

软件数字社会学

周明辉^{1,2*}, 张宇霞^{1,2}, 谭鑫^{1,2}

1. 北京大学信息科学技术学院, 北京 100871
2. 高可信软件技术教育部重点实验室(北京大学), 北京 100871
* 通信作者. E-mail: zhmh@pku.edu.cn

收稿日期: 2018-12-13; 接受日期: 2019-03-13; 网络出版日期: 2019-11-04

国家自然科学基金(批准号: 61432001, 61825201)和国家重点研发计划(批准号: 2018YFB10044200)资助项目

摘要 随着互联网不断发展, 软件开发(尤其是全球开源开发)面临诸多问题和挑战, 如分布在全球的开发者个体差异明显, 群体协作的困难度增加, 并且广泛的社会参与形成复杂生态等。这些问题使得软件开发呈现出很强的社会学特征。因此, 本文提出软件数字社会学来刻画和应对以上问题的挑战, 涉及个体学习、群体协作和可持续生态。本文对核心研究方法——软件开发活动数据的挖掘和分析进行了讨论, 并初步探讨了已经形成态势的开源供应链的重要问题。软件数字社会学可以启发研究者们更好地理解软件开发面临的关键挑战并探索更好的解决方案。

关键词 软件数字社会学, 软件活动数据, 个体学习, 群体协作, 开源生态, 软件供应链, 数据质量

1 什么是软件数字社会学

在当前的时代背景下, 开源像一股洪流席卷了全球软件产业。最近微软斥资 75 亿美元购买代码托管平台 GitHub, IBM 花 340 亿美金购买 RedHat, 这给软件行业带来的影响是深远的。全世界都在探讨开源的走向和利弊: GitHub 和 RedHat 的开源商业模式会改变吗? 开源开发者会继续给这些商业公司主导的社区贡献吗? 开源开发本身会受商业影响吗? 这些成功的开源生态还能持续下去吗? 这些讨论影射出了当前以开源为代表的全球分布式群体开发的重要特征——基于互联网的广泛的商业公司和个体开发者参与, 存在于软件、开发者和项目之间的错综复杂的依赖关系, 从开发到维护、应用和市场的一体化生态环境等。这些特征为个体学习和群体协作的效率和质量的提升带来新的挑战, 在某些方面突破了传统软件工程的研究边界, 非常契合社会学的研究范畴。

社会学是对个人与社会的关系以及对差异性结果的系统性研究, 以发展及完善一套有关人类社会结构及活动的知识体系, 并以运用这些知识去寻求或改善社会福利为目标^[1]。社会学家主要研究社会对人们的态度和行为的影响, 以及人们相互作用和改变社会的方式。社会学的研究对象涉及微观的个

引用格式: 周明辉, 张宇霞, 谭鑫. 软件数字社会学. 中国科学: 信息科学, 2019, 49: 1399–1411, doi: 10.1360/N112018-00319
Zhou M H, Zhang Y X, Tan X. Software digital sociology (in Chinese). Sci Sin Inform, 2019, 49: 1399–1411, doi: 10.1360/N112018-00319

体活动、个体和个体之间的关系, 以及宏观的全球化的系统结构和发展趋势。20世纪末兴起的信息科技浪潮使得社会学将其研究重点扩展到互联网相关行业, 并为社会学带来了崭新的计算分析技术, 例如社交网络^[2]。

因此, 我们提出软件数字社会学这个概念来刻画新的时代条件下复杂软件开发 (尤其是全球开源开发) 的挑战。为什么是软件数字社会学? 首先, 我们的研究对象是软件及其开发; 其次, 鉴于软件开发工具 (例如版本控制系统和缺陷追踪系统等) 的广泛使用, 其所积累的丰富的软件开发活动数据 (software activity data) 为我们提供了对复杂软件及其复杂开发的认知、实践和验证途径; 最后, 软件开发的复杂性归根结底体现在开发者个体、群体和生态的复杂性, 我们的研究可以很大程度借鉴社会学的研究思路和角度甚至是已有的研究结果进行展开。特别值得一提的是, 互联网所展现的网络效应不断深化, 正拓展到无处不在的物理世界, 逐渐形成了“人 – 机 – 物”三元融合的新型应用模式。这种模式使得软件数字社会学的内涵和外延更加丰富, 相比传统社会学, 其研究对象从个体及其关系延展到了“人 – 机 – 物”及其关系。

2 软件数字社会学的关键研究内容

自 1968 年“软件危机”^[3]以来, 大规模软件工程常被类比为困住恐龙的史前焦油坑, 其复杂性一直难以控制, 今天还在持续增长。例如, Linux 内核项目的代码行数几乎呈现指数级增长。现在一个发布版本通常包含约两百个公司、上千个开发者贡献的超过一万个代码包。在软件日益复杂化的情形下, 个体开发者如何快速学习和掌握复杂软件并积极高效的参与项目, 开发者群体如何高效高质地协作完成各类开发任务, 涵盖公司、个体开发者及用户在内的广泛社会力量如何围绕软件构建可持续性演化的生态系统等, 是软件数字社会学的主要研究内容。

2.1 个体学习

开发者个体作为软件开发过程中任务的承担者, 是影响软件开发质量和效率的关键因素。近年涌现的一些软件开发现象, 诸如全球化、越来越多的高层次认知任务外包、针对大规模复杂项目的高效参与需求, 正在改变人们思考、学习、工作以及合作的方式。开源软件开发这一低成本高质量以及透明开放的软件开发方式^[4]越来越受到人们的关注。与传统软件开发模式相比, 开源软件开发的开发者组织相对自由松散, 开发者的流动性较强。已有开发者的快速流失和新开发者的加入使得开源开发面临诸多风险和挑战。尤其是许多关键开源系统的复杂性不断增强^[5], 使得新人进入非常困难。因此, 个体开发者如何学习和掌握复杂系统成为了一个亟待解决的问题, 引起了工业界和学术界的广泛关注。

“个体学习”包含了两层含义。第一, 学习什么? 一些复杂的大型软件项目, 除了要求开发者掌握必要的编程技能, 项目所涉及的领域知识也需要开发者理解。除此之外, 开发者还需要学习如何与社区沟通 (根据 Astromskis 等^[6]的研究, 开发者近乎一半的工作时间都花费在沟通上), 需要熟悉使用各种开发支持工具——这些工具及其积累的数据也是分布式环境下传播知识和学习项目的重要媒介^[7]。第二, 如何学习? 一个开发者如何从初学新手成长为专业行家, 如何从初进社区的新手角色变为核心团队成员, 他怎么获取各种知识和信息, 又是以一种什么顺序掌握等。在开源项目中, 并无层级组织、训练计划、集中式的环境给予初学者必要的熏陶和培育, 更多的是“边干边学”(learning by doing)。研究那些高产的开发者的学习轨迹, 看他们如何获得技能、解决关键而复杂的任务, 了解是什么激励了他们, 也许能够将他们获得技能的方法具体化, 并进一步地用于训练其他人; 同时, 高手和新手解决问题的心理模型不一样, 因而在其指导新手时可能会出现失配问题 (zone of proximal development, 认知

科学的一个重要发现^[8]). 理解开源社区中失配问题的本质, 并建立相应技术和工具来更好适配新手的需求和高手的指导不仅能帮助个体成长, 也将帮助开源项目和社区的发展; 另外, 对于要处理各种复杂编程任务的程序员, 如何应对人类本身的认知局限性, 例如工作记忆系统容量有限、认知负荷等问题, 也是软件工程领域要考虑的问题. 总的来说, 我们需要对个体如何学习软件项目、如何使项目更具可读性 (self-documenting^[9])、如何创造革命性的资源和工具来提高新人的学习能力和生产力有更深入的理解. 为了解决上述问题, 我们的研究着重从以下两个方面开展.

首先, 研究如何利用数据来度量个体学习能力. 软件开发最根本问题在于开发者个体的复杂性, 主要挑战在于开发者 (学习) 能力的度量和评估——无法度量也就无法揭示问题所在, 进而难以控制其复杂性. 海量的软件开发历史数据和先进的分析方法和工具使得探索并解决该问题成为可能. 例如, 我们有一项工作通过对软件开发中的多源数据 (包括版本控制、任务追踪、人力资源等数据) 进行融合来度量开发者的开发任务随时间的变化, 对任务的价值和难度、所体现的开发者人际关系进行多维度的刻画, 以此来评估开发者经验的成长轨迹^[10,11]. 通过拟合广义线性相加模型观察任务开发者的效率或成熟度 (developer fluency) 随时间的变化, 我们发现, 尽管开发者所完成的任务数量在其加入项目的 20 月后达到平原状态, 但是综合任务难度、个体开发者的效率在三年内都是持续增加的. 上述程序员成熟度的度量框架可支持评估项目中程序员的技能与效率, 并支持对程序员成长轨迹的精确分析. 可以服务于为初学者选择合适的开发任务, 可以在项目面临外包或替换时量化程序员所需要的学习时间. 这些度量方法和结果为我们更好地探究软件项目个体开发者的学习和能力成长提供了坚实的基础和依据.

其次, 借鉴社会学的研究方法和洞见来研究软件开发的社会性特征及相关问题. 社会科学领域的诸多学科, 例如认知学、发展心理学、组织管理学, 已经对人类学习有许多研究. 更确切地说, 意识和脑、思考和学习的方式、解决问题过程中神经的行为和能力的提升都被集中地研究. 并且, 多个分支学科发现的证据存在交集^[12], 使得多学科交叉发展以追求对人类学习的更全面的认识成为一个重要趋势. 因此, 我们能借用这些学科的视角去理解开发者的学习. 例如 Curtis 论证认知学就是这样一个范例^[13], 它提供了很好的方式去理解影响项目表现的最重要因素. 此外, 我们还能基于对开发者学习的研究结果普适性地理解人类学习. 在过去的几十年里, 人们花费了大量的努力试图解开工作绩效的决定因素. 在一开始人们普遍认为导致异常表现的特征是天生的, 并且是遗传的^[14]. 然而, Ericsson 等^[15] 的研究表明, 许多曾经被认为反映天赋的特征实际上是经过了长达十年的紧张练习的结果. 我们的一项工作借用管理科学的一个框架来表征开源贡献者进入社区的初始行为特征^[16], 并利用其在社区的初始活动数据来度量该框架的三个维度, 如图 1 所示. 我们发现, 新加入的贡献者成为长期贡献者 (long term contributor, 在社区中持续贡献三年以上) 的可能性与他的能力 (ability)、意愿 (willingness) 以及环境因素 (environment) 有关, 例如, 其参与社区的意愿 (量化为其所报告的问题是否会得到解决, 表征了该问题的质量, 即报告者为报告该问题所倾入的精力) 能够成倍地增加其成为长期贡献者的概率^[10,14]. 这个工作建立了一个开发者初始行为度量体系, 支持对开发者长期贡献的分析预测.

2.2 群体协作

在经济力量不断将国内市场转变为国际市场的前提下, 全球化软件开发已经发展成为一种不可逆转的趋势^[17]. 分布在世界各地的软件开发者需要协同发布一个可用的高质量软件, 这将面临大规模的通信、协调和合作. 软件开发活动中涉及到的众多开发者能否积极地参与、良好地协作这些社会性的问题同样影响到软件开发的效率以及产品的质量.

基于互联网的软件开发活动, 以开源开发为代表, 基本都在开放透明的分布式环境下合作实施, 具

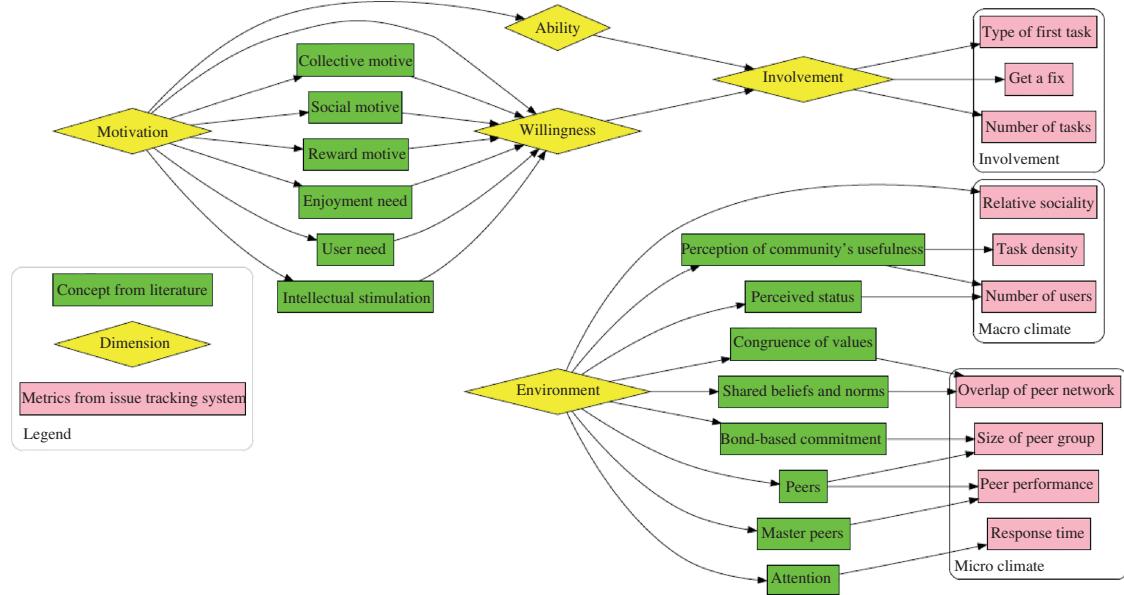


图 1 开发者行为特征的理论框架与相关量度

Figure 1 Theoretical framework and related measures of developer behavior's characteristics

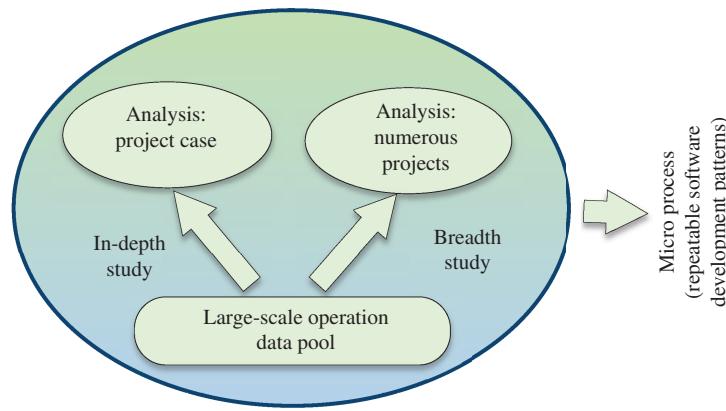


图 2 微过程研究模式图

Figure 2 Micro-process research pattern

有沟通媒介和形式多样化(例如版本控制系统、缺陷追踪系统等软件开发支撑工具和 Twitter, StackOverflow 等互联网新媒体)、协作内容和信息碎片化的特点,再加上软件项目上下文千差万别,使得找到通用的(可复现的)最佳实践非常困难,从而导致群体社会化开发的可知与可控难题有加剧的趋势。

我们认为,从大规模数据挖掘细粒度的微过程^[18],是找到可复现群体协作活动的可能途径。微过程是项目在完成各项特定任务(例如解决 bug, 提交代码, 沟通需求, 指导新手等)时所采用的方式方法或活动流程,对微过程的度量是获取细粒度可复制最佳实践的关键。如图 2 所示,微过程研究可以从广度和深度两个方面来开展,分这两种选择主要是从时间和成本上来考虑。广度研究是指,从尽可能广阔的视角上(例如覆盖尽可能多的项目)去探索问题。例如,我们以 GitHub 中海量软件项目为样本研究了项目中文件目录的使用模式及其对项目流行度的影响^[19],发现标准文件夹(例如文档、测试和

示例) 不仅是最常用的文件夹, 而且其使用与项目流行度(即项目 fork 数量)密切相关。对文件夹使用的初步研究表明, 可以借鉴频繁模式挖掘获得的文件夹使用模式, 量化和改进文件的组织实践。Ohira 等^[20] 对拥有大量开发人员和项目的大型在线社区 SourceForge.net 进行分析, 发现开发者群体协作关系呈现无标度网络(scale-free network)特征。即, 只有少数开发者加入了许多项目并与其它开发者建立了丰富的链接, 而大多数开发者加入的项目很少, 并且与其他项目的社交关系非常有限。通过对开发者社交网络的构建, 有助于帮助开发者定位沟通对象, 识别有经验的开发者, 从而提高群体协作的效率和质量。

深度研究是指以典型案例为研究对象进行深入探索。例如, 针对软件项目缺陷追踪工作流中的一个微过程: 产品定位, 我们发现了影响其质量的若干关键因子并提出了度量方法, 进而设计了一个工具 PAR (product assignment recommender) 用于预测缺陷报告是否被准确定位, 然后应用到了 Mozilla 十年历史中的 88000 个缺陷报告^[21]。结果表明, 该工具的精度达到 73.65%, 比随机预测的精确度高 3.4 倍, 有助于改进缺陷报告的质量。又例如, 我们研究了开源贡献者对同一代码文件进行修改的最佳合作模式。我们提出以集中度、复杂度和稳定性来刻画同一个文件的代码贡献组成, 并以 OpenStack 的核心项目 Nova 为例归纳出 3 种贡献组成模式。这些模式可以用来对代码贡献提出推荐, 例如, 对于逻辑复杂、代码量大的文件(例如实现系统核心功能的文件等), 一旦出现问题会直接导致系统不能正常工作, 最好由两名以上核心开发者进行开发和维护, 这样在某个开发者离开时, 不会出现无监管状态^[22]。以上研究都从多个维度对群体协作进行了度量, 所得出的研究结果可以为复杂的群体社会化开发提供相关建议。

2.3 可持续生态

随着开源的迅速发展, 社会力量不断涌入, 开源生态已经成为当下开源发展的一个核心关键词。它指商业组织、非盈利机构以及广大个体开发者和用户, 围绕核心开源技术(软件)的开发、应用和市场建立互利互补的依赖关系, 从而形成的生态系统。

开源生态系统由于汇聚了企业、开发者、开源基金会、产业联盟、政府、院校等众多参与方, 持续吸纳来自全球各界的贡献, 具有极快的发展速度和极大的创新潜力。随着协助开源软件开发和维护的现代平台的出现和不断完善, 如 GitHub, Bitbucket 和 GitLab, 开源软件项目正在以前所未有的速度增长^[23]。近年来, 开源生态系统在全球科技创新和产业发展领域的影响力不断增加, 例如 Apache 的 web 服务器主导了其所属的产品领域; Linux 内核、Android 和 OpenStack 等开源软件不仅是重要的计算基础设施, 也成为了我们社会生活依赖的基础设施。然而, 鉴于各种复杂因素的存在, 开源软件及生态的持续发展面临诸多潜在的风险和挑战, 主要表现在如下 3 个方面。

首先, 广泛的社会参与使得开源生态的管理控制复杂性激增。越来越多的商业组织和非营利机构参与到开源生态系统, 并扮演重要的角色。例如, 175 个公司在 OpenStack 最近发布的 Rocky 版本中贡献了近 90% 的代码¹⁾。来自各个行业、出于不同目标和动机的商业公司, 对开源生态的参与方式也千差万别^[24]。此外, 参与到同一开源生态系统的商业组织之间又会存在(或者产生)合作或竞争等关系。总的来说, 不断增加的各种组织, 其差别迥异的参与方式以及相互间错综复杂的关联关系, 都使得开源生态系统控制和管理的难度迅速且持续增加。

其次, 商业参与具有很大的不稳定性。开源生态系统中的商业公司始终需要在“开放”与“盈利”之间权衡, 一旦确定其目标无法完成或参与战略发生转变等, 就有可能直接停止(撤走)对开源软件的所有投入。特别是那些在开源社区占据主导地位的公司的离开, 将可能直接导致该开源生态走向失

1) OpenStack Foundation. http://stackalytics.com/?project_type=all&metric=commits. [2018-12-12].

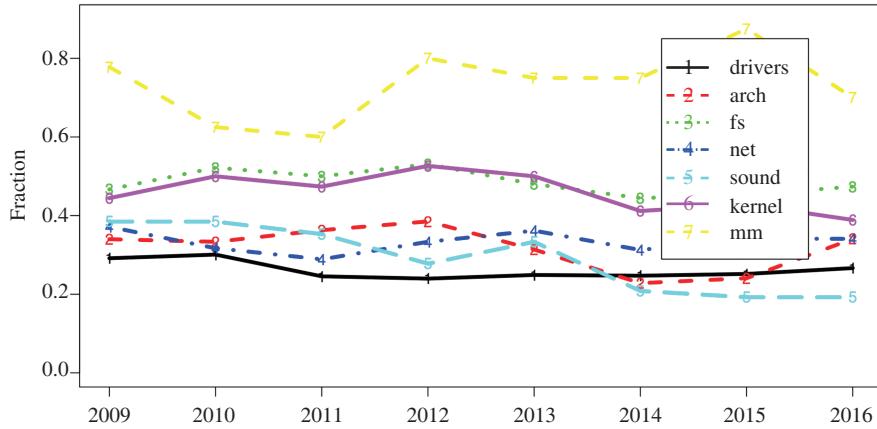


图 3 (网络版彩图) 负责审查 80% 代码提交的 MAINTAINER 比例

Figure 3 (Color online) Fraction of maintainers who are responsible for 80% of files

败^[25]. 例如, IBM 取消了对开源项目 Aperi 的支持后, 该项目被迫关闭^[26].

第三, 商业文化与自由开源精神存在冲突. 商业公司传统的软件开发经验习惯, 如组织安排、任务分工等可能伤害那些崇尚“黑客文化”、“自由软件精神”的个体开发者. 因为这类开源开发者更加倾向于自发参与、按照个人意愿选择任务, 例如目前世界第一大开源 JavaEE 应用服务器 JBossAS 的创始人 Marc Fleury 在 RedHat 收购 JBoss 之后离开²⁾.

因此, 探究开源生态系统构建及其持续发展的理论、方法与技术是软件数字社会学的又一大关键研究内容. 开源生态在近十多年的发展中得到了学术界和产业界的广泛关注. 我们认为, 其研究的进展及未来挑战主要体现在下述两个方面.

一方面是开源生态系统的形成和演化. 此类研究的主要目的是通过分析开源生态的历史数据度量开源生态的健康状态并建立可持续发展理论等. 例如, Gamalielsson 等^[27] 基于邮件列表中回复者的质量, 提出将响应能力作为开源生态系统健康度的评价指标. Lehman 等^[28] 提出开源生态系统的成熟理论用来评估软件的进化, 例如生存能力、增长潜力等. 我们有项工作是通过挖掘 Linux 内核的开发历史数据, 从 Maintainer 工作量的角度来探究影响 Linux kernel 生态系统可持续演化的关键因素^[5]. 结果表明 Linux kernel 的任务规模和 Maintainer 数量都随时间持续增长, 且尽管 Maintainer 的平均负载随时间有降低的趋势, 但因为工作量分布的不均衡性, 少数 Maintainer 仍然存在过劳现象. Linux kernel 的部分模块如硬件驱动程序只有不足 20% 的 Maintainer 负责审查 80% 的代码提交, 见图 3. 这种核心 Maintainer 负载过重的情况将直接影响开源生态的持续稳定发展^[5]. 该工作还发现为一个文件指派多个 Maintainer 只能提高 1/2 的生产率. 我们认为, 寻找降低开源生态管理复杂度的解决方案是未来工作的一个重点.

另一方面是开源生态系统中的商业参与. 此类研究的目的是探究商业参与模式及其对开源生态的影响并寻找可能的有效实践. 例如, 我们面向 3 个技术同构的混合项目 (JBossAS, JOnAS 和 Geronimo 均为实现 JavaEE 规范的应用服务器项目) 进行了实证分析^[25], 通过收集分析互联网上大量文本信息以及项目开发活动数据建立了 3 个商业参与模型, 并量化了不同模型对贡献者参与的影响. 该研究发现重度商业参与会减少新来者但同时提高贡献者的持续度. 我们的另一项研究 OpenStack (开源云

2) CIO Staff. <https://www.cio.com/article/2442437/open-source-tools/jboss-head-marc-fleury-leaves-red-hat.html> [2018-12-12].

计算平台) 的工作 [29] 也发现, 个别商业公司对开源项目的高强度控制会降低其他公司以及个体开发者的参与。我们还探索了商业公司广泛参与 OpenStack 的现状 [24], 研究表明, 来自不同行业的公司会有选择地对 OpenStack 生态中某些子项目做贡献来实现它们的特定目标, 例如 IBM 和 Intel 会向 OpenStack 贡献大量的驱动和插件来实现软硬件兼容。不同公司的贡献方式也有很大差别, 例如有的公司会将贡献重心放在文档编写上。总的来说, 现有工作尚未很好地解决众多商业组织和非营利机构的参与给开源生态带来的诸多挑战。相反很多研究案例折射出了目前开源生态系统在商业与开源混合时的困局。我们认为, 未来还需要从各个角度探索以深入并全面地刻画开源生态系统中的商业参与。

3 数据是保障

鉴于开发支持工具的广泛使用, 开源软件的历史发展过程几乎被完整地记录下来, 供我们去理解、反思和改进。这些数据覆盖了开源软件开发活动的各个方面, 为软件数字社会学的研究提供了宝贵的资源, 使研究者得以应用数据对问题进行广阔和深入的研究。数据体现了一种现实存在, 其揭示的本质规律通常具有可自证的性质(当然, 数据本身可能存在局限性或错误^[30])。与此同时, 越来越多的数据分析算法被引入软件工程研究中, 以帮助更好地理解和利用软件开发活动数据来解决问题, 例如, 我们团队提出了一个双模态自编码器来训练代码和自然语言描述对数据, 以提供更好的代码搜索和代码总结功能^[31]。然而, 尽管软件工程研究中所使用的方法日渐精进, 所积累的数据与日俱增, 所能回答的问题日渐复杂, 我们对软件开发活动数据的本质和能力还缺乏了解, 主要体现在下述两个方面。

首先, 软件开发数据有其自身固有特性, 这些特性包括: 多种类型、多种来源、不同类型和来源的数据存在内在关联; 不同类型的数据在不同的开发时间和开发环境下被不同的开发人员采用不同的工具提交, 导致数据存在大量的缺失、噪声和不确定性; 同一语义的数据在不同数据仓库中所呈现的不同描述方式造成数据语义上不一致等。这些都对传统的数据处理和分析技术提出挑战。并且, 面向大规模结构化或非结构化数据进行有效分析并获得能解释的结果, 一直是现今大数据时代的挑战。因此, 剖析软件数据的固有特征(例如, 分析软件数据在分布、类型和维度上与经典统计数据分布假设的差异), 研究特定于软件数据的数据采样和统计实验设计方法是最近的一个趋势。例如, Nagappan 等^[32] 研究了怎样的一个样本集合是具有代表性的; Rahman 等^[33] 研究了样本大小是否对缺陷预测(bug prediction)的效果有影响等。

其次, 目前数据质量并没有被很好理解, 使得由它推演的结论不可靠甚至无效。数据质量问题的本质是: 数据使用者在使用数据时未完全理解所使用数据在软件开发中的上下文, 进而在使用数据时做出了与其上下文不匹配的假设。此处的上下文包括但不限于: 软件开发数据所担任职责、软件开发数据的生成过程、开发者对相关工具的使用方式、不同项目的实践的差异、不同开发者的实践的差异等。例如一个项目在其演变过程中会更换版本控制系统(例如 Mozilla 就从 CVS 切换到了 Mercurial), 在更换过程中许多日志可能会遗失, 造成数据不完整; 人们在建立项目分支时会产生许多提交日志, 如果采用提交数来度量效率, 可能会有很大偏差。若软件开发数据的质量存在问题, 由这些数据所得到的结论的可信度将会受到影响。因此, 探讨数据本身存在的问题及其对研究和实践的影响也正在逐步形成一个重要方向。

例如, 通过分析 DBLP 库³⁾ 中使用缺陷报告数据的相关研究, 我们发现 58 篇论文中有 55 篇可能(或一定) 存在数据泄漏(data leakage, 即在建模时使用了来自未来的数据)^[34]。这是因为, 缺陷报告的(属性) 数据是随时间动态变化的(为了澄清缺陷的范围、定位缺陷所在等, 开发者会不断修改缺陷报

3) <https://dblp.uni-trier.de>.

告的属性), 但数据使用者对其应用场景和数据产生上下文缺乏认识, 可能会用到并不匹配其应用场景的来自未来的数据。研究结果表明数据泄漏可能导致研究结果无效。我们的另外一项工作^[30] 研究了人们在计算缺陷修复时间时所存在的偏差。以 Mozilla 社区的缺陷追踪和版本控制数据为例, 我们使用概率模型对个体每天的任务完成数量进行建模, 通过概率分布发现异常数据, 并利用数据的冗余性(一个事件可能存在于多个数据源中) 对问题数据进行修正。这一方法可被推广对更多的数据质量问题进行识别和修正。

关于数据质量的代表性研究工作还有很多。例如, Bird 等^[35] 研究了版本控制系统与缺陷追踪系统间的链接问题并探讨了其对研究结果的影响, 他们分析了多个项目的历史数据并发现这些数据中存在显著的系统偏差。Bachmann 等^[36] 则提出工具对这部分数据进行逆向恢复, 他们的工具使用户可以快速地发现及检查相关的缺陷报告及代码变更, 并根据需要对其进行标注。Herzig 等^[37] 及 Tantithamthavorn 等^[38] 则调查了缺陷追踪数据中的错误对代码缺陷预测的影响。他们发现缺陷报告中关于“defect”及“feature request”的分类并不可靠(缺陷追踪系统中的报告可分为缺陷报告和新功能开发等)。受数据噪声的影响, 代码文件常被错误地标记为存在缺陷并随后被错误地预测为存在缺陷。

总之, 在大数据条件下, 人工地理解每个项目的应用领域和最佳实践以获得对研究结果的解释将不太可能, 因此自动化方法是趋势; 同时, 鉴于软件数据存在大量的缺失、噪声和不确定性并具有自身的独特性质, 使得分析软件数据不能照搬经典的数据分析和挖掘方法。因此, 为利用软件数据指导软件开发和决策, 需要在充分理解所使用数据产生的软件开发上下文的基础上, 融合多场景、多层次、多类别的信息, 采用更精细的方法来确保或增进数据的质量。

4 开源供应链的挑战

分布在全球的开发者组成松散社区, 围绕各自需求开发各种开源软件, 形成开放生态。一个软件可能同时依赖数千个其他软件项目。图 4 展示了 CRAN (comprehensive R archive network) 运行时复杂庞大的依赖关系, 节点表示软件包 (package), 连线表示运行时依赖关系。不同于围绕同一开源软件形成的生态系统, 这种由于软件或软件项目之间互相依赖 (如软件的构建或运行时依赖, 开发者同时参与多个开源项目, 软件代码的复制粘贴等) 形成的复杂关系网络被称为开源供应链。图 5 展示了开源供应链的可能组成成分, 涉及了底层硬件固件、系统软件、开发平台软件以及与用户最为相关的应用软件, 大型的开源供应链几乎囊括了开发、运行过程中涉及到的所有开放软硬件及其生态。

开源软件从过去单纯由志愿者开发, 到不同组织机构参与下的生态系统, 再到数以万计互相依赖的软件或项目形成的开源供应链, 其转变给软件开发带来了前所未有的创新水平。同时, 规模指数级增长的开源软件或项目及其之间庞杂的依赖关系使得开源供应链的复杂度激增, 进而给开源软件开发带来诸多挑战。

第 1, 个体开发者学习成本进一步增大。首先因为广泛存在的依赖关系使得掌握一个新的软件更加复杂。例如, 对某个软件进行调错需要学习的相关软件依赖包可能会很多; 其次, 复杂依赖关系带来了新的问题, 涉及更多的学习内容。例如, 来自不同开源项目的代码片段需要遵循相应许可证 (license) 的约束, 并且不同许可证之间存在兼容问题^[39]。这就要求开发者在借鉴其他开源项目的代码时先了解对应的许可证之间的关系, 进而增大学习成本。此外, 一个开发者可能 (需要) 同时对多个开源项目做贡献, 熟悉不同开源项目的开发上下文、合理有效地分配时间精力以及挑选合适的开发任务等都同样会增加个体开发者的学习成本。

第 2, 群体协作更加复杂。开源供应链上的软件项目互相依赖, 开发者需要跨越多个项目去实现

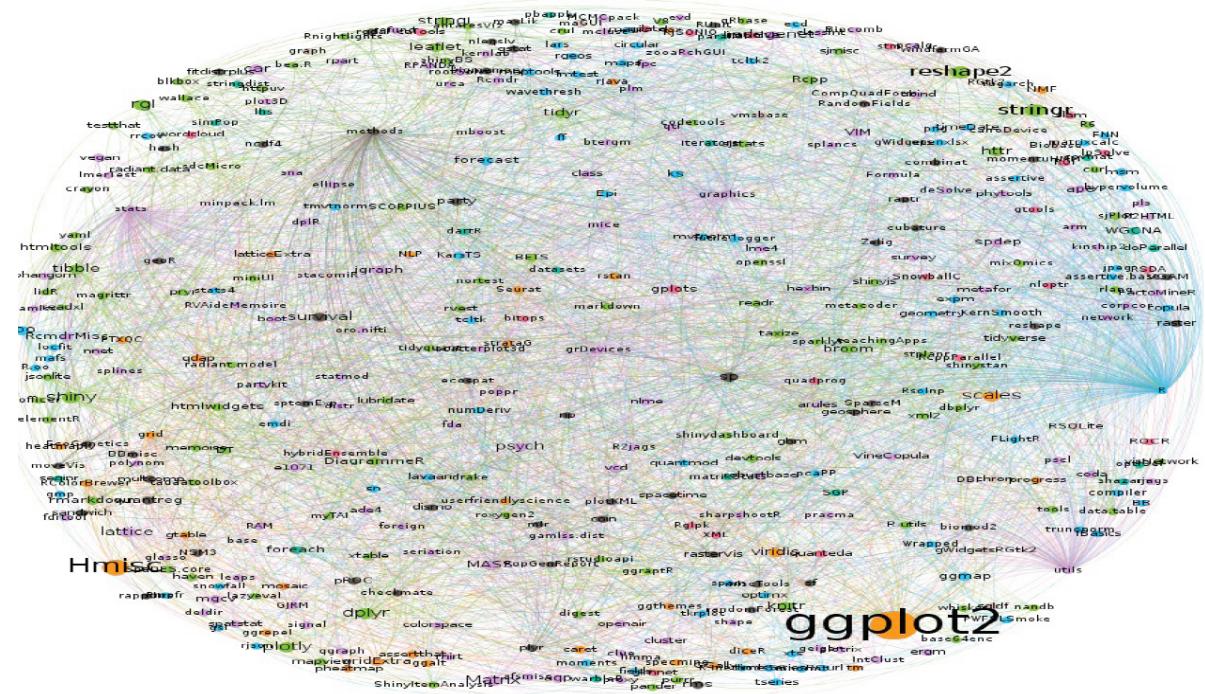


图 4 (网络版彩图) CRAN 运行时的依赖关系
 Figure 4 (Color online) CRAN runtime dependencies

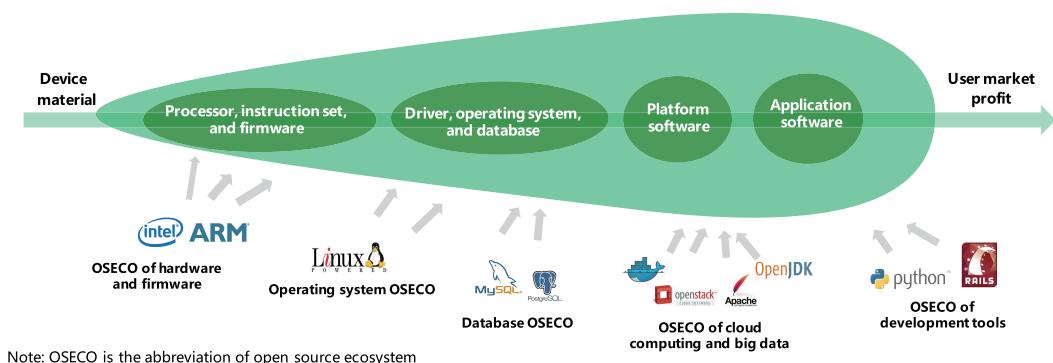


图 5 (网络版彩图) 开源软件供应链示意图
Figure 5 (Color online) Open source software supply chain diagram

目标功能。开发者之间的协作不再拘泥于单个项目。例如，为实现 Debian 和 OpenStack 的兼容，来自 Debian 社区的开发者向 OpenStack 提交了一千多行代码⁴⁾。已有的群体协作机制、平台支撑工具大多聚焦对单个项目的开发活动的支持。互相依赖的开源项目之间因缺少有效的信息沟通与集成机制使得群体协调的复杂度增大。

第3,生态可持续性受到的威胁持续增加.开源供应链上的项目节点之间是互相影响的.一个开源软件的漏洞有可能使得其他依赖(例如调用或复用)该软件的项目面临同样的危机.例如,影响恶劣

4) Stackalytics. <http://stackalytics.com/?release=all&metric=marks&company=debian>. [2018-12-12].

的 Heartbleed 漏洞⁵⁾所涉及的 OpenSSL 项目中的两个文件的某些版本, 至少存在于其他六千多个开源软件项目中^[40]. 然而开源供应链上项目节点间的依赖关系隐藏在开发活动数据中, 看不见摸不着但广泛存在, 这就使得节点软件的生态可持续受到更多潜在威胁.

总之, 尽管有数千万个开源软件和项目及超过一千亿的源代码文件, 但人们对开源供应链的形成和发展, 及其可能带来的风险挑战却知之甚少. 随着开源生态的内涵和外延的快速拓展, 各类供应链关系逐步显现, 如开发供应、技术供应以及产销供应等. 与传统的供应链不同, 开源供应链中的生产活动是公开的. 开源供应链中数以千万计的个体开发者、软件项目以及围绕软件产生的复杂生态的过程数据可以被方便地获取, 这为我们更好地理解和降低或消除上述提到的风险、识别其他可能存在的风险提供了可能. 利用社会学理论对海量数据可视化出来的开源供应链网络进行分析, 可以允许我们从个体学习、群体协作以及生态持续的角度去识别评估风险, 进而更好地保障开源生态的可持续发展.

5 结束语

现如今全球化的软件开发面临复杂性不断扩张的问题: 软件规模急剧增长、代码依赖错综复杂; 开发者个体差异大、群体协作难以控制; 广泛的社会参与形成复杂生态等. 这些问题使得软件开发呈现出很强的社会学特征, 个体学习、群体协作和生态持续成为管理和控制大规模复杂软件及开发的关键挑战.

软件数字社会学的目标是理解软件开发面临的关键挑战并寻找可能的解决方案. 一方面, 我们可以借鉴社会科学(涉及认知科学、发展心理学、组织管理学、经济学等)在人类个体学习、群体协作和生态演化等方面的研究方法和洞察来探索复杂开发活动; 另一方面, 鉴于开发支持工具的广泛使用, 开源软件项目在互联网上积累了海量的数据. 这些数据覆盖了开源软件开发活动的各个方面, 几乎完整地记录了开源软件的历史发展过程, 为软件数字社会学的研究提供了丰富的资源, 使得研究者拥有前所未有的机会对上述问题进行定量研究.

具体来说, 软件数字社会学的研究对象从微观的开发者个体、群体协作到宏观的生态演化, 涉及了复杂软件开发的多个方面. 研究问题包括但不限于, 个体开发者如何快速掌握复杂软件系统、如何成长为核开发者, 开发者之间如何协同完成分布式任务、其合作行为如何发展, 以及涉及商业参与和个体志愿者参与的多种参与模式作用下的软件生态系统如何构建、发展和持续演化等. 特别值得一提的是, 随着开源软件的快速演进, 开源世界的代码、开发者、项目、用户之间形成了广泛的依赖和供应链关系, 一方面为软件复用和软件生态形成带来了前所未有的便利, 另一方面也存在广泛的供应链风险. 如何能够应对复杂网络构建开源供应链, 以识别和消除(或缓解)软件风险, 也是软件数字社会学的一个重要挑战.

软件数字社会学希望企及的目标不仅体现在软件开发效率和质量的提升上, 还体现了更高抽象层次的社会理念. 例如, 开源软件的开放协同共享、生态健康演化等理念, 已经扩展到了软件行业之外(如开源硬件运动), 促进了一种新的经济现象(即协同共享经济)的萌芽与持续发展. 某种意义上, 开源理念正在对整个人类文明的发展产生深远的影响. 对开源软件个体、群体以及生态这3个方面的开发模式和机理的量化分析的深刻意义是, 通过对历史的量化理解, 指导我们如何通过大规模的社会化协作去建设一个更加美好的信息化人类文明.

5) <http://heartbleed.com/>.

参考文献

- 1 Witt J. SOC 2018. 6. New York: McGraw-Hill Higher Education, 2018. 121–135
- 2 Lazer D, Pentland A S, Adamic L, et al. Life in the network: the coming age of computational social science. *Science*, 2009, 323: 721–723
- 3 Brooks F. *The Mythical Man-Month: Essays on Software Engineering*. Anniversary Edition, 2nd Edition. New York: Pearson Education India, 1995. 25–35
- 4 Mockus A, Fielding R T, Herbsleb J. A case study of open source software development: the Apache server. In: *Proceedings of the 22nd International Conference on Software Engineering*. New York: ACM, 2000. 263–272
- 5 Zhou M, Chen Q, Mockus A, et al. On the scalability of Linux kernel maintainers' work. In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. New York: ACM, 2017. 27–37
- 6 Astromskis S, Bavota G, Janes A, et al. Patterns of developers behaviour: a 1000-hour industrial study. *J Syst Softw*, 2017, 132: 85–97
- 7 Manikas K, Hansen K M. Software ecosystems – a systematic literature review. *J Syst Softw*, 2013, 86: 1294–1306
- 8 Vygotsky L. Interaction between learning and development. *Readings Develop Child*, 1978, 23: 34–41
- 9 Schach S R. *Object-Oriented and Classical Software Engineering*. New York: McGraw-Hill, 2007. 125–135
- 10 Zhou M, Mockus A. What make long term contributors: willingness and opportunity in OSS community. In: *Proceedings of International Conference on Software Engineering*. Piscataway: IEEE Press, 2012. 518–528
- 11 Tan X, Qin H, Zhou M. Understanding the variation of software development tasks: a qualitative study. In: *Proceedings of the 9th Asia-Pacific Symposium on Internetworks*. New York: ACM, 2017
- 12 National Research Council. *How People Learn: Brain, Mind, Experience, and School*. Expanded Edition. Washington: National Academies Press, 2000. 134–145
- 13 Curtis B. Fifteen years of psychology in software engineering: individual differences and cognitive science. In: *Proceedings of the 7th International Conference on Software Engineering*. Piscataway: IEEE Press, 1984. 97–106
- 14 Neisser U, Boodoo G, Jr Bouchard T J, et al. Intelligence: knowns and unknowns. *Am Psychol*, 1996, 51: 77–101
- 15 Ericsson K A, Krampe R T, Tesch-Römer C. The role of deliberate practice in the acquisition of expert performance. *Psychol Rev*, 1993, 100: 363–406
- 16 Zhou M, Mockus A. Who will stay in the FLOSS community? Modeling participant's initial behavior. *IEEE Trans Softw Eng*, 2015, 41: 82–99
- 17 Herbsleb J D, Moitra D. Global software development. *IEEE Softw*, 2001, 18: 16–20
- 18 Zhou M, Mockus A. Mining micro-practices from operational data. In: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. New York: ACM, 2014: 845–848
- 19 Zhu J, Zhou M, Mockus A. Patterns of folder use and project popularity: a case study of github repositories. In: *Proceedings of ACM/IEEE International Symposium on Empirical Software Engineering & Measurement*. New York: ACM, 2014. 1–4
- 20 Ohira M, Ohoka T, Kakimoto T, et al. Supporting knowledge collaboration using social networks in a large-scale online community of software development projects. In: *Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC'05)*, Taipei, 2005. 6
- 21 Xie J, Zheng Q, Zhou M, et al. Product assignment recommender. In: *Proceedings of the 36th International Conference on Software Engineering*. New York: ACM, 2014. 556–559
- 22 Tan X, Lin Z Y, Zhang Y X, et al. Analysis of contribution composition patterns of code files. *J Softw*, 2018, 29: 2283–2293 [谭鑫, 林泽燕, 张宇霞, 等. 代码文件贡献组成模式的分析. 软件学报, 2018, 29: 2283–2293]
- 23 Coelho J, Valente M T. Why modern open source projects fail. In: *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. New York: ACM, 2017. 186–196
- 24 Zhang Y X, Zhou M H, Zhang W, et al. How commercial organizations participate in openstack open source projects. *J Softw*, 2017, 28: 1343–1356 [张宇霞, 周明辉, 张伟, 等. OpenStack 开源社区中商业组织的参与模式. 软件学报, 2017, 28: 1343–1356]
- 25 Zhou M, Mockus A, Ma X, et al. Inflow and retention in OSS communities with commercial involvement: a case study of three hybrid projects. *ACM Trans Softw Eng Methodol*, 2016, 25: 1–29
- 26 Chris Mellor. Aperi dies on its arise. *The Register*. https://www.theregister.co.uk/2009/01/30/aperi_is_dead/ [2009-09-01]

- 27 Gamalielsson J, Lundell B, Lings B. Responsiveness as a measure for assessing the health of OSS ecosystems. In: Proceedings of the 2nd International Workshop on Building Sustainable Open Source Communities. Tampere: Tampere University of Technology, 2010. 1–8
- 28 Lehman M M, Ramil J F, Wernick P D, et al. Metrics and laws of software evolution—the nineties view. In: Proceedings of the 4th International Software Metrics Symposium, Albuquerque, 1997. 20–32
- 29 Zhang Y, Tan X, Zhou M, et al. Companies' domination in FLOSS development: an empirical study of OpenStack. In: Proceedings of the 40th International Conference on Software Engineering. New York: ACM, 2018. 440–441
- 30 Zheng Q, Mockus A, Zhou M. A method to identify and correct problematic software activity data: exploiting capacity constraints and data redundancies. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. New York: ACM, 2015. 637–648
- 31 Chen Q, Zhou M. A neural framework for retrieval and summarization of source code. In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. New York: ACM, 2018. 826–831
- 32 Nagappan M, Zimmermann T, Bird C. Diversity in software engineering research. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. New York: ACM, 2013. 466–476
- 33 Rahman F, Posnett D, Herraiz I, et al. Sample size vs. bias in defect prediction. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. New York: ACM, 2013. 147–157
- 34 Tu F, Zhu J, Zheng Q, et al. Be careful of when: an empirical study on time-related misuse of issue tracking data. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York: ACM, 2018. 307–318
- 35 Bird C, Bachmann A, Aune E, et al. Fair and balanced? Bias in bug-fix datasets. In: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. New York: ACM, 2009. 121–130
- 36 Bachmann A, Bird C, Rahman F, et al. The missing links: bugs and bug-fix commits. In: Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM, 2010. 97–106
- 37 Herzig K, Just S, Zeller A. It's not a bug, it's a feature: how misclassification impacts bug prediction. In: Proceedings of the 2013 International Conference on Software Engineering. Piscataway: IEEE Press, 2013. 392–401
- 38 Tantithamthavorn C, McIntosh S, Hassan A E, et al. The impact of mislabelling on the performance and interpretation of defect prediction models. In: Proceedings of 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering. Piscataway: IEEE Press, 2015. 1: 812–823
- 39 Laurent A M S. Understanding Open Source and Free Software Licensing: Guide to Navigating Licensing Issues in Existing & New Software. California: O'Reilly Media, 2004
- 40 Dey T, Mockus A. Are software dependency supply chain metrics useful in predicting change of popularity of npm packages? In: Proceedings of the 14th International Conference on Predictive Models and Data Analytics in Software Engineering. New York: ACM, 2018. 66–69

Software digital sociology

Minghui ZHOU^{1,2*}, Yuxia ZHANG^{1,2} & Xin TAN^{1,2}

1. Electronics Engineering and Computer Science, Peking University, Beijing 100871, China;
2. Key Lab of High Confidence Software Technologies (Peking University), Ministry of Education, Beijing 100871, China

* Corresponding author. E-mail: zhmh@pku.edu.cn

Abstract With the continuous development of the internet, software development (especially global open-source development) is facing critical challenges, such as individual differences in developers distributed worldwide, the increasing difficulty of group collaboration, and the complex ecosystems formed by extensive social participation. These problems exhibit strong sociological characteristics in regard to the activity of software development. This study proposes the concept of software digital sociology that involves mining abundant software-activity data to investigate individual learning, group collaboration, and sustainable ecosystems. We discuss the primary investigation of the critical issues associated with the open-source supply chain and suggest that software digital sociology can inspire software-development researchers to understand the key challenges posed by extensive (computer-mediated) social participation to promote exploration of solutions using a different paradigm.

Keywords software digital sociology, software-activity data, individual learning, group collaboration, open-source ecosystem, software supply chain, data quality



Minghui ZHOU received her B.S., M.S., and Ph.D. degrees in computer science from the National University of Defense Technology in 1995, 1999, and 2002, respectively. She is a professor at the Software Institute, School of Electronics Engineering and Computer Science, Peking University. She is interested in digital sociology (i.e., understanding the relationships among people, project cultures, and software products) through mining the repositories of software projects.



Yuxia ZHANG is a Ph.D. candidate at the School of Electronics Engineering and Computer Science, Peking University. She received her B.S. degree in software engineering from Northwest University in 2015. Her research interests include mining software repositories and open-source software ecosystems, mainly focusing on the commercial participation in open-source software development.



Xin TAN is a Ph.D. candidate at the School of Electronics Engineering and Computer Science, Peking University. She received her B.S. degree in software engineering from Northwest Polytechnical University in 2016. Her research interests include mining software repositories and open-source software ecosystems, mainly focusing on the micro-process of open-source software development.