Risk Analysis on Multi-Granular Flow Network for Software Integration Testing

Ying Wang, Zhiliang Zhu, Member, IEEE, Hai Yu, and Bo Yang

Abstract—This brief presents a model, a methodology, and an application scheme of risk assessment for information exchange system. The multi-granular flow network (MGFN) model serves as a basis for measuring the vulnerabilities and threats of components, and the failure consequences they bring to the system when a failure occurs. The risk factors of components are then quantified, assisted by a probabilistic risk analysis model. Furthermore, we apply the MGFN model and the risk assessment scheme in ordering class integration testing for object-oriented software system. By comparing our approach with the state-of-the-art integration test order algorithms from the perspectives of detection efficiency of severe faults and stubbing efforts, we show that classes with higher risk indexes can be tested in earlier integration steps, and that the total complexity of the established test rests is minimized.

Index Terms—Risk analysis, information exchange system, multi-granular network, software integration testing.

I. INTRODUCTION

C OMPLEX network is the skeleton of a system, which is used to model the dynamic and static dependencies between components. For an information exchange system (IES), e.g., communication system, software system, electrical system, etc., the nodes carry characters of specific behaviors and information content, and the edges depict the relationships between nodes [1]. The topology of network plays a crucial role in determining the ultimate efficiency, reliability and performance of information flow [2].

Network failure takes place when nodes or edges are attacked deliberately or accidentally. Liu et al. [3] extended the reliability problem to the node failure case. Once information flows are blocked or transmitted erroneously, the failure of information exchange network occurs. After removing the failed nodes and edges, network tolerance indicates the system reliability. Xia and Hill [4] defines the vulnerability as the weaknesses of system. For example, the computer nodes within a communication network are vulnerable to attack by viruses or worms, due to the vulnerability of firewall. From this perspective, risk index of a system can be measured by the expected failure consequence when the attack occurs. The process of risk assessment is useful in identifying complex components that require detailed inspection, estimating potentially troublesome modules, and evaluating maintenance or testing efforts for an information system [5].

To support decision making for critical nodes protection and resilience, risk needs to be described and possibly measured: the consequences, in terms of losses, damages, injuries etc., and the uncertainties therein, in terms, for example, of probabilities (frequencies) [6]. Several researchers have addressed this problem, introducing new perspectives and methods of risk analysis and applying them for identifying the vulnerable parts of system. Moreover, they protect the network and buy out the risk by strengthening the critical nodes and optimizing resource allocation [1], [5], [6].

A limitation exists in the network model that only maps the components and information exchanges into nodes and edges and ignores the hierarchy of the data flow interaction. In this brief, we present a multi-granular flow network (MGFN) model, which depicts the information system at both high and low resolution. Combining with the probabilistic risk analysis, we define risk as a combination of three factors: probability of being attacked, fault-proneness, and the damage of malfunctioning. Furthermore, we consider software as a typical information exchange system and provide an application of the proposed risk assessment model in software integration testing. The experimental results demonstrate that for limited test resources, the risk assessment model can help improve software productivity during the testing process.

II. MULTI-GRANULAR FLOW NETWORK MODEL

The concept of multi-granular flow network model (MGFN) is based on the following assumptions:

- Information exchange network is the abstract and mapping of complex system in information domains, which is comprised of the nodes with information processing capacity and the edges representing information exchanges.
- System is a compound of interactive multiple-components. The interactions between components have characteristics of hierarchy, diversification, and dynamics.

Definition 1 (MGFN): Suppose complex information system $S$ enables a $N$-level disaggregation. At $i$-level granularity, if
Fig. 1. Illustration of components disaggregation levels in MGFN model.

The PRA model provides useful means for identifying potential failures of a component or asset under attack, and the probability of a component or asset being attacked or stressed, \( V(C_i) \), is any functional component, \( i \in \{1, 2, \ldots, N\} \). The structural complexity of a component reflects the probability of potential errors [8]. Zimmermann and Nagappan [9] investigated the correlations between the structure of network motifs and defect data. More specifically, the “ego” network motif can capture the centrality of the critical system component by identifying the size of the star pattern surrounding it, which depicts the failure risk of the main component caused by dependencies. Thus, the more central the component is in the network, the more defect-prone it is. In this brief, we treat the ego network metrics as prominent indicators of component vulnerability.

In economics, probabilistic risk analysis (PRA) defines risk index \( R \) as the product of threat \( T \), vulnerability \( V \), and consequence \( C \), i.e., \( R = T \times V \times C \) [7]. \( T \) is the probability of a component or asset being attacked or stressed, \( V \) is the failure probability of a component or asset under attack, and \( C \) is the financial or fatality consequence if a failure occurs. The PRA model provides useful means for identifying potentially troublesome components that require higher priority and effort during the maintenance and development process. For an information exchange system, we define the heuristic risk indices as follows:

- \( T(C_i) \) - the probability of component \( C_i \) being activated
- \( V(C_i) \) - the fault proneness of component \( C_i \)
- \( C(C_i) \) - the expected damage to the system caused by the failure of component \( C_i \)

Then, \( R(C_i) = T(C_i) \times V(C_i) \times C(C_i) \). More-detailed descriptions of \( T(C_i) \), \( V(C_i) \) and \( C(C_i) \) based on MGFN model are given below.

(1) Threat of components

In flow network \( G_i \), any path \( p_k \) between any pair of nodes in \( V_i \) can be treated as an information flow trajectory. Naturally, we equate the occurrence frequency of each node in all the information flow trajectories of the system with the total number of times that it is activated. Let \( \mathcal{P} = \{p_1, p_2, \ldots, p_{|\mathcal{P}|}\} \) be the path set between all pairs of components in the system. Suppose that all the paths in \( \mathcal{P} \) can be executed \( t_{p_k} \) times with equal probability. Let function \( \tau_k^i(\mathcal{P}) \) represent the total number of trajectories that pass through component \( C_i \), and let \( tc_k^i \) be the total number of times that component \( C_k^i \) is executed; then, we have \( tc_k^i = t_k \times \tau_k^i(\mathcal{P}) \). Consequently, \( T(C_i) \) can be calculated as:

\[
T(C_i) = \frac{tc_k^i}{\sum_{m} tc_m^i} = \frac{tn \cdot \tau_k^i(\mathcal{P})}{\sum_{m} \tau_k^i(\mathcal{P})} = \frac{\tau_k^i(\mathcal{P})}{\sum_{m} \tau_k^i(\mathcal{P})}
\] (3)

(2) Vulnerability of components

The structural complexity of a component reflects the probability of potential errors [8]. Zimmermann and Nagappan [9] investigated the correlations between the structure of network motifs and defect data. More specifically, the “ego” network motif can capture the centrality of the critical system component by identifying the size of the star pattern surrounding it, which depicts the failure risk of the main component caused by dependencies. Thus, the more central the component is in the network, the more defect-prone it is. In this brief, we treat the ego network metrics as prominent indicators of component vulnerability.

Any node \( C_i \) in flow network \( G_i \) at the i-level resolution, has a corresponding ego network that describes how the node is connected to its neighbors. Figure 2 explains the construction of an ego network. The motif contains the component \( C_i \) itself (ego), entities that depend on \( C_i \) (fan-in), and the entities on which \( C_i \) depends (fan-out). We use the size...
of ego network to access the vulnerability of atomic component. The vulnerability of composite component is then the total vulnerability of all the atomic components aggregated in it. Thus, by normalization processing, we have

$$\mathcal{V}(C^N_k) = \frac{\sum_{C_i \in C_k^N} \varphi(C^N_i)}{\sum_{C_m \in \mathcal{V}_N} \sum_{C_i^N \in C_m^N} \varphi(C^N_i)}$$  \hspace{1cm} (4)$$

where $\varphi(C^N_i)$ is the total number of nodes in the ego network corresponding to component $C^N_i$.

(3) Consequence of components

Risk severity considers the worst-case consequence of a failure determined by the degree of system damage and mission loss that can ultimately occur [5]. Suppose that if an atomic component $C^N_k$ contains errors, the faults must be propagated via information transmission relationships to the other components in the most fine-grained network $G_N$. In other words, the failure of component $C^N_k$ leads to the blocking of information flows; therefore, the ratio of flow loss to the maximum information flows transferred in the system can be treated as its failure consequence. According to the definition of MGFN, we introduce the concept of a failed MGFN model to explain and evaluate the consequences to the system caused by the failed modules.

Definition 2 (Failed MGFN): Errors spread to other components via the transmission of information flows if component $C^N_k \in \mathcal{V}_N$ fails, which leads to the failure of overall system. The failed MGFN caused by component $C^N_k$ can be described as the residual network after removing $C^N_k$ and all the failed nodes and edges that are reachable to $C^N_k$ from the original MGFN model.

We formulate the total flow transferred in the information system as:

$$\Sigma \xi = \sum_{C_m^N \in \mathcal{VN}} \sum_{C^N_i \in \mathcal{V}_N} f(C^N_m, C^N_i)$$  \hspace{1cm} (5)$$

Furthermore, let $\mathcal{M}^i_k$ be the failed components that are reachable to $C^N_k$ in network $G_N$, and let $\mathcal{K}^i_k$ be the failed edge set connecting to the nodes in $\mathcal{M}^i_k$. We write the failed network result from component $C^N_k$ as $G^N_k = (\mathcal{VN} \setminus \mathcal{M}^i_k, E_N \setminus \mathcal{K}^i_k, W^N_k)$, where $W^N_k$ denotes the dependency weight matrix corresponding to the residual nodes and edges in $G^N_k$. Then, the residual information flows in network $G^N_k$ satisfy

$$\Sigma \xi_k = \sum_{C^N_m \in \mathcal{V}_N \setminus \mathcal{V}_N^k} \sum_{C^N_i \in \mathcal{V}_N^k} f(C^N_m, C^N_i)$$  \hspace{1cm} (6)$$

where $\mathcal{V}_N^k = \mathcal{VN} \setminus \mathcal{M}^i_k$. Let function $\mathcal{C}(C^N_k)$ be the consequence to the system if component $C^N_k$ fails. Then, we have

$$\mathcal{C}(C^N_k) = 1 - \frac{\Sigma \xi_k}{\Sigma \xi}$$  \hspace{1cm} (7)$$

From the above analysis, we denote the average failure consequence caused by all the atomic components aggregated in composite component $C^N_k$ as $\mathcal{C}(C^N_k)$. Let function $g(C^N_k)$ be the number of atomic components aggregated in $C^N_k$. For $C^N_i \in C^N_k$, $i \in \{1, 2, \ldots, N\}$, $\mathcal{C}(C^N_k)$ then satisfies

$$\mathcal{C}(C^N_k) = \frac{1}{g(C^N_k)} \sum_{C^N_i \in C^N_k} \mathcal{C}(C^N_i)$$  \hspace{1cm} (8)$$

Based on the evaluation scheme for component risk factors, we can identify the critical modules that require more attention and resources therefore optimizing resource allocation to improve the system reliability at minimal cost.

IV. APPLICATION OF RISK ANALYSIS IN SOFTWARE TESTING BASED ON MGFN MODEL

Software is a typical information exchange system. During the runtime, the information flow transmission is equivalent to the process that one method directly or indirectly passes parameters to another by invocations. From coarse to fine granularity, the software system can be mapped into package-level, class-level, and method-level networks. For MGFN model, if we consider methods as the atomic functional components, then function $f(C^N_m, C^N_n)$ defined in Eq. (5) represents the total number of invocations performed by method $C^N_m$ on method $C^N_n$. For the failed MGFN model, nodes that directly or indirectly depend upon the failed methods are affected via parameter passing, which leads to a loss of certain functions [10]. The loss of flows caused by method $C^N_k$ are considered as the failed functions if $C^N_k$ contains errors, i.e., the failure consequence of $C^N_k$. In the other scenarios, the residual network that haven’t been affected by $C^N_k$ can still work well. In this brief, we provide an application of the risk assessment scheme in ordering class integration testing for object-oriented software system.

A. The Risk-Based Class Integration Test Order Strategy

The main concept underlying class order testing is to ensure that non-dependent classes are assigned a higher priority for integration testing, followed by the classes that are dependent on classes that have already been tested [11]. Integration test orders are generated by a reverse sort of the classes based on the directed relationships between them if there are no cyclical dependencies in the system. Moreover, due to the structural complexity of the software, testers must perform the break cycle operations. Once a dependency has been broken, a test stub should be constructed [12].

There are limitations to existing class integration test orders (CITO) algorithms, which only consider stubbing efforts [13]. Special attention should be paid and early testing priority should be assigned to critical classes with high risk indices. To address this problem, the cycle-breaking operations in our strategy have two goals: to provide a higher priority to class with a higher risk index, and to minimize the total complexity of test stubs. For the class-level network $G_C$, we give the following definitions:

Definition 3 (Dependent Path): Suppose that $SV$ is a collection of nodes whose in-degree is zero and out-degree is greater than zero, and that $EV$ is a collection of nodes whose in-degree is greater than zero and out-degree is zero. For any
class $C_i \in SV$, class $C_j \in EV$, $i \in [1,|SV|]$, and $j \in [1,|EV|]$, if there is a path $p_{ij}$ from $C_i$ to $C_j$ in the system, then $p_{ij}$ is defined as a dependent path between $C_i$ and $C_j$.

Definition 4 (Dependent Depth): Suppose that class $C_i \in SV$, class $C_j \in EV$, and that there is a dependent path $p_{ij} = C_i^1 \cdot C_j^1 \cdot \cdot \cdot C_k^1 \cdot C_j$. Then, the dependent depth of $C_j$ satisfies $D_k = |n - k| + 1$.

Definition 5 (Maximum Dependent Depth): Suppose that $q$ dependent paths pass through class $C_i$, and that $D_k$ is the dependent depth of $C_j$ in the $i$-th dependent path, $k \in [1,|V_c|]$, and $i \in [1,q]$. Then, the maximum dependent depth satisfies $D_{k_{max}} = \{D_{k1}, D_{k2}, \ldots, D_{kq}\}$.

For any directed edge $< C_i, C_j >$, let $R_i$ be the normalized risk index of starting node $C_i$, function $\omega(C_i, C_j)$ be the total cycles that pass through $< C_i, C_j >$, and $\omega(C_i, C_j)$ the normalized value of $\omega(C_i, C_j)$. Let $SCplx(C_i, C_j)$ be the stubbing efforts that can be evaluated by the following complexity metric proposed by Briand et al. [11]. We can then assign a weight to each edge in the cycles based on Eq. (9):

$$W_{ij} = \frac{\gamma \times \frac{R_i}{\sqrt{\omega(C_i, C_j)}} + \mu \times \omega(C_i, C_j)}{SCplx(C_i, C_j)}$$  \hspace{1cm} (9)

where $\gamma + \mu = 1$.

The steps for generating class integration test orders are described as follows:

1. Construct the MGFLN model of software system and evaluate the risk factor of each component based on PRA.
2. Find all the strongly connected components (SCCs) in the system by Tarjan’s algorithm [14].
3. For each SCC, we repeat step (i) - (iii).
(i) Find and record all the cycles in SCC, and then assign weight to each edge of each cycle based on Eq. (9).
(ii) Remove edges from the network to break cycles according to the following principles: a) If there is only one edge with the greatest weight in the network, then the edge is deleted. b) If there are more than one edge with the greatest weight, then we delete the edge whose corresponding stubbing effort is lower. c) If there are more than one edge with the greatest weight, and the test stub complexity values corresponding to them are equal, then the edge whose the starting node has the highest risk index is removed.
(iii) Stop the breaking cycle operation if there are no cycles in SCC.

4. The nodes with the same $D_{max}^k$ are sorted by their risk indexes in descending order, while the other nodes are sorted by their $D_{max}^k$ value in ascending order. Finally, we obtain the inner-class integration test order $O_{test}$.

V. CASE STUDY

We adopt three software systems as our experimental data to assess the risk assessment model and its application in integration testing, including Jmeter 1.8 (http://jmeter.apache.org/jmeter), Xml-security 1.0.4 (http://xml.apache.org/security), Joda-time 2.8 (http://github.com/JodaOrg/joda-time). Note that these three systems contain seeded and real faults that have been used widely in research on fault localization, test-case suite selection, etc [15]. Detailed information of the above systems is given in Table I, where $|V_c|$ and $|E_c|$ are the total number of nodes and edges in their corresponding class-level network $G_c$, respectively; and $N_{sys}$ and $N_{cl}$ denote the total number of cycles and faults in the system, respectively.

### A. Evaluation Metric

The average percentage of faults detected per cost (APFDc) has been proposed as a way to measure the effectiveness of the prioritized list of test cases in detecting faults [15]. To assess the fault-detection efficiency of the integration test orders, we redefine APFDc metric based on the risk severity of classes and stubbing efforts:

$$APFD_c = \frac{\sum_{i=1}^{n}(|F_i| \times \left(\frac{\sum_{i \in F_i} s_i \times \left(\frac{\sum_{j=1}^{m} |V_{c} \times c_t|}{c_{t_i}}\right)}{\sum_{m=1}^{t_i} |V_{c} \times c_t|\times |V_{c} \times c_t|}\right)\right) - \sum_{i=1}^{n}(|F_i| \times \left(\frac{\sum_{i \in F_i} s_i \times \left(\frac{\sum_{j=1}^{m} |V_{c} \times c_t|}{c_{t_i}}\right)}{\sum_{m=1}^{t_i} |V_{c} \times c_t|\times |V_{c} \times c_t|}\right)\right)}{\sum_{i=1}^{n}(|F_i| \times \left(\frac{\sum_{i \in F_i} s_i \times \left(\frac{\sum_{j=1}^{m} |V_{c} \times c_t|}{c_{t_i}}\right)}{\sum_{m=1}^{t_i} |V_{c} \times c_t|\times |V_{c} \times c_t|}\right)\right)}$$  \hspace{1cm} (10)

Table II shows the statistics of test orders, where $|V_c|$ and $|E_c|$ are the total number of nodes and edges in their corresponding class-level network $G_c$, respectively; and $N_{sys}$ and $N_{cl}$ denote the total number of cycles and faults in the system, respectively.

### TABLE II

| System          | $|V_c|$ | $|E_c|$ | $N_{sys}$ | $N_{cl}$ | Fault Type |
|-----------------|--------|--------|-----------|----------|------------|
| Jmeter 1.8      | 259    | 762    | 160       | 13       | seeded     |
| Xml-security 1.0.4 | 218  | 803    | 1119      | 10       | seeded     |
| Joda-time 2.8   | 171    | 592    | 526       | 16       | real       |

B. Experimental Results

Figure 3 depicts the metric-value distributions of the three systems, and the statistics of risk factors in software system are described in Table II. Here, $R_{max}$, $R_{min}$, and $R_{med}$ denote the maximum, minimum and median values, respectively, of $R(C_i^k)$. Clearly, only a few classes in the three systems have high risk indexes. Using the Jmeter software as an example, the results of risk analysis for all the classes vary from 1.3e-07 to 0.1843. There are 10 nodes whose risk indexes are greater than 0.03, whereas approximately 80% of nodes have risk indexes that are less than 0.01. The data suggest that only a limited fraction of nodes have relatively high fault rates and error propagation capabilities; therefore, this fraction should be tested first.

Furthermore, we compared the CITO results of these systems obtained by our algorithm with those generated by the approaches of [11] and [12] from multiple perspectives. Note that the aims of [11] and [12] are to minimize the overall test stub complexity, and the number of test stubs, respectively. Moreover, we discuss the results obtained by our strategy both with and without taking the risk factors into consideration. Table III shows the statistics of test orders, where $N_s$ is the
TABLE II

<table>
<thead>
<tr>
<th>System</th>
<th>$R_{\text{max}}$</th>
<th>$R_{\text{min}}$</th>
<th>$R_{\text{med}}$</th>
<th>$T_{\text{max}}$</th>
<th>$T_{\text{min}}$</th>
<th>$T_{\text{med}}$</th>
<th>$V_{\text{max}}$</th>
<th>$V_{\text{min}}$</th>
<th>$V_{\text{med}}$</th>
<th>$V_{\text{top}}$</th>
<th>$V_{\text{bottom}}$</th>
<th>$V_{\text{med}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jmeter</td>
<td>0.1434</td>
<td>1.308E-09</td>
<td>0.0035</td>
<td>0.0001</td>
<td>0.0031</td>
<td>0.00065</td>
<td>1.90E-05</td>
<td>0.0039</td>
<td>0.0277</td>
<td>0.0003</td>
<td>0.0300</td>
<td>0.0003</td>
</tr>
<tr>
<td>Xml-security</td>
<td>0.2095</td>
<td>3.808E-05</td>
<td>0.0045</td>
<td>0.0007</td>
<td>0.0001</td>
<td>0.0051</td>
<td>0.0361</td>
<td>0.0001</td>
<td>0.0241</td>
<td>0.0014</td>
<td>0.0014</td>
<td>0.0047</td>
</tr>
<tr>
<td>Joda-Time</td>
<td>0.2734</td>
<td>1.008E-05</td>
<td>0.0058</td>
<td>0.0001</td>
<td>0.0056</td>
<td>0.0446</td>
<td>0.0012</td>
<td>0.0059</td>
<td>0.0872</td>
<td>0.0014</td>
<td>0.0014</td>
<td>0.0055</td>
</tr>
</tbody>
</table>

Fig. 3. Class risk distribution diagram.

TABLE III

<table>
<thead>
<tr>
<th>Software</th>
<th>Algorithm</th>
<th>$N_F$</th>
<th>$OC_{\text{PK}}$</th>
<th>$N_T$</th>
<th>$N_P$</th>
<th>APFD$_{\text{r}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jmeter</td>
<td>Brand's [11]</td>
<td>10</td>
<td>3.01</td>
<td>6/13</td>
<td>45/90</td>
<td>0.5317</td>
</tr>
<tr>
<td></td>
<td>Ours' (non-risk)</td>
<td>10</td>
<td>2.79</td>
<td>6/13</td>
<td>45/90</td>
<td>0.5109</td>
</tr>
<tr>
<td></td>
<td>Ours'</td>
<td>12</td>
<td>2.83</td>
<td>13/12</td>
<td>79/90</td>
<td>0.8764</td>
</tr>
<tr>
<td>Xml</td>
<td>Brand's [12]</td>
<td>8</td>
<td>4.92</td>
<td>6/10</td>
<td>38/65</td>
<td>0.5716</td>
</tr>
<tr>
<td></td>
<td>Ours' (non-risk)</td>
<td>8</td>
<td>4.96</td>
<td>6/10</td>
<td>35/65</td>
<td>0.5285</td>
</tr>
<tr>
<td></td>
<td>Ours'</td>
<td>9</td>
<td>5.01</td>
<td>10/10</td>
<td>60/65</td>
<td>0.8997</td>
</tr>
<tr>
<td>Joda</td>
<td>Brand's [12]</td>
<td>26</td>
<td>4.95</td>
<td>8/16</td>
<td>26/51</td>
<td>0.5003</td>
</tr>
<tr>
<td></td>
<td>Ours' (non-risk)</td>
<td>23</td>
<td>4.42</td>
<td>9/15</td>
<td>25/51</td>
<td>0.5292</td>
</tr>
<tr>
<td></td>
<td>Ours'</td>
<td>24</td>
<td>4.45</td>
<td>8/16</td>
<td>25/51</td>
<td>0.5357</td>
</tr>
<tr>
<td></td>
<td>Ours</td>
<td>29</td>
<td>4.39</td>
<td>16/16</td>
<td>46/51</td>
<td>0.8982</td>
</tr>
</tbody>
</table>

For IES, special focus should be placed on critical classes with three characteristics: higher probability of being executed, more complexity, and appreciable influence on the system. In this brief, we propose a new strategy for mapping the data flow interactions into multi-granular flow network model. By analyzing the topological structure of IES at both high and low resolution, we evaluate the importance of each component based on a probabilistic risk assessment model. Moreover, we provide an application of the proposed risk assessment scheme in software system to solve the class integration test order problem. The experiment results show that, for limited test resources, the proposed integration strategy can improve software productivity and software quality.

VI. Conclusion

For IES, special focus should be placed on critical classes with three characteristics: higher probability of being executed, more complexity, and appreciable influence on the system. In this brief, we propose a new strategy for mapping the data flow interactions into multi-granular flow network model. By analyzing the topological structure of IES at both high and low resolution, we evaluate the importance of each component based on a probabilistic risk assessment model. Moreover, we provide an application of the proposed risk assessment scheme in software system to solve the class integration test order problem. The experiment results show that, for limited test resources, the proposed integration strategy can improve software productivity and software quality.

REFERENCES