

每个项目的提交者数量非常不平衡: 67 %的项目只有一个提交者，87 %的项目有两个或更少的提交者，93 %的项目有三个或更少的提交者。正如预期的那样，存储库的提交者比项目少: 72 %有一个提交者，91 %有 2 个或更少，95 %有 3 个或更少。作者的比例是一样的。我们手动样本中提交者的数量相似: 65 %的人只提交一个提交者，83 %的人提交两个或更少，90 %的人提交三个或更少。

这些结果表明，尽管 GitHub 致力于社会编码，但它主持的大多数项目只由一个人使用。很可能很大一部分只有一个委托人的项目是为了实验或储存目的。详情见 http://pages.github.com/。

避免危险策略:为了避免个人项目，应该考虑承诺者的数量。

4.4 拉取请求

承诺 I: GitHub 提供了一个有价值的数据来源，用于研究拉请求和它们引用的提交形式的代码审查。

GitHub 使“叉和拉”开发模式变得更加流行，但是拉请求并不是 GitHub 独有的；事实上，git 包括 git-request-pull 实用程序，它在命令行中提供相同的功能。GitHub 和其他代码托管网站通过整合代码审查、讨论和问题，极大地改善了这一过程，因此 e↵ectively 降低了临时投稿的门槛。组合、分叉和拉动请求创建了一个新的开发模型，在这个模型中，变更被推给项目维护者，并在被集成之前由社区进行代码审查。

危险六:只有一小部分项目使用拉请求。在那些使用它们的人中，它们的使用是非常倾斜的。在 GitHub 中，拉请求的使用并不普遍。拉请求只在开发者之间有用，因此，在个人项目中不存在(67 %的项目，见危险五)。在代表实际协作项目的 260 万个项目中(至少 2 个提交者)，只有 268，853 个(10 %)至少使用了一次拉请求模型来合并提交；剩下的 240 万个项目将在共享存储库模型中专门使用 GitHub (没有传入的拉请求)，所有开发人员都被授予提交访问权限。此外，拉请求在项目之间的分布是高度倾斜的，如图 5 所示。每个项目的请求数量中位数是 2 (44.7 %的项目只有 1 个，95 %的项目有 25 个或更少)。

尽管如此，仅在 2013 年，就有一些项目，如盖亚电话应用框架和家酿包管理器收到了超过 5000 个拉请求。事实上，大量的项目(···1700)在 2013 年收到了 100 多个拉请求，这使得样本足够大，可以为许多研究问题提供具有统计意义的结果。

危险避免策略:当研究 GitHub 上的代码审查过程时，在选择适当的项目时，必须考虑拉请求的数量。

4 . 4 . 1 拉动请求作为代码审查机制

GitHub 拉取请求包含一个分支(本地或其他存储库中)，核心团队成员应该从该分支中拉取提交。GitHub 会自动发现要合并的提交，并将它们显示在请求中。通过排除故障，拉请求被提交到基础(用 git 的说法是“上游”)存储库进行审查。有两种类型的审查意见:

讨论:对拉动请求的总体内容的评论。感兴趣的各方就整个拉动请求的适用性进行技术讨论。

代码审查:对代码特定部分的评论。评审者对 di↵的承诺做了记录，通常是技术性的，以确定潜在的改进。

任何 GitHub 用户都可以参与这两种类型的审查。检查的结果是，拉取请求可以用新的提交更新，或者拉取请求可以被拒绝——要么是冗余的，要么是无趣的，要么是重复的。拉取请求被拒绝的确切原因没有记录下来，但是可以从评论中推断出来。

在更新的情况下，贡献者在分叉存储库中创建新的提交，并且在更改被推送到要合并的分支后，GitHub 将自动更新请求中的提交。然后可以在刷新的提交上重复代码审查。在我们的 434 个项目数据集中，17 %的拉请求在评论(讨论或代码审查)后收到更新。解释这个结果时必须小心，因为许多评论，尤其是在讨论部分，仅仅是对贡献者工作的感谢，而不是正确的代码审查。

关于拉请求的讨论通常很简短；80 %的请求不到 3 条评论(代码重新查看和讨论)。此外，代码审查的参与者数量在 0 到 19 之间，80 %的拉请求参与者少于 2 人。在 80 %的请求中，每次同行评审检查的提交数量不到 4 个。这些数字与《[法典审查》23、22、2]中的其他工作相当，这表明同行审查过程可能有更基本的基础有待探索。如果考虑到以下缺点，GitHub 数据可能会成为同行评审的一个非常好的定量数据来源，这是由于各个项目存储库之间的异构化。

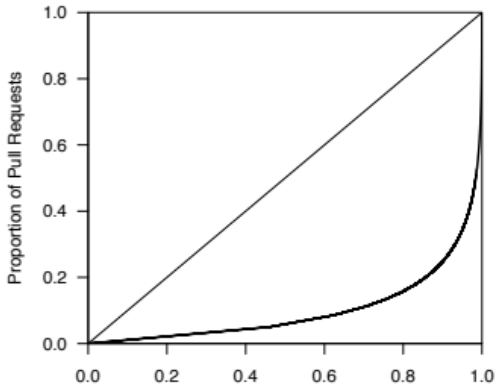
需要注意的是，拉请求中的代码审查在许多情况下是隐式的，因此不可见。例如，许多拉请求(在我们的 434 个项目示例中占 46 %)在它们仍被合并时没有收到代码注释和讨论。通常可以安全地预期，进行合并的开发人员在合并请求之前确实检查了该请求(除非项目策略是接受任何请求而不进行审查)。

危险七:如果拉请求中的提交被修改(响应评论)，GitHub 只记录同行评审的结果，而不是最初的提交。

使用 GitHub 数据进行同行重新查看研究的另一个缺点是，被查看的提交集可能不容易观察到，并且可能需要进一步处理才能恢复它们。在项目中，在提交集与主存储库合并之前，通常需要提交挤压(将所有 di↵erent 提交合并为单个提交)。尽管 GitHub 确实记录了中间提交，但它不会通过 api 报告它们作为拉请求的一部分。此外，如果源存储库被删除，原始提交将被删除。这意味着在分析时，研究人员只能观察最新的提交，这是审查过程的结果。危险避免策略:为了对代码审查中涉及的全部提交进行分析，研究人员不能依赖 GitHub 报告的提交。

危险八:大多数拉请求看起来没有被合并，即使它们实际上被合并了。

当代码审查完成并且拉动请求被认为令人满意时，拉动请求可以被合并。git 和 GitHub 的多功能性至少支持三种合并策略:



项目比例

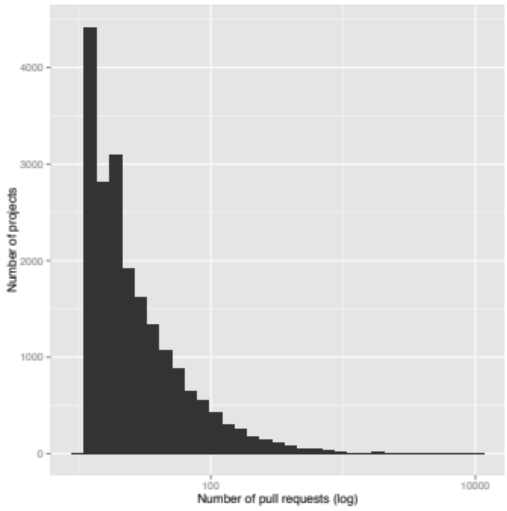


图 5 :每个项目的拉请求数的 Lorenz 曲线(左)和相应的直方图(右)。前 1.6 %的项目使用总请求量的 50 %。这些地块仅包括至少有一个拉动请求的项目。

通过 GitHub 工具，使用合并按钮通过合并主存储库分支和

拉请求分支。这种合并策略的一个变体是挑选，其中只有从拉请求分支中手工选择的提交被合并到主分支。

通过在拉取请求和主存储库分支之间创建文本补丁，并应用于海量分支。这也称为承诺挤压。

以上 di↵er 提出的合并策略包含了预先提供的历史(提交顺序)和作者信息。具体来说，通过 git 或 GitHub 的合并会保留完整的历史信息——除非是只保留作者身份的精选案例。基于补丁的合并不维护作者身份或历史记录。

此外，GitHub 只能通过其拉请求合并工具来检测和报告合并。因此，如果一个项目的策略是只使用 git 合并，所有的拉重请求都将在 GitHub 中被记录为未合并。然而，在实践中，大多数项目使用 gitHub 和 Git 合并策略的组合。

为了简化请求和问题的关闭，GitHub 提供了一种通过提交日志内容关闭请求和问题的方法。例如，如果提交日志包含字符串修复# 321 和 321 是一个请求或问题，那么这个请求或问题就结束了。修复是可以

使用的九个关键词之一。例如，home - brew / homebrew 项目已经打开了 13，164 个拉请求，关闭了 12，966 个，但是只有 129 个合并了。然而，其日志显示，6，947 个请求(占总数的 48 %)和 2，013 个问题(19 %)已经从提交日志中关闭。这表明，至少在一些项目中，人们不能依赖 GitHub 的拉请求的合并属性。

为了识别在 GitHub 之外合并的合并拉取请求，我们基于 GitHub 倡导的惯例开发了一套启发式方法。以下是最重要的(关于这些启发式的完整描述和评估，见[10)。

对于整个列表，请访问 https://help.github.com/的文章/通过提交消息解决问题。

H1 拉取请求中至少有一个提交出现在目标项目的主分支中。

H2 A 提交使用请求日志关闭请求(例如，如果

提交的日志包括一个结束关键字，见上文)，该提交出现在项目的主分支中。这意味着拉请求限制被压缩到一个提交中，这个提交被合并了。

H3 最后 3 个讨论之一(按出现顺序)

注释包含提交唯一标识符，这个组件出现在项目的主分支中，相应的注释可以通过以下正则表达式匹配:

(？: merg |应用|拉|推|集成) (？ :英|意？编辑)

H4 关闭请求前的最新注释与上面的正则表达式匹配。

在 GitHub 中，2，552，868 个请求中有 1，145，099 个请求被报告为合并请求，占请求总数的 44 %。在 434 个项目样本中，只有 37 %的拉请求使用 GitHub 规则进行了合并。通过应用上述启发式，额外的 42% (H1: 32%，H2: 1%，H3: 5%，H4: 4%)的拉请求被识别为合并请求，而 19 %的拉请求不能被分类。在其他作品《[10》中，我们使用了一个精心挑选的 297 个项目样本，这些项目严重依赖于拉请求，其中 65 %的拉请求与 GitHub 设施合并，而试探法确定另外 19% (H1: 7%，H2: 1%，H3: 3%，H4: 7%)被合并。在另一个数据集《[12》中，我们包括了近 1000 个使用拉请求的项目，58 %的拉请求使用 GitHub 的设施进行合并，而 18 %的拉请求被识别为未合并。剩余的 24 %使用启发式算法被识别为合并(H1: 11%，H2: 3%，H3: 3%，H4: 7%)。

上面提出的启发式算法并不完整，即它们可能无法识别所有合并的拉取请求，也无法发出声音，即它们可能导致误报(尤其是 H4)。在《[10》的其他作品中，我们手动检查了 350 个未被识别为合并的拉请求，发现其中 65 个实际上合并了。这意味着合并请求的实际百分比可能更高。然而，事实再次证明，只有一小部分合并是通过 GitHub 报告的，而启发式可以改进合并检测

在某些情况下是戏剧性的。

危险避免策略:不要依赖 GitHub 的合并状态，而是考虑在分析合并的拉取请求时使用如上所述的启发式方法来改进合并检测。

4 . 4 . 2 拉动请求作为问题解决机制

承诺二:开发人员之间的相互联系、拉拽请求、问题和提交提供了软件开发活动的全面视图。

GitHub 上的问题和拉取请求是双重的；对于每个打开的拉取请求，都会自动打开一个问题。还可以将限制附加到问题上，以将其转换为拉请求(尽管使用外部工具)。拉请求的问题部分用于跟踪讨论内容。鼓励开发人员在提交消息或问题评论中引用问题或请求，而 GitHub 会自动提取这些引用并将其作为讨论流程的一部分。此外，问题和拉动请求都可以链接到特定于存储库的英里石，这有助于项目跟踪进度。

问题和拉请求如此紧密地结合在一起，这为非常详细地研究开发人员活动打开了一扇机会之窗。例如，研究人员可以通过修改源代码、重新查看代码和最终整合修复程序，从报告阶段跟踪问题的解决。由于用户行为总是由 a↵ect 发出和拉动请求，人们也可以对特定类型的活动中用户群的形成进行投资，这将揭示出新兴的用户组织(团队或阶层)。此外，起诉、拉请求和提交之间的相互联系创造了一个复杂的行动网络，可以利用社交网络技术来分析，发现有趣的合作模式。

尽管有大量相互关联的数据，但仍有两个不足之处。首先，如果项目之间的记录是一致的，那么用于问题跟踪回复的存储库挖掘就大大增强了。GitHub 的问题跟踪程序只需要文本描述就可以打开问题。发布属性注释(例如 a↵ected 版本、严重性级别)被授权重新定位特定标签。这意味着不能在项目中统一检查目标的特性。其次，在 GitHub 中，只有一小部分(12 %)的存储库在 2013 年活跃时同时使用拉请求和问题。许多有趣的存储库，尤其是迁移到 GitHub 的存储库，都有一个外部问题数据库。

4.5 关于非 GitHub 基础设施的使用

危险九:许多活跃的项目并没有在 GitHub 中进行所有的软件开发。

阿迪教要回答的问题是 GitHub 中的数据是否代表了一个开发项目的大多数(如果不是全部)可见活动。换句话说，GitHub 中的项目在其过程中是否使用了其他形式的协作?

调查答复中有迹象表明，项目活动在 GitHub 以外的地方进行。正如一位受访者所说:

“任何严肃的项目都必须有一些独立的基础设施——邮件列表、论坛、irc

渠道和他们的档案，建立农场等。[...]因此，尽管 GitHub 和所有其他项目主机都用于协作，但它们不是也不可能是一个完整的解决方案。"

这促使我们研究存储库是否在 GitHub 上承载项目代码和其他内容，但在其他地方执行开发和协作活动。

有几种方法可以评估这一点。其中之一是确定所有提交者和作者是否都是 GitHub 中的用户。如果提交者不是 GitHub 用户，则 GitHub 会将电子邮件地址记录为提交者，而不是 GitHub 用户。在 GitHub 中，提交者或提交者中有 23 %不是 GitHub 用户。这一结果的可能原因是来自非用户的一些 git 操作已经在 GitHub 之外被合并，并且由于设置镜像来跟踪 GitHub 之外的其他存储库中的活动而加剧了这种情况。

镜像是托管在另一个存储库中的代码的副本。在某些情况下，镜像项目清楚地表明 GitHub 不用于提交代码。例如，postgres-xc/postgres-xc 项目在其描述“社交 Postgres-XC GIT 存储库的镜像”中声明。请注意，这只是一面*镜子* -我们不接受 github 上的拉重任务……”。尽管如此，它还是由 15 个不同的存储库组成。

我们发现了许多镜像存储库。GitHub 本身已经建立了许多流行项目的镜像，这些项目占了 91 个 GitHub 存储库。通常，存储库的描述表明它是否是镜像。例如，存储库 abishekk92 / voipmonitor 的描述为“https://voipmonitor 处 SVN 报告的镜像”。svn.sourceforge.net/...".描述还可以表明镜子是自动的，

并记录它的更新频率(例如，“位于 http://llvm.org/git/clang.的官方金属仓库镜子每小时更新一次。”)中。的不区分大小写的正则表达式镜像。*repo|git 镜像找到了 1，851 个项目(12，709 个存储库)，作为 GitHub 以外其他存储库的镜像。我们检查了其中的 100 个储存库，发现它们都是外部镜像。我们从 SourceForge 存储库和 Bitbucket (一个竞争的 git 托管存储库)中识别出许多镜像。我们在表 3 中总结了这些结果。

这些结果的含义是，项目开发的至少一部分发生在 GitHub，但不一定全部发生。

外部存储库 GitHub 镜像中的开发意味着项目的一些成员正在使用 GitHub 用于两个目的之一: (1)开发他们的工作，然后将其提交给外部存储库；例如，位于 kgene/linux-samsung 的 Linux-Samsung 项目(根据 GitHub 的说法，该项目没有叉子，本身也不是叉子)定期向 Linux 内核提交(我们在 Linus Torvald 的存储库中观察到 123 个提交，这些提交源自这里)。(2)独立于原始开发团队，为 di↵erent 的目的开发原始项目的定制；在这一类别中，我们找到了多个用于开发内核变体的存储库，比如三星 Galaxy S 系列手机的 2 . 6 . 35 内核，或者 Dropad A8T 和类似产品的 2.6.35.7 版。https://github.com/mirrors

我们目前跟踪 Linux 频道中提交的所有来源: hydraladder.turingmachine.org

ninety-nine

表 3 : GitHub 上托管的被标记为镜像的存储库。GitHub 承载来自许多来源的镜像，包括源伪造和位桶。底部显示顶部的子集。正则表达式不区分大小写。

设置已使用的常规表达式编号项目编号回购

的镜子。*回购|git 镜子，1，851 12，709

子集

位于 Sourceforge Sourceforge | SF \。净 117 511

位于位桶位桶 91 249

从颠覆回复\ W (svn |颠覆) \ W 622 4966

从水银柱\W (水银柱|汞柱) \ W 113 590

来自 CVS 回复\ WVS \ W55 212

有趣的是，一些镜像来自使用其他版本控制系统的存储库，如 Mercurial、Superson 或 cvs。这意味着，在某些情况下，贡献者更喜欢 git 而不是其他版本控制系统来完成他们的日常工作，但是这需要进一步的研究来证实。同样，许多项目使用自己的缺陷跟踪系统来处理问题。例如，GitHub 中最活跃的项目之一 Mozilla 的 Gaia (mozilla-b2g/gaia)已经禁用了 GitHub 中的问题，并希望用户在 bugzilla.mozilla.org 提交问题。

我们进行了一项小型调查，要求回复者告诉我们，他们是使用 GitHub 的工具还是外部工具来完成一系列任务，例如打开和合并拉请求、跟踪问题，还是用于沟通。我们通过电子邮件向 100 名 GitHub 用户发送了在线问卷。其中，27 个国家作出了答复(答复率为 27 %)。尽管 52 %的人说他们使用 GitHub 打开拉请求，60 %的人说他们使用该网站接受和合并代码更改，但只有 24 %的人说他们使用 GitHub 进行代

码审查。32 %的人说他们使用外部工具进行评论。这进一步验证了对于许多项目来说，所有软件开发活动都不会发生在 GitHub 内部。

避免危险策略:避免那些拥有大量未注册 GitHub 用户的提交者的项目，以及那些在其描述中明确声明他们是镜像者的项目。

5 .对有效性的威胁

我们的研究对有效性有几个威胁。这项探索性调查的参与者来自有偏见和自我选择的人群，人数相对较少。虽然它促使我们更详细地调查危险，但我们不能从中得出进一步的结论。我们对 434 个项目的手工探索说明了 GitHub 的各种用途，但是我们没有将我们的结果推广到其他项目。

这项研究的可靠性取决于正确数据集的可靠性。ghtorrent 是一种从 GitHubapi 收集数据的 best-e↵ort 方法。以前的作品《[》分析了为什么光不能完全复制 GitHub。检测 GitHub 之外合并的拉请求的启发式算法的准确性在[12 中有详细说明。

为了缓解这些威胁，我们为我们的研究提供了一个复制包。正确的数据是公开的。我们手动分析的结果以及本工作中使用的其他数据和脚本可在补充包中单独获得。

http://ghtorrent.org/downloads.html

这篇论文的复制包可在 http : / / turingmachine . org / GitMiningParlils 2014 上获得。

6 .讨论和结论

挖掘的数据所讲述的故事并不总是整个故事。这是在评估从项目档案中挖掘的数据的质量和完整性的研究中发现的，但也是在将挖掘的数据与[1]的定性证据进行比较的罕见情况下。在这个实证研究中，我们开始批判性地审视来自 GitHub 的公开数据，并评估它是否适合作为软件工程研究的数据源。这些数据可以很容易地用于报告几个项目属性。如果研究人员试图看到编程语言的使用趋势、所构建的工具类型、贡献的数量和规模等，公开可用的数据可以提供关于 GitHub 环境描述性特征的可靠信息。

使用 GitHub 合成信息来得出关于更抽象的结构的结论，尽管需要考虑一些因素。我们展示了关于存储库活动和内容以及开发和协作实践的假设如何受到挑战的证据。我们建议对使用 GitHub 数据进行研究感兴趣的研究人员首先评估其适用性，然后将能够真正提供信息的数据用于回答他们的研究问题。

对任何不加区别地使用 GitHub 数据的研究来说，最大的威胁可能是对个人使用的偏见。尽管许多存储库正在 GitHub 上积极开发，但其中大多数只是个人的、不活跃的存储库。因此，当使用 GitHub 数据时，需要考虑的最重要问题之一是研究需要什么类型的存储库，然后相应地对合适的存储库进行采样。

虽然我们认为有必要根据 GitHub 项目的目的，对其识别和自动分类进行研究，但我们提出了一条经验法则。根据我们自己的经验，识别活跃软件开发项目的最佳方法是考虑在最近一段时间内，在限制和拉请求数量之间保持良好平衡，并且提交者和作者数量大于 2 的项目。问题的数量也可以作为一个指标，但并非所有活动项目都使用 GitHub 的问题跟踪程序，例如几个 Mozilla 项目。异常值，尤其是每个提交者提交的数量非常大的异常值，指向自动机器人。https://github.com/mozilla

当研究任何特定的项目时，研究人员需要记住，项目中可能存在其他储存库——有些储存库致力于一个共同的目标，有些可能是独立的版本，永远不会有回报。根据我们的工作，我们认为确定存储库是否与另一个存储库积极合作的简单方法可能是确定提交是否已经从一个方向飞到另一个方向，但这一策略需要进一步验证。

GitHub 是一种非凡的资源。它继续以加速的速度增长，用户正在寻找利用它的创新方法。尽管如此，软件开发在 GitHub 的基础设施中正在开放中蓬勃发展，并将继续成为我软件工程研究的一个有吸引力的来源。

7。参考

[1]阿兰达和威立雅。错误的秘密生活:克服软件仓库中的错误和遗漏。进行中。第 31 国际号的。糖膏剂《软件工程》，第 298 - 308 页，2009 年。

[2]巴克利和伯德。现代代码审查的期望、结果和挑战。进行中。int。糖膏剂软的。《工程项目》，ICSE '13，第 712 - 721 页，2013 年。

[3]巴赫曼、伯德、拉赫曼、德万布和伯恩斯坦。缺少的链接:错误和错误修复提交。进行中。第 18 届 ACM SIGSOFT 国际软件工程基础研讨会，第 97 - 106 页，2010 年。

[4] A·比格尔、J·博施和 M. - A 斯托里。社交网络符合软件开发:来自 github、msdn、堆栈交换和顶级编码器的观点。软件，IEEE，30 (1) : 52 - 66，2013。

[·伯德、巴赫曼、昂、Du↵y、伯恩斯坦等人。公平和平衡？:错误修复数据集中的偏差。进行中。软件工程基础研讨会，第 121 - 130 页，2009 年。

[6]伯德、里格比、巴尔、汉密尔顿、德国和德万布。采矿的承诺和危险。在挖掘软件仓库中，(MSR'09)，第 1 - 10 页。IEEE，2009 年。

[7]达比什、斯图尔特、蔡和赫伯。github 中的社会编码:开放软件仓库中的透明度和协作。进行中。糖膏剂关于计算机支持的合作工作，第 1277 - 1286 页，2012 年。

[8] K·芬利。Github 在流行程度上已经超过了 sourceforge 和谷歌代码。http://readwrite.com/ 2011 / 06 / 02 / github 已通过源代码锻造，2011 年。

[9]戈西奥斯。GHTorrent 数据集和工具套件。《第十届采矿软件储存库会议记录》，MSR '13，第 233 - 236 页，2013 年。

[10]古西奥斯、平茨格和范·德乌森。基于拉的软件开发模式探索。ICSE '14 :继续。第 36 国际米兰。糖膏剂《软件工程》，2014 年 6 月。出现。

[11]戈西奥斯和斯宾利斯。GitHub 的数据来自消防软管。在 MSR '12 中:继续。第九次工作会议的报告。《采矿软件储存库》，第 12 - 21 页，2012 年 6 月。

[12]戈西奥斯和扎伊曼。用于拉请求研究的数据集。在提交给 MSR '14 -数据轨道中。

[13] I .格里高利。github 档案。http://www.githubarchive.org/, 2012。

[14]豪生和克鲁斯顿。矿源锻造的危险和陷阱。进行中。国际的。采矿软件储存库研讨会，第 7 - 11 页，2004 年。

[15]卡尔利亚姆瓦库、达米安、歌手和德国人。以代码为中心的协作观点:来自 github 的证据。技术报告 DCS-352-IR，维多利亚大学，2014 年 2 月。

[16] J·马洛、L·达比什和 J·赫布莱布。在线同行制作中的印象形成: github 中的活动轨迹和个人简介。进行中。糖膏剂计算机支持的合作工作，第 117 - 128 页，2013 年。

[17]麦当劳和高金斯。github 上开源软件的性能和参与度。CHI'13《计算系统中人的因素扩展摘要》，第 139 - 144 页。ACM，2013 年。

[18]阮氏、亚当斯和哈桑。错误修复数据集偏差的案例研究。《逆向工程》(WCRE)，2010 年第 17 届工作会议，第 259 - 268 页。IEEE，2010 年。

[19] R. Pham，L. Singer，O. Liskin，F. Figueira Filho 和 K. Schneider。在社交编码网站上创建对测试文化的共同理解。进行中。int。糖膏剂软的。英格。，ICSE '13，第 112 - 121 页，2013 年。

[20]拉赫曼、波内特、赫雷兹和德万布。缺陷预测中的样本大小与偏差。进行中。2013 年第 9 届软件工程基础联席会议，第 147 - 157 页，2013 年。

[21]雷恩和盖尔。评估开源软件项目数据的质量和数量。《第一次开放源码系统国际会议记录》(OSS 2005)，第 29 - 36 页，2005 年。

[22]普里比和伯德。融合当代软件同行评审实践。进行中。2013 年第 9 届软件工程基础联席会议的报告，ESEC/FSE 2013，第 202 - 212 页，2013 年。

[23]彼得·里奇比、德裔美国人和斯托里。开源软件同行评审实践: Apache 服务器的案例研究。进行中。第 30 个整数。糖膏剂《软件工程》，ICSE '08，第 541 - 550 页，2008 年。

[24] Y·塔克提耶夫和 A·希尔兹。通过 github 调查开源软件的地理位置。http://takhteyev.org/papers/·Takhteyev-Hilts-2010.pdf，2010 年。

[25]图恩、比塞尚德、卢德和江 L。github 中社会编码的网络结构。在第 17 届欧洲软件维护和重组会议(CSMR)上，2013 年，第 323 - 326 页。

[26]蔡京特、达比什和赫布莱布。社交媒体和开源项目的成功。进行中。计算机支持的合作工作伙伴，第 223 - 226 页，2012 年。

[27]瓦格纳、杰根森和萨玛。轨道网络:轨道上的红宝石和相关项目的图形数据集。进行中。第十届国际米兰。工作。糖膏剂《挖掘软件仓库》，第 229 - 232 页，2013 年。

[·维斯。源代码锻造上开源项目的定量分析。进行中。第一个整数。糖膏剂开源系统(OSS 2005)，第 140 - 147 页，2005 年。

one hundred and one