ICIC Express Letters
Part B: Applications
Volume **7**, Number **2**, February **2016**

ICIC International ©2016 ISSN 2185-2766
pp. **257–264**

IDENTIFYING IMPORTANT NODES IN COMPLEX SOFTWARE NETWORK BASED ON RIPPLE EFFECTS

Jiadong Ren[1,2], Hongfei Wu[1,2,*], Rui Gao[1,2], Guoyan Huang[1,2] and Jun Dong[1,2]

[1]College of Information Science and Engineering
Yanshan University
[2]Computer Virtual Technology and System Integration Laboratory of Hebei Province
No. 438, Hebei Avenue, Qinhuangdao 066004, P. R. China
*Corresponding author: theworkone@sina.com

Received July 2015; accepted October 2015

Abstract. *The structure of complex software systems can be extracted as a complex software network, and the quality of software systems largely depends on the topological structure of the software network. A small portion of important nodes plays a critical role in ensuring the stability and quality of the software system. Identifying these important nodes in software network has become an important and challenging task of software engineering. In this paper, firstly we construct a model of directed weighted software network. Then, by analyzing the bug propagation process on weighted software network, the values of some metrics related to ripple effects are calculated. Finally, important nodes whose ripple effects are larger than average ripple effect are identified by our proposed approach. Experimental results on real software data show the utility of the approach.*
Keywords: Complex software network, Important node, Bug propagation, Ripple effect

1. **Introduction.** In modern society, many of our daily activities are dependent on the correct working of software systems. Therefore, ensuring the reliability and stability of software system is becoming an important and challenging task of software engineering.

The latest research implies that the quality of a software system is highly affected by a small portion of important nodes [1, 2]. Identifying these important nodes has remarkable significance for predicting the quality of software early and providing guidance for software structure design and optimization.

Myers [3] firstly adopted complex software networks to represent software systems. The nodes and edges correspond to software components and interactions between each component respectively. Callaw et al. [4] conducted an analysis of nodes' degree and pointed out that a node with larger degree was more important than others, while the method is low-relevant to global structure information and may come to an unsatisfactory result. Considering a node that connects two communities in a network, it does not have a large degree, but it is very important because the two communities will not connected if the node is failed. Li et al. [5] introduced a model of weighted software network to represent a software's structure and the importance of a specific class is defined as the average number of nodes that have been affected by the faulty node. Chen et al. [6] proposed a local centrality measure which considers both the nearest and the next nearest neighbors of a node, while the research was conducted in undirected and unweighted networks and it is not appropriate in the research of identifying important nodes in directed weighted software network. He et al. [7] adopted relative node measurement score (RNMS) to measure the importance of nodes in software execution network and treated nodes whose RNMS equals median of all nodes' RNMS as important nodes. Ren et al. [1] proposed a key node mining approach based on weighted semi-global structure information (KNMWSG)

to mine key function nodes in directed weighted software network. To some extent, it is neither low-relevant to global information nor time-consuming. However, only taking part of the structure information may lose some important information inevitably. The concept of ripple degree proposed in [8] can be used to identify important nodes, i.e., these nodes which have large ripple effects have a significant influence on the reliability and stability of a software system. However, the ripple effect of a node was just defined as the size of its reachable nodes set, which does not take reachable probability into account. The other shortcoming is that it did not differentiate whether a node is a maker of the ripple effect, or an affected object of the ripple effect. Approach Software Network Key Node miNing (SN-KNN) mentioned in [9] mines key function callers and key function callees respectively. However, the weight of directed edges among a node's neighbors is neglected.

In fact, in the directed weighted software network, if a node fails, it can affect other nodes with a certain probability (a maker of ripple effect). On the other hand, if a node is affected by other failed nodes with a certain probability, it is an affected object. Therefore, in this paper, we study two different cases: forward weighted ripple effect and reverse weighted ripple effect (detailed definitions are given in Section 2). The nodes that have larger forward ripple effect than average forward ripple effect are deemed as important fragile nodes. On the other hand, the nodes that have larger reverse ripple effect than average reverse ripple effect are deemed as important rigescent nodes.

The major contributions of this paper can be summarized as follows.

- Some practical metrics are defined to measure the importance of nodes within a software network.
- Important fragile nodes and rigescent nodes are identified by our proposed approach.

This paper is arranged as follows. Section 2 provides preliminary definitions. Section 3 is a detailed description of our approach. Section 4 shows the experiment results, followed by conclusions and future work in Section 5.

2. Preliminary Definitions.

In this section, we describe preliminary details about some metrics related to ripple effects.

Definition 2.1. Forward Weighted Ripple Effect (ForWere). *Given a node v_i, $NSF(v_i)$ is the nodes set that is reachable from v_i (If there is a directed path of arbitrary length from node v_1 to node v_2, then we say v_2 is reachable from v_1). $PSE(v_i, v_j)$ is a paths set that v_i is the source node and node v_j is the end node of each path. Notation P_n means that if node v_j fails, it will affect node v_i with probability P_n via $path_n$. Notation $W_{v_s \to v_t}$ is the weight of directed edge$< v_s, v_t >$.*

$$ForWere(v_i) = 1.0 + \sum_{v_j \neq v_i}^{v_j \in NSF(v_i)} \sum_{n=1}^{|PSE(v_i,v_j)|} P_n \tag{1}$$

$$P_n = \prod_{v_s \neq v_t}^{v_s \to v_t \in PSE(v_i,v_j)} W_{v_s \to v_t} \tag{2}$$

A node v_i of course would be affected by itself if it fails; therefore, we add 1.0 to $ForWere(v_i)$ in Equation (1). A large value of $ForWere(v_i)$ indicates that node v_i may be affected by other nodes with a large probability, implying it is fragile.

Definition 2.2. Reverse Weighted Ripple Effect (ReWere). *For a node v_i, $NST(v_i)$ is the nodes set that is reachable to v_i. $PSE(v_j, v_i)$ is a paths set that v_j is the source node and node v_i is the end node of each path. Notation P_n means that if node v_i fails, it*

will affect node v_j with probability P_n via $path_n$.

$$ReWere(v_i) = 1.0 + \sum_{v_j \neq v_i}^{v_j \in NST(v_i)} \sum_{n=1}^{|PSE(v_j,v_i)|} P_n \tag{3}$$

$$P_n = \prod_{v_s \neq v_t}^{v_s \to v_t \in PSE(v_j,v_i)} W_{v_s \to v_t} \tag{4}$$

A node v_i with large value of *ReWere* indicates that if it fails, it may affect other nodes with a large probability, implying it is rigescent.

Definition 2.3. Average Weighted Ripple Effect (AvgWere). *AvgWere of a software network is defined as the following formulas, in which $N_{ForWere=1.0}$ refers to the number of nodes whose ForWere is 1.0 (out-degree is 0) and $N_{ReWere=1.0}$ refers to the number of nodes whose ReWere is 1.0 (in-degree is 0).*

$$AvgForWere = \frac{\sum_{i=1}^{|V|} ForWere(v_i) - N_{ForWere(v_i)=1.0}}{|V| - N_{ForWere(v_i)=1.0}} \tag{5}$$

$$AvgReWere = \frac{\sum_{i=1}^{|V|} ReWere(v_i) - N_{ReWere(v_i)=1.0}}{|V| - N_{ReWere(v_i)=1.0}} \tag{6}$$

Definition 2.4. Important Node (ImpND). *If $ForWere(v_i)$ is larger than AvgForWere, then node v_i is deemed as an important fragile node. Similarly, if $ReWere(v_i)$ is larger than AvgReWere, then node v_i is deemed as an important rigescent node.*

3. **Important Node Mining Approach.** The framework of our approach works in a four-step process. Firstly, the model of directed weighted function dependency network (DWFDN) is constructed by taking functions as nodes and function dependency relationships as directed edges. The detailed construction process is given in our previous work [9]. Secondly, the network is traversed with Depth-First-Search strategy to obtain all possible execute paths. Next, values of metrics defined in Section 2 are calculated. Finally, important nodes are identified by our approach. Figure 1 shows the work flow of our approach.

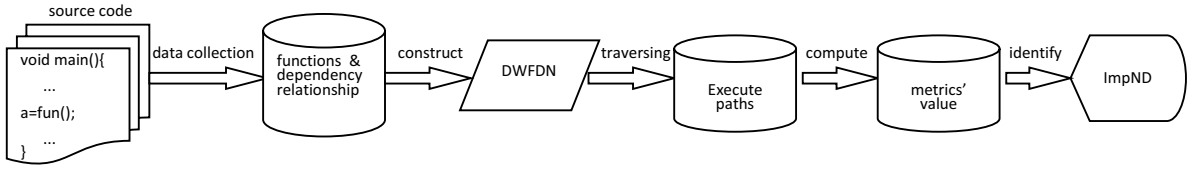


FIGURE 1. Framework of our research

Taking the process of identifying important fragile node as demonstration, Algorithm 1 is proposed to compute the *ForWere* of each node v_i in software network. *HMP* is a map (a data structure in JAVA language), in which the map's key corresponds to the paths, and the corresponding value of a key is the product of edges' weight. *EPS* is a data set storing all possible execute paths. *Sub-EPS* is a subset of *EPS*. Line 3 prunes away the execute paths that do not contain node v_i, leaving a subset *Sub-EPS*. For an execute path $ep \in$ *Sub-EPS*, the nodes after the index of node v_i are extracted (node v_i may be affected by these nodes, line 5 to line 7). All keys and corresponding values of map *HMP* are identified (Line 8 to Line 18). $ForWere(v_i)$ is obtained by totalizing all the products (Line 20 to line 22).

Algorithm 1: calculateForWere(v_i, EPS)

Input: v_i, EPS
Output: ForWere(v_i)

```
1: HashMap<String, Double> HMP = new HashMap<String, Double>()
2: initialize HMP.put<v_i, 1.0>  // affected by node v_i itself
3: get a Sub-EPS of EPS that each execute path ep ∈ Sub-EPS contains v_i
4: for each ep ∈ Sub-EPS do
5:    int index = index of v_i in the ep
6:    Sub-ep = ep.substring(index)
7:    initialize an array NodeArr with the nodes contained in Sub-ep
8:    TempPath = PreNode = v_i
9:    for each node v_j in NodeArr do  //v_j is reachable from v_i
10:      PreTempPath = TempPath
11:      Insert node v_j into the tail of string TempPath //extending a path step by step
12:      if (!HMP.containsKey(TempPath)) then
13:         PreValue = HMP.get(PreTempPath)
14:         NewValue = PreValue*W_{ProNode→v_j}
15:         HMP.put(TempPath, NewValue)
16:      end if
17:      PreNode = v_j
18:   end for
19: end for
20: for each entry in HMP do
21:   ForWere(v_i) += entry.getValue()
22: end for
23: return ForWere(v_i)
```

Example 3.1. *Take Figure 2 as an example. The weight of each directed edge is computed according to the method mentioned in our previous work* [9]. *The software network's all possible execute paths are obtained by traversing the network based on Depth-First-Search strategy. For node* v_2, $NSF(v_2) = \{v_3, v_4, v_5, v_6, v_7, v_8\}$, $PSE(v_2, v_3) = \{v_2 \to v_3\}$, $PSE(v_2, v_4) = \{v_2 \to v_4\}$, $PSE(v_2, v_5) = \{v_2 \to v_4 \to v_5\}$, $PSE(v_2, v_6) = \{v_2 \to v_3 \to v_6, v_2 \to v_4 \to v_5 \to v_6\}$, $PSE(v_2, v_7) = \{v_2 \to v_3 \to v_6 \to v_7, v_2 \to v_4 \to v_5 \to v_6 \to v_7\}$, $PSE(v_2, v_8) = \{v_2 \to v_3 \to v_6 \to v_8, v_2 \to v_4 \to v_5 \to v_6 \to v_8\}$. *According to Equation (1) and Equation (2),* $ForWere(v_2) = 1.0 + (W_{v_2 \to v_3} + W_{v_2 \to v_4} + W_{v_2 \to v_4} * W_{v_4 \to v_5} + (W_{v_2 \to v_3} * W_{v_3 \to v_6} + W_{v_2 \to v_4} * W_{v_4 \to v_5} * W_{v_5 \to v_6}) + (W_{v_2 \to v_3} * W_{v_3 \to v_6} * W_{v_6 \to v_7} + W_{v_2 \to v_4} * W_{v_4 \to v_5} * W_{v_5 \to v_6} * W_{v_6 \to v_7}) + (W_{v_2 \to v_3} * W_{v_3 \to v_6} * W_{v_6 \to v_8} + W_{v_2 \to v_4} * W_{v_4 \to v_5} * W_{v_5 \to v_6} * W_{v_6 \to v_8})) = 2.0$.

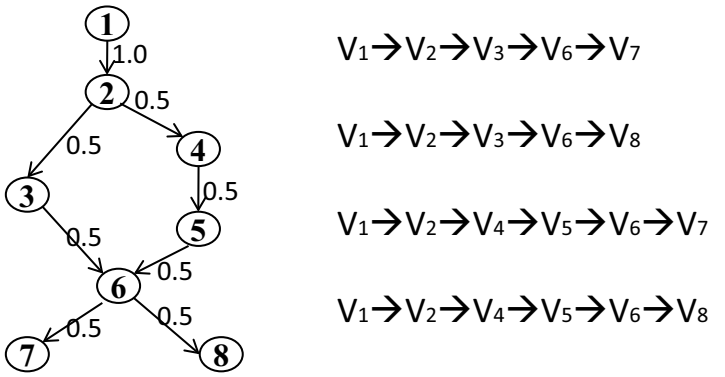


FIGURE 2. A directed weighted software network and its all execute paths

As stated in Section 2, if $ForWere(v_i)$ is larger than *AvgForWere*, then node v_i is deemed as an important fragile node. Algorithm 2 is introduced to identify all important fragile nodes of a software network. *ImpND-DB-fragile* is a data set storing all important fragile nodes. Notation $N_{ForWere=1.0}$ is the number of nodes whose *ForWere* is 1.0. Line 2 to line 8 compute the *AvgForWere*. For each node v_i in software network, if $ForWere(v_i)$ is larger than *AvgForWere*, then node v_i is added into *ImpND-DB-fragile* (line 9 to line 13).

Algorithm 2: Identifying important fragile nodes based on ripple effects

Input: ForWere(v_i) of each node v_i in software network
Output: ImpND-DB-fragile
1: initialize TotalForWere = 0.0, $N_{ForeWere=1.0} = 0$, TempValue = 0.0
2: **for** each node v_i in software network **do**
3: TotalForWere += ForWere(v_i)
4: **if** (ForWere(v_i) == 1.0) **then**
5: $N_{ForWere=1.0}$++
6: **end if**
7: **end for**
8: AvgForWere = (TotalForWere – $N_{ForWere=1.0}$) ÷ (|V| – $N_{ForWere=1.0}$)
9: **for** each node v_i in software network **do**
10: **if** (ForWere(v_i) is larger than AvgForWere) **then**
11: ImpND-DB-fragile.add(v_i)
12: **end if**
13: **end for**

The process of identifying important rigescent nodes is similar to that of fragile nodes. Due to space limitation, we do not give the detailed description here.

4. **Experiments.** Open source softwares *Cflow* and *Tar* are chosen to validate the utility of our approach based on ripple effects (represented as *AppBORE*). The approach based on degree centrality (abbr. as *D-Based*), the approach mentioned in [8] (represented as *AppBase*) and the approach mentioned in [9] (abbr. as *SN-KNN*) are chosen as the comparing approaches.

Table 1, Table 2, Table 3 and Table 4 show the ranking comparisons of fragile and rigescent nodes among different approaches respectively. Notations $R_{AppBORE}$, $R_{AppBase}$ and $R_{SN\text{-}KNN}$ represent the ranking result obtained by approach *AppBORE*, *AppBase* and *SN-KNN* respectively. Notations K_o and K_i correspond to the value of out-degree and in-degree of a node. As it is shown, the ranking has no correlation with the value of K_o and K_i. Therefore, it is not appropriate to identify important nodes by considering the nodes' degree.

Approach *SN-KNN* is only taking part of the structure information into consideration. What is more, the weight of directed edges among a node's neighbors is neglected. Therefore, the approach may lose some important overall structure information inevitably, leading to that the ranking result may not be reasonable enough. As has been stated in Section 1, approach *AppBase* just defines the ripple effect of a node as the size of its reachable nodes set, which does not take reachable probability into account. In other words, it assumed that a bug in one node will definitely propagate to other nodes that point to it directly and indirectly in software networks, while it is contrary to the objective facts. Consequently the ranking result obtained by approach *AppBase* may not be reasonable enough, too. In our approach *AppBORE*, we think that a bug in one node can propagate to other nodes via all possible execute paths, and the weight of each directed edge is

TABLE 1. Ranking comparison of fragile nodes of *Cflow*

function_node	$R_{AppBORE}$	$R_{AppBase}$	$R_{SN\text{-}KNN}$	K_o
yyparse	1	1	1	5
parse_typedef	2	2	10	5
dcl	3	4	6	3
dirdcl	4	5	9	4
parse_declaration	5	3	2	4
maybe_parm_list	6	6	4	3
parse_function_declaration	7	7	3	3
func_body	8	8	5	6
parse_variable_declaration	9	9	8	8
parse_dcl	10	11	12	5

TABLE 2. Ranking comparison of rigescent nodes of *Cflow*

function_node	$R_{AppBORE}$	$R_{AppBase}$	$R_{SN\text{-}KNN}$	K_i
yyrestart	1	20	34	2
yywrap	2	24	7	1
yy_get_next_buffer	3	27	6	1
yylex	4	28	2	1
delete_statics	5	22	43	1
static_free	6	7	9	1
linked_list_destroy	7	10	28	3
yy_load_buffer_state	8	11	21	3
yy_create_buffer	9	17	47	1
ident	10	25	22	1

TABLE 3. Ranking comparison of fragile nodes of *Tar*

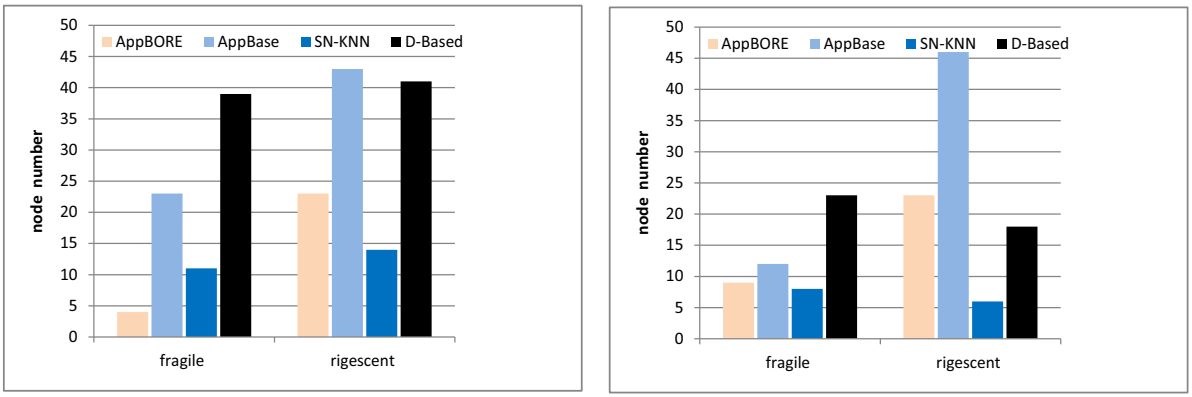
function_node	$R_{AppBORE}$	$R_{AppBase}$	$R_{SN\text{-}KNN}$	K_o
create_archive	1	1	2	8
dump_file	2	2	1	3
dump_file0	3	3	3	17
start_header	4	8	9	9
dump_regular_file	5	6	6	8
dump_dir	6	4	4	3
dump_hard_link	7	7	7	7
dump_dir0	8	5	5	7
to_chars	9	41	19	1
open_archive	10	9	8	2

considered. So the ranking of each function node is different among different approaches, as it is shown in Table 1, Table 2, Table 3 and Table 4.

The number of important fragile and rigescent nodes identified by different approaches are shown in Figure 3(a) and Figure 3(b). For approach *D-Based*, a node with a degree larger than the average degree of the network is deemed as an important node. We have analyzed that the importance of a node has no correlation with its degree. So the number of important nodes identified by *D-Based* is not much meaningful. For approach *SN-KNN*, it does not take the weight of edges among a node's neighbors into consideration, accordingly the result may not be very precise. The definition of ripple effect and the bug propagate process are not reasonable in approach *AppBase*, so the number of important

TABLE 4. Ranking comparison of rigescent nodes of *Tar*

function_node	$R_{AppBORE}$	$R_{AppBase}$	$R_{SN\text{-}KNN}$	K_i
dump_file0	1	85	2	1
dump_file	2	98	3	2
create_archive	3	109	10	1
dump_regular_file	4	61	5	1
dump_dir	5	63	7	1
dump_hard_link	6	62	9	1
start_header	7	31	4	3
dump_dir0	8	50	13	1
open_archive	9	101	39	1
to_octal	10	3	1	1



(a) ImpND number of software *Cflow* (b) ImpND number of software *Tar*

FIGURE 3. Comparisons of ImpND number among different approaches

nodes is not quite reasonable, too. For our approach *AppBORE*, there is no big difference between the number of important nodes for different softwares, as shown in Figure 3(a) and Figure 3(b). For example, Softwares *Cflow* and *Tar* both have 23 important rigescent nodes, and the number of important fragile nodes display little difference. In other words, approach *AppBORE* is widely suitable.

5. **Conclusions and Future Work.** A small portion of important nodes play a critical role in ensuring the stability and quality of the software system. In this paper, we focus on identifying important fragile and rigescent nodes in software network. Firstly, the model of directed weighted software network is constructed. Then, some metrics such as *ForWere* and *ReWere* are adopted to measure the importance of the nodes. If $ForWere(v_i)$ is larger than *AvgForWere* or $ReWere(v_i)$ is larger than *AvgReWere* of the software network, then node v_i is deemed as an important node. Experimental results on real software data show the utility of the approach.

As future work, we are looking into extending the approach to other granularity of software component and evaluating utility of the approach.

Acknowledgment. This work is partially supported by the National Natural Science Foundation of China under Grant No. 61170190, No. 61472341 and the Natural Science Foundation of Hebei Province P. R. China under Grant No. F2013203324, No. F20142031 52, No. F2015203326. The authors also gratefully acknowledge the helpful comments and suggestions of the reviewers, which have improved the presentation.

REFERENCES

[1] J. Ren, H. Wu, T. Yin, Y. Liu and H. He, Mining important nodes in directed-weighted software network based on semi-global structure information, *ICIC Express Letters*, vol.9, no.10, pp.2859-2866, 2015.

[2] A. E. Motter and Y.-C. Lai, Cascade-based attacks on complex networks, *Physical Review E*, vol.66, 2002.

[3] C. R. Myers, Software systems as complex networks: Structure, function, and evolvability of software collaboration graphs, *Physical Review E*, vol.68, 2003.

[4] D. S. Callaw, M. E. J. Newman and S. H. Strogatez, Network robustness and fragility: Percolation on random graphs, *Physical Review Letters*, vol.85, no.25, pp.5468-5471, 2000.

[5] D. Li, B. Li and W. Pan, Ranking the importance of classes via software structural analysis, *Future Communication, Computing, Control and Management*, vol.141, pp.441-449, 2012.

[6] D. Chen, L. Lv, M.-S. Shang, Y.-C. Zhang and T. Zhou, Identifying influential nodes in complex networks, *Physica A*, vol.391, no.4, pp.1777-1787, 2012.

[7] H. He, J. Wang and J. Ren, Measuring the importance of functions in software execution network based on complex network, *International Journal of Innovative Computing, Information and Control*, vol.11, no.2, pp.719-731, 2015.

[8] K. He, Y. Ma, J. Liu, B. Li and R. Peng, *Software Network*, Science Press, Beijing, 2008 (in Chinese).

[9] J. Ren, H. Wu, T. Yin, L. Bai and B. Zhang, A novel approach for mining important nodes in directed-weighted complex software network, *JCIS*, vol.11, no.8, pp.3059-3071, 2015.