InEx-Bug：NPM 生态系统中内在和外在错误的人工标注数据集

Tanner Wright
tanner.wright@ubc.ca 英属哥伦比亚大学 加拿大不列颠哥伦比亚省基洛纳

Adams Chen
achychen@student.ubc.ca 英属哥伦比亚大学 加拿大不列颠哥伦比亚省基洛纳

Gema Rodríguez-Pérez
gema.rodriguezperez@ubc.ca 英属哥伦比亚大学 加拿大不列颠哥伦比亚省基洛纳

arXiv:2602.13400v1 [cs.SE] 13 Feb 2026

摘要

了解软件缺陷的成因对于可靠的软件维护和生态系统稳定性至关重要。然而，现有缺陷数据集并未区分项目内部产生的问题与由外部依赖或环境因素导致的问题。本文介绍 InEx-Bug，这是一个由人工标注、包含来自 103 个 NPM 仓库的 377 个 GitHub 问题的数据集，并将问题归类为内在（内部缺陷）、外在（依赖/环境问题）、非缺陷或未知。除标签外，该数据集还包含丰富的时间和行为元数据，例如维护者参与情况、代码变更和重新开启模式。分析表明，与外生缺陷相比，内生缺陷解决得更快（中位数为 8.9 天，而非 10.2 天）、关闭率更高（92%，而非 78%），且更常需要代码变更（57%，而非 28%）。外生缺陷的重新开启率则更高（12%，而非 4%），复发延迟也更长（中位数为 157 天，而非 87 天）。该数据集为进一步研究 NPM 生态系统中的内在和外在缺陷奠定了基础。

CCS 概念

• **软件及其工程 → 软件开发中的协作**; **软件库与代码仓库**;**软件维护工具**.

关键词

内在缺陷, 外生缺陷,NPM,GitHub, 缺陷报告, 软件开发, 开源软件

ACM 参考文献格式：

Tanner Wright、Adams Chen和Gema Rodríguez-Pérez。2026。InEx-Bug: NPM 生态系统中内在和外在错误的人工标注数据集。载于第*23*届国际软件仓库挖掘会议（*MSR '26*），*2026*年*4*月*13*日至*14*日，里约热内卢，巴西。ACM，纽约，NY，美国，5页。https://doi.org/10.1145/3793302.3793309

1 引言

现代软件系统日益依赖由第三方软件包和库组成的大型动态生态系统。虽然这种依赖结构加快了开发进程并促进了代码复用，但也带来了风险，因为单个缺陷可能通过传递依赖传播，从而破坏数千个下游项目的功能 [1, 4, 9, 13]。在拥有超过 300 万个软件包的 NPM 生态系统中， [3],维护者必须定期判断所报告的问题究竟源于自身代码库（内生缺陷 [17]），还是由依赖更新、环境变更或上游 API 修改等外部因素引起（外生缺陷 [17]）。区分这两类问题通常并非易事，但对于有效维护至关重要，因为内生缺陷和外生缺陷在解决所需代码变更的规模上存在显著差异 [16]。

此外，先前的研究表明，缺陷跟踪系统中的问题约有 33.8% [7] 至 40% [15] 被错误标记，这意味着它们并非真正的缺陷，而是用户错误、疑问或功能请求。这种错误标记会威胁数据集的可靠性，并可能对自动化工具的开发和评估产生负面影响 [7]。迄今为止，唯一包含内在和外在标签的数据集是在 [16], 整理的；然而，该数据集仅限于单个项目（OpenStack）的提交级数据，因此，在 NPM 等多样化生态系统中，问题跟踪级别上的缺陷起源特征仍有待研究。

为弥补这些不足，我们引入**InEx-Bug [19]**,，这是一个由人工标注的数据集，包含来自 103 个 NPM 仓库的 377 个 GitHub 问题。基于对问题描述、讨论、关联代码变更和时间线事件的分析，每个问题都被分类为内在、外在、非缺陷或未知。不同于现有的缺陷数据集仅关注代码级或文本级标签 [5, 6, 10, 12, 18],，InEx-Bug 将缺陷起源标签与丰富的上下文与行为元数据相结合，包括时间指标、参与数据以及代码变更特征。

通过提供关于 NPM 中缺陷起源和维护者行为的细粒度、经人工验证的视角，InEx-Bug 能够支持自动化缺陷分类、依赖风险分析或维护者决策等研究。借此，它为进一步理解内在缺陷和外在缺陷如何在大规模软件生态系统中产生、传播及得到管理，贡献了基础性数据。

2 数据集构建方法

2.1 数据筛选与抽样

本研究使用的数据集源自 Saeidi 等人 [18],，其中包含约 30,000 条 GitHub 问题和拉取请求，涵盖 500 个依赖度最高的 NPM 包。为聚焦于与内在和外在缺陷分析相关的问题报告，我们从最初的 30,000 条缺陷报告中移除了所有拉取请求，最终得到 23,682 条问题。

这些剩余问题构成了我们数据收集过程的输入。在 23,682 份缺陷报告中，我们确定抽取 377 个问题作为样本，可在 ±5% 的误差范围内，以 95% 的置信水平

本作品根据知识共享署名 4.0 国际许可协议获得许可。*MSR '26*，里约热内卢，巴西© 2026 版权所有归所有者/作者所有。ACM ISBN 979-8-4007-2474-9/2026/04 https://doi.org/10.1145/3793302.3793309

估计总体比例。因此，我们从筛选后的数据集中随机选取了 377 个问题 URL 进行数据收集。

2.2 元数据提取

我们于2025年10月使用自定义 Python 脚本收集元数据，该脚本通过 GitHub REST API v3 获取全面的元数据。对于每个问题，我们收集了核心文本元数据（标题、正文、状态、时间戳、标签、受指派者、里程碑）、相关评论及其作者、完整的时间线事件（例如关闭、重新打开、引用和交叉引用），以及在适用时，详细的拉取请求或提交信息，包括代码变更、审阅者以及合并或提交时间戳。所有收集属性的完整列表均记录在我们的复现包 [19] 中。

为识别通过代码变更（即通过拉取请求或直接提交）关闭的问题 [11], ，我们采用了多策略检测方法。我们检查问题的关闭事件中是否直接引用了拉取请求，随后搜索时间线事件，查找指向拉取请求或提交的交叉引用。对于每个候选项，我们确认代码变更发生在问题创建之后，且合并/提交时间戳在问题关闭后的 7 天内。尽管 GitHub 会在拉取请求使用关闭关键词时于数秒内自动关闭问题，但这 7 天的窗口也涵盖了维护者在生产环境中验证修复后，或部署周期结束后手动关闭问题的工作流。我们指出，该阈值是一项设计选择，旨在兼顾纳入合理的基于代码变更的关闭和尽量减少无关提交，但可能会对本研究结果构成威胁。不过，其他研究人员可根据项目活跃度进行调整。通过直接提交关闭的问题会在 closing_commit 字段中单独跟踪。

生成的数据集采用 JSONL 格式，以便高效处理。每条问题记录包含：**(1)**核心元数据（标题、正文、状态、时间戳、标签、受指派者、里程碑）；**(2)**讨论数据（所有评论及其作者关联、评论数）；**(3)** 派生时间指标（首次响应时间、关闭时间及其他基于时间线的度量）；**(4)**参与度指标（独立评论者人数、维护者参与情况及角色）；**(5)** 关闭模式（重新开启事件、重新开启所需时间）；**(6)** 适用时基于代码的关闭详情（通过交叉引用时间线事件识别出的拉取请求或提交，包括代码变更、审阅者和合并时间戳）；以及**(7)** 人工分类标签（内在、外在、非缺陷和未知）。完整的列表定义请参见 [19] 。

维护者定义为其作者关联为所有者、成员、协作者或贡献者的项目用户。通过人工分析，我们在数据集中识别出两个自动化机器人账户（stale[bot] 和 vue-bot），并将其标记为排除在人工活动分析之外（12 个问题，占数据集的 3.2%）。关闭率反映维护者的人工活动，衡量的是通过人工操作而非自动化流程关闭的问题所占比例。在第 3 节中，所有 12 个由机器人关闭的问题均完全排除在时间分析之外。

2.3 标注流程

缺陷分类 每个问题都经过人工审查，并依据通过多次标注迭代完善的结构化标准，被分类为四个类别之一：内在、外在、非缺陷或未知。该过程确保了跨代码仓库对根本原因的一致理解，并准确区分内部项目缺陷与由外部依赖或环境引发的问题。该分类反映了根本问题的来源和性质，其视角是接收缺陷报告的代码仓库。例如，即使解决某个问题需要修改本地代码，只要缺陷源于外部，该问题仍可被分类为外在 。表1总结了标注过程中采用的操作性标准。

表1：各分类类别的定义。

标签	定义和常见类型
内在	该错误起源于其所属的软件包内部报告属于 [17]。典型案例包括回归错误、数据库或逻辑错误，或引入的任何缺陷在项目自身的代码中。
外在	该错误是由项目外部的变更或不兼容性引起的，例如依赖项升级，操作系统差异，或浏览器特定的行为。尽管项目代码此前运行正常，但在这些外部修改后便会失效 {v1}。其中包括依赖项不兼容（“依赖项 X 中的错误”）、操作系统特定错误，或之后需要进行适配。另一个项目的更新。
并非 Bug	该报告并不代表软件中存在实际缺陷。这包括测试或文档相关问题，功能增强请求、疑问，许可证澄清、配置误用或错误报告提交到了错误的代码仓库。
未知	缺乏足够背景信息来确定根本原因的报告。

人工标注流程： 该数据集采用严格的多阶段流程进行人工标注。两名标注者（第一作者和第三作者）在五轮迭代中共同标注了 161 个问题，并在每轮结束后讨论分歧，以确保对分类标准的理解一致，并准确识别错误报告类型。使用科恩卡帕系数 [2],衡量的评估者间信度（IRR）在每一轮都有所提高，最终一轮达到 $\kappa > 0.80$。达成较高的一致性后，第一作者根据最终确定的定义独立标注了剩余的 216 个问题。为验证独立标注结果和分类标准，另一名研究人员对 377 个问题中的 234 个进行了分类。所有差异均通过讨论解决；对于仍存在歧义的案例，则予以标记，并由研究团队共同审查以达成共识。

局限性： 该数据集反映的是截至2025年10月各代码仓库的状态，之后的变化可能未被纳入。诸如“关闭时间”或“维护者响应率”等指标可能受到工作流程因素而非缺陷复杂度的影响。尽管该数据集涵盖了不同规模的库和不同活跃度，但观察到的模式（例如外部错误比例、维护者参与度）可能无法推广到其他生态系统或工业

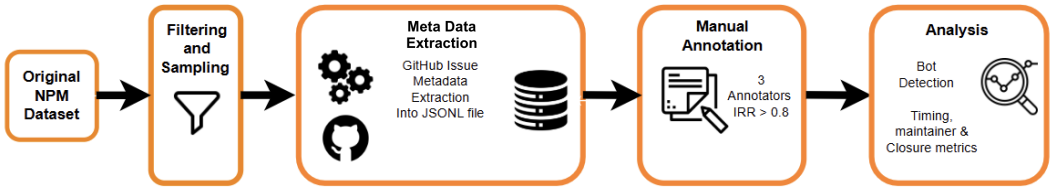

图 1：流程包括 (1) 原始数据、(2) 筛选和抽样、(3) 元数据提取、(4) 人工标注和 (5) 数据分析

表 2：按类别划分的问题特征 。所有时间指标均排除机器人关闭的问题；时间值报告中位数

类别	数量 关闭比例		中位关闭时间（天）	中位首次响应时间（小时）	平均评论数	平均参与人数	维护者响应比例	重新打开率	代码关闭率	平均行数	行数中位数	平均文件数	按拉取请求/提交的百分比
内在	105 (27.9%)	92.4	8.9	8.6	5.4	3.4	86.7	3.8	56.7	369	51	4.6	43.3 / 13.4
外在	37 (9.8%)	78.4	10.2	12.4	5.5	3.6	89.2	11.8	28.1	34	11	3.2	18.8 / 9.4
非缺陷	222（58.9%）	93.2	0.7	6.2	4.0	2.9	80.6	1.9	7.5	146	49	4.1	5.6 / 1.9
未知	13（3.4%）	76.9	2.1	28.4	2.1	2.2	46.2	0.0	0.0	–	–	–	— / —

环境。统计比较仅用于描述，该数据集旨在提供高质量的标注资源，而非用于假设检验。

3 数据分析

分布：表 2 展示了分类分布。非缺陷报告占 58.9%（222 个问题），其次是内生缺陷，占 27.9%（105 个问题）；外生缺陷占 9.8%（37 个问题），未知类别占 3.4%（13 个问题）。在实际缺陷报告中，内生缺陷的数量约为外生缺陷的 3 倍。

响应时间：表 2 按分类列出了关键时间指标。内生缺陷的首次响应时间中位数为 8.6 小时，外生缺陷为 12.4 小时，非缺陷报告为 6.2 小时，未知问题为 28.4 小时。各类别的维护者参与率均较高：内生缺陷为 86.7%，外生缺陷为 89.2%，非缺陷报告为 80.6%。不同缺陷类型的评论活动相近：内生问题平均有 5.4 条评论，外生问题平均有 5.5 条评论，非缺陷报告平均每个问题有 4.0 条评论。

关闭模式：内生缺陷的关闭速度快于外生缺陷（中位数分别为 8.9 天和 10.2 天），关闭率也更高（92.4% 和 78.4%）。非缺陷报告关闭迅速（中位数为 0.7 天，关闭率为 93.2%）。图 2 展示了关闭时间分布：虽然中位数差异不大，但内在和外在缺陷的分布相较于非缺陷报告都呈现长尾特征；其中，内在缺陷的关闭时间不超过 275 天，而外在缺陷则超过了 500 天。

重新打开：内生缺陷中有 3.8% 发生了重新开启事件，外生缺陷中为 11.8%，非缺陷报告中为 1.9%。对于重新打开的问题，从首次关闭到首次重新打开的中位时间，内生缺陷为 87 天，外生缺陷为 157 天。

代码变更：在问题关闭后七天内通过已合并的拉取请求或直接提交解决的、基于代码的问题，占所有问题的 21.2%。表 2 展示了按分类划分的细分情况。内生缺陷最常通过代码变更解决（占已关闭问题的 56.7%），而外生缺陷和非缺陷报告的这一比例分别为 28.1% 和 7.5%。代码变更的规模也有所不同。内生缺陷修复平均涉及 369 行变更（中位数为 51 行）

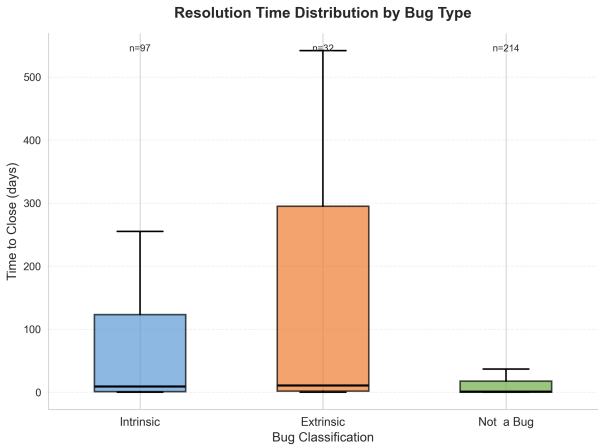

图 2：按缺陷分类划分的关闭时间分布。

涉及 4.6 个文件；相比之下，外在适配涉及的修改要少得多，平均变更 34 行（中位数为 11 行），涉及 3.2 个文件。

维护者参与度：图3展示了不同分类下维护者的参与模式。实际缺陷的响应率始终较高：内在问题中有 86.7%、外在问题中有 89.2% 至少收到一条维护者响应，而非缺陷报告的这一比例为 80.6%。每个问题涉及的维护者平均人数在不同缺陷类型间相近（1.3–1.4 人），总体讨论参与度也相近（缺陷为 3.4–3.6 名参与者，非缺陷为 2.9 名）。这些模式表明，在讨论阶段，维护者以相近的参与程度处理内在和外在缺陷。

4 讨论与启示

对 377 个人工标注的问题进行分析后发现，内在缺陷、外生缺陷和非缺陷报告呈现出截然不同的行为特征。尽管内在缺陷和外生缺陷的维护者参与度及讨论深度相近，但其背后的生命周期

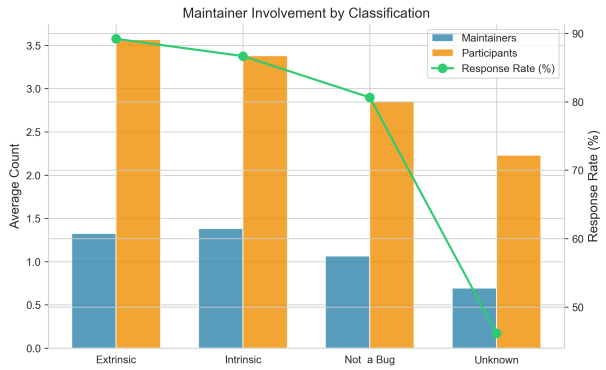

图3：各类别中的维护者参与情况，展示了响应率和维护者参与比例。

却存在显著差异，反映出开源维护生态系统的整体结构。

内生缺陷与外生缺陷的动态：内生缺陷往往会促使维护者直接进行代码修改，并更快地完成解决。其较短的中位关闭时间（8.9天）和较高的基于代码的关闭率（56.7%）表明，维护者能够迅速识别并修复其控制范围内的问题。相比之下，外生缺陷通常需要跨越项目边界进行协调，而且在最初解决很久之后仍会再次出现。较高的重新打开频率（11.8%）以及重新打开前较长的延迟时间（157天）表明，依赖相关回归问题可能会缓慢且不可预测地传播，这与 NPM 生态系统的去中心化特性相符。

非缺陷报告与维护者工作负担：所有分析的问题中，有58.9%被归类为非缺陷，凸显了维护者的大量精力用于用户支持、澄清配置错误以及提供文档反馈。尽管此类问题解决得很快（中位时间为0.7天），但它们占用了原本可用于处理技术缺陷的精力。这一发现表明，开源项目始终需要投入精力处理此类问题；对于 NPM 而言尤其如此，因为软件包不断依赖其他软件包，情况可能令人十分困惑。

4.1 数据集效用与研究应用

InEx-Bug 数据集为维护者和工具设计者提供了宝贵资源，带来可靠的洞见，可用于指导更有效的缺陷管理、工具开发以及软件生态系统的实证研究。

改进社区报告指南。非缺陷报告占主导这一现象表明，有必要制定更明确的问题模板、验证步骤或自动诊断方法。未来的工作可以探索减少 NPM 生态系统中非缺陷报告数量的策略。我们的数据集可作为起点，帮助识别相关模式，引导用户提交质量更高、可采取行动的报告。

理解维护者行为利用收集到的 InEx-Bug 元数据，可以研究不同代码仓库中的维护者行为模式和响应动态。研究人员可以调查错误来源如何影响维护者对不同类型问题的关注，并分析不同类型问题中的参与模式。

分析生态系统依赖关系InEx-Bug 支持对依赖引发的问题及其在软件包网络中的传播开展实证研究。研究人员可以考察外生缺陷的表现模式，调查依赖更新与问题重新打开之间的关系，并量化修复工作涉及多个项目或生态系统时所需的协调成本。

5 相关工作

缺陷起源。Rodríguez-Pérez 等人 [17] 的奠基性研究建立了理论框架，用于区分提交级数据中的内生缺陷与外生缺陷。他们将外生缺陷定义为由版本控制系统未记录的因素引起的缺陷，例如依赖更新或需求变更。后续研究 [16] 表明，OpenStack 中有 9–21% 的缺陷属于外生缺陷，并且混淆缺陷类型会降低即时缺陷预测模型的准确性。然而，这两项研究都仅限于单个项目内的提交级分析，因此尚未回答缺陷起源如何在跨生态系统的问题跟踪层面呈现。BuggIn [5] 等工具提出了针对内生缺陷的基于自然语言处理的自动分类方法，但缺乏人工验证标签，也未明确处理外生缺陷。我们的工作拓展了这条研究路线，提供了首个在问题级别对不同 NPM 包中的内生缺陷与外生缺陷进行区分的人工标注数据集。

缺陷报告数据集。Herzig 等人 [7]发现，约 33% 被标记为缺陷的报告实际上属于误分类（功能需求、重构或非问题），这凸显了人工核验的必要性。Eclipse 和 Mozilla [14]等数据集提供了丰富的问题跟踪历史，但并未按起源对缺陷进行分类。AndroR2 [8]等较新的数据集侧重于复现移动应用缺陷，而 BugsPHP 和 BugsJS 则针对特定语言的问题，服务于自动化程序修复。相比之下，InEx-Bug 明确区分内部缺陷与依赖引发的问题，并提供全面的行为元数据，这对于研究生态系统维护动态至关重要。

6 结论

本文介绍了 InEx-Bug，这是一个经过人工标注的数据集，包含 377 个 GitHub 问题；这些问题通过严格的多轮标注流程，并经由共识与可靠性检查，被分类为内在、外在或非缺陷。该数据集整合了文本和行为元数据，可用于详细分析项目维护动态。我们的研究结果揭示了内在缺陷与外生缺陷之间的显著差异，强调了在实证研究中区分缺陷起源的重要性。本文提供了可复用的脚本，用于复现定量结果，并将数据集扩展至新的代码仓库或生态系统 [19]。

7 致谢

我们谨向加拿大自然科学与工程研究理事会（NSERC）联盟（RGPIN-2022-03265）致谢，感谢其为本研究提供资助。

参考文献

[1] Filipe Roseiro Cogo、Gustavo A Oliva 和 Ahmed E Hassan。2019年。对 npm 生态系统中依赖降级的实证研究。*IEEE* 软件工程汇刊47，11（2019），2457–2470。[2] Jacob Cohen。1960年。名义量表的一致性系数。教育与心理测量 20, 1 1960 37 46 10 1177 001316446002000104（），–。[3] 2018 数字对象标识符：。/ Alexandre Decan、Tom Mens 和 Eleni Constantinou。。npm 软件包依赖网络中安全漏洞的影响。载于第*15*届国际软件仓库挖掘会议论文集。181–191。数字对象标识符：10.1145/3196398.3196401[4] Johannes Düsing 和 Ben Hermann。2022年。分析漏洞对不同制品仓库的直接和传递影响。*Digital Threats: Research and Practice* 3，4（2022年），1–25。[5] Rudolf Ferenc、Zoltán Tóth、Gergely Ladányi、István Siket 和 Tibor Gyimóthy。2018年。面向 Java 的公开统一漏洞数据集。载于第*14*届软件工程预测模型与数据分析国际会议论文集。12–21。[6] Steffen Herbold、Alexander Trautsch、Benjamin Ledel、Alireza Aghamohammadi、Taher A Ghaleb、Kuljit Kaur Chahal、Tim Bossenmaier、Bhaveet Nagaria、Philip Makedonski、Matin Nili Ahmadabadi 等。2022年。对漏洞修复提交中纠缠现象的细粒度数据集与分析。*Empirical Software Engineering* 27，6（2022年），125。[7] Kim Herzig、Sascha Just 和 Andreas Zeller。2013年。这不是漏洞，而是特性：错误分类如何影响漏洞预测。载于第*35*届国际软件工程会议论文集。392–401。数字对象标识符：10.1109/ICSE.2013. 6606585[8] Yixue Huang、Junjie Chen、Shu Xu、Zhenyu Chen、Xiaoyuan Xie 和 Jing Dong。2021年。AndroR2：一个针对 Android应用的人工复现缺陷报告数据集。载于第*18*届国际软件仓库挖掘会议论文集。IEEE，502–506。数字对象标识符：10.1109/MSR52588.2021.00064[9] Dhanushka Jayasuriya、Samuel Ou、Saakshi Hegde、Valerio Terragni、Jens Dietrich 和 Kelly Blincoe。2025 年。对实际场景中语法破坏性变更的扩展研究。*Empirical Software Engineering* 30，2（2025年），42。[10] René Just、Darioush Jalali 和 Michael D Ernst。2014年。Defects4J：一个用于支持 Java 程序受控测试研究的现有故障数据库。载于*2014*年软件测试与分析国际研讨会论文集。437–440。[11] Eirini Kalliamvakou、Georgios Gousios、Kelly Blincoe、Leif Singer、Daniel M German 和 Daniela Damian。2014年。GitHub 挖掘的承诺与风险。载于第*11*届软件仓库挖掘工作会议论文集。92–101。[12] Rafael-Michael Karampatsis 和 Charles Sutton。2020年。单语句缺陷多久会出现一次？ manysstubs4j 数据集。载于第*17*届国际软件仓库挖掘会议论文集。573–577。[13] Riivo Kikas、Georgios Gousios、Marlon Dumas 和 Dietmar Pfahl。2017年。软件包依赖网络的结构与演变。载于*2017*年 *IEEE/ACM* 第*14*届软件仓库挖掘国际会议（*MSR*）。IEEE，102–112。[14] Ahmed Lamkanfi、Javier Perez 和 Serge Demeyer。2013年。Eclipse 和 Mozilla 缺陷跟踪数据集：用于挖掘缺陷信息的真实数据集。载于第*10*届软件仓库挖掘工作会议论文集。IEEE，203–206。[15] Gema Rodríguez-Pérez、Jesús M Gonzalez-Barahona、Gregorio Robles、Dorealda Dalipaj 和 Nelson Sekitoleko。2016年。缺陷跟踪：辅助识别缺陷报告的工具。载于*IFIP* 开源系统国际会议。施普林格，192–198。[16] Gema Rodríguez-Pérez、Meiyappan Nagappan 和 Gregorio Robles。2022年。警惕外生缺陷！它们对 OpenStack 项目即时缺陷预测模型影响的案例研究。*IEEE* 软件工程汇刊 48, 4 2022 1400 1416 10 1109 2020 3021380 ( )，–。数字对象标识符：。/TSE.。Gema Rodríguez-Pérez、Gregorio Robles、Alexander Serebrenik、Andy Zaidman、Daniel M Germán 和 Jesus M Gonzalez-Barahona。2020年。缺陷如何产生：一种识别软件组件中缺陷引入方式的模型。*Empirical Software Engineering* 25，2（2020），1294–1340。数字对象标识符：10.1007/s10664-019-09781-y[18] Mohammadreza Saeidi、Raula Gaikovina Kula、Gema Rodríguez-Pérez 和 Ethan Thoma。2025年。我们的缺陷又如何？NPM 软件包维护者响应情况研究。载于*ACM/IEEE* 第*17*届实证软件工程与度量国际研讨会论文集（*ESEM*）。https://arxiv.org/abs/2511 04986。Tanner Wright、Adams Chen 和 Gema Rodriguez-Perez。2025年。*InEx-Bug*: [19] *NPM* 生态系统中内在和外在错误的人工标注数据集。不列颠哥伦比亚大学，基洛纳，不列颠哥伦比亚省，加拿大。https://github.com/STASER-Lab/InEx-Bug-A-Human-Annotated-Dataset-of-Intrinsic-and-Extrinsic-Bugs-in-the-NPM-Ecosystem 版本 1.0。