计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.190300140

面向群体协作开发的开源软件峭壁分析

何 鹏[1,2] 喻绿君[1]
1 湖北大学计算机与信息工程学院 武汉 430062
2 应用数学湖北省重点实验室 武汉 430062

摘 要 开源软件项目因门槛低、自由度高,在开发过程中存在进度缓慢、效率低下和项目质量偏低等问题;同时,软件峭壁(Software Cliff Wall)作为一种判定项目鲁棒性的依据,表现为软件开发过程中在短时间内完成远超过常规增量开发的一种代码贡献行为,是软件演化过程中可持续发展的一种潜在威胁。为了深入研究开源项目的开发过程,更准确地刻画软件演化,从而提高软件开发效率,分析软件峭壁的成因是一种行之有效的方法。实验以GitHub上9个时间跨度至少有5年的开源软件项目为研究对象,分别以月份和季度为周期,基于150000多个commits上开发者的关注与评论信息构建开发者合作网络(Developer Collaboration Networks,DCN),将代码行数超过1万行的单次commit视为软件峭壁,并从网络规模、网络结构、网络质量3个方面,利用节点数、连边数、节点更新率、模块度、平均路径长度、平均度、节点入度指数、节点出度均值、多样性这9个度量指标来分析软件开发过程中DCN与软件峭壁的关系。研究结果表明:1)当开发团队规模偏小,且成员更新幅度较大时,容易形成软件峭壁;2)保持开发者之间良好的"小世界"特性,有助于避免峭壁的产生;3)以季度为周期来分析软件开发过程中DCN与软件峭壁的关系更为合适,且开发团队成员的组织来源多样化也会在一定程度上促进软件峭壁的产生。

关键词:群体协作开发;开发者合作网络;软件峭壁;软件演化;开源软件
中图法分类号 TP301

Analysis of Open Source Software Cliff Walls for Group Collaborative Development

HE Peng[1,2] and YU Lv-jun[1]
1 School of Computer Science and Information Engineering,Hubei University,Wuhan 430062,China
2 Hubei Key Laboratory of Applied Mathematics,Wuhan 430062,China

Abstract Due to the characteristics of low threshold and high freedom,open source software encounters slow progress,low efficiency and low quality in the development process. Software cliff wall as a criterion of project robustness,indicates unexpected acceleration in common incremental development activities over short periods of time,which is a potential threat to the sustainable development in software evolution. Therefore,analyzing the causes of software cliff walls is an effective method to deeply understand the development process of open source projects,to more accurately describe the evolution of software,and to improve the efficiency of software development. The experiment firstly constructes a series of developer collaboration networks (DCNs) over more than 150 thousand commits from 9 GitHub projects by month and quarter respectively. This paper consideres a single commit of more than 10000 lines of code as a software cliffs. And then it introduces 9 metrics,such as the number of nodes,the number of connected edges,the node update rate,the module degree,the average path length,the average degree,the node penetration index,the node out-of-mean,and the diversity,to analyze the relationship between DCN and cliff walls from the perspectives of network scale,network structure and network quality. The results show that:1)smaller development teams and greater member turnover tend to cause a cliff wall;2)'small world' features among developers is helpful to avoid the emergence of software cliff walls;3)the relationship between DCN and software cliffs in the software development process is more appropriate in a quarterly cycle,and the diversity of the development team will also affect the creation of cliff walls in software development.

到稿日期:2019-03-27 返修日期:2019-07-20 本文已加入开放科学计划(OSID),请扫描上方二维码获取补充信息。
基金项目:国家重点研发计划(2018YFB1003801);国家自然科学基金项目(61902114);湖北省教育厅青年人才项目(Q20171008);应用数学湖北省重点实验室开放基金(HBAM201901)
This work was supported by the National Key R & D Program of China (2018YFB1003801),National Natural Science Foundation of China (61902114),Hubei Province Education Department Youth Talent Project(Q20171008) and Hubei Provincial Key Laboratory of Applied Mathematics(HBAM201901).
通信作者:何鹏(penghe@hubu.edu.cn)

Keywords Group collective development, Developer collaboration network, Software cliff walls, Software evolution, Open source software

1 引言

软件开源是一种有效的软件复用模式[1]。近年来，开源软件迅速发展，为软件项目的可用性提供了便利，有力地促进了软件行业以及相关技术的进步[2]。开源软件项目作为一类典型的虚拟群体开发社区，来自不同国家、有着不同背景的开发者通过加入社区与其他开发者互惠合作[3]，知名的开源社区有 GitHub 和 SourceForge 等。与传统软件项目不同[4-5]，开源软件项目的开发存在着门槛低、自由参与的特征；但同时伴随着开发效率低下、项目支持率和参与度低以及开发无法持续等问题，软件质量难以保证。随着多年的开发运行，软件项目在逐渐丰富与完善以适应社会发展的过程中，吸纳了相关领域的众多知识，软件的规模变得越来越大，单个软件项目的结构也变得更为复杂。开源软件的源代码、邮件列表、缺陷报告等软件资源蕴含了规模庞大、结构复杂、语义关联的丰富的软件知识[6]，充分利用这些资源，选择高质量的软件项目，降低软件项目的开发成本，变得十分重要。

在开源社区中研究软件项目的质量有多种方式，如问题跟踪[7]、环路复杂度[7]、结构优化[8]、缺陷预测[9]、信誉程度[10]等。软件峭壁是在软件项目开发过程中在短时间内完成远超过常规增量开发的一种代码贡献行为，如开发者在一天内提交的代码规模超过十万行的 commit 行为[11-12]。软件峭壁作为判断软件项目鲁棒性的一个依据[13]，是对软件项目可持续发展的一种威胁。本文以 GitHub 社区中软件项目开发者的 commit 提交行为的相关信息，来分析软件开发过程中的峭壁现象，以发掘开发者交互行为对软件峭壁的影响。

本文第 2 节总结了目前该领域的国内外的研究现状；第 3 节详细介绍了本文研究的理论基础和实验方法；第 4 节介绍了实验数据的来源和实验步骤，并对实验结果进行了分析和讨论；最后总结本文，并展望未来。

2 相关工作

与软件项目相关的研究比较多。Yang 等[2]通过分析 GitHub 开源软件项目的开发过程，提出了问题(issue)解决速度、问题增加速度等影响因素，对这些影响因素间的相关性进行了分析，并给出了一些促进软件项目持续发展的建议。He 等[3]对 Sourceforge 平台上软件项目开发者之间的重复合作行为进行了分析。Cheng 等[14]从开源软件生态系统的角度分析了在 GNAME 系统上影响开发者角色演化的因素，发现开发者的主观意愿和软件项目环境的部分指标(如开发者参与的软件项目总数)显著影响了开发人员发展为核心开发人员和软件项目负责人的机会，并对开发者角色演变进行了预测。

峭壁一般被用来形容地理中非常险峻的山势。Maclean 等[11-12]最早将峭壁一词引入软件演化过程中，以年份为横坐标、软件项目规模为纵坐标，生成随时间变化的项目增长折线图，将一次性提交大量的代码使项目增长趋势呈垂直式(或接近垂直)时的现象定义为峭壁，并分析了峭壁出现的可能的原因。图 1 给出了项目 Firebird 的代码提交量随时间变化的分布情况[11]。随后，Pratt 等[15]在此基础上以 Sourceforge 平台为基础，使用 LDA(Latent Dirichlet Allocation)模型进一步分析了峭壁出现的原因。他们指出峭壁是对软件项目在演化过程中可持续发展的一种威胁，因为它们并不代表增量开发[13]。

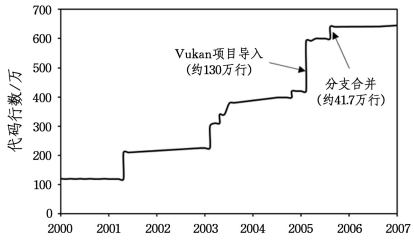


图 1 项目 Firebird 的代码提交量分布[11]

Fig. 1 Project Firebird's code commits distribution[11]

Chen 等[16]指出开发者之前的合作次数越多越有利于他们再次建立合作，并通过分析 commit 提交者与创作者的关系来进一步指出软件项目 commit 90%以上的提交都以核心成员为主。Grönlund 等[17]试图用软件项目的受欢迎程度来解释 commit 提交行为的频率。Sinha 等[18]从基于知识和技能的信任发展、基于相关项目工作经验的信任发展、基于用人组织的信任这 3 方面对开发人员作为 commit 提交者的可信度做出分析。Ma 等[19]通过分析软件开发人员的集体和个人 commit 提交行为在提交间隔的分布，指出项目级 commit 提交间隔的数据集大致遵循幂律分布，即主动提交者的行为模式非常相似，大多数人喜欢从晚上到清晨定期执行他们的项目，在相邻提交的 commit 之间的等待时间长可能是由于个人原因，如生病、休假和意外事件。

已有文献都只是单独分析开源社区中开发者间的关系、commit 提交行为和因代码量产生的软件峭壁对软件项目的影响，并未综合考虑软件峭壁对软件项目的影响。因此，本文从开发者之间的合作关系出发，重点考虑因 commit 提交行为表现出的软件项目峭壁与开发人员之间的关系，主要贡献归纳为：

(1)相对已有研究，本文做出了进一步的探索，对不同时间段参与软件项目 commit 提交的开发者行为进行网络建模，从网络规模、网络结构和网络质量 3 个角度分析它们对软件峭壁产生的影响。

(2)研究发现，保持开发团队的稳定性和“小世界”性并降低开发团队成员来源的多样性，可有效抑制峭壁的产生。

3 理论基础

3.1 网络模型

近年来，GitHub 已经成为主流的项目托管、镜像和协作的开源平台，提供了一个广泛的 REST API，使研究人员能够

获取高质量的、相互关联的数据[20]。据统计，截至 2018 年 10 月，从框架到数据可视化，该平台共构建了约 9 600 万个存储库，拥有 3100 万用户，且数据仍在持续增加[21]。GitHub 开源社区提供的在开发者之间建立交互的渠道较多，如写邮件、相互关注、互相评论等。

定义 1（开发者合作网络）　根据开发者之间的交互关系，可构建一个有向开发者网络，即开发者合作网络，表示为 $DCN=(V_f,E_f)$，其中 V_f 为开发者集合，E_f 为开发者之间的合作关系。如果开发者 v_i 关注（follow）了开发者 v_j，或者开发者 v_i 评论了开发者 v_j 提交的 commit，则认为它们之间存在一条有向合作连边，在 DCN 网络中表示为 $e_{ij}=1$，否则 $e_{ij}=0$。

如图 2 所示，如果两个开发者之间的关系类似 v_1 和 v_4，即同一个方向既具有 follow 关系又有 comment 关系，则在 DCN 中将不重复计算，亦即本文的 DCN 为非加权网络。如果两个开发者之间的关系类似 v_5 和 v_6，即既具有 follow 关系又有 comment 关系，但方向不同时，DCN 中将存在一条双向连边。

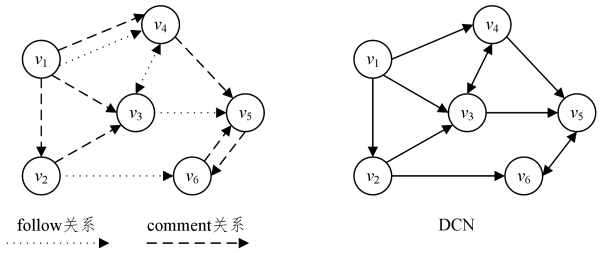


图 2　开发者合作网络

Fig. 2　Developer collaboration network

3.2　软件峭壁

软件峭壁多以 commit 大小来衡量，其中一些研究者用 commit 修改的文件数来定义 commit 大小[22]，还有一些研究者根据 commit 中增加、删除和修改的代码行来定义 commit 大小[15,23]。对于以上两种处理，简单地统计修改文件的个数难以真实体现一个 commit 的工作量；而对于一个已有代码文件，如果只是做小幅度的修改，且没有实际代码行的增加和删除，则对代码规模并没有实际影响。因此，本文把 commit 大小定义为在一个给定的 commit 中所有源代码文件中增加和删除的代码行数，并增加在对应 commit 的评论中增加和删除的代码行数。

最后，类似于文献[23]，不排除存在个别开发能力极强的开发者可能在几天或数周内完成上万行的代码工作，但我们仍采取将大小超过 1 万代码行的单个 commit 视为峭壁。

3.3　度量指标

对于一个软件项目，尤其是开源软件，在不同的时间段其开发者可能不同，而一个开发者也会参与软件项目的多个时间段的开发。因此，导致软件峭壁的因素比较复杂。针对构建的开发者合作网络，参考文献[24-27]，我们从网络规模、网络质量、网络结构 3 个方面，引入 9 个度量指标来分析它们对软件峭壁产生的影响，如表 1 所列。

表 1　度量指标信息

Table 1　Metric information

序号	度量指标	分析角度
M1	节点数$\|V\|$	网络规模
M2	节点更新率 δ	
M3	连边数$\|E\|$	
M4	模块度 Q	网络结构
M5	平均路径长度 $AvgPath$	
M6	平均度 k	
M7	节点入度指数 α	网络质量
M8	节点出度均值 u	
M9	多样性 θ	

（1）节点数（$|V|$）与连边数（$|E|$）：通过分析峭壁周期与非峭壁周期对应的 DCN 中节点数$|V|$与连边数$|E|$的差异，探析开发人员数量以及人员之间的交互关系对软件峭壁产生的影响。

（2）节点更新率（δ）：其为相邻两个周期的 DCN 中所有节点交集与并集的比值。节点更新率 δ 的计算式如式（1）所示：

$$\delta_t=\frac{|V_t|\bigcap|V_{t+1}|}{|V_t|\bigcup|V_{t+1}|} \tag{1}$$

其中，V_t 表示在第 t 个时间周期参与软件项目开发的开发者集合。

（3）模块度（Q）：用来度量一个网络内部节点模块化（也称社区化）的程度[28]。网络的模块度越大，其节点连边越密集。模块度的计算式如式（2）所示：

$$Q=\frac{\sum_i^n e_{ii}-\sum_i^n a_i b_i}{1-\sum_i^n a_i\,b_i} \tag{2}$$

其中，e_{ii} 表示模块 i 中节点之间的实际连边数与最大连边数的比例；$a_i=\sum_j^n e_{ij}$，$b_i=\sum_j^n e_{ji}$ 分别为起始节点、终止节点在模块 i 中的连边比例；n 为模块被划分的总数。

（4）平均路径长度（$AvgPath$）：其为网络中所有节点对之间最短路径的平均距离。若节点 v_i 与 v_k 和 v_j 之间存在有向连边 e_{ik}，e_{kj}，e_{ij}，则 v_i 与 v_j 这两个节点之间的最短距离 $d_{ij}=1$。平均最短路径的计算式如式（3）所示：

$$AvgPath=\frac{1}{n}\sum_{i=1}^{n}\frac{\sum_{v_j\in U_i}d_{ij}}{|U_i|} \tag{3}$$

其中，节点 v_i 可连接的节点集合为 U_i。

（5）平均度（k）：可用来表示网络结构的整体复杂性，表现为网络内部节点之间的相互依赖趋势。计算平均度时不区分出度与入度，其计算式如式（4）所示：

$$k=\frac{\sum_{i=1}^{n}degree_i}{count_v} \tag{4}$$

其中，$degree_v$ 表示节点 v 的度，即节点 v 的所有入度与出度之和；$count_v$ 表示节点总数。

（6）入度指数（α）与出度指数（μ）：在 DCN 网络中，一个节点的入度越大，表示该开发者受关注的程度越高或提交的 commit 引起了更多其他成员的评论；出度越大，表示该开发者受关注的程度越低或评论行为越频繁。考查节点的度分布是分析复杂网络的常用方法，度分布 P_k 为开发者网络中节点度为 k 的占比。研究表明，开发者网络的节点入度表现为幂律分布，出度则呈对数正态分布[24]。因此，入度指数的计

算方式如式(5)所示：

$$f_{in}(k)=c(k+k_0)^{-\alpha},c>0,k_0=1 \tag{5}$$

其中，c 和 k_0 为常数，入度分布特征采用幂指数 α 进行描述。

出度指数的计算方式如式(6)所示：

$$f_{out}(k)=\frac{1}{\sqrt{2\pi\sigma}(k+k_0)}e^{-\frac{(\ln(k+k_0)-\mu)^2}{2\sigma^2}},k_0=1 \tag{6}$$

其中，对数均值 μ 用于分析出度的分布特征。

(7)组织多样性(θ)：根据 Github 提供的数据，每个开发者有具体所属的组织(organization)。理论上，一个软件项目的开发成员来源的组织越多，成员之间交互的背景就越复杂。开发成员可能来自多个不同的组织，一个开发成员也可能加入了多个不同的组织。因此，我们用多样性来表征每个时间段软件项目的开发团队成员的组织来源情况。为了衡量软件项目成员的多样性，我们引入信息熵的概念。在某个时间段，令 p_i 为该时间段中属于第 i 个组织的开发者比例，则成员组织多样性 θ 的计算式如式(7)所示：

$$\theta=-\sum_{i=1}^{n}p_i\ln p_i \tag{7}$$

其中，n 为社区存在的组织数。

4 实验分析

4.1 实验数据

本文获取的数据来源于 GHTorent API 公开的 GitHub 上的开源软件开发数据。为分析软件项目的峭壁现象，我们选取了 9 个 commit 提交次数大于 5000 且时间跨度至少有 5 年的软件项目，相关统计信息如表 2 所列。我们利用软件项目信息、项目 commit 提交信息、commit 详情信息、commit 评论信息、开发者关注信息、开发者所属组织信息、评论信息等，提取每个软件项目在 5 年间的 commit 信息、对应的评论，以及开发者相互关注和所属团队的信息。

表 2 开源软件项目的信息统计(2009 年 1 月－2013 年 12 月)

Table 2 Open source software project information statistics

序号	项目编号	软件名称	地址	提交总次数	参与提交总人数
1	78852	Rails	https://api.github.com/repos/rails/rails	36802	1728
2	3583	Xbmc	https://api.github.com/repos/xbmc/xbmc	35654	441
3	9	Mongo	https://api.github.com/repos/mongodb/mongo	22782	176
4	69158	Django	https://api.github.com/repos/django/django	16603	431
5	107534	Scala	https://api.github.com/repos/scala/scala	16240	158
6	63250	Cakephp	https://api.github.com/repos/cakephp/cakephp	12526	299
7	59607	Codeigniter	https://api.github.com/repos/EllisLab/CodeIgniter	7104	429
8	25875	Jquery	https://api.github.com/repos/jquery/jquery	6251	271
9	76945	Boto	https://api.github.com/repos/boto/boto	5203	386

在构建 DCN 网络模型时，如果时间周期太短，DCN 的网络规模可能过小，不利于分析；如果时间周期太长，则容易导致峭壁现象，不易于区分。鉴于此，我们选取月份和季度两个时间窗口作为划分开发者数据的周期。另外，考虑到此处划分的周期并不代表实际软件项目开发版本的周期，为减少建模过程中的偏见，我们每次以 1 个月为跨度对时间窗口进行平滑处理。

4.2 实验步骤

本节重点介绍研究方法的框架，如图 3 所示。

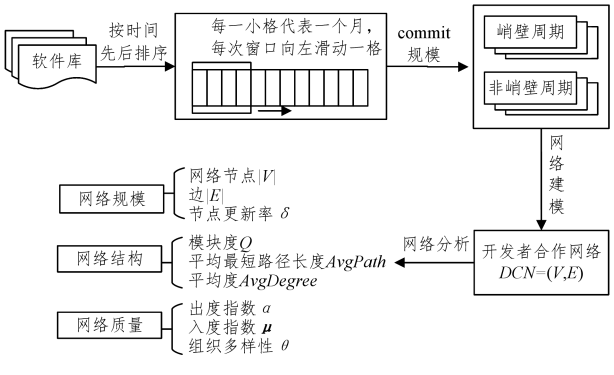


图 3 整体研究框架

Fig. 3 Overall research framework

该研究主要分为 4 个步骤：1)数据预处理，根据 commit 提交、开发者 follow 和评论记录的时间进行排序，并先后以月份和季度为时间窗口进行滑动，将相关信息进行周期划分；2)峭壁周期的确定，按照 3.2 节中的定义，将对应周期划分为峭壁和非峭壁两类；3)开发者网络建模，依据步骤 1)中的周期划分，按照 3.1 节中的方式为每个周期构建对应的 DCN 网络模型；4)网络分析，结合式(1)－式(7)计算系列度量指标，并分析它们在峭壁周期与非峭壁周期的差异。

为便于表示，我们把发生峭壁的周期称为峭壁周期，未发生峭壁的周期称为非峭壁周期。采用“cw_month”和“month”分别表示按月份划分时的峭壁周期和非峭壁周期，用“cw_quarter”和“quarter”分别表示按季度划分时的峭壁周期和非峭壁周期。

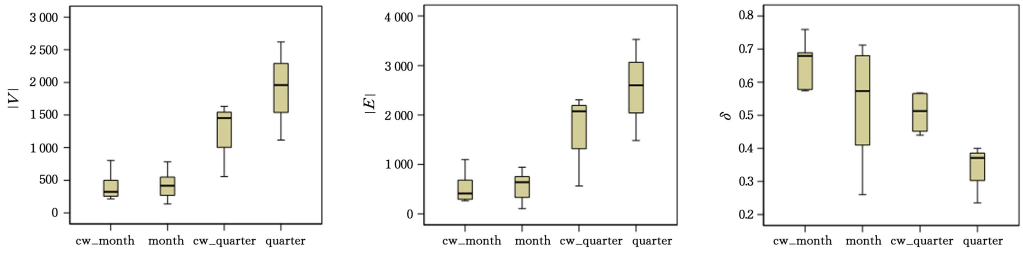
4.3 实验结果分析

4.3.1 网络规模

图 4(a)和图 4(b)分别给出了以月份和季度为时间周期窗口时，9 个软件项目的 DCN 网络中节点数、连边数与峭壁的关系。实验结果显示，无论以月份还是季度为时间窗口，峭壁周期的 DCN 网络节点数和连边数都略少于非峭壁周期的，表现为箱图中的中位数更小。这说明在发生峭壁的时间周期内，开发者数量倾向于更少，且开发者之间有效的直接交互也更少。一个软件项目的团队成员之间的交互关系越少，信息交流越不通畅，解决问题的效率会受到影响。

图 4(c)是分别以月份和季度为时间窗口时，在峭壁周期与非峭壁周期内 DCN 网络的节点更新率分布。不难发现，不论按月份还是季度划分，相比非峭壁周期，峭壁周期的 DCN 节点的更新率整体都更大，表现为：“cw-month”与“month”的 δ 中位数分别为 0.679 和 0.573，“cw-quarter”与“quarter”的 δ 中位数分别为 0.513 和 0.371。以上数据说明，相比非峭壁

周期，峭壁周期的开发者流动性更大，团队成员的结构更不稳定，新成员因缺乏对项目的了解而不当地做了大修改或无用功，或者人员的较大流动，导致遇到临时开发任务时开发者被迫采取代码自动生成或已有部分代码的复制重用和导入，致使出现一次规模过大的 commit 提交。为统计峭壁周期与非峭壁周期内指标值的差异，我们进一步采用曼-惠特尼 U 检验对其进行显著性分析。如表 3 所列，“cw-month”与“month”的 3 个网络规模指标的 *p-value* 值均大于 0.05，说明虽然峭壁周期与非峭壁周期的指标值中位数有所不同，但从统计显著性角度看，两者并没有显著的差别。然而，“cw-quarter”与“quarter”下的检验结果 *p-value* 值均小于 0.05，尤其是节点更新率。



(a)节点数$|V|$　(b)连边数$|E|$　(c)节点更新率δ

图 4　网络规模与软件峭壁关系的分析

Fig. 4　Analysis of network scale and software cliff walls relationship

表 3　网络规模指标——曼-惠特尼 U 检验(显著性水平为 0.05)

Table 3　Network scale indicator Mann-Whitney U test

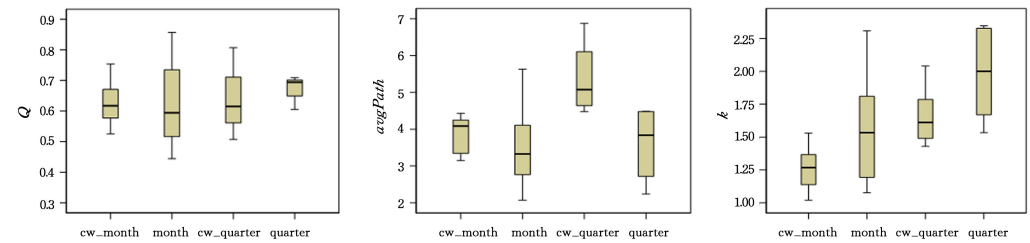
周期	度量指标	显著性(*p-value*)
cw-month vs. month	节点数$\|V\|$	0.294
	连边数$\|E\|$	0.460
	节点更新率δ	0.107
cw-quarter vs. quarter	节点数$\|V\|$	0.024
	连边数$\|E\|$	0.042
	节点更新率δ	0.000

综合上述结果可发现，在软件项目开发过程中，虽然成员多未必一定更好，但当成员更少且他们之间只有有限的交互，同时人员更替较频繁时，从长期角度看更容易出现峭壁现象。

4.3.2　网络结构

图 5(a)给出了两种时间窗口下，峭壁周期与非峭壁周期内 DCN 网络模块度 Q 的分布情况。以月份为时间窗口时，峭壁周期的 DCN 网络模块度 Q 值略高于非峭壁周期的，表现为“cw-month”与“month”的 Q 值中位数分别为 0.614 和 0.599；按季度划分时，峭壁周期的模块度值低于非峭壁周期的，表现为“cw-quarter”与“quarter”的 Q 值中位数分别为 0.615 和 0.694。以上数据说明，短期内，峭壁周期开发者及其之间的交互关系均相对较少，但群体成员的社区特性反而更明显，说明此时的开发者主要还是局部小范围的交互；但是从更长的周期来看，峭壁周期开发者之间交互的紧密程度还是会低于非峭壁周期，因为在非峭壁周期内成员更稳定，随着时间的延长群体之间的整体抱团效果更明显。

图 5(b)和图 5(c)的结果进一步验证了这一结论。峭壁周期的 *avgPath* 中位数值大于非峭壁周期的(cw-month：4.08＞month：3.33，cw-quarter：5.07＜quarter：3.84)，而平均度低于非峭壁周期的(cw-month：1.27＜month：1.53，cw-quarter：1.61＜quarter：2.0)，表现为峭壁周期开发者的“小世界”特性不如非峭壁周期，说明峭壁周期内开发者之间的整体信息连通性比非峭壁情况下的更差。



(a)模块度 Q　(b)平均路径长度 $avgPath$　(c)平均度 k

图 5　网络结构与软件项目的峭壁关系图

Fig. 5　Network structure and software cliff walls diagram

采用曼-惠特尼 U 检验分别对 3 个网络结构指标进行差异显著性分析。如表 4 所列，“cw-month”与“month”下的 3 个网络结构指标以及“cw-quarter”与“quarter”下模块度 Q 的 *p-value* 值，均大于 0.05，说明从统计显著性角度看，两者并没有显著的差别。但是，“cw-quarter”与“quarter”下的平均路径长度 $avgPath$ 和平均度 k 的检验结果 *p-value* 值均小于 0.05，说明按季度分析时，峭壁周期与非峭壁周期内 DCN 的“小世界”特性差异明显。

表 4　网络结构指标——曼-惠特尼 U 检验(显著性水平为 0.05)

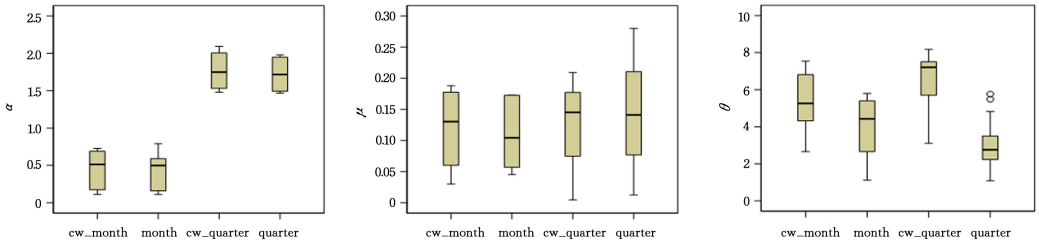
Table 4　Network structure indicator Mann-Whitney U test

对比周期	度量指标	显著性(*p-value*)
cw-month vs. month	模块度 Q	0.682
	平均路径长度 $AvgPath$	0.417
	平均度 k	0.096
cw-quarter vs. quarter	模块度 Q	0.306
	平均路径长度 $AvgPath$	0.009
	平均度 k	0.048

综合上述结果发现，峭壁周期与非峭壁周期的社区抱团现象有差异但不显著，开发者之间的“小世界”特性只在以季度为周期时存在，而且峭壁周期与非峭壁周期 DCN 的差异显著。

4.3.3 网络质量

图 6(a)和图 6(b)给出了不同周期下峭壁周期与非峭壁周期内 DCN 的入度指数 α、出度均值 μ 的分布。结合表 5 的显著性统计差异对比分析不难发现，在 DCN 网络中，无论按哪种方式划分，峭壁周期与非峭壁周期的入度指数 α 的分布均无明显差异，表现为“cw-month”与“month”的 α 值中位数分别为 0.508 和 0.501($p\text{-}value = 0.964$)，“cw-quarter”与“quarter”的 α 值中位数分别为 1.74 和 1.71($p\text{-}value = 0.928$)，说明各阶段内开发人员的活跃情况(受关注程度和被评论情况)对软件项目峭壁的形成造成的影响不明显。峭壁周期的出度指数稍高于非峭壁周期的出度指数，但随着时间周期的扩大，峭壁周期的出度指数逐渐趋近于非峭壁周期的出度指数，表现为“cw-month”与“month”的 μ 值中位数分别为 0.141 和0.104($p\text{-}value = 0.656$)，“cw-quarter”与“quarter”的 μ 值中位数分别为 0.145 和 0.141($p\text{-}value = 0.421$)。



(a)入度指数 α (b)出度指数 μ (c)组织多样性 θ

图 6 网络质量与软件项目的峭壁关系图

Fig. 6 Network quality and software cliff walls diagram

表 5 网络质量指标——曼-惠特尼 U 检验(显著性水平为 0.05)

Table 5 Network quality indicator Mann-Whitney U test

对比周期	度量指标	显著性($p\text{-}value$)
cw-month vs. month	节点入度指数 α	0.964
	节点出度均值 μ	0.656
	多样性 θ	0.107
cw-quarter vs. quarter	节点入度指数 α	0.928
	节点出度均值 μ	0.421
	多样性 θ	0

图 6(c)为软件项目开发成员的组织多样性与峭壁之间的关系图。结果显示，以月为划分周期时，峭壁周期的组织多样性中位数虽然比非峭壁周期大(5.26>4.43)，但统计意义上表现为不明显($p\text{-}value > 0.05$)，而以季度划分时，“cw-quarter”与“quarter”的 θ 值中位数分别为 7.20 和 2.76($p\text{-}value = 0.000$)，即差异明显。结果表明，开发成员来源的组织越多，即成员组织越多样化，对软件峭壁形成的影响就越大，且在以季度为周期时影响更为显著。一种可能的解释是，来自不同组织的成员，如果彼此合作良好，则表现出“1+1>2”的群体互补作用；反之，如果成员因为来源的组织存在差异，彼此之间交互受阻，则表现出“1+1<2”的干扰作用。综上，开发者的活跃情况对软件峭壁有一定影响但并不显著，而开发团队成员的组织来源多样化对峭壁的形成会产生较为明显的影响。

4.4 讨论

综合以上分析结果，我们进一步对影响软件峭壁形成的原因进行归纳，如表 6 所列。

(1)从网络规模角度看，一个周期的 DCN 网络中的节点和连边偏小，且相比上一周期节点更新率变大时，更容易出现峭壁。例如，在项目 Rails 中 2009 年第 3 季团度相比前一个季度，开发人员减少了 60 人，DCN 的连边也相应减少了 92 条，节点更新率达到 0.65，而在该周期提交的 commits 中有两个 commit 的规模超过本文 3.2 节中规定的峭壁定义阈值。对两个大 commit 的内容进行分析，发现其中的 commit (ID=130535)因引入第三方库(code import)增加了大量代码行；另一个 commit(ID=136303)本身规模不大，但有一个对应的 comment 提交了一个很大的源代码文件，其中增加和删除的代码行非常多。因此，在实际应用中，应根据需求合理选择开发团队规模，过小或过大都可能导致峭壁的产生，且应尽量确保团队规模的稳定性。

表 6 3 种视角下软件峭壁的分析结果汇总

Table 6 Summary of software cliff walls analysis results from three perspectives

序号	度量指标	分析角度	分析结论
M1	节点数$\lvert V\rvert$	网络规模	从长期角度看，保持团队规模的稳定性有利于抑制软件峭壁的产生
M2	连边数$\lvert E\rvert$		
M3	节点更新率 δ		
M4	模块度 Q	网络结构	峭壁周期与非峭壁周期的开发者社区抱团现象有差异但不显著，且在以季度为周期时，开发者 DCN 的“小世界”特性差异显著
M5	平均路径长度 $AvgPath$		
M6	平均度 k		
M7	节点入度指数 α	网络质量	社区开发者的活跃情况对软件峭壁有一定影响但并不显著，而开发团队成员的组织来源多样化，对峭壁形成的影响较为明显
M8	节点出度指数 μ		
M9	多样性 θ		

(2)从网络结构角度看，峭壁周期与非峭壁周期的 DCN 模块度有差异，但在统计意义上表现为不显著，说明整体上峭壁周期与非峭壁周期中开发者的合作模式基本是一致的，如新加入成员主要与团队中的核心成员发生合作，而新成员之间的合作非常少，从而形成类星型的子图结构。关于开发者偏好合作模式的详细分析请参考文献[3]。但随着时间周期窗口的加大，存在交互关系的成员会越来越多，此时非峭壁周期 DCN 的“小世界”特性比峭壁周期的更明显，并且这种差异在统计意义上也非常显著。由此可见，使得软件项目成员之间保持有效的连通和聚集，可以在一定程度上避免峭壁的产生。

(3)从网络质量的角度看，无论按月份还是季度划分，出入度指数都偏小，说明社区内大部分开发者的活跃度有限，从图 5(c) 所示的平均度也可以发现，很大一部分开发者的直接合作者数量不超过 2。分析结果表明，在以季度为划分周期时，峭壁与非峭壁周期的成员多样性指标的差异显著。例如，对于项目 Mongo，2010 年第一季度为峭壁周期，其中开发者组织的多样性最大达到 0.821，且该周期相比上一季度，DCN 节点的更新率也增大，达到 0.524。可见，在开发者交互顺畅的前提下加大团队成员的多样性，有助于凸显群体智慧；但若在成员交互不够的情形下增加成员的多样性，则可能导致软件峭壁的形成。

可见，分析开源社区中软件项目开发者的交互关系对软件峭壁的影响时，以季度为周期更为合适，且利用开发成员的更新率、“小世界”特性、组织来源多样性等指标分析软件峭壁时效果更明显。

结束语　针对软件项目群体协作开发过程中异常的 commit 提交行为，本文利用开发者之间的关注和评论关系构建开发者合作网络，并分别从网络规模、网络结构和网络质量 3 个角度尝试了解开发者交互行为与软件峭壁形成的关系。研究结果表明：1)以季度为周期进行分析的效果更好；2)开发团队规模偏小，且成员更新幅度较大时，容易形成软件峭壁；3)保持开发者之间的“小世界”特性有助于避免峭壁的产生；4)开发者的活跃情况对软件峭壁有一定影响但并不显著，而开发团队成员的组织来源多样化对峭壁形成的影响较为明显。

目前，本文仅对 GitHub 上的 9 个代表性开源软件进行分析，虽得出了一些有趣的结论，但仍存在一些有待改进的地方：1)对开发者行为进行分析时，其本身不可避免地存在一定的主观性，因此结论还有待后期的进一步验证；2)本文主要采取 9 个网络度量指标对峭壁周期与非峭壁周期的差异进行分析，并没有具体对每个引发峭壁的 commit 进行详细分析；3)除本文利用的数据外，可进一步考虑代码审查结果、邮件列表信息等，以进一步丰富对软件峭壁的分析，从而加强分析结果的一般性和代表性。

参 考 文 献

[1] BROWN A W, BOOCH G. Reusing Open-Source Software and Practices: The Impact of Open-Source on Commercial Vendors [C]// International Conference on Software Reuse. 2002: 123-136.

[2] YANG B, YU Q, ZHANG W, et al. Influence Factors Correlation Analysis in Github Open Source Software Development Process[J]. Journal of Software, 2017, 28(6): 1330-1342.

[3] HE P, LI B, YANG X H, et al. Research On Developer Preferential Collaboration in Open-Source Software Community [J]. Computer Science, 2015, 42(2): 161-166.

[4] ZHOU M H. Looking for micro-process in large-scale data[C]// Proceedings of the 2nd International Workshop on Evidential Assessment of Software Technologies. New York: ACM, 2012: 39-42.

[5] KALLIAMVAKOU E, GOUSIOS G, BLINCOE K, et al. The Promises and Perils of Mining Github[C]// Proceedings of the 11th Working Conference on Mining Software Repositories. New York: ACM, 2014: 92-101.

[6] LI W P, WANG J B, LIN Z Q, et al. Software Knowledge Graph Building Method for Open Source Project[J]. Journal of Frontiers of Computer Science & Technology, 2017, 11(6): 851-862.

[7] JUNG H W, KIM S G, CHUNG C S. Measuring Software Product Quality: A Survey of ISO/IEC 9126[J]. IEEE Software, 2004, 21(5): 88-92.

[8] GIRBEA A, SUCIU C, NECHIFOR S, et al. Design and Implementation of a Service-Oriented Architecture for the Optimization of Industrial Applications[J]. IEEE Transactions on Industrial Informatics, 2014, 10(1): 185-196.

[9] MENEELY A, WILLIAMS L, SNIPES W, et al. Predicting Failures with Developer Networks and Social Network Analysis [C]// Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM, 2008: 13-23.

[10] LIMAM N, BOUTABA R. Assessing Software Service Quality and Trustworthiness at Selection Time[J]. IEEE Transactions on Software Engineering, 2010, 36(4): 559-574.

[11] MACLEAN A C, PRATT L J, KREIN J L, et al. Trends that Affect Temporal Analysis using Sourceforge data[C]// Proceedings of the 5th International Workshop on Public Data about Software Development. North Carolina, USA, 2010: 6-11.

[12] MACLEAN A C. Commit Patterns and Threats to Validity in Analysis of Open Source Software Repositories [D]. Utah: Brigham Young University, 2012.

[13] PRATT L J. Cliff Walls: Threats to Validity in Empirical Studies of Open Source Forges[D]. Utah: Brigham Young University, 2013.

[14] CHENG C, LI B, LI Z Y, et al. Developer Role Evolution in Open Source Software Ecosystem: An Explanatory Study on GNOME[J]. Journal of Computer Science and Technology, 2017, 32(2): 396-414.

[15] PRATT L J, MACLEAN A C, KNUTSON C D, et al. Cliff Walls: An Analysis of Monolithic Commits Using Latent

Dirichlet Allocation[C]// IFIP International Conference on Open Source Systems. Springer, 2011: 282-298.

[16] CHEN D, WANG X, HE P, et al. Towards Understanding Existing Developers' Collaborative Behavior In OSS Communities [J]. ComputerScience, 2016, 43(6A): 476-479.

[17] GRÖNLUND M, JEFFORD-BAKER J. Measuring correlation between commit frequency and popularity on GitHub[D]. Stockholm: KTH Royal Institute of Technology, 2017.

[18] SINHA V S, MANI S, SINHA S. Entering the circle of trust: developer initiation as committers in open-source projects[C]// Proceedings of the 8th Working Conference on Mining Software Repositories. 2011: 133-142.

[19] MA Y T, WU Y, XU Y W. Dynamics of open-source software developer's commit behavior[C]// Proceedings of the 29th Annual ACM Symposium on Applied Computing (SAC'14). New York, USA: ACM Press, 2014: 1171-1173.

[20] GOUSIOS G. The GHTorent dataset and tool suite[C]// Proceedings of the 10th Working Conference on Mining Software Repositories. 2013: 233-236.

[21] Struggling in IT. GitHub 2018 Annual Report [EB/OL]. http://www.wh-ford.com/f828820/20181030A1WJZ800.html.

[22] HINDLE A, GERMAN D M, HOLT R C, et al. Automatic Classification of Large Changes into Maintenance Categories[C]// IEEE International Conference on Program Comprehension. IEEE, 2009: 99-108.

[23] ARAFAT O, RIEHLE D. The Commit Size Distribution of Open Source Software[C]// Hawaii International Conference on System Science. IEEE, 2009: 1-8.

[24] GU Q, CHEN D X. Validation and Simulation of Software System Evolution Rules Using Software Networks[J]. Scientia Sinica Informationis, 2014, 44(1): 20-36.

[25] GU Q, XIONG S J, CHEN D X. Correlations between characteristics of maximum influence and degree distributions in software networks[J]. SCIENCE CHINA Information Sciences, 2014, 57(7): 1-12.

[26] HE P, WANG P, LI B, et al. An Evolution Analysis of Software System Based On Multi-Granularity Software Network[J]. Acta Electronica Sinica, 2018, 46(2): 257-267.

[27] PAN W F, LI B, MA Y T, et al. Multi-Granularity Evolution Analysis of Software Using Complex Network Theory[J]. Journal of Systems Science and Complexity, 2011, 24(6): 1068-1082.

[28] NEWMAN M E J. Fast Algorithm for Detecting Community Structure in Networks[J]. Physical Review E, 2003, 69(6): 066133.

HE Peng, born in 1988, Ph.D, associate professor, master supervisor, is a member of China Computer Federation. His main research interests include software measure, software defect prediction, and service recommendationand complex network.