基于多粒度软件网络模型的软件系统演化分析

何 鹏[1],王 鹏[1],李 兵[2],胡思文[2]
(1. 湖北大学计算机与信息工程学院,湖北武汉 430062; 2. 武汉大学国际软件学院,湖北武汉 430079)

摘 要: 软件系统是一类典型的人工参与的复杂系统,理解软件系统的演化规律有助于更好地指导软件工程实践. 本文从包、类和特征三个粒度上构建软件系统的网络模型,利用复杂网络理论依次从网络规模、质量、结构控制三方面定量分析软件系统演化规律. 以经典的 Lehman 演化定律为基准,对比软件系统在不同粒度下的演化差异. 研究结果表明:(1)不同粒度下软件系统表现出的演化特性有所不同,其中在类粒度下效果更好;(2)持续增长、持续变化、自我调节和主体维持四项定律表现出与构建软件网络的粒度无关;(3)包粒度下系统演化对软件质量影响不大,而反馈系统定律仅在类粒度下成立.

关键词: 软件演化; 软件网络; 复杂网络; 复杂系统
中图分类号: TP301 **文献标识码:** A **文章编号:** 0372-2112 (2018)02-0257-11
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2018.02.001

An Evolution Analysis of Software System Based on Multi-granularity Software Network

HE Peng[1], WANG Peng[1], LI Bing[2], HU Si-wen[2]
(1. *School of Computer Science and Information Engineering, Hubei University, Wuhan, Hubei* 430062, *China*;
2. *International School of Software, Wuhan University, Wuhan, Hubei* 430079, *China*)

Abstract: Software as a man-made system is a typical complex system, understanding its evolution contributes to better software engineering practice. In this paper, we construct software network model from a multi-granularity perspective, namely the level of package, class and feature respectively. Then we analyze the evolutions of three open-source software systems in terms of network scale, quality and structure control indicators, using complex network theory. Finally, taking Lehman's evolution laws as the benchmarks, we compare the evolution of software networks based on multi-granularity. The results show that: (1) the evolution characteristics are varied under different granularity levels, and software network built in the level of class supports the most Lehman laws; (2) the laws of continuing growth, increasing complexity, self-regulation and conservation of familiarity are independent of the levels of granularity; (3) the impact of software evolution in the level of package on software quality is trivial, but feedback system is only supported in the case of class level.

Key words: software evolution; software network; complex networks; complex system

1 引言

在软件工程中,软件系统经历着"改进-保持-更新"的迭代生长节奏. 这一过程中表现出的特性对提高软件系统的鲁棒性和适应性提出了新的挑战,关系到软件生态系统的可持续发展,且与软件开发与维护成本控制息息相关. 众所周知,软件维护成本占总投入的70%以上. 因此,探索软件的演化特性有助于降低软件开发维护成本,但同样也极具挑战性.

早在20世纪70年代,Lehman 等人就指出软件演化过程具有持续性,并针对软件演化问题,提出了8项经典的软件演化定律(Lehman law)[1,2],分别从软件质量、规模、复杂度、开发效率以及开发过程等方面进行阐述(如表1). 然而,通过传统软件工程相关理论和工具,难以控制整个系统的复杂性,急需从全局的角度重新审视软件系统的内部复杂结构.

随着复杂网络相关研究的兴起,软件系统被抽象为网络形式,且被证实具有复杂网络的基本特性[3,4],

收稿日期:2016-10-24;修回日期:2016-12-26;责任编辑:蓝红杰
基金项目:国家973重点基础研究计划(No. 2014CB34040);国家自然科学基金(No. 61572371);湖北省知识创新专项(No. 2016CFB309)

表 1 软件系统演化的经典 Lehman 定律

	特性	描述
L1	持续改变 continuing change	随着需求的改变,软件系统在生命期内,持续更新迭代
L2	持续增长 continuing growth	随着软件系统的演化,软件规模呈现持续增长的趋势
L3	稳定更新 invariant work rate	软件系统的演化速率趋于保持稳定,即软件系统每个版本之间节点变化速率基本一致
L4	复杂度提升 increasing complexity	软件复杂度在演化过程当中,表现为持续上升
L5	质量折旧 declining quality	软件系统更新迭代过程中,系统总体质量呈逐步下降的趋势
L6	自我调节 self-regulation	软件系统的演化呈现一定程度的控制规律,不完全取决于需求变化
L7	主体维持 conservation of familiarity	软件系统的相邻版本变化差异率不会很高,主版本内的各个软件版本的更新频率趋于稳定.
L8	反馈系统 feedback system	软件系统相邻阶段的演化结果并非独立,上一阶段演化结果会作用于软件系统下一阶段演化特性

称之为软件网络. 在软件网络中,以软件元素(如方法、属性、类、接口、包)为节点,元素之间的关系为连边,构建软件的网络模型. 将软件系统抽象为软件网络有助于软件工程实践人员认识和理解软件整体结构特性与软件质量之间关系(结构决定功能),指导他们开发设计出高质量软件的实践原则,并为软件结构稳定性、复杂性度量、演化特性等提供新的量化指标和评价保证. 因此,通过分析软件网络结构的动态变化对了解软件的演化特性非常有帮助. 例如 Li 等人[5]从软件模块角度模拟软件网络的增长,顾等人[2]以类为节点分析软件网络的演化规律. 然而,当前大部分研究主要集中于单一粒度下的软件系统结构的动态分析,缺乏从不同粒度角度对比分析软件网络的演化规律. 潘等人[6]虽从包、类、方法三个粒度上分析了 Azure 软件演化情况,但并没有进一步对比评价不同粒度下的软件网络演化效果对软件实际演化特性的反映程度,且实验对象单一,结论可能不具一般性.

针对一个软件系统,从不同粒度的软件网络模型分析其演化特性是否会存在明显差异,以及采用哪种粒度的软件网络模型进行系统演化分析会更合适的问题,至今尚未有答案. 因此,本文在前人已有工作的基础上,综合考虑从多粒度视角抽取软件网络模型,并引入 Lehman 演化定律为参照,使用多个数据集进行软件网络的演化验证分析,从而丰富软件系统的演化研究.

2 相关工作

近年来,复杂网络的思想逐渐被引入软件工程领域,一些研究者在度量软件复杂性时,利用网络形式描述软件的拓扑结构,以软件元素(如方法、属性、类、接口、包等)为节点,它们之间的各种依赖关系[7]为连边,建立软件网络模型. 采用复杂网络演化分析方法研究软件网络的演化特性,从而理解软件系统的演化特性. 目前,构建软件网络模型的方式较多,包括考虑软件的静态代码结构[8]、软件的动态执行轨迹[9]、软件演化镜像图[1,2]等. 大量已有研究验证表明:软件网络同样具有“小世界”和“无标度”等复杂网络特性[10~15].

在使用软件网络研究软件系统演化特性的工作中,Myers 通过对比软件网络的度分布、聚类系数、路径长度等特性,提出了一种基于重构的软件演化模型[16]. Sole 和 Valverde 根据软件网络表现出来的复制(duplication)与分叉(divergence)现象,提出了一种新的软件演化模型[17]. 何克清等人从设计模式的角度提出了基于类角色间依赖关系的软件演化模型[18]. Jenkins 等人将有向网络理论应用在软件包的建模中,利用指标分析了一系列软件版本之间的稳定性[10,11]. Shi 等人在类粒度上研究了 5 个不同版本的 JDK 软件网络差异[12]. Wang 等人使用复杂网络理论研究了 233 个 Linux 内核版本模块的演化[13]. Li 等人发现软件系统服从某些演化规则,诸如节点之间的距离呈增长趋势以及网络表现出无标度特性[15].

近几年,Bhattacharya 等人[19]从方法(function)与模块(module)粒度分析了软件网络的演化,发现用网络度量指标不仅有利于理解软件系统的演化,还可用于评估软件缺陷和维护成本. 顾庆等人通过定义软件的类依赖网络,采用网络度量指标对 Lehman 的 8 项演化定律进行分析与验证,实验结果表明,软件网络的演化支持其中 4 项,而否定了复杂度提升、持续增长、以及质量折旧定律等 3 项[2]. 但他们的工作仅考虑了类粒度级别的软件网络. 一个软件系统可从不同粒度来抽取其网络模型,不同粒度下的软件网络的演化特性是否一致还是一个开放问题. 代等人基于进程代数相关理论,利用代数分析软件演化过程模型[20],在该理论框架下,依据有关代数理论分析软件演化过程有关特征[20,21],相关概念的提出有力地支持软件演化过程的形式化验证,这些研究从形式上分析了软件演化规律,但是缺乏实际的实证研究[6,22]. 总之,已有相关研究普遍仅停留在对软件系统节点的度分布、节点之间关系的分析,且多局限于单一粒度的软件网络模型[23].

本文从类、特征和包三个粒度分别构建软件系统的网络模型,结合复杂网络理论与软件工程实践,分析

对比开源软件系统连续版本演化过程存在的差异. 主要贡献归纳为:

(1)在已有研究工作的基础上,突破单一粒度和单一项目的限制,从包、类、特征三个粒度角度对多个软件系统进行演化分析;

(2)引入 Lehman 定律为参照,对比分析了三种粒度下软件网络在网络规模、网络质量和网络结构三个方面的演化差异,结果发现:不同粒度下的软件网络演化对 8 项演化定律的支持程度不同,其中持续增长、持续变化、自我调节和主体维持四项定律表现出与粒度无关.

3　软件演化

3.1　软件网络模型

将软件系统抽象为软件网络是一个逆向工程. 需先从源代码中解析出元素之间的依赖关系,然后可从不同粒度(包、类、特征等)上对依赖关系进行提取,生成网络可视化软件 UCINET 或 Pajek 支持的. net 网络文件格式[10,24],具体网络模型定义如下:

特征粒度软件网络 FSN(Feature-level Software Network),定义为一个有向网络 $\boldsymbol{G}_f=(\boldsymbol{V}_f,\boldsymbol{E}_f)$,其中节点 v_f ($v_f\in\boldsymbol{V}_f$)代表软件系统中的特征(变量或方法). 在 FSN 中,若特征 v_{fi} 与特征 v_{fj} 存在调用关系,则他们之间存在一条有向边 e_{ij}($e_{ij}\in\boldsymbol{E}_f$).

类粒度软件网络 CSN(Class-level Software Network),定义为一个有向网络 $\boldsymbol{G}_c=(\boldsymbol{V}_c,\boldsymbol{E}_c)$,其中节点 v_c ($v_c\in\boldsymbol{V}_c$)代表软件系统的类(Class)或者接口(Interface),如果两个节点之间存在依赖关系,则构成有向边 e_{ij}($e_{ij}\in\boldsymbol{E}_c$). 在 CSN 中,两节点的依赖关系主要考虑以下 3 种情况:

(1)继承,假如 v_{c1} 类继承于类 v_{c2},或实现接口 v_{c2},则存在有向边 $e_{12}=\langle v_{c1},v_{c2}\rangle$;

(2)聚合,假如类 v_{c1} 包含类 v_{c2} 的属性,则存在有向边 $e_{12}=\langle v_{c1},v_{c2}\rangle$;

(3)参数,假如类 v_{c1} 的方法调用了类 v_{c2} 的方法,则存在有向边 $e_{12}=\langle v_{c1},v_{c2}\rangle$.

包粒度软件网络 PSN(Package-level Software Network),定义为一个有向网络 $\boldsymbol{G}_p=(\boldsymbol{V}_p,\boldsymbol{E}_p)$,在 PSN 网络中,软件系统中每个包为节点 v_p($v_p\in\boldsymbol{V}_p$),每两个存在依赖关系的包之间构成一条有向边 e_{ij}($e_{ij}\in\boldsymbol{E}_p$). 两个包之间的关系间接来源于他们各自包含的类之间的依赖关系,即如果包 v_{p1} 的类调用了包 v_{p2} 中的某个类,那么存在有向边 $e_{12}=\langle v_{p1},v_{p2}\rangle$.

图 1 给出了 FSN、CSN 和 PSN 的一个简单示例. 软件系统中两个元素之间的关系可具有多重性,即连边的权重大于 1. 如图 1 中所示,类 W 的两个方法 e 和 f 均调用了类 V 的 d 方法,在 CSN 中,类 W 与 V 之间连边权重应为 2. 然而,考虑到本文主要从软件网络拓扑结构的变化上分析软件系统的演化,边有无权重不会影响网络结构,所以本文涉及的软件网络均为非加权的有向网络.

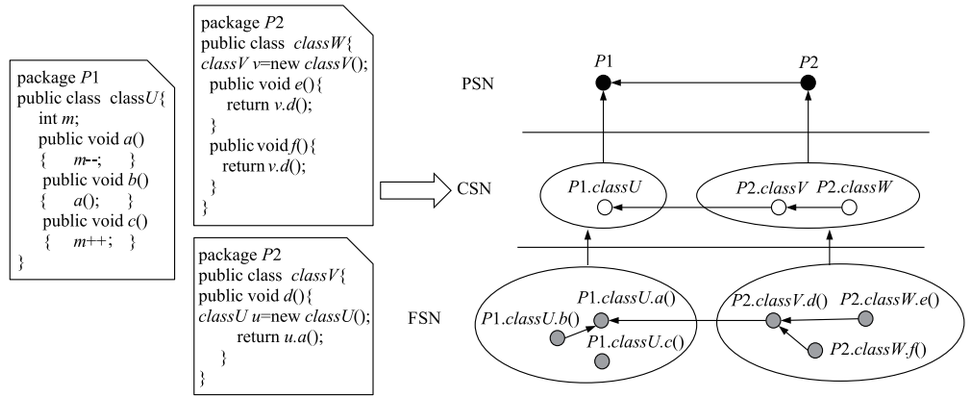


图1　构建FSN、CSN和PSN网络的一个简单示例图

3.2　度量指标

分析三种粒度下软件网络的演化情况并验证是否符合 Lehman 定律,需引入与各条定律相对应的度量指标(见表 2). 据我们了解,目前尚未形成统一的软件度量指标来定量评价 Lehman 定律[25]. 本次在度量指标的选取上综合考虑了以下因素:(1)参照 Lehman 等人提出的指标;(2)借鉴已有文献[2,26]中使用过的指标;(3)引入常用的网络指标[6]. 最后,本文从软件系统规模、质量与结构三个方面,分别考虑网络节点与连边数、更新率、复杂度、协调系数、模块度以及网络节点度分布等一系列度量指标. 各条定律相对应的度量指标描述如下:

L1:持续改变　持续改变指软件系统会通过动态调整来满足用户需求,对应表现为软件网络规模的变化. 通过分析不同版本下软件网络的节点数$|\boldsymbol{V}|$和有向连边$|\boldsymbol{E}|$的累积变化来验证不同粒度下软件系统的持

表 2 本文与 Lehman 定律相应的度量指标

	度量指标	分析角度
L1	节点数$\|\boldsymbol{V}\|$，连边数$\|\boldsymbol{E}\|$	网络规模
L2	Cum($\Delta\|\boldsymbol{V}\|$)，Cum($\Delta\|\boldsymbol{E}\|$)	网络规模
L3	节点更新率δ_v	网络规模
L4	平均度$\langle k\rangle$	网络质量
L5	入度指数α 出度均值μ	网络质量
L6	模块度 Q	网络结构
L7	平均最短路径长度 AvgP	网络结构
L8	协合系数 Asc	网络结构

续改变特性. 由于不同版本更新的时间跨度 DBR(Days Between Releases)可能有差异,使得持续改变程度不一样. 因此,我们进一步分析随版本的演化,软件系统持续改变程度是减弱还是加强.

L2:持续增长 软件系统在迭代更新过程中普遍呈现持续增长的规律,直观表现为软件网络中代码行 LOC、节点数$|\boldsymbol{V}|$和有向连边数$|\boldsymbol{E}|$的增长变化. 参照文献[29]中的方法,我们用不同粒度下的软件网络节点数$|\boldsymbol{V}|$和边数$|\boldsymbol{E}|$来定量分析软件系统的持续增长特性.

L3:稳定更新 稳定更新规定相邻版本之间软件系统内容保持稳定,软件系统的更新频率可通过定义软件网络中节点的更新率来加以体现,即对于相邻版本的两个软件网络$\boldsymbol{G}_i$和$\boldsymbol{G}_j(j=i+1)$,以后一个版本$\boldsymbol{G}_j$的网络节点总数$m=|\boldsymbol{V}_j|$作为基数,通过节点集做差运算去掉前后两个网络中未发生变化的那部分节点,得到变更节点数$n=|\boldsymbol{V}_j-\boldsymbol{V}_i|$,变更节点数$n$与后一个版本$\boldsymbol{G}_j$节点总数$m$的比值即为更新率$\delta_v$.

$$\delta_v=\frac{n}{m} \tag{1}$$

L4:复杂度提升 根据 L2 定律所述,除非及时采取一定的措施,否则软件系统的复杂性随时间增加[12]. 度量软件系统复杂性的指标较多,如 C-K 套件、MOOD 指标、过程指标等. 节点的平均度$\langle k\rangle$常用于刻画软件系统的整体复杂性,表现为软件系统内部依赖关系的一种全局趋势. 令变量degree_i为节点i的度(此处不区分出度和入度),$|\boldsymbol{V}|$为节点总数,其中节点度的计算为$degree_i=\sum|e_{\cdot i}|+\sum|e_{i\cdot}|$,而$\sum|e_{\cdot i}|$表示所有指向节点$i$的连边,$\sum|e_{i\cdot}|$表示所有由节点$i$出发指向的连边.

$$\langle k\rangle=\sum_{i=1}^{|V|}\frac{\text{degree}_i}{|V|} \tag{2}$$

L5:质量折旧 通常软件质量会随时间不断恶化,或老化(ageing)、技术债务(technical debt)堆积. 在软件工程领域,软件网络中节点的入度表示代码重用,出度代表耦合,一个节点入度越大表示这部分代码重用越频繁,出度越大表示对其他节点的依赖越高. 考察网络节点的度分布是复杂网络方法中常用的指标之一,度分布$P(k)$表示网络中度为k的节点比例,在有向网络中还区分入度分布$P^{\text{in}}(k)$和出度分布$P^{\text{out}}(k)$. 研究表明[26],软件网络的节点入度呈幂律分布,而出度满足对数正态分布. 因此,网络节点的入度分布函数$f_{\text{in}}(k)$如式(3)所示,其中c和k_0为常数,幂指数α可用于考察网络入度分布的相关特征.

$$f_{\text{in}}(k)=c(k+k_0)^{-\alpha},\ c>0,k_0=1 \tag{3}$$

出度分布的概率密度函数$f_{\text{out}}(k)$如式(4)所示. 对出度分布特征,本文采用对数均值μ进行描述.

$$f_{\text{out}}(k)=\frac{1}{\sqrt{2\pi}\delta(k+k_0)}\mathrm{e}^{-\frac{(\ln(k+k_0)-\mu)^2}{2\delta^2}},k_0=1 \tag{4}$$

软件系统开发遵循"高内聚,低耦合"原则,故软件网络表现为具有较大的入度幂指数α和较小的出度对数均值μ特性,这将有利于减少软件维护时所需修改代码的范围,既能降低软件维护成本,还可提高软件开发质量[2]. 通过分析软件网络中节点入度和出度的分布情况,可度量软件质量演化特性.

L6:自我调节 Lehman 定律表示软件系统演化是一个自我调节的过程. 模块度Q是用来度量一个网络内部节点模块化(也称集群、社区化)的程度[27]. 高度模块化的网络,模块内部节点连接密集,不同模块之间的节点连接相对稀疏. 可用式(5)计算:

$$Q=\frac{\sum_i^n e_{ii}-\sum_i^n a_ib_i}{1-\sum_i^n a_ib_i} \tag{5}$$

其中e_{ij}为模块i中节点与模块j中节点建立的连边与总连边数的比例,$a_i=\sum_j^n e_{ij}$、$b_i=\sum_j^n e_{ji}$分别为起点、终点在模块i中的连边比例,n为总的划分模块个数.

L7:主体维持 主体维持要求相邻版本之间软件系统的整体拓扑结构保持稳定,可引入平均最短路径长度(Average shortest path length)指标来验证这一定律. 最短路径长度是指网络中任意两个节点v_i和v_j之间的距离d_{ij},即连接两个节点所需历经的最短有向连边数. 网络的平均路径长度即为任意两个节点之间距

离的平均值,假设节点 v_i 的可达节点集为 S_i,则 AvgP 的计算可表示为:

$$\mathrm{AvgP}=\frac{1}{N}\sum_{i=1}^{N}\frac{\Sigma_{v_j\in S_i}d_{ij}}{|S_i|} \tag{6}$$

L8:反馈系统　该条定律强调上一阶段演化结果会对下一阶段演化起作用. 在复杂网络中,协合系数 Asc(Assortativity Coefficient)反映网络中直接相连的两个节点度的联合分布特征. 按照 Newman 提出的 Asc 定义[28],对于第 i 条有向边,假如用 j_i 和 k_i 分别表示该边的连出节点与指向节点的出度和入度,网络的协同系数 Asc 可用式(7)计算:

$$\mathrm{Asc}=\frac{\boldsymbol{M}^{-1}\Sigma_i j_i k_i-[\boldsymbol{M}^{-1}\Sigma_j\frac{1}{2}(j_i+k_i)]^2}{\boldsymbol{M}^{-1}\Sigma_i\frac{1}{2}(j_i^2+k_i^2)-[\boldsymbol{M}^{-1}\Sigma_j\frac{1}{2}(j_i+k_i)]^2} \tag{7}$$

其中 $\mathrm{Asc}\in[-1,1]$,若 $\mathrm{Asc}>0$ 表明网络节点度呈正相关,即度大的节点倾向于与其他度大的节点相连;反之,$\mathrm{Asc}<0$ 表明度大的节点更可能与度小的节点相连.

4　实验分析

4.1　数据

本文以三个 Java 开源软件 Jung、Maven、Azure 为研究对象,选择依据为:(1)开源软件源代码易于获取,便于抽取各种粒度的软件网络模型;(2)它们都是较成功的开源软件,发布版本(releases)均在 20 个以上,为本文研究软件系统的演化提供充足的数据基础;(3)选择 Java 语言的软件,是考虑到我们用于抽取依赖关系的工具暂时只支持 Java 代码.

研究软件系统演化,Lehman 等人建议每个软件最好分析 10 个以上版本. 因此,文章获取 Jung 项目从版本 1.0.0 到 1.7.6 的 21 个版本,Azure 从版本 3.0.0.8 到版本 4.5.0.2 的25 个版本,Maven 从版本 2.0.0 到版本 3.2.2 的 25 个版本. 表 3 给出了三个开源软件首尾两个版本对应的三类软件网络的统计信息,包括网络中节点数 $|\boldsymbol{V}|$ 与边数 $|\boldsymbol{E}|$.

表 3　本文实验所使用的 3 个软件系统的统计信息

项目	第一个版本 Id = 1						最后一个版本 Id = l						Versions
	FSN		CSN		PSN		FSN		CSN		PSN		
	$\|V_{fl}\|$	$\|E_{fl}\|$	$\|V_{cl}\|$	$\|E_{cl}\|$	$\|V_{pl}\|$	$\|E_{pl}\|$	$\|V_{fl}\|$	$\|E_{fl}\|$	$\|V_{cl}\|$	$\|E_{cl}\|$	$\|V_{pl}\|$	$\|E_{pl}\|$	
Jung	833	1227	198	93	28	274	3121	4581	494	485	43	394	21
Maven	7803	15754	706	2618	109	561	42613	79044	4736	17332	299	1612	25
Azure	25119	46753	4945	26827	417	2999	39711	76618	5353	43985	476	4461	25

4.2　实验设计

图 2 为实验整体框架,由五个部分组成:(1)从互联网上下载开源软件不同版本的源代码文件;(2)利用我们自行开发的 MSN Analyzer 工具解析代码文件获取不同粒度下元素之间的依赖关系;(3)生成. net 格式的网络文件,形成三种粒度的软件网络;(4)利用引入的度量指标对 FSN、CSN、PSN 进行演化分析.

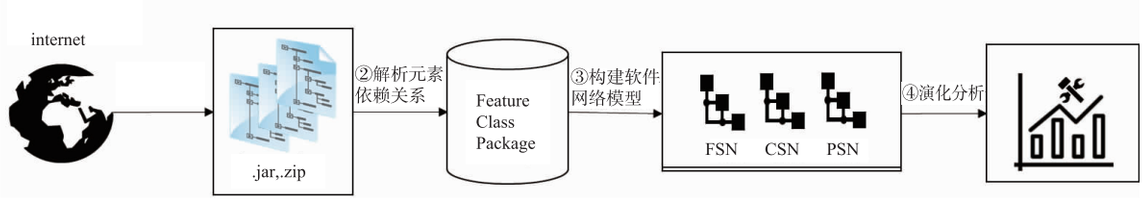


图2　实验整体框架

4.3　实验结果

4.3.1　网络规模的演化

分析 CSN、FSN 和 PSN 网络节点和边规模 $|V|$、$|E|$,以及累积变化(Cum($\Delta|V|$),Cum($\Delta|E|$)和节点更新变化率 δ_v 的演化. 针对三个目标软件系统,以上 5 个指标随软件版本演化的变化趋势如图 3 所示(图中 x 轴为软件版本对应的序号 Id,为了表示方便,对三个目标软件系统的 $|V|$ 和 $|E|$ 结果数值分别进行了适当比例的缩放,使其便于在一幅图中绘制,但并不影响演化趋势).

三个软件系统在不同粒度的软件网络中,节点数和有向边数都不断变化,且呈逐渐增长趋势,满足 Lehman 的持续增长定律. 根据实验结果不难发现,$|V|$ 和 $|E|$ 的增长具有比较明显的阶段性,结论大体与文献[2]中的一致. 各跳转处与软件系统不同时间段的主版本相匹配,如 Azure 在版本序号 6、8、16 处分别对应项

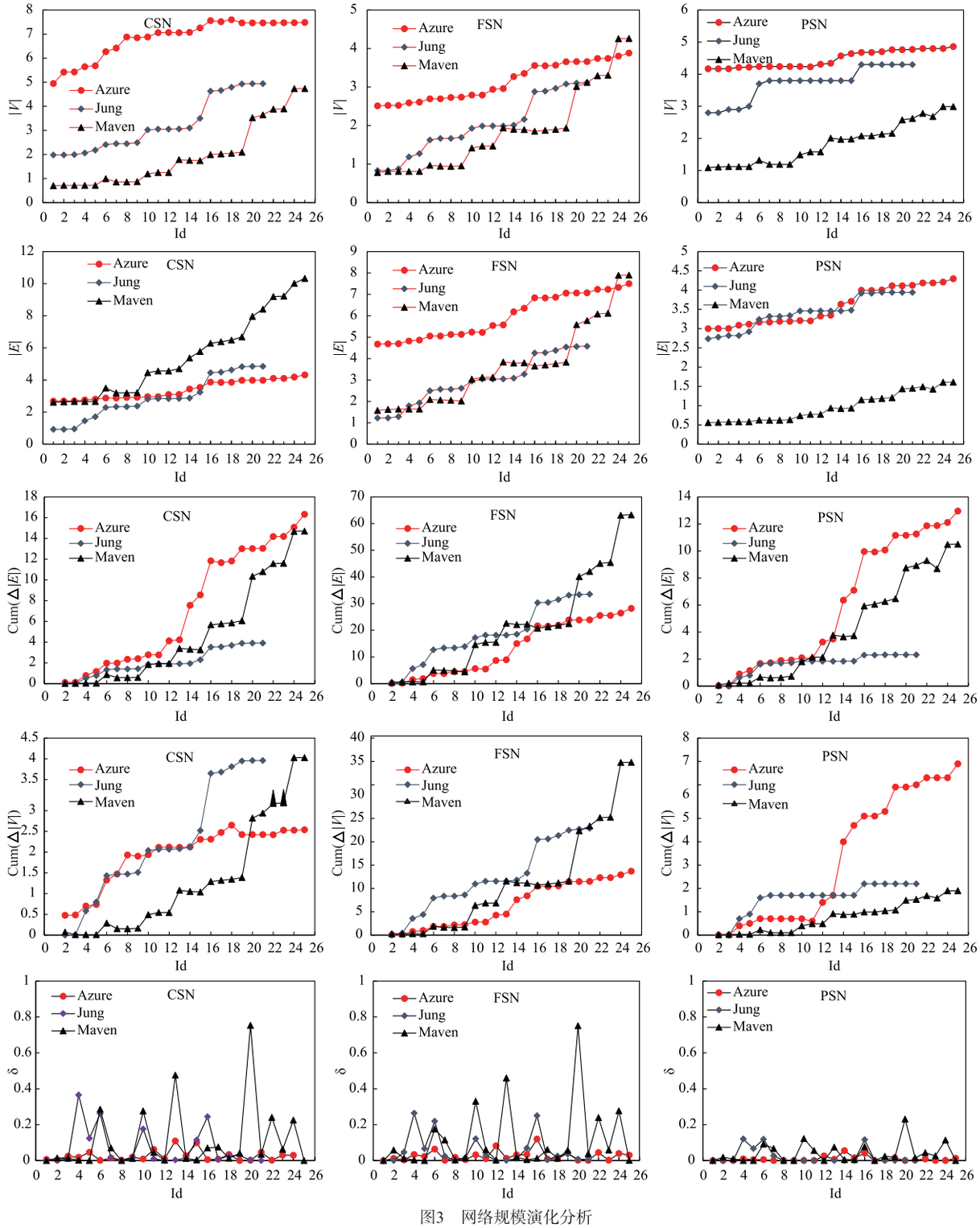


图3 网络规模演化分析

目的主版本号3.0、3.2、3.4;Jung中版本序号3、6、9、15分别对应主版本号1.0、1.1、1.2、1.6.此外,PSN中|V|和|E|结果的整体增长相比其他两类网络更平缓,其原因应该是软件演化过程中,开发者不会轻易对高层次的包结构做大幅度的调整.

根据图中节点的累积变化Cum(Δ|V|)和连边的累积变化Cum(Δ|E|),结果显示三个粒度下整体上依旧保持阶段性持续增长的趋势,也即验证了系统的持续

改变演化特性. 虽然在 PSN 中,项目 Jung 的改变较为平缓,但不难看出,系统随版本的增长呈周期性改变的趋势.

Lehman 定律指出软件系统版本节点的演化速率将趋于保持稳定,图 3 结果还显示包粒度下三个软件的更新率 δ_v 相对稳定,新增节点大体控制在原版本规模的 0.2 倍以内,Azure 甚至更小;然而,在类和特征粒度下的 Jung 和 Maven 的值都呈现较大幅度的波动,尤其是 Maven 软件在第 20 个版本时变化特别大. 不难发现,Azure 在类和特征粒度下的 δ_v 值仍比较稳定,可能原因是 Azure 相比其他两个软件历经的版本更多且项目相对更成熟,项目的规模增长也较为平缓,所以在持续地更新过程中,相比已有规模并无明显大的变动,表现出稳定更新的演化特性.

4.3.2　网络质量的演化

软件开发保持“高内聚,低耦合”至关重要,也即模块(包)之间关联尽可能小,而模块内部(类和特征)之间关系紧密. 依据 Lehman 等人对复杂度提升定律的定义,随版本的演化平均度$\langle k\rangle$值应呈上升趋势. 图 4 结果显示仅 CSN 和 FSN 网络的$\langle k\rangle$值有增大趋势,而在 PSN 中$\langle k\rangle$值整体上保持相对稳定,甚至有降低的趋势. 结果说明软件系统升级完善过程,更新维护工作主要在于改变软件内部类或特征之间的依赖关系,对于粗粒度的包之间依赖关系影响不大. 在 CSN 中 Maven 软件从第 15 个版本到第 20 个版本中间$\langle k\rangle$值有一较大幅度的增长,之后趋于稳定;在 FSN 中 Jung 软件从第 4 个版本到第 10 个版本也同样如此. 根据目标软件网站提供的信息显示,$\langle k\rangle$值的增长与软件功能调整有关,比如 JUNG 在 2004 年上半年这段时间内对原有版本的旧文档库进行了清理.

从依赖关系的方向性评价系统质量,表现为入度幂指数 α 越大越好,而出度对数均值 μ 越小越好. 实验结果显示 Jung 和 Maven 软件的 CSN 和 FSN 中 α 值均表现出下降趋势,而 μ 值有变大趋势,说明开发者在完善软件系统时,新的改动会对软件质量带来负面影响,正如常说的修改一个缺陷有可能会引发更多的缺陷. 然而,对于粗粒度的 PSN 网络,三个软件的 α 值和 μ 值均相对稳定,说明演化过程中,开发者对软件系统顶层架构不会做太大的调整,尤其是像 Azure 这类规模较大且相对成熟的软件.

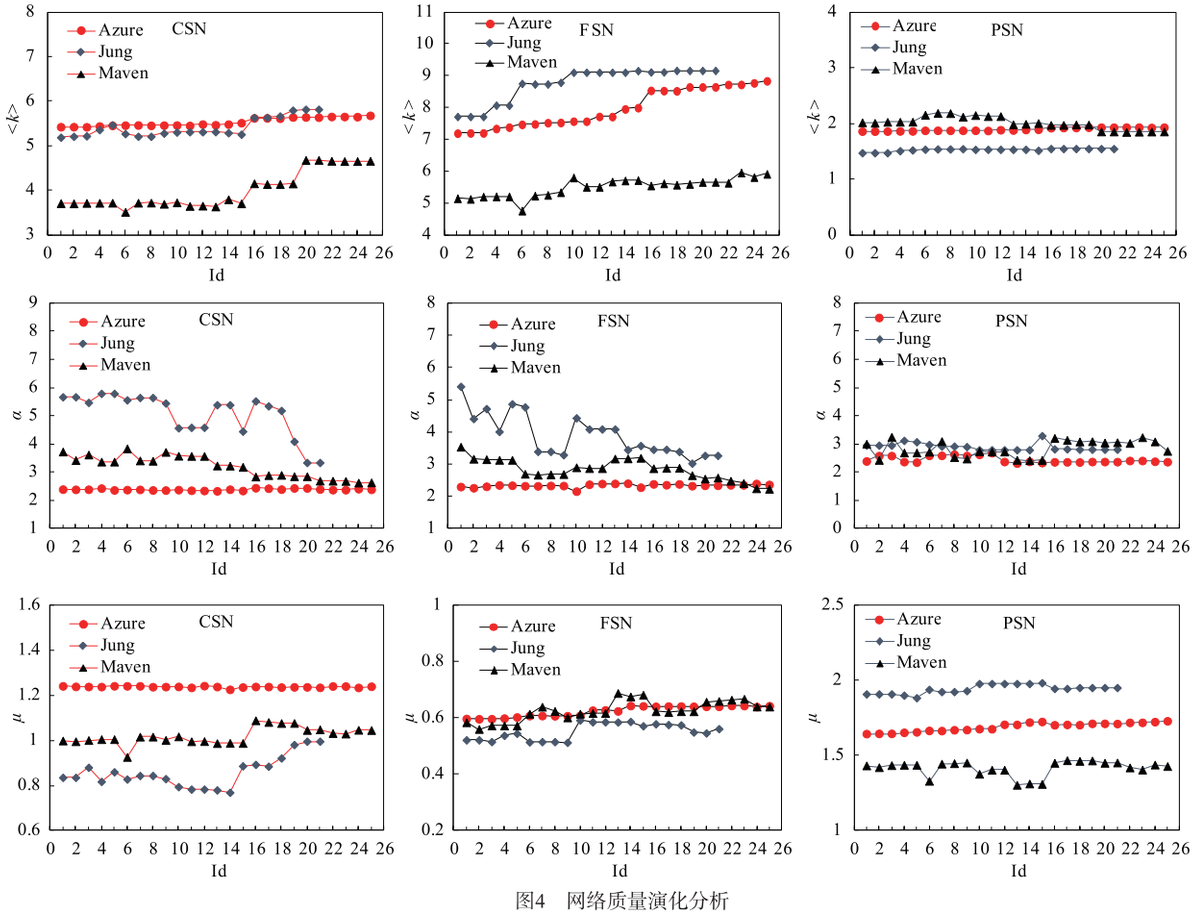


图4　网络质量演化分析

因此,随着开发者对软件系统的不断完善,系统高层次结构质量相对稳定,但在类和特征粒度下软件复杂度不断提升,且表现出质量折旧特性.

4.3.3 网络结构的演化

随软件版本的演化,软件系统结构预期将得到优化. 基于 CSN、FSN 和 PSN 网络,采用网络模块度 Q、平均最短路径 AvgP 和协合系数 Asc 度量指标进行演化分析,演化结果如图 5 所示. 三种粒度下的网络模块度 Q 均有逐渐增长趋势,表明网络的抱团结构越来越明显,即通过自我调节软件系统的模块化效果不断增强. 不同粒度下项目 Azure 和 Jung 的 AvgP 值保持相对稳定,而 Maven 的 AvgP 值在第 16 个版本后发生跳涨,随后版本中仍保持相对稳定. Maven 的这种跳涨可能是因项目在该版本中做了较大幅度的修改,与图 3 中 δ_v 波动较大的原因类似.

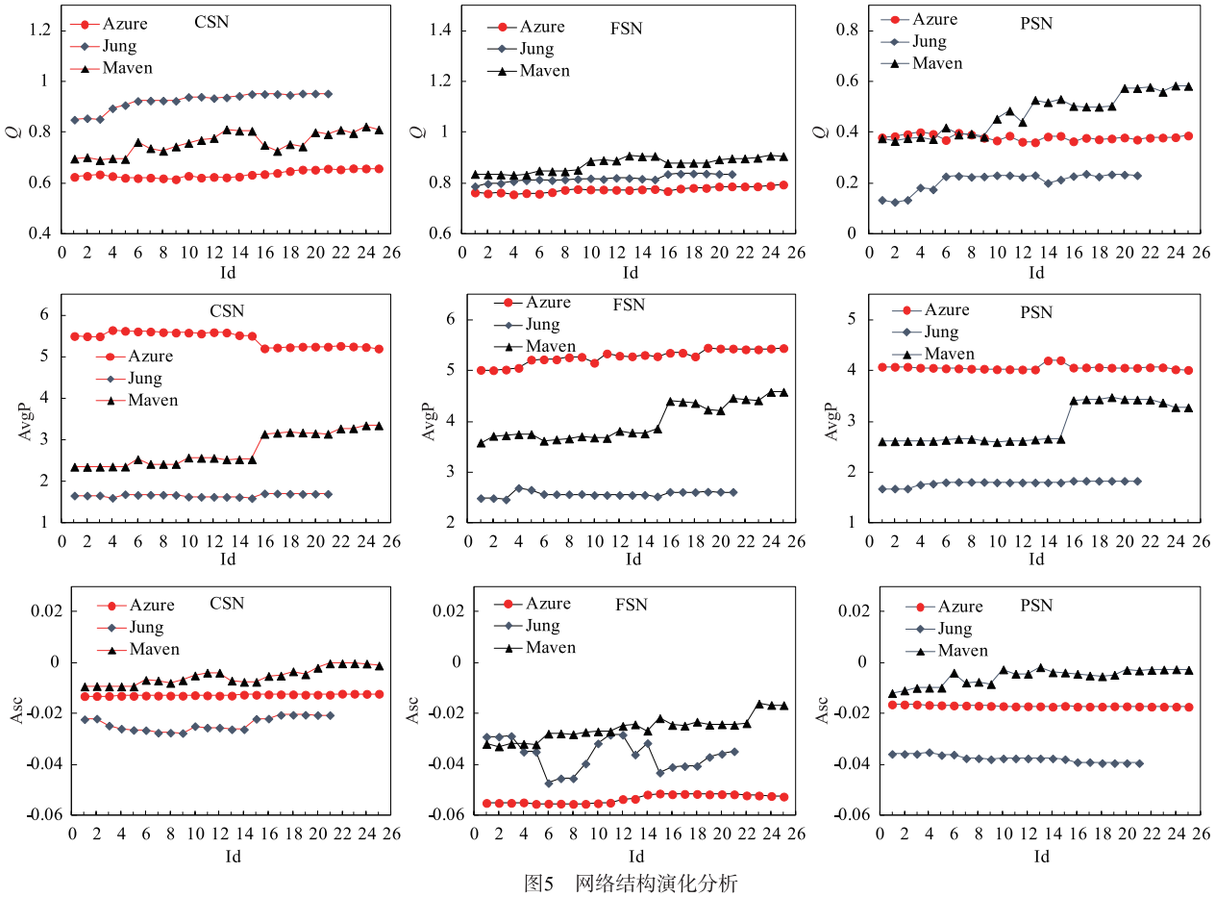


图5 网络结构演化分析

软件网络作为一类技术网络,节点度之间表现为异配性(Asc <0),即度大的节点倾向于与度小的节点连接. 从图中 Asc 值的变化趋势可知,在 CSN 中异配性有不断缩小的趋势且逐渐接近 0,FSN 和 PSN 中不同软件变化有差异. 另外,三种粒度下 Azure 系统节点度在异配性上变化轻微,而 Maven 系统在三种粒度下 Asc 值均有变大趋势,这种变化趋势说明软件开发人员在不断改善软件的内部结构,降低核心节点的复杂度,避免系统内部出现星型结构,体现出反馈系统特性. 然而,Jung 系统在三种粒度下 Asc 值的变化趋势截然不同,在 CSN 种呈增长趋势,FSN 中不断波动,而 PSN 中却表现出下降趋势.

因此,整体上三种粒度下软件系统的演化支持 Lehman 的自我调节、主体维持定律,但对反馈系统的支持仅在类粒度下略显成效.

4.4 讨论

综合以上实验分析结果,我们进一步对软件系统在不同粒度下的演化特性进行归纳,表 4 汇总了包、类、特征三个粒度下软件网络的演化对 Lehman 定律的支持情况. 具体概括为:

(1)从包粒度视角,软件系统的演化在网络质量上不满足 Lehman 的复杂度提升与质量折旧两项定律,在反馈系统方面具有不确定性,其他 5 项都满足. 软件系统中包对应为功能模块,一般较为成熟的系统在持续演化过程中多以功能完善为主,更新操作着重于细粒度的类或特征上,新的改动可能会影响包之间依赖关系,但主要体现在它们之间的连边权重,并不会影响软

表 4　三种粒度下的软件系统演化分析情况

Lehman 演化定律	粒度级别	实验结果	是否支持
持续变化	包	在软件版本的更新过程中,包节点持续变化	Yes
	类	在软件版本的更新过程中,类节点持续变化	Yes
	特征	在软件版本的更新过程中,特征节点持续变化	Yes
持续增长	包	在软件版本的更新过程中,网络连边数不断扩大	Yes
	类	在软件版本的发布更新过程中,网络连边数不断扩大	Yes
	特征	在软件版本的发布更新过程中,网络连边数不断扩大	Yes
稳定更新	包	三个项目的值保持相对稳定,更新率低于 0.2	Yes
	类	Azure 保持稳定的值,其他两个项目(Jung 和 Maven)都未能	Unknown
	特征	Azure 保持稳定的值,其他两个项目(Jung 和 Maven)都未能	Unknown
复杂度提升	包	平均度 k 保持相对稳定,复杂度没明显提升现象	No
	类	平均度 k 有不断增长的趋势	Yes
	特征	平均度 k 有不断增长的趋势	Yes
质量折旧	包	质量指标 α 和 μ 的演化趋势,表明版本演化不会影响软件系统的质量	No
	类	质量指标 α 和 μ 的演化趋势,表明版本演化会降低软件系统的质量	Yes
	特征	质量指标 α 和 μ 的演化趋势,表明版本演化会降低软件系统的质量	Yes
自我调节	包	模块度 Q 指标值不断增大,表明软件系统演化过程是一个自我调节的过程	Yes
	类	模块度 Q 指标值不断增大,表明软件系统演化过程是一个自我调节的过程	Yes
	特征	模块度 Q 指标值不断增大,表明软件系统演化过程是一个自我调节的过程	Yes
主体维持	包	平均路径长度 AvgP 值在演化过程中整体保持稳定,软件系统在主版本内主体特性基本不变	Yes
	类	平均路径长度 AvgP 值在演化过程中整体保持稳定,软件系统在主版本内主体特性基本不变	Yes
	特征	平均路径长度 AvgP 值在演化过程中整体保持稳定,软件系统在主版本内主体特性基本不变	Yes
反馈系统	包	节点度匹配系数 Asc 的值有增有减,无法确认是否支持反馈系统	Unknown
	类	节点度匹配系数 Asc 有变大趋势,支持反馈系统	Yes
	特征	部分节点度匹配系数 Asc 的值增加,无法确认是否支持反馈系统	Unknown

件系统的整体拓扑结构. 本文的网络模型为有向非加权网络,用于度量网络质量的指标为节点度. 所以,包粒度下软件网络在质量上的演化影响不明显.

(2)在类和特征粒度下,软件系统的演化情况大体一致,主要的不同体现在反馈系统定律上. 随版本的演化,类节点之间差异会不断变小,特征节点之间的匹配关系对于小规模软件(如 Jung)变化不断波动. 反馈系统定律仅在类粒度下成立,一种可能的解释是类相比特征和包,更能体现面向对象程序设计思想,上一版本中类的变化更容易对下一版本造成影响.

综上所述,不同粒度下软件系统演化特性有所不同,且对经典的 Lehman 定律的支持程度也有差异,整体上使用类粒度的软件网络分析软件系统的演化效果倾向于更好. 需要指出的是,目标软件系统的规模、成熟度对演化分析结果有一定的影响,如 Azure 软件系统的演化趋势相比其他两个软件系统都要更为稳定. 因此,在研究软件系统的演化时,对目标对象的选取要慎重.

4.5　MSNA 工具

为了协助上文多粒度环境下软件网络演化分析,我们也开发了一款软件网络度量工具,MSNA(Multi-Granularity Software Network Analyzer 简称 MSNA),如图 6 所示. 该工具主要包括三部分,首先是选择软件网络模型类型(粒度),默认情况下选择包粒度. 根据导入的软件项目编译后的源代码,支持包、类和特征不同粒度下元素之间的依赖关系解析,并生成不同的软件网络模型. 其次,左下角部分为对应选择粒度下的软件网络可视化窗口,以 Jung 1.6.0 为例,包粒度下抽取的软件网络模型如图 7 所示,工具中默认采用了自动化布局类中的 YifanHuLayout 布局. 最后,工具的右边部分分别从上到下为网络规模、网络质量、网络结构三类度量指标,其中网络规模中节点数与边数随网络模型的构建自动生成,其他指标通过点击对应的“运算”按钮进行计算,如图 8 所示.

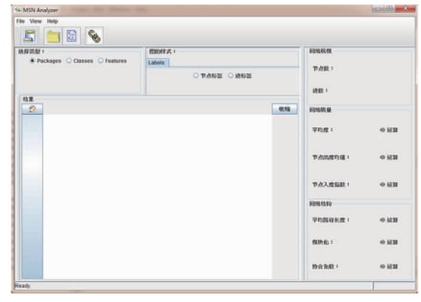


图6　MSNA操作界面

另外,工具中还附加了一个样式设置模块,该部分主要用于显示网络节点的标签和边的权重,由于文章中的软件网络暂时没有考虑边的权重,所有该部分目前主要以显示节点对应的类名为主,如图 9 所示.

图7 对应的PSN网络

图8 度量指标值

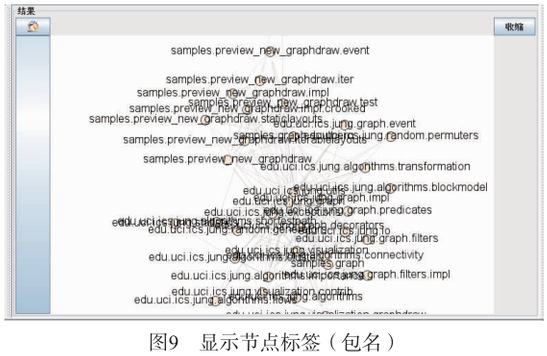


图9 显示节点标签（包名）

5 总结与展望

本文从多粒度角度对软件系统开展演化分析，引入 Lehman 的 8 项经典演化定律为参照，利用复杂网络相关理论，从网络规模、网络质量和网络结构方面进行定量分析. 研究结果表明：(1)从不同粒度构建网络模型分析软件系统的演化，效果有所不同，使用类粒度的软件网络分析效果更好；(2)持续增长、持续变化、自我调节和主体维持四项定律表现出与构建软件网络的粒度无关；(3)包粒度下系统演化对软件质量影响不大，而反馈系统仅在类粒度下成立. 从不同的视角理解软件拓扑结构和动态属性，有助于提高软件开发人员对软件演化过程进行自适调控，通过合理安排软件版本发布周期，减少软件维护成本，改善软件产品质量.

后续工作可从以下方面开展：(1)研究其他语言开发的软件系统，进一步验证实验结果的一般性；(2)构建有向加权的精细化软件网络模型，丰富已有软件演化规律结果；(3)引入开发者行为信息，从社会-技术网络的角度，分析软件的演化规律.

参考文献

[1] Lehman M M, Ramil J F, Wernick P D, et al. Metrics and laws of software evolution-the nineties view[A]. Proceedings of International Software Metrics Symposium[C]. Albuquerque: IEEE Press, 1997. 20 – 32.

[2] 顾庆，陈道蓄. 基于软件网络的软件系统演化规律验证和模拟[J]. 中国科学：信息科学，2014，44(1)：20 – 36.
Gu Q, Chen D. Validation and simulation of software system evolution rules using software networks[J]. Science China, 2014, 44(1): 20 – 36. (in Chinese)

[3] Lu J, Yu X, Chen G, et al. Characterizing the synchronizability of small-world dynamical networks[J]. Circuits & Systems I Regular Papers IEEE Transactions on, 2004, 51(4): 787 – 796.

[4] Z J, GH R, SR G, et al. Adaptive synchronization of an uncertain complex dynamical network[J]. IEEE Transactions on Automatic Control, 2005, 51(4): 652 – 656.

[5] Li H, Zhao H, Cai W, et al. A modular attachment mechanism for software network evolution[J]. Physica A Statistical Mechanics & Its Applications, 2013, 392(9): 2025 – 2037.

[6] Pan W, Li B, Ma Y, et al. Multi-granularity evolution analysis of software using complex network theory[J]. Journal of Systems Science and Complexity, 2011, 24(6): 1068 – 1082.

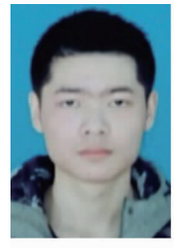
[7] Ambrose J A, Peddersen J, Parameswaran S, et al. SDG2KPN: system dependency graph to function-level KPN generation of legacy code for MPSoCs[A]. 19th Asia and South Pacific Design Automation Conference [C]. Sun Tec, Singapore: IEEE Press, 2014. 267 – 273.

[8] Ren J, Wu H, Yin T, et al. A novel approach for mining important nodes in directed-weighted complex software net-

work[J]. Journal of Computational Information Systems, 2015,11(8):3059 – 3071.
[9] Huang L, Ai J, Pei H. Software network models based on dynamic execution for fault propagation research [A]. IEEE International Conference on Software Quality, Reliability and Security[C]. Vancouver, Canada: IEEE Press, 2015. 56 – 61.
[10] Jenkins S, Kirk S R. Software architecture graphs as complex networks: A novel partitioning scheme to measure stability and evolution[J]. Information Sciences, 2007, 177(12):2587 – 2601.
[11] Shi M, Li X, Wang X. Evolving topology of java networks [A]. The Sixth World Congress on Intelligent Control and Automation(WCICA 2006)[C]. Dalian, China: IEEE Press, 2006. 26 – 30.
[12] Wang L, Wang Z, Yang C, et al. Linux kernels as complex networks: A novel method to study evolution[A]. Proceedings of IEEE International Conference on Software Maintenance[C]. Edmonton, Canada: IEEE Press, 2009. 41 – 50.
[13] Maillart T, Sornette D, Spaeth S, et al. Empirical tests of Zipf's law mechanism in open source Linux distribution. [J]. Physical Review Letters, 2008, 101(21):218701.
[14] Li H, Huang B, Lu J. Dynamical evolution analysis of the object-oriented software systems[A]. IEEE Congress on Evolutionary Computation[C]. Hong Kong, China: IEEE Press, 2008. 3030 – 3035.
[15] Concas G, Marchesi M, Pinna S, et al. On the suitability of Yule process to stochastically model some properties of object-oriented systems[J]. Physica A Statistical Mechanics & Its Applications, 2006, 370(2):817 – 831.
[16] Valverde S, Cancho R F I, Sole R V. Scale-free networks from optimal design[J]. Epl, 2002, 60(4):512 – 517.
[17] Zhang H, Zhao H, Cai W, et al. Using the k-core decomposition to analyze the static structure of large-scale software systems[J]. Journal of Supercomputing, 2010, 53(2):352 – 369.
[18] Sergi V, Solé R V. Network motifs in computational graphs: a case study in software architecture[J]. Physical Review E Statistical Nonlinear & Soft Matter Physics, 2005, 72(2):254 – 271.
[19] Bhattacharya P, Iliofotou M, Neamtiu I, et al. Graph-based analysis and prediction for software evolution[A]. 34th International Conference on Software Engineering[C]. Zürich, Switzerland: IEEE Press, 2012. 419 – 429.
[20] Dai F, Tong L I, Xie Z W, et al. Towards an algebraic semantics of software evolution process models[J]. Journal of Software, 2012, 23(4):846 – 863.
[21] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation[J]. ACM Sigmod Record, 2000, 29(2):1 – 12.
[22] Cheung Y L, Fu W C. Mining frequent itemsets without support threshold: with and without item constraints[J]. Knowledge & Data Engineering IEEE Transactions on, 2004, 16(9):1052 – 1069.
[23] Qiu S L, Chu D H, Meng F C. Component dynamic evolution mechanism based on spring[J]. Computer Engineering, 2012, 38(2):68 – 69.
[24] Li B, Ma Y, Liu J, et al. Advances in the studies on complex networks of software systems[J]. Advances in Mechanics, 2008, 38(6):805 – 814.
[25] Israeli A, Feitelson D G. The Linux kernel as a case study in software evolution[J]. Journal of Systems & Software, 2010, 83(3):485 – 501.
[26] Gu Q, Xiong S J, Chen D X. Correlations between characteristics of maximum influence and degree distributions in software networks[J]. Sciece China Information Sciences, 2014, 57(7):1 – 12.
[27] Newman M E J. Fast algorithm for detecting community structure in networks[J]. Physical Review E Statistical Nonlinear & Soft Matter Physics, 2004, 69(6):066133 – 066133.
[28] Newman M E. Assortative mixing in networks[J]. Physical Review Letters, 2002, 89(20):111 – 118.
[29] Drouin N, Badri M. Investigating the Applicability of the Laws of software Evolution: A Metrics Based Study[M]. Evaluation of Novel Approaches to Software Engineering. Springer Berlin Heidelberg, 2013. 174 – 189.

作者简介

何　鹏　男,1988 年生于江西宜春. 湖北大学计算机与信息工程学院讲师. 研究方向为软件工程、复杂网络.
E-mail:pegnhe@ hubu. edu. cn



王　鹏　男,1997 年生于湖北武汉. 湖北大学计算机与信息工程学院本科生. 研究方向为软件工程、复杂网络.
E-mail:657511017@ qq. com