

开源软件库生态治理技术研究综述: 二十年进展*

王莹^{1,3,5}, 伍盈欣¹, 高天¹, 陈子莺¹, 许畅^{2,3}, 于海^{1,4}, 张成志⁵



¹(东北大学 软件学院, 辽宁 沈阳 110169)

²(南京大学 计算机科学与技术系, 江苏 南京 210023)

³(计算机软件新技术国家重点实验室(南京大学), 江苏 南京 210023)

⁴(智能工业数据解析与优化教育部重点实验室(东北大学 工业智能与系统优化国家级前沿科学中心), 辽宁 沈阳 110169)

⁵(香港科技大学 计算机科学及工程学系, 香港 999077)

通信作者: 于海, E-mail: yuhai@mail.neu.edu.cn; 张成志, E-mail: scc@cse.ust.hk

摘要: 在“人-机-物”三元融合、泛在计算的时代蓝海上,“开放多变”“需求多样”和“场景复杂”的软件部署和运行环境对开源软件库生态的治理技术提出了更多需求和更高期望。为进一步推动构建可信软件供应链生态,围绕泛在计算模式、打造自主可控的技术体系,聚焦于开源软件库管理生态,收集近 20 多年来(2001–2023)发表于软件工程领域较高影响力的学术期刊和会议的 348 篇论文,对开源软件库生态治理技术的研究工作进行梳理。讨论开源软件库生态的建模与分析、演化与维护、质量保证和管理等方面的工作,总结研究现状、问题、挑战与趋势。
关键词: 开源软件供应链; 开源治理; 软件库生态系统

中图法分类号: TP311

中文引用格式: 王莹, 伍盈欣, 高天, 陈子莺, 许畅, 于海, 张成志. 开源软件库生态治理技术研究综述: 二十年进展. 软件学报, 2024, 35(2): 629–674. <http://www.jos.org.cn/1000-9825/6983.htm>

英文引用格式: Wang Y, Wu YX, Gao T, Chen ZY, Xu C, Yu H, Cheung SC. Survey on Governance Technology of Open-source Software Library Ecosystem: Twenty Years of Progress. Ruan Jian Xue Bao/Journal of Software, 2024, 35(2): 629–674 (in Chinese). <http://www.jos.org.cn/1000-9825/6983.htm>

Survey on Governance Technology of Open-source Software Library Ecosystem: Twenty Years of Progress

WANG Ying^{1,3,5}, WU Ying-Xin¹, GAO Tian¹, CHEN Zi-Ying¹, XU Chang^{2,3}, YU Hai^{1,4}, CHEUNG Shing-Chi⁵

¹(Software College, Northeastern University, Shenyang 110169, China)

²(Department of Computer Science and Technology, Nanjing University, Nanjing 210023, China)

³(State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

⁴(Key Laboratory of Data Analytics and Optimization for Smart Industry, Ministry of Education (Frontier Science Center for Industrial Intelligence and System Optimization, Northeastern University), Shenyang 110169, China)

⁵(Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong 999077, China)

Abstract: Under the new era of “human-machine-thing” ternary integration and ubiquitous computing, the software deployment and operation environment of “open and changeable”, “diverse needs”, and “complex scenarios” have put forward more requirements and higher expectations for the governance of open-source software library ecosystems. To further promote the construction of trusted software supply chain ecosystems and create an independent and controllable technical system based on the ubiquitous computing model, this study focuses on open-source software library ecosystems. It collects 348 authoritative papers in this field in the past two decades (2001–2023)

* 基金项目: 国家自然科学基金(62141210, 61932021, 61902056, 61802164, 61977014); 中央高校基本科研业务费(N2217005); 南京大学软件新技术国家重点实验室开放基金(KFKT2021B01); 111 项目(B16009)

收稿时间: 2022-10-31; 修改时间: 2023-01-04, 2023-04-27; 采用时间: 2023-06-07; jos 在线出版时间: 2023-11-08

CNKI 网络首发时间: 2023-11-10

and sorts out the research work of open-source software library management ecological governance technology. The study discusses the modeling and analysis, evolution and maintenance, quality assurance, and management of open-source software supply chain ecosystems, and summarizes the research status, problems, challenges and trends.

Key words: open-source software supply chain; open-source governance; software library ecosystem

1 引言

当前,数字化浪潮与新产业革命正迎来历史性交汇,软件新技术正在对全球经济进行全方位、全角度、全链条的谱写和改造,软件产业对数字经济的支撑作用与日俱增^[1].与此同时,开源模式凭借强大的资源汇集和协同创新优势,已经成为全球软件开发的主导模式.开源软件遍布基础设施、数据管理、开发平台、应用程序等多个不同的领域,如今成为各领域软件开发不可或缺的“原材料”^[2].开源软件的生态建设随之成为全球软件产业乃至数字经济发展竞争的新赛道.

在“人-机-物”三元融合、泛在计算的时代蓝海下,“开放多变”“需求多样”和“场景复杂”的软件部署和运行环境对开源软件库生态治理提出了更多需求和更高期望^[2].从国家安全和产业安全的视角,汇聚和治理跨界涉众的开源软件制品、构建和维护可信、可持续演化的开源软件库生态是泛在计算新时代所面临的重要挑战^[3].然而,现代软件制品在生产过程中4个维度上的不可预见性、不可控制性、复杂性与多样性,使得“软件危机”已高悬在开源软件库供应链之上^[4].首先,软件演化行为的不可预见性.开源软件库生态中软件库有彼此不同的开发和维护计划,呈现出异步演化的趋势.软件项目“按需组装”的过程中,需要对它依赖软件库的演化状态及可信性进行验证,然而当前缺乏有效的技术为库版本的可靠保障机制提供决策支持.其次,软件构建环境的不可控制性.构建工具对软件库的安装规则复杂多变,与依赖配置属性存在干扰性.因此,开放难控的软件构建环境已经成为大型软件部署的痛点.第三,上下游软件间依赖关系的复杂性.开源软件库之间错综复杂的依赖关系使得缺陷和安全漏洞肆意传播,严重地威胁软件的可信性.第四,程序语言特征的多样性.在多样化程序语言配置模式的作用下,软件自身及其依赖资源的动态演化使得软件项目具有更高的复杂性,将其部署在泛在计算环境中会大大增加软件质量的不确定性.鉴于此,本文聚焦于开源软件库生态治理技术,总结和梳理该领域的研究现状、问题、挑战与趋势,以进一步推动构建可信开源软件供应链,打造自主可控的技术体系.

开源软件供应链是指现代软件开发和维护活动中涉及的所有上下游开源软件社区、源码软件库、二进制软件库、依赖管理器、软件中央仓库,以及开发者、维护者和基金会组织等实体,按照依赖、组合等约束关系形成的供应链网络^[1-3].本文聚焦于开源软件库生态相关的研究工作,它是开源软件供应链最基本的组成部分,涉及上下游开源软件社区中源码、二进制软件库、依赖管理器、软件中央仓库等实体在相互协作过程中的建模与分析、演化与维护、质量保证和管理等方面.全面收集近20多年来(2001-2023)该领域的实证研究、案例研究、分析方法及框架、技术、工具等权威学术期刊和会议论文,综述开源软件库生态治理技术的研究进展.

本文第2节介绍开源软件库管理与维护的基本概念.第3节描述研究方法.第4节分析研究现状.第5节讨论开源软件库生态的建模方法与多样化的演化行为.第6节和第7节梳理备受研究者关注的依赖缺陷问题及其治理技术.第8节讲述当前研究的局限性,展望未来的研究方向.第9节对比了相关研究工作.第10节总结全文.

2 开源软件库管理与维护的基本概念

为便于表述开源软件库生态的建模、演化、依赖缺陷问题和相应的治理技术,本节引入开源软件库管理与维护相关的4个基本概念:依赖管理器、软件中央仓库、软件构建方式和语义版本控制规则.软件中央仓库是开源软件库生态的实体形态,是建模分析的基础;依赖管理器决定了软件项目依赖配置和安装规则,在现代软件构建方式的作用下,是引发依赖缺陷问题的重要因素之一;在软件库演化的过程中,库开发者是否严格遵守语义版本控制规则影响软件库版本更新的兼容性问题,决定了开源软件库生态的可持续演化.

- 依赖管理器.在软件项目构建过程中,依赖管理器是用于解析依赖配置,以及自动安装、升级、配置和使用

软件库的系统或工具的集合. 依赖管理器能够极大地促进第三方软件库的重用, 简化软件项目构建和测试过程. 依赖管理器于 20 世纪 90 年代最早在 Unix 发行版中引入^[5]. 在基于 Linux 和 Unix 系统环境中, 为了便于依赖管理, 社区开发了多种依赖管理器, 例如 RedHat 的 yum, Debian 的 apt 等. 此后, 在其他程序语言社区也陆续推出了功能特点不同的依赖管理器, 例如 Java 的 Maven 和 Gradle, JavaScript 的 npm 等.

目前很多依赖管理器的功能也逐步扩展与完善, 除了能够管理第三方软件库, 同时兼具管理软件生命周期的构建、测试、部署和发布等全过程^[6]. 在过去的 20 年间, 研究人员^[7-18]在依赖管理器的解析能力改进方面也付出了巨大努力.

- 软件中央仓库. 软件中央仓库是依赖管理器团队自己创建并维护, 服务于整个开源社区的软件库存储库. 软件中央仓库包含了对应程序语言社区中大部分常用的软件库^[11,19-22]. 例如, 截至 2023 年 4 月, 软件中央仓库 PyPI 存储了近 45 万个 Python 软件库的 446 万个版本^[23].

- 软件构建方式. 纵观软件构建方式的演化历史, 其依赖资源已经从单一、封闭、静态、可控的自主软件库管理方式走向多样、开放动态、难控的“软件中央仓库”式集中化管理和自动化配置模式^[24]. 如图 1 所示, 依赖管理器能够在软件项目的构建环境中, 解析依赖配置文件——通过从软件中央仓库中远程检索直接依赖与间接依赖的软件库、计算依赖约束、获得依赖解决方案, 并且根据用户需求按照一定顺序和规则安装软件库.

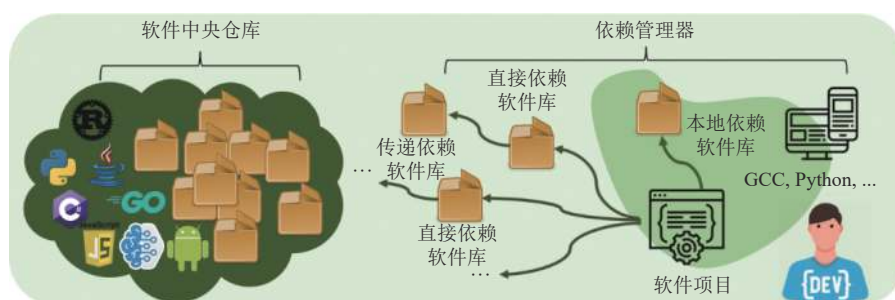


图 1 软件构建方式

- 语义版本控制规则. 语义版本控制是指软件版本的编号控制规则, 由 3 位版本号构成, 分别为主版本号、次版本号和补丁版本号: major.minor.patch^[25]. 主版本号增加表明当前版本较以往版本发生了 API 不兼容的修改, 次版本号增加表明软件以向后兼容的方式添加了新功能, 补丁版本号增加表明软件以向后兼容的方式修复了缺陷. 为保证开源软件库生态的可持续性演化, 语义版本控制规则在多种程序语言社区中被广泛采纳, 软件库的开发者在版本迭代之时需要遵守上述约定.

3 研究方法

本文采用 Wohlin 等人^[26]提出的系统文献综述方法, 结合 de Paulo Sobrinho 等人^[27]提出的文献收集方法开展开源软件库生态治理技术研究的综述工作. 本节中, 我们将详细介绍本文的研究方法.

3.1 研究问题

我们通过总结和分析相关论文来呈现开源软件库生态的研究历程, 并探究如何刻画、抽象并分析开源软件库生态, 如何诊断和治理不同程序语言软件库生态中的依赖缺陷问题, 为未来可能的研究方向提供参考. 为了实现这个目标, 本文提出并尝试回答如下问题.

- (1) 问题 1 (研究文献分布): 近 20 多年来 (2001–2023), 开源软件库生态研究文献的分布情况如何?
- (2) 问题 2 (建模与分析): 如何建模与分析开源软件库生态? 如何理解软件库生态的演化行为?
- (3) 问题 3 (依赖缺陷问题): 开源软件库生态的演化行为会导致哪些依赖缺陷问题?
- (4) 问题 4 (治理技术): 如何诊断与修复开源软件库生态中的依赖缺陷问题?

问题 1 是对目前领域内近 20 多年来文献数据的分析, 以了解本领域的研究现状. 问题 2 探究不同程序语言开

源软件库生态的建模方法和演化行为. 问题 3 讨论开源软件库生态多样化的演化行为导致的多种依赖缺陷问题. 问题 4 针对上述的依赖缺陷问题, 总结现有文献的诊断和治理关键技术.

3.2 论文收集

文献收集过程主要包括期刊/会议范围的确定、论文筛选、“滚雪球”式相关论文的筛选^[27]. 在论文收集过程中, 为了保证论文的相关性和权威性, 我们在第 3.3 节制定了筛选标准. 如图 2 所示, 为了避免搜索引擎关键字查询方法的漏查缺点, 尽可能全面地找到开源软件库生态研究领域权威期刊/会议的论文, 本文进行两轮筛选和扩充, 得到最终论文集. 具体步骤如下.

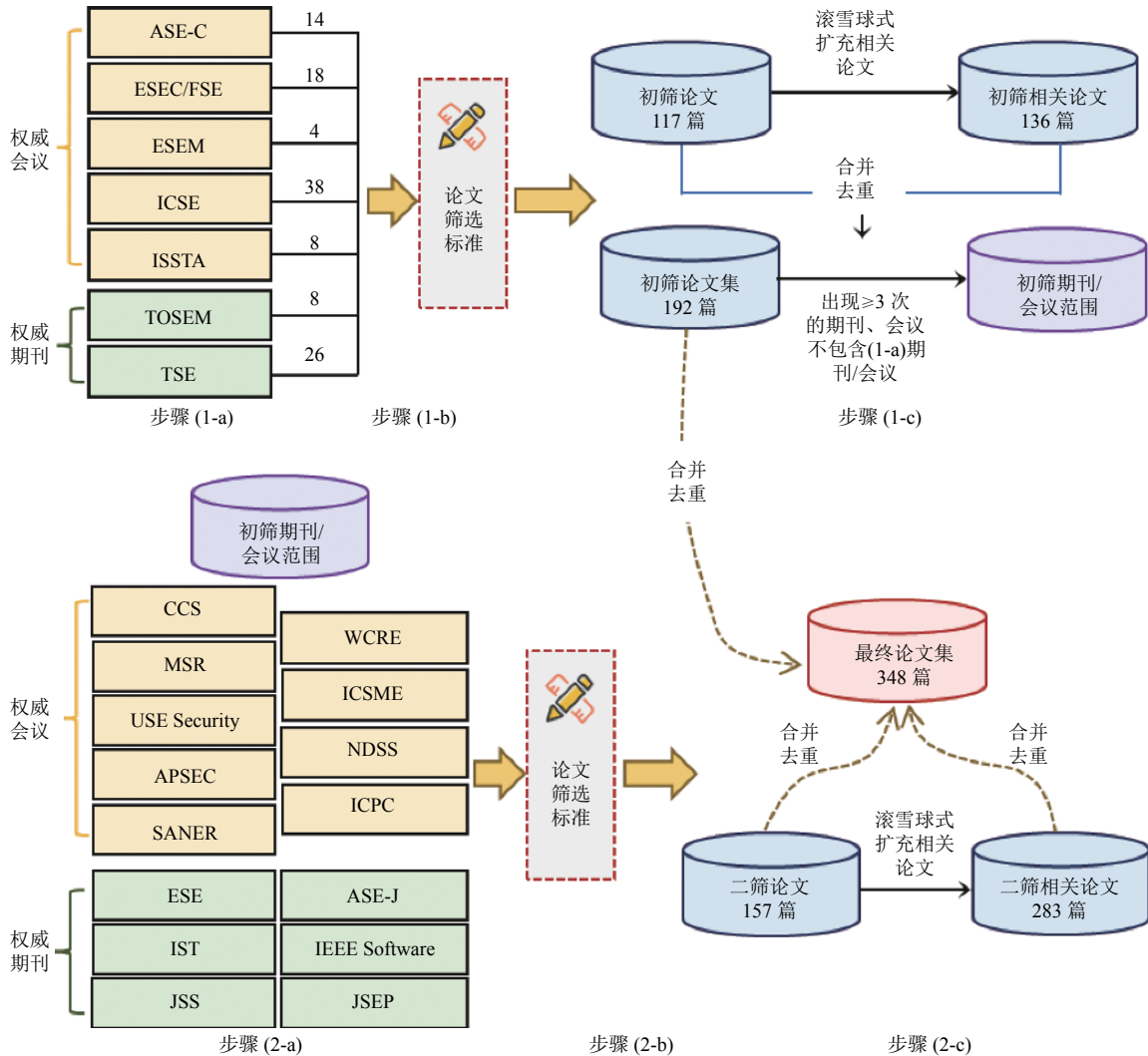


图 2 论文收集过程

(1) 初轮论文筛选. (1-a) 初轮筛选期刊/会议范围: 如表 1(1-a) 所示, 我们选取软件工程领域较高影响力的 7 个学术期刊和会议, 作为初次筛选的范围; (1-b) 初轮筛选论文: 对 (1-a) 步骤所得期刊/会议中发表于 2001 年 1 月–2023 年 4 月期间的所有论文按照第 3.3 节提出的 4 条标准进行筛选, 得到 117 篇论文; (1-c) 初轮筛选相关论文: 以“滚雪球”的方式^[27], 整理与收集 (1-b) 步骤所得论文中引用的参考文献, 同样按照第 3.3 节的标准从参考文献中筛选出 136 篇论文. 与 (1-b) 筛选论文进行合并去重后, 成为初筛论文集, 共计 192 篇.

(2) 二轮论文筛选. (2-a) 二轮筛选期刊/会议范围: 对初筛论文集中的期刊/会议出现的频次进行统计, 选取出现次数 3 次及以上的期刊/会议作为最终期刊/会议范围. 如表 1 所示, 我们共得到 19 个会议和 8 个期刊; (2-b) 二轮筛选论文: 在最终期刊/会议范围中除 (1-b) 外的期刊/会议, 针对发表于 2001 年 1 月–2023 年 4 月期间的所有论文按照第 3.3 节标准进行筛选, 共得到论文 157 篇; (2-c) 二轮筛选相关论文: 再次以“滚雪球”的方式^[27], 按照第 3.3 节的标准筛选 (2-b) 所得论文中引用的参考文献 283 篇, 与 (2-b) 筛选论文进行合并去重得到最终论文集, 共计 348 篇.

表 1 筛选期刊/会议集

步骤	编号	缩写	全称	种类	
(1-a)	1	ASE-C	International Conference on Automated Software Engineering	会议	
	2	ESEC/FSE	European Software Engineering Conference and Symposium on the Foundations of Software Engineering		
	3	ESEM	International Symposium on Empirical Software Engineering and Measurement		
	4	ICSE	International Conference on Software Engineering		
	5	ISSTA	International Symposium on Software Testing and Analysis		
	6	TOSEM	ACM Transactions on Software Engineering and Methodology		期刊
	7	TSE	Transactions on Software Engineering		
(2-a)	8	CCS	Conference on Computer and Communications Security	会议	
	9	MSR	Mining Software Repositories		
	10	USENIX Security	USENIX Security Symposium		
	11	APSEC	Asia-Pacific Software Engineering Conference		
	12	SANER	IEEE International Conference on Software Analysis, Evolution, and Reengineering		
	13	WCRE	Working Conference on Reverse Engineering		
	14	ICSME	IEEE International Conference on Software Maintenance and Evolution		
	15	NDSS	Network and Distributed System Security Symposium		
	16	ICPC	International Conference on Program Comprehension		
	17	ESE	Empirical Software Engineering		期刊
18	IST	Journal of Information Security and Applications			
19	JSS	Journal of Systems and Software			
20	ASE-J	Automated Software Engineering			
21	IEEE Software	IEEE Software			
22	JSEP	Journal of Software: Evolution and Process			
(2-c)	23	ECOOP	European Conference on Object-oriented Programming	会议	
	24	OOPSLA	ACM SIGPLAN International Conference on Object-oriented Programming Systems, Languages, and Applications		
	25	ICSR	International Conference on Software Reuse		
	26	OSS	International Conference on Open Source Systems		
	27	WISEC	Wireless Network Security		

3.3 论文筛选标准

为保证收集论文的相关性和权威性, 我们制定了文献筛选标准.

(1) 论文相关性: 根据引言和第 3.1 节中定义的研究范围和研究问题, 本文聚焦于开源软件库生态的建模与分析、演化与维护、质量保证和管理等方面的研究工作. 本文以研究对象“软件供应链”“软件库生态”“软件依赖”以及“第三方软件库”作为搜索关键词, 以收集与之主题相关的研究文献. 即论文标题、关键字和摘要出现“software ecosystem”“software supply chain”“software dependencies”“third-party libraries”的论文. 搜索可能会返回偶然使用关键字, 但却与该领域的建模与分析、演化与维护、质量保证和管理等方面无关的文献, 本文通过仔细阅读, 以交叉验证的方式排除无关文献.

(2) 论文完整性: 最全面、最新版本的论文. 如果在会议论文的基础上进行了扩展, 进一步形成期刊论文, 只保

留期刊论文版本, 排除会议论文版本.

- (3) 论文权威性: 经过同行评审且已正式发表的论文. 排除书籍章节、硕博论文等论文.
 (4) 论文页数: 正文页数大于等于 8 页的论文.

3.4 论文数据提取与收集

为回答研究问题 1-4, 我们对论文集进行研读, 并对所需数据项进行了分析和提取. 表 2 描述了从最终论文集的 324 篇论文中数据提取和收集的详细信息. “提取数据项”一列表明从每篇论文中提取的相关数据信息, “研究问题”一列表明对应的数据项所尝试回答的研究问题. 为了避免在人工分析过程中出现错误, 本文由两名研究人员独立分析和提取数据项, 并进行交叉验证, 以保证数据的正确性.

表 2 论文数据收集与提取

研究问题	提取数据项
问题1	论文的基本信息 (标题、发表年份、出版刊物等) 研究的主要贡献类型 (实证研究、案例研究、综述或算法)
问题2	关注的程序语言类型 对开源软件库生态生态的建模与分析方法 关注于开源软件库生态的哪种演化行为
问题3	研究开源软件库生态的哪种依赖缺陷问题
问题4	提出什么关键技术来检测和治理依赖缺陷问题

3.5 研究方法的局限性

(1) 搜索关键字. 根据文献 [28] 中的方法, 搜索关键字的建立应当是一个迭代扩充的过程, 即通过多次检索不断地补充与“software ecosystem”“software supply chain”“software dependencies”“third-party libraries”同义或相关的关键字. 由于上述关键字已经帮助我们定位到相当数量的研究论文, 本文没有进一步扩充关键字, 这可能会导致漏掉部分相关文献.

(2) 文献收集. 文献 [27] 提出的滚雪球式文献收集方式包括“前向搜索” (收集论文引用的文献) 和“后向搜索” (收集引用该论文的文献), 本文仅采用了“前向搜索”方式, 结合定义的关键字收集到 348 篇领域内权威文献, 但可能会因此而遗漏部分相关文献.

4 研究文献分布

我们通过分析每篇论文的发表时间、出版刊物和主要贡献类型等基本信息来理解开源软件库生态研究现状与趋势.

4.1 论文发表趋势

虽然我们尝试收集了 2001-2022 年这 20 多年间开源软件库生态治理技术领域的研究工作, 并对 2023 年 4 月之前的权威期刊和会议论文进行了补充. 但是结果表明, 自 2005 年起才开始涌现出相关的研究工作. 后文图 3(a) 描述了每年论文发表数量的分布情况, 可以观察到在 2010-2022 年的 10 多年间, 论文的发表数量迎来了快速增长. 大量研究的涌现应该是由近 10 年间开源生态的快速发展与演化以及现代软件项目愈发广泛地采用第三方软件库进行开发, 衍生出了一系列开源软件供应链的质量问题, 吸引了实践者和研究者的关注.

我们将论文累计发表数量拟合为幂函数, 来观察过去 20 年间开源软件库生态研究领域的论文发表趋势. 如图 3(b) 所示, 拟合曲线的斜率在 2010-2022 年间显著增加, 决定系数 (R^2) 达到峰值 (0.99746). 这意味着, 随着开源生态的繁荣与复杂度激增, 我们可以预测开源软件库生态治理技术的研究在未来会持续大幅增长.

4.2 论文发表刊物分布

图 4 呈现了论文发表刊物的分布情况. 在我们收集到的 348 篇论文集中, 68.1% 的研究发表于软件工程领域

的权威学术会议, 31.9% 的研究发表于期刊. 在 237 篇会议论文中, ICSE, MSR, SANER, ESEC/FSE, ICSM 和 ASE 这 6 个会议占据了 33.0% 的开源软件库生态治理研究工作的论文发表数量. 其中, ICSE 是收录相关研究最多的学术会议, 2001–2022 年间共发表论文 36 篇, MSR 和 SANER 紧随其后, 分别发表论文 23 和 16 篇, 并且在近 5 年呈现显著的增长趋势. 在 111 篇期刊论文中, 大多数发表于近 10 年间, 27.5% 的发表来自 TSE、ESE 和 JSS, 而 TOSEM 和 IST 的发表数量仅占 11.2%.

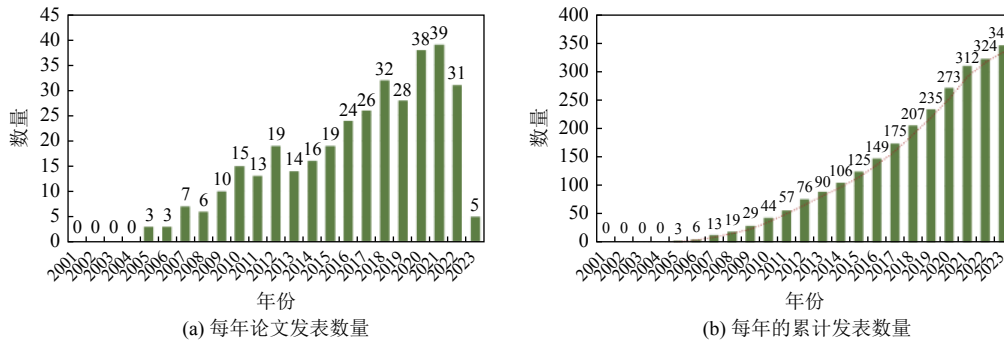


图 3 论文发表数量分布

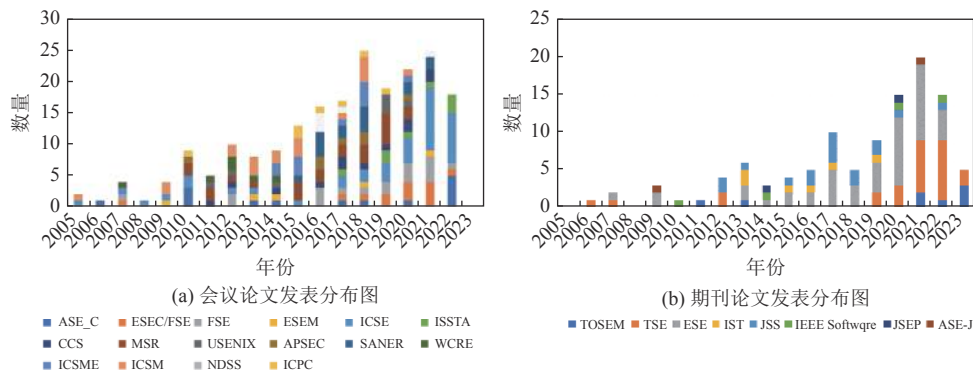


图 4 期刊/会议论文发表分布

4.3 主要贡献类型及研究热点

如表 3 所示, 我们把软件生态系统的研究内容划分为实证研究、案例研究、方法及框架、技术、工具及综述这 6 个类别, 并将收集到的论文集按照上述研究内容归类. 收集的论文集中: 149 篇论文以实证研究的方式收集并分析开源社区中软件仓库的演化元数据 (开源软件库生态依赖镜像、依赖树规模、引入的漏洞数量等) 来探索有价值的发现; 5 篇文章通过访谈或问卷的方式调查开发者和用户等人员对实际问题的看法来探索有价值的发现; 16 篇论文通过收集并分析缺陷问题的原因、症状和修复方式等信息为开发者提供问题解决的启发式发现; 34 篇论文提出度量方法或分析框架来调查开源软件库生态的某些演化行为或特定问题; 为解决特定的软件库依赖缺陷问题, 77 篇文献提出有效的技术方案或框架, 96 篇论文研究并发布了相应的工具; 目前仅有 7 篇文章综述了开源生态系统/安卓社区中第三方软件库相关研究工作. 部分论文在开展实证研究的基础上, 结合启发式的方法开发了解决实际问题的新技术/工具.

我们对每篇论文的关键词进行分析, 统计关键词的出现频次, 揭示论文的研究热点. 将关键词词频降序排序, 得到研究热点如表 4 所示. 频次排名前 20 的关键词为第三方软件库、实证研究、软件仓库挖掘、软件生态系统演化、安全漏洞、开源软件生态系统、API 破坏性变更、API 使用、语义版本、npm、Android、开源许可证、软件重用、API 演化、PyPI、依赖管理、依赖网络、软件维护和程序分析.

表 3 论文主要贡献类型

主要贡献	贡献说明	论文数量
实证研究	通过收集并分析开源社区中软件仓库的演化元数据 (开源软件库生态依赖镜像、依赖树规模、引入的漏洞数量等) 来探索有价值的发现	149
	通过访谈或问卷的方式调查开发者和用户等人员对实际问题的看法来探索有价值的发现	5
	通过收集并分析开源社区中某些特定缺陷问题的issue报告及知识问答等数据, 以分析和提炼缺陷原因、症状和修复方式等信息, 为开发人员提供问题解决的启发式发现	16
分析方法及框架	提出度量方法或分析框架来调查开源软件库生态的某些演化行为或特定问题	34
技术	提出技术方案或框架来检测或解决特定的开源软件库生态中依赖缺陷问题	77
工具	研究并发布了一个新的工具来解决特定的开源软件库生态中依赖缺陷问题	96
综述	开源软件库生态主题相关的研究论文中的数据、资料 and 主要观点进行归纳整理、分析提炼而写成的论文	7

注: 部分论文在开展实证研究的基础上, 开发了新技术/工具

表 4 Top20 关键词热点统计

关键词	频次	中文释义
Third-party libraries	123	第三方软件库
Empirical study	102	实证研究
Mining software repositories	91	软件仓库挖掘
Software ecosystem evolution	67	软件生态系统演化
Security vulnerabilities	65	安全漏洞
Open-source software ecosystem	58	开源软件生态系统
API breaking change	40	API破坏性更改
API usage	39	API使用
Semantic versioning	37	语义版本
npm	34	npm
Android	31	Android
Software licenses	20	开源许可证
Software reuse	19	软件重用
API evolution	19	API演化
PyPI	18	PyPI
Dependency management	17	依赖管理
Dependency network	11	依赖网络
Software maintenance	10	软件维护
Program analysis	10	程序分析

结合论文内容解读表 3 和表 4, 实证研究和软件仓库挖掘等技术是该领域中最常用的研究方法, 被 154 篇论文所采纳, 来开展开源生态系统的演化分析 (54 篇, 35.1%)、安全漏洞 (42 篇, 27.3%)、API 破坏性变更和使用 (36 篇, 23.4%)、开源许可证 (22 篇, 14.3%) 等方面的研究, 尝试挖掘出有价值的知识, 以提高开源软件库生态的可持续演化性。而 npm、PyPI 和 Android 是研究人员关注度较高的开源软件库生态社区。目前开源软件库生态治理的研究方兴未艾, 在质量保证、工具支持以及软件生态系统新范型等方面亟需研究人员开展深入探索

研究问题 1 结论总结: 在我们收集到的 348 篇论文集中, 68.1% 的研究发表于软件工程领域影响力较高的学术会议 (包括 ICSE, MSR, SANER, ESEC/FSE, ICSM 和 ASE 等), 31.9% 的研究发表于软件工程领域影响力较高的期刊 (包括 TSE, TOSEM, JSS 和 IST 等)。实证研究和软件仓库挖掘等技术是该领域中最常用的研究方法, 被 154 篇论文所采纳, 来开展开源生态系统的演化分析 (54 篇, 35.1%)、安全漏洞 (42 篇, 27.3%)、API 破坏性变更和使用 (36 篇, 23.4%)、开源许可证 (22 篇, 14.3%) 等方面的研究, 尝试挖掘出有价值的知识, 以提高开源软件库生态

的可持续演化性. 而 npm、PyPI 和 Android 是研究人员关注度较高的开源软件库生态社区.

5 开源软件库生态的建模与分析

数以万计相互依存的软件项目所形成的供应链网络为软件开发带来了便利, 同时也为依赖管理带来了前所未有的挑战. 规模指数级增长的软件项目及其之间庞杂的依赖关系使得供应链的复杂度激增^[29]. 一方面, 供应链上任何一个节点的问题都可能波及链条上其他节点, 使得损失被放大. 另一方面, 任何一个节点都可能受到供应链上其他节点的问题影响, 使得问题的定位与解决变得困难^[30]. 在这样的背景下, 许多研究者聚焦在如何刻画、抽象和分析复杂的依赖关系网络, 以及如何基于网络模型探索生态的演化行为.

5.1 开源软件库生态网络模型

开源软件库依赖关系建模是理解和观察软件生态演化的重要手段与前提. 研究工作中通常将软件库表示成有向图 $G=(V, E)$, 其中, 节点 V 代表软件中央仓库中的软件库版本集合, 边 $E=\{(Lib_i, Lib_j, Constraint) \mid Lib_i, Lib_j \in V\}$ 表示软件库版本之间的依赖关系集合. 对于依赖关系 $Lib_i \rightarrow Lib_j$ 而言, 下游软件库的版本 Lib_j 由上游软件库 Lib_i 对其的版本约束 $Constraint$ 以及构建环境对软件库版本的依赖解析规则共同决定.

不同程序语言社区在软件库的依赖管理与依赖解析方面定制了语法规则, 表 5 是对开源软件库生态建模的相关工作的总结, 具体来说有以下几个方面.

- 在依赖管理方面, 除了允许软件明确地声明它依赖的上游软件库特定版本以外, 还允许声明版本范围 $Constraint$ 来约束第三方软件库的版本 (例如 $< 2.1.0$). 更为重要的是, Python 和 JavaScript 等语言社区定制了语义版本约束规则, 开发者可以采用 $*$, $^$, \sim 等操作符进行灵活的控制. 例如, 版本约束 $2.1.* \sim 2.1.0$ 表示可以安装 $[2.1.0, 2.2.0)$ 范围内向后兼容的软件库版本.

- 在依赖解析方面, Python 和 JavaScript 等语言的依赖管理器 pip 和 npm 会从依赖配置脚本中解析满足该版本范围 $Constraint$ 中的最新版本 (例如, 在 $[2.1.0, 2.2.0)$ 版本范围中解析并安装软件库的 2.1.9 版本), 而 C# 的依赖管理器 NuGet 则会选择版本约束范围中的最旧版本 (例如, 在 $[2.1.0, 2.2.0)$ 版本范围中解析并安装软件库的 2.1.0 版本).

现有的研究工作^[31-43]尝试对多种程序语言软件库生态建模, 涵盖了 10 个软件中央仓库. 综合上述文献, 我们将对应的依赖管理器、依赖配置文件、依赖版本解析方式总结于后文表 5 中. 文献^[33,34,40]在建模过程中, 依赖元数据的获取源自监控开源软件库生态演化的开源数据平台 *libraries.io*, 其余研究通过爬取软件项目的依赖配置文件, 根据每种程序语言的依赖解析规则构建生态网络模型.

Mora-Cantalops 等人^[39]在对 CRAN 生态建模的基础上, 利用复杂网络理解进行分析, 证明依赖网络符合无标度网络的特性以及小世界特性 (节点之间的平均距离很短). Zimmermann 等人^[43]结合 npm 生态依赖模型定义了一系列度量指标, 系统地分析了开源软件库的安全漏洞攻击面. Lertwittayatrai 等人^[38]将 npm 生态剖析成高维度拓扑结构数据, 通过软件库之间的依赖结构提供对生态系统的洞察与见解. 文献^[33,34,36,37,40]对多种程序语言的软件生态进行建模, 揭示了不同生态拓扑结构、软件库版本演化方面的差异. 而 Bavota 等人^[32]、Nielsen 等人^[44]和 Wittern 等人^[41]分别针对 Maven 生态、Apache 社区与 npm 生态进行依赖拓扑的演化分析. Abate 等人^[31]、Zheng 等人^[42]和 German 等人^[35]等面向 Linux 社区的软件库依赖关系进行建模分析, 其中文献^[35]提出技术框架以形式化建模与可视化分析 C 语言软件库之间的依赖关系, 模型中的节点嵌入了多维度的属性信息, 包括版本和开源许可证等.

5.2 开源软件库生态的演化行为

在开源软件库生态建模的基础上, 关于演化行为的研究也随之展开. 开源生态中上下游软件库的异步演化, 是引发依赖缺陷问题的根源. 因此, 了解开源软件库生态的演化行为是开源生态治理的重要前提. 如后文表 6 所示, 我们收集到的 95 篇研究开源软件库生态演化行为的文献大致涵盖这 3 个方面, 分别面向不同的程序语言社区探讨其规模增长、持续更新和复杂性的增加.

表 5 开源软件库生态建模的相关工作

图标	语言	软件中央仓库	仓库规模	依赖管理工具	依赖配置文件	依赖版本解析方式	文献
 Java	Java	Maven	477k	Maven/Gradle	pom.xml/build.gradle	特定版本	[32,37,40,44]
 Python	Python	PyPI	450k	pip	setup.py/setup.cfg/ requirements.txt/ pyproject.toml	语义版本解析规则(约束 范围内最新版本)	[40]
 Go	Go	GitHub	375k	Goland	go.mod	Go语义导入版本控制规 则(规则下最新版本)	[35]
 JavaScript	JavaScript	npm	2390 k	npm	package.json	语义版本解析规则(约束 范围内最新版本)	[33,34,36, 38,40,41,43]
 Rust	Rust	Crate	84.9k	Cargo	cargo.toml/cargo.lock	语义版本解析规则(约束 范围内最新版本)	[33,34,36]
 R	R	CRAN	22.3k	CRAN	Decription	不指定版本	[33,34,37,39]
 C#	C#	Nuget	383k	NuGet	PackageReferrence	语义版本解析规则(约束 范围内最旧版本)	[33,34]
 PHP	PHP	Packagist	362k	Composer	composer.json	语义版本解析规则(约束 范围内最新版本)	[33,34,40]
 Ruby	Ruby	Rubygems	179k	Bundler	gemfile	语义版本解析规则(约束 范围内最新版本)	[33,34,36,40]
 C/C++	C/C++	RedHat/ Debian等 Linux发行版	各发行版社 区规模大小 不一	apt/yum	metadata	特定版本、约束范围内 最新版本	[31,35,42]

注: 软件中央仓库规模来自2023年4月30日的统计数据

5.2.1 开源软件库生态的规模增长

现有的研究表明, 各个程序语言的开源软件库生态规模增长明显, 无论是软件库的数量还是依赖关系数量均呈现持续增长趋势, 但不同社区之间又存在一定差异. 本节从上述两个维度, 阐述开源软件库生态规模增长的趋势.

● 软件库数量: 综合研究工作^[32-34,41,45-55]的实践调查结果, 软件库数量呈线性增长的生态社区包括 Maven、Cargo、CPAN、NuGet 和 Linux, 演化趋势呈指数增长的程序语言生态社区有 npm、CRAN 和 Packagist. 而 RubyGems 生态社区中由于统计年份的不同, 虽然都是用回归模型对软件包的增长进行分析, 软件包数量在不同年份的统计分析中却呈现出不同的增长特性, 2007–2016 年呈线性增长^[33], 2012–2017 年呈指数增长^[34].

● 依赖关系数量: 文献^[32-34,36,41,45,46,48-51]的研究表明, 依赖关系数量呈线性增长的生态社区包括 RubyGems、Packagist、CRAN、Cargo 和 Packagist, 演化趋势呈指数增长的程序语言生态社区有 Maven、npm、CPAN、NuGet 和 Linux. 通过对比可以观察到, 不同程序语言生态社区中软件库依赖关系演化趋势可能不同, 但总体上都呈现出规模增长.

表 6 开源软件库生态演化行为的相关工作

类型	属性	npm	CRAN	Packagist	CPAN	Maven	Cargo	NuGet	RubyGems	PyPI	Linux	Android	无特定语言	
规模增长	软件库数量	[32-34,41,43,45-47,52-55]									[54]	[48-51]		
	依赖关系数量	[32-34,36,41,45,46]												
持续更新	新增库数量	[41]	[46,56]								[57]	[58]		
	版本更新数量	[33,34,41,46]						[33,34,41]				[48,50]		
	版本更新频次	[34,46]					[59-65]					[66]	[60]	[67]
	版本更新滞后	[33,34,43,68-77]					[78-92]		[33,34,36,73,81]	[71,81,93,94]	[93,95,96]	[97-106]	[107]	
	软件库迁移	[108,109]					[108-129]				[130]		[111]	
复杂性增加	直接依赖软件库数量	[33,36,41,43]	[33,41,43,46,56]				[36,41]		[33,36,41,43]		[49,51,131]			
	传递依赖软件库数量													
	上游软件数量	[36]					[36]		[36]					
	下游软件数量	[36,41]		[56]			[36,41]		[36,41]					

注: 表格中空白部分表示没有论文涉及过相关研究

5.2.2 开源软件库生态的持续更新

软件库的增加与版本更新推动着开源软件库生态的可持续演化. 现有的文献针对如下几个方面讨论不同程序语言社区的持续更新行为.

● 新增软件库数量: 文献^[41,46,52,56-58]对每月新增软件库数量的统计结果显示, 每种程序语言生态社区均呈现稳步增长趋势. 其中, npm 生态每月新增软件库数量最多, CRAN 生态每月版本更新数量与新增软件库数量比值稳定, 版本更新数量大约为新增软件库数量的 7.5 倍. 调查结果体现了开源软件库生态的持续更新与繁荣发展.

● 版本更新数量: 综合文献^[33,34,41,46,48,50,57]在多个程序语言生态社区的实践调查, 可以发现软件库每月版本更新数量整体呈现上升趋势. 然而, 由于不同程序语言社区的年龄、开发者习惯和版本发布规则不同, 导致他们在版本更新数量方面存在差异: (1) 2015 年 60% 的 npm 软件库、48% 的 CRAN 软件库、27% 的 RubyGems 软件库都至少更新过一次, 整体呈现出活跃性. (2) 在 1999–2018 年的统计中, 发现 CRAN 生态每月版本更新的数量

符合指数增长,这可能与 CRAN 的“滚动发布 (rolling release)”规则有关,社区强制软件项目引用第三方软件库的最新版本。(3)在 2012–2017 年的统计中,Cargo 和 CPAN 生态的软件库版本更新数量保持稳定.研究者分析,可能是由于社区的年龄与活跃度的原因,Cargo 是最年轻的生态社区,CPAN 是最古老的生态社区.相比之下,npm、NuGet、RubyGems 和 Packagist 生态的软件库版本更新数量要大得多.

- 版本更新频次:文献 [34,46] 调查并对比了 CRAN、CPAN、Cargo、RubyGems 和 Packagist 几个社区的软件库版本更新频次.结果表明,npm 生态的持续更新性最强,NuGet 生态的持续更新性较高且不断增加.综合看来,每个程序语言生态社区中,接近 1/3 的软件库在一年中版本更新了 5 次以上,而 1/3 软件库没有任何版本发布.文献 [59–67] 研究发现,软件库的更新频次与其流行度存在密切的联系,更新频次更高的软件库往往流行度也更高.

- 版本更新滞后:综合文献 [33,34,36,43,68–74,78–82,97–100,107] 中对不同程序语言生态社区中软件库版本更新滞后现象的实践调查,可以发现除了 CRAN 社区外(“滚动发布”规则),其他生态均呈现一定程度的版本更新滞后现象.该现象也是导致大量安全漏洞在开源软件库生态中肆意传播的主要原因,漏洞补丁无法及时传播进下游的软件项目中.针对上述问题,近年来涌现出大量研究 [75–77,83–96,101–106] 帮助开发者对第三方软件库进行安全的版本升级.

- 软件库迁移:在软件演化过程中,当缺陷无法通过升级第三方软件库的版本而解决时,软件库需要被完全移除,被另一个软件库所取代.现有研究提出了在开源软件库生态中识别库迁移的方法 [110,111] 和迁移演化规则的方法 [108,112,113],开展了库迁移趋势的实证调查 [132],并开发出能够量化依赖变化和软件库迁移的框架,以全面地捕捉迁移的演化行为特征.

同时,为了解决软件库迁移引发的代码修改和兼容性问题,近年来研究者提出了大量技术和工具 [109,114–130] 以辅助开发者完成迁移操作.

5.2.3 开源软件库生态的复杂性增加

伴随着开源软件库生态的规模增长与持续更新,其复杂性必然会与日俱增.现有文献关于复杂性增加的研究与调查,从如下几个方面开展.

- 直接依赖/间接依赖软件库数量:开源软件库生态的复杂性最直观的表现是软件库的传递性依赖关系.文献 [33,36,41,43,46,49,51,56,131] 的研究表明,软件在演化过程中,通常直接依赖软件库的数量增长缓慢,而间接依赖软件库却呈现快速增长趋势.每种程序语言生态中直接依赖软件库的小幅线性增长会导致间接依赖软件库数量的指数增长.研究者提供了 npm、CRAN 和 RubyGems 生态社区在 2007–2016 年间的统计情况:(1) npm 生态中有一半的软件库至少引入 22 个间接依赖软件库,1/4 的软件库至少有 95 个间接依赖,2018 年平均软件的间接依赖数量达到 80 个,呈指数增长.平均一个软件库的间接依赖软件库比直接依赖软件库多 22.1 倍.(2) CRAN 生态中有一半的软件最多有 5 个直接依赖软件库,平均每个软件库的间接依赖比直接依赖数量多 3.64 倍.(3) RubyGems 生态中一半的软件最多有 8 个直接依赖软件库.平均每个软件库的间接依赖比直接依赖数量多 6.35 倍.与 RubyGems 生态相比,npm 软件具备更多的传递依赖软件库,但直接依赖软件库较少.

- 上游软件/下游软件数量:开源软件库生态的复杂性的另一种表现是上游软件和下游软件数量的激增.文献 [36,41,56] 的研究表明,很多程序语言生态社区中平均下游软件数量增长比平均上游软件数量增长快,例如 RubyGems 和 npm 社区.而 RubyGems 生态中软件库的发布数量虽然远远少于 npm 生态,但是每个软件库的平均下游软件数量相比于 npm 生态更多.因此,当有携带安全漏洞的软件包出现时,会在生态社区中造成更大的“涟漪效应”.另有研究发现,在 npm、RubyGems 和 CRAN 这 3 个生态社区中,孤立软件库所占比例不断减少,3 个社区各自呈现出一个巨大的连通软件库网络.在连通软件库网络分别占据 CRAN 社区 96.1% 的软件库,npm 社区 98.2% 的软件库,RubyGems 社区 100% 的软件库.开源软件库生态依赖的拓扑结构符合复杂网络的聚类特性,也预示着生态复杂性的不断增加.

随着开源软件库生态的规模增长、持续演化以及复杂性的增加,在海量的软件中央仓库中如何选择合适的第三方软件库以及如何正确使用第三方软件库给开发者带来了巨大挑战 [133–145].为软件项目引入大量第三方软件库的同时,也必将伴随着诸如易理解性 [146]、可维护性 [146–149]、性能降低 [150,151] 和构建失败 [152] 等风险,导致多样化的依赖缺陷问题.

研究问题 2 结论总结: 现有的研究工作利用图论思想对多种程序语言软件库生态建立网络依赖模型, 涵盖了包括 Maven, NuGet, PyPI, npm 等在内的十几个软件中央仓库. 将上下游软件库视为节点, 软件库之间的依赖约束关系视为有向边, 其中依赖关系由版本约束范围依赖管理和依赖版本解析规则共同决定. 在开源软件库生态建模的基础上, 关于演化行为的研究也随之展开. 我们收集到的 95 篇研究开源软件库生态演化行为的文献大致涵盖这 3 个方面, 分别面向不同的程序语言社区探讨其规模增长、持续更新和复杂性的增加.

6 开源软件库生态的依赖缺陷问题

在开源软件库生态多样化的异步演化行为、构建环境限制、依赖配置模式复杂性等因素的相互影响和制约下, 会导致 4 种常见的依赖缺陷问题: 软件版本演化的兼容性问题、软件库依赖冲突问题、冗余依赖问题与安全漏洞传播问题. 图 5 描述了演化行为与依赖缺陷问题的因果关系, 本节针对 4 种备受研究者关注的问题分别展开介绍.

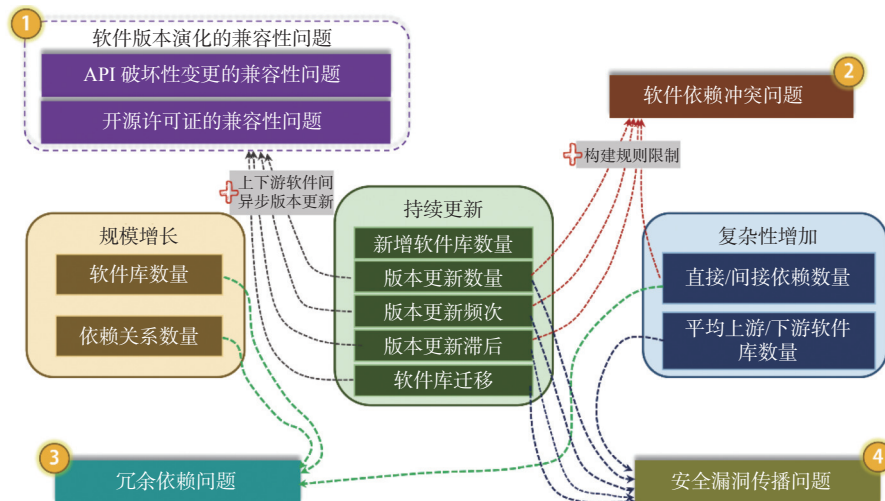


图 5 开源软件库生态演化行为导致的依赖缺陷问题

6.1 软件版本演化的兼容性问题

在开源软件库生态中, 软件项目与其依赖的第三方软件库分别由不同的开发者实现与维护, 因此上下游软件之间的异步演化, 时常会导致兼容性问题. 具体而言, 软件库的演化不仅包括功能的增加, 还涉及多维度属性的变更, 如方法签名、许可证和配置文件等. 从收集到的现有文献看来, 研究者近年来聚焦于软件库异步演化过程中 API 破坏性变更 (40 篇) 和开源许可证相关的兼容性问题 (29 篇). 通过提炼文献中的相关概念和描述, 我们对软件版本演化的兼容性问题给出如下释义.

6.1.1 API 破坏性变更的兼容性问题

问题定义: 在版本演化的过程中, API 的 4 种变更会破坏软件库的向后兼容性, 包括 API 签名变更、API 实现变更、API 弃用和 API 删除. 软件项目在对第三方软件库版本升级之时, 上述变更更容易引发构建或运行时刻的异常行为^[153,154].

问题影响: 针对软件库版本演化过程中 API 破坏性变更所带来的影响, 我们从生态社区和软件项目两个角度进行分析: (1) 对生态社区的影响: Brito 等人^[155]的研究表明, NuGet 生态是出现 API 破坏性变更问题最多的社区, 而问题数量排名第 2 和第 3 名的生态社区分别是 npm 和 Maven. 其中, npm 构建规则允许不同版本的同一软件库共存于软件项目中, 因此 API 破坏性变更在该语言社区的软件库生态上不会造成“涟漪效应”. Zhang 等人^[156]发现 API 破坏性变更在 Python 框架中发生得更加频繁, 并且可以发生在软件演变的任何阶段, 而 Python 的动态特性和灵活性使得更改不易被发现, 因此 Python 框架中的 API 破坏性变更已经影响了广泛的客户端应用软件, Java 相对

来说影响较小。(2)对软件开发者的影响:不同的程序语言生态依赖管理机制使得开发者在 API 破坏性变更问题防范的意愿方面有很大的差异。虽然软件库的开发者尽量避免 API 的破坏性变更,但仍然不可避免地以演进的名义进行向后不兼容的 API 修改,持续性地与技术债务作斗争。文献 [157] 指出,R 语言的 CRAN 软件存储仓库与其他生态的不同之处在于,它要求软件库开发者在提交 API 不兼容性修改之前通知依赖于它的软件项目,让下游开发者按照自己的开发计划更新,对版本号修改更为宽容。以此方式进行版本异步演化行为的协调,使得开发者之间保持持续同步,有效地避免兼容性问题。R 语言开发者针对 API 破坏性变更的治理策略与经验,值得向其他语言社区的开发者推广。

问题类型:根据软件库版本演化中 API 变更的方式,我们将其引发兼容性问题的原因分为如下 3 类。

- API 签名变更:软件库中 API 签名(包括方法名,参数类型和返回值类型)的变更,会导致二进制不兼容或源代码不兼容:二进制不兼容是在程序链接时出错,源代码不兼容是程序编译时出错^[158,159]。

- API 实现变更:在软件库中 API 签名不变的情况下,改变方法或字段的实现(例如前置条件、默认值),同样也会引起向后不兼容的程序行为^[160,161]。Mostafa 等人^[162]通过对软件库演化过程中 API 实现变更的研究,根据不兼容的行为、调用条件等因素对程序向后不兼容原因进行了分类,发现向后不兼容问题在 Java 软件库中普遍存在。Dig 等人^[163]研究了 Java 社区流行框架和软件库中 API 实现的演化历史,发现在 80% 的情况下,API 破坏性变更是由软件重构操作导致的。

- API 弃用与删除:在软件库版本演化过程中,API 的弃用与删除会导致兼容性问题,甚至在开源软件库生态中会引发“涟漪效应”^[164]。API 弃用是其删除前的警告方式,但实际上很多软件库开发者并没有严格地给出 API 弃用警告信息,从而造成软件运行崩溃的情况。Robbes 等人^[165]是首先调查软件库中 API 变更对依赖于它的下游软件项目兼容性影响的研究者,通过对 Squeak 和 Pharo 软件生态的研究得出结论,由 API 弃用可能会对开源软件库生态造成破坏性的影响。Qiu 等^[166]通过对大量 Java 软件项目的分析发现 51.1% 的弃用类、43.5% 的弃用方法和 18.1% 的弃用字段均在实际开发中被采用,会引发相当规模的兼容性问题。而 Sawant 等人^[167,168]在文献 [165] 的基础上,又增加了对 Java 生态社区的研究,证实大多数客户端软件项目不会对弃用实体做出反应,即便做出反应,也是倾向于通过删除有问题的 API 调用来解决。同时,他们认为,Java 中 API 的弃用机制没有为开发人员提供正确的激励作用。

在发生如上 3 种 API 破坏性变更的情况下,为避免对开源软件库生态造成传播式影响,开发者应遵守语义版本控制规则,使用主版本(major),次要版本(minor)和补丁(patch)版本号来表示 API 的破坏性变更,非破坏性变更与缺陷修复的版本演化^[169]。然而,实际开发中软件库语义版本的更新不规范是兼容问题暴露的首要原因。Raemaekers 等人^[170,171]及 Zhang 等人^[172]研究指出,语义版本更新不正确是软件生态中的常见问题。调查发现,平均每个流行项目有 30 个向后兼容性问题,并且在本应该满足向后兼容性的补丁版本和次要版本中出现了 API 破坏性变更。

6.1.2 开源许可证的兼容性问题

问题定义:开源许可证是一种法律许可,一般会以副本或声明的形式附带在开源软件中,表明允许用户可以免费地使用、修改和共享版权软件^[173]。开源许可证间最大的差异是许可证对分发衍生软件的限制性不同,即他人对代码修改和扩展后分发的限制要求存在差别。开源许可证按照限制的强弱通常分为 3 种类型:宽松型、限制型和弱限制型^[174]。宽松型许可证(permissive permission)对软件的再分配方式提出了最低要求,它对衍生软件没有过多限制,允许它成为私有软件,如 BSD 系列许可证、MIT、Apache 2.0 和 MulanPSL-2.0。限制型(copyleft permission)许可证条款要求对软件的修改和扩展必须按照获得软件的许可证进行开源,如 GPL 系列许可证和 OSL 系列许可证。弱限制型许可证要求对软件的修改和重新分发必须按照获得该软件的许可证进行开源,然而合并这些软件的大型作品可以成为私有软件,如 MPL 和 EPL 许可证。在开源软件库异步演化的过程中,随着上下游软件对许可证的添加或移除、许可证升级或降级、许可证变更等操作,在同一个软件项目很容易引入授权条款冲突的源文件或第三方软件库,引发许可证兼容性问题^[175,176]。

问题影响:结合图 6 中描述的常用开源协议条款兼容性指示图,我们可以从两个方面描述许可证兼容性问题给软件项目带来的影响。

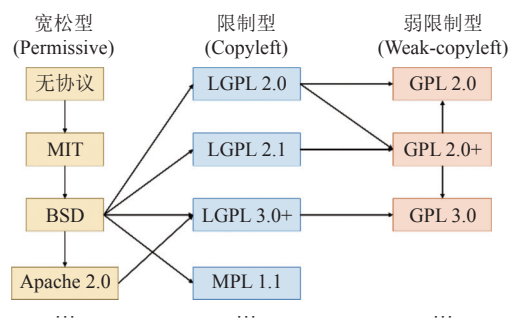


图6 常用开源协议条款兼容性指示

● 许可证的决定: 一般来说, 限制型许可证可以向下兼容弱限制型许可证和宽松型许可证。在图6中, 用直线相连的两个许可证协议是兼容的, 箭头指向为限制性更严格的许可证, 即直线相连的许可证协议可以组合使用, 且软件项目最终的许可证取决于强限制型协议^[173]。例如, 采纳有向路径 MIT→BSD→Apache 2.0→LGPL 3.0 中任意两个协议的开源软件都可组合使用, 而软件项目最终的许可证取决于箭头末端的 LGPL 3.0 协议。

● 许可证的冲突: 限制条件矛盾的两个许可证协议互不兼容。在图6中, 没有直线相连的两个许可证协议互不兼容, 即两种许可证不可组合使用。例如, MPL←BSD←Apache 是一个双向链路, 链路两端的 MPL 和 Apache 是不兼容的, 无法组合使用。

由于许可证兼容性问题的频发性, 它给软件项目的开发者带来3个方面的影响。

● 法律纠纷: 开源许可证是第三方软件库的作者与软件项目开发者之间具有法律约束力的合同。若开发者误用或误解许可证协议, 在违背法律条款的情况下使用软件库, 将会给开发组织机构或公司带来法律风险。例如, 2017年 CoKinetic 公司在纽约联邦法院向 Panasonic Avionics 公司发起诉讼, 原因是其违反了 GPL 2.0 开源许可证协议。根据 CoKinetic 的诉讼, Panasonic Avionics 公司的软件开发基于支持 GNU 通用公共许可证的 Linux 操作系统。GNU GPL 协议要求, 该许可证要求对软件的修改和重新分发必须按照 GPL 许可证进行开源, 然而, Panasonic Avionics 公司为了打压商业竞争对手, 拒绝开源其操作系统源代码。CoKinetic 声称这是故意违反 GPL 2.0 许可证的行为, 要求赔偿1亿美元的版权费用^[177]。

● 软件项目许可证的选择: 开源许可证协议是软件项目的开发者向下游用户传达软件版权的条款。软件项目的开源许可证一旦选定, 通常不会在项目的生命周期内改变。因此, 许可证的选择可能影响该软件项目的成败。开发人员会时常面临软件项目许可证选择的难题^[178]。Sen 等人^[179]采用效用最大化的方法研究了软件开发者对开源许可证的最优选择。他们调查了开发者对许可证的态度, 发现在许可证的选择方面投入的思考和努力越大, 在后续演化中, 软件项目许可证不受限制的可能性就越大。即使所有开源软件开发者都更倾向于限制型许可或弱限制型许可证协议, 这一结果仍然成立。

● 第三方软件库许可证的选择: 当软件项目开发者选择使用第三方软件库时, 他们必须了解软件许可证协议条款与其使用预期是否匹配。在错综复杂的开源软件库生态依赖关系下, 来自第三方软件库的多样化许可证协议对软件项目的影响和约束变得愈发难以理解。例如, Almeida 等人^[180,181]针对3种流行开源许可证 (GNU GPL 3.0、GNU LGPL 3.0 和 MPL 2.0) 单独使用和组合使用的开发场景, 在375名软件从业者中开展调查。结果表明, 只有约62%的开发者能够在42个许可证使用实例中给出与法律专家相同的判断。尽管开发者清楚地理解单一许可证的使用影响, 但当涉及多个软件库的多种许可证时, 他们通常无法给出明确的答案。

问题类型: 随着第三方软件库的引入和版本演化, 软件项目极易引入两种类型的许可证兼容性问题:

● 多许可证的兼容性问题: 在软件项目演化过程中, 极易从不同的源文件/第三库中引入多个许可证协议。在多个许可证协议条款冲突的情况下, 便会引发兼容性问题。Moraes 等人^[182]研究了 npm 生态中的多许可证现象, 分析了1552个 JavaScript 项目中引入的数百万个软件库, 发现大约有61%的项目使用了多于一个开源许可证。平均每个软件项目引入4.7个许可证 (最大为256个), 其中373个项目引入了一对协议冲突的开源许可证, 体现了多

许可证兼容性问题的普遍性.

● 许可证不一致问题: 如果被下游软件项目使用的第三方许可证协议与该软件库发布的许可证协议不一致, 则会导致项目对第三方软件库的使用侵权. 造成这种不一致问题发生的原因分为两种^[183]: (1) 软件库的开发者更新了许可证协议, 然而包含其原始许可证协议的旧版本软件库依然在下游软件项目使用; (2) 下游软件项目擅自删除或修改第三库中原始的许可证协议. 第 1 种许可证不一致问题, 随着第三方软件库版本的升级便可消除. 而第 2 种许可证不一致问题如果没有得到版权所有者的认可与批准, 便会导致侵权. 例如, 在 XimpleWare 公司起诉 Versata 软件公司的案件中, Versata 被指控将 GPL 许可的代码包含在其产品中, 但删除了 GPL 要求的版权和使用协议^[184].

6.2 软件库依赖冲突问题

在收集到的领域论文集中, 12 篇讨论了依赖冲突问题. 通过提炼文献中的相关概念和描述, 我们对问题给出如下释义.

问题定义: 软件项目在演化过程中, 很容易以直接和间接依赖的方式引入: (1) 同一软件库的不同版本; (2) 不同软件库中的同名类; (3) 软件项目与软件库中的同名类; (4) 软件库与平台框架之间的版本范围冲突, 在构建环境的约束下, 项目只能从软件库的众多版本中仲裁并加载其中一个版本而遮蔽掉其他版本, 亦或无法安装下载的软件库版本, 从而导致构建或运行时刻的异常行为^[185].

问题影响: 依赖冲突问题对开源软件库生态的影响, 可从 3 个角度进行分析: (1) 普遍性: 经验调查发现, 软件库依赖冲突问题频繁发生在 Python、Java、C#、Rust 等多个流程序语言社区, 且平均在各自生态社区的 GitHub 版本维护系统中遗留数以百万计的依赖缺陷报告^[186,187]; (2) 多样性: 依赖冲突问题的表现形式取决于软件库的不同版本在依赖配置脚本中的声明顺序、依赖树的拓扑结构、构建环境的平台版本以及构建工具的安装规则. 上述任一条件发生变化, 冲突的软件库版本被置换的方式便随之变化, 对程序行为的影响也会发生变化^[24]; (3) 严重性: 依赖冲突问题可能会导致软件构建失败 (例如 Python 和 C# 软件), 更有可能引发 ClassNotFoundException、NoSuchMethodException 等程序崩溃的异常行为或程序的语义行为发生不一致等严重后果 (例如 Java 软件)^[185].

问题类型: 在软件项目演化过程中, 第三方软件库以及项目平台框架的升级/降级/迁移时常会触发依赖冲突问题. 如图 7 所示, 根据冲突的原因, 我们可将问题分为 4 种类型.

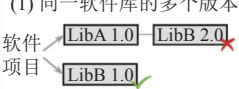
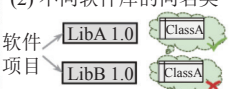
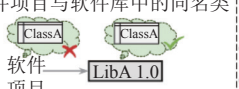
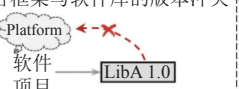
问题触发	问题原因	问题表现	问题修复
(1) 软件库升级/降级 (2) 软件库新增/迁移 (3) 平台升级/降级 (4) 平台迁移	(1) 同一软件库的多个版本  (2) 不同软件库的同名类  (3) 软件项目与软件库中的同名类  (4) 平台框架与软件库的版本冲突 	(1) 运行阶段的软件崩溃 (2) 运行阶段的语义异常 (3) 构建失败	(1) 重命名类文件 (2) 调整软件库的声明顺序 (3) 升级/降级软件库版本 (4) 定制类加载器 (5) 升级/降级平台框架版本 (6) 与上游软件库协调版本

图 7 软件库依赖冲突问题类型描述

● 同一软件库的多个版本: 若多个版本的同一软件库以直接或间接依赖的方式引入软件项目中, 构建工具会根据特定的版本仲裁策略加载其中一个版本 (Maven 和 pip 工具的“最短路径优先”和“先声明优先”, Gradle 工具的“最高版本优先”, NuGet 工具的“直接依赖优先”等), 而遮蔽掉其他版本. 研究表明, 对于 Java 项目而言, 如果加载的版本中不包含软件项目需要调用的类或方法, 则会引发运行时刻的软件崩溃^[185]; 对于 C# 及 Python 项目而言^[24,188], 若冲突软件库的安装版本不满足被遮蔽版本的依赖约束范围, 则会导致构建失败.

● 不同软件库的同名类: 若不同软件库中包含同名类文件, 构建工具则会加载依赖配置脚本中优先声明的类文件, 而遮蔽掉其他同名类. 若加载的类文件中缺失软件项目实际调用的方法, 则会导致运行时刻的软件崩溃^[185].

● 软件项目与软件库的同名类: 若软件项目与第三方软件库中包含同名类文件, 在采用 UberJar 方式 (将项目代码与第三方软件库打包在一起) 构建项目的情况下, 第三方软件库中的类文件被加载, 而软件项目自身的同名类

会被遮蔽。同理, 如果加载的类文件中缺失被实际调用的方法, 则会导引发运行时刻的软件崩溃^[185]。

- 平台框架与软件库的版本冲突: 在 C#, C/C++ 等平台框架相关的程序语言社区中, 为保证兼容性, 软件库会约束其可安装的平台框架版本。在软件项目演化过程中, 极易引发平台框架与软件库版本的冲突, 导致构建失败^[24,188]。

6.3 软件冗余依赖问题

在收集到的领域论文集中, 17 篇讨论了软件冗余依赖问题。通过提炼文献中的相关概念和描述, 我们对问题给出如下释义。

问题定义: 第三方软件库以直接或间接依赖的方式引入软件项目中, 但是却只有部分特性或函数被实际调用。我们将软件项目在构建和运行过程中不必要的代码依赖称之为冗余依赖^[189]。

问题影响: 冗余依赖主要为软件项目带来 3 个方面的影响^[190]: (1) 构建效率: 增加了构建文件的大小, 使得构建效率降低。在存储空间有限的小型设备上, 可能无法部署运行; (2) 维护代价: 冗余依赖为开发人员带来了更大的测试和维护代价; (3) 安全风险: 冗余依赖意味着更大的受攻击面, 增加了软件项目的安全风险。

问题类型: 根据代码粒度的不同, 可以将冗余依赖划分为如下 3 种类型。

- 冗余软件库: 依赖管理器通过解析软件项目依赖配置脚本中声明的直接依赖软件库, 自动化地下载并安装间接依赖软件库。上述过程在保证项目正确构建的同时, 也会引入冗余的软件库。目前对于冗余软件库问题的研究主要集中于 Maven 生态中, Soto-Valero 等人^[191,192]对 Maven 生态中冗余的依赖关系进行定性和定量分析, 研究发现冗余的依赖在 Java 项目中非常普遍, 75.1% 的软件项目均引入了冗余依赖。也有研究发现问题发生的原因除了库开发者不合理的模块化与依赖设置外, 项目开发者使用了诸如编译、执行和测试等错误的依赖作用范围也导致了冗余依赖的引入^[189]。

- 冗余类文件: 软件库在演化过程中倾向于扩展现有特性与增加新特性, 导致库的体积增大, 失去了原本的功能聚焦。然而, 实际情况下, 软件项目只会调用到第三方软件库中的部分类文件, 去除冗余的类文件依然能够保证软件的正确构建和运行。文献^[193,194]研究了软件项目中错综复杂的库间依赖关系、包间依赖关系以及类间依赖关系, 从而探索如何有效消除不必要的类文件。Fan 等人^[190]探索了依赖配置脚本中声明的依赖与源文件中实际依赖不一致而导致的冗余依赖问题, 以提高软件的构建效率。文献^[195]为减少 Web 应用程序中的安全漏洞代码, 而消除其中冗余且风险较高的源文件。

- 冗余函数/代码块: 在加载的类文件中, 也可能只有少量函数/代码块被实际调用。因此, 有选择地删除类文件中的函数/代码块是一种更细粒度的去除冗余依赖的方式, 它允许开发者进一步缩减软件项目的体积, 而不只是保守的软件库/类文件粒度的去冗余策略。为了满足复杂的用户需求或者为了支持不同的客户端软件, 第三方软件库实现了大量的功能特性, 使得体积日趋膨胀。Vázquez 等人^[196]分析了 JavaScript 应用程序中捆绑的第三方软件库, 通过去除冗余的函数来提高网页的加载速度。近期研究也分别尝试在大型浏览器^[197]、商用软件^[198]、Web 应用程序^[195]和 Unix 应用程序^[199]中删减不必要代码或者函数来减少安全风险, 提升构建和运行效率。

6.4 安全漏洞的传播问题

在收集到的领域论文集中, 90 篇讨论了安全漏洞的传播问题。通过提炼文献中的相关概念和描述, 我们对问题给出如下释义。

问题定义: 攻击者时常在流行的软件库中注入恶意代码, 或者软件库中的不良设计被攻击者所利用, 以达到窃取用户隐私信息^[200,201]、获取不当牟利^[202]、提升权限^[201]或拒绝服务^[203]等目的。具有安全风险的第三方软件库可能携带未被公开披露的安全漏洞, 也可能包含已知的安全漏洞, 它们随着开源软件库生态的依赖关系, 以直接或间接的方式^[33,204]传播进大量的下游软件项目中, 对软件项目的可信性带来威胁。

问题影响: 安全漏洞的传播对开源软件库生态的影响, 可以从以下 3 个方面进行分析。

- 安全风险: 在软件项目中引入包含安全漏洞的软件库可导致大规模用户信息、密码等隐私泄露或发生异常行为, 造成重大事故或经济损失。例如, OpenSSL 开源密码库在 2012 年公开纰漏的 Heartbleed 漏洞使得 24%–55%

使用网络传输层安全协议的流行网站受到攻击^[72,205,206]。安全漏洞允许攻击者远程访问私钥和密码,导致用户信息泄露^[207]。此外,攻击者可以通过替换广告库,从应用开发者方面添加恶意功能,从应用用户方面窃取信息,造成经济损失^[36]。研究表明,约 50.6% 的安全漏洞是在公开宣布后才修复的,这可能会增加其被攻击者利用的可能性^[71]。

- 影响范围:流行的软件库不仅会影响直接依赖于它的软件项目,还会以传递性依赖的方式影响到大量的下游软件^[208]。根据 Zimmermann 等人^[43]的调查表明,在开源软件库生态中,单一软件包平均影响了 230 个下游软件,排名前 5 的软件库分别影响了 134 774–166 086 数量不等的下游软件,最终成为生态级别的安全漏洞问题。由于传递性依赖的数量远远大于直接依赖软件库的数量^[77],目前在多个程序语言开源软件库生态中均造成了大规模的安全漏洞传播^[77]。

- 修复代价:对于软件项目开发者而言,安全漏洞问题的修复方案可能是:(1) 升级/降级直接或间接引入安全漏洞的软件库^[207];(2) 迁移到具备相似功能的可靠软件库^[207];(3) 开发者自己修复漏洞并维护软件库^[209]。大部分的安全漏洞可以使用更新直接依赖的方式解决,然而第三方软件库的更新或迁移可能会引入兼容性问题(详细内容可见“6.1.1 API 破坏性变更的兼容性问题”),需要开发者付出更大的努力来测试^[49,74,107]。对于服务关键业务功能的大型企业软件系统,任何更改都可能会导致系统停机,并带来不可预见的损失和风险。

问题类型:根据软件库中的安全漏洞是否被公开披露,我们将问题分为如下两种类型。

- 0-Day 安全漏洞:软件库中可被攻击者利用的未被公开披露的安全漏洞。0-Day 安全漏洞的常见攻击手段包括:(1) 攻击者加窃取了软件库开发者凭证,恶意地注入漏洞代码^[210–212];(2) 攻击者将自己开发的漏洞软件库伪装成与流行软件库相似的命名^[213–216];(3) 攻击者利用流行软件库自身包含的漏洞代码特点进行攻击^[217,218]。常被攻击者利用的漏洞代码包括:① 字段暴露^[219,220]:通过方法调用读取或修改私有字段;② 权限过度^[49,107,202,221–232]:访问过多的方法或数据;③ 生命周期滥用^[217]:服务注销之后(或在服务注册之前)可对对象进行访问;④ 拒绝服务^[77]:阻止客户端或服务器的正常访问。

- 1-Day 安全漏洞:软件库中可被攻击者利用的已经公开披露的安全漏洞。当软件库中的安全漏洞被披露时,开发者可能会延迟发布补丁版本,亦或发布的补丁版本由于上游软件的更新滞后而阻塞在开源软件库依赖链路中,依然对软件项目造成安全威胁^[97,100]。

研究问题 3 结论总结:在开源软件库生态多样化的异步演化行为、构建环境限制、依赖配置模式复杂性等因素的相互影响和制约下,会导致 4 种常见的依赖缺陷问题,现有文献对其开展了深入研究:(1) 软件版本演化的兼容性问题(包括 API 破坏性变更 39 篇和开源许可证相关的兼容性问题 28 篇);(2) 软件库依赖冲突问题(11 篇);(3) 冗余依赖问题(12 篇);(4) 安全漏洞传播问题(80 篇)。

7 开源软件库生态的治理技术

围绕前文介绍的 4 种依赖缺陷问题,我们综述其相应的检测及修复技术。

7.1 软件版本演化兼容性问题的治理技术

7.1.1 API 破坏性变更兼容性问题的治理技术

在开源软件库生态的宏观视角下,在软件库异步演化的过程中,鼓励开发者使用语义版本控制规则来标识兼容性变更是有效地对抗 API 破坏性变更问题的治理策略^[108,233–237]。然而,对语义版本控制规则的尊重程度可能因不同程序语言生态的特性和开发者的行为而异^[157,165,238–240]。Bogart 等人^[157]的研究说明“长期稳定”是 Eclipse 社区的核心价值观,不允许打破变化;npm 社区重视上游开发者的“轻松部署”,接受破坏性变更,并且对快速变化持开放态度;CRAN 社区则倾向于与最新版本软件库之间的兼容性,核心价值是使客户端项目可以轻松地安装最新的库版本。近期研究^[165,238,239]表明,受到生态社区价值观的影响,总体看来 API 通常是不稳定且向后不兼容的,同时 API 破坏性变更的情况随着时间的推移呈现显著增加趋势^[240]。在软件库 API 破坏性变更频繁地引发兼容性问题的现象背后,开发者和维护者都在不断地做出回应,研究人员对治理技术的探索也在不断地深入。如表 7 所示,针对 API 变更的不同方式,我们综述其相应的治理策略。

表 7 API 破坏性变更导致的兼容性问题治理技术的研究现状

问题类型	研究内容	关键技术	文献
API 签名变更	识别Eclipse/Apache社区的API签名变更	实践调查、静态分析	ACUA ^[241]
	检测Android软件库中API不兼容用法	实践调查、数据流分析	IctApiFinder ^[242]
	捕捉Python社区API签名的变更方式	静态分析、实践调查	PYCOMPAT ^[156]
	定位API破坏性变更代码的位置	静态分析、模式匹配	Muller等人 ^[243] 、Jsfix ^[244] 、Maracas ^[245]
API 实现变更	检测API签名变更引起的兼容问题	实践调查、模块化测试	Api-compliance-checker ^[246]
	跨版本检测API的行为不兼容问题	回归测试、实践调查、聚类算法	Mostafa等人 ^[162]
API 弃用与删除	跨项目检测API行为不兼容问题	信息检索、回归测试、实践调查	Debbi ^[247]
	调查Python社区API弃用现状与治理建议	实践调查	DLocator ^[248]
	调查Java/C#社区中API弃用的影响与治理建议	实践调查	Brito等人 ^[249]

(1) API 签名变更兼容性问题的治理技术. 观察、理解与分析软件库演化过程中 API 的实际变更方式, 是有效地检测、修复和预测 API 变更问题的理论基础. Wu 等人^[241]提出了工具 ACUA 自动化地分析 Eclipse 和 Apache 社区中 22 个开源框架的版本演化, 以识别相邻两个版本间的 API 签名变更. 在此基础上, 他们提出了能够有效降低 API 签名变更对开源软件库生态影响的若干编码规范. He 等人^[242]和 Liu 等人^[250]通过调查不同 Android 平台版本与海量 Android 应用的演化历史, 识别 Android 平台版本引入的新类型、新方法以及新属性. 然而, 目前 Android 社区的软件库仅支持 23% 所有的演化特性, 导致兼容性问题频繁发生.

在了解软件库中 API 签名变更常见模式的基础上, 研究人员开发了一系列有效的工具帮助软件项目开发者检测以及修复兼容性问题. Zhang 等人^[156]针对 Python 社区中 API 演化的特点, 开发自动化工具通过静态分析捕捉 API 重命名、参数增删/类型变更/声明顺序变更、参数默认值变更等因素导致的兼容性问题. 当客户端软件项目需要适配软件库中的 API 签名变更时, 面临的主要挑战是如何在软件项目中定位到受变更影响的 API 调用. Muller 等人^[243]面向 JavaScript 软件项目, 提出一种用于表示 API 访问点的简单模式语言和基于轻量级静态分析的模式匹配工具来解决这一难题. Nielsen 等人^[244]开发工具 Jsfix, 借助于静态分析来定位 API 破坏性变更影响的位置, 进而建议开发者对定位的代码进行适配, 使其与新版本的库兼容. Ochoa 等人^[245]面向 Java 软件项目提出了一个静态分析工具 Maracas, 来计算软件库的两个版本之间的 API 破坏变更集合, 进而在客户端软件项目中检测受到这些变更影响的代码位置. Ponomarenko 等人^[246]面向 C/C++ 软件项目开发工具 Api-compliance-checker 自动化检测 API 签名变更引起的兼容性问题. 与现有方法不同的是, 除了分析软件库二进制文件中的符号外, 还通过比较从软件库头文件中解析的函数签名和类型定义来验证广泛的向后兼容性问题. Dietrich 等人^[158]调查发现, 当整个软件项目被编译和测试时, 客户端代码和它使用的第三方软件库之间的一致性会在构建时被检查. 然而, 只通过重新部署已演化的软件版本来部分升级软件项目, 会导致构建工具跳过这些关键的验证步骤. 针对这一问题, 他们调查了编译器和虚拟机中用于处理软件库 API 的提供者和使用之间契约的不同规则集, 以检测在这种场景下 API 签名变更引发的兼容性问题.

(2) API 实现变更兼容性问题的治理技术. 即使 API 签名在软件库的版本演化中保持相同, 该方法内部实现的变更也有可能引发不兼容的程序行为. 为准确地捕捉上述问题, Mostafa 等人^[162]对 Java 软件项目的相邻版本开展研究, 发现采用回归测试检测到的大多数程序行为不兼容都是由于返回变量/字段的赋值变化或异常引起的. 文献虽然对 API 实现变更引发兼容性问题开展了深入探索, 但是基于人工测试 Java 软件项目历史版本的研究结论难以泛化或扩展到其他软件项目中. 在此基础上, Chen 等人^[247]提出了 Debbi 技术, 利用 IR 模型 (information retrieval models) 和 MMR 算法 (maximal marginal relevance), 开展跨软件项目之间的测试验证. 将检测问题转化为信息检索问题, 实现更高效更广泛范围的检测程序行为的不兼容.

(3) API 弃用与删除引发兼容性问题的治理技术. Wang 等人^[248]系统地调查了 Python 社区中 API 弃用问题的现状, 以全面了解: ① 开发者是如何在源代码中声明或者在开发文档中注明被弃用的 API; ② 软件库的使用者如何处理被弃用的 API. 在此基础上, 为 Python 社区中 API 弃用问题的治理方式, 提出了建设性意见. 结果表明, 多

数情况下, Python 软件库中对 API 弃用的处理不妥当, 开发者仅采用临时策略来进行声明, 极易引发兼容性问题. 研究人员强烈建议 Python 社区能够开发一种通用的面向软件项目开发者和软件库贡献者的 API 弃用范例. Brito 等人^[249]深入调查了 622 个 Java 和 229 个 C# 软件项目中弃用 API 的治理策略, 结果表明 66.7% 和 77.8% 的 API 在弃用时开发者都及时注明了替换 API 的信息. 在替换 API 的提供策略方面, Java 和 C# 软件对 API 弃用的管理显著优于 Python 社区. 在软件库贡献者未提供弃用 API 替换信息的情况下, 文献 [251–253] 呼吁研究人员开发自动化弃用 API 管理工具, 通过挖掘下游软件开发者对弃用 API 解决方案的历史记录, 来推断与之对应替换 API 信息.

7.1.2 开源许可证兼容性问题的治理技术

German 等人^[254]、Di Penta 等人^[255]、Scacchi 等人^[256]和 Tuunanen 等人^[257]做出了奠基性的研究工作, 帮助开发者自动识别软件项目中引入的多样化许可证协议. Kapitsaki 等人^[258]开发了工具 findOSSLicense 向软件项目开发者推荐适当的许可证协议. 在此基础上, 面向不同类型许可证兼容性问题的治理技术研究才陆续开展.

(1) 多许可证兼容性问题的治理技术. 为有效地避免由第三方软件库使用而引入的多许可证兼容性问题, 构建工具应当为开发者提供有效的许可证分析技术^[259]. Kapitsaki 等人^[260]对现有的开源许可证分析工具进行评估对比, 发现工具的能力大致分为 3 种类别: ① 从源代码或二进制代码中提取许可证信息; ② 解析代码仓库中的软件元数据; ③ 许可证兼容性建模与相关推理操作. 作者通过形式化描述兼容性问题, 详细地阐述了现有工具和技术在许可证选择决策中的作用, 为软件开发者提供了引导.

采用不同开源许可证的软件库间依赖关系很容易引发兼容性问题. 文献 [261] 中开展了关于 GPL 许可证与其他开源许可证组合使用的实践调查. 结果发现, 目前开源软件开发者已经找到了很多有趣的方法来创建 GPL 软件的衍生制品, 使得发布的软件制品可以在法律上规避 GPL 协议的限制. 文中记录了这些方法, 并警示开发者开源许可证正在以复杂且意想不到的方式相互作用.

Kapitsaki 等人^[262]的研究通过检查软件库之间数据交换的结构来捕获许可证兼容性问题, 以帮助软件项目开发者找到第三方软件库之间的正确组合使用方式. 研究人员还通过开源社区中代表性的案例详细地阐述了软件产品中许可证检测的复杂性和模糊性. 最终提供了工具 SPDX-VT, 在许可证使用和兼容性方面, 为软件的自动化分析提供技术支持.

Xu 等人^[263]开发了工具 LiDetector, 可以提取和解释开源软件许可证 (包括官方许可证和自定义许可证), 并检测这些许可证之间的许可证不兼容性. 具体来说, LiDetector 引入了一种基于学习的方法, 从任意许可中自动识别有意义的许可条款, 并使用概率上下文无关语法 (probabilistic context-free grammar, PCFG) 来推断不兼容性检测的权利和义务.

近年来也涌现出采用形式化建模的方式来检测多许可证的兼容性问题的系列研究工作^[241–243].

- 软件库-许可证依赖图. German 等人^[264]将软件项目中第三方软件库的依赖关系与其声明的许可证相结合, 来进行问题诊断. 利用该模型, 在 Linux 的 Fedora 操作系统社区进行了大规模实证研究, 结果表明: (1) 软件库声明的许可证协议与其源代码文件声明的许可证存在不一致; (2) 上下游软件库之间许可证协议存在相互冲突. 在工作^[241]的基础上, German 等人^[265]进一步改进了建模方式, 将软件项目抽象成一个以软件库为节点, 软件库间依赖关系为边的有向依赖图. 值得注意的是, 模型中的软件库是多种粒度 (方法、类或模块) 的软件制品, 每个软件库均有各自声明的许可证协议; 模型中的边能够描述包括链接、I/O 重定向与管道、继承和进程间通信这 4 种类型的软件库间交互关系. 该模型形式化描述了现代软件集成不同许可证软件库的模式, 也通过实证研究证明了法律协议对应用程序架构的影响. 此项研究旨在让缺乏法律专业知识的软件从业者可以使用这些文档化的模式来理解和解决重用具有不同或可能相互冲突许可证的软件库所涉及的法律问题.

文献 [266] 提出了一种建模方法, 同样以软件库为节点, 而模型中的软件库可以为源代码软件库、二进制软件库、软件服务、API 和配置的系统; 边定义为系统构建时的动态链接及客户端-服务器链接. 基于上述模型分析多许可证之间的不兼容问题, 并对许可证的选择提出建议. 指导开发者在这样的部署环境中获取、集成并开发开源软件.

- 构建依赖图. 文献 [267] 提出了通过跟踪构建时发生的系统调用来建模依赖图 CBDG (concrete build

dependency graph) 的方法. 模型记录构建过程中每一步都读和写了哪些文件. 具体来说就是追踪操作系统的调用, 即在构建过程中用户进程对操作系统内核的调用. CBDG 的节点分为任务节点 (构建过程中的步骤) 和文件节点 (构建过程中读取或创建的文件), 并为 CBDG 中的文件节点注释上它们各自的许可证, 最终通过分析 CBDG 来识别客户端软件项目中的多许可证下的兼容性问题.

(2) 开源许可证不一致问题的治理技术. 通常情况下, 源文件的许可证声明只能由其版权所有人修改, 除非得到著作权人的许可或许可条件的允许, 否则开发者的擅自修改将会面临潜在的法律风险^[268,269]. Wu 等人^[270]是领域内首先对许可证不一致问题进行深入调查的研究人员. 通过实证研究, 他们对许可证不一致问题给出了描述和分类: ① 两个文件中只有一个包含许可证: 通常由许可证的增加或删除引起的问题; ② 两个文件包含同一许可证的不同版本: 通常由许可证的升级或降级导致的问题; ③ 两个文件包含不同的许可证: 通常由许可证变更引起的问题. 文中开发了一种针对上述问题的自动检测技术, 并将其应用于 Linux Debian 7.5 中, 发现了很多许可证不一致现象, 以此引起开发者的关注.

在此基础上, Wu 等人^[184]对数据集进行了扩展, 在 Debian 7.5 和 GitHub 上的 10514 个 Java 软件项目中检测许可证不一致问题, 也发现了同样的现象. 在文献^[271]中, 他们进一步调查了许可证不一致的演化模式, 并将他们的检测方法应用于 Debian 8.2 和 Debian 7.5. 在两个版本中捕捉到 4062 组许可证不一致问题, 其中, 2701 组只存在于 Debian 7.5 中, 2947 组只在 Debian 8.2 包含. 通过观察对比两个版本, 发现大部分许可证不一致都可以通过升级上游软件库的版本而解决.

此外, 为了解决多许可证兼容性问题, 有时也可通过定义许可证例外, 来为许可证的使用添加特定条件. 例如 GNU 许可证定义了 GCC 运行环境库的例外, 目的是允许开发者使用 GCC 编译非 GPL 的软件, 前提是他们不在编译器中使用专有的插件或其他扩展. Vendome 等人^[272]对超过 5 万个开源软件项目的演化历史进行了大规模实证研究, 定量地调查已知许可证例外的流行度并识别新的许可证例外. 他们通过机器学习模型, 面向 6 种程序语言软件中的许可证例外定义进行识别和分析, 在其中 5 种程序语言软件的 298 个文件中发现了 14 种不同的许可证例外类型, 它们有效地解决了许可证兼容性问题.

7.2 软件库依赖冲突问题的治理技术

在面向 Java^[185,186], Python^[24], C#^[188]和 Go^[187]等多种程序语言依赖冲突问题的系列工作中, 研究人员发现并总结了开发者经常采纳的 6 种问题修复的原子操作: (1) 重命名冲突的类文件, 使得多个版本的类文件可以同时被加载; (2) 调整软件库在依赖配置脚本中的声明顺序, 使得包含全部被调用方法的版本可以优先被加载; (3) 升级或降级软件库的版本以化解冲突; (4) 定制类的加载器, 使用不同的加载器加载冲突的类; (5) 升级或降级平台框架的版本以化解与软件库的版本约束冲突; (6) 对于没有找到合适修复方案的问题, 开发者请求上游软件库的开发者协调版本以化解冲突. 如表 8 所示, 我们详细介绍面向不同程序语言生态的依赖冲突问题检测与修复方法.

表 8 依赖冲突问题治理技术的研究现状

语言社区	研究内容	关键技术	文献
Java	依赖冲突问题检测与严重性评估	实践调查、静态分析、依赖解析	Wang 等人 ^[185]
	依赖冲突问题的动态测试技术	条件突变、测试用例生成	Wang 等人 ^[186]
	依赖冲突问题引入的语言行为不一致检测	静态分析、差异测试	Wang 等人 ^[273]
	多模块软件项目的库版本不一致检测	软件库版本协调技术	Soto-Valero 等人 ^[192]
Python	库版本动态演化引发的依赖冲突实时检测	实践调查、生态依赖建模与演化监控	Wang 等人 ^[24]
C#	依赖冲突的自动化修复技术	二进制整数线性优化	Li 等人 ^[188]
JavaScript	由相同命名空间导致的依赖冲突检测	软件库加载动态分析	Patra 等人 ^[274]
Go	上下游软件库依赖管理模式冲突检测	实践调查、生态依赖建模与演化监控	Wang 等人 ^[187]
C (Linux 社区)	化解依赖冲突问题的软件库安装策略	软件库依赖解析技术	Abate 等人 ^[275,276] 、Vouillon 等人 ^[277,278] 和 Artho 等人 ^[279]

(1) Java 依赖冲突的治理技术: Wang 等人^[185]通过对 Apache 社区中真实依赖冲突的案例实证研究, 阐述了依赖冲突问题的普遍性和严重性. 系统地总结了在 Maven 构建规则下依赖冲突问题发生的原因、表现形式和修复策略. 结合软件库依赖解析技术、程序静态分析与实践调查经验, 开发了自动化工具 Decca, 来帮助开发者诊断软件项目依赖冲突问题的严重等级 (是否会引发软件崩溃) 以及修复和维护的成本. 在此基础上, 又进一步设计了工具 Riddle^[186]和 Sensor^[273], 借助条件变异等程序分析策略自动化地生成测试用例, 帮助开发人员自动化地触发依赖冲突引发的软件崩溃以及语义行为不一致问题, 并收集堆栈记录等问题诊断信息. 工具 LibHarmo^[280]针对多模块的 Java 项目, 自动化检测并帮助开发者修正每个子模块中对软件库版本声明不一致的问题, 以有效地避免依赖冲突问题.

(2) Python 依赖冲突的治理技术: Python 语言的依赖管理使用版本约束范围来引用第三方软件库, 根据构建工具 pip 的安装规则, 只选择安装满足软件库约束范围的最新版本. 这就意味着, 即使软件项目的依赖配置脚本不发生更改, 软件库的依赖拓扑结构也会随着直接或间接依赖软件库的版本更新而发生动态演化. 这样的软件库依赖树的自演化行为, 时常会引发依赖冲突问题^[24,281], 并且依赖冲突会伴随开源软件库生态的依赖关系导致到众多下游软件项目的构建失败. Wang 等人^[24]为了捕捉上述问题, 开发了工具 Watchman 对 Python 开源软件库生态的依赖关系进行建模, 并持续增量式地监控生态中任意软件库的版本演化及其引发的依赖冲突.

(3) C#依赖冲突的治理技术: 在.NET 平台版本严重碎片化的背景下, 同一软件库的相同版本在不同.NET 平台环境下可能会引入不同的传递性依赖软件库; 同一软件库的不同版本可能会兼容不同的.NET 平台环境, 也可能引入对.NET 平台版本限定不同的传递性依赖软件库. 因此, 在对第三方软件库亦或.NET 平台版本升级之时, 时常引入互不兼容的库版本组合, 带来“依赖地狱”问题. 通过挖掘 C#软件库依赖管理配置文件的变更历史, Wang 等人^[188]通过经验调查发现近 1 万次第三方软件库版本的修改记录中, 为升级其中一个软件库版本, 平均要协同升级或降级 35 个其他软件库才能化解软件库依赖树的兼容性问题, 最终得到可以共存的软件库版本组合. 为此, 他们开发了工具 Nufix, 利用二进制整数线性优化模型帮助开发人员生成符合他们配置偏好的最优软件库版本组合方案.

(4) JavaScript 依赖冲突的治理技术: JavaScript 是一种不提供任何显式命名空间或模块的语言, 这意味着项目中不同的第三方软件库可能会共享同一个命名空间. 因此, 若不同的第三方软件库经常对全局空间的同一位置进行写操作, 这可能导致一个软件库的 API 被另一个软件库所修改甚至覆盖. Patra 等人^[274]通过对真实案例的研究, 总结了 4 种依赖冲突的类型, 同时显示了这种冲突在 JavaScript 项目中的普遍性. 为此, 他们开发了工具 ConflictJS^[274], 通过对库在全局命名空间的写操作进行动态分析, 识别潜在的依赖冲突, 并在不同的客户端比较库的行为来验证依赖冲突.

(5) Go 依赖冲突的治理技术: 为了避免引入依赖冲突问题, 自 2018 年起, Go 语言软件生态正在经历依赖配置模式的迁移, 从原始的 GOPATH 依赖管理模式逐步迁移到新 Go Modules 模式. 然而由于这两个依赖管理模式互不兼容, 同时 Go 语言社区中近 60% 的软件由于不支持语义版本发布规则、过度依赖第三方依赖管理器等原因依然停留在 GOPATH 依赖管理模式. 上述依赖配置模式演化会引发兼容性问题的 3 种常见场景: GOPATH 模式的项目依赖 Go Modules 模式项目、Go Modules 模式的项目依赖 GOPATH 模式项目以及 Go Modules 模式的项目依赖 Go Modules 模式项目时发生的兼容性问题. Wang 等人^[187]开发了工具 Hero 帮助开发者全面分析与诊断由于上下游软件采用异步的依赖管理模式引发的兼容性“冲撞”.

(6) C (Linux 社区) 依赖冲突的治理技术: 在 Linux 安装环境中, 时常会引入版本约束范围相互冲突的软件库组合, 无法同时进行安装. Artho 等人^[279]通过调查 Linux 主流发行版 Debian、RedHat 社区的缺陷跟踪系统, 筛选并分析其中关于依赖冲突的错误报告, 指出在 RedHat 中存在一种特殊的冲突, 即支持 32 位和 64 位不同架构的同一软件库, 引入到同一安装环境内带来的依赖冲突. 此外, 他们总结了几种频发的冲突类型, 包括资源冲突、配置冲突、软件库之间的版本约束冲突以及由于库演变产生的方法级别的冲突等. DistCheck^[275]能够识别软件仓库中无法安装的软件库, 提供详细的信息来帮助开发人员理解问题的原因以及推导出修复方案. DebianWheezy^[276]则是

通过软件库依赖关系的元数据来预测软件库在 Linux 操作系统环境下的可安装性. 针对 Linux 社区中存在的依赖冲突, Vouillon 等人^[277,278]开发了一种新技术, 使用语义保留的图论转化方法, 将大型软件仓库中的共安装问题简化到一个小型依赖图中. 同时, 他们提供了工具 OCaml^[277], 用于检测原本可以共同安装的软件库集合, 由于其版本的升级而引发依赖冲突问题, 同时提供丰富的依赖图谱对问题的修复方案进行说明.

7.3 软件冗余依赖问题的治理技术

针对不同粒度的冗余依赖问题, 现有的研究通过解析源代码/字节码和依赖配置脚本, 利用静态分析和动态测试等方式进行有效的检测与修复 (如表 9 所示).

表 9 冗余依赖问题治理技术的研究现状

问题类型	分析对象	关键技术	研究内容	文献
冗余软件库	字节码	静态调用图分析	Maven项目冗余依赖检测与消除技术	DepClean ^[191]
	字节码	静态类型检查	Maven项目冗余依赖检测技术	Jezek等人 ^[189]
	字节码	构造统一依赖图 (UDG)	构建脚本与构建系统依赖不一致检测技术	veriBuild ^[190]
冗余类文件	源代码	软件库使用方法聚类	软件库分割技术	de Matos等人 ^[194]
	源代码&字节码	反模式检测、抽象语法树分析	Java项目中的依赖破坏性的自动重构技术	CARE ^[183]
	源代码	漏洞特性-代码映射、程序用法分析	Web应用程序冗余代码消除的安全性评估	Azad等人 ^[195]
	源代码&字节码	“代码-特性”映射、程序动态执行跟踪	大型浏览器中的冗余代码消除技术	Slimium ^[197]
	字节码	静态调用图分析、AT函数剪枝	动态共享库中的冗余检测和消除技术	Nibbler ^[282]
冗余函数/代码块	源代码	静态调用图分析、程序动态执行跟踪	JavaScript捆绑包冗余检测和消除技术	UFFRemover ^[196]
	字节码	动态测试覆盖率	Java软件冗余检测和消除技术	Soto-Valero等人 ^[283]
	字节码	程序动态执行跟踪、启发式扩展流程图	商用软件中的冗余代码消除技术	Razor ^[198]
	源代码&规约说明	马尔可夫决策过程、强化学习	Unix程序中的冗余代码消除技术	Chisel ^[199]

(1) 冗余软件库的治理技术: 文献^[189,191]提出的技术, 均通过静态分析 Maven 软件项目的字节码来检测并删除冗余的第三方软件库. DepClean^[191]通过解析软件项目的第三方软件库依赖树获取引入的软件库, 通过构建静态调用图来标识软件项目实际调用的软件库, 进而返回软件库使用状态的诊断报告, 同时生成去除冗余依赖依赖配置脚本. Jezek 等人^[189]开发了一个静态分析工具, 首先利用类加载器收集非私有类及其非私有成员, 以此来识别实际使用的软件库. 进一步通过分析类的字节码来收集项目引入的软件库. 最终通过对比两个软件库的集合, 得到冗余依赖信息.

(2) 冗余类文件的治理技术: veriBuild^[190]使用设计的统一依赖图 (UDG), 利用静态和动态分析技术来统一编码构建声明和实际调用的类依赖关系, UDG 的构造使得可以通过图遍历的方式来检测依赖错误. de Matos 等人^[194]提出了一种库分割技术, 他们挖掘并建立了依赖于流行开源软件库的 Java 软件项目数据集. 根据流行软件库中类被调用的情况, 计算 Java 项目的相似性矩阵, 进而对软件库的类文件进行聚类. 目的是将软件库分割成更小的粒度, 以去除下游软件项目的冗余依赖. Viseur 等人^[183]的工作通过构建类间依赖图, 计算 4 种反模式实例, 以确定可以被删除的依赖边. 将源代码解析为抽象语法树, 以验证是可以删除或重新组织这些依赖, 进而对软件项目的结构进行重构.

(3) 冗余函数/代码块的治理技术: Azad 等人^[195]关注于 Web 应用程序中冗余代码的消除, 通过不同技术的组合来模拟应用程序的使用, 利用 XDebug 记录模拟期间触发的代码行、函数和类文件信息以评估代码删除的风

险. Slimium^[197]首先定义了 164 个安全漏洞相关的特性, 进而构建“代码-特性”映射关系. 在此基础上, 通过跟踪和分析用户访问对网站的动态访问行为, 获取执行函数列表, 最终去除容易引入安全风险的冗余函数. Nibbler^[282]通过对给定的 C 语言共享库进行反编译操作, 构造整个项目的函数调用图, 进行不可达代码检测. 借助于 AT (address-taken) 函数剪枝算法迭代分析, 最后删除所有未使用函数. UFFRemover^[196]定义了未使用的外部函数 (UFF), 并通过 JavaScript 源码进行静态分析识别出需要的模块以及其中的引入的函数, 同时对源代码进行执行跟踪, 检测得出 UFF 列表, 在软件库中删除对应的函数. Razor^[198]利用软件项目的测试用例执行并跟踪程序, 构建启发式扩展控制流图. 进而根据扩展控制流图来重写原始的字节码, 以生成支持所需功能的最精简版本. Chisel^[199]将待缩减的代码和所需功能的高级规约说明作为输入, 输出为符合规约说明的代码简化版本. Chisel 通过使用一种新的基于强化学习的方法, 大大改进了现有的程序简化系统, 加快了对简化程序的搜索, 并将其扩展到大型软件的分析中. Soto-Valero 等人^[283]利用软件项目捆绑的测试用例及测试覆盖率度量工具 JaCoCo 来精确捕获在特定工作负载下运行时调用软件自身及其第三方软件库的哪些函数及代码块.

此外, 2022 年也涌现出关于软件库依赖“坏气味”检测与分析的研究^[53,77,284,285], 均将冗余依赖定义为影响软件依赖管理与维护的不良设计, 类似软件“坏气味”与重构理论, 建议开发者对其进行优化.

7.4 安全漏洞传播问题的治理技术

针对软件库中的 0-Day 安全漏洞和 1-Day 安全漏洞, 我们综述其相应的治理技术 (如表 10 所示).

表 10 安全漏洞传播问题治理技术的研究现状

问题类型	研究内容	关键技术	文献
0-Day 安全漏洞的治理技术	检测开源软件库生态中重复出现的安全漏洞	代码特征提取与表示	SecureSync ^[286]
	研究依赖管理工具的安全漏洞		Duan 等人 ^[212]
	验证安卓应用中软件库的恶意篡改		Hu 等人 ^[218]
	Python 软件打包分发过程中的恶意篡改		LastPyMile ^[287]
	检测面向服务 Java 和 NodeJS 软件库的安全漏洞	静态污点分析、动态污点分析	Goichon 等人 ^[288] 、Staicu 等人 ^[289]
	测试安卓应用中软件库的安全漏洞	静态分析、动态测试	Bhoraskar 等人 ^[290]
	根据已知的恶意软件库特征来检测未知恶意软件库	机器学习	Amalfi ^[216]
	检测识别未知来源的第三方软件库及版本	白名单、特征匹配	文献[291-318]
1-Day 安全漏洞的治理技术	统计分析多生态漏洞库传播现状	实证研究	Wang 等人 ^[78] 、Koishybayev 等人 ^[319] 、Lauinger 等人 ^[320] 、Alfadel 等人 ^[71] 、Zerouali 等人 ^[69] 、Cox 等人 ^[82] 、Zhao 等人 ^[321] 、Xu 等人 ^[322] 、Liu 等人 ^[323] 、Latendresse 等人 ^[324] 、Haryono 等人 ^[325] 、Decan 等人 ^[326] 、Costa 等人 ^[327]
	统计分析安全漏洞延迟披露/软件库开发者漏洞修复延迟现状	实证研究	Lauinger 等人 ^[320] 、Prana 等人 ^[81] 、Alfadel 等人 ^[71]
	统计分析软件库版本更新滞后现状	实证研究	Yasumatsu 等人 ^[98] 、Decan 等人 ^[70] 、Zerouali 等人 ^[73] 、Launinger 等人 ^[320]
	安全漏洞代码可达性的预警技术	静态分析	Vulas ^[328] 、VAS ^[329] 、Wang 等人 ^[78] 、Boldi 等人 ^[330] 、Kang 等人 ^[331]
	SCA 工具的对比分析	软件成分分析	Dann 等人 ^[332] 、Imtiaz 等人 ^[333] 、Chen 等人 ^[334]
	影响开发者对安全漏洞修复决策主要因素/漏洞库修复代价度量标准	实证研究	Yasumatsu 等人 ^[98] 、Pashchenko 等人 ^[335] 、Dashevskiy 等人 ^[336] 、Zerouali 等人 ^[69] 、Cox 等人 ^[82]
	检测安全漏洞代码的可达性	代码特征提取与表示、动态测试	Vuln4Real ^[337]

● **0-Day 安全漏洞的治理技术.** 0-Day 安全漏洞的治理通常是由安全研究人员对开源生态中的软件库进行扫描分析, 检测出安全漏洞并公开披露给 CVE (common vulnerabilities and exposures) 组织机构. 0-Day 安全漏洞检测的关键技术包括代码特征提取与表示、污点分析、动态测试和机器学习.

(1) 代码特征提取与匹配: Pham 等人^[286]的实践调查表明, 随着软件重用, 大量的安全漏洞会在开源软件库生态中重复出现. 为检测重复出现的安全漏洞, 他们开发了工具 SecureSync, 将漏洞代码片段表示成抽象语法树 (AST), 进而在不同的软件中进行结构特征的相似度匹配. Duan 等人^[212]调查发现, 软件项目依赖管理器也经常被供应商利用来分发恶意第三方软件库, 给用户带来严重的安全威胁. 他们提出了一个分析框架来提取解释型语言依赖管理器的功能和安全特征, 进而结合控制流和数据流分析来研究注册表滥用问题. Hu 等人^[218]为检测安卓应用中的第三方软件库是否被攻击者所恶意篡改, 分别提取出原始软件库和应用软件库中的类文件结构信息作为“特征摘要”, 进而通过相似度匹配的方式进行验证. Vu 等人在文献^[287]中指出, Python 源代码在最后打包分发的过程中, 容易受到攻击者的恶意篡改. 针对这一问题, 他们开发了工具 LastPyMile 来对比源代码和分发软件包的特征差异, 以检测被注入的恶意代码.

(2) 污点分析: 通过分析数据流来判断源代码中哪些变量可能会受到攻击, 是第三方软件库安全漏洞检测的关键. 污点分析技术就是分析程序中由污点源引入的数据是否能够不经无害处理, 而直接传播到污点汇聚点. 如果不能, 说明软件是信息流安全的; 否则, 说明软件产生了隐私数据泄露或危险数据操作等安全问题. 文献^[288]和文献^[289]分别提出了静态污点分析和动态污点分析来检测面向服务的 Java 软件和 NodeJS 软件的安全漏洞. Goichon 等人^[288]认为, 与动态分析技术相比, 静态污点分析有更好的非侵入性、精确性和更大的覆盖范围.

(3) 动态测试: Bhoraskar 等人^[290]提出一种新的 GUI 测试的方法来精准地触发第三方软件库中的代码, 以检测应用程序中的广告库是否违反了用户隐私信息保护法. 为避免穷举测试, 他们首先利用静态分析构建安卓应用的页面调用图, 剪枝出软件库的程序执行路径进而通过动态测试的方法触发其执行.

(4) 机器学习: Sejfia 等人^[216]通过机器学习技术来学习恶意代码库的特征, 同时结合代码克隆检测技术来识别被分类器遗漏的恶意代码库. 他们开发的轻量级的工具 Amalfi 只需要几秒钟来提取代码库特征并运行分类器, 识别出了大量以前未知的恶意软件库样本.

● **1-Day 安全漏洞的治理技术.** 对开源软件库生态中已知安全漏洞的治理技术可分为漏洞库的检测、传播影响分析、智能预警、修复代价评估与可达性分析这 5 个方面.

(1) 漏洞库的检测技术: 由于安卓应用通常与第三方软件库一同打包发布, 导致用户很难从应用中识别出第三方软件库名单及其版本号, 因而难以判断应用是否引入了漏洞库. 为避免安全风险, 近年来涌现了一系列面向安卓应用的第三方软件库检测技术. 第三方软件库检测算法整体上分为两种类别.

① 基于白名单的第三方软件库检测方法: 该方法需要建立一个流行度较高的软件库标准集, 进而在反编译的安卓应用中, 将第三方软件库与标准集中的软件库名称进行匹配. Li 等人^[291,292]提出一种从海量安卓应用中根据第三方软件库的使用频次来提取公共软件库的方法. 最终在 GooglePlay 的大约 150w 个应用中收集到了 1113 通用库和 240 个广告库的白名单, 用于实现第三方软件库检测. 然而, 事实上软件库的重命名和第三方软件库的混淆技术, 极大地影响了该方法的有效性. 此外, 白名单一般结合手动分析生成, 必须经常保持更新, 所以很难生成全面的第三方软件库列表.

② 基于特征识别的第三方软件库检测方法: 该方法首先根据安卓应用的包结构或类间依赖结构解耦成不同的软件库, 这些软件库被视为候选库. 进而提取出软件库的特征信息与构建的软件库标准集做特征匹配, 最终确定其第三方软件库的名称及版本号. Woo 等人^[293]在实现上述第三方软件库检测技术时, 融入了冗余代码的消除技术, 并使用哈希函数加快了特征匹配效率. 该项技术在数十万个开源应用的 800 亿行代码的评估测试中, 均取得了较高的准确率和召回率. Titze 等人^[294]提出了 Ordol 方法, 侧重于提取语义暴露的特征, 以一种抗混淆的方式检测应用程序中的软件库版本. Liu 等人^[295]关注于广告库的特征, 提出 Pedal 工具将广告库与安卓应用程序分离. 为应对混淆安卓应用程序, Huang 等人^[296]开发了一种可扩展的两阶段检测方法 Libloom, 并提出一种新的基于熵的度量来专门处理重新包装和扁平的应用. Zhan 等人在文献^[297-299]中对比评估了主流的第三方软件库检测技

术 LibExtractor^[300]、LibRoad^[301]、LibID^[302]、LibPecker^[303]、ORLIS^[304]、Han 等人^[305]、OssPolice^[306]、LibD^[307,308]、LibScout^[309]、LibRadar^[310]、LibSift^[311]和 AdDetect^[312]. 结果表明: (a) 大多数的检测技术基于包结构来对第三方软件库软件库解耦, 无法对抗代码混淆技术; (b) 在特征提取阶段, 如果包含更丰富的语义信息 (例如, 类的依赖关系), 就能够对软件库重命名和代码混淆有更好的处理能力; (c) 现有工具通常召回率很低, 因为算法设计忽略了安卓应用和软件库的一些特色属性, 配置模式和编译机制等; (d) 现有技术的能力通常是互补的, 结合各个算法的优势可以建立一个更好的工具.

除面向安卓应用的第三方软件库检测技术外, 近年的研究也同样涌现出针对 Linux 社区的 Java 软件^[313-315]、C 软件^[316,317]和 iOS 应用^[338]的第三方软件库版本检测技术, 以追溯安全漏洞的传播影响.

(2) 漏洞库的传播影响分析: 漏洞库在多种程序语言的开源软件库生态中均广泛传播. 研究人员针对上述问题开展了多维度的实证研究, 从不同角度对漏洞库的传播现状、规律和影响进行了总结和讨论. 文献 [69,71,78,82,318-327,339,340] 针对 Java、npm、PyPI 等流行语言软件项目、GitHub 开源项目以及 Docker 容器项目进行调查分析, 结果表明, 软件项目中至少 1/3 的第三方软件库包含严重等级较高的安全漏洞. 造成上述现象的原主要原因是: ① 安全漏洞被延迟披露, 亦或在披露后漏洞库的开发者延迟漏洞修复; ② 软件项目开发者对第三方软件库的版本更新滞后.

针对安全漏洞被延迟披露/漏洞库的开发者延迟漏洞修复问题, 文献 [71,81,320] 进行了深入分析. 文献 [320] 指出, 很多漏洞库已经不再被维护, 导致软件项目开发者没有修复版本可引用. Prana 等人^[81]在 Java、Python、Ruby 开源项目的实证研究中发现, 安全漏洞大约在披露后的 3-5 个月才被软件库开发者所修复. Alfadel 等人^[71]在 PyPI 生态中调查发现, 软件库引入的安全漏洞平均需要 3 年以上的时间才会被开发者发现和披露, 漏洞库被发现时已经大约影响了 68% 的版本, 并且 51% 的漏洞库在信息公开之后才会发布相应的补丁版本. 然而, 在 npm 生态中, 漏洞库被修复延迟的现象并不明显, 大多数安全漏洞 (82%) 在披露时就已经发布了补丁版本.

针对软件项目开发者对第三方软件库的版本更新滞后问题, 文献 [70,73,80,98,99,320] 开展了进一步的实证调查, 发现不同程序语言生态的更新滞后状况有所差别. Yasumatsu 等人^[98]发现在安卓生态中, 近一半的软件库在发布最新版本后, 至少需要 3 个月的时间才会被开发者采纳, 其余软件库的滞后时间甚至要达到 10 个月以上. Decan 等人^[70]、Zerouali 等人^[73]和 Launinger 等人^[320]在 npm 生态的研究中, 得到了相近的结论, 软件库的版本更新滞后时长平均在 7-9 个月之间. 相比之下, RubyGems 生态中, 只有 10% 的软件库版本更新时间在 90 天以上, 半数之上的漏洞库可以在被披露后的一周内更新.

此外, 在文献 [322] 中, 作者结合混合程序语言分析技术调查证实: C 语言第三方软件库的安全漏洞已经随着依赖关系, 大量传播进 Python 和 Java 开源软件生态中.

(3) 漏洞库的智能预警技术: 通过漏洞库的跟踪与智能预警来及时提醒软件开发者更新到软件库的补丁版本, 能够极大程度地阻断安全漏洞的传播影响. 文献 [78,330,341] 的实证调查表明, 开发者希望漏洞库检测工具能够提供细粒度的漏洞警告, 即只针对程序执行路径能够触发到的漏洞代码发出预警. 然而, 目前学术界 (如 Vulas^[328]和 VAS^[329]) 和工业界 (如 Black Duck^[342]、GreenKeeper^[343]、Snyk^[344]和 SourceClear^[345]) 的漏洞库预警系统均只结合开源软件库的依赖关系与安全漏洞数据来提示软件项目是否引入漏洞库, 无法提供 API 粒度的分析. 为解决上述问题, Wang 等人^[78]、Boldi 等人^[330]和 Huang 等人^[79]开发了智能预警系统原型, 一方面通过构造软件项目对第三库 API 的调用图, 并检查 API 是否存在于漏洞数据库, 来判断安全漏洞的可达性. 另一方面通过统计漏洞库补丁版本中 API 的变更次数来评估软件项目的修复代价, 以便开发者在更新第三方软件库版本之时给出修复建议.

软件成分分析 (SCA) 工具越来越多地被软件开发者采用来检测与警告漏洞库. Dann 等在文献 [332] 评估了工具 OWASP Dependency Check、Eclipse Steady、GitHub Security Alerts 和 3 个商业工具 Snyk、Black Duck、WhiteSource 在漏洞警报方面的能力. 而 Imtiaz 等人^[333]以大型 Web 应用程序作为案例, 比较了 9 个行业领先的 SCA 工具生成的漏洞报告. Chen 等人^[334]通过从大量漏洞信息源中识别漏洞信息相关度, 以辅助 SCA 工具构造漏洞数据库. 结果表明, 现有的 SCA 工具都存在一定程度的漏报. 文献 [77,83,346-348] 建议开发者组合使用这些工具以全面捕捉漏洞库, 同时建议研究者未来可在两个方向深入研究: ① 建立框架和指标来识别漏洞库的误报和漏

报; ② 开发自动化技术用于持续地监控来自开源软件库生态的漏洞库传播。

(4) 漏洞库的修复代价评估技术: 虽然大部分的安全漏洞在被披露之后, 漏洞包的开发者能够在短时间内进行修复, 但在语义版本控制、更新滞后和依赖约束机制等多种因素的交互下, 漏洞补丁很容易在开源软件库生态中发生“传播阻塞”的现象^[323]。因此, 软件项目中的漏洞库通常难以及时根治, 并且升级漏洞库到其补丁版本可能带来很高的修复代价^[349]。文献^[98,335,336]的实证研究表明, 影响开发者对安全漏洞修复决策的主要因素包括软件库新版本的功能性、规模大小以及代码修改代价等。Zerouali 等人^[69]和 Cox 等人^[82]探讨了漏洞库修复代价的度量标准, 量化软件库的引用版本与补丁版本之间的差异, 目的是确保在修复安全漏洞的同时, 避免引入兼容性问题。

(5) 漏洞库的可达性分析技术: 被 SCA 工具警告的漏洞库可能并未被软件项目实际访问到。因此, 研究人员提出了更细粒度的静态分析与动态测试技术以帮助开发者检测漏洞代码的可达性, 减少不必要的漏洞库修复代价。Ponta 等人^[328]和 Plate 等人^[350]提出了自动化工具 Vulas 来完成两个任务: ① 首先, 通过挖掘软件库版本控制系统中安全漏洞的修复补丁记录, 判断软件项目引入的软件库版本是否包含漏洞代码片段; ② 其次, 结合静态分析与动态测试技术, 判断软件项目的方法调用路径是否会触发到漏洞代码片段。随后, 他们又进一步开发工具 Eclipse Steady^[337], 提出了系列标准来对比漏洞代码片段与项目代码片段抽象语法树的相似度, 以定位漏洞位置。进而利用改进的静态分析与动态测试技术诊断漏洞代码的可达性。目前该工具已经成功地用于 1500 个应用程序的 100 多万次漏洞扫描。Pashchenko 等人^[351]提出了更准确的分析框架 Vuln4Real 来识别真正被调用漏洞库, 综合漏洞补丁记录、软件项目的构建和测试信息与软件库的更新日期等多维度的信息, 对第三方软件库进行分组分析, 最终定位到对具有安全威胁的依赖软件库。为了提高漏洞可达性报告的精度, Haryono 等人^[325]和 Kang 等人^[331]利用测试用例生成方法为客户端软件库漏洞代码生成测试用例, 从而验证安全漏洞是否会被真正触发。

研究问题 4 结论总结: 通过分析和调查 4 种常见的依赖缺陷问题, 现有研究提出如下有效的治理策略: ① 针对 API 签名变更兼容性问题、API 实现变更兼容性问题、API 弃用与删除引发兼容性问题、多许可证兼容性问题以及开源许可证不一致问题的治理技术; ② 面向 Java、Python、C#、JavaScript、Go、C (Linux) 程序语言生态的依赖冲突治理技术; ③ 针对冗余软件库、冗余类文件和冗余函数和代码块的治理技术; ④ 针对 0-Day 安全漏洞和 1-Day 安全漏洞的治理技术。

8 研究趋势与展望

经过 20 年的研究与实践, 开源软件库生态治理技术取得了长足发展。尽管如此, 为保障开源软件库生态的可信性与可持续性演化, 在建模与分析、依赖缺陷问题和治理技术这 3 个方面, 研究人员仍需深入探索。

(1) 开源软件生态建模与分析能力的增强

现代软件以直接依赖的和间接依赖的方式引入混源异构的第三方软件库, 导致开源软件生态建模时依赖拓扑结构错综复杂, 软件库成分模糊不清。因此, 建模与分析技术应该考虑如下因素。

- 动态更新软件库依赖关系: 实时地捕捉并增量式地更新多种程序语言社区中开源软件库生态的依赖拓扑结构, 是开源生态治理需要解决的首要问题。

- 软件库依赖关系与公开安全漏洞/关键缺陷数据集的贯通: 在开源软件库生态依赖关系、演化行为实时更新的基础上, 贯通公开安全漏洞数据集和关键缺陷数据集等知识库, 才可实现对依赖问题的有效感知与诊断, 同时在生态视角下进行多维度治理决策的利弊权衡。开源软件生态知识库的构建可以综合 3 个维度。

- ① 全面性: 与传统工业化软件生产相比, 开源软件生产的数据开放且种类繁多、规模巨大, 包含了大量有价值的历史记录, 如源代码、版本发布信息、代码提交和修改等软件制品和过程数据, 同时包括软件问答、软件评价等知识交流和反馈数据。全面地构建知识库能够为问题诊断提供有力支撑。

- ② 关联性: 多维度开源软件生态元数据在收集和存储的过程中, 需要建立它们之间的关联记录, 以便于追溯和判断缺陷问题在开源软件库生态中的影响性及传播性。

- ③ 演化性: 开源数据具有碎片分散、快速膨胀的特点。针对多源异质、广泛关联的开源大数据环境, 需要考虑如何定期增量式补充、更新与维护软件演化信息, 提升知识库的可演化性。

(2) 开源软件库生态中依赖缺陷问题的深入挖掘与全面覆盖

针对软件版本演化的兼容性、软件库依赖冲突、冗余依赖与安全漏洞传播 4 种常见的依赖缺陷问题, 研究人员在以下关键技术仍需要做出努力与尝试。

- 精准的程序分析能力: 精细、高效的程序分析能够识别到不同粒度软件库或代码片段间的调用路径、控制关系与数据流依赖关系。上述信息有助于判断兼容性问题是否会引发程序异常行为, 哪些软件库版本组合会互不兼容, 安全漏洞会不会真正对软件造成攻击。对于软件成分分析、冗余依赖、依赖冲突等问题的诊断同样需要精准的程序分析能力, 以避免依赖缺陷问题的误报。

- 不同程序语言的差异与共性建模: 不同的程序语言社区存在不同的: ① 依赖配置模式; ② 软件库版本的约束规则; ③ 依赖资源冲突下的覆盖规则; ④ 软件中央仓库的更新与维护特性; ⑤ 依赖缺陷的表现形式。针对上述 5 个方面的差异和共性进行建模, 有助于打破不同程序语言生态之间壁垒, 形成贯通式问题诊断技术。

- 生态视角下的利弊权衡: 在开源软件库生态中, 多样化的版本组合极易引发兼容性问题。考虑到软件库在开源软件库生态中的缺陷传播性, 借用 Watchman^[24]和 Hero^[187]等技术治理 Python 和 Go 语言生态依赖问题的观点, 应当根据知识库中软件生态的全局视角, 结合多维度的利弊权衡, 得到软件项目之间相互影响更小且更有益于软件生态的治理策略。

此外, 研究人员需要力求做到依赖缺陷问题的全面覆盖。例如, 近期涌现出文献 [349,352–355] 探讨由于依赖缺失而导致的软件不可重复构建问题; 面向特定领域的软件 (如人工智能领域) 探索具备领域特点的依赖缺陷问题^[356–360]等。研究人员仍需不断探索和挖掘开源软件库生态中的依赖缺陷问题, 不断推进开源软件库生态治理技术的发展。

(3) 开源软件库生态依赖缺陷问题治理技术的持续优化与提升

大多数的安全漏洞在被披露后短期内便伴随着修复补丁的发布, 但是由于开源软件库生态中软件库间版本更新的滞后性, 使得漏洞补丁无法快速传播, 阻塞在开源软件库生态的依赖路径中。因此, 研究人员应当攻克如下挑战, 以实现开源软件库生态中安全漏洞大规模清理。

- 监控软件库依赖链路的版本约束范围和更新状态 (补丁得以传播的前提)。
- 感知软件库的活跃程度 (软件库版本得以更新的前提)。
- 诊断软件库依赖链路中漏洞补丁未能得以传播的原因 (定位携带补丁的软件库版本被“阻塞”的症结)。

解决上述 3 个问题, 有助于推动开源生态的少数核心软件库以较低的修复代价, 最大程度地加速漏洞补丁沿着依赖链路广泛传播, 使得更多下游软件库自动化地引入安全漏洞补丁。

(4) 面向特定领域的开源软件库生态构建

现有的开源软件库生态治理技术可以改进并泛化到面向特定领域的软件库生态构建与维护中。例如, 面向泛在操作系统的开源软件库生态构建。在泛在计算的时代蓝海上, 泛在操作系统的新形态正在兴起。然而, 泛在操作系统面对的泛在资源呈现海量、异质、异构等特性, 资源管理的复杂度呈指数级增加。因此, 它需要软件库生态便于在泛在环境下按需加载和管理、支持各种程序语言的泛在应用, 并且能给保证其安全可靠运行。攻克开源软件库生态演化与维护和质量保证的技术, 有助于推动构建面向泛在操作系统的软件供应链, 围绕泛在计算模式打造自主可控的技术体系。

9 相关研究

开源软件库生态治理技术不断地被研究人员拓展、梳理与总结。过往的相关综述研究大多关注于开源软件生态系统整体架构的分析或单一生态的第三方软件库分析技术。本文是第 1 篇针对多种程序语言社区中开源软件库生态的建模方法和演化行为、依赖缺陷问题及其治理技术进行全面综述的文章, 系统地梳理了领域内近 20 多年的研究进展。

Mainikas 等人^[361,362]对软件生态系统的演化历程进行了研究综述, 先后梳理了发表于 2007–2012 年间的 90 篇

文献, 以及 2007–2014 年间的 231 篇文献, 关注论文发表数量和趋势、经验模型数量和生态系统类型等方面。他们认为特定领域软件生态系统的理论、方法和工具的研究是未来的发展方向, 并且给出趋势与展望的建议。Axelsson 等人^[363]针对软件生态系统的研究进展, 综述了 2011–2014 年间的 6 篇相关文献。关注于软件生命周期中各个阶段的质量保证问题。类似地, Franco-Bedoya 等人^[364]对开源软件生态系统的定义、建模和分析技术进行深入研究、分类和总结。他们收集了 2003–2015 年间的 82 篇相关论文, 发现研究者在软件生态系统的建模与分析、质量模型与标准定义等方面亟需更加深入的探索。然而, 上述工作并未探讨开源软件库生态的演化和依赖缺陷问题的治理技术。

Breivold 等人^[365]系统地回顾了开源软件演化研究领域的进展, 综述了 2007–2009 年间的 134 篇文献, 深入探讨了软件演化趋势、软件演化评估度量等方面的工作。他们建议研究人员统一术语解释; 更多关注开源软件的可变性、可扩展性等特性; 在软件架构层面理解开源软件的演化历程。文中仅关注了开源软件演化研究, 而未深入讨论不同软件生态的演化趋势, 更未涉及开源软件库生态的治理技术。

Zhan 等人^[28]对安卓第三方软件库的相关研究进行了文献综述。他们收集了 2012–2020 年间安卓第三方软件库的 74 篇相关研究论文, 总结了安卓生态中第三方软件库相关的缺陷与维护问题, 聚焦于安卓应用的第三方软件库检测技术、安全相关问题、安卓软件库的权限降级、安卓软件库维护等方面的工作。

10 结束语

历经 20 年的发展, 开源软件库生态治理技术得到了产业界和学术界的广泛关注与深入研究, 推动了软件产业的转型与升级。现有工作围绕着开源软件库生态的建模与分析、演化行为、依赖缺陷问题及其相应的治理技术开展研究, 形成了清晰脉络的理论与技术体系。然而, 依然缺少全局和共性的视角去审视软件演化行为的不可预见性、构建环境的不可控制性、上下游软件库依赖关系的复杂性以及程序语言的多样性, 以软件生态是视角持续地监测及治理依赖缺陷问题。

本文综述了领域相关研究进展, 并对未来发展方向进行展望。开源软件库的生态观不仅为软件科学带来新观点和新视角, 也可以帮助人们更好地理解和构建软件生态。面对新时代的新蓝海, 本文的愿景是能够推动领域发展、加快布局, 构建可信开源软件库生态。

References:

- [1] Mei H, Cao DG, Xie T. Ubiquitous operating system: Toward the blue ocean of human-cyber-physical ternary ubiquitous computing. *Bulletin of Chinese Academy of Sciences*, 2022, 37(1): 30–37 (in Chinese with English abstract). [doi: [10.16418/j.issn.1000-3045.20211117009](https://doi.org/10.16418/j.issn.1000-3045.20211117009)]
- [2] Liang GY, Wu YJ, Wu JZ, Zhao C. Open source software supply chain for reliability assurance of operating systems. *Ruan Jian Xue Bao/Journal of Software*, 2020, 31(10): 3056–3073 (in Chinese with English abstract). <http://www.jos.org.cn/1000-9825/6070.htm> [doi: [10.13328/j.cnki.jos.006070](https://doi.org/10.13328/j.cnki.jos.006070)]
- [3] Jin Z, Zhou MH, Zhang YX. Open source software and its eco-systems: Today and tomorrow. *Science & Technology Review*, 2016, 34(14): 42–48 (in Chinese with English abstract). [doi: [10.3981/j.issn.1000-7857.2016.14.005](https://doi.org/10.3981/j.issn.1000-7857.2016.14.005)]
- [4] Xu C, Qin Y, Yu P, Cao C, Lü J. Theories and techniques for growing software: Paradigm and beyond. *Scientia Sinica Informationis*, 2020, 50(11): 1595–1611 (in Chinese with English abstract). [doi: [10.1360/SSI-2020-0079](https://doi.org/10.1360/SSI-2020-0079)]
- [5] Abate P, Di Cosmo R, Gousios G, Zacchiroli S. Dependency solving is still hard, but we are getting better at it. In: *Proc. of the 27th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER)*. London: IEEE, 2020. 547–551. [doi: [10.1109/SANER48275.2020.9054837](https://doi.org/10.1109/SANER48275.2020.9054837)]
- [6] Maven. Welcome to Apache Maven. 2023. <https://maven.apache.org/what-is-maven.html>
- [7] Abate P, Di Cosmo R, Treinen R, Zacchiroli S. Dependency solving: A separate concern in component evolution management. *Journal of Systems and Software*, 2012, 85(10): 2228–2240. [doi: [10.1016/j.jss.2012.02.018](https://doi.org/10.1016/j.jss.2012.02.018)]
- [8] Abate P, DiCosmo R, Treinen R, Zacchiroli S. MPM: A modular package manager. In: *Proc. of the 14th Int'l ACM SIGSOFT Symp. on Component based Software Engineering*. Boulder: ACM, 2011. 179–188. [doi: [10.1145/2000229.2000255](https://doi.org/10.1145/2000229.2000255)]
- [9] Tucker C, Shuffelton D, Jhala R, Lerner S. OPIUM: Optimal package install/uninstall manager. In: *Proc. of the 29th Int'l Conf. on*

- Software Engineering (ICSE 2007). Minneapolis: IEEE, 2007. 178–188. [doi: [10.1109/ICSE.2007.59](https://doi.org/10.1109/ICSE.2007.59)]
- [10] Miranda A, Pimentel J. On the use of package managers by the C++ open-source community. In: Proc. of the 33rd Annual ACM Symp. on Applied Computing. Pau: ACM, 2018. 1483–1491. [doi: [10.1145/3167132.3167290](https://doi.org/10.1145/3167132.3167290)]
- [11] Mancinelli F, Boender J, Di Cosmo R, Vouillon J, Durak B, Leroy X, Treinen R. Managing the complexity of large free and open source package-based software distributions. In: Proc. of the 21st IEEE/ACM Int'l Conf. on Automated Software Engineering. Tokyo: IEEE, 2006. 199–208. [doi: [10.1109/ASE.2006.49](https://doi.org/10.1109/ASE.2006.49)]
- [12] Trezentos P, Lynce I, Oliveira AL. Apt-pbo: Solving the software dependency problem using pseudo-boolean optimization. In: Proc. of the 25th IEEE/ACM Int'l Conf. on Automated Software Engineering. Antwerp: ACM, 2010. 427–436. [doi: [10.1145/1858996.1859087](https://doi.org/10.1145/1858996.1859087)]
- [13] Internal Solver Competition. 2023. <https://www.mancoosi.org/misc-live/>
- [14] Argelich J, Le Berre D, Lynce I, Marques-Silva J, Rapicault P. Solving Linux upgradeability problems using boolean optimization. arXiv:1007.1021, 2010.
- [15] Di Cosmo R, Lhomme O, Michel C. Aligning component upgrades. arXiv:1109.0456, 2011.
- [16] Gebser M, Kaminski R, Schaub T. aspcud: A Linux package configuration tool based on answer set programming. arXiv:1109.0113, 2011.
- [17] Abate P, Di Cosmo R, Treinen R, Zacchiroli S. A modular package manager architecture. Information and Software Technology, 2013, 55(2): 459–474. [doi: [10.1016/j.infsof.2012.09.002](https://doi.org/10.1016/j.infsof.2012.09.002)]
- [18] Ossher J, Bajracharya S, Lopes C. Automated dependency resolution for open source software. In: Proc. of the 7th IEEE Working Conf. on Mining Software Repositories (MSR 2010). Cape Town: IEEE, 2010. 130–140. [doi: [10.1109/MSR.2010.5463346](https://doi.org/10.1109/MSR.2010.5463346)]
- [19] Claes M, Mens T, Di Cosmo R, Vouillon J. A historical analysis of Debian package incompatibilities. In: Proc. of the 12th IEEE/ACM Working Conf. on Mining Software Repositories. Florence: IEEE, 2015. 212–223. [doi: [10.1109/MSR.2015.27](https://doi.org/10.1109/MSR.2015.27)]
- [20] Di Cosmo R, Boender J. Using strong conflicts to detect quality issues in component-based complex systems. In: Proc. of the 3rd India Software Engineering Conf. Mysore: ACM, 2010. 163–172. [doi: [10.1145/1730874.1730905](https://doi.org/10.1145/1730874.1730905)]
- [21] Ossher J, Sajjani H, Lopes C. Astra: Bottom-up construction of structured artifact repositories. In: Proc. of the 19th Working Conf. on Reverse Engineering. Kingston: IEEE, 2012. 41–50. [doi: [10.1109/WCRE.2012.14](https://doi.org/10.1109/WCRE.2012.14)]
- [22] Decan A, Mens T, Claes M, Grosjean P. When GitHub meets CRAN: An analysis of inter-repository package dependency problems. In: Proc. of the 23rd IEEE Int'l Conf. on Software Analysis, Evolution, and Reengineering (SANER). Osaka: IEEE, 2016. 493–504. [doi: [10.1109/SANER.2016.12](https://doi.org/10.1109/SANER.2016.12)]
- [23] PyPI: The Python package index, PyPI. 2023. <https://pypi.org/>
- [24] Wang Y, Wen M, Liu YP, Wang YB, Li ZM, Wang C, Yu H, Cheung SC, Xu C, Zhu ZL. Watchman: Monitoring dependency conflicts for Python library ecosystem. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul: IEEE, 2020. 125–135. [doi: [10.1145/3377811.3380426](https://doi.org/10.1145/3377811.3380426)]
- [25] Preston-Werner T. Semantic Versioning 2.0.0. 2023. <https://semver.org/>
- [26] Wohlin C. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proc. of the 18th Int'l Conf. on Evaluation and Assessment in Software Engineering. London: ACM, 2014. 38. [doi: [10.1145/2601248.2601268](https://doi.org/10.1145/2601248.2601268)]
- [27] de Paulo Sobrinho EV, De Lucia A, de Almeida Maia M. A systematic literature review on bad smells-5 W's: Which, when, what, who, where. IEEE Trans. on Software Engineering, 2021, 47(1): 17–66. [doi: [10.1109/tse.2018.2880977](https://doi.org/10.1109/tse.2018.2880977)]
- [28] Zhan X, Liu TM, Fan LL, Li L, Chen S, Luo XP, Liu Y. Research on third-party libraries in Android APPs: A taxonomy and systematic literature review. IEEE Trans. on Software Engineering, 2022, 48(10): 4181–4213. [doi: [10.1109/TSE.2021.3114381](https://doi.org/10.1109/TSE.2021.3114381)]
- [29] Palepu VK, Xu GQ, Jones JA. Improving efficiency of dynamic analysis with dynamic dependence summaries. In: Proc. of the 28th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Silicon Valley: IEEE, 2013. 59–69. [doi: [10.1109/ASE.2013.6693066](https://doi.org/10.1109/ASE.2013.6693066)]
- [30] Utture A, Palsberg J. Fast and precise application code analysis using a partial library. In: Proc. of the 44th Int'l Conf. on Software Engineering (ICSE). Pittsburgh: IEEE, 2022. 934–945. [doi: [10.1145/3510003.3510046](https://doi.org/10.1145/3510003.3510046)]
- [31] Abate P, Di Cosmo R, Boender J, Zacchiroli S. Strong dependencies between software components. In: Proc. of the 3rd Int'l Symp. on Empirical Software Engineering and Measurement. Lake Buena Vista: IEEE, 2009. 89–99. [doi: [10.1109/ESEM.2009.5316017](https://doi.org/10.1109/ESEM.2009.5316017)]
- [32] Bavota G, Canfora G, Di Penta M, Oliveto R, Panichella S. The evolution of project inter-dependencies in a software ecosystem: The case of Apache. In: Proc. of the 2013 IEEE Int'l Conf. on Software Maintenance. Eindhoven: IEEE, 2013. 280–289. [doi: [10.1109/ICSM.2013.39](https://doi.org/10.1109/ICSM.2013.39)]
- [33] Decan A, Mens T, Claes M. An empirical comparison of dependency issues in OSS packaging ecosystems. In: Proc. of the 24th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Klagenfurt: IEEE, 2017. 2–12. [doi: [10.1109/SANER.2017.7884604](https://doi.org/10.1109/SANER.2017.7884604)]

- [34] Decan A, Mens T, Grosjean P. An empirical comparison of dependency network evolution in seven software packaging ecosystems. *Empirical Software Engineering*, 2019, 24(1): 381–416. [doi: [10.1007/s10664-017-9589-y](https://doi.org/10.1007/s10664-017-9589-y)]
- [35] German DM, Gonzalez-Barahona JM, Robles G. A model to understand the building and running inter-dependencies of software. In: *Proc. of the 14th Working Conf. on Reverse Engineering (WCRE 2007)*. Vancouver: IEEE, 2007. 140–149. [doi: [10.1109/WCRE.2007.5](https://doi.org/10.1109/WCRE.2007.5)]
- [36] Kikas R, Gousios G, Dumas M, Pfahl D. Structure and evolution of package dependency networks. In: *Proc. of the 14th IEEE/ACM Int'l Conf. on Mining Software Repositories (MSR)*. Buenos Aires: IEEE, 2017. 102–112. [doi: [10.1109/MSR.2017.55](https://doi.org/10.1109/MSR.2017.55)]
- [37] Kula RG, De Roover C, German DM, Ishio T, Inoue K. A generalized model for visualizing library popularity, adoption, and diffusion within a software ecosystem. In: *Proc. of the 25th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER)*. Campobasso: IEEE, 2018. 288–299. [doi: [10.1109/SANER.2018.8330217](https://doi.org/10.1109/SANER.2018.8330217)]
- [38] Lertwittayatrai N, Kula RG, Onoue S, Hata H, Rungsawang A, Leelaprute P, Matsumoto K. Extracting insights from the topology of the JavaScript package ecosystem. In: *Proc. of the 24th Asia-Pacific Software Engineering Conf. (APSEC)*. Nanjing: IEEE, 2017. 298–307. [doi: [10.1109/APSEC.2017.36](https://doi.org/10.1109/APSEC.2017.36)]
- [39] Mora-Cantalops M, Sánchez-Alonso S, García-Barriocanal E. A complex network analysis of the comprehensive R archive network (CRAN) package ecosystem. *Journal of Systems and Software*, 2020, 170: 110744. [doi: [10.1016/j.jss.2020.110744](https://doi.org/10.1016/j.jss.2020.110744)]
- [40] Tantisuwankul J, Nugroho YS, Kula RG, Hata H, Rungsawang A, Leelaprute P, Matsumoto K. A topological analysis of communication channels for knowledge sharing in contemporary GitHub projects. *Journal of Systems and Software*, 2019, 158: 110416. [doi: [10.1016/j.jss.2019.110416](https://doi.org/10.1016/j.jss.2019.110416)]
- [41] Wittern E, Suter P, Rajagopalan S. A look at the dynamics of the JavaScript package ecosystem. In: *Proc. of the 13th IEEE/ACM Working Conf. on Mining Software Repositories*. Austin: ACM, 2016. 351–361.
- [42] Zheng XL, Zeng D, Li HQ, Wang FY. Analyzing open-source software systems as complex networks. *Physica A: Statistical Mechanics and Its Applications*, 2008, 387(24): 6190–6200. [doi: [10.1016/j.physa.2008.06.050](https://doi.org/10.1016/j.physa.2008.06.050)]
- [43] Zimmermann M, Staicu CA, Tenny C, Pradel M. Small world with high risks: A study of security threats in the npm ecosystem. *arXiv:1902.09217*, 2019.
- [44] Nielsen BB, Torp MT, Møller A. Modular call graph construction for security scanning of Node.js applications. In: *Proc. of the 30th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis*. Virtual: ACM, 2021. 29–41. [doi: [10.1145/3460319.3464836](https://doi.org/10.1145/3460319.3464836)]
- [45] Bavota G, Canfora G, Di Penta M, Oliveto R, Panichella S. How the Apache community upgrades dependencies: An evolutionary study. *Empirical Software Engineering*, 2015, 20(5): 1275–1317. [doi: [10.1007/s10664-014-9325-9](https://doi.org/10.1007/s10664-014-9325-9)]
- [46] Mora Cantalops M, Sicilia MÁ, García-Barriocanal E, Sánchez-Alonso S. Evolution and prospects of the comprehensive R archive network (CRAN) package ecosystem. *Journal of Software: Evolution and Process*, 2020, 32(11): e2270. [doi: [10.1002/smr.2270](https://doi.org/10.1002/smr.2270)]
- [47] Hornik K. Are there too many R packages? *Austrian Journal of Statistics*, 2016, 41(1): 59. [doi: [10.17713/ajs.v41i1.188](https://doi.org/10.17713/ajs.v41i1.188)]
- [48] Caneill M, Germán DM, Zacchiroli S. The Debsources Dataset: Two decades of free and open source software. *Empirical Software Engineering*, 2017, 22(3): 1405–1437. [doi: [10.1007/s10664-016-9461-5](https://doi.org/10.1007/s10664-016-9461-5)]
- [49] Massacci F, Pashchenko I. Technical leverage in a software ecosystem: Development opportunities and security risks. In: *Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE)*. Madrid: IEEE, 2021. 1386–1397. [doi: [10.1109/ICSE43902.2021.00125](https://doi.org/10.1109/ICSE43902.2021.00125)]
- [50] Caneill M, Zacchiroli S. Debsources: Live and historical views on macro-level software evolution. In: *Proc. of the 8th ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement*. Torino: ACM Press, 2014. 28. [doi: [10.1145/2652524.2652528](https://doi.org/10.1145/2652524.2652528)]
- [51] Gonzalez-Barahona JM, Robles G, Michlmayr M, Amor JJ, German DM. Macro-level software evolution: A case study of a large software compilation. *Empirical Software Engineering*, 2009, 14(3): 262–285. [doi: [10.1007/s10664-008-9100-x](https://doi.org/10.1007/s10664-008-9100-x)]
- [52] Chen XW, Abdalkareem R, Mujahid S, Shihab E, Xia X. Helping or not helping? Why and how trivial packages impact the npm ecosystem. *Empirical Software Engineering*, 2021, 26(2): 27. [doi: [10.1007/s10664-020-09904-w](https://doi.org/10.1007/s10664-020-09904-w)]
- [53] Chowdhury MAR, Abdalkareem R, Shihab E, Adams B. On the untriviality of trivial packages: An empirical study of npm JavaScript packages. *IEEE Trans. on Software Engineering*, 2022, 48(8): 2695–2708. [doi: [10.1109/TSE.2021.3068901](https://doi.org/10.1109/TSE.2021.3068901)]
- [54] Abdalkareem R, Oda V, Mujahid S, Shihab E. On the impact of using trivial packages: An empirical case study on npm and PyPI. *Empirical Software Engineering*, 2020, 25(2): 1168–1204. [doi: [10.1007/s10664-019-09792-9](https://doi.org/10.1007/s10664-019-09792-9)]
- [55] Abdalkareem R, Nourry O, Wehaibi S, Mujahid S, Shihab E. Why do developers use trivial packages? An empirical case study on npm. In: *Proc. of the 11th Joint Meeting on Foundations of Software Engineering*. Paderborn: ACM, 2017. 385–395. [doi: [10.1145/3106237.3106267](https://doi.org/10.1145/3106237.3106267)]
- [56] Plakidas K, Schall D, Zdun U. Evolution of the R software ecosystem: Metrics, relationships, and their impact on qualities. *Journal of Systems and Software*, 2017, 132: 119–146. [doi: [10.1016/j.jss.2017.06.095](https://doi.org/10.1016/j.jss.2017.06.095)]
- [57] Bommarito E, Bommarito M. An empirical analysis of the Python package index (PyPI). *arXiv:1907.11073*, 2019.

- [58] Xie G W, Chen J B, Neamtiu I. Towards a better understanding of software evolution: An empirical study on open source software. In: Proc. of the 2009 IEEE Int'l Conf. on Software Maintenance. Edmonton: IEEE, 2009. 51–60. [doi: [10.1109/ICSM.2009.5306356](https://doi.org/10.1109/ICSM.2009.5306356)]
- [59] de la Mora FL, Nadi S. An empirical study of metric-based comparisons of software libraries. In: Proc. of the 14th Int'l Conf. on Predictive Models and Data Analytics in Software Engineering. Oulu: ACM, 2018. 22–31. [doi: [10.1145/3273934.3273937](https://doi.org/10.1145/3273934.3273937)]
- [60] Lima C, Hora A. What are the characteristics of popular APIs? A large-scale study on Java, Android, and 165 libraries. *Software Quality Journal*, 2020, 28(2): 425–458. [doi: [10.1007/s11219-019-09476-z](https://doi.org/10.1007/s11219-019-09476-z)]
- [61] Piccioni M, Furia CA, Meyer B. An empirical study of API usability. In: Proc. of the 2013 ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement. Baltimore: IEEE, 2013. 5–14. [doi: [10.1109/ESEM.2013.14](https://doi.org/10.1109/ESEM.2013.14)]
- [62] Raemaekers S, Nane GF, Van Deursen A, Visser J. Testing principles, current practices, and effects of change localization. In: Proc. of the 10th Working Conf. on Mining Software Repositories (MSR). San Francisco: IEEE, 2013. 257–266. [doi: [10.1109/MSR.2013.6624037](https://doi.org/10.1109/MSR.2013.6624037)]
- [63] Raemaekers S, van Deursen A, Visser J. Measuring software library stability through historical version analysis. In: Proc. of the 28th IEEE Int'l Conf. on Software Maintenance (ICSM). Trento: IEEE, 2012. 378–387. [doi: [10.1109/ICSM.2012.6405296](https://doi.org/10.1109/ICSM.2012.6405296)]
- [64] Sajjani H, Saini V, Ossher J, Lopes CV. Is popularity a measure of quality? An analysis of maven components. In: Proc. of the 2014 IEEE Int'l Conf. on Software Maintenance and Evolution. Victoria: IEEE, 2014. 231–240. [doi: [10.1109/ICSME.2014.45](https://doi.org/10.1109/ICSME.2014.45)]
- [65] Soto-Valero C, Benelallam A, Harrand N, Barais O, Baudry B. The emergence of software diversity in maven central. In: Proc. of the 16th IEEE/ACM Int'l Conf. on Mining Software Repositories (MSR). Montreal: IEEE, 2019. 333–343. [doi: [10.1109/MSR.2019.00059](https://doi.org/10.1109/MSR.2019.00059)]
- [66] Nguyen R, Holt R. Life and death of software packages: An evolutionary study of Debian. In: Proc. of the 2012 Conf. of the Center for Advanced Studies on Collaborative Research. Toronto: IBM Corporation, 2012. 192–204.
- [67] Mileva YM, Dallmeier V, Zeller A. Mining API popularity. In: Proc. of the 5th Int'l Academic and Industrial Conf. on Testing, Practice and Research Techniques. Windsor: Springer-Verlag, 2010. 173–180. [doi: [10.1007/978-3-642-15585-7_17](https://doi.org/10.1007/978-3-642-15585-7_17)]
- [68] Chinthanet B, Kula RG, McIntosh S, Ishio T, Ihara A, Matsumoto K. Lags in the release, adoption, and propagation of npm vulnerability fixes. *Empirical Software Engineering*, 2021, 26: 47. [doi: [10.1007/s10664-021-09951-x](https://doi.org/10.1007/s10664-021-09951-x)]
- [69] Zerouali A, Constantinou E, Mens T, Robles G, González-Barahona J. An empirical analysis of technical lag in npm package dependencies. In: Proc. of the 17th Int'l Conf. on New Opportunities for Software Reuse. Madrid: Springer, 2018. 95–110. [doi: [10.1007/978-3-319-90421-4_6](https://doi.org/10.1007/978-3-319-90421-4_6)]
- [70] Decan A, Mens T, Constantinou E. On the evolution of technical lag in the npm package dependency network. In: Proc. of the 2018 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Madrid: IEEE, 2018. 404–414. [doi: [10.1109/ICSME.2018.00050](https://doi.org/10.1109/ICSME.2018.00050)]
- [71] Alfadel M, Costa DE, Shihab E. Empirical analysis of security vulnerabilities in Python packages. In: Proc. of the 2021 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Honolulu: IEEE, 2021. 446–457. [doi: [10.1109/SANER50967.2021.00048](https://doi.org/10.1109/SANER50967.2021.00048)]
- [72] Decan A, Mens T, Constantinou E. On the impact of security vulnerabilities in the npm package dependency network. In: Proc. of the 15th IEEE/ACM Int'l Conf. on Mining Software Repositories. Gothenburg: ACM, 2018. 181–191. [doi: [10.1145/3196398.3196401](https://doi.org/10.1145/3196398.3196401)]
- [73] Zerouali A, Mens T, Decan A, de Roover C. On the impact of security vulnerabilities in the npm and RubyGems dependency networks. *Empirical Software Engineering*, 2022, 27(5): 107. [doi: [10.1007/s10664-022-10154-1](https://doi.org/10.1007/s10664-022-10154-1)]
- [74] Zerouali A, Mens T, Gonzalez-Barahona J, Decan A, Constantinou E, Robles G. A formal framework for measuring technical lag in component repositories and its application to npm. *Journal of Software: Evolution and Process*, 2019, 31(8): e2157. [doi: [10.1002/smr.2157](https://doi.org/10.1002/smr.2157)]
- [75] Møller A, Torp MT. Model-based testing of breaking changes in Node.js libraries. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Tallinn: ACM, 2019. 409–419. [doi: [10.1145/3338906.3338940](https://doi.org/10.1145/3338906.3338940)]
- [76] Cogo FR, Oliva GA, Hassan AE. An empirical study of dependency downgrades in the npm ecosystem. *IEEE Trans. on Software Engineering*, 2021, 47(11): 2457–2470. [doi: [10.1109/TSE.2019.2952130](https://doi.org/10.1109/TSE.2019.2952130)]
- [77] Rombaut B, Cogo FR, Adams B, Hassan AE. There's no such thing as a free lunch: Lessons learned from exploring the overhead introduced by the GreenKeeper dependency bot in npm. *ACM Trans. on Software Engineering and Methodology*, 2023, 32(1): 11. [doi: [10.1145/3522587](https://doi.org/10.1145/3522587)]
- [78] Wang Y, Chen BH, Huang KF, Shi BW, Xu CY, Peng X, Wu YJ, Liu Y. An empirical study of usages, updates and risks of third-party libraries in Java projects. In: Proc. of the 2020 IEEE Int'l Conf. on Software Maintenance and Evolution. Adelaide: IEEE, 2020. 35–45. [doi: [10.1109/ICSME46990.2020.00014](https://doi.org/10.1109/ICSME46990.2020.00014)]
- [79] Huang KF, Chen BH, Xu CY, Wang Y, Shi BW, Peng X, Wu YJ, Liu Y. Characterizing usages, updates and risks of third-party libraries

- in Java projects. *Empirical Software Engineering*, 2022, 27(4): 90. [doi: [10.1007/s10664-022-10131-8](https://doi.org/10.1007/s10664-022-10131-8)]
- [80] Kula RG, German DM, Ouni A, Ishio T, Inoue K. Do developers update their library dependencies? *Empirical Software Engineering*, 2018, 23(1): 384–417. [doi: [10.1007/s10664-017-9521-5](https://doi.org/10.1007/s10664-017-9521-5)]
- [81] Prana GAA, Sharma A, Shar LK, Foo D, Santosa AE, Sharma A, Lo D. Out of sight, out of mind? How vulnerable dependencies affect open-source projects. *Empirical Software Engineering*, 2021, 26(4): 59. [doi: [10.1007/s10664-021-09959-3](https://doi.org/10.1007/s10664-021-09959-3)]
- [82] Cox J, Bouwers E, van Eekelen M, Visser J. Measuring dependency freshness in software systems. In: *Proc. of the 37th IEEE/ACM Int'l Conf. on Software Engineering*. Florence: IEEE, 2015. 109–118. [doi: [10.1109/ICSE.2015.140](https://doi.org/10.1109/ICSE.2015.140)]
- [83] Hejderup J, Gousios G. Can we trust tests to automate dependency updates? A case study of Java projects. *Journal of Systems and Software*, 2022, 183: 111097. [doi: [10.1016/j.jss.2021.111097](https://doi.org/10.1016/j.jss.2021.111097)]
- [84] Zaimi A, Ampatzoglou A, Triantafyllidou N, Chatzigeorgiou A, Mavridis A, Chaikalis T, Deligiannis I, Sfetsos P, Stamelos I. An empirical study on the reuse of third-party libraries in open-source software development. In: *Proc. of the 7th Balkan Conf. on Informatics Conf*. Craiova: ACM, 2015. 4. [doi: [10.1145/2801081.2801087](https://doi.org/10.1145/2801081.2801087)]
- [85] Dig D, Negara S, Johnson R, Johnson R. *ReBA*: Refactoring-aware binary adaptation of evolving libraries. In: *Proc. of the 30th Int'l Conf. on Software Engineering*. Leipzig: ACM, 2008. 441–450. [doi: [10.1145/1368088.1368148](https://doi.org/10.1145/1368088.1368148)]
- [86] Wu W, Guéhéneuc YG, Antoniol G, Kim M. AURA: A hybrid approach to identify framework evolution. In: *Proc. of the 32nd ACM/IEEE Int'l Conf. on Software Engineering*. Cape Town: IEEE, 2010. 325–334. [doi: [10.1145/1806799.1806848](https://doi.org/10.1145/1806799.1806848)]
- [87] Wu W, Serveaux A, Guéhéneuc YG, Antoniol G. The impact of imperfect change rules on framework API evolution identification: An empirical study. *Empirical Software Engineering*, 2015, 20(4): 1126–1158. [doi: [10.1007/s10664-014-9317-9](https://doi.org/10.1007/s10664-014-9317-9)]
- [88] Dagenais B, Robillard MP. Recommending adaptive changes for framework evolution. *ACM Trans. on Software Engineering and Methodology*, 2011, 20(4): 19. [doi: [10.1145/2000799.2000805](https://doi.org/10.1145/2000799.2000805)]
- [89] Kalra S, Goel A, Khanna D, Dhawan M, Sharma S, Purandare R. POLLUX: Safely upgrading dependent application libraries. In: *Proc. of the 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering*. Seattle: ACM, 2016. 290–300. [doi: [10.1145/2950290.2950345](https://doi.org/10.1145/2950290.2950345)]
- [90] Flores A, Usaola MP. Testing-based component assessment for substitutability. In: *Proc. of the 10th Int'l Conf. on Enterprise Information Systems*. Barcelona, 2008. 386–393.
- [91] Gyori A, Legunsen O, Hariri F, Marinov D. Evaluating regression test selection opportunities in a very large open-source ecosystem. In: *Proc. of the 29th IEEE Int'l Symp. on Software Reliability Engineering (ISSRE)*. Memphis: IEEE, 2018. 112–122. [doi: [10.1109/ISSRE.2018.00022](https://doi.org/10.1109/ISSRE.2018.00022)]
- [92] Hou DQ, Yao XJ. Exploring the intent behind API evolution: A case study. In: *Proc. of the 18th Working Conf. on Reverse Engineering*. Limerick: IEEE, 2011. 131–140. [doi: [10.1109/WCRE.2011.24](https://doi.org/10.1109/WCRE.2011.24)]
- [93] Orsila H, Geldenhuys J, Ruokonen A, Hammouda I. Update propagation practices in highly reusable open source components. In: *Proc. of the 20th IFIP World Computer Congress*. Milano: Springer, 2008. 159–170. [doi: [10.1007/978-0-387-09684-1_13](https://doi.org/10.1007/978-0-387-09684-1_13)]
- [94] Dilhara M, Ketkar A, Dig D. Understanding software-2.0: A study of machine learning library usage and evolution. *ACM Trans. on Software Engineering and Methodology*, 2021, 30(4): 55. [doi: [10.1145/3453478](https://doi.org/10.1145/3453478)]
- [95] Padioleau Y, Lawall J, Hansen RR, Muller G. Documenting and automating collateral evolutions in Linux device drivers. *ACM SIGOPS Operating Systems Review*, 2008, 42(4): 247–260. [doi: [10.1145/1357010.1352618](https://doi.org/10.1145/1357010.1352618)]
- [96] Padioleau Y, Lawall JL, Muller G. SmPL: A domain-specific language for specifying collateral evolutions in Linux device drivers. *Electronic Notes in Theoretical Computer Science*, 2007, 166: 47–62. [doi: [10.1016/j.entcs.2006.07.022](https://doi.org/10.1016/j.entcs.2006.07.022)]
- [97] Almanee S, Ünal A, Payer M, Garcia J. Too quiet in the library: An empirical study of security updates in Android APPs' native code. In: *Proc. of the 43rd Int'l Conf. on Software Engineering*. Madrid: IEEE, 2021. 1347–1359. [doi: [10.1109/ICSE43902.2021.00122](https://doi.org/10.1109/ICSE43902.2021.00122)]
- [98] Yasumatsu T, Watanabe T, Kanei F, Shioji E, Akiyama M, Mori T. Understanding the responsiveness of mobile APP developers to software library updates. In: *Proc. of the 9th ACM Conf. on Data and Application Security and Privacy*. Richardson: ACM, 2019. 13–24. [doi: [10.1145/3292006.3300020](https://doi.org/10.1145/3292006.3300020)]
- [99] Derr E, Bugiel S, Fahl S, Acar Y, Backes M. Keep me updated: An empirical study of third-party library updatability on Android. In: *Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security*. Dallas: Association for Computing Machinery, 2017. 2187–2200. [doi: [10.1145/3133956.3134059](https://doi.org/10.1145/3133956.3134059)]
- [100] McDonnell T, Ray B, Kim M. An empirical study of API stability and adoption in the Android ecosystem. In: *Proc. of the 2013 IEEE Int'l Conf. on Software Maintenance*. Eindhoven: IEEE, 2013. 70–79. [doi: [10.1109/ICSM.2013.18](https://doi.org/10.1109/ICSM.2013.18)]
- [101] Salza P, Palomba F, Di Nucci D, D'Uva C, De Lucia A, Ferrucci F. Do developers update third-party libraries in mobile APPs? In: *Proc. of the 26th IEEE/ACM Int'l Conf. on Program Comprehension*. Gothenburg: IEEE, 2018. 255–265. [doi: [10.1145/3196321.3196341](https://doi.org/10.1145/3196321.3196341)]

- [102] Ahasanuzzaman M, Hassan S, Bezemer CP, Hassan AE. A longitudinal study of popular ad libraries in the Google Play Store. *Empirical Software Engineering*, 2020, 25(1): 824–858. [doi: [10.1007/s10664-019-09766-x](https://doi.org/10.1007/s10664-019-09766-x)]
- [103] Mahmud T, Che MR, Yang GW. Android compatibility issue detection using API differences. In: *Proc. of the 2021 IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER)*. Honolulu: IEEE, 2021. 480–490. [doi: [10.1109/SANER50967.2021.00051](https://doi.org/10.1109/SANER50967.2021.00051)]
- [104] Xia H, Zhang Y, Zhou YT, Chen XT, Wang Y, Zhang XY, Cui SS, Hong G, Zhang XH, Yang M, Yang ZM. How Android developers handle evolution-induced API compatibility issues: A large-scale study. In: *Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering (ICSE)*. Seoul: IEEE, 2020. 886–898. [doi: [10.1145/3377811.3380357](https://doi.org/10.1145/3377811.3380357)]
- [105] Cai HP, Zhang ZY, Li L, Fu XQ. A large-scale study of application incompatibilities in Android. In: *Proc. of the 28th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis*. Beijing: ACM, 2019. 216–227. [doi: [10.1145/3293882.3330564](https://doi.org/10.1145/3293882.3330564)]
- [106] Salza P, Palomba F, Di Nucci D, De Lucia A, Ferrucci F. Third-party libraries in mobile APPs: When, how, and why developers update them. *Empirical Software Engineering*, 2020, 25(3): 2341–2377. [doi: [10.1007/s10664-019-09754-1](https://doi.org/10.1007/s10664-019-09754-1)]
- [107] Stringer J, Tahir A, Blincoe K, Dietrich J. Technical lag of dependencies in major package managers. In: *Proc. of the 27th Asia-Pacific Software Engineering Conf. (APSEC)*. Singapore: IEEE, 2020. 228–237. [doi: [10.1109/APSEC51365.2020.00031](https://doi.org/10.1109/APSEC51365.2020.00031)]
- [108] Cogo FR, Oliva GA, Hassan AE. Deprecation of packages and releases in software ecosystems: A case study on NPM. *IEEE Trans. on Software Engineering*, 2022, 48(7): 2208–2223. [doi: [10.1109/TSE.2021.3055123](https://doi.org/10.1109/TSE.2021.3055123)]
- [109] Xing ZC, Stroulia E. API-evolution support with Diff-CatchUp. *IEEE Trans. on Software Engineering*, 2007, 33(12): 818–836. [doi: [10.1109/TSE.2007.70747](https://doi.org/10.1109/TSE.2007.70747)]
- [110] Teyton C, Falleri JR, Palyart M, Blanc X. A study of library migrations in Java. *Journal of Software: Evolution and Process*, 2014, 26(11): 1030–1052. [doi: [10.1002/smr.1660](https://doi.org/10.1002/smr.1660)]
- [111] Lamothe M, Shang WY, Chen THP. A3: Assisting Android API migrations using code examples. *IEEE Trans. on Software Engineering*, 2022, 48(2): 417–431. [doi: [10.1109/TSE.2020.2988396](https://doi.org/10.1109/TSE.2020.2988396)]
- [112] Meng SC, Wang XY, Zhang L, Mei H. A history-based matching approach to identification of framework evolution. In: *Proc. of the 34th Int'l Conf. on Software Engineering (ICSE)*. Zurich: IEEE, 2012. 353–363. [doi: [10.1109/ICSE.2012.6227179](https://doi.org/10.1109/ICSE.2012.6227179)]
- [113] Wang CL, Jiang JJ, Li J, Xiong YF, Luo XY, Zhang L, Hu ZJ. Transforming programs between APIs with many-to-many mappings. In: *Proc. of the 30th European Conf. on Object-Oriented Programming (ECOOP)*. Dagstuhl, 2016. 25: 1–25. 26. [doi: [10.4230/LIPIcs.ECOOP.2016.25](https://doi.org/10.4230/LIPIcs.ECOOP.2016.25)]
- [114] Henkel J, Diwan A. CatchUp!: Capturing and replaying refactorings to support API evolution. In: *Proc. of the 27th Int'l Conf. on Software Engineering*. St. Louis: ACM, 2005. 274–283. [doi: [10.1145/1062455.1062512](https://doi.org/10.1145/1062455.1062512)]
- [115] Dig D, Johnson R. The role of refactorings in API evolution. In: *Proc. of the 21st IEEE Int'l Conf. on Software Maintenance (ICSM)*. Budapest: IEEE, 2005. 389–398. [doi: [10.1109/ICSM.2005.90](https://doi.org/10.1109/ICSM.2005.90)]
- [116] Lämmel R, Pek E, Starek J. Large-scale, AST-based API-usage analysis of open-source Java projects. In: *Proc. of the 2011 ACM Symp. on Applied Computing*. Taichung: ACM, 2011. 1317–1324. [doi: [10.1145/1982185.1982471](https://doi.org/10.1145/1982185.1982471)]
- [117] Alrubaye H, Alshoabi D, Alomar E, Mkaouer MW, Ouni A. How does library migration impact software quality and comprehension? An empirical study. In: *Proc. of the 19th Int'l Conf. on Software and Software Reuse*. Hammamet: Springer-Verlag, 2020. 245–260. [doi: [10.1007/978-3-030-64694-3_15](https://doi.org/10.1007/978-3-030-64694-3_15)]
- [118] Teyton C, Falleri JR, Blanc X. Mining library migration graphs. In: *Proc. of the 19th Working Conf. on Reverse Engineering*. Kingston: IEEE, 2012. 289–298. [doi: [10.1109/WCRE.2012.38](https://doi.org/10.1109/WCRE.2012.38)]
- [119] Teyton C, Falleri JR, Blanc X. Automatic discovery of function mappings between similar libraries. In: *Proc. of the 20th Working Conf. on Reverse Engineering*. Koblenz: IEEE, 2013. 192–201. [doi: [10.1109/WCRE.2013.6671294](https://doi.org/10.1109/WCRE.2013.6671294)]
- [120] Alrubaye H, Mkaouer MW. Automating the detection of third-party Java library migration at the function level. In: *Proc. of the 28th Annual Int'l Conf. on Computer Science and Software Engineering*. Markham: IBM Corp., 2018. 60–71. [doi: [10.1109/ICPC.2019.00053](https://doi.org/10.1109/ICPC.2019.00053)]
- [121] Alrubaye H, Mkaouer MW, Ouni A. On the use of information retrieval to automate the detection of third-party Java library migration at the method level. In: *Proc. of the 27th IEEE/ACM Int'l Conf. on Program Comprehension (ICPC)*. Montreal: IEEE, 2019. 347–357.
- [122] Balaban I, Tip F, Fuhrer R. Refactoring support for class library migration. In: *Proc. of the 20th Annual ACM SIGPLAN Conf. on Object-oriented Programming Systems, Languages, and Applications*. San Diego: ACM, 2005. 265–279. [doi: [10.1145/1094811.1094832](https://doi.org/10.1145/1094811.1094832)]
- [123] Chen CY, Xing ZC, Liu Y, Xiong KOL. Mining likely analogical APIs across third-party libraries via large-scale unsupervised API semantics embedding. *IEEE Trans. on Software Engineering*, 2021, 47(3): 432–447. [doi: [10.1109/TSE.2019.2896123](https://doi.org/10.1109/TSE.2019.2896123)]
- [124] Winter VL, Mamejtjanov A. Generative programming techniques for Java library migration. In: *Proc. of the 6th Int'l Conf. on Generative*

- Programming and Component Engineering. Salzburg: ACM, 2007. 185–196. [doi: [10.1145/1289971.1290001](https://doi.org/10.1145/1289971.1290001)]
- [125] Cossette BE, Walker RJ. Seeking the ground truth: A retroactive study on the evolution and migration of software libraries. In: Proc. of the 20th ACM SIGSOFT Int'l Symp. on the Foundations of Software Engineering. Cary: ACM, 2012. 55. [doi: [10.1145/2393596.2393661](https://doi.org/10.1145/2393596.2393661)]
- [126] Robillard MP, Kutschera K. Lessons learned while migrating from swing to JavaFX. IEEE Software, 2020, 37(3): 78–85. [doi: [10.1109/MS.2019.2919840](https://doi.org/10.1109/MS.2019.2919840)]
- [127] Suwa H, Ihara A, Kula RG, Fujibayashi D, Matsumoto K. An analysis of library rollbacks: A case study of Java libraries. In: Proc. of the 24th Asia-Pacific Software Engineering Conf. Workshops (APSEC). Nanjing: IEEE, 2018. 63–70. [doi: [10.1109/APSECW.2017.25](https://doi.org/10.1109/APSECW.2017.25)]
- [128] Brito A, Xavier L, Hora A, Valente MT. Why and how Java developers break APIs. In: Proc. of the 25th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Campobasso: IEEE, 2018. 255–265. [doi: [10.1109/SANER.2018.8330214](https://doi.org/10.1109/SANER.2018.8330214)]
- [129] Bartolomei TT, Czarniecki K, Lämmel R, van der Storm T. Study of an API migration for two XML APIs. In: Proc. of the 2nd Int'l Conf. on Software Language Engineering. Denver: Springer, 2009. 42–61. [doi: [10.1007/978-3-642-12107-4_5](https://doi.org/10.1007/978-3-642-12107-4_5)]
- [130] Ni A, Ramos D, Yang AZH, Lynce I, Manquinho V, Martins R, Le Goues C. SOAR: A synthesis approach for data science API refactoring. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Madrid: IEEE, 2021. 112–124. [doi: [10.1109/ICSE43902.2021.00023](https://doi.org/10.1109/ICSE43902.2021.00023)]
- [131] Tang W, Xu ZZ, Liu CW, Wu JH, Yang SG, Li Y, Luo P, Liu Y. Towards understanding third-party library dependency in C/C++ ecosystem. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2022. 106. [doi: [10.1145/3551349.3560432](https://doi.org/10.1145/3551349.3560432)]
- [132] He H, He RZ, Gu HQ, Zhou MH. A large-scale empirical study on Java library migrations: Prevalence, trends, and rationales. In: Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Athens: ACM, 2021. 478–490. [doi: [10.1145/3468264.3468571](https://doi.org/10.1145/3468264.3468571)]
- [133] Robillard MP, Bodden E, Kawrykow D, Mezini M, Ratchford T. Automated API property inference techniques. IEEE Trans. on Software Engineering, 2013, 39(5): 613–637. [doi: [10.1109/TSE.2012.63](https://doi.org/10.1109/TSE.2012.63)]
- [134] Acharya M, Xie T, Pei J, Xu J. Mining API patterns as partial orders from source code: From usage scenarios to specifications. In: Proc. of the 6th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering. Dubrovnik: ACM, 2007. 25–34. [doi: [10.1145/1287624.1287630](https://doi.org/10.1145/1287624.1287630)]
- [135] Borges HS, Valente MT. Mining usage patterns for the Android API. PeerJ Computer Science, 2015, 1: e12. [doi: [10.7717/peerj-cs.12](https://doi.org/10.7717/peerj-cs.12)]
- [136] Holmes R, Walker RJ, Murphy GC. Approximate structural context matching: An approach to recommend relevant examples. IEEE Trans. on Software Engineering, 2006, 32(12): 952–970. [doi: [10.1109/TSE.2006.117](https://doi.org/10.1109/TSE.2006.117)]
- [137] Zhong H, Xie T, Zhang L, Pei J, Mei H. MAPO: Mining and recommending API usage patterns. In: Proc. of the 23rd European Conf. on Object-oriented Programming. Genoa: Springer, 2009. 318–343. [doi: [10.1007/978-3-642-03013-0_15](https://doi.org/10.1007/978-3-642-03013-0_15)]
- [138] Saied MA, Sahraoui H, Dufour B. An observational study on API usage constraints and their documentation. In: Proc. of the 22nd IEEE Int'l Conf. on Software Analysis, Evolution, and Reengineering (SANER). Montreal: IEEE, 2015. 33–42. [doi: [10.1109/SANER.2015.7081813](https://doi.org/10.1109/SANER.2015.7081813)]
- [139] Saied MA, Abdeen H, Benomar O, Sahraoui H. Could we infer unordered API usage patterns only using the library source code? In: Proc. of the 23rd IEEE Int'l Conf. on Program Comprehension. Florence: IEEE, 2015. 71–81. [doi: [10.1109/ICPC.2015.16](https://doi.org/10.1109/ICPC.2015.16)]
- [140] Saied MA, Sahraoui H. A cooperative approach for combining client-based and library-based API usage pattern mining. In: Proc. of the 24th IEEE Int'l Conf. on Program Comprehension (ICPC). Austin: IEEE, 2016. 1–10. [doi: [10.1109/ICPC.2016.7503717](https://doi.org/10.1109/ICPC.2016.7503717)]
- [141] Zhu ZX, Zou YZ, Xie B, Jin Y, Lin ZQ, Zhang L. Mining API usage examples from test code. In: Proc. of the 2014 IEEE Int'l Conf. on Software Maintenance and Evolution. Victoria: IEEE, 2014. 301–310. [doi: [10.1109/ICSME.2014.52](https://doi.org/10.1109/ICSME.2014.52)]
- [142] Niu HR, Keivanloo I, Zou Y. API usage pattern recommendation for software development. Journal of Systems and Software, 2017, 129: 127–139. [doi: [10.1016/j.jss.2016.07.026](https://doi.org/10.1016/j.jss.2016.07.026)]
- [143] Saied MA, Ouni A, Sahraoui H, Kula RG, Inoue K, Lo D. Improving reusability of software libraries through usage pattern mining. Journal of Systems and Software, 2018, 145: 164–179. [doi: [10.1016/j.jss.2018.08.032](https://doi.org/10.1016/j.jss.2018.08.032)]
- [144] Nguyen HA, Nguyen TT, Wilson G, Nguyen AT, Kim M, Nguyen TN. A graph-based approach to API usage adaptation. In: Proc. of the 2010 ACM Int'l Conf. on Object Oriented Programming Systems Languages and Applications. Reno: ACM, 2010. 302–321. [doi: [10.1145/1869459.1869486](https://doi.org/10.1145/1869459.1869486)]
- [145] Nielebock S, Heumüller R, Schott KM, Ortmeier F. Guided pattern mining for API misuse detection by change-based code analysis. Automated Software Engineering, 2021, 28(2): 15. [doi: [10.1007/s10515-021-00294-x](https://doi.org/10.1007/s10515-021-00294-x)]
- [146] Bauer V, Heinemann L, Deissenboeck F. A structured approach to assess third-party library usage. In: Proc. of the 28th IEEE Int'l Conf. on Software Maintenance (ICSM). Trento: IEEE, 2012. 483–492. [doi: [10.1109/ICSM.2012.6405311](https://doi.org/10.1109/ICSM.2012.6405311)]

- [147] Yoon IC, Sussman A, Memon A, Porter A. Prioritizing component compatibility tests via user preferences. In: Proc. of the 2009 IEEE Int'l Conf. on Software Maintenance. Edmonton: IEEE, 2009. 29–38. [doi: [10.1109/ICSM.2009.5306357](https://doi.org/10.1109/ICSM.2009.5306357)]
- [148] Yoon I, Sussman A, Memon A, Porter A. Testing component compatibility in evolving configurations. *Information and Software Technology*, 2013, 55(2): 445–458. [doi: [10.1016/j.infsof.2012.09.010](https://doi.org/10.1016/j.infsof.2012.09.010)]
- [149] Dogguy M, Glondu S, Le Gall S, Zacchiroli S. Enforcing type-safe linking using inter-package relationships. *Studia Informatica Universalis*, 2011, 9(1): 129–157.
- [150] Mitchell N, Schonberg E, Sevitsky G. Four trends leading to Java runtime bloat. *IEEE Software*, 2010, 27(1): 56–63. [doi: [10.1109/MS.2010.7](https://doi.org/10.1109/MS.2010.7)]
- [151] Tan JL, Chen Y, Liu ZM, Ren B, Song SL, Shen XP, Liu X. Toward efficient interactions between Python and native libraries. In: Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Athens: Association for Computing Machinery, 2021. 1117–1128. [doi: [10.1145/3468264.3468541](https://doi.org/10.1145/3468264.3468541)]
- [152] Macho C, McIntosh S, Pinzger M. Automatically repairing dependency-related build breakage. In: Proc. of the 25th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). Campobasso: IEEE, 2018. 106–117. [doi: [10.1109/SANER.2018.8330201](https://doi.org/10.1109/SANER.2018.8330201)]
- [153] Jezek K, Dietrich J, Brada P. How Java APIs break—An empirical study. *Information and Software Technology*, 2015, 65: 129–146. [doi: [10.1016/j.infsof.2015.02.014](https://doi.org/10.1016/j.infsof.2015.02.014)]
- [154] Hora A, Robbes R, Anquetil N, Etien A, Ducasse S, Valente MT. How do developers react to API evolution? The Pharo ecosystem case. In: Proc. of the 2015 Int'l Conf. on Software Maintenance and Evolution (ICSME). Bremen: IEEE, 2015. 251–260. [doi: [10.1109/ICSM.2015.7332471](https://doi.org/10.1109/ICSM.2015.7332471)]
- [155] Brito A, Valente MT, Xavier L, Hora A. You broke my code: Understanding the motivations for breaking changes in APIs. *Empirical Software Engineering*, 2020, 25: 1458–1492. [doi: [10.1007/s10664-019-09756-z](https://doi.org/10.1007/s10664-019-09756-z)]
- [156] Zhang ZX, Zhu HC, Wen M, Tao YD, Liu YP, Xiong YF. How do Python framework APIs evolve? An exploratory study. In: Proc. of the 27th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER). London: IEEE, 2020. 81–92. [doi: [10.1109/SANER48275.2020.9054800](https://doi.org/10.1109/SANER48275.2020.9054800)]
- [157] Bogart C, Kästner C, Herbsleb J, Thung F. When and how to make breaking changes: Policies and practices in 18 open source software ecosystems. *ACM Trans. on Software Engineering and Methodology*, 2021, 30(4): 42. [doi: [10.1145/3447245](https://doi.org/10.1145/3447245)]
- [158] Dietrich J, Jezek K, Brada P. Broken promises: An empirical study into evolution problems in Java programs caused by library upgrades. In: Proc. of the 2014 IEEE Conf. on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE). Antwerp: IEEE, 2014. 64–73. [doi: [10.1109/CSMR-WCRE.2014.6747226](https://doi.org/10.1109/CSMR-WCRE.2014.6747226)]
- [159] Dietrich J, Jezek K, Brada P. What Java developers know about compatibility, and why this matters. *Empirical Software Engineering*, 2016, 21(3): 1371–1396. [doi: [10.1007/s10664-015-9389-1](https://doi.org/10.1007/s10664-015-9389-1)]
- [160] Jia ZY, Li SS, Yu TT, Zeng C, Xu EC, Liu XD, Wang J, Liao XK. DepOwl: Detecting dependency bugs to prevent compatibility failures. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Madrid: IEEE, 2021. 86–98. [doi: [10.1109/ICSE43902.2021.00021](https://doi.org/10.1109/ICSE43902.2021.00021)]
- [161] Savga I, Rudolf M. Refactoring-based support for binary compatibility in evolving frameworks. In: Proc. of the 6th Int'l Conf. on Generative Programming and Component Engineering. Salzburg: ACM, 2007. 175–184. [doi: [10.1145/1289971.1290000](https://doi.org/10.1145/1289971.1290000)]
- [162] Mostafa S, Rodriguez R, Wang XY. Experience paper: A study on behavioral backward incompatibilities of Java software libraries. In: Proc. of the 26th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. Santa Barbara: ACM, 2017. 215–225. [doi: [10.1145/3092703.3092721](https://doi.org/10.1145/3092703.3092721)]
- [163] Dig D, Johnson R. How do APIs evolve? A story of refactoring. *Journal of Software Maintenance and Evolution: Research and Practice*, 2006, 18(2): 83–107. [doi: [10.1002/smr.328](https://doi.org/10.1002/smr.328)]
- [164] Sawant AA, Huang GZ, Vilen G, Stojkovski S, Bacchelli A. Why are features deprecated? An investigation into the motivation behind deprecation. In: Proc. of the 2018 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Madrid: IEEE, 2018. 13–24. [doi: [10.1109/ICSME.2018.00011](https://doi.org/10.1109/ICSME.2018.00011)]
- [165] Robbes R, Lungu M, Röthlisberger D. How do developers react to API deprecation? The case of a smalltalk ecosystem. In: Proc. of the 20th ACM SIGSOFT Int'l Symp. on the Foundations of Software Engineering. Cary: ACM, 2012. 56. [doi: [10.1145/2393596.2393662](https://doi.org/10.1145/2393596.2393662)]
- [166] Qiu D, Li BX, Leung H. Understanding the API usage in Java. *Information and Software Technology*, 2016, 73: 81–100. [doi: [10.1016/j.infsof.2016.01.011](https://doi.org/10.1016/j.infsof.2016.01.011)]
- [167] Sawant AA, Robbes R, Bacchelli A. On the reaction to deprecation of 25357 clients of 4+1 popular Java APIs. In: Proc. of the 2016 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Raleigh: IEEE, 2017. 400–410. [doi: [10.1109/ICSME.2016.64](https://doi.org/10.1109/ICSME.2016.64)]
- [168] Sawant AA, Robbes R, Bacchelli A. To react, or not to react: Patterns of reaction to API deprecation. *Empirical Software Engineering*,

- 2019, 24(6): 3824–3870. [doi: [10.1007/s10664-019-09713-w](https://doi.org/10.1007/s10664-019-09713-w)]
- [169] Mezzetti G, Møller A, Torp MT. Type regression testing to detect breaking changes in Node.js libraries. In: Proc. of the 32nd European Conf. on Object-oriented Programming (ECOOP). Dagstuhl, 2018. 7: 1–7: 24. [doi: [10.4230/LIPIcs.ECOOP.2018.7](https://doi.org/10.4230/LIPIcs.ECOOP.2018.7)]
- [170] Raemaekers S, van Deursen A, Visser J. Semantic versioning versus breaking changes: A study of the maven repository. In: Proc. of the 14th IEEE Int'l Working Conf. on Source Code Analysis and Manipulation. Victoria: IEEE, 2014. 215–224. [doi: [10.1109/SCAM.2014.30](https://doi.org/10.1109/SCAM.2014.30)]
- [171] Raemaekers S, van Deursen A, Visser J. Semantic versioning and impact of breaking changes in the Maven repository. *Journal of Systems and Software*, 2017, 129: 140–158. [doi: [10.1016/j.jss.2016.04.008](https://doi.org/10.1016/j.jss.2016.04.008)]
- [172] Zhang L, Liu CW, Xu ZZ, Chen S, Fan LL, Chen BH, Liu Y. Has my release disobeyed semantic versioning? Static detection based on semantic differencing. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2022. 51. [doi: [10.1145/3551349.3556956](https://doi.org/10.1145/3551349.3556956)]
- [173] Di Penta M, German DM, Guéhéneuc YG, Antoniol G. An exploratory study of the evolution of software licensing. In: Proc. of the 32nd ACM/IEEE Int'l Conf. on Software Engineering. Cape Town: IEEE, 2010. 145–154. [doi: [10.1145/1806799.1806824](https://doi.org/10.1145/1806799.1806824)]
- [174] Kapitsaki GM, Kramer F. Open source license violation check for SPDX files. In: Proc. of the 14th Int'l Conf. on Software Reuse. Miami: Springer, 2014. 90–105. [doi: [10.1007/978-3-319-14130-5_7](https://doi.org/10.1007/978-3-319-14130-5_7)]
- [175] Mlouki O, Khomh F, Antoniol G. On the detection of licenses violations in the Android ecosystem. In: Proc. of the 23rd IEEE Int'l Conf. on Software Analysis, Evolution, and Reengineering (SANER). Osaka: IEEE, 2016. 382–392. [doi: [10.1109/SANER.2016.73](https://doi.org/10.1109/SANER.2016.73)]
- [176] Meloca R, Pinto G, Baiser L, Mattos M, Polato I, Wiese I, German DM. Understanding the usage, impact, and adoption of non-OSI approved licenses. In: Proc. of the 15th Int'l Conf. on Mining Software Repositories. Gothenburg: ACM, 2018. 270–280. [doi: [10.1145/3196398.3196427](https://doi.org/10.1145/3196398.3196427)]
- [177] CoKinetic Systems Pursues \$100 Million GPL License Violation Case Against Panasonic Avionics. WP Tavern. 2023. <https://wptavern.com/cokinetic-systems-pursues-100-million-gpl-license-violation-case-against-panasonic-avionics>
- [178] Vendome C, Linares-Vásquez M, Bavota G, Di Penta M, German DM, Poshyvanik D. When and why developers adopt and change software licenses. In: Proc. of the 2015 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Bremen: IEEE, 2015. 31–40. [doi: [10.1109/ICSM.2015.7332449](https://doi.org/10.1109/ICSM.2015.7332449)]
- [179] Sen R, Subramaniam C, Nelson ML. Open source software licenses: Strong-copyleft, non-copyleft, or somewhere in between? *Decision Support Systems*, 2011, 52(1): 199–206. [doi: [10.1016/j.dss.2011.07.004](https://doi.org/10.1016/j.dss.2011.07.004)]
- [180] Almeida DA, Murphy GC, Wilson G, Hoye M. Do software developers understand open source licenses? In: Proc. of the 25th IEEE/ACM Int'l Conf. on Program Comprehension (ICPC). Buenos Aires: IEEE, 2017. 1–11. [doi: [10.1109/ICPC.2017.7](https://doi.org/10.1109/ICPC.2017.7)]
- [181] Almeida DA, Murphy GC, Wilson G, Hoye M. Investigating whether and how software developers understand open source software licensing. *Empirical Software Engineering*, 2019, 24(1): 211–239. [doi: [10.1007/s10664-018-9614-9](https://doi.org/10.1007/s10664-018-9614-9)]
- [182] Moraes JP, Polato I, Wiese I, Saraiva F, Pinto G. From one to hundreds: Multi-licensing in the JavaScript ecosystem. *Empirical Software Engineering*, 2021, 26(3): 39. [doi: [10.1007/s10664-020-09936-2](https://doi.org/10.1007/s10664-020-09936-2)]
- [183] Viseur R, Robles G. First results about motivation and impact of license changes in open source projects. In: Proc. of the 11th IFIP WG 2.13 Int'l Conf. on Open Source Systems: Adoption and Impact. Florence: Springer, 2015. 137–145. [doi: [10.1007/978-3-319-17837-0_13](https://doi.org/10.1007/978-3-319-17837-0_13)]
- [184] Wu YH, Manabe Y, Kanda T, German DM, Inoue K. Analysis of license inconsistency in large collections of open source projects. *Empirical Software Engineering*, 2017, 22(3): 1194–1222. [doi: [10.1007/s10664-016-9487-8](https://doi.org/10.1007/s10664-016-9487-8)]
- [185] Wang Y, Wen M, Liu ZW, Wu RX, Wang R, Yang B, Yu H, Zhu ZL, Cheung SC. Do the dependency conflicts in my project matter? In: Proc. of the 26th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Lake Buena: ACM, 2018. 319–330. [doi: [10.1145/3236024.3236056](https://doi.org/10.1145/3236024.3236056)]
- [186] Wang Y, Wen M, Wu RX, Liu ZW, Tan SH, Zhu ZL, Yu H, Cheung SC. Could I have a stack trace to examine the dependency conflict issue? In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Montreal: IEEE, 2019. 572–583. [doi: [10.1109/ICSE.2019.00068](https://doi.org/10.1109/ICSE.2019.00068)]
- [187] Wang Y, Qiao L, Xu C, Liu YP, Cheung SC, Meng N, Yu H, Zhu ZL. Hero: On the chaos when PATH meets modules. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Madrid: IEEE, 2021. 99–111. [doi: [10.1109/ICSE43902.2021.00022](https://doi.org/10.1109/ICSE43902.2021.00022)]
- [188] Li ZM, Wang Y, Lin ZQ, Cheung SC, Lou JG. Nufix: Escape from nuget dependency maze. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Pittsburgh: IEEE, 2022. 1545–1557. [doi: [10.1145/3510003.3510118](https://doi.org/10.1145/3510003.3510118)]
- [189] Jezek K, Dietrich J. On the use of static analysis to safeguard recursive dependency resolution. In: Proc. of the 40th EUROMICRO Conf. on Software Engineering and Advanced Applications. Verona: IEEE, 2014. 166–173. [doi: [10.1109/SEAA.2014.35](https://doi.org/10.1109/SEAA.2014.35)]

- [190] Fan G, Wang CP, Wu RX, Xiao X, Shi QK, Zhang C. Escaping dependency hell: Finding build dependency errors with the unified dependency graph. In: Proc. of the 29th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2020. 463–474. [doi: [10.1145/3395363.3397388](https://doi.org/10.1145/3395363.3397388)]
- [191] Soto-Valero C, Harrand N, Monperrus M, Baudry B. A comprehensive study of bloated dependencies in the Maven ecosystem. *Empirical Software Engineering*, 2021, 26(3): 45. [doi: [10.1007/s10664-020-09914-8](https://doi.org/10.1007/s10664-020-09914-8)]
- [192] Soto-Valero C, Durieux T, Baudry B. A longitudinal analysis of bloated Java dependencies. In: Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Athens: ACM, 2021. 1021–1031. [doi: [10.1145/3468264.3468589](https://doi.org/10.1145/3468264.3468589)]
- [193] Shah SMA, Dietrich J, McCartin C. On the automation of dependency-breaking refactorings in Java. In: Proc. of the 2013 IEEE Int'l Conf. on Software Maintenance. Eindhoven: IEEE, 2013. 160–169. [doi: [10.1109/ICSM.2013.27](https://doi.org/10.1109/ICSM.2013.27)]
- [194] de Matos AS, Filho JBF, Rocha LS. Splitting APIs: An exploratory study of software unbundling. In: Proc. of the 16th IEEE/ACM Int'l Conf. on Mining Software Repositories (MSR). Montreal: IEEE, 2019. 360–370. [doi: [10.1109/MSR.2019.00062](https://doi.org/10.1109/MSR.2019.00062)]
- [195] Azad BA, Laperdrix P, Nikiforakis N. Less is more: Quantifying the security benefits of debloating Web applications. In: Proc. of the 28th USENIX Security Symp. Santa Clara: USENIX Association, 2019. 1697–1714.
- [196] Vázquez HC, Bergel A, Vidal S, Díaz Pace JA, Marcos C. Slimming JavaScript applications: An approach for removing unused functions from JavaScript libraries. *Information and Software Technology*, 2019, 107: 18–29. [doi: [10.1016/j.infsof.2018.10.009](https://doi.org/10.1016/j.infsof.2018.10.009)]
- [197] Qian CX, Koo H, Oh C, Kim T, Lee W. Slimium: Debloating the chromium browser with feature subsetting. In: Proc. of the 2020 ACM SIGSAC Conf. on Computer and Communications Security. Virtual Event: ACM, 2020. 461–476. [doi: [10.1145/3372297.3417866](https://doi.org/10.1145/3372297.3417866)]
- [198] Qian CX, Hu H, Alharthi M, Chung PH, Kim T, Lee W. RAZOR: A framework for post-deployment software debloating. In: Proc. of the 28th USENIX Security Symp. Santa Clara: USENIX Association, 2019. 1733–1750.
- [199] Heo K, Lee W, Pashakhanloo P, Naik M. Effective program debloating via reinforcement learning. In: Proc. of the 2018 ACM SIGSAC Conf. on Computer and Communications Security. Toronto: ACM, 2018. 380–394. [doi: [10.1145/3243734.3243838](https://doi.org/10.1145/3243734.3243838)]
- [200] Grace MC, Zhou W, Jiang XX, Sadeghi AR. Unsafe exposure analysis of mobile in-APP advertisements. In: Proc. of the 5th ACM Conf. on Security and Privacy in Wireless and Mobile Networks. Tucson: ACM, 2012. 101–112. [doi: [10.1145/2185448.2185464](https://doi.org/10.1145/2185448.2185464)]
- [201] Yang WB, Li JR, Zhang YY, Li Y, Shu JL, Gu DW. APKLancet: Tumor payload diagnosis and purification for Android applications. In: Proc. of the 9th ACM Symp. on Information, Computer and Communications Security. Kyoto: ACM, 2014. 483–494. [doi: [10.1145/2590296.2590314](https://doi.org/10.1145/2590296.2590314)]
- [202] Shekhar S, Dietz M, Wallach DS. AdSplit: Separating smartphone advertising from applications. In: Proc. of the 21st USENIX Conf. on Security Symp. Bellevue: USENIX Association, 2012. 28.
- [203] Parrend P, Frénot S. Classification of component vulnerabilities in Java service oriented programming (SOP) platforms. In: Proc. of the 11th Int'l Symp. on Component-based Software Engineering. Karlsruhe: Springer, 2008. 80–96. [doi: [10.1007/978-3-540-87891-9_6](https://doi.org/10.1007/978-3-540-87891-9_6)]
- [204] Vasilakis N, Benetopoulos A, Handa S, Schoen A, Shen JS, Rinard MC. Supply-chain vulnerability elimination via active learning and regeneration. In: Proc. of the 2021 ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2021. 1755–1770. [doi: [10.1145/3460120.3484736](https://doi.org/10.1145/3460120.3484736)]
- [205] Dashevskiy S, Brucker AD, Massacci F. A screening test for disclosed vulnerabilities in FOSS components. *IEEE Trans. on Software Engineering*, 2019, 45(10): 945–966. [doi: [10.1109/TSE.2018.2816033](https://doi.org/10.1109/TSE.2018.2816033)]
- [206] Gkortzis A, Feitosa D, Spinellis D. A double-edged sword? Software reuse and potential security vulnerabilities. In: Proc. of the 18th Int'l Conf. on Software and Systems Reuse. Cincinnati: Springer, 2019. 187–203. [doi: [10.1007/978-3-030-22888-0_13](https://doi.org/10.1007/978-3-030-22888-0_13)]
- [207] Walden J. The impact of a major security event on an open source project: The case of OpenSSL. In: Proc. of the 17th Int'l Conf. on Mining Software Repositories. Seoul: ACM, 2020. 409–419. [doi: [10.1145/3379597.3387465](https://doi.org/10.1145/3379597.3387465)]
- [208] Mojica IJ, Adams B, Nagappan M, Dienst S, Berger T, Hassan AE. A large-scale empirical study on software reuse in mobile APPs. *IEEE Software*, 2014, 31(2): 78–86. [doi: [10.1109/MS.2013.142](https://doi.org/10.1109/MS.2013.142)]
- [209] Massacci F, Neuhaus S, Nguyen VH. After-life vulnerabilities: A study on FireFox evolution, its vulnerabilities, and fixes. In: Proc. of the 3rd Int'l Symp. on Engineering Secure Software and Systems. Madrid: Springer, 2011. 195–208. [doi: [10.1007/978-3-642-19125-1_15](https://doi.org/10.1007/978-3-642-19125-1_15)]
- [210] Ferreira G, Jia LM, Sunshine J, Kästner C. Containing malicious package updates in npm with a lightweight permission system. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering. Madrid: IEEE, 2021. 1334–1346. [doi: [10.1109/ICSE43902.2021.00121](https://doi.org/10.1109/ICSE43902.2021.00121)]
- [211] Nikiforakis N, Invernizzi L, Kapravelos A, Van Acker S, Joosen W, Kruegel C, Piessens F, Vigna G. You are what you include: Large-scale evaluation of remote JavaScript inclusions. In: Proc. of the 2012 ACM Conf. on Computer and Communications Security. Raleigh:

- ACM, 2012. 736–747. [doi: [10.1145/2382196.2382274](https://doi.org/10.1145/2382196.2382274)]
- [212] Duan R, Alrawi O, Pai Kasturi R, Elder R, Saltaformaggio B, Lee W. Towards measuring supply chain attacks on package managers for interpreted languages. arXiv:2002.01139, 2020.
- [213] Ohm M, Plate H, Sykosch A, Meier M. Backstabber’s knife collection: A review of open source software supply chain attacks. In: Proc. of the 17th Int’l Conf. on Detection of Intrusions and Malware, and Vulnerability Assessment. Lisbon: Springer, 2020. 23–43. [doi: [10.1007/978-3-030-52683-2_2](https://doi.org/10.1007/978-3-030-52683-2_2)]
- [214] Taylor M, Vaidya R, Davidson D, de Carli L, Rastogi V. Defending against package typosquatting. In: Proc. of the 14th Int’l Conf. on Network and System Security. Melbourne: Springer, 2020. 112–131. [doi: [10.1007/978-3-030-65745-1_7](https://doi.org/10.1007/978-3-030-65745-1_7)]
- [215] Kwon T, Su ZD. Automatic detection of unsafe dynamic component loadings. IEEE Trans. on Software Engineering, 2012, 38(2): 293–313. [doi: [10.1109/TSE.2011.108](https://doi.org/10.1109/TSE.2011.108)]
- [216] Sejfia A, Schäfer M. Practical automated detection of malicious npm packages. In: Proc. of the 44th IEEE/ACM Int’l Conf. on Software Engineering (ICSE). Pittsburgh: IEEE, 2022. 1681–1692. [doi: [10.1145/3510003.3510104](https://doi.org/10.1145/3510003.3510104)]
- [217] Parrend P. Enhancing automated detection of vulnerabilities in Java components. In: Proc. of the 2009 Int’l Conf. on Availability, Reliability and Security. Fukuoka: IEEE, 2009. 216–223. [doi: [10.1109/ARES.2009.9](https://doi.org/10.1109/ARES.2009.9)]
- [218] Hu WH, Oceau D, McDaniel PD, Liu P. Duet: Library integrity verification for Android applications. In: Proc. of the 2014 ACM Conf. on Security and Privacy in Wireless & Mobile Networks. Oxford: ACM, 2014. 141–152. [doi: [10.1145/2627393.2627404](https://doi.org/10.1145/2627393.2627404)]
- [219] Demetriou S, Merrill W, Yang W, Zhang A, Gunter CA. Free for all! Assessing user data exposure to advertising libraries on Android. In: Proc. of the 2016 NDSS. San Diego, 2016. [doi: [10.14722/ndss.2016.23082](https://doi.org/10.14722/ndss.2016.23082)]
- [220] Musch M, Steffens M, Roth S, Stock B, Johns M. ScriptProtect: Mitigating unsafe third-party JavaScript practices. In: Proc. of the 2019 ACM Asia Conf. on Computer and Communications Security. Auckland: ACM, 2019. 391–402. [doi: [10.1145/3321705.3329841](https://doi.org/10.1145/3321705.3329841)]
- [221] Stevens R, Gibler C, Crussell J, Erickson J, Chen H. Investigating user privacy in Android ad libraries. In: Proc. of the 2012 Workshop on Mobile Security Technologies (MoST). 2012.
- [222] Wang FB, Zhang YQ, Wang K, Liu P, Wang WJ. Stay in your Cage! A sound sandbox for third-party libraries on Android. In: Proc. of the 21st European Symp. on Research in Computer Security. Heraklion: Springer, 2016. 458–476. [doi: [10.1007/978-3-319-45744-4_23](https://doi.org/10.1007/978-3-319-45744-4_23)]
- [223] Son S, Kim D, Shmatikov V. What mobile ads know about mobile users. In: Proc. of the NDSS 2016. San Deigo, 2016. [doi: [10.14722/ndss.2016.23407](https://doi.org/10.14722/ndss.2016.23407)]
- [224] Paturi A, Kelley P, Mazumdar S. Introducing privacy threats from ad libraries to Android users through privacy granules. In: Proc. of the 2015 NDSS Workshop on Usable Security (USEC 2015). San Deigo, 2015. [doi: [10.14722/usec.2015.23008](https://doi.org/10.14722/usec.2015.23008)]
- [225] Wang YF, Hariharan S, Zhao CX, Liu JM, Du WL. Compac: Enforce component-level access control in Android. In: Proc. of the 4th ACM Conf. on Data and Application Security and Privacy. San Antonio: ACM, 2014. 25–36. [doi: [10.1145/2557547.2557560](https://doi.org/10.1145/2557547.2557560)]
- [226] Sun MT, Tan G. NativeGuard: Protecting Android applications from third-party native libraries. In: Proc. of the 2014 ACM Conf. on Security and Privacy in Wireless & Mobile Networks. Oxford: ACM, 2014. 165–176. [doi: [10.1145/2627393.2627396](https://doi.org/10.1145/2627393.2627396)]
- [227] Felt AP, Chin E, Hanna S, Song D, Wagner D. Android permissions demystified. In: Proc. of the 18th ACM Conf. on Computer and Communications Security. Chicago: ACM, 2011. 627–636. [doi: [10.1145/2046707.2046779](https://doi.org/10.1145/2046707.2046779)]
- [228] Politz JG, Eliopoulos SA, Guha A, Krishnamurthi S. ADSafety: Type-based verification of JavaScript sandboxing. In: Proc. of the 20th USENIX Security Symp. San Francisco: USENIX Association, 2011. 1–16.
- [229] Narayan S, Disselkoe C, Garfinkel T, Froyd N, Rahm E, Lerner S, Shacham H, Stefan D. Retrofitting fine grain isolation in the FireFox renderer. In: Proc. of the 29th USENIX Conf. on Security Symp. USENIX Association, 2020. 699–716.
- [230] Vasilakis N, Staicu CA, Ntousakis G, Kallas K, Karel B, DeHon A, Pradel M. Preventing dynamic library compromise on Node.js via RWX-based privilege reduction. In: Proc. of the 2021 ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2021. 1821–1838. [doi: [10.1145/3460120.3484535](https://doi.org/10.1145/3460120.3484535)]
- [231] de Groef W, Massacci F, Piessens F. NodeSentry: Least-privilege library integration for server-side JavaScript. In: Proc. of the 30th Annual Computer Security Applications Conf. New Orleans: ACM, 2014. 446–455. [doi: [10.1145/2664243.2664276](https://doi.org/10.1145/2664243.2664276)]
- [232] Seo J, Kim D, Cho D, Shin I. FlexDroid: Enforcing in-app privilege separation in Android. In: Proc. of the 2016 NDSS. San Diego, 2016. [doi: [10.14722/ndss.2016.23485](https://doi.org/10.14722/ndss.2016.23485)]
- [233] Dietrich J, Pearce D, Stringer J, Tahir A, Blincoe K. Dependency versioning in the wild. In: Proc. of the 2019 IEEE/ACM 16th Int’l Conf. on Mining Software Repositories (MSR). Montreal: IEEE, 2019. 349–359. [doi: [10.1109/MSR.2019.00061](https://doi.org/10.1109/MSR.2019.00061)]
- [234] Decan A, Mens T. What do package dependencies tell us about semantic versioning? IEEE Trans. on Software Engineering, 2021, 47(6): 1226–1240. [doi: [10.1109/TSE.2019.2918315](https://doi.org/10.1109/TSE.2019.2918315)]
- [235] Li L, Gao J, Bissyandé TF, Ma L, Xia X, Klein J. Characterising deprecated Android APIs. In: Proc. of the 15th Int’l Conf. on Mining

- Software Repositories. Gothenburg: ACM, 2018. 254–264. [doi: [10.1145/3196398.3196419](https://doi.org/10.1145/3196398.3196419)]
- [236] Li L, Gao J, Bissyandé TF, Ma L, Xia X, Klein J. CDA: Characterising deprecated Android APIs. *Empirical Software Engineering*, 2020, 25(3): 2058–2098. [doi: [10.1007/s10664-019-09764-z](https://doi.org/10.1007/s10664-019-09764-z)]
- [237] Scalabrino S, Bavota G, Linares-Vásquez M, Lanza M, Oliveto R. Data-driven solutions to detect API compatibility issues in Android: An empirical study. In: *Proc. of the 16th Int'l Conf. on Mining Software Repositories*. Montreal: IEEE, 2019. 288–298. [doi: [10.1109/MSR.2019.00055](https://doi.org/10.1109/MSR.2019.00055)]
- [238] Hora A, Robbes R, Valente MT, Anquetil N, Etien A, Ducasse S. How do developers react to API evolution? A large-scale empirical study. *Software Quality Journal*, 2018, 26(1): 161–191. [doi: [10.1007/s11219-016-9344-4](https://doi.org/10.1007/s11219-016-9344-4)]
- [239] Bogart C, Kästner C, Herbsleb J, Thung F. How to break an API: Cost negotiation and community values in three software ecosystems. In: *Proc. of the 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering*. Seattle: ACM, 2016. 109–120. [doi: [10.1145/2950290.2950325](https://doi.org/10.1145/2950290.2950325)]
- [240] Xavier L, Brito A, Hora A, Valente MT. Historical and impact analysis of API breaking changes: A large-scale study. In: *Proc. of the 24th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER)*. Klagenfurt: IEEE, 2017. 138–147. [doi: [10.1109/SANER.2017.7884616](https://doi.org/10.1109/SANER.2017.7884616)]
- [241] Wu W, Khomh F, Adams B, Guéhéneuc YG, Antoniol G. An exploratory study of API changes and usages based on Apache and eclipse ecosystems. *Empirical Software Engineering*, 2016, 21(6): 2366–2412. [doi: [10.1007/s10664-015-9411-7](https://doi.org/10.1007/s10664-015-9411-7)]
- [242] He DJ, Li L, Wang L, Zheng HJ, Li GW, Xue JL. Understanding and detecting evolution-induced compatibility issues in Android APPs. In: *Proc. of the 33rd IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE)*. Montpellier: IEEE, 2018. 167–177. [doi: [10.1145/3238147.3238185](https://doi.org/10.1145/3238147.3238185)]
- [243] Møller A, Nielsen BB, Torp MT. Detecting locations in JavaScript programs affected by breaking library changes. *Proc. of the ACM on Programming Languages*, 2020, 4: 187. [doi: [10.1145/3428255](https://doi.org/10.1145/3428255)]
- [244] Nielsen BB, Torp MT, Møller A. Semantic patches for adaptation of JavaScript programs to evolving libraries. In: *Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE)*. Madrid: IEEE, 2021. 74–85. [doi: [10.1109/ICSE43902.2021.00020](https://doi.org/10.1109/ICSE43902.2021.00020)]
- [245] Ochoa L, Degueule T, Falleri JR, Vinju J. Breaking bad? Semantic versioning and impact of breaking changes in Maven Central. *Empirical Software Engineering*, 2022, 27(3): 61. [doi: [10.1007/S10664-021-10052-Y](https://doi.org/10.1007/S10664-021-10052-Y)]
- [246] Ponomarenko A, Rubanov V. Backward compatibility of software interfaces: Steps towards automatic verification. *Programming and Computer Software*, 2012, 38(5): 257–267. [doi: [10.1134/S0361768812050052](https://doi.org/10.1134/S0361768812050052)]
- [247] Chen LC, Hassan F, Wang XY, Zhang LM. Taming behavioral backward incompatibilities via cross-project testing and analysis. In: *Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering*. Seoul: IEEE, 2020. 112–124. [doi: [10.1145/3377811.3380436](https://doi.org/10.1145/3377811.3380436)]
- [248] Wang JW, Li L, Liu K, Cai HP. Exploring how deprecated Python library APIs are (not) handled. In: *Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering*. ACM, 2020. 233–244. [doi: [10.1145/3368089.3409735](https://doi.org/10.1145/3368089.3409735)]
- [249] Brito G, Hora A, Valente MT, Robbes R. On the use of replacement messages in API deprecation: An empirical study. *Journal of Systems and Software*, 2018, 137: 306–321. [doi: [10.1016/j.jss.2017.12.007](https://doi.org/10.1016/j.jss.2017.12.007)]
- [250] Liu P, Zhao YJ, Cai HP, Fazzini M, Grundy J, Li L. Automatically detecting API-induced compatibility issues in Android APPs: A comparative analysis (replicability study). In: *Proc. of the 31st ACM SIGSOFT Int'l Symp. on Software Testing and Analysis*. ACM, 2022. 617–628. [doi: [10.1145/3533767.3534407](https://doi.org/10.1145/3533767.3534407)]
- [251] Brito G, Hora A, Valente MT, Robbes R. Do developers deprecate APIs with replacement messages? A large-scale analysis on Java systems. In: *Proc. of the 23rd IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER)*. Osaka: IEEE, 2016. 360–369. [doi: [10.1109/SANER.2016.99](https://doi.org/10.1109/SANER.2016.99)]
- [252] Nascimento R, Figueiredo E, Hora A. JavaScript API deprecation landscape: A survey and mining study. *IEEE Software*, 2022, 39(3): 96–105. [doi: [10.1109/MS.2021.3103134](https://doi.org/10.1109/MS.2021.3103134)]
- [253] Zhou J, Walker RJ. API Deprecation: A retrospective analysis and detection method for code examples on the Web. In: *Proc. of the 24th ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering*. Seattle: ACM, 2016. 266–277. [doi: [10.1145/2950290.2950298](https://doi.org/10.1145/2950290.2950298)]
- [254] German DM, Manabe Y, Inoue K. A sentence-matching method for automatic license identification of source code files. In: *Proc. of the 25th IEEE/ACM Int'l Conf. on Automated Software Engineering*. Antwerp: ACM, 2010. 437–446. [doi: [10.1145/1858996.1859088](https://doi.org/10.1145/1858996.1859088)]
- [255] Di Penta M, German DM, Antoniol G. Identifying licensing of jar archives using a code-search approach. In: *Proc. of the 7th IEEE Working Conf. on Mining Software Repositories (MSR 2010)*. Cape Town: IEEE, 2010. 151–160. [doi: [10.1109/MSR.2010.5463282](https://doi.org/10.1109/MSR.2010.5463282)]
- [256] Scacchi W, Alspaugh TA. Understanding the role of licenses and evolution in open architecture software ecosystems. *Journal of Systems and Software*, 2012, 85(7): 1479–1494. [doi: [10.1016/j.jss.2012.03.033](https://doi.org/10.1016/j.jss.2012.03.033)]

- [257] Tuunanen T, Koskinen J, Kärkkäinen T. Automated software license analysis. *Automated Software Engineering*, 2009, 16(3–4): 455–490. [doi: [10.1007/s10515-009-0054-z](https://doi.org/10.1007/s10515-009-0054-z)]
- [258] Kapitsaki GM, Charalambous G. Find your open source license now! In: *Proc. of the 23rd Asia-Pacific Software Engineering Conf. (APSEC)*. Hamilton: IEEE, 2016. 1–8. [doi: [10.1109/APSEC.2016.012](https://doi.org/10.1109/APSEC.2016.012)]
- [259] Hemel A, Kalleberg KT, Vermaas R, Dolstra E. Finding software license violations through binary code clone detection. In: *Proc. of the 8th Working Conf. on Mining Software Repositories*. Waikiki: ACM Press, 2011. 63–72. [doi: [10.1145/1985441.1985453](https://doi.org/10.1145/1985441.1985453)]
- [260] Kapitsaki GM, Tselikas ND, Foukarakis IE. An insight into license tools for open source software systems. *Journal of Systems and Software*, 2015, 102: 72–87. [doi: [10.1016/j.jss.2014.12.050](https://doi.org/10.1016/j.jss.2014.12.050)]
- [261] German DM, González-Barahona JM. An empirical study of the reuse of software licensed under the GNU general public license. In: *Proc. of the 5th IFIP WG 2.13 Int'l Conf. on Open Source Systems*. Skövde: Springer, 2009. 185–198. [doi: [10.1007/978-3-642-02032-2_17](https://doi.org/10.1007/978-3-642-02032-2_17)]
- [262] Kapitsaki GM, Kramer F, Tselikas ND. Automating the license compatibility process in open source software with SPDX. *Journal of Systems and Software*, 2017, 131: 386–401. [doi: [10.1016/j.jss.2016.06.064](https://doi.org/10.1016/j.jss.2016.06.064)]
- [263] Xu SH, Gao Y, Fan LL, Liu ZL, Ji H. LiDetector: License incompatibility detection for open source software. *ACM Trans. on Software Engineering and Methodology*, 2023, 32(1): 22. [doi: [10.1145/3518994](https://doi.org/10.1145/3518994)]
- [264] German DM, Di Penta M, Davies J. Understanding and auditing the licensing of open source software distributions. In: *Proc. of the 18th IEEE Int'l Conf. on Program Comprehension*. Braga: IEEE, 2010. 84–93. [doi: [10.1109/ICPC.2010.48](https://doi.org/10.1109/ICPC.2010.48)]
- [265] German DM, Hassan AE. License integration patterns: Addressing license mismatches in component-based development. In: *Proc. of the 31st IEEE Int'l Conf. on Software Engineering*. Vancouver: IEEE, 2009. 188–198. [doi: [10.1109/ICSE.2009.5070520](https://doi.org/10.1109/ICSE.2009.5070520)]
- [266] Alspaugh TA, Scacchi W, Asuncion HU. Software licenses in context: The challenge of heterogeneously-licensed systems. *Journal of the Association for Information Systems*, 2010, 11(11): 730–755. [doi: [10.17705/1jais.00241](https://doi.org/10.17705/1jais.00241)]
- [267] van der Burg S, Dolstra E, McIntosh S, Davies J, German DM, Hemel A. Tracing software build processes to uncover license compliance inconsistencies. In: *Proc. of the 29th ACM/IEEE Int'l Conf. on Automated Software Engineering*. Vasteras: ACM, 2014. 731–742. [doi: [10.1145/2642937.2643013](https://doi.org/10.1145/2642937.2643013)]
- [268] Vendome C, Bavota G, Penta MD, Linares-Vásquez M, German D, Poshyvanyk D. License usage and changes: A large-scale study on GitHub. *Empirical Software Engineering*, 2017, 22(3): 1537–1577. [doi: [10.1007/s10664-016-9438-4](https://doi.org/10.1007/s10664-016-9438-4)]
- [269] Vendome C, Linares-Vasquez M, Bavota G, Di Penta M, German D, Poshyvanyk D. License usage and changes: A large-scale study of Java projects on GitHub. In: *Proc. of the 23rd IEEE Int'l Conf. on Program Comprehension*. Florence: IEEE, 2015. 218–228. [doi: [10.1109/ICPC.2015.32](https://doi.org/10.1109/ICPC.2015.32)]
- [270] Wu YH, Manabe Y, Kanda T, German DM, Inoue K. A method to detect license inconsistencies in large-scale open source projects. In: *Proc. of the 12th IEEE/ACM Working Conf. on Mining Software Repositories*. Florence: IEEE, 2015. 324–333. [doi: [10.1109/MSR.2015.37](https://doi.org/10.1109/MSR.2015.37)]
- [271] Wu YH, Manabe Y, German DM, Inoue K. How are developers treating license inconsistency issues? A case study on license inconsistency evolution in FOSS projects. In: *Proc. of the 13th IFIP WG 2.13 Int'l Conf. on Open Source Systems (OSS)*. Buenos Aires: Springer, 2017. 69–79. [doi: [10.1007/978-3-319-57735-7_8](https://doi.org/10.1007/978-3-319-57735-7_8)]
- [272] Vendome C, Linares-Vásquez M, Bavota G, Di Penta M, German D, Poshyvanyk D. Machine learning-based detection of open source license exceptions. In: *Proc. of the 39th IEEE/ACM Int'l Conf. on Software Engineering (ICSE)*. Buenos Aires: IEEE, 2017. 118–129. [doi: [10.1109/ICSE.2017.19](https://doi.org/10.1109/ICSE.2017.19)]
- [273] Wang Y, Wu RX, Wang C, Wen M, Liu YP, Cheung SC, Yu H, Xu C, Zhu ZL. Will dependency conflicts affect my program's semantics? *IEEE Trans. on Software Engineering*, 2022, 48(7): 2295–2316. [doi: [10.1109/TSE.2021.3057767](https://doi.org/10.1109/TSE.2021.3057767)]
- [274] Patra J, Dixit PN, Pradel M. ConflictJS: Finding and understanding conflicts between JavaScript libraries. In: *Proc. of the 40th IEEE/ACM Int'l Conf. on Software Engineering*. Gothenburg: IEEE, 2018. 741–751. [doi: [10.1145/3180155.3180184](https://doi.org/10.1145/3180155.3180184)]
- [275] Abate P, Di Cosmo R, Gesbert L, Le Fessant F, Treinen R, Zacchiroli S. Mining component repositories for installability issues. In: *Proc. of the 12th IEEE/ACM Working Conf. on Mining Software Repositories*. Florence: IEEE, 2015. 24–33. [doi: [10.1109/MSR.2015.10](https://doi.org/10.1109/MSR.2015.10)]
- [276] Abate P, Di Cosmo R, Treinen R, Zacchiroli S. Learning from the future of component repositories. *Science of Computer Programming*, 2014, 90: 93–115. [doi: [10.1016/j.scico.2013.06.007](https://doi.org/10.1016/j.scico.2013.06.007)]
- [277] Vouillon J, Di Cosmo R. Broken sets in software repository evolution. In: *Proc. of the 35th Int'l Conf. on Software Engineering (ICSE)*. San Francisco: IEEE, 2013. 412–421. [doi: [10.1109/ICSE.2013.6606587](https://doi.org/10.1109/ICSE.2013.6606587)]
- [278] Vouillon J, Cosmo RD. On software component co-installability. *ACM Trans. on Software Engineering and Methodology*, 2013, 22(4): 34. [doi: [10.1145/2522920.2522927](https://doi.org/10.1145/2522920.2522927)]

- [279] Artho C, Suzuki K, Di Cosmo R, Treinen R, Zacchiroli S. Why do software packages conflict? In: Proc. of the 9th IEEE Working Conf. on Mining Software Repositories (MSR). Zurich: IEEE, 2012. 141–150. [doi: [10.1109/MSR.2012.6224274](https://doi.org/10.1109/MSR.2012.6224274)]
- [280] Huang KF, Chen BH, Shi BW, Wang Y, Xu CY, Peng X. Interactive, effort-aware library version harmonization. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. ACM, 2020. 518–529. [doi: [10.1145/3368089.3409689](https://doi.org/10.1145/3368089.3409689)]
- [281] Wang C, Wu RX, Song HH, Shu JW, Li GQ. smartPip: A smart approach to resolving Python dependency conflict issues. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2022. 93. [doi: [10.1145/3551349.3560437](https://doi.org/10.1145/3551349.3560437)]
- [282] Agadakos I, Demarinis N, Jin D, Williams-King K, Alfajardo J, Shteinfeld B, Williams-King D, Kemerlis VP, Portokalidis G. Large-scale debloating of binary shared libraries. Digital Threats: Research and Practice, 2020, 1(4): 19. [doi: [10.1145/3414997](https://doi.org/10.1145/3414997)]
- [283] Soto-Valero C, Durieux T, Harrand N, Baudry B. Coverage-based debloating for Java bytecode. ACM Trans. on Software Engineering and Methodology, 2023, 32(2): 38.
- [284] Cao YL, Chen L, Ma WWY, Li YH, Zhou YM, Wang LZ. Towards better dependency management: A first look at dependency smells in Python projects. IEEE Trans. on Software Engineering, 2023, 49(4): 1741–1765. [doi: [10.1109/TSE.2022.3191353](https://doi.org/10.1109/TSE.2022.3191353)]
- [285] Jafari AJ, Costa DE, Abdalkareem R, Shihab E, Tsantalis N. Dependency smells in JavaScript projects. IEEE Trans. on Software Engineering, 2022, 48(10): 3790–3807. [doi: [10.1109/TSE.2021.3106247](https://doi.org/10.1109/TSE.2021.3106247)]
- [286] Pham NH, Nguyen TT, Nguyen HA, Nguyen TN. Detection of recurring software vulnerabilities. In: Proc. of the 25th IEEE/ACM Int'l Conf. on Automated Software Engineering. Antwerp: ACM, 2010. 447–456. [doi: [10.1145/1858996.1859089](https://doi.org/10.1145/1858996.1859089)]
- [287] Vu DL, Massacci F, Pashchenko I, Plate H, Sabetta A. LastPyMile: Identifying the discrepancy between sources and packages. In: Proc. of the 29th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Athens: ACM, 2021. 780–792. [doi: [10.1145/3468264.3468592](https://doi.org/10.1145/3468264.3468592)]
- [288] Goichon F, Salagnac G, Parrend P, Frénot S. Static vulnerability detection in Java service-oriented components. Journal of Computer Virology and Hacking Techniques, 2013, 9(1): 15–26. [doi: [10.1007/s11416-012-0172-1](https://doi.org/10.1007/s11416-012-0172-1)]
- [289] Staicu CA, Torp MT, Schäfer M, Möller A, Pradel M. Extracting taint specifications for JavaScript libraries. In: Proc. of the 42nd ACM/IEEE Int'l Conf. on Software Engineering. Seoul: IEEE, 2020. 198–209.
- [290] Bhoraskar R, Han S, Jeon J, Azim T, Chen S, Jung J, Nath S, Wang R, Wetherall D. Brahmastra: Driving APPs to test the security of third-party components. In: Proc. of the 23rd USENIX Security Symp. San Diego: USENIX Association, 2014. 1021–1036.
- [291] Li L, Bissyandé TF, Klein J, Le Traon Y. An investigation into the use of common libraries in Android APPs. In: Proc. of the 23rd IEEE Int'l Conf. on Software Analysis, Evolution, and Reengineering. Osaka: IEEE, 2016. 403–414. [doi: [10.1109/SANER.2016.52](https://doi.org/10.1109/SANER.2016.52)]
- [292] Li L, Riom T, Bissyandé TF, Wang HY, Klein J, Yves LT. Revisiting the impact of common libraries for Android-related investigations. Journal of Systems and Software, 2019, 154: 157–175. [doi: [10.1016/j.jss.2019.04.065](https://doi.org/10.1016/j.jss.2019.04.065)]
- [293] Woo S, Park S, Kim S, Lee H, Oh H. CENTRIS: A precise and scalable approach for identifying modified open-source software reuse. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering. Madrid: IEEE, 2021. 860–872. [doi: [10.1109/ICSE43902.2021.00083](https://doi.org/10.1109/ICSE43902.2021.00083)]
- [294] Titze D, Lux M, Schuette J. Ordol: Obfuscation-resilient detection of libraries in Android applications. In: Proc. of the 2017 IEEE Trustcom/BigDataSE/ICSS. Sydney: IEEE, 2017. 618–625. [doi: [10.1109/Trustcom/BigDataSE/ICSS.2017.292](https://doi.org/10.1109/Trustcom/BigDataSE/ICSS.2017.292)]
- [295] Liu B, Liu B, Jin H, Govindan R. Efficient privilege de-escalation for ad libraries in mobile APPs. In: Proc. of the 13th Annual Int'l Conf. on Mobile Systems, Applications, and Services. Florence: ACM, 2015. 89–103. [doi: [10.1145/2742647.2742668](https://doi.org/10.1145/2742647.2742668)]
- [296] Huang JJ, Xue B, Jiang JS, You W, Liang B, Wu JZ, Wu YJ. Scalably detecting third-party Android libraries with two-stage bloom filtering. IEEE Trans. on Software Engineering, 2023, 49(4): 2272–2284. [doi: [10.1109/TSE.2022.3215628](https://doi.org/10.1109/TSE.2022.3215628)]
- [297] Zhan X, Fan LL, Chen S, We F, Liu TM, Luo XP, Liu Y. ATVHunter: Reliable version detection of third-party libraries for vulnerability identification in Android applications. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Madrid: IEEE, 2021. 1695–1707. [doi: [10.1109/ICSE43902.2021.00150](https://doi.org/10.1109/ICSE43902.2021.00150)]
- [298] Zhan X, Fan LL, Liu TM, Chen S, Li L, Wang HY, Xu YF, Luo XP, Liu Y. Automated third-party library detection for Android applications: Are we there yet? In: Proc. of the 35th IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Melbourne: IEEE, 2020. 919–930.
- [299] Zhan X, Liu TM, Liu YP, Liu Y, Li L, Wang HY, Luo XP. A systematic assessment on Android third-party library detection tools. IEEE Trans. on Software Engineering, 2022, 48(11): 4249–4273. [doi: [10.1109/TSE.2021.3115506](https://doi.org/10.1109/TSE.2021.3115506)]
- [300] Zhang ZC, Diao WR, Hu CY, Guo SQ, Zuo CS, Li L. An empirical study of potentially malicious third-party libraries in Android APPs. In: Proc. of the 13th ACM Conf. on Security and Privacy in Wireless and Mobile Networks. Linz: ACM, 2020. 144–154. [doi: [10.1145/3395351.3399346](https://doi.org/10.1145/3395351.3399346)]

- [301] Xu J, Yuan QT. Libroad: Rapid, online, and accurate detection of TPLs on Android. *IEEE Trans. on Mobile Computing*, 2022, 21(1): 167–180. [doi: [10.1109/TMC.2020.3003336](https://doi.org/10.1109/TMC.2020.3003336)]
- [302] Zhang JX, Beresford AR, Kollmann SA. LibID: Reliable identification of obfuscated third-party Android libraries. In: *Proc. of the 28th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis*. Beijing: ACM, 2019. 55–65. [doi: [10.1145/3293882.3330563](https://doi.org/10.1145/3293882.3330563)]
- [303] Zhang Y, Dai JR, Zhang XH, Huang SR, Yang ZM, Yang M, Chen H. Detecting third-party libraries in Android applications with high precision and recall. In: *Proc. of the 25th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER)*. Campobasso: IEEE, 2018. 141–152. [doi: [10.1109/SANER.2018.8330204](https://doi.org/10.1109/SANER.2018.8330204)]
- [304] Wang Y, Wu HW, Zhang HL, Rountev A. ORLIS: Obfuscation-resilient library detection for Android. In: *Proc. of the 5th IEEE/ACM Int'l Conf. on Mobile Software Engineering and Systems (MOBILESoft)*. Gothenburg: IEEE, 2018. 13–23.
- [305] Han HM, Li RX, Tang JW. Identify and inspect libraries in Android applications. *Wireless Personal Communications*, 2018, 103(1): 491–503. [doi: [10.1007/s11277-018-5456-4](https://doi.org/10.1007/s11277-018-5456-4)]
- [306] Duan R, Bijlani A, Xu M, Lim T, Lee W. Identifying open-source license violation and 1-day security risk at large scale. In: *Proc. of the 2017 ACM SIGSAC Conf. on Computer and Communications Security*. Dallas: ACM, 2017. 2169–2185. [doi: [10.1145/3133956.3134048](https://doi.org/10.1145/3133956.3134048)]
- [307] Li MH, Wang P, Wang W, Wang S, Wu DH, Liu J, Xue R, Huo W, Zou W. Large-scale third-party library detection in Android markets. *IEEE Trans. on Software Engineering*, 2020, 46(9): 981–1003. [doi: [10.1109/TSE.2018.2872958](https://doi.org/10.1109/TSE.2018.2872958)]
- [308] Li MH, Wang W, Wang P, Wang S, Wu DH, Liu J, Xue R, Huo W. LibD: Scalable and precise third-party library detection in Android markets. In: *Proc. of the 39th IEEE/ACM Int'l Conf. on Software Engineering (ICSE)*. Buenos Aires: IEEE, 2017. 335–346. [doi: [10.1109/ICSE.2017.38](https://doi.org/10.1109/ICSE.2017.38)]
- [309] Backes M, Bugiel S, Derr E. Reliable third-party library detection in Android and its security applications. In: *Proc. of the 2016 ACM SIGSAC Conf. on Computer and Communications Security*. Vienna: ACM, 2016. 356–367. [doi: [10.1145/2976749.2978333](https://doi.org/10.1145/2976749.2978333)]
- [310] Ma Z, Wang HY, Guo Y, Chen XQ. LibRadar: Fast and accurate detection of third-party libraries in Android APPs. In: *Proc. of the 38th Int'l Conf. on Software Engineering Companion*. Austin: ACM, 2016. 653–656. [doi: [10.1145/2889160.2889178](https://doi.org/10.1145/2889160.2889178)]
- [311] Soh C, Kuan Tan HB, Armatovich YL, Narayanan A, Wang LP. LibSift: Automated detection of third-party libraries in Android applications. In: *Proc. of the 23rd Asia-Pacific Software Engineering Conf. (APSEC)*. Hamilton: IEEE, 2016. 41–48. [doi: [10.1109/APSEC.2016.017](https://doi.org/10.1109/APSEC.2016.017)]
- [312] Narayanan A, Chen LH, Chan CK. AdDetect: Automated detection of Android ad libraries using semantic analysis. In: *Proc. of the 9th IEEE Int'l Conf. on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*. Singapore: IEEE, 2014. 1–6. [doi: [10.1109/ISSNIP.2014.6827639](https://doi.org/10.1109/ISSNIP.2014.6827639)]
- [313] Ishio T, Kula RG, Kanda T, German DM, Inoue K. Software ingredients: Detection of third-party component reuse in Java software release. In: *Proc. of the 13th Int'l Conf. on Mining Software Repositories (MSR)*. Austin: ACM, 2016. 339–350. [doi: [10.1145/2901739.2901773](https://doi.org/10.1145/2901739.2901773)]
- [314] Davies J, German DM, Godfrey MW, Hindle A. Software bertillonage: Determining the provenance of software development artifacts. *Empirical Software Engineering*, 2013, 18(6): 1195–1237. [doi: [10.1007/s10664-012-9199-7](https://doi.org/10.1007/s10664-012-9199-7)]
- [315] Davies J, German DM, Godfrey MW, Hindle A. Software bertillonage: Finding the provenance of an entity. In: *Proc. of the 8th Working Conf. on Mining Software Repositories*. Waikiki: ACM, 2011. 183–192. [doi: [10.1145/1985441.1985468](https://doi.org/10.1145/1985441.1985468)]
- [316] Tang W, Chen D, Luo P. BCFinder: A lightweight and platform-independent tool to find third-party components in binaries. In: *Proc. of the 25th Asia-Pacific Software Engineering Conf. (APSEC)*. Nara: IEEE, 2018. 288–297. [doi: [10.1109/APSEC.2018.00043](https://doi.org/10.1109/APSEC.2018.00043)]
- [317] Tang W, Luo P, Fu JL, Zhang D. LibDX: A cross-platform and accurate system to detect third-party libraries in binary code. In: *Proc. of the 27th IEEE Int'l Conf. on Software Analysis, Evolution and Reengineering (SANER)*. London: IEEE, 2020. 104–115. [doi: [10.1109/SANER48275.2020.9054845](https://doi.org/10.1109/SANER48275.2020.9054845)]
- [318] Alqahtani SS, Eghan EE, Rilling J. Tracing known security vulnerabilities in software repositories—A semantic Web enabled modeling approach. *Science of Computer Programming*, 2016, 121: 153–175. [doi: [10.1016/j.scico.2016.01.005](https://doi.org/10.1016/j.scico.2016.01.005)]
- [319] Koishybayev I, Kapravelos A. Mininode: Reducing the attack surface of Node.js applications. In: *Proc. of the 23rd Int'l Symp. on Research in Attacks, Intrusions and Defenses*. USENIX Association, 2020. 121–134.
- [320] Lauinger T, Chaabane A, Arshad S, Robertson W, Wilson C, Kirda E. Thou shalt not depend on me: Analysing the use of outdated JavaScript libraries on the Web. In: *Proc. of the 2017 NDSS*. San Diego, 2017. [doi: [10.14722/ndss.2017.23414](https://doi.org/10.14722/ndss.2017.23414)]
- [321] Zhao BB, Ji SL, Xu JC, Tian Y, Wei QY, Wang QY, Lyu CY, Zhang XH, Lin CT, Wu JZ, Beyah R. A large-scale empirical analysis of the vulnerabilities introduced by third-party components in IoT firmware. In: *Proc. of the 31st ACM SIGSOFT Int'l Symp. on Software Testing and Analysis*. ACM, 2022. 442–454. [doi: [10.1145/3533767.3534366](https://doi.org/10.1145/3533767.3534366)]

- [322] Xu MQ, Wang Y, Cheung SC, Yu H, Zhu ZL. Insight: Exploring cross-ecosystem vulnerability impacts. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2022. 58. [doi: [10.1145/3551349.3556921](https://doi.org/10.1145/3551349.3556921)]
- [323] Liu CW, Chen S, Fan LL, Chen BH, Liu Y, Peng X. Demystifying the vulnerability propagation and its evolution via dependency trees in the npm ecosystem. In: Proc. of the 44th Int'l Conf. on Software Engineering. Pittsburgh: ACM, 2022. 672–684. [doi: [10.1145/3510003.3510142](https://doi.org/10.1145/3510003.3510142)]
- [324] Latendresse J, Mujahid S, Costa DE, Shihab E. Not all dependencies are equal: An empirical study on production dependencies in npm. In: Proc. of the 37th IEEE/ACM Int'l Conf. on Automated Software Engineering. Rochester: ACM, 2022. 73. [doi: [10.1145/3551349.3556896](https://doi.org/10.1145/3551349.3556896)]
- [325] Haryono SA, Kang HJ, Sharma A, Sharma A, Santosa A, Yi AM, Lo D. Automated identification of libraries from vulnerability data: Can we do better? In: Proc. of the 30th IEEE/ACM Int'l Conf. on Program Comprehension. Pittsburgh: IEEE, 2022. 178–189. [doi: [10.1145/3524610.3527893](https://doi.org/10.1145/3524610.3527893)]
- [326] Decan A, Mens T, Zerouali A, De Roover C. Back to the past—analysing backporting practices in package dependency networks. IEEE Trans. on Software Engineering, 2022, 48(10): 4087–4099. [doi: [10.1109/TSE.2021.3112204](https://doi.org/10.1109/TSE.2021.3112204)]
- [327] Costa DE, Mujahid S, Abdalkareem R, Shihab E. Breaking type safety in Go: An empirical study on the usage of the unsafe package. IEEE Trans. on Software Engineering, 2022, 48(7): 2277–2294. [doi: [10.1109/TSE.2021.3057720](https://doi.org/10.1109/TSE.2021.3057720)]
- [328] Ponta SE, Plate H, Sabetta A. Beyond metadata: Code-centric and usage-based analysis of known vulnerabilities in open-source software. In: Proc. of the 2018 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Madrid: IEEE, 2018. 449–460. [doi: [10.1109/ICSME.2018.00054](https://doi.org/10.1109/ICSME.2018.00054)]
- [329] Cadariu M, Bouwers E, Visser J, van Deursen A. Tracking known security vulnerabilities in proprietary software systems. In: Proc. of the 22nd IEEE Int'l Conf. on Software Analysis, Evolution, and Reengineering (SANER). Montreal: IEEE, 2015. 516–519. [doi: [10.1109/SANER.2015.7081868](https://doi.org/10.1109/SANER.2015.7081868)]
- [330] Boldi P, Gousios G. Fine-grained network analysis for modern software ecosystems. ACM Trans. on Internet Technology, 2021, 21(1): 1. [doi: [10.1145/3418209](https://doi.org/10.1145/3418209)]
- [331] Kang HJ, Nguyen TG, Le B, Păsăreanu CS, Lo D. Test mimicry to assess the exploitability of library vulnerabilities. In: Proc. of the 31st ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2022. 276–288. [doi: [10.1145/3533767.3534398](https://doi.org/10.1145/3533767.3534398)]
- [332] Dann A, Plate H, Hermann B, Ponta SE, Bodden E. Identifying challenges for OSS vulnerability scanners—A study & test suite. IEEE Transactions on Software Engineering, 2022, 48(8): 3613–3625. [doi: [10.1109/TSE.2021.3101739](https://doi.org/10.1109/TSE.2021.3101739)]
- [333] Imtiaz N, Thorn S, Williams L. A comparative study of vulnerability reporting by software composition analysis tools. In: Proc. of the 15th ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement (ESEM). Bari: ACM, 2021. 5. [doi: [10.1145/3475716.3475769](https://doi.org/10.1145/3475716.3475769)]
- [334] Chen Y, Santosa AE, Yi AM, Sharma A, Sharma A, Lo D. A machine learning approach for vulnerability curation. In: Proc. of the 17th Int'l Conf. on Mining Software Repositories. Seoul: ACM, 2020. 32–42. [doi: [10.1145/3379597.3387461](https://doi.org/10.1145/3379597.3387461)]
- [335] Pashchenko I, Plate H, Ponta SE, Sabetta A, Massacci F. Vulnerable open source dependencies: Counting those that matter. In: Proc. of the 12th ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement. Oulu: ACM, 2018. 42. [doi: [10.1145/3239235.3268920](https://doi.org/10.1145/3239235.3268920)]
- [336] Dashevskiy S, Brucker AD, Massacci F. On the effort for security maintenance of open source components. In: Proc. of the 2018 Workshop on the Economics of Information Security (WEIS). 2018.
- [337] Ponta SE, Plate H, Sabetta A. Detection, assessment and mitigation of vulnerabilities in open source dependencies. Empirical Software Engineering, 2020, 25(5): 3175–3215. [doi: [10.1007/s10664-020-09830-x](https://doi.org/10.1007/s10664-020-09830-x)]
- [338] Chen K, Wang XQ, Chen Y, Wang P, Lee Y, Wang XF, Ma B, Wang AH, Zhang YJ, Zou W. Following Devil's footprints: Cross-platform analysis of potentially harmful libraries on Android and iOS. In: Proc. of the 2016 IEEE Symp. on Security and Privacy (SP). San Jose: IEEE, 2016. 357–376. [doi: [10.1109/SP.2016.29](https://doi.org/10.1109/SP.2016.29)]
- [339] Zerouali A, Mens T, De Roover C. On the usage of JavaScript, Python and Ruby packages in Docker Hub images. Science of Computer Programming, 2021, 207: 102653. [doi: [10.1016/j.scico.2021.102653](https://doi.org/10.1016/j.scico.2021.102653)]
- [340] Xu CY, Chen BH, Lu CH, Huang KF, Peng X, Liu Y. Tracking patches for open source software vulnerabilities. In: Proc. of the 30th ACM Joint European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Singapore: ACM, 2022. 860–871. [doi: [10.1145/3540250.3549125](https://doi.org/10.1145/3540250.3549125)]
- [341] Watanabe T, Akiyama M, Kanei F, Shioji E, Takata Y, Sun B, Ishi Y, Shibahara T, Yagi T, Mori T. Understanding the origins of mobile app vulnerabilities: A large-scale measurement study of free and paid APPs. In: Proc. of the 14th IEEE/ACM Int'l Conf. on Mining Software Repositories (MSR). Buenos Aires: IEEE, 2017. 14–24. [doi: [10.1109/MSR.2017.23](https://doi.org/10.1109/MSR.2017.23)]

- [342] Black Duck. 2022. <https://www.blackducksoftware.com>
- [343] Greenkeeper. 2022. <https://greenkeeper.io>
- [344] Snyk. 2022. <https://snyk.io>
- [345] SourceClear. 2022. <https://www.sourceclear.com>
- [346] Pashchenko I, Vu DL, Massacci F. A qualitative study of dependency management and its security implications. In: Proc. of the 2020 ACM SIGSAC Conf. on Computer and Communications Security. ACM, 2020. 1513–1531. [doi: [10.1145/3372297.3417232](https://doi.org/10.1145/3372297.3417232)]
- [347] Mitropoulos D, Louridas P, Salis V, Spinellis D. Time present and time past: Analyzing the evolution of JavaScript code in the wild. In: Proc. of the 16th IEEE/ACM Int'l Conf. on Mining Software Repositories (MSR). Montreal: IEEE, 2019. 126–137. [doi: [10.1109/MSR.2019.00029](https://doi.org/10.1109/MSR.2019.00029)]
- [348] Mirhosseini S, Parnin C. Can automated pull requests encourage software developers to upgrade out-of-date dependencies? In: Proc. of the 32nd IEEE/ACM Int'l Conf. on Automated Software Engineering (ASE). Urbana: IEEE, 2017. 84–94. [doi: [10.1109/ASE.2017.8115621](https://doi.org/10.1109/ASE.2017.8115621)]
- [349] Mukherjee S, Almanza A, Rubio-González C. Fixing dependency errors for Python build reproducibility. In: Proc. of the 30th ACM SIGSOFT Int'l Symp. on Software Testing and Analysis. ACM, 2021. 439–451. [doi: [10.1145/3460319.3464797](https://doi.org/10.1145/3460319.3464797)]
- [350] Plate H, Ponta SE, Sabetta A. Impact assessment for vulnerabilities in open-source software libraries. In: Proc. of the 2015 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Bremen: IEEE, 2015. 411–420. [doi: [10.1109/ICSM.2015.7332492](https://doi.org/10.1109/ICSM.2015.7332492)]
- [351] Pashchenko I, Plate H, Ponta SE, Sabetta A, Massacci F. Vuln4Real: A methodology for counting actually vulnerable dependencies. IEEE Trans. on Software Engineering, 2022, 48(5): 1592–1609. [doi: [10.1109/TSE.2020.3025443](https://doi.org/10.1109/TSE.2020.3025443)]
- [352] Horton E, Parnin C. Gistable: Evaluating the executability of Python code snippets on GitHub. In: Proc. of the 2018 IEEE Int'l Conf. on Software Maintenance and Evolution (ICSME). Madrid: IEEE, 2018. 217–227. [doi: [10.1109/ICSME.2018.00031](https://doi.org/10.1109/ICSME.2018.00031)]
- [353] Horton E, Parnin C. DockerizeMe: Automatic inference of environment dependencies for Python code snippets. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Montreal: IEEE, 2019. 328–338. [doi: [10.1109/ICSE.2019.00047](https://doi.org/10.1109/ICSE.2019.00047)]
- [354] Wang JW, Li L, Zeller A. Restoring execution environments of jupyter notebooks. In: Proc. of the 43rd IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Madrid: IEEE, 2021. 1622–1633. [doi: [10.1109/ICSE43902.2021.00144](https://doi.org/10.1109/ICSE43902.2021.00144)]
- [355] Ye HJ, Chen W, Dou WS, Wu GQ, Wei J. Knowledge-based environment dependency inference for Python programs. In: Proc. of the 44th Int'l Conf. on Software Engineering. Pittsburgh: ACM, 2022. 1245–1256. [doi: [10.1145/3510003.3510127](https://doi.org/10.1145/3510003.3510127)]
- [356] Islam MJ, Nguyen G, Pan R, Rajan H. A comprehensive study on deep learning bug characteristics. In: Proc. of the 27th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. Tallinn: ACM, 2019. 510–520. [doi: [10.1145/3338906.3338955](https://doi.org/10.1145/3338906.3338955)]
- [357] Pham HV, Lutellier T, Qi WZ, Tan L. CRADLE: Cross-backend validation to detect and localize bugs in deep learning libraries. In: Proc. of the 41st IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Montreal: IEEE, 2019. 1027–1038. [doi: [10.1109/ICSE.2019.00107](https://doi.org/10.1109/ICSE.2019.00107)]
- [358] Wang JN, Lutellier T, Qian SS, Pham HV, Tan L. EAGLE: Creating equivalent graphs to test deep learning libraries. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Pittsburgh: IEEE, 2022. 798–810. [doi: [10.1145/3510003.3510165](https://doi.org/10.1145/3510003.3510165)]
- [359] Wang Z, Yan M, Chen JJ, Liu S, Zhang DD. Deep learning library testing via effective model generation. In: Proc. of the 28th ACM Joint Meeting on European Software Engineering Conf. and Symp. on the Foundations of Software Engineering. ACM, 2020. 788–799. [doi: [10.1145/3368089.3409761](https://doi.org/10.1145/3368089.3409761)]
- [360] Wei AJ, Deng YL, Yang CY, Zhang LM. Free lunch for testing: Fuzzing deep-learning libraries from open source. In: Proc. of the 44th IEEE/ACM Int'l Conf. on Software Engineering (ICSE). Pittsburgh: IEEE, 2022. 995–1007. [doi: [10.1145/3510003.3510041](https://doi.org/10.1145/3510003.3510041)]
- [361] Manikas K. Revisiting software ecosystems research: A longitudinal literature study. Journal of Systems and Software, 2016, 117: 84–103. [doi: [10.1016/j.jss.2016.02.003](https://doi.org/10.1016/j.jss.2016.02.003)]
- [362] Manikas K, Hansen KM. Software ecosystems—A systematic literature review. Journal of Systems and Software, 2013, 86(5): 1294–1306. [doi: [10.1016/j.jss.2012.12.026](https://doi.org/10.1016/j.jss.2012.12.026)]
- [363] Axelsson J, Skoglund M. Quality assurance in software ecosystems: A systematic literature mapping and research agenda. Journal of Systems and Software, 2016, 114: 69–81. [doi: [10.1016/j.jss.2015.12.020](https://doi.org/10.1016/j.jss.2015.12.020)]
- [364] Franco-Bedoya O, Ameller D, Costal D, Franch X. Open source software ecosystems: A systematic mapping. Information and Software Technology, 2017, 91: 160–185. [doi: [10.1016/j.infsof.2017.07.007](https://doi.org/10.1016/j.infsof.2017.07.007)]
- [365] Breivold HP, Chauhan MA, Babar MA. A systematic review of studies of open source software evolution. In: Proc. of the 2010 Asia Pacific Software Engineering Conf. Sydney: IEEE, 2010. 356–365. [doi: [10.1109/APSEC.2010.48](https://doi.org/10.1109/APSEC.2010.48)]

附中文参考文献:

- [1] 梅宏, 曹东刚, 谢涛. 泛在操作系统: 面向人机器融合泛在计算的新蓝海. 中国科学院院刊, 2022, 37(1): 30–37. [doi: 10.16418/j.issn.1000-3045.20211117009]
- [2] 梁冠宇, 武延军, 吴敬征, 赵琛. 面向操作系统可靠性保障的开源软件供应链. 软件学报, 2020, 31(10): 3056–3073. <http://www.jos.org.cn/1000-9825/6070.htm> [doi: 10.13328/j.cnki.jos.006070]
- [3] 金芝, 周明辉, 张宇霞. 开源软件与开源软件生态: 现状与趋势. 科技导报, 2016, 34(14): 42–48. [doi: 10.3981/j.issn.1000-7857.2016.14.005]
- [4] 许畅, 秦逸, 余萍, 曹春, 吕建. 可成长软件理论方法和实现技术: 从范型到跨越. 中国科学: 信息科学, 2020, 50(11): 1595–1611. [doi: 10.1360/SSI-2020-0079]



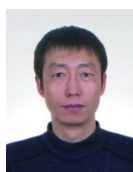
王莹(1987—), 女, 博士, 副教授, 博士生导师, CCF 专业会员, 主要研究领域为开源治理技术, 软件测试及分析学.



许畅(1977—), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为大数据软件工程, 智能软件测试与分析, 自适应和自控软件系统.



伍盈欣(1998—), 女, 硕士生, 主要研究领域为智能软件开发.



于海(1971—), 男, 博士, 副教授, 主要研究领域为软件测试, 软件重构, 软件体系结构, 复杂网络理论, 混沌加密技术.



高天(2000—), 男, 硕士生, 主要研究领域为智能软件开发.



张成志(1963—), 男, 博士, 教授, 博士生导师, 主要研究领域为软件测试, 机器学习, 程序分析, 开源软件供应链.



陈子懿(2001—), 女, 硕士生, 主要研究领域为智能软件开发.