开源软件项目 DevOps 实践的基准测试方案

Jos´e Manuel S´anchez Ruiz, 研究员，塞维利亚大学，Francisco Jos´e Dom´ınguez Mayo, 副教授，塞维利亚大学，Xavier Oriol, 研究员，加泰罗尼亚理工大学，Jos´e Francisco Crespo, 研究员，加泰罗尼亚理工大学，David Benavides,正教授，塞维利亚大学，and Ernest Teniente, 正教授，加泰罗尼亚理工大学

arXiv:2304.14790v1 [cs.SE] 28 Apr 2023

摘要—过去几年中，开源软件（OSS）项目日益普及，越来越多的组织开始依赖这些项目。随着项目规模不断扩大，对更高质量的需求也随之增加。事实证明，DevOps 实践有助于提升质量和组织绩效。DORA 基准测试报告提供了有用的信息，可用于比较不同组织的 DevOps 实践绩效，但其关注的是持续部署以及向生产环境交付，而开源软件项目关注的是代码的持续发布及其对第三方的影响。DORA 报告提到，开源软件项目的应用日益广泛，在业界也得到普遍使用，但从未将其用于衡量开源软件项目的绩效级别。本研究表明，DORA 基准无法直接应用于开源软件项目，并提出了用于 DevOps 实践基准测试的度量指标；这是首项调整 DORA 指标并将其应用于开源软件项目情境的研究。本研究分析了 DORA 基准测试的局限性，并对其度量指标进行调整，提出适用于开源软件项目的新指标。本研究提出的开源软件项目基准测试指标包括用于衡量吞吐量的发布频率和已发布变更的前置时间，以及用于评估稳定性的代码修复时间和缺陷问题率。与通过人工调查问卷收集数据的 DORA 报告不同，我们的方案使用自行开发的 Performance-Tracker 工具自动收集数据，该工具从公开的 GitHub 仓库中提取信息。这降低了基于调查问卷的数据收集所带来的风险。本研究还通过将该基准应用于四个热门开源软件项目，验证了其可行性：Angular、Kubernetes、TensorFlow 和 VS Code。开源软件项目的基准测试结果显示，各项目的度量值各不相同；总体而言，Angular 的表现最佳。此外，我们还提出了一些挑战，涉及本研究的主题及未来工作，以拓展相关知识与研究发现。总体而言，本研究的发现有助于推动开源软件项目的后续研究，并增进对 DevOps 实践在开源软件项目中所发挥作用的理解，同时指出相关挑战。

索引词　条目—开源软件、DevOps、软件开发、软件交付性能、基准测试

g

1 引言

开PEN源软件（开源软件）项目近年来日益普及。参与这些项目的成员数量不断增加，而且仍在持续增长 [1]。根据一项与开放源代码促进会合作开展的近期研究， [1], 80%的组织在过去一年中增加了对开源软件的使用，这表明人们对这类软件的依赖日益加深。开源软件项目社区遵循项目管理员提出的指导方针，以便在可管理的环境中开发高质量的软件。随着社区不断壮大，这有时会变得更加困难。

随着开源软件项目规模不断扩大、复杂度不断提高，它们对软件开发质量的要求也越来越高。DevOps [2] 是一种在协作文化中拓展敏捷实践的方法，旨在改进软件开发与交付流程。许多研究证明，采用 DevOps 实践能够带来诸多益处，例如提升质量和组织绩效 [3]。尽管 DevOps 各种定义的核心都是开发与运维之间的互动，但 DevOps 也有不同的定义 [4]。然而，这也给组织带来了挑战 [5]。因此，世界各地的组织都致力于通过基准测试 和实施 DevOps 实践来提升自身绩效 [6]。

在此语境中，性能由质量和时间定义。快速生产产品/服务并将其交付给用户至关重要。团队、组织或项目社群的性能可以在一个时间段内进行衡量。数据可能随时间段而变化，因此需要进行恰当的背景化。可以使用不同时间段的数据来呈现性能的演变。

文献中有许多关于基准测试的定义。分析各种定义后，Anand 和 Kodali [7] 将基准测试描述为：“[.....] 持续分析战略、职能、流程、产品或服务、绩效等，并通过适当的数据收集方法，在同类最佳组织内部或之间进行比较，旨在评估组织当前的标准，并据此通过实施变更进行自我改进，以达到或超越这些标准”。

• *Jos´e Manuel S´anchez Ruiz*、*Francisco Jos´e Dom´ınguez Mayo* 和 *David Benavides* 任职于信息语言与系统系、计算机工程高等技术学院，地址：*Avda. ReinaMercedes s/n. 41012*塞维利亚，西班牙。电子邮箱：{*jsanchez7,fjdominguez,benavides*}*@us.es*

• *Xavier Oriol*、*Jos´e Francisco Crespo* 和 *Ernest Teniente* 均供职于信息服务与系统工程系（*ESSI*）及 *inLab FIB*，*UPC* 北校区，*Jordi Girona 1-3*，*08034* 巴塞罗那。电子邮箱：{*jose.francisco.crespo-sanjusto,xavier.oriol,ernest.teniente*}*@upc.edu M*

稿件收到日期为*2005*年*4*月*19*日；修订日期为*8*月*26*日，*2015*年

绩效等，并通过适当的数据收集方法，在同类最佳组织内部或之间进行比较，旨在评估组织当前的标准，并据此通过实施变更进行自我改进，以达到或超越这些标准”。根据 ASQ [1]，的定义，基准测试是将产品、服务和流程与那些在一个或多个运营方面被公认为领先者的组织所提供的产品、服务和流程进行衡量的过程。因此，即使业务领域不同或客户群体不同，基准测试也能提供必要的洞见，帮助了解一个组织与类似组织相比处于什么水平。

在软件组织中，基准是一项可衡量性能和效率的指标，用于评估软件组织实现其目标和宗旨的情况，并将其性能与同一行业内的其他软件组织进行比较。

自2011年以来，Google/DORA 每年都会发布一份关于 IT 行业 DevOps 现状的基准测试报告（最初与 Puppet Labs 合作），以了解哪些实践和能力会影响软件开发中的质量和组织绩效。DORA 基准测试报告提出了四种不同的度量指标。其中两项，部署频率和 变更前置时间，旨在评估项目的吞吐量；另外两项，服务恢复时间和 变更失败率，则用于评估项目的稳定性。直观而言，部署频率衡量组织将代码发布到生产环境的频率；变更前置时间衡量代码从提交到在生产环境中成功运行所需的时间；服务恢复时间是指发生影响最终用户的服务事件或缺陷后，恢复服务所需的时间；变更失败率则是导致服务质量下降的已发布变更所占的百分比。DORA 报告提供了来自各个团队和组织的关于软件开发中 DevOps 实践的有用信息。然而，这些报告基于 IT 专业人员填写的调查问卷，没有考虑其他方法所具备的客观性；其他方法能够从软件交付物中自动提取信息。

此外，DORA 基准旨在收集由 IT 行业开发的项目的结果，因此无法提供足够的信息来评估 DevOps 实践对开源软件项目的影响。

开源软件项目不同于典型的软件项目，因为不存在用于交付软件即服务的集中式运营服务。相反，它们会发布可由第三方实例化的代码。因此，它们具有一些独特的功能，例如：

1）去中心化的协作团队，采用分布式设计与构建流程，且缺乏明确的项目规划和任务分配；2）依赖同行评审和社区生产；3）采用开源许可证；4）由社区驱动开发，并在某些情况下获得公司的支持；5）透明、包容；以及6）大量无偿贡献者和地理分布式开发 [8]。

1. https://asq.org/quality-resources/benchmarking

这些特点与 DORA 报告中分析的特点不同。因此，DORA 指标可能无法直接用于评估开源软件项目中的 DevOps 实践，必须针对这些特定情境重新思考并加以调整。

除了前述观点之外，还值得注意的是，有一些实证证据（例如调查研究、访谈或案例研究）可以帮助我们了解 DevOps 在软件开发中的益处与挑战 [9], [10], [11]。然而，关于 DevOps 原则和实践的有效性，以及用于衡量它们的度量指标，现有证据仍存在不足。特别是，缺乏能够提供定量数据的研究，用以说明实施 DevOps 的组织所观察到的益处 [12]。

本文提出了一组受 DORA 启发、用于开源软件项目 DevOps 实践的基准测试度量指标。我们首先分析 DORA 基准测试的局限性，并提出四项度量指标，以替代 DORA 面向开源软件项目的指标：发布频率和已发布变更的前置时间用于衡量吞吐量；修复时间和 缺陷问题率用于评估稳定性。发布频率衡量代码发布的频率；已发布变更的前置时间衡量已提交代码发布所需的时间；代码修复时间是修复缺陷所需的时间；缺陷问题率则是为修复漏洞或代码而提出的变更所占的比例。

我们还将该基准应用于四个热门开源软件项目 Angular、Kubernetes、TensorFlow 和 VS Code，展示了该基准的价值。借此，我们能够评估 DevOps 对这些特定开源软件项目的影响。由于代码发布属于部署和交付的范畴，因此可以使用 DORA 报告中的性能数据，将受评估开源软件项目的性能与 DORA 报告中各组织的平均性能进行比较。

与依赖人工调查收集数据的 DORA 报告不同，我们提出的方案使用自主开发的 Performance-Tracker 工具，自动从 GitHub 公共代码仓库中收集数据。这得益于 GitHub 开放的 API，该接口可提供发布、问题和提交等信息。我们的 Performance-Tracker 工具基于 Google Cloud Platform Four Keys 项目的代码[2] ，可在我们的 GitHub 代码仓库中找到[3]。因此，对开源软件项目进行基准测试时，数据收集可以自动化；而 DORA 报告中的数据收集则是人工完成的。

总结而言，本文的主要贡献如下：

- **我们指出，DORA 基准无法直接用于对开源软件项目的 DevOps 实践进行基准测试**。为此，我们需要厘清所用术语，重点关注代码的持续发布及其对第三方的影响，而非持续部署和交付至生产环境。

- **我们提出新的度量指标，用于评估开源软件项目中的 DevOps 实践**。这些指标是通过对 DORA 指标进行调整而得出的

2. https://github.com/GoogleCloudPlatform/fourkeys 3. https://github.com/diverso-lab/performance-tracker

以适应与开源软件开发相关的特定问题。

- **我们展示了四个热门开源软件项目的绩效等级**。这使我们能够比较这些项目的绩效，并建立识别对绩效产生影响的 DevOps 实践所需的机制。

- **我们提出了一种基于项目代码仓库自动测量的基准**，从而降低了基于调查收集数据的风险。我们提供了一种名为 Performance-Tracker 的新工具，它连接到开源软件项目代码仓库，并对开源软件项目应用我们的性能基准。

- **我们提出了一系列挑战**，这些挑战可在未来应对，以拓展本研究所提供的知识。

我们的研究发现为未来研究提供了启示，并有助于更好地理解 DevOps 背景下的开源软件项目。本研究独具特色，因为它提出了此前未曾报告过的开源软件项目绩效客观度量方法。所有材料和数据均已公开，以支持开放科学。

本文其余部分结构如下：第 2 节说明了 DORA 基准指标在应用于开源软件项目的特定情境时的局限性。第 3 节定义了适应开源软件项目特性的基准度量指标。第 4 节通过开展基准测试可行性研究，验证开源软件基准度量指标的适用性。第 5 节探讨了本研究中发现的不足与挑战。第 6 节概述了有关开源软件项目绩效评估及基准测试方法的研究，例如《DevOps 状况报告》。最后，第 6 节给出结论并提出未来研究方向。

2 DORA 基准与开源软件项目

鉴于开源软件项目与 DORA 报告中组织开发软件的方式在性质上存在差异，我们需要审查最新 DORA 报告中使用的术语，以准确描述每项度量指标，并使其适用于开源软件项目的情境。因此，虽然《2022 年 DevOps 状态报告》中的度量指标采用更广泛的组织视角，考量持续部署、持续交付代码及其对服务的影响，但开源软件项目的软件开发则更关注发布给第三方的软件所产生的影响。

因此，DORA 报告中使用的“将代码部署到生产环境”和“在生产环境中运行”等术语，并不适用于部署不被视为软件开发流程一部分的情况。相反，我们建议对开源软件项目的发布代码进行度量。发布代码是指在代码仓库中公开提供代码更改或新增代码，使第三方能够使用、测试，并可能将其纳入其他项目。此外，DORA 报告将事件称为“服务事件”或“影响用户的缺陷”，而我们认为，在开源软件项目中，使用“缺陷问题”等术语来指代报告应用程序漏洞的问题更为恰当，因为这些项目并未提供特定服务，也没有涉及用户。

另一个需要考虑的重要因素是度量指标所处的环境，这对于评估性能和稳定性至关重要。这引出了一个问题：在为开源软件项目定义度量指标时，区分开发（Dev）和运维（Ops）环境是否有意义，以及是否需要明确“发布”“部署”和“交付”等概念。此外，还需要界定 DORA 报告中所评估组织的开源软件项目范围，因为本研究考察的四个开源软件项目并不部署其代码，而只是将其发布给预期用户。

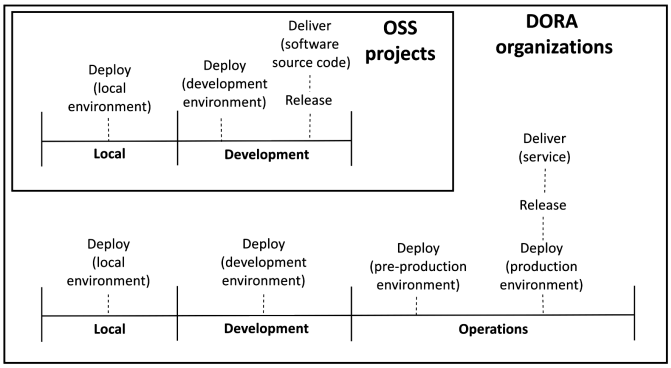

图 1. 发布、部署与交付

如图 1 所示，开源软件项目的代码会经过不同的环境，DORA 报告中涵盖的项目也是如此。实际上，我们可以区分出三种不同的环境：

- **本地环境：** 这是开发人员在自己的计算机上编写和测试代码的地方。他们可能会部署软件进行测试，但这些部署并非面向最终用户，而只是供在该本地环境中工作的开发人员使用。
- **开发环境：** 这是软件在面向更广泛的受众发布之前进行测试和完善的地方。通常会在此环境中部署软件进行测试，并制作一个发布版本。在开源软件项目中，此环境中的发布版本被视为一次交付，因为发布后最终用户即可访问该软件。另一方面，DORA 报告中包含的项目在其流程中还涉及另一个环境。

- **运维环境：** 这是部署软件并使最终用户能够使用软件的地方。通常涉及两个环境：预生产环境和生产环境。预生产环境会模拟生产环境，目的是检测软件的新版本是否按预期运行，或是否会导致任何错误。部署到运维环境通常由运维团队或专门的部署团队完成。软件一旦部署到生产环境，就意味着已经完成发布，并交付了服务。

我们还可以观察到“部署”“发布”和“交付”之间的一些差异。为了避免混淆，我们对它们进行定义。

- **部署：** 我们可以将该术语定义为实例化代码新版本的操作，包括配置、

在服务器上测试和安装软件。

- **发布：** 指使某个版本的代码可供用户使用这一行为。
- **交付：** 这是向用户提供软件的过程，其中包括部署和发布软件。

为了说明 DORA 基准与本研究所制定基准之间的差异，我们从效度、信度、准确度和精确度方面评估这两种基准度量指标。

- **精确度：** 这一概念指的是测量结果，而不是度量指标本身。它指的是用于区分不同项目性能的测量阈值范围有多宽。在我们的方案中，就精确度而言，若采用与 DORA 基准相同的量表，那么开源软件项目基准的精确度也将与 DORA 基准相同。

- **准确度：** 准确度指通过度量指标获得的测量结果与我们要测量对象的真实值有多接近。我们的开源软件项目基准度量指标基于自动化工具 Performance-Tracker 得出结果，其中的度量指标均有明确定义，不依赖于解释。相比之下，DORA 基准基于人们对调查问卷中问题的理解；不同的人可能会以不同方式理解同一个问题，并给出不同的答案。

- **信度：** 度量指标的信度是指其结果是否可复现且一致。我们针对开源软件项目提出的基准度量指标采用自动测量机制（Performance-Tracker 工具），因此能够复现结果。这些结果是客观的，因为它们由自动化工具测量得出。相比之下，DORA 基准依赖人工测量和调查问卷，参与调查的人可能会对同一问题给出不同答案。

- **效度：** 度量指标的效度是指它是否确实测量了其预期测量的内容。换言之，它指的是测量度量指标后得到的结果，是否根据我们的测量目标得到了恰当的解读。效度并非度量指标本身的属性，而是我们对其结果所作解读的属性。就效度而言，两种基准度量指标都需要进一步开展专家验证工作。

接下来，根据上述概念，我们将更精确、准确地定义针对开源软件项目所提出基准中的度量指标。此外，我们确保计算这些度量指标所需的信息能够从 GitHub 仓库中自动获取。

3 度量指标定义

我们为所提出的每项用于对开源软件项目进行基准测试的度量指标制定了公式。对于每项度量指标，我们都提供了说明、公式，以及研究中考虑的因素，包括我们测试过的各种测量方法。此外，对于涉及频率或时间的度量指标，我们还提供标准差，以便更全面地了解开源软件项目的行为。我们会在相关受影响的度量指标中进一步说明这一点。

接下来介绍我们针对开源软件项目中的 DevOps 实践提出的度量指标，如表1所示。顶行回顾了最新 DORA 报告中为业界主导的软件项目定义的度量指标。底行则说明了我们针对开源软件项目的提案，即根据开源软件环境的特点对 DORA 基准进行调整。因此，我们提议使用发布频率 和已发布变更的前置时间 来衡量开源软件项目的吞吐量，并使用代码修复时间和缺陷问题率 来评估其稳定性。

需要特别指出的是，要使用我们的提案衡量项目的绩效水平，必须先确定度量指标的测量周期。

3.1 发布频率

部署频率是 DORA 的一项吞吐量指标，用于衡量组织向生产环境部署代码或向最终用户发布软件的频率。

鉴于对该度量指标的描述及其与 DORA 所定义的部署频率指标之间的差异，我们可以应用以下公式（1）来衡量这些项目的发布频率。

$$f_r = \frac{\sum_{i=0}^{n_r} t_{r_{i+1}} - t_{r_i}}{N} \tag{1}$$

设 t_{r_i}、 $t_{r_{i+1}}$ 为两次连续发布的日期， n_r为该期间内的发布次数， $N = n_r - 1$为发布次数减 1， f_r 为发布频率。日期可以用时间戳表示，以便进行日期运算。

公式（1）计算特定期间内连续发布之间的平均时间。也可以用最后一次发布的日期与第一次发布的日期之差，除以发布次数减 1，得到相同的结果。不过，我们使用公式（1），因为它能让我们获得一组数据，其中包含每两次连续发布之间的时间。这些数据使我们能够计算发布间隔的标准差，从而衡量该期间内发布间隔的变化程度。

标准差反映了发布的规律性。在这种情况下，标准差较高表示发布不规律，因此两次连续发布之间相隔的天数可能会有明显差异。相反，标准差较低表示发布频率更规律（例如每周发布一次）。在这种情况下，基于时间的发布策略 [14] 的标准差将接近零，因为每次发布之间的时间都相同。我们将公式定义如下:

$$d_{fr} = \sqrt{\frac{\sum_{i=0}^{N} t_{rd_i} - m_{rd}}{N}} \tag{2}$$

设 t_{rd_i} 为两次连续发布之间的时间（此前已计算为发布频率 $t_{ri+1} - t_{ri}$），$m_{rd} = f_r$ 为连续发布之间时间的平均值（即发布频率）， $N = n_r - 1$ 为

表1 指标定义：2022 年 DevOps 状态报告与开源软件项目

	部署频率	变更前置时间	服务恢复时间	变更失败率
2022 状态 DevOps 状况报告 报告	对于主要应用程序 **或您负责的服务**，您或-ganization**部署代码到生产环境或发布**它到最终用户？ [13]	对于主要应用程序 **或服务**你负责的，你的前置时间是多久变更（即，需要多长时间才能从代码提交到代码成功-完全**运行于生产-环境**？ [13]	对于主要应用程序 **或你负责的**服务，通常需要多长时间才能**恢复服务** **当发生服务事件或影响用户的缺陷** **时**（例如，意外的服务中断——年龄或服务受损）？ [13]	对于主要应用程序 **或服务**你负责的，什么百分比的**变更** **进入生产环境或发布** **对用户造成影响会导致性能下降** **服务**（例如，导致服务受损或服务中断）并随后需进行补救（例如，需要热修复、回滚、向前修复或打补丁）？ [13]
	发布频率	**已发布变更的前置时间** **变更**	**代码修复时间**	**缺陷问题率**
OSS 项目	对于主要应用程序 你负责的工作，频率如何贵组织**重新** **租赁代码** 给最终用户?	对于主要应用程序 你从事的工作是什么？你的变更前置时间（即，需要多长时间才能从代码提交到代码成功**发布** **在项目仓库**中）？	对于主要应用程序 你负责的工作通常需要多长时间 通常需要多长时间来重新 **配对代码，当出现漏洞** **或影响** **用户发生**（例如，漏洞问题）？	对于主要应用程序 您所负责的应用程序，有百分之多少的 变更比例**请求** **意味着错误修复或** **代码修复**（例如，应用程序故障）？

发布次数（减 1），而 d_{fr} 为发布频率的标准差。

关于本文定义的发布频率公式，我们发现有两种例外情况，无法计算发布频率：

- **该期间内仅有一次发布：**由于没有后续发布，我们无法衡量首次发布之后需要多长时间才能再次发布。在这种情况下，近似的结果是估计发布频率等于该期间的时长。

- **该期间没有发布：**由于该期间完全没有发布，因此无法计算发布频率。此情况只能确认发布频率长于该期间。

在本研究中，我们认为，每当指定代码仓库中发布一项 GitHub 发布版本时，即视为进行了一次发布。我们必须考虑到，发布版本具有创建日期和发布日期。纳入测量范围的发布版本，是指在该时间段内正式发布的版本。图2展示了一个示例场景。在此示例中，发布版本按以下方式进行计算：

- **发布 1** 由于在该期间发布（但在该期间之外创建），因此会被纳入。
- **发布 2** 由于在该期间发布（且也在该期间创建），因此会被纳入。
- **发布 3** 由于在该期间之外发布（但在该期间创建），因此不会被纳入。

3.2 变更前置时间

变更前置时间是 DORA 用于衡量吞吐量的一项度量指标。对于这项度量指标，有三个不同的概念分别指代三种不同的度量指标，其名称容易造成混淆。因此，下面将介绍这些概念及其区别。图 3 展示了这三种度量指标之间的关系：

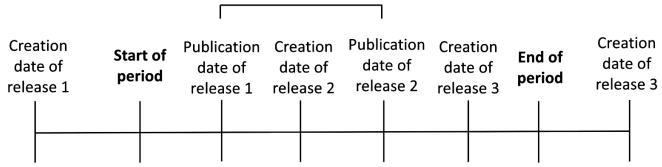

图 2. 发布频率示例

- **前置时间** 用于衡量从提出变更请求到该变更部署至生产环境所需的平均时间。
- **周期时间** 衡量从团队开始处理一项变更到该变更部署至生产环境之间的平均时间。
- **变更前置时间** 衡量从代码提交到代码成功在生产环境中运行所需的平均时间。

因此，这三项度量指标代表了开发与运维流程中的三种不同时间度量。DORA 项目将变更前置时间列为其度量指标之一。

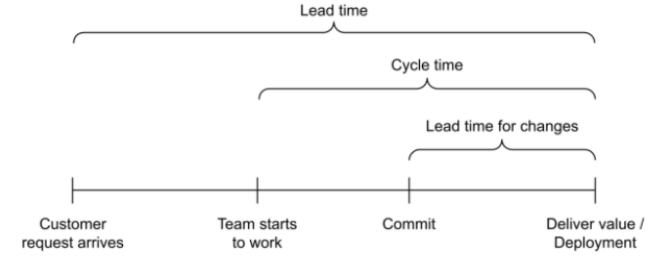

图 3. 前置时间、周期时间和变更前置时间

在这项度量指标中，我们测量代码从提交到成功发布所需的时间，并将其命名为

“已发布变更的前置时间”度量指标。要计算这项度量指标，我们测量某个问题的最后一次提交与包含该问题的部署之间的时间。公式如下：

$$l_t = \frac{\sum_{i=0}^{n_s} t_r - t_{c_i}}{n_s} \tag{3}$$

即 t_r 变更发布的日期（包含该变更的代码版本发布的日期）、 t_{c_i} 问题的最后一次提交日期、 n_s 在给定时期内导致代码变更的请求变更数量，以及 l_r 已发布变更的前置时间。

如果在测量期间没有对代码进行任何变更，则由于缺少数据，无法计算此度量指标。

在此度量指标中，标准差表示不同问题的代码提交与发布之间的时间差异。标准差高，说明从代码提交到发布每个问题所需的时间可能差异很大，有些问题所需时间比其他问题长。标准差低，则表示每个问题从代码提交到发布所需的时间相近，变化不大。我们提出如下公式：

$$d_l = \sqrt{\frac{\sum_{i=0}^{n_s} t_{ld_i} - m_{ld}}{n_s}} \tag{4}$$

其中，t_{ld_i} 每项已提交变更从代码提交到代码发布所需的时间（此前计算已发布变更的前置时间时，记为 $t_r - t_{c_i}$），$m_{ld} = l$从代码提交到代码发布所需时间的平均值（即已发布变更的前置时间）， n_s 给定时期内导致代码变更的请求数量，以及 d_l 已发布变更前置时间的标准差。

由于每个开源软件项目及其社区的工作方式各不相同，已发布变更的前置时间度量指标可能难以衡量；因此，试图将该度量指标的衡量方式通用化，可能会导致结果不精确，无法反映实际情况。为了恰当地衡量这一指标，我们研究了不同的可能方法。如上所述，已发布变更的前置时间应表示从代码提交到代码发布所需的平均时间。

这一度量指标的主要难点之一，是确定在被衡量的时期内应纳入哪些问题。我们需要界定哪些问题属于某一时期。起初，我们只纳入在该时期内创建并关闭的问题。这样得出的结果会使已发布变更的前置时间始终不超过该时期的时长。这一度量指标并不准确，因为我们遗漏了在其他时期创建、但在当前时期关闭的问题，以及在当前时期创建、但在未来关闭的问题（某个问题对应的变更可能已经提交，但该问题仍处于未关闭状态，并可能在未来关闭）。图 4 展示了一个示例情形。在此示例中，发布按以下方式界定：

- **问题 1** 会被纳入第 1 期（它在该期创建并关闭）。
- **问题 2** 不会被纳入任何一期（它在第 1 期创建，但在第 2 期关闭）。
- **问题 3** 会被纳入第 2 期（它在该期创建并关闭）。

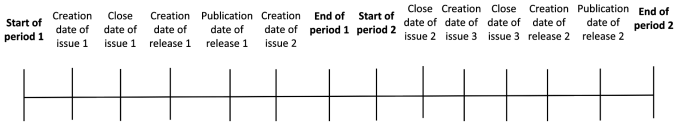

图 4. 已发布变更的前置时间示例

另一种方法是只考虑该期内提出的问题，但其中一些问题可能尚未发布，因此无法利用这些问题衡量已发布变更的前置时间。此外，这种方法也无法恰当地反映实际情况：这些变更的提交和发布可能会在未来进行，因而无法准确反映所测量的实际期间内已发布变更的前置时间（图 5）。

- **问题 1** 会被纳入第 1 期（它在第 1 期创建，在第 2 期关闭）。
- **问题 2** 会被纳入第 1 期（它在第 1 期创建并关闭）。
- **问题 3** 会被纳入第 2 期（它在该期创建并关闭，但尚未部署）。

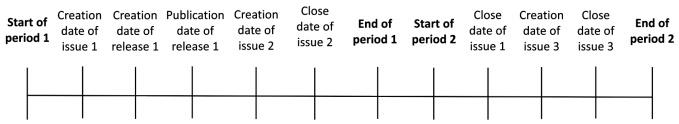

图 5. 已发布变更的前置时间示例 2

我们逐渐清楚地认识到，重点应放在最后一次提交的时间以及包含这些变更的发布的时间上。我们决定纳入该期间内发布的所有问题。为此，我们可以选取该期间发布的所有版本（无论它们是否在该期间创建），并纳入代码包含在该版本中的每个问题。很难准确知道某个版本具体包含了哪些问题，因此我们决定将该期间内最后一次提交时间介于该版本创建时间与前一个版本创建时间之间的所有问题纳入，作为一种近似。这就是该度量指标最终采用的近似方法。以下图示展示了一些示例（图 6、7、8 和 9）：

- **问题 1** 将被纳入该期间（其最后一次提交发生在发布 1 创建之前，而发布 1 则在该期间内）。
- **问题 2** 将被纳入该时段（依据其在创建发布 1 之前所做的最后一次提交；发布 1 是该时段内唯一的发布）。

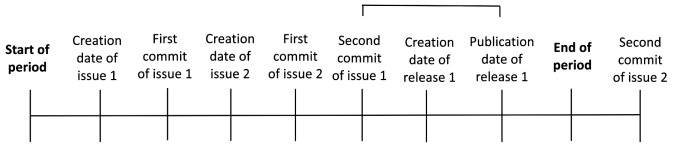

图 6. 已发布变更的前置时间示例 3

- **问题 1** 将被纳入该时段（尽管其首次提交是在创建发布 1 之前完成的，但最后一次提交是在创建发布 2 之前完成的，因此该问题被视为纳入该发布）。

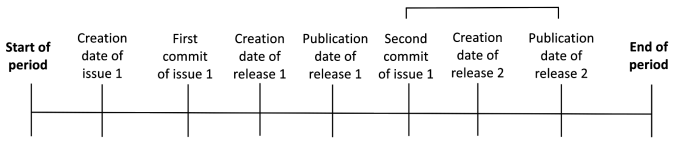

图 7. 已发布变更的前置时间示例 4

- **问题 1** 将作为发布 1 中所含代码的一部分计入该时段（其最后一次提交是在创建发布 1 之前完成的，因此属于该时段，尽管它是在发布 2 之后发布的）。
- **问题 2** 将作为发布 2 中所含代码的一部分计入该时段（其最后一次提交是在创建发布 2 之后完成的）。

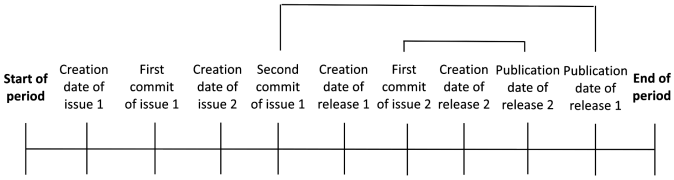

图 8. 已发布变更的前置时间示例 5

- **问题 1** 将作为以下内容的一部分纳入该期间：发布 1 中包含的代码（其最后一次提交是在发布 1 创建之前完成的，尽管该提交是在该期间之外完成的）。
- **问题 2** 不会被纳入该期间，因为它唯一的提交是在发布 1 创建之后完成的（（该期间仅包含一次发布）。

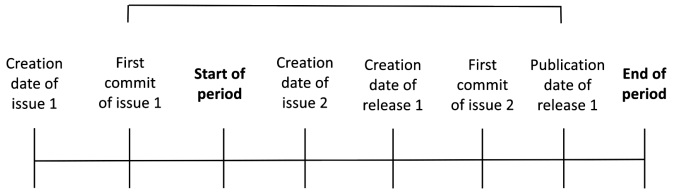

图 9. 已发布变更的前置时间示例 6

3.3 代码修复时间

在 DORA 报告中，服务恢复时间是一项用于衡量稳定性的度量指标。它衡量的是发生影响用户的漏洞或缺陷后，通常需要多长时间才能恢复服务。

然而，在开源软件项目中，这项度量指标并不直接适用，因为不存在集中式运营服务来部署软件，也没有团队负责维护服务质量。相反，开源软件项目会发布可供第三方使用的软件版本。因此，开源软件项目中的服务恢复时间度量指标调整为代码修复时间，指的是发布包含修复缺陷的变更的新版本代码所需的时间。

为了衡量这一度量指标，我们尝试根据从 GitHub 获取的信息制定一种近似计算方法。修复漏洞所需的时间，是从提出缺陷问题到修复该漏洞的变更随新版本代码发布之间经过的时间。我们定义如下公式：

$$m_r = \frac{\sum_{i=0}^{n_e} t_r - t_{e_i}}{n_e} \tag{5}$$

其中， t_r 是修复漏洞的变更发布的时刻， t_{e_i} 是团队收到漏洞通知的时刻（报告该漏洞的问题被创建）， n_e 是用户在一段时间内报告且导致代码变更的漏洞数量， m_r 是代码修复时间。

如果在测量期间没有对代码进行任何修复漏洞的变更，由于缺少数据，便无法计算这一度量指标。

在这一度量指标中，标准差反映了解决漏洞所需时间的变化。标准差高表示变化幅度大，即解决不同漏洞所需的时间各不相同。标准差低则表示解决不同漏洞所需时间的变化较小。我们提出以下公式：

$$d_m = \sqrt{\frac{\sum_{i=0}^{n_e} t_{md_i} - m_{md}}{n_e}} \tag{6}$$

设 t_{md_i} 为解决一个漏洞所需的时间（此前计算代码修复时间时记为 $t_r - t_{e_i}$）， $m_{md} = m$ 为解决漏洞所需时间的平均值（即代码修复时间）， n_e 为代码中为解决漏洞而引入的变更数量， d_m 为代码修复时间的标准差。

我们可以注意到，代码修复时间这一度量指标与已发布变更的前置时间相似。实际上，它是该度量指标的一个版本，其中仅纳入与漏洞相关的问题，所测量的时间为问题创建到修复漏洞的变更发布之间的时间。图 10 展示了一个示例场景。在此示例中，缺陷问题按如下方式定义：

- **缺陷问题 1** 将作为发布 1 中代码的一部分计入该时段（其最后一次提交发生在发布 1 创建之前，但在该时段之外）。
- **问题 2** 不会计入该时间段，因为它唯一的提交是在发布 1 创建之后完成的（该时间段内唯一纳入的发布）。

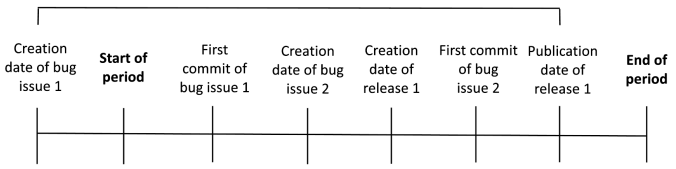

图 10. 代码修复时间示例

3.4 缺陷问题率

变更失败率是 DORA 报告中用于衡量稳定性的度量指标。它表示对生产环境进行的变更或交付给用户的发布中，导致服务质量下降的百分比。与服务恢复时间度量指标类似，在分析交付产品的开源软件项目时，确定漏洞何时发生以及哪些变更导致了漏洞，可能颇具挑战性。

我们再次提出了一种名为“缺陷问题率”的近似度量指标，用于衡量某一时期内提出的缺陷问题数量与同期提出的问题总数之比。定义如下公式：

$$q = \frac{n_e}{n_s} \tag{7}$$

其中，n_s 表示某一时期内开启的问题数量，n_e 表示某一时期内出现的缺陷问题数量，q 表示缺陷问题率。该度量指标表示某一时期内开启的问题中，缺陷问题（标记为漏洞）所占的比例。

4 基准测试可行性研究

为了精心设计并实施本研究，以得出准确、公正且有参考价值的结果，我们遵循了有关基准测试方法的研究建议和指南 [15]。基准测试可以由以下人员开展：(1) 新方法的作者，用于证明性能提升或其他优势；(2) 有意系统比较现有方法的独立团队；(3) 或以“中立”基准测试研究的形式组织 [16], [17]。在本研究中，我们开展了一项力求公正、侧重于比较本身的基准测试研究，并比较了应用于多个开源软件项目的方法。此外，我们遵循可复现研究的最佳实践，使研究过程和数据公开可用。本研究分为以下几个阶段：

- **规划基准测试**：在此阶段，我们明确定义了基准测试的目的和范围，制定了适当的测量方法，并确定了来自开源软件项目的代表性数据集。我们还选择了相关参数值和软件版本，为研究的设计与实施提供了指导。具体而言：1）我们**开发了一种测量工具**，用于采集样本中每个开源软件项目的关键度量指标，详见第 3 节

2）我们定义了**选择开源软件项目代码仓库**的标准，并为每种实践和工具撰写了初步描述，以准确说明其相关实践。这项工作与度量指标的定义同步进行，且本文所有作者都审阅了初稿。

- **开展并报告基准测试：**在此阶段，我们使用提出的定量度量指标评估开源软件项目的性能。我们分析了这些项目使用的实践和工具，以深入了解其性能。为解读结果，我们将 DORA 基准测试绩效等级作为参照。这有助于我们将开源软件项目划分为高、中或低绩效级别，并为我们的建议提供背景。总体而言，我们从开源软件项目社区的视角提出了建议。

最后，我们总结了最相关的结果，并讨论了研究效度可能面临的主要威胁，以及研究的局限性。

4.1 基准测试规划

4.1.1 开发测量工具

我们基于 Google Cloud Platform Four Keys 项目[4]开发了一款名为 Performance-Tracker 的软件工具。该工具可测量开源软件项目的四项度量指标，并根据本文定义的公式计算这些指标。现有的软件度量工具可以自动从代码仓库中收集数据，例如 Google Cloud Platform Four Keys 和 Apache DevLake，但必须将它们集成到软件组织的基础设施中，包括 GIT 工作流、问题管理系统、代码仓库和 CI/CD 自动化工具。

其次，GIT 工作流的多样性给数据收集带来了挑战 [18]，，因为数据采集取决于具体的 GIT 工作流。

4.1.2 基准测试开源软件项目的选择

在本研究中，我们选取了四个热门的开源软件项目，旨在对其进行测量和比较。为纳入研究，我们设定了具体标准：项目的贡献者人数必须超过 1,500，创建时间早于 2016 年，问题数超过 20,000，且拉取请求数超过 10,000。我们使用 GitHub API，从 GitHub 这一公开的软件开发平台收集数据，时间段为 2022年7月1日至2022年12月1日，共六个月；收集的信息包括该时间段内项目的发布、提交及其他数据。我们选择六个月的数据收集时长，因为这足以获取必要数据，从而准确呈现各项目的结果。我们没有采用更长的测量周期，以免请求数量过多，因为 GitHub API 对每位用户每小时的请求数限制为 5,000 次。

根据前述标准，选定的开源软件项目包括 Angular[5]，、Kubernetes[6]，、TensorFlow[7] 和 VS

4. https://github.com/GoogleCloudPlatform/fourkeys 5. https://github.com/angular/angular 6. https://github.com/kubernetes/kubernetes 7. https://github.com/tensorflow/tensorflow

Code[8]。我们对每个项目进行了分析和研究，以更好地了解其运作方式。表2列出了各项目的介绍及相关数据。

- **Angular：** Angular 是一个在全球范围内用于开发前端和单页应用程序的开源软件框架。它采用 TypeScript 编写，并由 Google 提供支持，于 2016 年首次发布。在 Angular 代码仓库中，可以知道新版本代码何时发布。这些标签用于定义代码的不同版本。鉴于 Angular 代码仓库所采用的工作流程，新版本发布时会包含自上个版本发布以来所做的变更。提交拉取请求后，授权用户会检查变更并进行合并，将变更引入 Angular 主分支。我们将被视为缺陷问题的问题作为系统中检测到漏洞的时刻的表征。这些问题可描述为涉及漏洞的问题（通常标记为漏洞）。通常，每次对代码进行变更时都会加入测试，以检查一切是否按预期运行；在许多情况下，为开源软件项目做贡献也要求加入测试。这降低了提交会导致应用程序出错的代码的风险。

- **Kubernetes：** Kubernetes（通常简称为 K8s）由 Google 于 2014年创建，是一个用于自动化服务管理的平台。它允许用户自动部署系统并管理容器化应用程序，还有其他用途。代码发布由 GitHub 上的机器人自动完成。发布内容包含格式为 vXX.YY.ZZ 的代码版本。问题由社区创建，通常由机器人进行分类，并分配有助于对问题进行分类的必要标签。许多提交与拉取请求相关联，但没有关联到问题，因此若不查看拉取请求，就很难知道某个问题对应的提交数量。对于缺陷问题，我们可以查找用于标识它们的 kind/bug 标签。

- **TensorFlow：** TensorFlow 是一个用于跨多种任务进行数据流处理和可微编程的开源软件库。它是一个符号数学库，也用于神经网络等机器学习应用程序。该库由 Google Brain 团队开发，并于 2015 年首次发布。如今，它由 TensorFlow 组织维护。该代码仓库的发布流程与前面提到的代码仓库相同。不过，值得指出的是，发布频率较低，但同一天会针对不同版本的代码进行多次发布，这可能导致度量指标混乱。提交拉取请求后，授权用户会检查其中的变更，并通过合并将这些变更引入 TensorFlow 主分支。对于缺陷问题，我们可以查找用于标识它们的 type:bug 标签。

- **VS Code：** VS Code 是微软为 Windows、Linux、macOS 和 Web 开发的集成开发环境。自 2015 年首次发布以来，它一直在持续开发。该编辑器包含代码调试和 Git 集成功能以及其他功能。与之前的项目类似，VS Code 在其代码仓库中使用 GitHub 发布。不过，与其他开源软件项目不同，它不使用预发布标签。因此，每次发布都被视为面向生产环境的发布。提交拉取请求后，变更会由授权用户审核，并通过合并引入 VS Code 主分支。VS Code 代码仓库中的缺陷问题会被标记为漏洞。

4.2 运行并报告基准测试

我们使用提出的度量指标评估了开源软件（OSS）项目的性能。我们还考察了这些项目采用的实践和工具，以便更好地了解其性能。

为重现本研究的结果，我们已将本项目中使用的软件工具放在以下代码仓库中[9]。该软件为本研究创建并使用，并在 Google 云平台 Four Keys 项目的代码基础上进行了改编，以支持 OSS 基准测试度量指标。要收集数据并计算度量指标，只需下载该软件并编辑 .env 文件，其中包含一些变量，具体说明见代码仓库。

为解读结果，我们将 DORA 基准测试绩效等级作为参考。表3展示了 DORA 报告如何根据度量指标结果对各项目的绩效等级进行分类。这有助于我们将开源软件项目归类为高、中或低绩效，并为我们的建议提供背景。表4列出了每个项目的名称，以及测量各项度量指标后获得的数据，其中包括本研究涉及的四个开源软件项目在发布频率、已发布变更的前置时间和代码修复时间这三项度量指标上的标准差。缺陷问题率这一度量指标没有标准差，因为它是比率，而不是根据一组数值计算出的平均值。标准差表示数据与平均值的偏离程度。

总体而言，我们基于开源软件项目社区的视角提出了建议。下面报告本研究取得的结果。对于每个项目，我们展示所获得的度量指标值，并给出结论。

4.2.1 发布频率结果

为了提高团队的绩效等级，发布频率应随时间推移而增加（在相同时间段内发布更多版本），如 DORA 报告 [1],所示，其中高绩效团队往往具有更高的部署频率。不过，我们衡量的是两次连续发布之间的平均天数，因此度量指标值越低，发布频率越高。发布频率高固然重要，定期发布也同样重要（数据的标准差可能会影响结果）。较高的发布频率使团队能够持续获得有价值且可执行的反馈，并快速交付高质量的软件。

以下报告了各个项目的结果：

8. https://github.com/microsoft/vscode

9. https://github.com/diverso-lab/性能-trac ker

表2 截至2022年12月的开源软件项目信息

项目	贡献者	创建时间	问题	拉取请求
Angular	1,573	2014年-09-18	24,017	21,467
Kubernetes	3,195	2014年6月6日	41,028	69,494
TensorFlow	3,127	2015年11月7日	34,910	21,189
VS Code	1,655	2015年3月9日	135,667	12,899

表3 2022年DORA报告绩效级别 [13]

绩效等级	部署频率	Lead Time For 变更	时间至恢复服务	变更故障 Rate
Low	每月一次与每月一次 ev-每6个月	介于一个月和6个月之间	介于一周和一个月之间	46-60%
中	介于每周一次和每月	介于一周和一个月	介于一天和一周之间	16-30%
High	按需（多次部署每天）	一天之间和一周	不到一天	0–15%

表4 开源软件项目结果与 DORA 绩效级别：平均值 / 标准差

项目	发布频率（平均值 / 标准差）	Lead Time For Re-已发布的变更	代码修复时间	缺陷问题率
Angular	6.09 天 / 3.28 天	3.41 天 / 2.36 天	79.22 天 / 94.11 天	3.84 %
Kubernetes	5.87 天 / 10.75 天	14.31 天 / 10.24 天	96.24 天 / 114.67 天	48.45 %
TensorFlow	10.91 天 / 24.55 天	38.72 天 / 27.99 天	176.09 天 / 215.90 天	45.10 %
VS Code	10.93 天 / 6.84 天	9.62 天 / 8.33 天	38.40 天 / 92.71 天	18.96 %

• **Angular：**如表4所示，Angular 项目的发布频率为 6.09 天，这意味着 Angular 距上一次发布后，平均不到一周就会再次发布。该项目拥有完善的组织架构，且有一支规模庞大的团队参与开发。在这种情况下，较高的发布次数（在给定时期内发布了 31 次）决定了其发布频率，也使团队能够在每次发布中引入较少的变更。此外，该项目的标准差较低，因此发布的时间间隔相对规律。使用 CircleCI 等 CI/CD 工具和自动化测试，有助于团队减少技术债务。

• **Kubernetes：** 与其他项目不同，Kubernetes 发布频率较低，但在很多情况下，他们会在同一天发布多个版本，从而产生多次发布。这在一定程度上得益于自动化机器人负责发布，而不是指派团队成员来完成这项工作，因为该团队规模不如 Angular 团队大。他们的标准差也更大，这意味着其发布不像 Angular 那样规律。他们使用 GolangCI-lint，这是一款 Go 代码检查器，可分析代码并帮助识别代码异味，同时也会进行自动化测试。不过，发布频率更加不规律，导致某些版本中引入的变更数量更多。

• **TensorFlow：** 发布频率为 10.91 天，TensorFlow 在测量期间仅发布了 8 次。不过，该期间的首次发布是在九月。TensorFlow 团队发布并不频繁（有些发布间隔数月），但有时会在同一天进行多次发布。这导致某些发布引入了更多变更。该度量指标的标准差在四个项目中最大，因此发布不规律，发布之间的时间间隔也更长。他们使用 Google Cloud 服务等 CI/CD 工具，以及有助于减少技术债务的自动化测试。有时，流程中会出现瓶颈，因为团队成员较少，无法处理大量问题。

• **VS Code：** 尽管发布频率与 TensorFlow 相似，但这些项目采用的发布方式各不相同。在这段时间内，VS Code 发布次数更多，且发布节奏更均匀。因此，每次发布引入的变更更少。其标准差表明，VS Code 的发布频率并不固定，但比 Kubernetes 和 TensorFlow 等项目更规律。他们在流程中使用 Azure DevOps 服务等 CI/CD 工具以及自动化测试，从而加快流程并减少技术债务。

4.2.2 已发布变更的前置时间结果

较高的已发布变更的前置时间可能表明团队流程存在效率低下的问题。获得的结果

各项目的报告如下：

• **Angular：** 该项目的已发布变更的前置时间较短。如发布频率部分所述，与其他项目相比，每次发布中引入的变更较少。其标准差表明，引入代码变更所需的时间较为相近。针对不同类型的问题，项目提供了相应的模板，贡献者可以使用这些模板填写所需信息。任何人都可以提交问题或修改代码，但只有 Angular 团队成员可以审查这些变更。较高的发布频率使团队能够快速将变更引入生产环境。

• **Kubernetes：** Kubernetes 的已发布变更的前置时间为 14.31 天。由于发布频率不规律，某些发布中会引入大量变更。这导致数据的标准差较高。在许多情况下，会同时存在多个未解决问题，因此审查和处理这些问题需要更多时间，这可能成为流程中的瓶颈。关于如何为项目做贡献的指南涵盖了许多主题，包括提出问题、问题分类以及提交拉取请求等。此外，针对不同类型的问题，也提供了相应的问题模板。每个人都可以提交问题并对其进行分类，因此团队成员能够更容易地处理这些问题。

•**TensorFlow：** TensorFlow 的已发布变更的前置时间较长，这可能与其发布次数有关。如“发布频率”一节所述，TensorFlow 会在同一天进行多次发布，但这些发布日之间往往间隔较长（这从发布频率的标准差可以看出），因此，从某一时期内首次出现问题到最终完成发布，往往需要较长时间。这也导致每次发布中引入的变更数量较多。由于发布频率不规律，其标准差也是最高的。关于如何为该项目做出贡献的指南较为清晰，但不如其他项目详尽。其中并未包含如何创建问题的相关章节（尽管提供了问题模板）。

•**VS Code：** VS Code 拥有一个由微软支持的大型团队。这样他们便能处理大量问题。此外，还有一个机器人负责将团队成员分配给问题，从而便于管理，并促成更好的组织架构。发布频繁且比其他项目更为定期，因此每次发布引入的变更更少，标准差也更低（在代码中引入变更所需的时间更加规律）。指南非常全面且详细，涵盖许多主题。其中的问题模板使提交问题的过程更快速、更便捷。

4.2.3 代码修复时间结果

处理缺陷对于维持代码的稳定性至关重要。该度量指标衡量引入修复漏洞的变更所需的时间。在软件服务中，稳定性是首要任务，但本研究中考察的项目背景不同，因为它们是产品。

以下报告各个项目的结果：

• **Angular：** Angular 的代码修复时间远长于已发布变更的前置时间。这表明，他们引入用于解决生产环境中漏洞的变更，比引入新功能需要更多时间。正如在缺陷问题率一节中所述，原因之一可能是使用了“漏洞”标签。该期间只有少数问题被标记为漏洞，因此数据量较少。填写缺陷模板时，问题不会自动标记为漏洞。这可能导致结果与实际情况不符。正如我们所见，代码修复时间的标准差高于其他度量指标的标准差。这是因为我们衡量的是从报告漏洞的问题被创建之时起，解决该漏洞所需的时间。因此，有些缺陷问题在解决前会持续数月处于未解决状态。不过，如前所述，Angular 拥有自动化测试、指南、CI/CD 工具和一支规模庞大的团队，这些都有助于缩短代码修复时间。

• **Kubernetes:** 与 Angular 项目一样，Kubernetes 也具有较长的代码修复时间。在许多情况下，机器人会提示该项目缺乏足够的贡献者，无法妥善处理所有问题。有些问题需要数月才能得到处理，而另一些问题则因长期无人跟进而被关闭。标准差较高，数据存在较大的差异。这可能表明其组织架构不够完善，或者缺乏足够的团队成员。

• **TensorFlow：** TensorFlow 的代码修复时间最长，在本研究的四个项目中，它引入能够解决发布中漏洞的变更所需的时间最长。正如我们在发布频率一节中所解释的，TensorFlow 在九月才发布了测量期间的第一个版本，因此有很长一段时间没有发布代码（出于同样的原因，它们的已发布变更的前置时间也最长），而且需要在单次发布中引入更多变更。其标准差最高，这表明平均值并不能真正代表实际数据，因为方差确实很大。他们的团队成员创建了一张图表（使用 GitHub 项目），用来整理需要审批的变更，但似乎变更数量很多，而负责审查的团队成员并不多，因为许多变更审查都被分配给了相同的团队成员。

• **VS Code：** VS Code 的代码修复时间最短。团队成员通常会迅速处理缺陷问题。其标准差也最低，这可能表明大多数缺陷问题都能得到快速处理。自动化测试、完善的指南、CI/CD 工具以及团队中较多的成员，都有助于 VS Code 缩短代码修复时间。

4.2.4 漏洞问题率结果

正如我们所定义的，缺陷问题率是指在该时期创建的问题总数中被标记为漏洞的问题所占的百分比。缺陷问题率低，表示相较于其他类型的功能，漏洞数量较少；而缺陷问题率高则表示相反。

以下报告了各项目的结果：

- **Angular：** 尽管漏洞数量较少可能反映出良好的开发与运维流程，但事实似乎并非如此。存在一些描述漏洞的问题，但这些问题并未被标记为缺陷。尽管漏洞报告指南十分明确，但团队成员并未将报告的每个漏洞都归类为漏洞，这使得跟踪已报告的漏洞变得更加困难。另一方面，测试和 CI/CD 工具有助于减少项目中的漏洞和代码异味。

- **Kubernetes：** Kubernetes 在给定时期内的缺陷问题率最高。所有这些漏洞都带有 kind/bug 标签。这表明社区遵循指南，并恰当地使用了标签。这些指南十分明确，贡献者也会妥善遵循。

- **TensorFlow：** TensorFlow 的缺陷问题数量较高，这也表明其对所提供的标签进行了良好的运用，以便对问题进行分类。与本研究中的其他项目一样，TensorFlow 使用了许多标签，以尽可能准确地描述和分类这些问题。
- **VS Code：** VS Code 代码仓库中每天都有大量问题被创建和关闭。尽管社区使用标签对问题进行分类，但在许多情况下并无必要，因为该项目有 450 个标签，足以让用户更准确地对问题进行分类，并更真实地呈现被标记为未知漏洞的问题。

4.3 结果总结

通过分析，我们确认，在本可行性研究中提出的基准测试度量指标可应用于开源软件项目。我们将这些指标应用于四个热门开源软件项目进行基准测试，以此证明这一点，它们分别是 Angular、Kubernetes、TensorFlow 和 VS Code。

我们发现，Angular 和 VS Code 在结构组织良好、团队成员充足方面表现突出，这有助于有效地进行问题管理和解决。它们还采用频繁发布小型更新，而非较少发布大型更新的策略。VS Code 还有许多用于问题分类的标签，并使用机器人分配问题，这也是所有项目中常见的做法。

所有项目都以某种形式使用机器人，例如 Angular 和 TensorFlow 使用机器人进行代码提交，而 Kubernetes 的独特之处在于拥有专门用于代码合并和测试执行的机器人。其他项目则使用工作流自动化工具来完成集成和代码部署等任务。在问题管理方面，Angular、TensorFlow 和 VS Code 使用机器人分配用户。TensorFlow 还使用机器人创建发布版本。

Angular、TensorFlow 和 VS Code 使用 GitHub Actions 作为工作流自动化工具，而每个项目使用不同的代码集成工具：Angular 使用 CircleCI，TensorFlow 使用 Google Cloud Services，VS Code 使用 Azure Pipelines。

根据我们的分析，我们可以得出结论：在发布频率 方面，四个项目均表现出中等水平的性能。在前置时间 *For* 在发布变更方面，Angular 表现出高水平的性能，Kubernetes 和 Visual Studio 表现为中等水平，而 TensorFlow 表现为低水平。

对于代码修复时间，四个项目的性能水平均较低。至于缺陷问题率，Angular 的性能水平较高（但由于问题标签质量较差，数据并不可靠），VS Code 的水平为中，Kubernetes 和 TensorFlow 的水平较低。

至于 DevOps 实践对开源软件项目绩效的潜在影响，广泛使用机器人和工作流自动化工具（例如 Kubernetes 用于代码合并和测试执行的独特机器人），似乎可能对四个项目观察到的中等发布频率绩效产生了一定影响。

Angular 和 VS Code 结构良好的组织、充足的团队规模以及定期的小版本更新，可能分别是它们在已发布变更的前置时间方面取得高绩效和中等绩效的促成因素。VS Code 使用标签和机器人进行问题管理，也可能是其在该度量指标上取得中等绩效的因素。

四个项目的稳定性指标（代码修复时间和缺陷问题率）绩效总体较低。这可能是由于开源软件项目缺乏运维服务，以及由此导致的社区压力较低。然而，Visual Studio 使用标签和机器人进行问题管理，有助于其在 缺陷问题率 方面取得中等绩效。与此同时，Angular 中不完整的缺陷分类使缺陷跟踪变得困难。

自动化工具或有效管理等 DevOps 实践可以从吞吐量指标方面提升开源软件项目绩效。然而，开源软件项目的性质使其难以实现高水平的稳定性。

4.4 效度威胁

我们遵循 Wohlin等人提出的建议 [19] ，描述可能影响本性能基准测试研究的效度威胁：

- **结论效度威胁：** 关于结果的可复现性，我们详细说明了开展此次开源软件项目绩效评估所遵循的所有步骤，以确保其可复现性。此外，为尽量减少所采取行动中的主观性，本研究的每位作者都独立检查了这些行动。此外，关于取得的性能结果，我们使用了软件度量工具，因此，所得结果比此前 DevOps 状况报告中采用的调查问卷更加客观（代码行数比调查问卷更可靠，因为其中不涉及人为判断）。本研究所用的测量材料均已公开，因此结果可以复现。

- **构念效度威胁：** 待检查的代码仓库中的实践和工具，是参照 Puppet Labs 和 DORA 近年来发布的报告确定的。所有作者都审阅了这些信息，以便更好地了解 DevOps 实践对开源软件项目性能的影响程度。

关于所选项目样本的效度，为确保这些项目具有足够的重要性，我们根据其规模（贡献者数量、问题数量、拉取请求数量）和存续时间（至少 8 年的活动）定义了一系列特征。

本研究的主要问题之一是，由于不同代码仓库项目所采用的工作流实践存在差异，度量指标难以应用。为缓解这一问题，我们调整了四项度量指标各自的定义，确保每项度量指标既符合基准定义的范围，又足够严格，能够在每种情况下进行计算。

• **内部效度威胁：** 关于度量指标和研究所用制品设计不当所造成的影响，需要指出的是，所有调整均未超出四项度量指标定义的范围。这些调整已在所使用的测量工具软件中实现。所有作者都检查了这些调整，以确保所得结果一致。

• **外部效度威胁：** 近期的 DevOps 状态报告指出了情境在组织绩效评估中的重要性。由于所评估的项目目的和目标各不相同，可能会影响其所处情境，进而影响绩效评估方法，因此所得结果未必能够完全推广。此外，所有分析的开源软件项目都位于 GitHub 平台的代码仓库中。因此，我们承认，研究结果仅适用于这些项目所处的情境。若要推广这些结果，未来还需开展更多研究，包括重复研究，以及对其他不同性质项目的评估。尽管如此，我们认为，本研究为所探讨的问题提供了充分的依据和覆盖范围，因为在规模和生命周期方面，所评估的项目具有相同的特征。

5 差距与挑战

在我们的研究过程中，我们发现了一系列有待解决的挑战，以拓展本研究获得的知识。这些挑战涉及若干空白，例如：（G1）概念定义不够明确；（G2）需要建立并验证基准度量指标；以及（G3）在开源软件项目中，DevOps 实践缺乏标准。

5.1 G1——定义概念时缺乏清晰度

•**挑战 1. 建立通用词汇**。 用于提及概念的词汇不够明确，可能导致结果不精确。这也会使沟通变得困难，并可能在传递想法和信息时造成困惑。本研究界定了发布、部署和交付等概念，因为我们认为 DORA 报告中对这些概念的阐释不够清晰。谈及 DevOps 实践与流程时，相关概念应当明确，以便正确应用，并在避免混淆的情况下衡量项目性能。

5.2 G2 -需要验证基准度量指标

•**挑战 2. 验证基准度量指标并探索新的度量指标**。本研究提出的度量指标已应用于四个不同的开源软件项目，但尚未经过专家验证。应对这些度量指标进行验证，以确保它们能够用于获得预期结果。探索不同的选项也很重要。开发新的度量指标有助于获得更准确、且更适用于特定情境的结果。

• **挑战 3. 研究是否需要为开源软件项目设立新的基准类别**。在本研究中，我们建议使用 DORA 绩效等级，对测量四个开源软件项目的性能后所获得的结果进行基准评估。此决定基于以下假设：DORA 组织在许多情况下依赖开源软件项目，因此可以使用相同的类别对其性能进行分类。然而，测量并确定 DORA 组织与这些项目之间的关系及其依赖程度至关重要。可能需要一套新的类别来对开源软件项目进行分类。

5.3 G3 - DevOps 实践背景下缺乏标准

•**挑战 4：为开源软件项目制定标准化的 DevOps 实践**。我们遇到了一个问题：不同项目的流程各不相同，因此难以对其进行衡量。这使得开发用于衡量这些项目性能的自动化工具更加困难，因为衡量度量指标所需的数据并不总是以相同的方式提供。为这些项目制定标准不仅能解决这一问题，还能帮助贡献者了解不同项目的开发方式，从而更容易为不同的开源软件项目做出贡献。

• **挑战 5：研究 DevOps 实践对开源软件项目性能的影响**。为了让所开发的基准和度量指标发挥作用，研究 DevOps 实践对开源软件项目及其性能的影响至关重要。了解哪些实践会影响性能以及它们以何种方式产生影响十分重要，这样才能制定提升性能的措施。

6 相关工作

我们关注能够揭示 DevOps 实践与开源软件项目性能提升之间关系的研究。此外，我们也重视那些提供基准测试方法（如《DevOps 状况报告》）以比较不同项目、团队或组织性能的研究。这些基准有助于我们了解不同实践和工具对性能的影响，并找出优化开源软件项目性能的最佳方法。此外，为了更好地了解 DORA 报告在多大程度上适用于开源软件项目的特点，我们希望找出这样的研究成果：它们发现，开源软件项目的某些属性会导致 DevOps 实践对项目性能的影响有所不同。

6.1 评估开源软件项目的性能

文献中有多项关于开源软件项目性能评估的研究。描述开源软件项目的文献中常见的一些功能包括 [8]:

- **大量无偿贡献者：**开源软件系统可能由大量志愿者（即数百人甚至数千人）构建。
- **地理分布式开发**：开发人员在不同地点工作，很少或从不面对面交流，并借助在线开发管理工具协调工作。
- **在某些情况下，公司会提供支持：**有许多开源软件项目得到公司的支持，其中一些参与者并非志愿者。
- **分布式设计与构建：**没有明确的系统级设计，甚至没有详细设计。
- **项目规划与任务分配：**开发人员会承担自己选择的工作。项目没有计划、进度安排或交付成果清单。此外，工作也不会被分派。取而代之的是，待办列表中列出了新功能和缺陷问题，这些问题反映了用户的需求；志愿贡献者会从中选择自己要承担的工作。他们会修复这些新功能或问题中的漏洞，并提出代码更改。

关于开源软件项目性能的大多数研究，都聚焦于某些实践和团队因素（例如生产力、规模、分散程度和经验）对项目效率与质量的影响 [20], [21], [22]。然而，这些研究均未引入基准测试方法，因此难以比较采用类似实践和工具的开源软件项目之间的改进幅度。在开源软件项目这类变化多端的环境中，采用基准测试方法至关重要。此外，不同的开源软件项目特征也会影响性能。全球软件开发（GSD）和分布式软件开发（DSD）是开源软件项目的重要特征，因为它们涉及大量贡献者在不同地点开展工作，并通过在线开发管理工具协调活动。实证研究表明，技术实践可能会影响性能。

- 与协同开发相比，协调机制（如 GitHub 工作流）似乎对地理分散式开发（即开源软件项目）的影响更大 [23]。此外，实证证据表明，地理分散会对团队绩效产生负面影响，而时间分散的影响则喜忧参半 [24]。
- 在开源软件项目中，大量贡献者寻找漏洞和错误，能够更快地发现并修复它们（这也被称为“Linus 定律”） [8]。
- 由于贡献者只会参与自己热衷的开源软件项目，他们似乎更具创造力，也往往更在意自己开发的代码[25]。贡献者的情绪和心境是工作体验的重要方面，会影响他们在开源软件项目中的性能 [26]。此外，开源软件项目及其社区的特性，使人们比起联系传统软件供应商，更容易找到并联系项目负责人 [27]。

实证证据支持采用实践和工具来提升开源软件项目的效率和效能。然而，结果会因各项目的具体特征而异，技术也在不断进步。20 年前，Mockus 等人开展了一项著名研究 [8],，分析了 Apache Web 服务器和 Mozilla 浏览器的数据，并探讨了高性能开源软件与商业混合体的潜力。然而，随着时间推移，该研究讨论的实践和工具如今未必仍能使开源软件项目达到高性能。

较新的研究是 Santos 等人的工作 [28]，他们收集了 90 个相关开源软件项目为期两年的持续集成子实践数据（构建时长、构建活动、构建健康状况、修复失败构建所需时间、提交活动），并进行了定性分析，以了解持续集成最佳实践是否与更高的项目质量相关。Joy 等人 [29] 利用 2017 年 GitHub 代码仓库的数据，研究了开源软件项目特征对其绩效的影响。研究证实，贡献者人数、项目年龄、项目规模和语言数量都很重要。该研究构建了包含七项假设的概念模型，并发现项目特征中的语言数量、项目规模、贡献者人数和项目年龄对项目绩效有显著影响。这些作者的另一项研究 [30], 提出了一个概念模型，描述项目、用户、贡献者和产权等因素如何影响开源软件项目绩效。然而，所有这些研究都揭示了影响项目效率和效能的因素，却没有一项能帮助开源软件社区判断自身项目的绩效水平。在此方面，这些研究都未提供用于比较项目的基准测试或分类模型；它们仅指出实践和所用工具方面的差异。

关于团队对开源软件项目生产力的影响，Pietrantonio 等人在其研究 [31] 中提出，使用开源游戏项目来评估团队绩效。他们分析了团队构成、工作经历、工作流程、沟通、性能、任务类型和环境等多个方面，并发现不同游戏在这些方面各具独特优势。此外，Scott 等人在其研究 [32] 中调查了开源软件项目中的七个敏捷软件开发团队的生产力，以确定关键因素。他们采用速度和专注因子作为衡量生产力的代理指标，并分析了生产力与开源软件相关因素之间的相关性，例如团队成员流动率、团队稳定性和迭代时长。他们的研究结果表明，较高的团队稳定性和较低的人员流动率与高速度迭代相关。然而，这些研究均未提供用于比较团队和项目之间生产力水平或相对绩效的基准。

此外，还有研究尝试预测开源软件项目的可行性和成功，这与开源软件项目绩效的概念相关 [33]。例如，Raja 等人 [34]发现，项目的可行性能够预测其存续，并指出编程语言、目标受众和项目类型等因素会影响可行性。他们还创建了三个维度（活力、韧性和组织）来评估项目的整体可行性。Schach 等人 [35]

分析了 SourceForge 上的 122,205 个项目，以确定开源软件项目的成功标准，并发现，在拥有一个或多个下载量的项目中，只有 25% 可被视为成功。Bayati [36] 通过分析新手在 GitHub 等社交编程环境中的声誉，聚焦于他们在开源软件社区中的成功。通过运用软件仓库挖掘技术，Bayati 找出了有助于新手取得成功的项目的主要属性。尽管这些研究揭示了影响项目成功的因素，但它们无法让开源软件社区确定自身的绩效水平。我们无法对项目进行相互比较并判断其表现如何，研究之间只有所用方法和工具的差异。

关于在组织中应用 DevOps 实践所带来的益处的实证证据，我们不仅可以在文献中找到调查研究，还可以通过访谈和案例研究进一步了解 DevOps [9], [10], [11]的益处与挑战。然而，Erich 等人[12] 指出，关于原则和 DevOps 实践有效性的实证证据，以及用于衡量其有效性的度量指标，仍存在一些空白。具体而言，作者未能在文献中找到这样的案例：某个组织声称已实施 DevOps，同时提供有关已观察到的益处的定量数据。

近年来，基准测试被用于比较开源软件项目的性能，重点仅限于比较使用机器人与不使用机器人时的性能。Markusse 等人 [37] 建议将机器人用作持续性能基准测试的工具。他们的研究表明，尽管 GitHub Actions 等工具让开发人员更容易使用机器人，但机器人在开源软件项目中仍未得到广泛采用。因此，这项研究证实了当前在开源软件项目中自动使用工具或机器人来衡量性能方面存在空白。

总而言之，关于评估开源软件项目性能的文献研究表明，持续集成和贡献者数量多等某些实践能够对生产力和质量产生积极影响。然而，我们仍需开展更多研究，以充分了解 DevOps 实践在多大程度上能够优化开源软件项目的性能，并提出用于比较不同项目性能的基准测试或分类模型。DevOps 状况报告为度量指标和基准测试提供了宝贵见解，但仍需进一步研究，以使这些度量指标适应开源软件项目的特定性质。总体而言，文献表明，在开源软件项目中采用 DevOps 实践有助于提升性能，但仍需开展更多研究，以充分了解优化这些项目性能的最佳方法。

6.2 DevOps 状况报告

《DevOps 状况报告》是由 Puppet Labs 和 Google/DORA 发布的一系列年度报告，介绍科技行业中 DevOps 运动的现状 [2], [38]。首份报告由 Puppet Labs 于 2011年 发布，收集了 4,000 多名 IT 专业人员的意见。2014年，Puppet Labs 和 DORA 开始合作编写这些报告，受访者人数增至来自 110 个国家的 9,200 多人。报告包含 IT 性能的度量指标，例如吞吐量指标（部署频率、变更前置时间）和稳定性指标（平均恢复时间、变更失败率），同时引入了基准测试的概念，使团队能够将自身的性能与高、中、低绩效团队进行比较。Puppet Labs 与 DORA 的这一合作于 2018 年结束。

2018年，DORA 发布的 Accelerate DevOps 现状报告引入了“精英”绩效者的概念，并增加了第五项度量指标，用于衡量运营性能。相比之下，Puppet Labs 于 2018年发布的 DevOps 状况报告分析了组织和团队如何发展，以达到 DevOps 演进的最高水平。该模型将受访者分为三个演进类别：“低”“中”和“高”。报告还得出结论，处于 DevOps 演进最高水平的组织也全面整合了安全实践。2021年，DORA 发布的报告纳入了有关站点可靠性工程（SRE）的运营实践，并指出，同时优先考虑交付和卓越运营的团队，其组织性能最高。

近年来，DevOps 状况报告也关注安全实践对于提升组织性能的重要性。报告指出，一个组织采用开发安全实践的最大预测因素似乎是文化因素，而非技术因素 [13]。为此，DORA 考虑了 SLSA（软件制品供应链级别）框架[39], ，该框架提供了一种通用语言，用于描述按级别组织的一系列软件供应链完整性实践；以及安全软件开发框架（SSDF） [40], ，该框架提供了一种通用语言，用于描述安全软件开发实践，帮助组织根据其业务/使命要求、风险承受能力和资源，对安全软件开发活动进行协调和排序。SLSA 框架基于业界认可的最佳实践，由四个逐步提高保证程度的级别组成；而 SSDF 则基于 BSA[10]、OWASP[11], 和 SAFECode[12] 等组织发布的文档。

7 结论与未来工作

我们提出了对 DevOps 实践及其对开源软件项目影响领域的重要贡献。我们证明，DORA 定义的度量指标并不完全适用于开源软件项目，因为它们的特征与 DORA 组织的特征不同。通过澄清基准测试中使用的术语，并关注代码的持续发布及其对第三方的影响，而非持续部署到生产环境并向其交付，我们得以开发出一种新的基准测试方法，用于衡量开源软件项目的性能。术语的澄清包括区分度量指标所测量的环境，将环境划分为三种：本地环境、开发环境和运维环境。这样的区分确保我们使用正确的度量指标来衡量绩效级别。度量指标的定义取决于该

10. https://www.bsa.org/
11. https://owasp.org/ 12. https://safecode.org/

环境，因为每种环境都有自身的特征和背景。开发环境是部署和测试软件的地方，而运维环境则是将软件作为服务交付给最终用户的地方。因此，适用于每种环境的实践各不相同，度量指标的描述也必须相应调整。为避免混淆，我们对“部署”“发布”和“交付”这几个术语作了更精确的定义。

此外，我们提出了专门针对开源软件开发相关问题的新度量指标。这些度量指标通过调整 DORA 度量指标以适应开源软件项目的特定特点而设计，能够更准确地反映其性能。我们建议使用发布频率和已发布变更的前置时间来衡量开源软件项目的吞吐量，并使用代码修复时间和缺陷问题率来评估其稳定性。对于每项度量指标，我们都给出了定义以及用于计算其数值的公式。我们认为，明确说明所使用的公式以及每项测量所包含的数据非常重要。在 DORA 报告中，每项度量指标所涉及的数据和概念定义得不够清晰，可能会造成混淆，进而导致结果和结论不准确。

我们发现，除了计算度量指标的平均值之外，还需要计算标准差，以便更详细地了解性能，因为这有助于深入认识度量指标随时间变化的稳定性。例如，发布频率的标准差较高，可能表明存在稳定性问题或缺乏妥善规划；标准差较低则可能表明规划得当且稳定性良好。通过标准差，我们发现，发布的周期性会显著影响已发布变更的前置时间和代码修复时间等度量指标。当发布频率的标准差较低（即发布更规律）时，已发布变更的前置时间和代码修复时间度量指标的数值更优，表明性能更好。

对选定的四个开源软件项目（Angular、Kubernetes、TensorFlow 和 Visual Studio Code）进行度量指标测量后，我们获得了有趣的结果和信息。这些项目的发布频率相近，差异不大。然而，它们的标准差差异显著。TensorFlow 的标准差远高于其他项目。已发布变更的前置时间这一度量指标显示出有趣的结果，因为各项目之间存在较大差异。我们发现，发布频率度量指标标准差较低的项目，在已发布变更的前置时间度量指标上表现更好。四个项目的代码修复时间度量指标表现都很低，这可能与开源软件项目的背景及其特点有关，因为它们未必会优先修复缺陷。缺陷问题率度量指标在各项目之间也存在较大差异。我们得出结论，代码仓库中缺陷问题的分类与识别方式对该指标有很大影响。缺陷问题率度量指标高度依赖于缺陷是否得到妥善识别，因此该指标的数值更好并不总是意味着性能更好。

我们的性能基准基于项目代码仓库进行自动测量，从而降低了通过调查收集数据所带来的风险。我们还开发了一款名为 Performance-Tracker 的新工具，它可以连接到开源软件项目代码仓库并应用我们的性能基准。该工具可以帮助从业者评估自己的开源软件项目并找出需要改进的方面。不过，我们遇到的主要挑战是，由于不同 GitHub 项目代码仓库中的工作流实践各不相同，应用这些度量指标颇具难度。要克服这一挑战，我们需要制定社区共享的最佳实践，或开发能够适应不同配置的工具。

总体而言，我们的研究有助于更好地理解开源软件项目的性能。我们希望，研究发现能为未来的 DORA 报告提供参考，并帮助希望提升其开源软件项目性能的实践者。

为了加深对开源软件项目中 DevOps 实践的理解，解决本文指出的挑战至关重要。开源社区各利益相关方之间的协作与合作对于克服这些挑战必不可少。

主要挑战之一是关键概念的定义不够明确。为此，未来的工作将聚焦于为 DevOps 概念建立一套通用词汇，并请该领域的专家对其进行验证。这将改善沟通，减少利益相关方之间的困惑，从而得出更精确的结果。

另一项挑战是需要可靠的度量指标和基准，以准确反映开源软件项目的性能。未来的工作将着重验证现有度量指标，并探索在不同情境下更相关或更适用的新指标。为开源软件项目制定标准化的 DevOps 实践，对于改善性能测量以及让贡献者更容易参与不同项目也至关重要。

此外，还需要研究是否有必要制定新的基准类别，以便更准确地对开源软件项目进行分类并衡量其性能。这包括探讨开源软件项目与 DORA 组织之间的关系，并确定现有基准类别是否适用于广泛的开源软件项目。

材料

为践行开放科学，本文使用的所有材料——包括工具、部署环境和全部原始数据——均可在线获取：https://github.com/diverso-lab/performance-tracker。

致谢

本研究部分由西班牙科学、创新与大学部资助，资助项目包括 NICO（PID2019-105455GB-C31）、SUDOQU（PID2021-126436OB-C21, 10.13039/ 501100011033, FEDER, 欧盟）、METAMORFOSIS（FEDER US-1381375）、Data-PL 项目，以及由欧盟 NextGenerationEU 资助的 2021-2023 年西班牙大学体系资格提升补助。此外，本研究还得到加泰罗尼亚自治区政府的支持

加泰罗尼亚的 IMP 综合研究组（2021- SGR-01252）以及 UPC 的 inLab FIB 也提供了支持。

参考文献

[1] O. S. I. OpenLogic by Perforce. 2023年开源报告：开源使用情况、市场趋势与分析。[在线]。可查阅：https://www.openlogic.com/resources/2023-state-open-source-report [2] G. Kim、J. Humble、P. Debois、J. Willis 和 N. Forsgren，《DevOps 手册：如何在技术组织中打造世界级的敏捷性、信度与安全性》。IT Revolution，2021年。[3] A. Mishra 和 Z. Otaiwi，《DevOps 与软件质量：系统映射》，《计算机科学评论》，卷38，100308, p. 。在线。2020 [ ] 10 1016 2020 可查阅：https://doi.org/ 。 /j.cosrev. 。 100308[4] F. Erich，《DevOps 只是开发与运维之间的互动》，载于《软件工程持续开发与软件生产和部署新范式专题研讨会论文集》。施普林格，2018年，第89–99页。[5] L. Leite、C. Rocha、F. Kon、D. Milojicic 和 P. Meirelles，《DevOps 概念与挑战调查》，《ACM 计算调查》（CSUR），卷52，第6期，第1–35页，2019年。[6] DORA 和 G. Cloud。DORA 研究计划 DevOps 状况报告。 [在线]。可查阅：https://www.devops-research.com/ research.html#reports [7] G. Anand 和 R. Kodali，《基准测试模型的基准测试》，《基准测试：国际期刊》，卷15，第3期，第257–291页，2008年。[8] A. Mockus、R. T. Fielding 和 J. D. Herbsleb，《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》（TOSEM），卷11，第3期，第309–346页，2002年。[9] F. M. Erich、C. Amrit 和 M. Daneva，《DevOps 实践应用的定性研究》，《软件期刊：演化与过程》，卷29，第6期，文章编号 e1885，2017年。[10] A. Caprarelli、E. Di Nitto 和 D. A. Tamburri，《迈向 DevOps 之路上的谬误与陷阱：一项纵向工业研究》，载于《软件工程持续开发与软件生产和部署新范式专题研讨会论文集》。施普林格，2020年，第200–210页。[11] E. Diel、S. Marczak 和 D. S. Cruzes，《分布式 DevOps 中的沟通挑战与战略》，载于《2016年第11届全球软件工程国际会议》（ICGSE）。电气电子工程师学会，2016年，第24–28页。[12] F. M. A. Erich、C. Amrit 和 M. Daneva，《DevOps 实践应用的定性研究》，《软件期刊：演化与过程》，卷29，第6期，文章编号 e1885，2017年，e1885 smr.1885。 [在线]。可查阅：https://onlinelibrary.wiley.com/doi/abs/ 10 1002 1885 。 /smr. [13] 2022 [D. . G. Cloud。加速。DevOps 状况报告。在线]。可查阅：https://cloud.google.com/devops/state-of-devops [14] K.-J. S. Martin Michlmayr、Brian Fitzgerald，《开源项目为何以及如何采用基于时间的发布？》，《IEEE 软件》，卷2015，第2期，第55–63页，2015年。[15] L. M. Weber、W. Saelens、R. Cannoodt、C. Soneson、A. Hapfelmeier、P. P. Gardner、A.-L. Boulesteix、Y. Saeys 和 M. D. Robinson，《计算方法基准测试的重要指南》，《基因组生物学》，卷20，第1期，第1–12页，2019年。[16] A.-L. Boulesteix、S. Lauer 和 M. J. Eugster，《呼吁对计算科学开展中立比较研究》，《公共科学图书馆·综合》，卷8，第4期，文章编号 e61562，2013年。[17] A.-L. Boulesteix、H. Binder、M. Abraha mowicz、W. Sauerbrei 等，《比较统计方法研究的必要性与设计》，《生物统计学杂志》，卷60，第1期，第216–218页，2017年。[18] J. C. Cort´es R´ıos、S. M. Embury 和 S. Eraslan，《Git 工作流系统分析的统一框架》，《信息与软件技术》，卷145，C期，2022年5月。 [在线]。可查阅：https://doi.org/ 10 1016 2021 106811 。 / [19] j.infsof. . C. Wohlin、P. Runeson、M. Host、M. Ohlsson、B. Regnell 和 A. Wessl´en，《软件工程实验》，计算机科学丛书。施普林格柏林海德堡，2012年。 [在线]。可查阅：https://books.google.es/books?id=QPVsM1U8nkC [20] R. Kazman、D. Goldenson、I. Monarch、W. Nichols 和 G. Valetto，《评估架构文档的影响：一个大型开源项目的案例研究》，《IEEE 软件工程汇刊》，卷42，第3期，第220–260页，2016年。

[21] P. H e, B. Li , and Y. H uang, “Applying cen trality 度量 to th 开源软件社区中开发人员行为分析”，载于《2012年第二届云计算与绿色计算国际会议》，2012年，第418–423页。[22] M. Grottke、L. M. Karg 和 A. Beckhaus，“团队因素与故障处理效率：对闭源和开源软件开发的探索性研究”，载于《2010年IEEE第34届年度计算机软件与应用会议》，2010年，第188–197页。[23] J. D. Herbsleb 和 R. E. Grinter，“拆分组织并整合代码：重新审视康威定律”，载于《第21届国际软件工程会议论文集》，系列：ICSE ’99。美国纽约州纽约市：计算机协会，1999年，第85–95页。 [在线]。链接：https://doi.org/10.1145/302405. 302455[24]N.-D. Anh、D. S. Cruzes 和 R. Conradi，“全球软件团队中的分散、协调与绩效：系统综述”，载于《ACM-IEEE国际实证软件工程与度量研讨会论文集》，系列：ESEM ’12。美国纽约州纽约市：计算机协会，2012年，第129–138页。 [在线]。链接：https://doi.org/ 10 1145 2372251 2372274 。 /。[25] G. Hertel、S. Niedner 和 S. Herrmann，“开源项目中软件开发人员的动机：一项针对Linux内核贡献者的互联网调查”，《研究政策》，第32卷，第7期，第1159–1177页，2003年，开源软件开发。 [在线]。链接：https://www.sciencedirect.com/science/article/pii/S 0048733303000477[26] H. M. Weiss 和 R. Cropanzano，“情感事件理论”，《组织行为研究》，第18卷，第1期，第1–74页，1996年。[27] A. I. Wasserman 和 E. Capra，“评估商业项目和社区开源项目中的软件工程过程”，载于《首届自由/开源软件研究与开发新兴趋势国际研讨会（FLOSS’07：ICSE 2007研讨会）》，2007年，第1–1页。[28] J. Santos、D. A. da Costa 和 U. Kulesza，“探究持续集成实践对开源项目生产力和质量的影响”，载于《ACM / IEEE国际实证软件工程与度量研讨会（ESEM）》。ACM，2022年9月。 [在线]。链接：https://doi.org/ 10 1145 2 3544902 3546244 。%[29] F . A. Joy、S. Thangavelu 和 A. Jyotishi，“GitHub开源软件项目的绩效：实证分析”，载于《2018年第二届电子、计算机与通信进展国际会议（ICAECC）》，2018年，第1–6页。[30] S. Thangavelu 和 A. Jyotishi，“开源软件项目绩效决定因素：对GitHub 项目的分阶段分析”，载于《2018年ACM SIGMIS计算机与人员研究会议论文集》，2018年，第41–42页。[31] J. D. Pietrantonio 和 D. Mendonc.a，“通过开源游戏揭示团队绩效的黑箱：综述与建议”，《IEEE游戏汇刊》，第14卷，第2期，第170–179页，2022年。[32] E. Scott、K. N. Charkie 和 D. Pfahl，“开源软件项目中敏捷团队的生产力、人员流动与团队稳定性”，载于《2020年Euromicro第46届软件工程与高级应用会议（SEAA）》，2020年，第124–131页。[33] P. V. Singh，“小世界效应：开发者协作网络宏观层面属性对开源项目成功的影响”，《《ACM 软件工程与方法学汇刊》（TOSEM）》，第20卷，第2期，第1–27页，2010年。[34] U. Raja 和 M. J. Tretter，“定义并评估开源项目存续能力的度量指标”，《IEEE 软件工程汇刊》，第38卷，第1期，第163–174页，2012年。[35] S. R. Schach、D. G. Feitelson 和 G. Z. Heller，“判定开源项目成功与否的实证标准”，载于《澳大利亚软件工程会议》。美国加利福尼亚州洛斯阿拉米托斯：IEEE计算机学会，2006年4月，第363–368页。 [在线]。链接：https://doi.ieeecomputersociety.org/10.1109/ASWEC.2006.12[36] S. Bayati，“理解新手在开源社区中的成功”，载于《第40届国际软件工程会议论文集：伴随论文集》，2018年，第224–225页。[37] F. Markusse、P. Leitner 和 A. Serebrenik，“使用基准测试机器人进行持续绩效评估”，《IEEE 软件》，第39卷，第5期，第50–55页，2022年。[38] N. Forsgren、J. Humble 和 G. Kim，《Accelerate：DevOps背后的科学：构建和扩展高绩效技术组织》，系列：G——参考、信息与跨学科

学科系列。IT Revolution，2018年。 [在线]。链接：https://books.google.es/books?id=85XHAQAACAAJ [39] S. 框架。软件制品供应链级别框架。 [在线]。链接：https://slsa.dev/ [40] 安全软件开发框架。安全软件开发框架。 [在线]。链接：https://csrc.nist.gov/Projects/ssdf



DavidBenavides 于 2000年 获得法国巴黎高等电子学院的信息系统理学学士学位，并分别于2001年 和 2007年 获得西班牙塞维利亚大学的计算机工程硕士学位和软件工程博士学位。他目前是塞维利亚大学的正教授。他的主要研究兴趣包括软件产品线，以及应用于工程教育的人工智能。



Jos´e Manuel S´anchez Ruiz于2021年7月获得塞维利亚大学计算机软件工程学位。目前，他是塞维利亚大学计算语言与系统系的研究人员。他参与与软件工程、软件团队开发以及工具、系统和应用程序开发相关的研究与课题。



Francisco Jos´eDom´ınguez Mayo 于2013年7月获得塞维利亚大学计算机科学博士学位。目前，他是塞维利亚大学计算语言与系统系的副教授。他与公共和私营公司合作，开展软件开发质量与质量保证相关工作。他的研究主要关注持续质量改进、软件产品质量保证以及软件开发过程。



ErnestTeniente 是加泰罗尼亚理工大学（UPC）的软件工程正教授。他同时担任inLab FIB主任，以及UPC信息建模与处理（IMP）研究组组长。他的研究兴趣包括本体与概念建模、业务流程建模、语义与自动推理、自动代码生成、完整性约束实施，以及数据集成。



Xavier Oriol 于2017年获得加泰罗尼亚理工大学（UPC）计算机科学博士学位，并获国际认可（优等）。他是inLab FIB的研究员和项目负责人，同时在该学院教授软件工程课程。他的研究兴趣包括增量完整性检查、语义，以及针对概念模式的自动推理。



Jos´e Francisco Crespo 是巴塞罗那信息学院的一名计算机科学家，该学院隶属于加泰罗尼亚理工大学（UPC）。他获得了 UPC 的计算机体系结构、网络与系统理学硕士学位。他拥有研发项目和开发敏捷软件项目的经验。他对将技术应用于协作环境和移动领域很感兴趣。目前，他参与 Web 项目和移动应用的开发。