The Influences of Edge Instability on Change Propagation and Connectivity in Call Graphs

Lei Wang(✉), Han Li, and Xinchen Wang

School of Computer Science and Engineering, Beihang University, Beijing, China
{wanglei,sy1406228,wxc11061106}@buaa.edu.cn

Abstract. During the lifetime of any software there are numerous changes, which lead to a large number of versions over time. The amount of effort in programming and debugging for these updates and therefore the reliability of the software depends substantially on how far the change propagates. We introduced the concept of Propagation Scope (*PS*) to quantify change propagation and investigated several open-source software systems. We found that the propagation property varies even with systems of similar scales. According to the asymmetry between the in-degree and out-degree distributions in call graphs of software, we defined Edge Instability (*EI*) to measure the change propagation of a call graph. Analyzing newly added nodes in six software, we found that the new nodes exhibited preferential attachment behaviors and were more likely to call new nodes. We proposed a model based on these observations to adjust *EI* and Clustering Coefficient (*CC*). *CC* has been believed to be the major factor determining the propagation scope in a network. Our experiments showed, however, that *EI* had a larger impact on the propagation of call graphs. In both real software and our model, we measured the connectivity of call graphs with *EI* and evaluated connectivity under three edge-removal strategies. Our experiments showed that removing edges with high *EI*s hurt network connectivity the most.

Keywords: Complex networks · Software evolution · Change propagation · Network model · Call graph

1 Introduction

It has been observed that ideas, information, viruses, and diseases often propagate in the form of complex networks [30] and a network's topological structure has a significant impact on the dynamics of change propagation [28]. In the domain of computer science, it has been demonstrated that class diagrams [34,35], collaboration graphs [29], package dependency networks [23], the object graphs [31], software component graphs [20], and call graphs in large-scale software systems [37] are all complex networks. In this paper, we studied change propagation in call graphs, a critical aspect in software evolution. As developers code to introduce new features or fix bugs for one part of a software system, other parts need to be updated accordingly to stay consistent with the changes.

P. Stevens and A. Wąsowski (Eds.): FASE 2016, LNCS 9633, pp. 197–213, 2016.
DOI: 10.1007/978-3-662-49665-7_12

For example, when a function's prototype changes, its callers have to be modified to call through the new interface. To understand the evolution of software systems, we collected call graphs of a large number of software systems of multiple versions. A call graph describes the calling relationship between functions in a program. Specifically, functions in the program are represented as nodes in a call graph. If one function calls another, an edge from the node representing the caller to the node of the callee function is added to the graph. We generated a call graph for each version of a selected software system and, by comparing the graphs of different versions, investigated change propagation as software evolves.

We selected 35 stable Linux kernels from version 1.0 to version 2.2.26 (available at http://ftp.kernel.org) and generated their call graphs using a modified version of GCC 3.4.6 [2]. To compare update propagation characteristics among different software, we collected call graphs for five additional open-source projects, including 80 versions of Samba, 25 versions of BIND, 55 versions of Sendmail, 76 versions of OpenSSH, and 59 versions of vsftpd obtained from the code repository [7]. To identify updates between two adjacent versions of a software system, we use *ctags* [1] to get the start and end points of a function in one version. We then compared the functions of the same name in the two versions to decide if the function is updated. We also identified functions removed from the older version or added in the newer version. All these changes, including updates, additions, and removals of functions, can be identified in the call graphs when they are propagated to other functions in the software. In this paper, we name the versions with ordered sequence numbers starting from 0 following their chronological order.

We quantified network propagation with Propagation Scope (PS). The scope that a change propagation can reach is mostly determined by the topological structure of the corresponding call graph, which can be characterized by many factors, including the graph's node count, edge count, average node degree, etc. Among the factors, the clustering coefficient (CC) measures how tightly nodes in a network are clustered and is believed to be the most powerful factor determining the propagation of networks [38,41]. We found, however, that propagation in a call graph was not sensitive to CC changes. In fact, for the studied software, the asymmetry between the in-degree and out-degree distributions were manifest [36] and have a significant influence on the change propagation of software [8]. Given the asymmetry, we introduce Edge Instability (EI) to measure the propagation. We found that the new nodes exhibit preferential attachment behaviors and are more likely to call new nodes. With these observations, we propose a model to adjust CC and EI based on Barabási and Albert (BA) model and it's extension [6,18]. Experiments showed that EI has a larger impact than CC on the propagation of call graph.

Inspired by the influences of EI on change propagation, we use EI to measure the connectivity of call graphs. In complex networks, researchers often study robustness by measuring connectivity after removing nodes or edges [10,32]. We adopted the same methodology to evaluate the connectivity of call graphs and compare three strategies to attack generated graphs by our model: 1. Removing edges randomly. 2. Removing edges with higher EIs. 3. Removing edges with higher "edge degrees" [19]. Our experiments showed that removing edges with high EIs hurt network robustness more than removing edges with high "edge degrees" or randomly.

The rest of this paper is organized as follows. Section 2 introduces the concept of propagation scope, edge instability and statistics with various software systems. Connectivity of call graphs under three edge-removal strategies are discussed in Sect. 3. Section 4 describes behaviors of new nodes and an innovative model of software evolution. The correlation among parameters of the proposed model, change propagation and connectivity are also discussed in Sect. 4. Section 5 introduces the related work briefly. The paper closes with our conclusions in Sect. 6.

2 Change Propagation

To quantify the change propagation in a network, we introduce the concept of propagation scope (PS), derived from the concepts of *Change Cost* [26] and *Average Propagation Ratio* [25], and edge instability. To reveal impact of the structure of a call graph on PS, we will measure the number of nodes and edges, the average node degree, the diameter, the clustering coefficient and the edge instability of the call graphs under investigation.

2.1 Propagation Scope

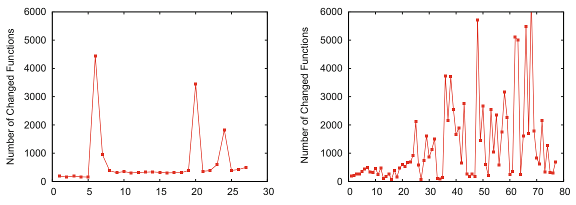
The concept of propagation scope is motivated by the observation that in a call graph a change propagates in one direction. For example, if Function A calls Function B and B's interface is changed, Function A has to change accordingly, or the change of B propagates to A. Changes do not propagate in the opposite direction. For example, the change of Function A does not affect function B. Formally, we state that Node n_i can reach Node n_j within a distance of 1 if there is a directed edge $< n_i, n_j >$ in a network G. We use R_j^d to denote the set of nodes that can reach n_j within distance d. Formally, we use Eq. (1) to define the propagation scope of Network G within distance d, or PS_G^d.

$$PS_G^d = \frac{\sum_{j=1}^{|N|} \left|R_j^d\right|}{|N|^2} \tag{1}$$

where $|N|$ is the number of nodes in Network G.

According to Eq. (1) and the definition of R_j^d, if $d_1 > d_2$, then $PS_G^{d1} \geqslant PS_G^{d2}$, because R_j^d monotonically increases with d. Using D to denote network diameter, we have $PS_G^d = PS_G^D$ for any d larger than D. Finally, the propagation scope of Network G, PS_G, can be defined using Eq. (2).

$$PS_G = PS_G^D = \frac{\sum_{j=1}^{|N|} \left|R_j^D\right|}{|N|^2} \tag{2}$$



(a) Linux (b) Samba

Fig. 1. Number of changed functions in Linux and Samba.

The propagation scope can be used to differentiate networks of different topologies in terms of impact of a node's change on other nodes. The larger the propagation scope, the greater the number of nodes affected by changes taking place at a node.

By definition, *PS* is obviously related to the number of nodes, the number of edges and the average node degree. We compare some versions of Linux with versions of Samba of similar scale. The results are shown in Table 1. In the table, "Linux1" stands for set of the Linux versions from 1.2.0 to 1.2.10, "Linux2" stands for set of the Linux versions from 2.0.0 to 2.0.40. "Samba1" stands for set of Samba versions from 2.2.8 to 2.2.12, and "Samba2" stands for set of Samba versions from 3.0.25 to 3.0.34. As shown in Table 1, Linux1 and Samba1 have similar node count and edge count. This is also the case for Linux2 and Samba2. However, we can see that the PS_G values of different Linux versions is always smaller than those of Sambas in Table 1.

Table 1. Statistics with Linux and Samba of similar scale.

software	Linux1	Samba1	Linux2	Samba2
Avg. node	3993	3803	8099	7373
Avg. edge	14996	13849	31400	30513
Ave. degree	7.51	7.28	7.75	8.27
PS_G	0.0135	0.0297	0.0112	0.0295

We also investigate changes between two adjacent versions of Linux and Samba, respectively. Figures 1 and 2 showed the number of changed functions and the size of the maximal connected subgraphs. To compare Linux with Samba using their call graphs of similar scales, we chose Linux 2.0.40 and Samba 3.0.34 as the last tested version shown in Figs. 1 and 2. For most versions, the number of changed functions in Linux was fewer than that of Samba. For the maximal connected subgraphs, the sizes in the two versions of Linux were larger than 200, and those in other versions were less than 90. On the other hand, in over one third of the versions of Samba, the sizes of the maximal connected subgraphs were larger than 200, with the largest of 1501. The change propagation in Samba seemed substantially larger than that in Linux. This observation is consistent with Table 1. Therefore, *PS* could be used to measure the change propagation in call graphs of software.

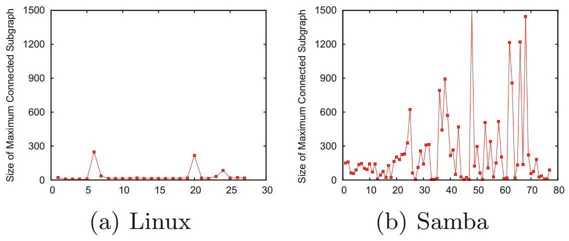


Fig. 2. Size of maximal connected subgraphs in Linux and Samba.

2.2 Edge Instability

The asymmetry between the in-degree and out-degree distributions of software appears obviously [9,29,34,36]. The in-degree distribution of software systems obviously obeys the power-law while the out-degree distribution are similar to the power-law distribution with a cutoff. Inspired by the asymmetry of degree distribution and direction of change propagation in software, we propose the Edge Instability of a call graph.

Firstly, we define the node stability S_i for Node n_i in Eq. (3).

$$S_i = \frac{k_{in}^i}{k_{in}^i + k_{out}^i} \tag{3}$$

where k_{in}^i and k_{out}^i are the in- and out-degree of node n_i, respectively. A greater value of S_i means that n_i has a smaller out-degree and therefore the changes of other nodes are less likely to propagate to n_i. Thus we say a node with a greater S_i is more stable. The value of S_i is always in the range of [0, 1].

The instability I_{ij} of the edge $< n_i, n_j >$ is derived from node stability and defined by Eq. (4).

$$I_{ij} = S_i - S_j \tag{4}$$

where S_i and S_j are the node stability of nodes n_i and n_j, respectively. With this definition, an edge with a greater I_{ij} propagates changes to more nodes. Thus we call I_{ij} as edge instability. The edge instability, *EI*, of a graph is the average I_{ij} of the edges in the graph.

Figure 3 includes two examples to explain this observation.

1. In Fig. 3(a), S_i is 3/4 and S_j is 1/4. Accordingly, I_{ij} of Edge $< i, j >$ is 1/2. A change at Nodes j, a, b or c will propagate to Nodes i, d, e, and f across Edge $< i, j >$, as indicated by the dotted lines.
2. In Fig. 3(b), S_i is 0 and S_j is 1/4. Accordingly, I_{ij} of Edge $< i, j >$ is -1/4. Changes at Nodes j, a, b, and c only spread to Node i but do not reach nodes d, e, and f.

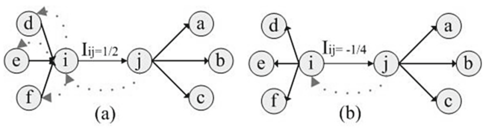


Fig. 3. Comparison between the effects of two edges with the higher and lower I_{ij} for the neighboring nodes

Apparently, Edge $< i, j >$ with higher EI in the first example has a higher impact on the propagation scope.

2.3 Statistics with Six Open Source Software

For a node in a call graph, the out-degree is the number of other functions that the function represented by this node calls, and the in-degree is the number of other functions that call the function of this node. The sum of the in-degree and out-degree is the degree of the node. We calculated average degree of the nodes in the call graphs of the selected systems. We found that the average degrees stayed stable as the software systems evolve over many versions. For each call graph, we measure the diameter, denoted as D.

Clustering, the tendency that a node's neighbors are likely to be neighbors themselves, has been commonly considered as one of the most important factors in the study of propagation [38]. We calculated the clustering coefficient (CC) in C_i of Node i using Eq. (5):

$$C_i = \frac{2E_i}{k_i(k_i - 1)} \tag{5}$$

where E_i is the number of edges connecting neighbors of Node i, and k_i is the degree of Node i. The clustering coefficient C of a graph is the average clustering coefficient of the nodes in the graph.

The number of nodes, the number of edges, the average node degree, the diameter (D), the clustering coefficient (CC), the edge instability (EI) and the propagation scope (PS_G) of six selected systems are showed in Fig. 4. Figure 4 shows that for all the systems both the number of nodes and the number of edges grow over time. A number of observations can be made as below.

1. Table 2 summarizes the correlation coefficient results between PS_G and other features of six software. Compared with the number of nodes, the number of edges, the average node degree, the diameter (D) and the clustering coefficient, the edge instability is the only one that had a positive correlation with PS_G in all six software.
2. As the increase of node number, the corresponding PS_G decreases except for BIND. In fact, PS_G is the ratio of the number of nodes a propagation reaches to the number of all nodes. The propagation of changes in a system is hard to maintain the same rate as a software system becomes larger.

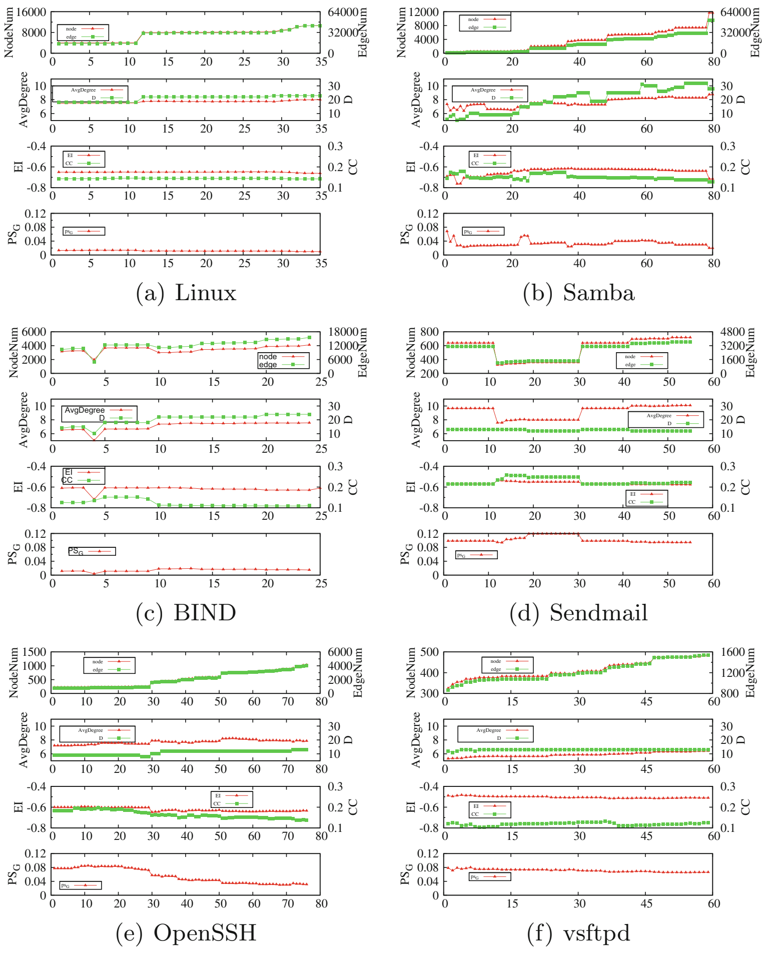


Fig. 4. The number of nodes, the number of edges, the average degree, CC, EI and PS_G of six software. X axis indicates the sequence numbers of versions.

Table 2. The correlation coefficient results between PS_G and other features.

Software	Node	Edge	Ave. deg	D	CC	EI
Linux	−0.99	−0.99	−0.96	−0.96	0.42	**0.41**
Samba	−0.11	−0.11	0.086	0.05	0.03	**0.29**
BIND	0.29	0.58	0.94	0.85	−0.75	**0.48**
Sendmail	−0.80	−0.80	−0.79	−0.37	0.76	**0.68**
OpenSSH	−0.96	−0.96	−0.85	−0.94	0.97	**0.94**
vsftpd	−0.93	−0.92	−0.91	−0.26	−0.21	**0.90**

3. Per the definition of PS, as the increases of the average degree, PS of a software system would also increase. This phenomenon can be observed clearly in BIND system (Fig. 4(c)). It is easy to understand that the more edges a system has, the faster the propagation would be. The tendency, however, is not apparent in other software systems. It is difficult to tell the impact of the average degree on PS_G in two cases: 1. The average degree is very stable. 2. The average degree and node number change at the same time.
4. Previous works suggest that as CC increases the propagation scope in a network would decrease [38]. This can be clearly observed in BIND and vsftpd. It is not apparent, however in other systems as CC is very stable in different versions.

3 Connectivity

In complex networks, researchers often study robustness by measuring connectivity after removing nodes or edges [10,32]. As many error conditions do not cause the crash of the whole software system, we assume that the other parts of the software keep working. For example, when the kernel panics in a loadable module of an Ethernet driver it can contain the failure and give out messages. The other parts of the system cannot use this driver but may be able to access the Ethernet device from other channels and certainly a user can continue to work in a text editor. Thus, we adopted the same methodology to evaluate the connectivity of call graphs in this paper.

We remove the edge to simulate the failure, and study how well the other nodes in the call graph stay connected. Connectivity among the nodes left represents a measure of robustness of the graph under edge removal. It is not a measure of how well the software handles crashes but how well its functions are designed and coded to minimize the impact on the rest of the system when one or more parts fail.

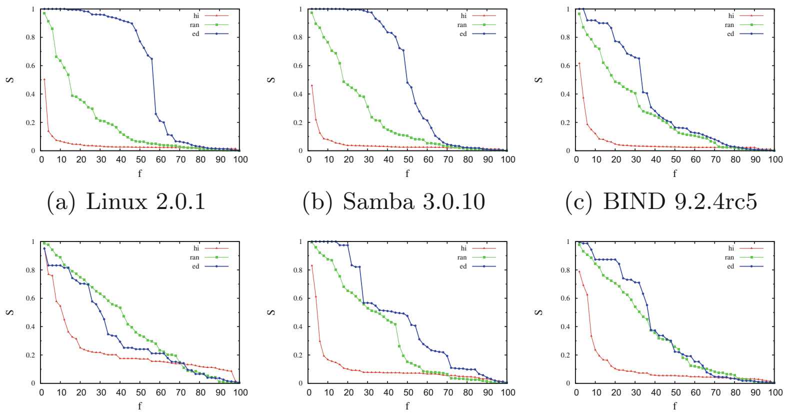
We try three strategies to attack generated call graphs. The first strategy, the "RA removal", removes edges randomly. The second removes edges with higher EIs and is called the "HL removal". The third strategy is proposed by Ref. [19] where the edge degree is defined by Eq. (6):

$$k_e = k_v * k_w \tag{6}$$

where the edge e connects two nodes v and w with node degrees k_v and k_w, respectively. The attack strategy selects the edges in descending degrees. We name this strategy as "ED removal".

We select Linux 2.0.1, Samba 3.0.10, BIND 9.2.4rc5, Sendmail 8.11.3, OpenSSH 2.5.2p1 and vsftpd 1.1.2 to compare the three strategies. Each time, 5 percent of edges are removed. We measure the change of the size of the maximal connected subgraph, S, with Eq. (7):

$$S = \frac{N_a}{N_i} \tag{7}$$



(a) Linux 2.0.1 (b) Samba 3.0.10 (c) BIND 9.2.4rc5

(d) Sendmail 8.11.3 (e) OpenSSH 2.5.2p1 (f) vsftpd 1.1.2

Fig. 5. Six software under three attack strategies.

where N_i and N_a are the numbers of nodes in the maximal connected subgraph before and after the attack, respectively.

The results are shown in Fig. 5 when a fraction f of the edges is removed in six software. In Fig. 5, it is difficult to distinguish the impact of the connectivity between the RA and ED removal. When the call graphs undergo the HL removal attacks, however, the sizes of maximal connected subgraph decreases rapidly. We observed the top 20 edges with high *EI* in Linux and found the similar structure as Fig. 6 when we extract all edges that directly connect to the edge with high *EI* from call graph. The edges with high *EI* behave like some kind of "weak ties" [15] between two parts of software modules. Removing these edges results in quick disintegration of the call graphs.

4 Features of Call Graphs and Evolution Model

To understand how software systems evolve into particular structures, we studied the ways new nodes were added into call graphs.

4.1 Preferential Attachment

Call graphs show that their in-degree distribution largely obeys the power-law while the out-degree exhibits the power-law distribution with a cutoff [36]. Thus, we investigated whether new nodes would contribute to the distributions of in-degree and out-degree. Specifically, for in-degree distribution we would like to know whether new nodes are more likely to be connected to existing nodes of higher in-degrees (in terms of function calls, the functions corresponding to the new nodes are the callers in this case). For out-degree distribution, we would like to know whether existing nodes of higher out-degree are more likely to be connected to the new nodes (the functions corresponding to the new nodes are the

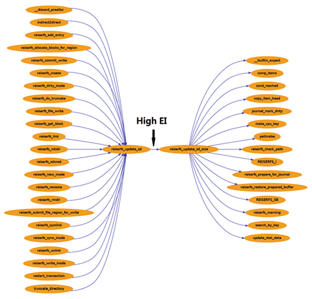


Fig. 6. The edge with high *EI*.

callees in this case). The tendency for new nodes to be connected to either high in-degree/out-degree nodes is commonly known as "preferential attachment". Preferential attachment has been considered as an important factor contributing to the scale-free feature of complex networks [5]. We would like to know whether in-degree and out-degree preferential attachments exist when the new nodes are added.

We studied how the newly added nodes are connected to the top 5 % nodes with the highest in-degrees and the top 5 % nodes with the highest out-degrees in each software version. To quantify the preferential attachment tendency, we define the concept of connecting probability (*CP*) as the probability that a new node is connected to the top 5 % nodes with the highest in-degree/out-degrees. Formally, *CP* can be calculated using Eq. (8):

$$CP = \frac{N_c}{N_t} \tag{8}$$

where N_c is the number of new nodes that call the top 5 % nodes with the highest in-degrees when we consider in-degree preferential attachment (the resulting *CP* is called in-degree *CP*) and N_t is the total number of new nodes in the corresponding version. When we consider out-degree preferential attachment, N_c is the number of new nodes that are called by the top 5 % nodes with the highest out-degrees and N_t stays the same (the resulting *CP* is called out-degree *CP*). The average in-degree and out-degree *CP* are summarized in Table 3. Table 3 indicates that the in-degree *CP* is consistently large for each system, which suggests high in-degree preferential attachment behaviors in all systems. Table 3 also shows that the out-degree *CP* is consistently lower than the corresponding in-degree *CP* for each version. The out-degree preferential attachment tendency, if any, seems much weaker than that of the in-degree preferential attachment.

4.2 Callers of New Nodes

In Sect. 4.1, the out-degree *CP* results indicate that the top 5 % nodes with the highest out-degrees do not call the new nodes extensively. It is interesting to

Table 3. Average connecting probability.

Software	Linux	Samba	BIND	Sendmail	OpenSSH	vsftpd
Avg. In-degree	73.06 %	60.28 %	60.36 %	60.66 %	62.46 %	64.65 %
Avg. Out-degree	4.43 %	12.35 %	5.79 %	6.99 %	10.91 %	34.32 %

know which nodes call the new nodes the most. We used N_{new} and N_{old} to denote the number of the new nodes that call the new nodes and the number of the old nodes that call the new nodes in each version, respectively. We use R_{new} to quantify the ratio of N_{new} to $(N_{new}+N_{old})$. It can be calculated using the Eq. (9):

$$R_{new} = \frac{N_{new}}{N_{new} + N_{old}} \tag{9}$$

Thus, we can obtain R_{new} for each software version. The average R_{new} of the Linux, Samba, BIND, Sendmail, OpenSSH, and vsftpd are 47.3 %, 38.6 %, 37.6 %, 23.4 %, 41.0 %, and 29.1 %, respectively. Note that the average ratio of the number of new nodes to that of old nodes for all Linux versions is only 1.8 %. These results indicate that compared with old nodes, new nodes are more likely to call new nodes. In other words, the "age" of a node is one factor to determine whether the node calls another node in real-life software, which ultimately will have an impact on the degree distribution.

4.3 Evolution Model for Software

The "age" of a node seemed to be critical to determine the likelihood for connecting new nodes. With these observations and analysis, we propose a novel model to compare the impact of *CC* and *EI* on propagation, in which the two properties can be tuned by changing some parameters. We build our model based on Barabási and Albert (BA) model [6,18] and extend it to adjust clustering coefficient and edge instability as follows:

1. In the beginning, a network consists of m_0 nodes and no edges. m_0 is a small integer. In our experiments it is set to 3.
2. Add Node v ($v = m_0 + 1$ intially).
3. Repeat the following two steps for Node v until m edges are added.
   (a) Preferential attachment (PA): Each edge of Node v is then attached to an existing node with the probability proportional to its degree and age, i.e., the probability for Node w (w = 1, 2, ..., m_0 + v - 1) to be attached to v is

$$P_w = \frac{k_w}{\sum_{i=1}^{m_0+v-1} k_i} \tag{10}$$

   where $k_i = (age_i)^{-\beta} * k_i^{'}$. age_i, with the initial value of 1, represents the age of Node i. When a new node is added to the network, the age of each existing node is incremented by 1. $k_i^{'}$ is the degree of Node i. β controls the influence of a node's age.

(b) Triad formation (TF): If an edge between Nodes v and w was added in the previous PA step, then add one more edge from Node v to a randomly chosen neighbor of Node w. If all neighbors of Node w have been connected to Node v, go back to Step 3(a).

4. If $v < |N| - m_0 + 1$, increment v by 1 and go back to step 2. Otherwise, the network is generated.

In generating the network, the total number of the PA and TF steps that produces edges for each new node is m. After one PA, we perform a TF step with a probability of P_t. P_t is a parameter that adjusts the CC of the generated network.

Equation (2) shows that the diameter of a graph has a major impact on PS_G. When a new edge does not cause the current diameter to exceed a threshold, LD, the edge is added. Otherwise, the new edge is dropped and a new possible edge is selected to repeat the above step.

Following [34], we first generate undirected graphs and then transform them into directed graphs by making edges to start from the newly added nodes and end at the existing nodes.

There are five parameters in our model (N, m, P_t, β, and LD). In the experiments, we used $N = 10000$ and $m = 3$. To understand the effect of the P_t and β, we select P_t and β in the ranges of 0 to 0.8 and 0 to 1.0, respectively. Figure 7 showed the relationship among PS_G, CC and EI with the different P_t and β parameters of the generated graphs by our model. We have some observations:

- P_t correlated positively with CC as shown in Fig. 7 (a) and (d).
- β correlated positively with EI as shown in Fig. 7 (b) and (e).
- β, which determines PS_G of a network, has a larger impact on the propagation than P_t, as shown in Fig. 7 (c) and (f). Thus, in our model, the propagation scope was highly correlated with β and therefore EI but less affected by the changes in CC.

To study the impact of EI on connectivity, we use the three strategies to attack graphs generated by different βs. Figure 8 summarized the results. Figure 8(a) showed that, under different βs (between 0 and 1), the effect of node age on network structure changed significantly. The effect of the node ages limited the preferential attachment and prevented a scale-free distribution of connectivities [4]. It seemed to be a critical point in a network and the network became scale-free when β was below a certain value. With large βs, the network showed no power law characteristics and the degree became exponentially distributed. As shown in Fig. 8(a), overall and for different βs, the networks had a good connectivity. This result was consistent with the experiments on network connectivity [3]. This suggested that the network was connected under the random attack due to the scale-free feature. When β fell between 0.4 and 0.8, however, the degree distribution of network transited from scale-free to exponential and the network more vulnerable to random attacks.

Figure 8(b) showed a fast decay for different values of β and f between 0.2 and 0.3. The network was less connected and therefore it became "harder" for

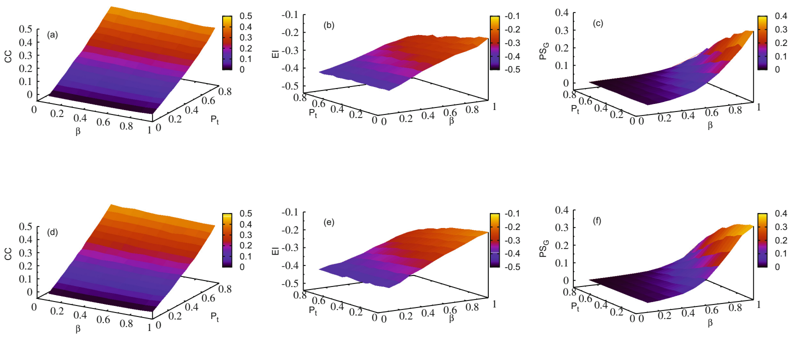


Fig. 7. Summary of PS_G, clustering coefficient and edge instability under different parameters. (a) CC with P_t and β, LD is 20. (b) EI with P_t and β, LD is 20. (c) PS_G with P_t and β, LD is 20. (d) CC with P_t and β, LD is 25. (e) EI with P_t and β, LD is 25. (f) PS_G with P_t and β, LD is 25.

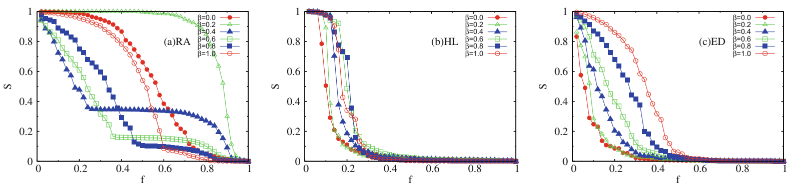


Fig. 8. Three attack strategies applied to the graphs generated with P_t 0.30, LD 20 and β from 0 to 1. (a) The RA attack strategies. (b) The HL attack strategies. (c) The ED attack strategies.

the remaining nodes to communicate with each other when the unstable edges (the edges with the EI values close to 1) were removed.

Figure 8(b) with 8(c) showed that when β is small the HL and ED removals had a similar effect for network connectivity. When β is large, however, the HL removal hurt the connectivity more than ED. As β increased the distribution of degree morphs from scale-free to exponential. Under exponential degree distribution, each node in the network has approximately the same degree, and therefore the damage to network connectivity by ED removal was less obvious. In HL removal, however, the EI highly affected the propagation of networks when β increased (see Fig. 7) and the edges with high EI had a structure of "weak ties" (see Fig. 6). Thus, removing the edges with high EIs in HL removal broke the network into small pieces fast for all β values. HL instead of ED removal became a good indicator for attacks when the degree distribution of networks shifts from scale-free to exponential.

5 Related Work

Two main approaches to study change propagation in software are Impact Analysis (IA) and Mining Software Repositories (MSR). IA uses dependency or traceability information and MSR uses historical information [21]. An in-depth review of impact analysis for software change can be found in [24]. Mirarab et al. [27] propose to use BBNs for impact analysis, and their approach achieves a precision of 63 % with a recall of 26 %. Formal Concept Analysis (FCA) [14], probabilistic approach [33], and Family Dependence Graph (FDG) [39] have been employed for change impact analysis. In the context of complex technical systems, Giffin et al. [13] analyze change propagation with design structure matrix (DSM) and categorize a number of typical change patterns. Recently, Zhang et al. [40] use requirement dependency as a tool to conduct change propagation analysis. They investigate whether existing dependency types are sufficient for change propagation analysis. In our work, we analyze the impact of software network structures on the change propagation without tracing the affected functions in software.

Hassan and Holt propose to determine how changes propagate with developer information, historical co-change information of entity, code structure, and code layout heuristic [16]. Hassan and Malik further improve the approach with an adaptive heuristic method [17]. Zimmermann et al. [42] apply data mining techniques to analyze version histories, and to uncover couplings between fine-grained entities to guide programmers among related changes. Gall et al. [12] propose an approach to extract software evolution patterns and dependencies from the CVS data. Their proposed methodology, QCR, is to examine the historical development of classes by analyzing changes of classes and common change behavior obtained from CVS. They further [11] classify changes according to their significance levels (low, medium, high, or crucial). In our work, we explain why asymmetric structures inside the software are formed after analyzing call graphs of many software system versions, and propose an evolution model capturing the way new nodes are added during the process of software evolution. By varying the parameters of the proposed model, we study the relationship between the change propagation and software structure.

Asymmetric structures can increase the fragility of software [8]. Studies on the mechanism of asymmetric software structures can help optimization of software. Myers points out that the asymmetry typically is due to the common practice of software reuse [29]. Others [34] argue that the asymmetry is rooted right in the economization of development effort and related costs. Additionally [9] notes that the out-degree of the class in the object oriented software systems is limited by the size of the class. They have found that the limitation leads to the asymmetry of degree distributions. We found, however, that the manner in which new nodes are added into the call graphs during software evolution contributes to the asymmetric feature of software, and that the asymmetric structures have a significant impact on the change propagation of software.

Many network models have been proposed in the past decade. In BA model [5], each newly added node is connected to nodes selected with a probability proportional to their respective degrees. Although the degree distribution of the BA model follows the power-law distribution, the resulting clustering coefficient is much lower than those measured in software systems. In the Copying model [22], each newly added node randomly selects a target node and connects to it, as well as to all neighbor nodes of the target node. The Copying model has a large clustering coefficient. In the process of selecting the target node in the Copying model, however, older nodes have greater priority to be connected to the newly added nodes. This feature is different from actual new node behavior demonstrated in real software systems, in which new nodes are more likely to be connected among themselves.

6 Conclusions

Using propagation scope, we quantified change propagation in different versions of six open-source software systems. Inspired by the asymmetry of degree distribution and direction of change propagation in software, we proposed the edge instability to measure the change propagation of a call graph. We calculated the number of nodes, the number of edges, the average node degree, the diameter, *CC*, *EI* in the call graphs of the selected systems, and found that *EI* is the only one that had a positive correlation with propagation scope in all six studied software. To compare the impact of *CC* and *EI* on change propagation, we have proposed a novel model allowing us to adjust these properties. Although *CC* is traditionally considered one of the most important factors in the study of propagation, our experimental results indicated that *EI* had a much bigger impact in call graphs. Furthermore, we showed the correlation between the connectivity of call graphs and *EI*, i.e., eliminating the edges with high *EI* breaks a network into small pieces faster in real software and networks generated by our model. In summary, we demonstrated that *EI* could be a good indicator of the change propagation and connectivity of software networks.

Acknowledgments. This work was supported by National Natural Science Foundation of China (No. 61272167).

References

1. ctags 5.8 release. http://ctags.sourceforge.net
2. Gcc 3.4 release series. http://gcc.gnu.org/gcc-3.4/
3. Albert, R., Jeong, H., Barabósi, A.: Error and attack tolerance of complex networks. Nature **406**(6794), 378–382 (2000)
4. Amaral, L., Scala, A., Barthélémy, M.: Classes of small-world networks. Proc. Nat. Acad. Sci. **97**, 11149–11152 (2000)
5. Barabási, A.L., Albert, R.: Emergence of scaling in random networks. Science **286**, 509–512 (1999)
6. Barabási, A.L., Albert, R.: Emergence of scaling in random networksscience. Science **286**(5439), 509–512 (1999)
7. Bhattacharya, P., Iliofotou, M., Neamtiu, I., Faloutsos, M.: Graph-based analysis and prediction for software evolution. In: ICSM, pp. 419–429 (2012)
8. Challet, D., Lombardoni, A.: Bug propagation and debugging in asymmetric software structures. Phys. Rev. E **70**, 046109 (2004)
9. Concas, G., Marchesi, S.P.M., Serra, N.: Powerlaws in a large object-oriented software system. IEEE Trans. Softw. Eng. **33**(10), 687–708 (2007)

10. Crucittia, P., Latorab, V., Marchiori, M., Rapisarda, A.: Error and attacktolerance of complex networks. Phys. A **340**, 388–394 (2004)
11. Fluri, B., Gall, H.C.: Classifying change types for qualifying change couplings. In: ICPC, pp. 35–45 (2006)
12. Gall, H., Jazayeri, M., Krajewski, J.: Cvs release history data for detecting logical couplings. In: IWPSE (2003)
13. Giffin, M., de Weck, O., Bounova, G., Keller, R., Eckert, C., Clakson, J.: Change propagation analysis in complex technical systems. J. Mech. Des. **131**(8), 081001-1–081001-14 (2009)
14. Girba, T., Ducasse, S., Kuhn, A.: Using concept analysis to detect co-change patterns. In: Ninth International Workshop on Principles of Software Evolution: In conjunction with the 6th ESEC/FSE, pp. 83–89 (2007)
15. Granovetter, M.: The strength of weak ties. Am. J. Socio. **78**(6), 1360–1380 (1973)
16. Hassan, A.E., Holt, R.C.: Predicting change propagation in software systems. In: International Conference on Software Maintenance, pp. 284–293 (2004)
17. Hassan, A.E., Malik, H.: Supporting software evolution using adaptive change propagation heuristics. In: ICSM, pp. 177–186 (2008)
18. Holme, P., Kim, B.: Growing scale-free networks with tunable clustering. Phys. Rev. E **65**(2), 026107 (2000)
19. Holme, P., Kim, B., Yoon, C.: Attack vulnerability of complex networks. Phys. Rev. E **65**(2), 056109 (2002)
20. Ichii, M., Matsushita, M., Inoue, K.: An exploration of power-law in use-relation of java software systems. In: 19th Australian Software Engineering Conference, pp. 422–4311 (2008)
21. Kagdi, H., Maletic, J.: Software-change prediction: estimated+actual. In: Software Evolvability, pp. 38–43 (2006)
22. Krapivsky, P.L., Redner, S.: Network growth by copying. Phys. Rev. E **71**(3), 036118 (2005)
23. LaBelle, N., Wallingford, E.: Inter-package dependency networks in open-source software. CoRR, cs.SE/0411096 (2004)
24. Lehnert, S.: A review of software change impact analysis. Technical University Ilmenau, pages Report ilm1-200618 (2011)
25. Liu, J., Lu, K.H.J., Li, B., Tse, C.: Characterizing the structural quality of general complex software networks. Int. J. Bifurcat. Chaos **18**(02), 605–613 (2008)
26. MacCormack, A., Rusnak, J., Baldwin, C.: Exploring the structure of complex software designs: an empirical study of open source and proprietary code. Manag. Sci. **52**(7), 1015–1030 (2006)
27. Mirarab, S., Hassouna, A., Tahvildari, L.: Using bayesian belief networks to predict change propagation in software systems. In: ICPC, pp. 177–188 (2007)
28. Moore, C., Newman, M.E.J.: Epidemics and percolation in small-world networks. Phys. Rev. E **61**(5), 5678 (2000)
29. Myers, C.R.: Software systems as complex networks: structure, function, and evolvability of software collaboration graphs. Phys. Rev. E **68**, 046116.1–046116.15 (2003)
30. Newman, M.E.J.: The structure and function of complex networks. SIAM Rev. **45**, 167–256 (2003)
31. Potanin, A., Noble, J., Frean, M., Biddle, R.: Scale-free geometry in oo programs. Commun. ACM **48**(5), 99–103 (2005)
32. Albert, A.B.R., Jeong, H.: Error and attack tolerance of complex networks. Nature **406**, 378–382 (2000)

33. Sharafat, A.R., Tahvildari, L.: Change prediction in object-oriented software systems: a probabilistic approac. J. Softw. **3**(5), 26–39 (2008). (1796217X)
34. Valverde, S., Cancho, R.F., Solé, R.V.: Scale-free networks from optimal design. Europhys. Lett. **60**, 512–517 (2002)
35. Valverde, S., Solé, R.V.: Hierarchical small worlds in software architecture. cond-mat/0307278 (2003)
36. Wang, L., Wang, Y., Zhao, Y.: Mechanism of asymmetric software structures: a complex network perspective from behaviors of new nodes. Phys. A Stat. Mech. Appl. **413**, 162–172 (2014)
37. Wang, L., Wang, Z., Yang, C., Zhang, L., Ye, Q.: Linux kernels as complex networks: a novel method to study evolution. In: ICSM, pp. 41–50 (2009)
38. Wu, X., Liu, Z.: How community structure influences epidemic spread in social networks. Phys. A **387**, 623–630 (2008)
39. Yazdanshenas, A.R., Moonen, L.: Fine-grained change impact analysis for component-based product families. In: ICSM, pp. 119–128 (2012)
40. Zhang, H., Li, J., Zhu, L., Zhu, L., Jeffery, R., Liu, Y., Wang, Q., Li, M.: Investigating dependencies in software requirements for change propagation analysis. Inf. Softw. Technol. **56**(1), 40–53 (2014)
41. Zhou, T., Yan, G., Wang, B.: Maximal planar networks with large clustering coefficient and power-law degree distribution. Phys. Rev. E **71**(4), 046141 (2005)
42. Zimmermann, T., Zeller, A., Weissgerber, P., Diehl, S., Zeller, A.: Mining version histories to guide software changes. IEEE Trans. Softw. Eng. **31**(6), 429–445 (2005)