Physica A 466 (2017) 249–258

Contents lists available at ScienceDirect

Physica A

journal homepage: www.elsevier.com/locate/physa

Evolution of Linux operating system network

Guanping Xiao, Zheng Zheng [*], Haoqin Wang

School of Automation Science and Electrical Engineering, Beihang University, Beijing, 100191, PR China

HIGHLIGHTS

- Networks of 62 major releases of Linux operating system are constructed.
- Manifestations of the evolution of network properties are observed.
- The evolution of functionality structures has revealed.
- Seven events are found in the evolution of functional modules.

ARTICLE INFO

Article history:
Received 15 June 2016
Received in revised form 23 August 2016
Available online 27 September 2016

Keywords:
Complex network
Linux operating system
Network evolution
Evolution events

ABSTRACT

Linux operating system (LOS) is a sophisticated man-made system and one of the most ubiquitous operating systems. However, there is little research on the structure and functionality evolution of LOS from the prospective of networks. In this paper, we investigate the evolution of the LOS network. 62 major releases of LOS ranging from versions 1.0 to 4.1 are modeled as directed networks in which functions are denoted by nodes and function calls are denoted by edges. It is found that the size of the LOS network grows almost linearly, while clustering coefficient monotonically decays. The degree distributions are almost the same: the out-degree follows an exponential distribution while both in-degree and undirected degree follow power-law distributions. We further explore the functionality evolution of the LOS network. It is observed that the evolution of functional modules is shown as a sequence of seven events (changes) succeeding each other, including continuing, growth, contraction, birth, splitting, death and merging events. By means of a statistical analysis of these events in the top 4 largest components (i.e., *arch*, *drivers*, *fs* and *net*), it is shown that continuing, growth and contraction events occupy more than 95% events. Our work exemplifies a better understanding and describing of the dynamics of LOS evolution.

1. Introduction

Many real-world complex systems ranging from nature to human society can be abstracted as networks, where entities are denoted by nodes and interactions between entities are denoted by edges [1–3]. At first, people used regular networks to represent real systems. After that, ER random-graph model was proposed [4], in which the edges are completely randomly connected. However, real-world networks are neither regular lattices nor simple random networks. Since the small-world network model and the scale-free network model were put forward at the end of the last century [5,6], complex networks science is flourishingly developed and greatly prompts the understanding of structures and functions of real-world complex systems, such as protein networks [7–9], Internet [10–12], power grids [13–16], airport networks [17–20] and scientific collaborations [21–23]. Over the past years, researchers primarily focus on several research aspects, involving epidemic

[*] Corresponding author.
E-mail address: zhengz@buaa.edu.cn (Z. Zheng).

http://dx.doi.org/10.1016/j.physa.2016.09.021
0378-4371/© 2016 Elsevier B.V. All rights reserved.

spreading [24–27], cascading failures [28–30], traffic dynamics [31–33], evolutionary games [34–36], network control [37,38], optimization process [39–42], coupling networks [43,44] etc. One interesting direction is to investigate structures and functions of software system networks [45–47].

Software systems represent one of the most sophisticated man-made systems which can be modeled as networks [48]. Over the past decades, a wealth of researches have been concentrated on software system networks from several aspects. Valverde et al. [49] presented the first evidence for the presence of scale-free and small-world behaviors in software systems from a well-defined local optimization. Cai et al. [50] modeled software dynamic execution processes as an evolving software mirror graph and found that execution processes might illustrate as a small-world network in the topological sense but no longer manifest in the temporal sense. Šubelj et al. [51] investigated the community structure in software networks. In the work, observations on several networks constructed from *Java* and various third party libraries were presented.

Operating system is a special software system which interacts with hardware devices and provides execution environments to the software executing on a computer [52]. Among various operating systems, Linux operating system (LOS) is typical and well deployed in most fields of the society nowadays. Due to its open source characteristic [53], the LOS network draws continuous attentions. In 2008, Zheng et al. [54] developed two new network growth models to better represent Gentoo Linux. Recently, Gao et al. [55] analyzed the core component of LOS as a complex network and showed that the large in-degree nodes providing basic services would do more damage on the whole system at the time of intentional attacks. Wang et al. [56] studied the network of networks in LOS and different manifestations of the coupling correlations among components were observed from the aspects of topologies and functions.

The functionality of LOS is continuously updated and optimized, brings it a strong vitality to adapt changing environments. Since version 1.0 was released in 1994, more than 1300 releases including versions 1.0 to 4.1[1] have been developed, and about every three months, a major version of LOS would be released. Understanding characteristics and processes of LOS evolution are meaningful to the development of operating systems, whereby what metrics could be utilized to evaluate changes of structures and functions of LOS during its evolution and how to reveal underlying principles of the evolution of LOS comprehensively are necessary to be explored. However, the study of the LOS network from the evolution aspect is still rare. To the best of our knowledge, the only work so far is that in 2011, Fortuna et al. [57] investigated packages in the first 10 releases of Debian GNU/Linux operating system and found that the modularity of the network of dependencies increased over time. In this paper, we systematically study the network evolution of 62 major releases of LOS ranging from version 1.0 released in 1994 to version 4.1 released in 2015, from the aspects of topological properties and functionality structures.

The paper is organized as follows. The following section describes the research data and the network modeling of LOS. Afterwards, results of the evolution of the LOS network are presented and discussed. Finally, the conclusion is given.

2. Methods

2.1. Research data

LOS is originally developed by Linus Torvalds in 1991. Over the past 20 years, more than 1300 releases ranging from versions 1.0 to 4.1 were put out, and the development of LOS has gone through three stages which can be classified by development methods [58]. The first stage contains versions 1.0 to 2.5, the second stage includes versions 2.6 series and the third stage contains version 3.0 and its subsequent releases. In the first stage, developers utilized 3 digits which labeled as “a.b.c” to represent the releases: the first digit “a” represents the LOS version, the second digit “b” denotes the major revision of LOS, and the third digit “c” indicates the minor revision of LOS. In this stage, a distinction was made between even and odd second major revision digits: odd digits were used to represent development releases, whereas even digits correspond to stable releases [59]. Differently, in the second stage, releases were not developed by dividing into development or stable versions, and 4 digits were used to denote the releases as a result of the change of development methods. The major revision of LOS was denoted by the third digit while the minor revision of LOS was represented by the forth digit. In the third stage, developers decided to stop the old numbering method which was utilized in versions 2.6, from which the numbering with 3 digits was reused to denote the releases. In this scheme, the development speed was accelerated by mainly separating the releases into mainline versions, stable versions and longterm versions [60].

The research data of LOS are obtained from 62 releases provided by the official website[2] of Linux ranging from versions 1.0 to 4.1, covering the three development stages of LOS, as exhibited in Fig. 1. According to the version numbering method of LOS, all of these releases are major releases.

2.2. Network modeling

LOS is a C language based operating system whose realization mainly depends on the function calls, which can be commonly termed as a call graph [48]. Consequently, a directed network based on the call graph where nodes are functions

[1] The latest release when we started this work.
[2] https://www.kernel.org/.

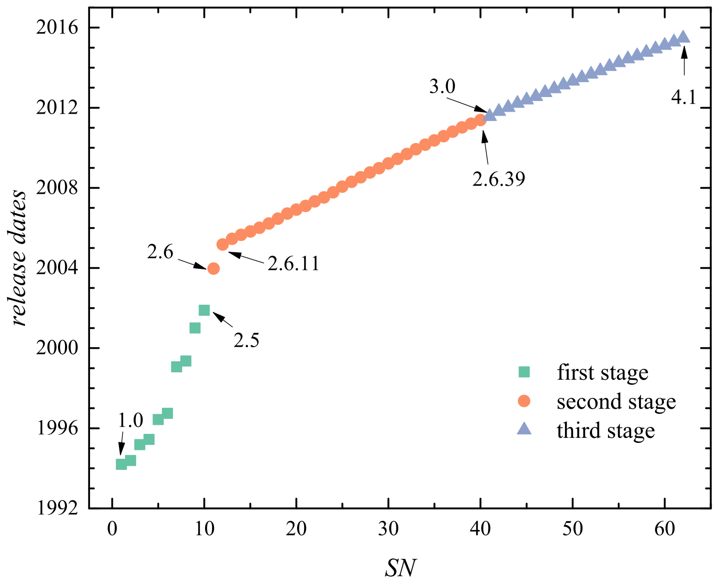


Fig. 1. Research data. 62 major releases ranging from version 1.0 released in 1994 to version 4.1 released in 2015. Note, *SN* represents the sequential number which is assigned to each version according to their release date, e.g., the sequence number of version 1.0 is 1 and that for version 4.1 is 62. The symbol will be used and discussed in the remaining parts of the paper.

and edges are function calls is constructed. For instance, a piece of C language program modeled as a directed network is illustrated in Fig. 2.

3. Results and discussion

3.1. Evolution of network properties

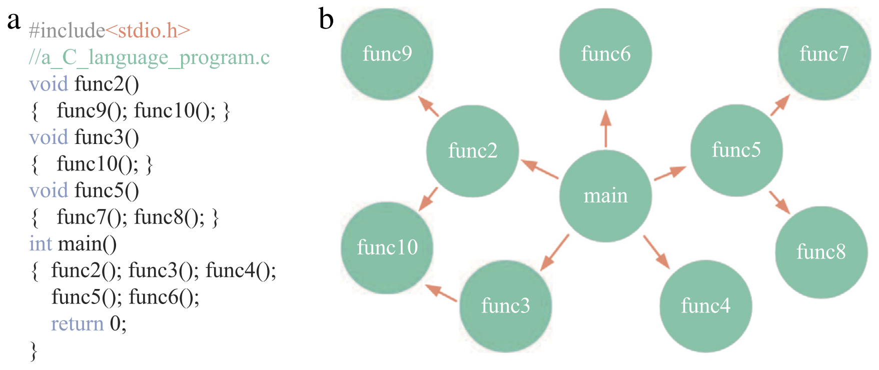
As shown in Fig. 3(a)(b), the size of the LOS network evolves almost linearly with the increasing of sequence numbers. It can be observed that the numbers of nodes and edges in each version increase about 6800 and 26000, respectively. A significant discrepancy can be easily obtained that the numbers of nodes n and edges m in version 4.1 are about 118 and 131 times larger than version 1.0, respectively. It is noticeable that the growth rates of n and m are slower after the sequence number 12 (version 2.6.11). This is due to the reason that, the interval between two major releases has become much shorter since the releasing of version 2.6.11 as a consequence of the change of development methods [58]. In Fig. 3(c), we plot clustering coefficient C with the increasing of sequence numbers. The clustering coefficient is utilized to describe the probability that a specified node's neighbors are also the neighbors of each other. In a directed network, the clustering coefficient for a node i is defined as [61]

$$C_i = \frac{1}{2}\frac{\sum\limits_j\sum\limits_k (a_{ij}+a_{ji})(a_{ik}+a_{ki})(a_{jk}+a_{kj})}{(k_i^{in}+k_i^{out})(k_i^{in}+k_i^{out}-1)-2\sum\limits_j a_{ij}a_{ji}} \tag{1}$$

where k_i^{in} and k_i^{out} are the in-degree and out-degree of node i, respectively. In Eq. (1), a_{ij} is set to be 1 if there exists an edge from i to j, otherwise $a_{ij} = 0$. The clustering coefficient for the whole network can be obtained by averaging C_i over all nodes. Thus, the clustering coefficient of the whole network is

$$C = \frac{1}{n}\sum_{i=1}^{n} C_i. \tag{2}$$

From Fig. 3(c), it is obvious that C decays with the increasing of sequence numbers, indicating the local connection of the LOS network will become looser with its evolution. Fig. 3(d) depicts the evolution of the average out-degree $\langle k_{out}\rangle$, the average in-degree $\langle k_{in}\rangle$ and the average undirected degree $\langle k\rangle$ of the LOS network. It can be observed that the average degree grows with the increasing of sequence numbers. It is worth mentioning that the growth rates of $\langle k_{out}\rangle$, $\langle k_{in}\rangle$ and $\langle k\rangle$ after the sequence number 12 (version 2.6.11) are slower, due to the same reason as previously described the growth rates of n and m in Fig. 3(a)(b). Besides, the correlations between average degrees and the number of nodes are depicted in Fig. 3(e). It shows a positive power-law relation (i.e., $\langle k\rangle \propto n^{\lambda}(\lambda = 0.023)$) between the average degree and the number of nodes. Average degrees will increase more and more slowly with the growth of the number of nodes.



```
#include<stdio.h>
//a_C_language_program.c
void func2()
{   func9(); func10(); }
void func3()
{   func10(); }
void func5()
{   func7(); func8(); }
int main()
{  func2(); func3(); func4();
   func5(); func6();
   return 0;
}
```

Fig. 2. An illustration for modeling a C language program as a directed network. (a) is an example of a C language program. The call graph depicted in (b) can be regarded as a directed network. The out-degree of a specified node represents the number of functions it calls (e.g., the out-degree of node main is 5) and the in-degree of a specified node represents the number of functions calling it (e.g., the in-degree of node func10 is 2).

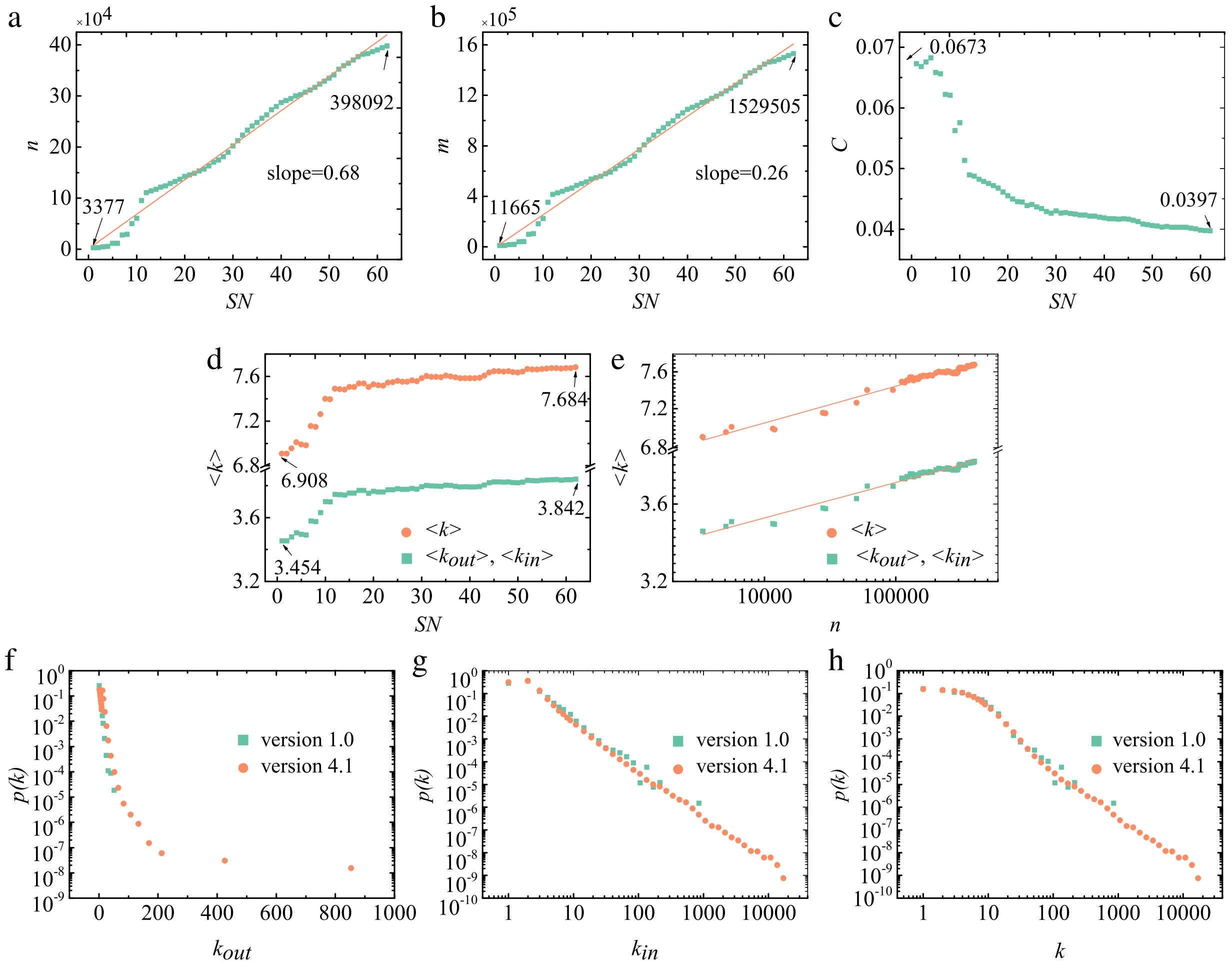


Fig. 3. Evolution of topological properties of the LOS network. (a) The number of nodes n. (b) The number of edges m. (c) Clustering coefficient C. (d) The average out-degree $\langle k_{out} \rangle$, the average in-degree $\langle k_{in} \rangle$ and the average undirected degree $\langle k \rangle$, all presented as functions of the sequence number (SN). (e) Correlation between the average degree and the number of nodes. (f) Out-degree distributions, (g) in-degree distributions and (h) undirected degree distributions of versions 1.0 and 4.1 (using logarithmic binning).

Fig. 3(f), (g) and (h) display degree distributions of versions 1.0 and 4.1 with logarithmic binning. The degree distribution, which defined the probability of a randomly selected node with degree k, is one of the most fundamental properties of networks [2]. It can be notably observed from Fig. 3(f), (g) and (h) that the two versions have similar degree distributions: the out-degree distribution follows an exponential distribution while both in-degree and undirected degree distributions

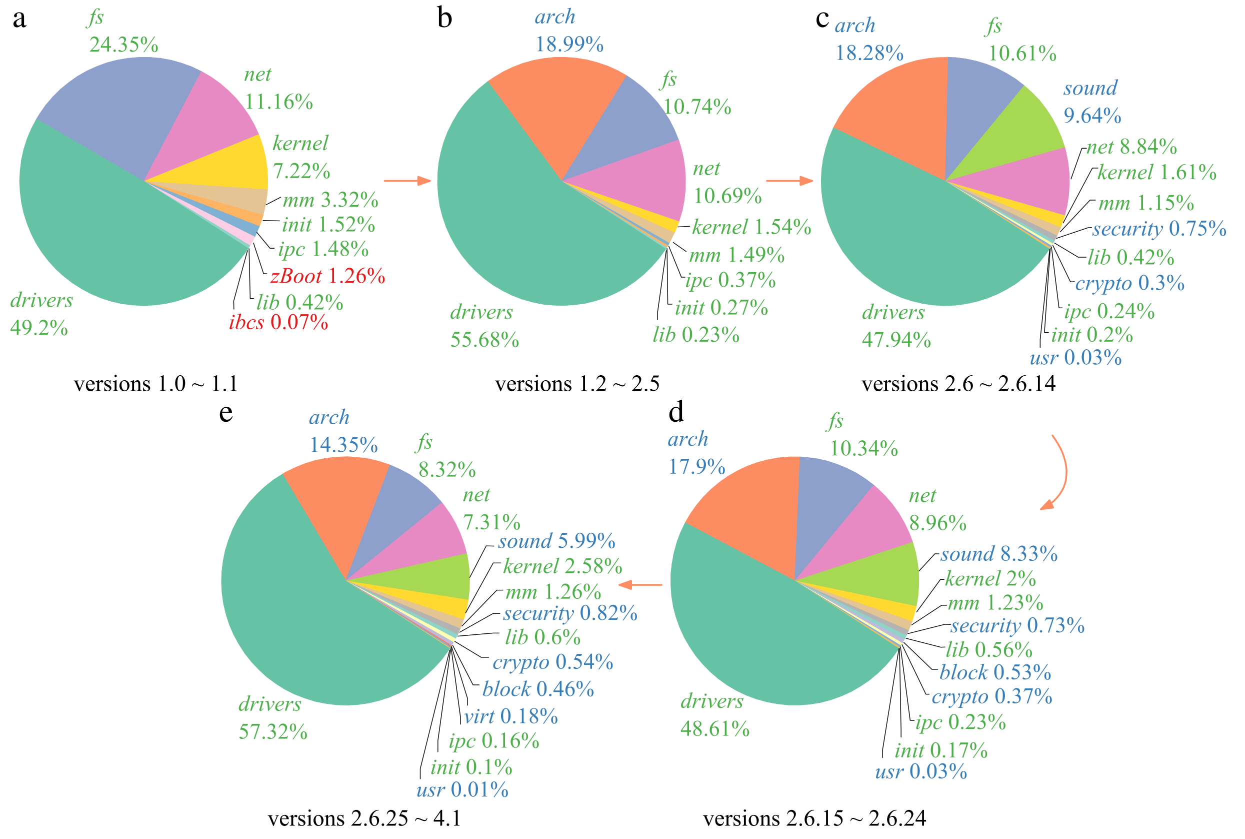


Fig. 4. Evolution of the number of components in the LOS network. The number of components in the LOS network has increased from 10 to 15 during its evolution. Note, components belonged to the first, second and the third categories are denoted by red, green and blue, respectively. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

experience power-law distributions, which is in well accordance with the results of Gao et al. [55] and Wang et al. [56]. Besides, the degrees have a wide range with the evolution of the LOS network, and maximal in-degrees are much larger than maximal out-degrees due to the reason that a reliable and readable programming method would usually prefer to write a function with only a few calls, which limits the value of the maximal out-degree. Meanwhile, the basic service functions would be called by a tremendous number of high-level functions, which brings extremely large in-degrees. By examining all versions, it is found that their degree distributions are similar to each other.

3.2. Evolution of components

LOS is a modular design software system, whose source codes are arranged in several major directories which can be considered as components [62], while subdirectories are usually viewed as functional modules, since they possess specific functions [63]. The components in the LOS network and their related functions are shown in Table 1.

As shown in Table 1, it can be found that 17 components have appeared in the LOS network during its evolution. However, not all of the 17 components would exist in each version. In Table 1, the first category indicates that components *ibcs* and *zBoot* exist in versions 1.0 and 1.1. Besides, from the second category of Table 1, it can be shown that eight components (i.e., *drivers*, *fs*, *init*, *ipc*, *kernel*, *lib*, *mm* and *net*) can be found in all versions and constitute the basic component of LOS. The third category in Table 1 indicates that these components (i.e., *arch*, *block*, *crypto*, *security*, *sound*, *usr* and *virt*) appear after version 1.1. In the following, a specific illustration of the evolution of the number of components in the LOS network will be elaborated.

It can be seen from Fig. 4 that the number of components has increased from 10 in version 1.0 to 15 in version 2.6.25 and its subsequent versions. In versions 1.0 and 1.1, the number of components is 10 but decreases to 9 in versions 1.2 to 2.5, which is due to that components *ibcs* and *zBoot* were removed while component *arch* was added in version 1.2 to support new processors (e.g., ARM processors and MIPS processors). Four new components including *crypto*, *sound*, *security* and *usr* were added in version 2.6. Therefore, the number of components increases to 13 in the versions from 2.6 to 2.6.14. The development of component *security* is due to the requirements of enhancing the security of LOS, making it becomes one of the most secure operating systems. Components *block* and *virt* were developed in versions 2.6.15 and 2.6.25, respectively. Thus, the number of components in the LOS network increases to 15 in the versions from 2.6.25 to 4.1. Another observation

Table 1
Components in the LOS network and their related functions. The first category represents components which only appear in versions 1.0 and 1.1; the second category represents components which exist in all versions; the third category denotes components which are added after version 1.1. Note, only those directories which contain.c files and relate to the functionality of LOS are investigated.

Category	Component	Related function	Component	Related function
1 (versions 1.0 and 1.1)	*ibcs*	Intel binary compatibility standard	*zBoot*	Decompress kernel
2 (all versions)	*drivers*	Device drivers	*fs*	File systems
	init	Initialization for the kernel	*ipc*	Inter-process communication
	kernel	Architecture independent main kernel	*lib*	Generic library functions
	mm	Architecture independent memory management	*net*	Networking
3 (after version 1.1)	*arch*	Different architectures	*block*	Generic block layer
	crypto	Cryptography framework	*security*	Security module
	sound	Sound system	*usr*	Early userspace
	virt	Virtualization infrastructure		

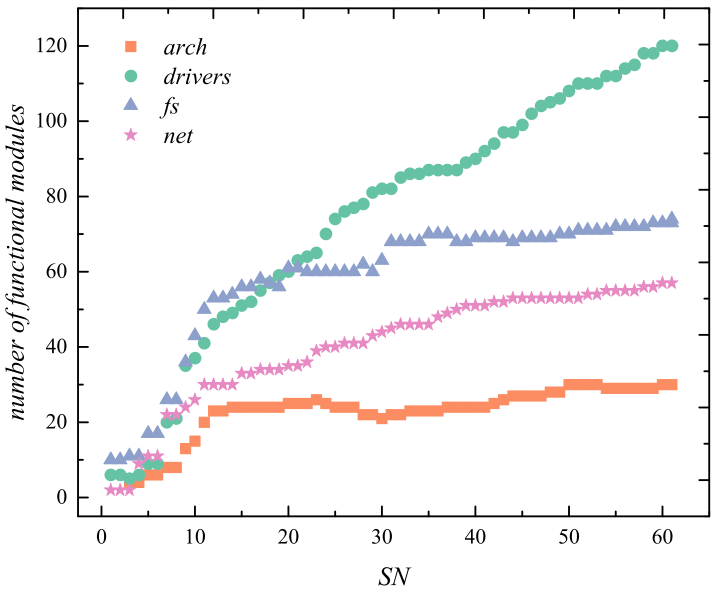


Fig. 5. Evolution of the number of functional modules in the top 4 largest components *arch*, *drivers*, *fs* and *net*. The number of functional modules grows with the increasing of sequence numbers.

worth mentioning is that, component *drivers* occupies about 50% nodes in all versions since LOS supports huge number of external devices.

3.3. Evolution of functional modules

Fig. 5 shows the evolution of the number of functional modules in the top 4 largest components *arch*, *drivers*, *fs* and *net*. It is found that the number of functional modules in *arch* becomes stable from version 2.6.11 (the sequence number 12) which attributes to the removal of out-of-date architectures, while the number of functional modules in *drivers* grows almost linearly due to the flourishing development of device drivers.

Researchers found that the evolution of some entities in real-world complex networks can be considered as a sequence of certain basic events (changes) succeeding each other [64,65]. By manually investigating change logs of LOS and the changing of the network size of each functional module with its evolution, seven events, including continuing, growth, contraction, birth, splitting, death and merging, are observed in the evolution of functional modules, as shown in Fig. 6. These events will be further discussed in the following.

Note, in this subsection, version N_i represents a specific version of LOS, while version N_{i+1} denotes the subsequent version of N_i.

- *Continuing:* a functional module M_1 in version N_i continues its existence in version N_{i+1}, and the numbers of nodes in the two versions are identical.
- *Growth:* a functional module M_1 grows when some nodes are added in version N_{i+1}, making its size bigger than version N_i. This event could be caused by the consideration of new requirements or bug fixing.

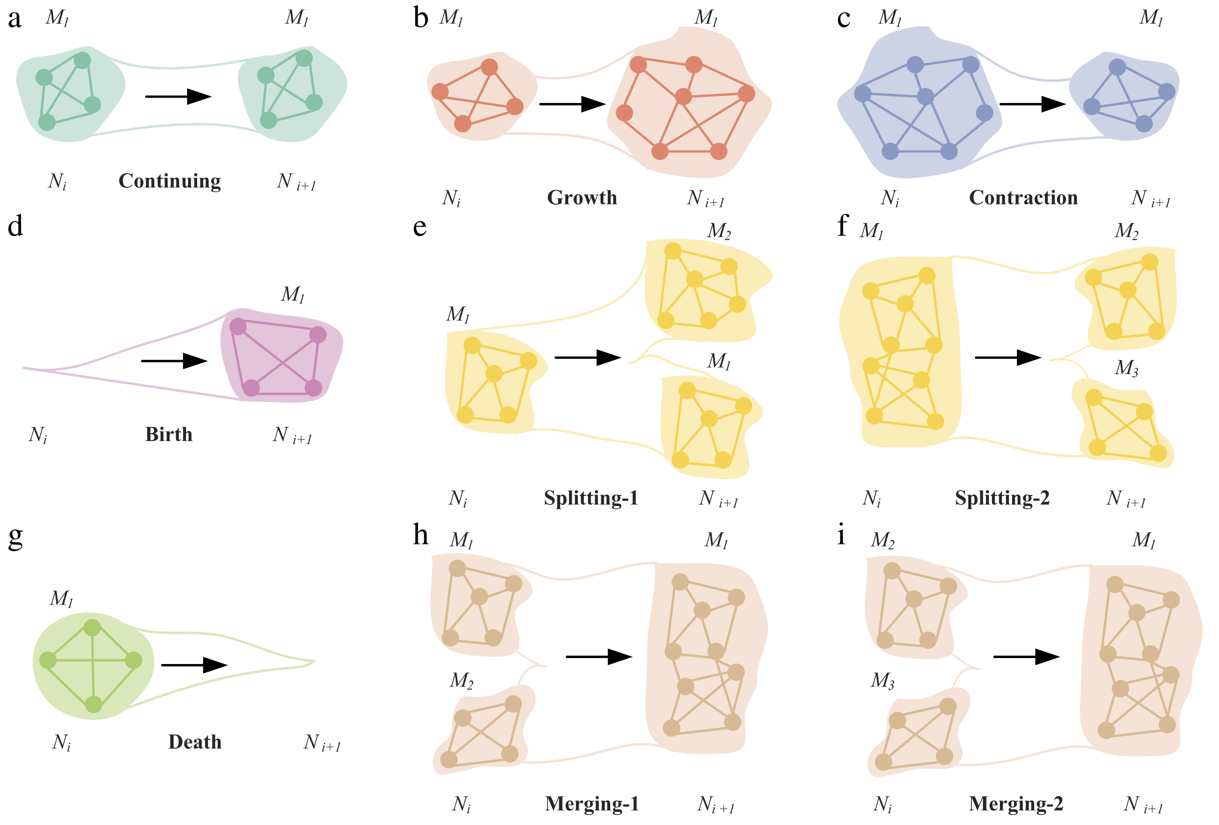


Fig. 6. Seven events in the evolution of functional modules. (a) Continuing. (b) Growth. (c) Contraction. (d) Birth. (e) Splitting-1. (f) Splitting-2. (g) Death. (h) Merging-1. (i) Merging-2.

- *Contraction:* a functional module M_1 contracts when some nodes are removed in version N_{i+1}, making its size smaller than version N_i. The event could be attributed to the old feature removal or bug fixing.
- *Birth:* a functional module M_1 did not exist in version N_i, but appears in version N_{i+1}. The birth of a functional module can be attributed to the new functionality requirement from the developers or users. For instance, functional module *ps3* was added in component *drivers* into version 2.6.20, indicating that Sony PS3 gained official Linux support.
- *Splitting:* a functional module M_1 splits into two or more functional modules in version N_{i+1}. In Ref. [65], the splitting events are simply divided into two types: equally or unequally splitting. However, in the LOS network, two different splitting types are observed, which are depicted in Fig. 6(e)(f). The first (splitting-1) is that a new functional module M_2 is forked from M_1, while M_1 is still existing. For example, in component *fs*, filesystem *ext2* splits into *ext2* and *ext3* in version 2.5 due to the reason that *ext3* was developed based on *ext2* by forking the source code of *ext2*. The second (splitting-2) is to separate M_1, which has complex functions, to several functional modules, whose function is more specific.
- *Death:* a functional module M_1 existed in version N_i, but disappears in version N_{i+1}. Event death is the opposite side of event birth. If one functional module is out of date or has not been maintained for a long time, it will be removed.
- *Merging:* a functional module M_1 has been created by merging several functional modules. In Ref. [65], the merging events are divided into two types: equally or unequally merging. However, in the LOS network, two different merging types are observed, which are depicted in Fig. 6(h)(i). The first one (merging-1) is some other functional modules are merged into M_1 in order to make the functional modules with the similar functions more convenient to be maintained. The second one (merging-2) is to create a new functional module M_1 via assembling some other functional modules. For example, in component *arch*, functional module *x86* in version 2.6.24 was formed by merging functional modules *i386* and *x86_64*.

Furthermore, we investigate the statistical results of the seven events in the top 4 largest components *arch*, *drivers*, *fs* and *net*, as illustrated in Table 2. It should be noted that, the difference of the total numbers of the seven events in Table 2 is mainly attributed to the different numbers of functional modules. In Table 2, it is notable that birth, splitting, death and merging events happened mostly on the first and second stages of LOS development, indicating the optimization and regulation of the functionality of LOS are mostly implemented in the first and the second development stages. Comparatively, continuing, growth and contraction events are majorly observed in the second and the third stages, since the number of functional modules is comparatively fewer in the first stage, which can also be figured out from Fig. 5.

As shown in Fig. 7, continuing, growth and contraction events are most likely to occur, and these events in components *arch*, *drivers*, *fs* and *net* occupy more than 95%. Comparably, the percentages of birth, death, splitting and merging events

Table 2
Statistics of the seven events in the top 4 largest components *arch*, *drivers*, *fs* and *net*. Note, Dev. Stage represents the development stages of LOS mentioned in Section 2.1.

Comp.	Dev. Stage	Continuing	Growth	Contraction	Birth	Splitting	Death	Merging	Tot.
	1st	4	38	7	15	3	0	0	
arch	2nd	90	400	195	20	5	11	8	1421
	3rd	133	240	243	7	1	1	0	
	1st	19	89	6	40	0	3	0	
drivers	2nd	569	1270	214	61	3	8	4	4681
	3rd	797	1276	275	35	3	5	4	
	1st	36	89	37	45	3	2	0	
fs	2nd	737	816	258	36	3	10	1	3632
	3rd	693	701	156	6	2	1	0	
	1st	30	67	11	27	1	1	0	
net	2nd	378	668	128	27	0	2	0	2532
	3rd	504	562	116	8	0	2	0	

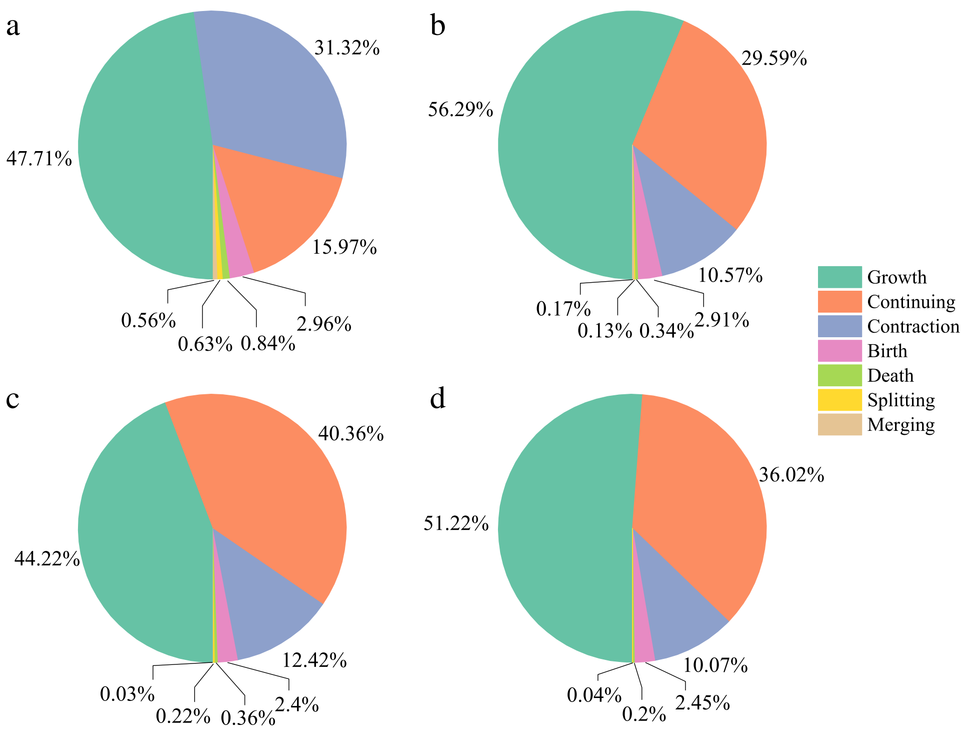


Fig. 7. Percentages of the seven events from versions 1.0 to 4.1 in the top 4 largest components. (a) *arch*. (b) *drivers*. (c) *fs*. (d) *net*.

are small. However, these events may play crucial roles, since these four events can be regarded as the optimization and regulation of the functionality of LOS to certain extent.

Another point worth mentioning is that the percentage of contraction events in component *arch* is the highest comparing with other components. This phenomenon is mainly due to the removal of out-of-date processors in architectures.

4. Conclusion

LOS evolves to improve their fitness to changing environments, commonly motivated by the new requirements of hardware architectures and users [66]. In this paper, we have focused on the evolution of network properties and functionality structures of the LOS network. Our work has revealed initial insights into LOS evolution, suggesting that there exist recognizable patterns (i.e., network properties evolution trends and evolution events) in the evolution of LOS. The variation of network properties and evolution events indicates that the evolution of LOS adheres to some principles. On the other hand, these patterns can be fairly considered as outcomes of the evolution of LOS. The major benefits of these outcomes reflect the variation of the evolution of LOS which are driven by the changing environments and user requirements and vice versa affecting them. For example, the functional module *x86_64* in component *arch* was developed by forking the codes of functional module *i386* due to the requirement for supporting 64-bit version of the *i386* architecture. However, with the evolution of the two functional modules, bug fixing during the maintenance and new features introduced in the development process often make developers confuse that whether all changes being made in both functional modules, since there is quite a lot of commonality between them. This drives developers to merge them, which results in the creation of the functional module *x86*. Furthermore, the results of this paper signify that the size of LOS and its functions keep continuously increasing, also indicate that the usage of core functions (e.g., basic service functions) will increase while the cluster tendency of LOS will become looser with its evolution. Additionally, a lack of sufficient knowledge of the evolution process would cause many of the problems in software maintenance, while the results of this paper could provide such data as references for the maintenance of LOS. In conclusion, these patterns revealed in this paper could be utilized as indicators of the maintenance or improvement of LOS. Besides, the evolution trends of the patterns could be further used as the evaluation of the next versions of LOS.

Acknowledgment

The authors would like to thank Dr. Wen-Bo Du from Beihang University for kindly providing valuable technical advices to this manuscript.

References

[1] R. Albert, A.-L. Barabási, Statistical mechanics of complex networks, Rev. Modern Phys. 74 (2002) 47.
[2] M.E. Newman, The structure and function of complex networks, SIAM Rev. 45 (2003) 167–256.
[3] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, D.-U. Hwang, Complex networks: Structure and dynamics, Phys. Rep. 424 (2006) 175–308.
[4] P. Erdős, A. Rényi, On random graphs, Publ. Math. Debrecen 6 (1959) 290–297.
[5] D.J. Watts, S.H. Strogatz, Collective dynamics of 'small-world'networks, Nature 393 (1998) 440–442.
[6] A.-L. Barabási, R. Albert, Emergence of scaling in random networks, Science 286 (1999) 509–512.
[7] H. Jeong, S.P. Mason, A.-L. Barabási, Z.N. Oltvai, Lethality and centrality in protein networks, Nature 411 (2001) 41–42.
[8] L. Giot, J.S. Bader, C. Brouwer, A. Chaudhuri, B. Kuang, Y. Li, Y. Hao, C. Ooi, B. Godwin, E. Vitols, A protein interaction map of drosophila melanogaster, Science 302 (2003) 1727–1736.
[9] J.-F. Rual, K. Venkatesan, T. Hao, T. Hirozane-Kishikawa, A. Dricot, N. Li, G.F. Berriz, F.D. Gibbons, M. Dreze, N. Ayivi-Guedehoussou, Towards a proteome-scale map of the human protein–protein interaction network, Nature 437 (2005) 1173–1178.
[10] R. Cohen, K. Erez, D. Ben-Avraham, S. Havlin, Breakdown of the internet under intentional attack, Phys. Rev. Lett. 86 (2001) 3682.
[11] R. Pastor-Satorras, A. Vázquez, A. Vespignani, Dynamical and correlation properties of the internet, Phys. Rev. Lett. 87 (2001) 258701.
[12] A. Vázquez, R. Pastor-Satorras, A. Vespignani, Large-scale topological and dynamical properties of the internet, Phys. Rev. E 65 (2002) 066130.
[13] R. Albert, I. Albert, G.L. Nakarado, Structural vulnerability of the north american power grid, Phys. Rev. E 69 (2004) 025103.
[14] P. Crucitti, V. Latora, M. Marchiori, A topological analysis of the italian electric power grid, Physica A 338 (2004) 92–97.
[15] M. Rosas-Casals, S. Valverde, R.V. Solé, Topological vulnerability of the european power grid under errors and attacks, nt, J. Bifurc. Chaos 17 (2007) 2465–2475.
[16] G.A. Pagani, M. Aiello, The power grid as a complex network: a survey, Physica A 392 (2013) 2688–2700.
[17] W. Li, X. Cai, Statistical analysis of airport network of china, Phys. Rev. E 69 (2004) 046106.
[18] R. Guimera, S. Mossa, A. Turtschi, L.A. Amaral, The worldwide air transportation network: Anomalous centrality, community structure, and cities' global roles, Proc. Natl. Acad. Sci. USA 102 (2005) 7794–7799.
[19] M. Guida, F. Maria, Topology of the italian airport network: A scale-free small-world network with a fractal structure? Chaos Solitons Fractals 31 (2007) 527–536.
[20] G. Bagler, Analysis of the airport network of india as a complex weighted network, Physica A 387 (2008) 2972–2980.
[21] M.E. Newman, The structure of scientific collaboration networks, Proc. Natl. Acad. Sci. USA 98 (2001) 404–409.
[22] A.-L. Barabási, H. Jeong, Z. Néda, E. Ravasz, A. Schubert, T. Vicsek, Evolution of the social network of scientific collaborations, Physica A 311 (2002) 590–614.
[23] M.E. Newman, Coauthorship networks and patterns of scientific collaboration, Proc. Natl. Acad. Sci. USA 101 (2004) 5200–5205.
[24] R. Pastor-Satorras, A. Vespignani, Epidemic spreading in scale-free networks, Phys. Rev. Lett. 86 (2001) 3200.
[25] M. Boguná, R. Pastor-Satorras, Epidemic spreading in correlated complex networks, Phys. Rev. E 66 (2002) 047104.
[26] Y. Wang, D. Chakrabarti, C.-X. Wang, C. Faloutsos, Epidemic spreading in real networks: An eigenvalue viewpoint, in: International Symposium on Reliable Distributed Systems, Proceedings, IEEE, 2003, pp. 25–34.
[27] C. Castellano, R. Pastor-Satorras, Thresholds for epidemic spreading in networks, Phys. Rev. Lett. 105 (2010) 218701.
[28] D.J. Watts, A simple model of global cascades on random networks, Proc. Natl. Acad. Sci. USA 99 (2002) 5766–5771.
[29] P. Crucitti, V. Latora, M. Marchiori, Model for cascading failures in complex networks, Phys. Rev. E 69 (2004) 045104.
[30] W.-X. Wang, Y.-C. Lai, Abnormal cascading on complex networks, Phys. Rev. E 80 (2009) 036109.
[31] G. Yan, T. Zhou, B. Hu, Z.-Q. Fu, B.-H. Wang, Efficient routing on complex networks, Phys. Rev. E 73 (2006) 046108.
[32] W.-X. Wang, B.-H. Wang, C.-Y. Yin, Y.-B. Xie, T. Zhou, Traffic dynamics based on local routing protocol on a scale-free network, Phys. Rev. E 73 (2006) 026111.
[33] W.-B. Du, Z.-X. Wu, K.-Q. Cai, Effective usage of shortest paths promotes transportation efficiency on scale-free networks, Physica A 392 (2013) 3505–3512.
[34] G. Szabó, G. Fath, Evolutionary games on graphs, Phys. Rep. 446 (2007) 97–216.
[35] W.-B. Du, X.-B. Cao, M.-B. Hu, W.-X. Wang, Asymmetric cost in snowdrift game on scale-free networks, Eur. Phys. Lett. 87 (2009) 60004.
[36] Z. Wang, A. Szolnoki, M. Perc, Interdependent network reciprocity in evolutionary games, Sci. Rep. 3.
[37] Y.-Y. Liu, J.-J. Slotine, A.-L. Barabási, Controllability of complex networks, Nature 473 (2011) 167–173.
[38] G. Yan, G. Tsekenis, B. Barzel, J.-J. Slotine, Y.-Y. Liu, A.-L. Barabási, Spectrum of controlling and observing complex networks, Nat. Phys. 11 (2015) 779–786.
[39] C. Liu, W.-B. Du, W.-X. Wang, Particle swarm optimization with scale-free interactions, PLoS One 9 (2014) e97822.
[40] Y. Gao, W.-B. Du, G. Yan, Selectively-informed particle swarm optimization, Sci. Rep. 5.
[41] W.-B. Du, Y. Gao, C. Liu, Z. Zheng, Z. Wang, Adequate is better: particle swarm optimization with limited-information, Appl. Math. Comput. 268 (2015) 832–838.
[42] W.-B. Du, W. Ying, G. Yan, Y.-B. Zhu, X.-B. Cao, Heterogeneous strategy particle swarm optimization, IEEE Trans. Circuits Syst. II Exp. Briefs 99 (2016) 1–1.
[43] S.V. Buldyrev, R. Parshani, G. Paul, H.E. Stanley, S. Havlin, Catastrophic cascade of failures in interdependent networks, Nature 464 (2010) 1025–1028.
[44] W.-B. Du, X.-L. Zhou, O. Lordan, Z. Wang, C. Zhao, Y.-B. Zhu, Analysis of the chinese airline network as multi-layer networks, Transp. Res. E 89 (2016) 108–116.

[45] S. Jenkins, S.R. Kirk, Software architecture graphs as complex networks: A novel partitioning scheme to measure stability and evolution, Inf. Sci. 177 (2007) 2587–2601.
[46] G. Concas, M. Marchesi, S. Pinna, N. Serra, Power-laws in a large object-oriented software system, IEEE Trans. Softw. Eng. 33 (2007) 687–708.
[47] P. Louridas, D. Spinellis, V. Vlachos, Power laws in software, ACM Trans. Softw. Eng. Methodol. 18 (2008) 2.
[48] C.R. Myers, Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs, Phys. Rev. E 68 (2003) 046116.
[49] S. Valverde, R.F. Cancho, R.V. Sole, Scale-free networks from optimal design, Eur. Phys. Lett. 60 (2002) 512.
[50] K.-Y. Cai, B.-B. Yin, Software execution processes as an evolving complex network, Inf. Sci. 179 (2009) 1903–1928.
[51] L. Šubelj, M. Bajec, Community structure of complex software systems: Analysis and applications, Physica A 390 (2011) 2968–2975.
[52] A. Silberschatz, P.B. Galvin, G. Gagne, A. Silberschatz, Operating System Concepts, Addison-Wesley Reading, 1998.
[53] G. Hertel, S. Niedner, S. Herrmann, Motivation of software developers in open source projects: an internet-based survey of contributors to the linux kernel, Res. Poli. 32 (2003) 1159–1177.
[54] X.-L. Zheng, D. Zeng, H.-Q. Li, F.-Y. Wang, Analyzing open-source software systems as complex networks, Physica A 387 (2008) 6190–6200.
[55] Y.-C. Gao, Z. Zheng, F.-Y. Qin, Analysis of linux kernel as a complex network, Chaos Solitons Fractals 69 (2014) 246–252.
[56] H.-Q. Wang, Z. Chen, G.-P. Xiao, Z. Zheng, Network of networks in linux operating system, Physica A 447 (2016) 520–526.
[57] M.A. Fortuna, J.A. Bonachela, S.A. Levin, Evolution of a modular software network, Proc. Natl. Acad. Sci. USA 108 (2011) 19985–19989.
[58] A. Israeli, D.G. Feitelson, The linux kernel as a case study in software evolution, J. Syst. Softw. 83 (2010) 485–501.
[59] D.P. Bovet, M. Cesati, Understanding the Linux Kernel, O'Reilly Media, Inc., 2005.
[60] Y. Tian, J. Lawall, D. Lo, Identifying linux bug fixing patches, in: International Conference on Software Engineering, IEEE, 2012, pp. 386–396.
[61] G. Fagiolo, Clustering in complex directed networks, Phys. Rev. E 76 (2007) 026107.
[62] R. Love, Linux Kernel Development, Pearson Education, 2010.
[63] L.-P. Lu, S.-W. Li, Y.-H. Li, Design of car bluetooth hands-free mobile phone system in linux system, in: International Conference on Automatic Control and Artificial Intelligence, IET, 2012, pp. 836–839.
[64] G. Palla, A.-L. Barabási, T. Vicsek, Quantifying social group evolution, Nature 446 (2007) 664–667.
[65] P. Bródka, S. Saganowski, P. Kazienko, Ged: the method for group evolution discovery in social networks, Soc. Netw. Anal. Min. 3 (2013) 1–14.
[66] M.M. Lehman, J.F. Ramil, Software evolution—background, theory, practice, Inf. Process. Lett. 88 (2003) 33–44.