文章编号:1672-3813(2019)04-0031-13;DOI:10.13306/j.1672-3813.2019.04.004

开源软件社区开发者协作网络结构演化分析
——以 Cloud Foundry 社区为例

刘 鹏,张鹏臣,王念新
(江苏科技大学经济管理学院,江苏 镇江 212003)

摘要:已有关于开源软件社区协作模式的研究针对开发者协作网络的静态结构展开了广泛探讨,对其结构演化特性的分析相对较少。本文以 Cloud Foundry 社区为例,通过代码修订关系构建开发者协作网络,并对其结构和演化过程进行了分析。研究结果表明开发者协作网络的演化以其最大连通子图结构变化为标志,分别为"松散连接"状态、"链式"结构和具有"核心—边缘"结构的多模块小世界状态。最大连通子图的演化过程与子项目内在关联。此外,开发者协作关系表现出倾向性连接、同质相吸、差异偏好相结合的特征。本研究有助于进一步认识开源软件社区协作的模式,亦是对大规模群体开放式协作创新活动的研究工作的丰富。

关键词:Cloud Foundry;开发者协作网络;开源软件社区;结构演化
中图分类号:C936;N93 **文献标识码**:A

Structure and Evolution of Developer Collaboration Network in Cloud Foundry OSS Community

LIU Peng, ZHANG Pengchen, WANG Nianxin
(School of Economics and Management, Jiangsu University of Science and Technology, Zhenjiang 212003, China)

Abstract: The static structures of developer collaboration networks in OSS communities have been widely discussed in the literature. However, there has been limited research on the evolutionary process of these networks. In this paper, we construct the developer collaboration network of the Cloud Foundry community by code-collaboration relationships and analyze its structure and evolution. The results show that the collaboration network evolution is characterized by the structural variations of its giant component, namely "loosely connected" state, "chain-module" structure, and multi-modular small-world network with a "core-periphery" structure. Meanwhile, the evolution of the giant component is intrinsically related to the sub-projects. Besides, the collaborative relationship between developers shows the combined features of preferential-attachment, homophily, and heterophily. The overall results may be not only helpful to deepen our understandings of the collaborative pattern in OSS communities, but also enrich the studies of open innovation in the large-scale crowd.

Key words: Cloud Foundry; developer collaboration network; OSS community; structure andevolution

收稿日期:2018-09-17;**修回日期**:2019-10-26
基金项目:国家自然科学基金(71871108);江苏高校哲学社会科学项目(2017SJB1092)
作者简介:刘鹏(1982-),男,河北赤城人,博士,讲师,主要研究方向为复杂社会网络演化及分析、知识管理。
通讯作者:张鹏臣(1996-),男,江苏如皋人,硕士研究生,主要研究方向为复杂网络。

0 引言

近十多年来，开源软件得以蓬勃发展，广泛应用于人工智能、虚拟现实、大数据分析等诸多领域。伴随着 web2.0/3.0 技术不断成熟而涌现的在线社区是孕育开源软件的主要场所。与传统商业软件相比，开源软件在开发模式上表现出很强的自主性。具体而言，在缺少中央调控的情况下，众多社区成员通过彼此间自发的协调与合作，完成了软件开发过程中复杂问题的发现与求解。这种看似“乌合之众”的软件开发模式的成功引起了学界的广泛关注，相应地，开源软件社区中开发者之间的协作模式逐渐成为了计算机科学、管理科学、创新管理等领域的重要的研究课题[1-4]。

关于开源软件社区的开发者协作模式的研究，Raymond 做出了先驱性工作，他指出大多数商业软件的开发采用了大教堂模式，而以 Linux 为代表的开源软件则使用了集市模式[5]，即在足够多的注视下，软件的缺陷将无所遁形。这一论断使人们对于开源软件的开发模式产生了全新的认识。然而，有学者指出，开源软件，尤其是大型开源软件的开发是一项知识密集型的工程，社区成员间的交互具有内在的自治性与复杂性，集市模式还不足以揭示社区成员的协作模式[6-8]。特别是，开源软件社区本质上可以理解为一个知识型复杂自适应系统[2,9]，针对这一特点，学界从复杂网络视角对开软社区的协作模式展开了诸多有益的探讨。

在宏观方面，相关研究主要关注整体社区层面上社区成员间协作关系所表现出的规律与特性[10-15]。例如，Bird 等利用 Apache 服务器项目社区邮件通讯数据构建了社区开发者协作网络，结果显示网络中度分布具有明显的无标度特性[13]。Singh 通过收集 Sourgeforge 上 15 个开源软件社区的日志文件和邮件通讯数据，构建开发者协作网络，研究结果表明这些协作网络具有不同程度的小世界特性，并且这一性质对软件的成功具有相关性[14]。除此之外，一些学者发现开源软件社区成员间的协作网络具有“核心—边缘”结构[16-19]，即网络中的节点可以分为两类，核心节点间交互密切，边缘节点间几乎不存在密切的协作关系。

微观层面的研究工作主要围绕建立交互关系双方的属性特点、现有关系结构等方面展开[20-24]。例如，Hu 等针对社区中个体属性特征（如相互熟识程度）与交互关系建立的相关性进行了分析[20]。Joblin 等通过分析 18 个开源软件的开发者网络，发现层级结构会随着时间推移而转变为一种混合结构（即层级结构只出现于核心开发者的交互关系之中，而边缘开发者之间并不具有这一特征），说明开发者在网络中的位置影响其新的交互关系的建立[21]。

已有相关工作使我们对开源软件社区的协作模式有了更加深入的认识，但是在以下两个方面还有待进一步深化。首先，已有工作主要利用邮件交流数据来构建开发者协作网络，而这些数据往往包含了开发者邮件和用户邮件，难免影响研究结果的准确性，同时也限制了更深层分析工作的开展。其次，现有研究工作主要关注社区开发者协作网络的静态结构特征，其结构随时间推移而展现出演化特性也不容忽视，特别是推动网络演化的协作关系的形成特点还有待更加深入的探讨。

对此，本文以 Cloud Foundry 项目社区为例，通过收集社区开发者代码提交数据，利用代码协作关系建立开发者协作网络（即代码协作网络），并对其结构及演化特性展开分析。本文力图从复杂网络视角为开源软件社区协作模式的分析工作提供一种新的思路。此外，开源软件开发也属于开放式创新的一种具体模式，现有相关工作主要聚焦于科研合作、专利联合研发等领域，本研究亦是对该方面工作的丰富。

1 数据及分析方法

1.1 数据的获取

Cloud Foundry 最初由 VMware 公司发起，现由 Cloud Foundry 基金会（非盈利性组织）管理的开源云平台项目，也是云应用和云服务领域最早的项目之一。截止至 2019 年 1 月，该项目共涵盖了 CLI（Official Command Line Client）、UAA（User Account & Authentication Server）、Routing 等 41 个子项目。这些子项目均通过 git（一个开源的分布式版本控制系统）进行代码的提交和修改，相应 git 记录了不同时间段所有参与项目开发人员的提交记录。

由此，本文利用 git 收集了 Cloud Foundry 从 2010 年 8 月（项目的初始期）到 2019 年 1 月所有子项目下的提交记录，共获取了 586 727 条提交数据。其中，每条提交数据包含了提交者的邮箱地址、提交时间、代码修改情况及所涉及的文件等信息，如图 1 示例所示。

```
zyjiaobj@cn.ibm.com##Fri Dec 7 11:39:16 2018 +0800##zyjiaobj
2 files changed, 4 insertions(+), 4 deletions(-)
app-autoscaler-release/templates/app-autoscaler-deployment-fewer.yml
app-autoscaler-release/templates/app-autoscaler-deployment.yml
```

图 1　开发人员提交记录数据示例

Fig.1　Illustration of one piece of commit records

1.2　分析方法

1.2.1　网络构建方法

本文主要通过社区开发者之间的代码协作关系构建开发者协作网络,网络构建过程分为以下步骤。

首先,通过每个提交记录中的电子邮件地址将开发人员彼此区别开来,即如果两条记录是通过相同的电子邮件地址进行提交,那么认为这两条记录的代码是由同一个开发者编写的,相应不同的邮箱地址抽象为网络中不同的节点。

然后,将开发者之间的代码协作关系抽象为网络中的边。关于代码协作关系的抽取,由于每个子项目都是按照版本进行发布的,新版本往往是对旧版本功能的完善和提升,并且由两个版本发布时间间隔内进行过代码提交的开发者共同完成,相应子项目的每个新版本可以看作是上述开发者相互协作而形成的独立的知识产品。由此,本文中开发之间协作关系的抽取标准是:在子项目两个相邻版本发布的时间间隔内,两个开发者是否针对同一个子项目文件进行过代码提交,具体抽取过程如图 2 所示。

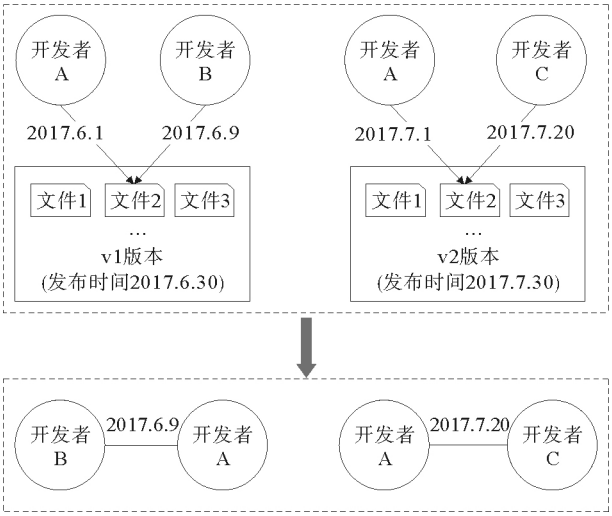


图 2　开发者协作关系抽取过程示意图

Fig.2　Diagram of the extraction of collaborative relationships between two developers

图 2 中,假设某个子项目的 v1 版本发布时间为 2017 年 6 月 30 日,开发者 A 和 B 分别在这个时间之前对该子项目中的文件 2 进行了代码提交;随后,在 v1 版本发布之后到 v2 版本发布之前,开发者 A 和 C 也针对文件 2 进行了提交。相应地,开发者 A 分别和 B、C 建立代码协作关系,而开发者 B 和 C 之间无代码协作关系,每组协作关系上的时间戳设定为 B、C 的提交时间。

最后,在对数据集中的所有代码协作关系抽取的基础上,根据协作关系建立的时间戳的不同,以 6 个月的时间间隔构建不同时间窗(17 个时间窗)下的累积协作网络,如 2010. 8～2011. 1 的协作网络、2010. 8～2011. 7 的协作网络等。

1.2.2　网络分析方法

本文主要采用社会网络分析方法对所构建的协作网络展开分析。网络规模用节点数量表示,节点度表示与目标节点直接相连的节点数量。连通子图表示网络中一部分直接或间接相连节点构成的子图,其中规模最大的连通子图称为最大连通子图。

除了上述基本统计指标,本文通过聚集系数、平均最短路径长度、模块度 3 个指标来分析网络的结构特性。聚集系数表示网络中三角形数量与三元组数量的比值;平均最短路径长度为任意两个节点实现连接所需最少边数的均值;模块度主要衡量网络中是否存在多个边密度较高的局部区域(即模块化结构)。本文采用 Louvain 算法[25]计算网络的模块度。

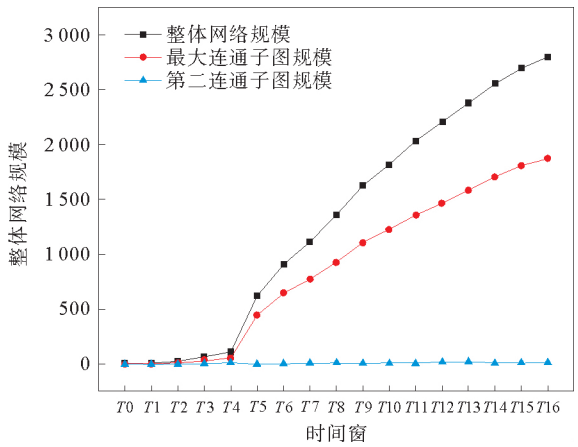


图 3　开发者协作网络整体结构演化特性

Fig.3　Structural characteristics of the overall network

2　分析结果

2.1　网络的整体结构演化特性

图 3 给出了所构建的协作网络的规模变化,其中 17 个

时间窗分别表示为 $T0\sim T16$。随着时间窗的推移，整个网络的规模不断增大，从 $T4$ 时间窗开始，一个规模远大于第二连通子图（蓝色实线）的最大连通子图（红色实线）逐渐涌现出来。从图 4 各个时间窗下网络的拓扑结构也可以直观的看出最大连通子图（彩色子图）规模增长的同时，其结构也发生了显著变化。这表明在 Cloud Foundry 项目实施的过程中，数量可观的开发者通过协作关系自发形成了汇聚，为了分析其结构特性，本文针对 $T16$ 网络（即所获取数据集中的终态网络）展开进一步探查。

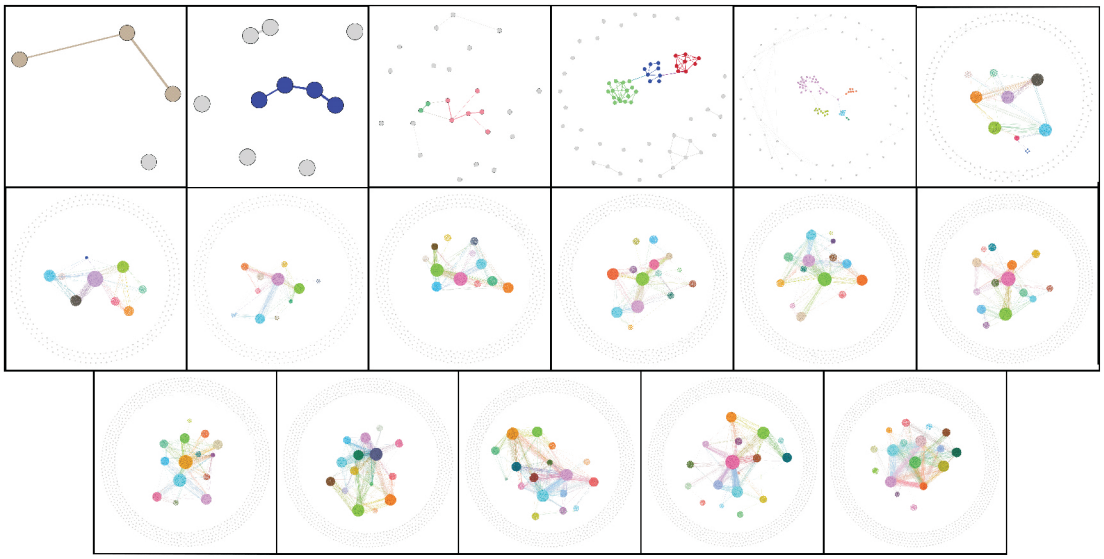


图 4 各个时间窗下网络的拓扑结构

Fig.4 Topologies of the developer collaboration network in different time windows

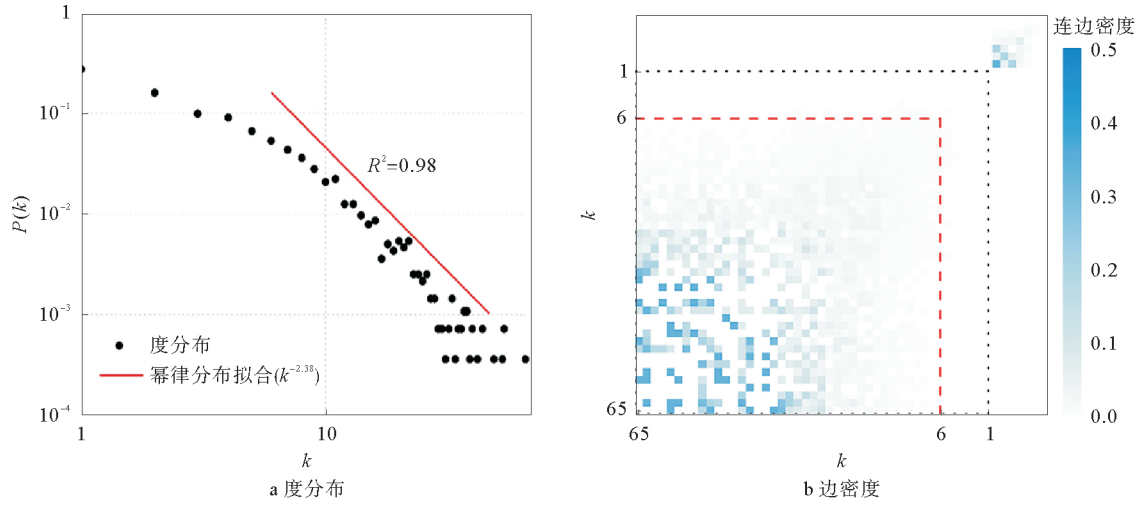


图 5 开发者协作网络的度分布及边密度（$T16$）

Fig.5 Degree distribution and Edge-density of the developer collaboration network（$T16$）

在双对数坐标下，图 5a 绘制了网络中不同度（k）节点的数量占比（$P(k)$）情况。从图中可以看出，随着节点度的升高，对应的节点数量占比（黑色实心圆）呈现下降趋势，并且在总体上符合幂指数为 -2.38 的幂律分布（红色实线）。这一现象说明少数开发者拥有大多数的协作关系，而大多数开发者的合作伙伴的数量相对较少。由此网络中可能存在一定的层次结构，图 5b 对这一现象进行了分析。

图 5b 中，分别针对最大连通子图和外围节点（即最大连通子图之外的子图），按照度由高到低的顺序，绘制了不同度（k）节点间的连边密度。其中，黑色虚线对最大连通子图和外围节点进行了标识（即黑色虚线左下区域为最大连通子图，右上区域为外围节点）。最大连通子图中，随着节点度的下降，不同度节点间的连边密度逐渐下降，基本可以划分为两个区域（由红色虚线表示），而从图中可以看出，第一个区域（$k\geqslant 6$）的边缘密度明显高于第

二个区域($k<6$)，这说明最大连通子图中存在“核心—边缘”结构。与此同时，与最大连通子图中的节点相比，外围节点的连边十分稀疏，说明外围节点的提交行为存在一定的偶发性。这一结果意味着协作网络中的开发者可以根据交互关系的密集程度划分为 3 个层次，分别是核心开发者(最大连通子图 $k\geqslant 6$ 节点)、边缘开发者(最大连通子图 $k<6$ 节点)、和外围开发者(非最大连通子图节点)。

表 1　开发者协作网络提交情况($T16$)

Tab.1　Code changed and the number of commits of three types of nodes ($T16$)

分类	度范围	规模(占比)	提交量(次)	代码修改量(行)
外围节点	$0\leqslant k<6$	33.1%	27 953	$1.166\ 46\times10^{7}$
最大连通子图节点	$1\leqslant k<6$	41.6%	155 000	$2.201\ 59\times10^{7}$
	$k\geqslant 6$	25.3%	403 774	$3.493\ 59\times10^{7}$

基于图 5 的结果，表 1 从提交量和代码修改量两个方面对整个协作网络中 3 类节点(即最大连通子图中的核心节点和边缘节点，以及外围节点)的提交记录进行了统计。对比 3 类节点的提交情况可以发现，整个项目的代码编写工作主要由最大连通子图中的节点完成(代码修改量和提交量分别占总体数量的 98%和 95.2%)；虽然外围节点对于整个项目的贡献并不显著，但是这些外围节点在一定程度上分担了最大连通子图节点的工作量，对于项目的推进具有积极作用。最大连通子图中，占整个网络规模约 1/4 的核心节点(即 $k\geqslant 6$ 节点)的代码修改量和提交数量均超过 60%，完成了项目开发的大部分工作，是 Cloud Foundry 项目实施的主力开发者；边缘节点($1\leqslant k<6$ 节点)在 3 类节点中数量最多，虽然代码修改量(37.9%)和提交量(26.4%)低于核心节点，但远远超过外围节点。结合图 5 的结果(即边缘节点不可避免地与核心节点发生联系)，我们基本上可以断定边缘节点对于核心节点的工作起到了补充作用。因此，在 Cloud Foundry 项目实施过程中，最大连通子图中的开发者是主力军，其中以“核心”开发者作为骨架；而最大连通子图之外的开发者作为后备力量参与开发工作。

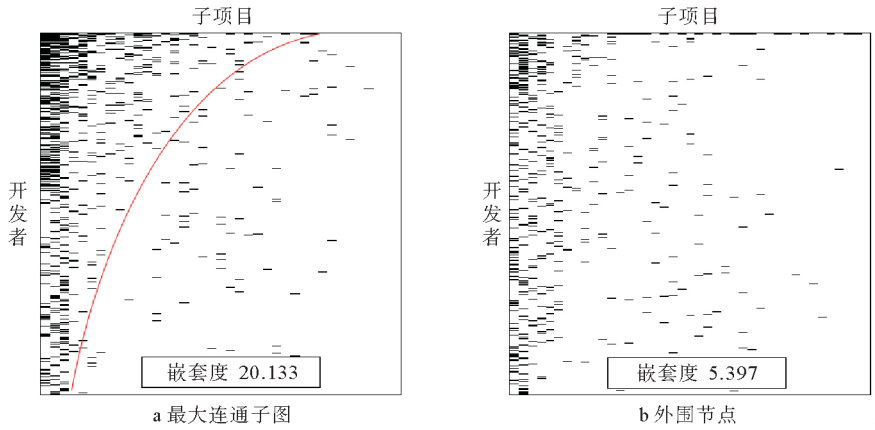


图 6　最大连通子图与外围节点中开发者——子项目二分网络嵌套性($T16$)

Fig.6　Nestedness of developer-subproject bipartite network in giant component and isolated nodes ($T16$)

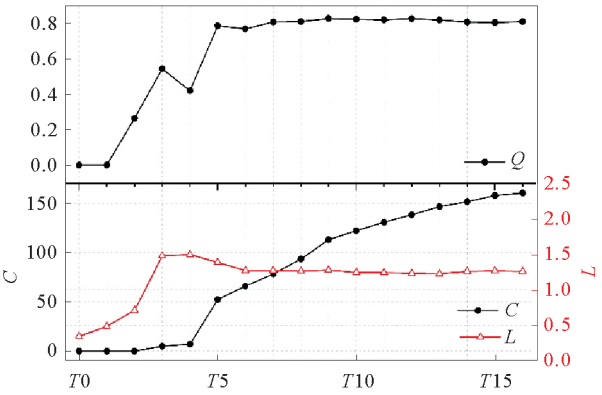
表 1 的提交量分析反映出最大连通子图的开发者承担了绝大部分的开发工作，然而就工作类别(即参与不同的子项目)而言，是否也具有类似现象还值得进一步考察。进而，本文利用生态学中嵌套性理论[26]，分别对最大连通子图和外围节点中，开发者与子项目对应关系构成的二分网络的嵌套结构进行了分析，如图 6 所示，其中嵌套度数值越高说明嵌套结构越清晰。从图中可以看出，最大连通子图(图 6a)的开发者—子项目网络左上角具有一个较清晰的三角形结构，并且嵌套度数值约为 20.133，而外围节点(图 6b)的二分网络中，开发者与子项目间的关系较为分散，相应的嵌套度数值为 5.397。这一结果说明，与外围节点相比，最大连通子图中开发者与子项目间存在明显的嵌套结构，即存在一部分开发者会对大多数子项目贡献代码，其余的开发者则围绕少数子项目进行开发。因而，这种嵌套性结构的存在对于整个项目开发延续性的维持具有一定的积极作用。换言之，只要关注多个子项目的开发者持续提交代码，关注少数子项目的开发者的随机流失并不会对整个项目的开发工作造成严重影响。

2.2 最大连通子图结构及演化特性

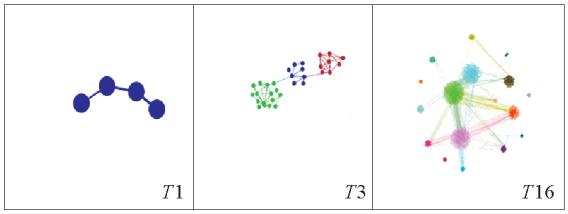
从 2.1 节的分析可以看出，最大连通子图对于整个项目的实施发挥了至关重要的作用，其结构在不同的时间窗下也表现出明显的变化。因此，本部分针对最大连通子图，从宏观、介观和微观 3 个层面对其结构演化展开进一步的探查。

2.2.1 宏观层面分析

图 7 绘制了开发者协作网络最大连通子图的聚集系数(C)、平均最短路径长度(L)和模块度(Q)随着时间窗推移的演化情况，其中，C 和 L 分别为实际数值与同规模随机网络聚集系数和平均最短路径长度的比值。整体上，3 个衡量指标的数值都出现了不同程度的升高，但是通过对比三者的变化，可以发现最大连通子图在演化过程中存在 3 种不同的结构状态。



a C、L、Q数值变化



b 典型拓扑结构

图 7 最大连通子图 C、L、Q 数值变化情况及典型拓扑结构

Fig.7 Evolution of C, L and Q in the giant component and Typical topologies of the developer collaboration network

第一种状态出现在 $T0\sim T2$ 时间窗下，最大连通子图的模块度、聚集系数和平均最短路径长度的数值很小($C\approx 0$，$L<1.0$，$Q<0.4$)，说明此时的网络中连边稀疏，且很难划分出显著的模块化结构，最大连通子图主要表现为“松散连接”的状态(如图 7b$T1$ 网络)。

随后，在时间窗由 $T3$ 到 $T5$ 的移动过程中，最大连通子的模块度和聚集系数分别出现了不同程度的升高($Q\rightarrow 0.8$，$C\rightarrow 50$)，平均最短路径长度则保持在相对较高的水平($L\approx 1.5$)。这一结果表明，此时最大连通子图出现了高度模块化的状态，并且大多数模块间并未形成相互连接，任意两个模块间的联系往往要通过第三方模块，相应最大连通子图在整体上出现了多个模块依次相连的“链式结构”(如图 7b$T3$ 网络)。

从 $T6$ 时间窗开始，最大连通子图在结构上展现出第三种状态：模块度和平均最短路径长度在数值上未发生明显改变($Q\approx 0.8$，$L\approx 1.3$)，说明最大连通子图维持高度模块化结构的同时，模块间逐渐相互连接起来；聚集系数的持续增大意味着节点间的连边变得更加紧密。由此，可以断定最大连通子图宏观上涌现出“多模块的小世界”状态(如图 7b$T16$ 网络)。

网络的典型拓扑结构也直观地显示了这种现象，虽然 3 个时间窗下的网络均呈现分割状态，但与 $T1$ 和 $T3$ 的网络相比，$T16$ 的最大连通子图(彩色的子图)规模明显增大，并且在结构上也发生了显著变化，其余的小规模聚簇和孤立点散布于其外围。

2.2.2 介观层面分析

本节主要围绕最大连通子图中模块的形成和演化展开介观层面的分析。通过比较相邻两个时间窗下最大连通子图模块成员的重叠情况，图 8 对最大连通子图中模块的演化过程进行了绘制。其中，黑色竖线表示最大连通子图中的模块，长度为对应模块的相对规模，文字表示模块编号；彩色流标识了模块间的演化关系。例如，#1 为 $T2$ 最大连通子图中规模最大的模块，逐渐与 #0 模块合并形成 $T3$ 最大连通子图的 #1 模块。

图 8 中，对比不同时间窗下模块间的演化关系可以发现，一方面，最大连通子图上不断有新的模块涌现出来(如 $T3$ 最大连通子图的 #0 和 #2 模块)；另一方面，最大连通子图上的模块主要通过自身发展和合并其它模块的方式实现规模的扩张，例如，时间窗 $T6$ 中规模最大的模块 #8 由 $T5$ 的 #3 模块演化而来，第二大模块 #7 则是通过 $T5$ 的 #4 和 #6 模块合并而形成。在这两方面因素的共同作用下，最大连通子图在规模扩张的同时展现出了多模块的结构状态。

为了进一步检测最大连通子图中模块的演化与软件子项目开发之间的关系，在图 8 的基础上，本文通过统计各个模块在不同子项目上的提交情况，分析了模块与子项目之间的关联性，如图 9 所示。其中，每个矩形表示模块，矩形上的文字标明了模块编号(与图 8 模块编号对应)、提交量降序排列后排名前 4 的子项目编号；每个时间窗后的数值为最大连通子图提交记录所涉及的子项目数量与相应时间段软件中子项目总量的比值。例如，$T4$ 时间窗下，#4 模块提交量前 4 的子项目分别是 *16(cf-release)、*20(cloud-controller-ng)、*10(bosh)、*27

(garden),$T4(1.0)$表示最大连通子图的提交记录涉及此时间窗下所有子项目。

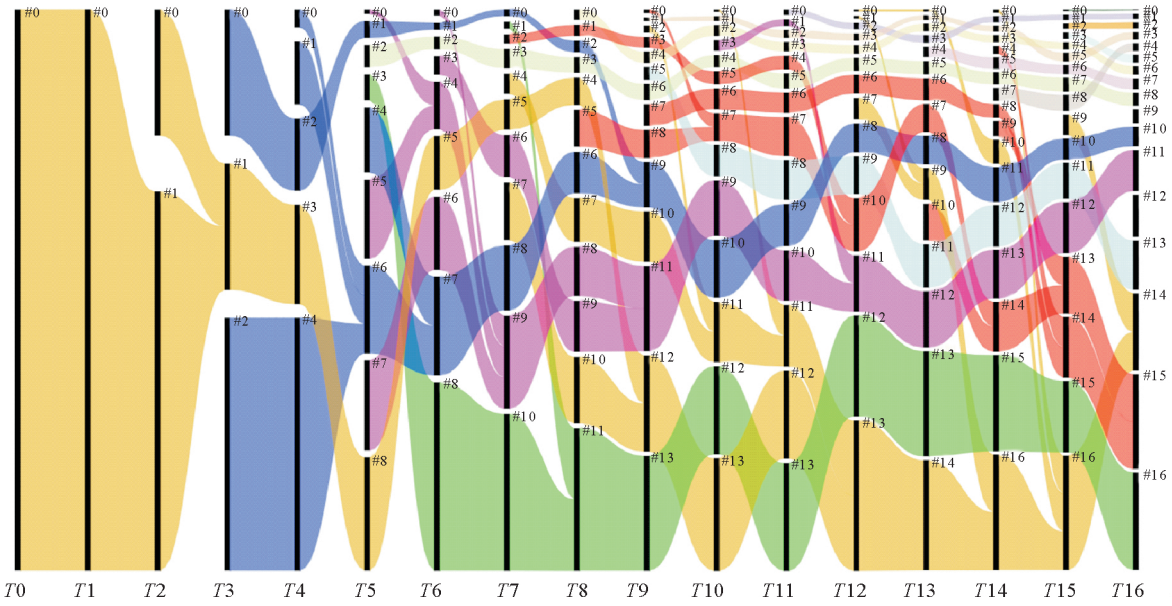


图 8　最大连通子图中模块的演化过程

Fig.8　Evolution of modules in the giant component

图 9 中,不同时间窗下,最大连通子图提交记录所涉及的子项目数量与相应子项目总量的比值均为 1.0,说明开发者协作网络的最大连通子图会对软件项目包含的所有子项目进行代码提交。进一步观察模块演化时高提交量子项目的变化情况可以发现,各个时间窗下最大连通子图上的模块均会围绕特定的子项目进行代码集中提交。例如,在 $T15$ 时间窗下,模块 #10 主要关注子项目 *16(cf-release),而模块 #13 对子项目 *19(cli)做了大量的代码贡献。此外,模块的演化和合并也与子项目有关。

在模块自身发展的过程中,每个模块代码集中提交的子项目较好地保持了延续性,并且这些子项目往往具有一定的软件功能相关性。例如,$T11$ 的 #0 模块演化为 $T16$ 的 #1 模块的过程中,始终主要围绕子项目 *25(fissile)、*20(cloud-controller-ng)、*21(consul-release)和 *16(cf-release)进行代码提交。其中,fissile 的功能是 Cloud Foundry 应用发布部署的容器调度;cloud-controller-ng 用于 Cloud Foundry 应用的服务、用户角色等的管理,实现开发者工作流的改善;consul-release 是一个分布式的键值存储程序,为 Cloud Foundry 应用基础框架提供服务发现、密钥值配置和分布式锁;cf-release 提供 Cloud Foundry 应用中单个组件发布的部署规范。这 4 个子项目均面向 Cloud Foundry 应用,涉及应用的开发管理、权限配置、组件发布和服务管理。

在模块合并的过程中,能够形成合并的两个模块在代码集中提交的子项目上往往存在交叉,如 $T11$ 的 #1 和 #10 合并为 $T12$ 的 #11(图 8),合并前两个模块提交量前四的子项目中均出现了子项目 *10、*16、*7,合并后依然会对这 3 个子项目进行代码集中提交。

综合图 8 和 9 的分析可以得出,开发者协作网络的模块与子项目的开发存在着内在联系,并且不断地为特定的子项目贡献源代码。与此同时,开发者协作网络的最大连通子图通过新模块的涌现和现有模块的发展与合并实现规模的扩张,以及多模块相互连接状态的维持。

2.2.3　微观层面分析

开发者之间的代码修订关系(即协作关系)是协作网络结构演化的基础,从前文的分析可以看出开发者间的协作关系兼具结构和属性的特性。在结构方面,协作关系分布并不均匀,呈现幂率特性(图 5a),说明开发者现有的协作伙伴数量(结构特性)会影响进一步协作关系的形成。在属性方面,每个模块内的开发者往往围绕特定的功能相关的子项目进行代码集中提交,不同模块所涉及的子项目之间也存在差异(图 9)。此外,由于子项目功能的实现往往需要开发者具备较强的相关技术背景,开发者所关注的子项目本质上反映了开发者的技术背景(即属性),由此可见,属性的差异可能存在于模块内与模块间的关系中。因此,本节针对 $T16$ 的最大连通子图(此时的最大连通子图是一个多模块小世界网络),从结构和属性两个方面对模块内外协作关系的特点展开进一步的检测。

1)协作关系的结构特性分析

借鉴 Guimerà 等的研究工作[27],本文提出了模块内外节点连边特征的考察方法,分别如式(1)和(2)所示。

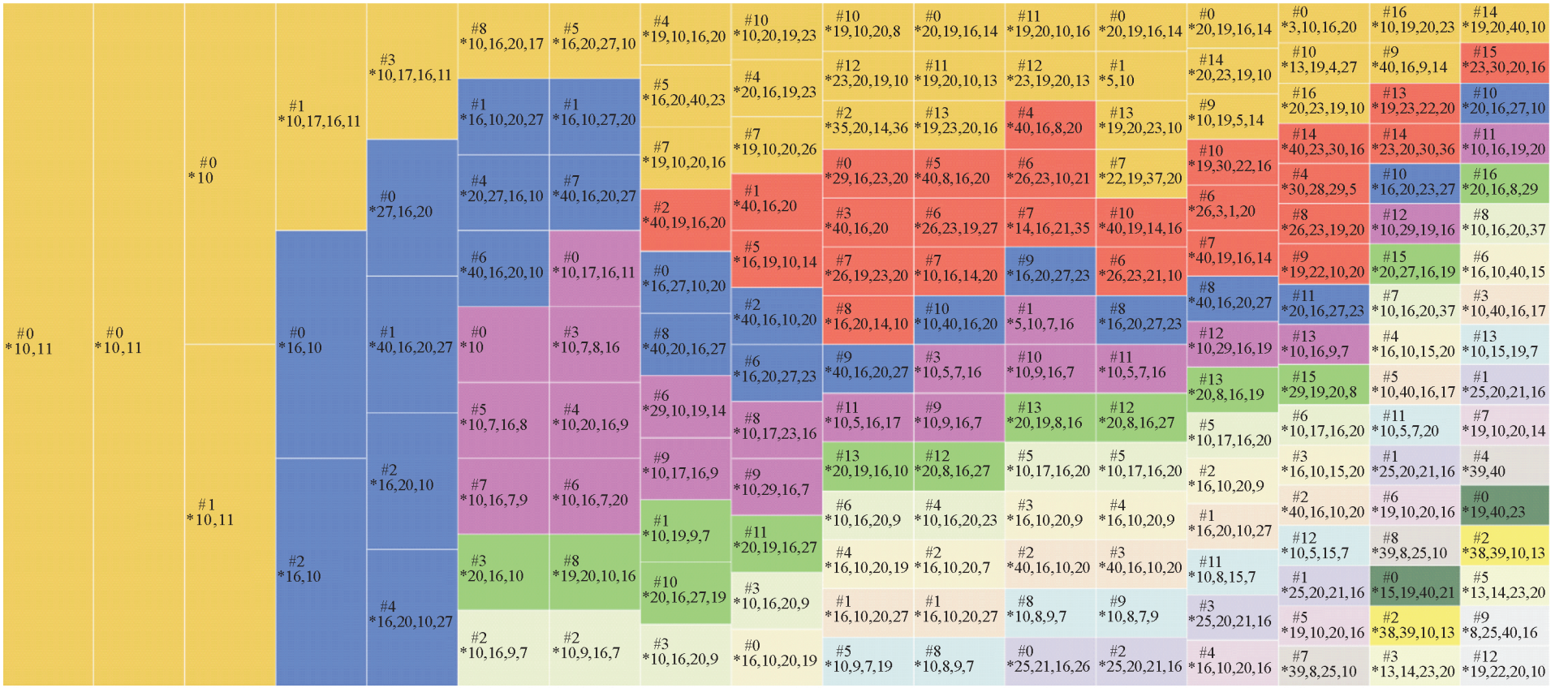


图9　最大连通子图模块演化过程中在子项目上的提交情况

Fig.9　Relations between module evolution and subproject development

式(1)中，$l_{i,km}$ 表示模块 m 中度为 k 的节点 i 与模块内其余节点的连边数量，$\bar{l}_m$ 和 σ_m 分别为模块 m 中所有节点在模块内连边数量的均值和标准差，n 表示模块 m 内 k 度节点的数量。相应地，z 值主要考察模块内 k 度节点连边的分布情况，即 z 值越高，k 度节点与模块内其余节点的连边数量越多；反之，z 值越低，k 度节点与模块内其余节点间的连边数量越少。

$$z=\frac{1}{n}\times\sum_{i=1}^{n}\frac{l_{i,km}-\bar{l}_m}{\sigma_m} \tag{1}$$

式(2)的 p 值则主要衡量模块 m 中 k 度节点在跨模块连边中发挥的作用，其中，$l_{io,km}$ 表示模块 m 的跨模块连边中 k 度节点 i 参与数量，n 为模块 m 内 k 度节点的数量。

$$p=\frac{1}{n}\times\sum_{i=1}^{n}\frac{l_{io,km}}{k} \tag{2}$$

图 10 对 $T16$ 最大连通子图中各个模块内不同度节点的 p、z 值进行了绘制。其中，上横坐标注明了模块编号及规模，下横坐标为各个模块中按照度值由低到高排列的节点度(k)，节点度为 6 的位置用红色虚线进行了标识。

从图 10 可以看出，除模块 #0 之外，各模块的 z 值随着节点度的升高主要呈现上升趋势，并且每个模块的 $k\geqslant 6$节点的 z 值基本上都大于 0，说明这些模块中 $k\geqslant 6$ 节点模块内连边数量较多。进而，结合表 1 统计结果可以基本断定，最大连通子图中的核心开发者($k\geqslant 6$ 节点，在最大连通子图规模占比约为 38%)分散在不同的模块中，与对应模块中的其他开发者连接更好。换言之，核心开发者在每个模块中都扮演中心角色，并且模块内的关系通常围绕其形成。对于模块 #0，由于其规模过小，不存在 $k\geqslant 6$ 节点，故模块内的关系几乎没有表现出任何结构性的特征。

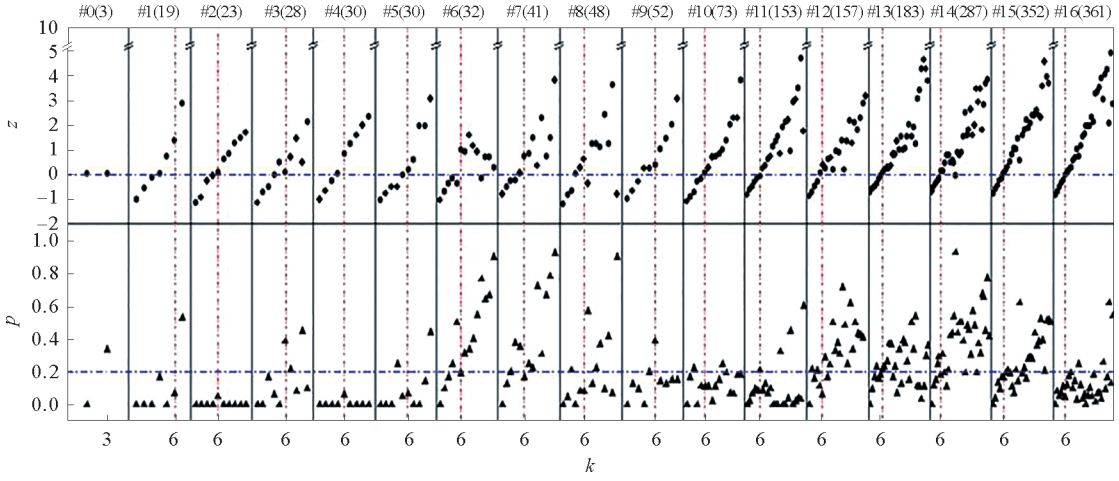


图 10　最大连通子图($T16$)模块中不同度节点的 p、z 值

Fig.10　p and z value of k-degree nodes in each module of the giant component ($T16$)

随着模块中节点度的升高，p 值的变化并不具有显著的规律性。但是通过比较分析每个模块高度和低度节点的 p 值可以发现，模块间连接的形成往往涉及 $k\geqslant 6$ 节点。例如，模块 #12 中，$k\geqslant 6$ 节点的 p 值高于 0.2，最大值可以达到 0.7 左右，而 $k<6$ 节点 p 值的最大值接近于 0.2。虽然在模块 #0 中没有 $k\geqslant 6$ 节点，但是度最高的节点(即 $k=3$)也有最大的 p 值。这些结果说明模块间连边的形成往往需要核心开发者的参与。

从 p、z 值的变化可以看出，模块内外连接的形成都与核心节点有关。结合网络中度分布的幂率特性，可以推测出最大连通子图上的连边具有结构上的倾向性。图 11 进一步绘制了最大连通子图由 $T15$ 到 $T16$ 的演化过程中，不同度(k)节点新增连边概率($\kappa(k)$)的变化情况。从图中看出，在双对数坐标下，节点新增连边概率与其度值在总体上符合幂指数

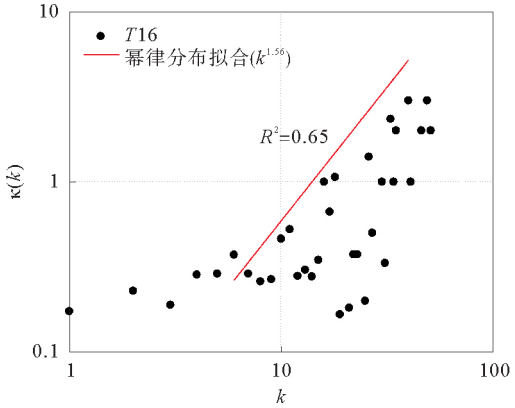


图 11　最大连通子图中节点度(k)与新增边概率($\kappa(k)$)的关系

Fig.11　Relations between node degree (k) and the probability of adding edge ($\kappa(k)$) of the giant component

为 1.56 的幂律分布(红色实线),相应二者呈现出较为明显的正相关关系,即节点度越高,新增连边数量越多。因此,网络中新的协作关系的建立在结构上具有倾向性连接的特点。

2)协作关系的属性特性分析

为了描述开发人员的技术背景,本文将每位开发者对不同子项目的提交情况作为一个向量,然后利用式(3)计算具有模块内(或模块间)连边的节点的平均属性相似度。

式(3)中,模块内外建立协作关系双方的平均属性相似度表示为 S。V_i 和 V_j 分别为开发者 i 和 j(双方拥有协作关系)的技术背景向量,r 为子项目编号,因而相应的,$V_{r,i}$ 则表示开发者 i 对子项目 r 的提交量,E 表示模块内(或模块间)协作关系的数量。

$$S=\frac{1}{E}\sum_{i\neq j}\frac{\sum_{r=0}^{40}V_{r,i}\times V_{r,j}}{\sqrt{\sum_{r=0}^{40}(V_{r,i})^2}\times\sqrt{\sum_{r=0}^{40}(V_{r,j})^2}} \tag{3}$$

图 12 绘制了 $T16$ 最大连通子图的模块内部及模块之间协作双方的平均属性相似度。图中,处于副对角线上的色块表示模块内协作双方的平均属性相似度,其它色块表示模块间协作双方的平均属性相似度,颜色深度与平均属性相似度数值正相关;带有斜线的色块表示对应的两个模块间不存在协作关系。

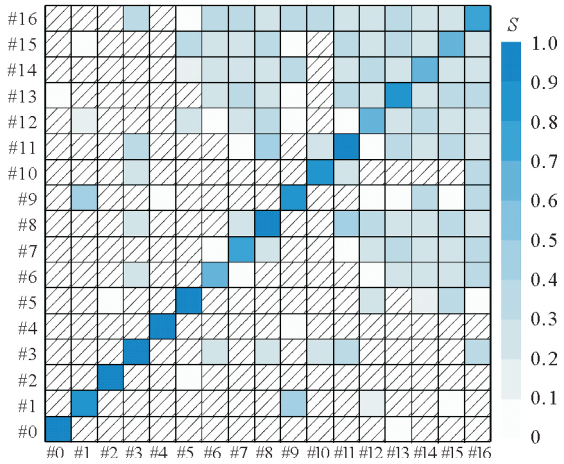


图 12 最大连通子图($T16$)模块内部及模块之间协作双方的平均属性相似度

Fig.12 Average attribute-similarity (S) of nodes with inner-module (inter-module) links ($T16$)

从图 12 可以观察到,副对角线上色块的数值较高($S\geqslant 0.6$),并且明显高于同行(列)其它色块的数值。例如,#4 和 #9 模块间存在协作关系,两个模块内 S 值接近于 0.9,而模块间 S 值不超过 0.5。这一结果意味着同一模块的开发者在技术背景上往往具有较高的相似性,而跨模块协作双方则存在一定的差异性。

综上,最大连通子图每个模块均由一定数量的边缘开发者($k<6$ 节点)围绕少数的核心开发者($k\geqslant 6$ 节点)构成,并且不同模块的核心开发者在一定程度上相互协作。由此,核心开发者在 Cloud Foundry 项目的开发过程中主要承担了两个角色,分别是模块内的中心角色和模块间的中介角色。与此同时,同模块的开发者往往表现出技术背景属性上的相似性,而不同模块的开发者间存在技术背景的差异性。因此,开发者协作行为表现出倾向性连接、同质相吸、差异偏好相结合的特征。

3 与已有相关工作的对比

在已有的相关研究中,Joblin 等[21]、以及夏昊翔等[2]的工作与本研究具有一定的相似之处,二者均聚焦于开源软件开发活动中由开发者自发协调而形成的协作网络的结构状态。但是,本研究与这两项工作具有实质上的不同。

Joblin 等收集了 18 个开源软件项目的提交记录,通过代码中函数的语义关系来描述开发者之间的协作关系并构建相应的协作网络。研究结果显示,这 18 个协作网络的规模介于 50 到 1 000 之间,其中最大规模项目涉及的代码量为 1.7×10^7 行。与此同时,这些网络均展现出了由松散连接状态逐渐发展为紧密连接状态的演化过程[16]。本文考查的 Cloud Foundry 社区开发者协作网络的规模(整个网络规模超过 2 500,最大连通子图规模接近 2 000)以及所涉及的代码量(6.86×10^7 行)明显高于上述网络。在结构演化方面,虽然 Cloud Foundry 社区的开发者协作网络的最大连通子图最初表现为松散连接状态,但是在随后的演化过程中并不是发展为单一的紧密连接状态,而是依次呈现出两种不同的结构特征(即链式结构和多模块的小世界状态)。由此可见,Cloud Foundry 社区的开发者协作网络与 Joblin 等所考察的网络具有不同的演化模式,这一现象可以通过软件规模的差异加以解释:Cloud Foundry 项目在规模上远高于与 Joblin 等工作中所考察的项目,相应会具有更高的功能复杂性,这不仅需要更多的开发者来实现软件功能,也需要开发者之间形成更加精细的分工合作。进而,大多数开发者围

绕特定的少数子项目展开提交工作，少数开发者对不同功能之间的开发工作进行协调（如图 5 和图 8 所示），促使协作网络呈现出相互连接的模块化结构。小规模的开源软件功能复杂性相对较低，开发者可以有足够的精力参与多个软件功能的开发工作，造成关注不同子项目的开发者之间协作关系紧密交织，继而协作网络宏观上表现为紧密连接的状态。

夏昊翔等利用开发者提交记录中的子父哈希码关系构建 OpenStack 云计算项目社区的开发者协作网络，通过对其静态结构及网络社区（模块）的演化的分析，发现网络的最大连通子图会呈现出多模块的“核心—边缘”结构，并且模块的演化（涌现、发展、合并）与子项目内在关联[2]。本文所考察的 Cloud Foundry 社区开发者协作网络在规模和项目所属领域上与夏昊翔等的研究工作是非常接近的，同时在网络静态结构上也得到了相似的结论，这也进一步说明大型开源软件社区可能具有共性的协作模式。除此之外，我们进一步发现最大连通子图中的开发者与其维护的子项目所构成的二分网络具有很强的嵌套性，这一结构一定程度上解释了在开发者自愿加入或退出开源社区的情形下，Cloud Foundry 社区依然能很好地保持项目开发的延续性。在网络社区演化方面，本研究与夏昊翔等的结论是基本一致的。但是，与 OpenStack 项目协作网络中的社区相比，Cloud Foundry 项目协作网络中的社区所关注的子项目更多。与此同时，OpenStack 项目协作网络中，两个关注完全不同子项目的社区往往会进行合并，而在 Cloud Foundry 项目协作网络中我们也并未发现这一现象。出现上述不同的原因可能是由于两个项目在云计算服务中的定位不同：与 OpenStack 的基础设施服务（IaaS，Infrastructure as a Service）相比，Cloud Foundry 所提供的平台服务（PaaS，Platform as a Service）往往需要不同方面服务的复杂交互（如用户在部署应用权限和运行环境的交互），相应关注功能关联的子项目的开发者之间会形成紧密协作，进而通过这些协作关系构成网络社区会涉及相对较多的子项目。另一方面，OpenStack 提供的服务更加侧重于云计算的基础架构，相应子项目的功能划分上更加独立，进而基础架构的变化会导致关注不同项目网络的社区的合并。Cloud Foundry 平台服务处于云计算架构的上层，使用户在部署自身应用时无需考虑计算、存储等资源的分配，因而网络社区会在软件功能上形成划分，相应只有所关注软件服务功能相近（如用户应用的部署及配置服务）的网络社区会发生合并。

4　结论

本文以 Cloud Foundry 开源软件社区为例，考察了开发者协作网络的结构与演化模式。研究结果显示，一个规模远大于其他子图的最大连通子图逐渐在网络中涌现，并且相应的开发人员承担了整个项目的绝大多数开发工作。通过进一步分析最大连通子图的结构演化，我们发现其呈现出与子项目（即软件功能）内在关联的阶段性演化过程，主要结论可以总结为以下两点。

首先，最大连通子图由最初的“松散连接”状态，逐渐形成“链式”结构，最终演化为具有“核心—边缘”结构的多模块小世界状态。在这一过程中，最大连通子图上模块的涌现、发展和合并均较好地保持了所维护的子项目的聚焦性和延续性。

其次，最大连通子图呈现多模块小世界特征时，模块内协作关系具有同质相吸和倾向性连接相结合的特点；模块间的协作关系具有差异偏好的特点，并且这种偏好的出现与开发者现有合作者的数量（即节点的度）密切相关。

通过上述结果不难看出，Cloud Foundry 开源软件社区并不是现有研究（如文献[4]）所提出的扁平化、自由流动的组织模式，而是自发形成了一种高度模块化的网络型组织模式，这为后续的相关研究工作提供了一定的借鉴。同时，开源软件开发作为一种具体的开放式创新活动，上述研究结果亦是对开放式创新相关研究的丰富。在后续研究中，笔者会针对不同领域的开源项目（如 Tensorflow、AngularJS 等）展开分析，以检验本文结论是否具有普遍性。此外，笔者将将结合协作关系的特点开展模型化研究，继而从复杂网络动力学视角进一步探查上述演化模式背后的动力学机制。

参考文献：

[1] 何鹏，李兵，杨习辉，等. 开源软件社区开发者偏好合作行为研究[J]. 计算机科学，2015，42(2)：161-166.

He Peng, Li Bing, Yang Xihui, et al. Research on developer preferential collaboration in open-source software community[J]. Computer Science, 2015, 42(2): 161-166.

[2] 夏昊翔, 张潇, 张醒洲. OpenStack 开源软件开发者协作网络分析[J]. 系统工程理论与实践, 2017, 37(5): 1373-1382.
Xia Haoxiang, Zhang Xiao, Zhang Xingzhou. Study on collaborative network of Open Stack OSS developers[J]. Systems Engineering-Theory & practice, 2017, 37(5): 1373-1382.

[3] Gencer M, Oba B.Taming of "Openness" in software innovation systems[J].International Journal of Innovation in the Digital Economy, 2017,8(2):1-15.

[4] Aljemabi M A, Wang Z. Empirical study on the evolution of developer social networks[J]. IEEE Access, 2018, 6: 51049-51060.

[5] Raymond E. The cathedral and the bazaar[J]. Knowledge, Technology & Policy, 1999, 12(3): 23-49.

[6] Shah S K. Motivation, governance, and the viability of hybrid forms in open source software development[J]. Management Science, 2006, 52(7): 1000-1014.

[7] De Laat P B.Governance of open source software: state of the art[J]. Journal of Management & Governance, 2007, 11(2): 165-177.

[8] Wu L, Wang D, Evans J A. Large teams develop and small teams disrupt science and technology[J]. Nature, 2019, 566(7744): 378.

[9] Behfar S K, Turkina E, Burger-Helmchen T.Knowledge management in OSS communities: Relationship between dense and sparse network structures[J]. International Journal of Information Management, 2018, 38(1): 167-174.

[10] Hong Q, Kim S, Cheung S C, et al. Understanding a developer social network and its evolution[C]// 2011 27th IEEE international conference on software maintenance (ICSM), IEEE, 2011: 323-332.

[11] Yang J, Li H, Liao H, et al. Localization of information on communication networks of an open-source online community[J]. International Journal of Modern Physics C, 2017, 28(7): 1750091.

[12] 叶培根, 毛建华, 刘学锋. 基于大数据的 GitHub 开源社区开源项目量化分析[J]. 电子测量技术, 2017, 40(8): 84-89.
Ye Peigen, Mao Jianhua, Liu Xuefeng. Quantitative analysis of open source project in GitHub community based on big data[J]. Electronic Measurement Technology, 2017, 40(8): 84-89.

[13] Bird C, Pattison D, D'Souza R, et al. Latent social structure in open source projects[C]// Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering, ACM, 2008: 24-35.

[14] Singh P V. The small-world effect: the influence of macro-level properties of developer collaboration networks on open-source project success[J]. ACM Transactions on Software Engineering & Methodology, 2010, 20(2): 1-27.

[15] Palazzi M J, Cabot J, Izquierdo J L C, et al.Online division of labour: emergent structures in open source software[J]. Scientific Reports, 2019, 9(1): 1-11.

[16] Crowston K, Shamshurin I.Core-peripherycommunication and the success of free/libre open source software projects[J]. Journal of Internet Services and Applications, 2017, 8(1): 10.

[17] Geldenhuys J. Finding the core developers[C]// 2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE, 2010: 447-450.

[18] Barcomb A, Kaufmann A, Riehle D, et al.Uncovering the periphery: a qualitative survey of episodic volunteering in free/libre and open source software communities[J]. IEEE Transactions on Software Engineering, 2018.

[19] Wei K, Crowston K, Eseryel U Y, et al. Roles and politeness behavior in community-based free/libre open source software development[J]. Information & Management, 2017, 54(5): 573-582.

[20] Hu D, Zhao J L. Discovering determinants of project participation in an open source social network[C]// ICIS 2009 Proceedings, 2009: 16.

[21] Joblin M, Apel S, Mauerer W. Evolutionary trends of developer coordination: a network approach[J]. Empirical Software Engineering, 2017, 22(4): 2050-2094.

[22] Kavaler D, Filkov V. Stochastic actor-oriented modeling for studying homophily and social influence in OSS projects[J]. Empirical Software Engineering, 2016, 22(1): 1-29.

[23] 汪文娟, 李兵, 何鹏. 开源软件社区开发者角色的演化分析[J].复杂系统与复杂性科学, 2015, 12(1): 1-7.
Wang Wenjuan, Li Bing, He Peng. An analysis of the evolution of developers' role in open source software community[J].

Complex Systems and Complexity Science, 2015, 12(1): 1-7.

[24] Cheng J, Guo J L C.Activity-based analysis of open source software contributors: roles and dynamics[C]// Proceedings of the 12th International Workshop on Cooperative and Human Aspects of Software Engineering, IEEE Press, 2019: 11-18.

[25] Blondel V D, Guillaume J L, Lambiotte R, et al. Fast unfolding of communities in large networks[J]. Journal of Statistical Mechanics: Theory and Experiment, 2008(10): P10008.

[26] Almeida-Neto M, Ulrich W. A straightforward computational approach for measuring nestedness using quantitative matrices [J]. Environmental Modelling & Software, 2011, 26(2): 173-178.

[27] Guimera R, Amaral L A N. Cartography of complex networks: modules and universal roles[J]. Journal of Statistical mechanics: Theory and Experiment, 2005(2): P02001.

(责任编辑　耿金花)

(上接第 30 页)

[8] Du M, Meng D. Consensus of multi-agent systems with antagonistic interactions and communication delays[C]// Control Conference. IEEE, 2014:1075-1080.

[9] Li P, Liu Y, Zhao Y, et al. Consensus control for second-order multi-agent with time-delay on antagonistic networks[C]// Control Conference. IEEE, 2016:8066-8071.

[10] Tian L, Ji Z, Hou T. Bipartite consensus for a class of double-order multi-agent systems with time-varying delays on antagonistic interactions[C]// Chinese Automation Congress. IEEE, 2018:184-188.

[11] Jiang Y, Zhang H W, Chen J. Sign-consensus of linear multi-agent systems over signed directed graphs[J].IEEE Transactions on Industrial Electronics,2017,64(6):5075-5083.

[12] Tian Y P, Liu C L. Consensus of multi-agent systems with diverse input and communication delays[J]. IEEE Transactions on Automatic Control, 2008, 53(9):2122-2128.

[13] Lee D, Spong M W. Agreement with non-uniform information delays[C]// American Control Conference. IEEE, 2006:756-761.

[14] Ni W, Cheng D. Leader-following Consensus of multi-agent systems under fixed and switching topologies[J]. Systems Control Letters, 2010, 59(3-4):209-217.

[15] Hong Y, Gao L, Cheng D, et al. Lyapunov-based approach to multiagent systems with switching jointly connected interconnection[J]. IEEE Transactions on Automatic Control, 2007, 52(5):0-948.

[16] **俞立. 鲁棒控制: 线性矩阵不等式处理方法**[M]. **北京:清华大学出版社**, 2002.

[17] Horn R A, Johnson C R. Matrix analysis[M]. Cambridge: Cambridge University Press, 1985.

[18] **吴敏. 时滞系统鲁棒控制**[M]. **北京:科学出版社**, 2008.

(责任编辑　耿金花)