Dominant Alarm Research and Implementation Based on Static Defect Detection

Jun-li JIAO*, Da-hai JIN and Ming-nan ZHOU
Institute of Network Technology, Beijing University of Posts and Telecommunications, Beijing, 100876, China
*Corresponding author

Keywords: Software testing, Static analysis, Defect correlation, Data source.

Abstract. Code defect detection technology plays an increasingly important role in software testing, but the problem of large number of alarms and high false positive rate is common, thus, the efficiency and difficulty of manual confirmation has seriously hindered the development of this technology. This paper proposes a generation method of dominant alarm, which can effectively reduce the number of the human confirmations and improve validation efficiency. Firstly, by analyzing the feature function of the alarms, the alarms with the same data source can be classified as a collection called 'Equivalent class collection'. Then, the dominant alarm in the equivalent class collection is determined by examining the contextual relationship of the codes where the alarm exits in the collection. Finally, the confirmation of the alarms in the collection can be accomplished by confirming the dominant alarm only. The experiment results show that this method can effectively improve the human conformation efficiency of 20%-30%.

Introduction

The static defect detection technology based on defect model is popular to software developers with its technical targeted, highly automated and accurate positioning. However, the experiment suggests that there are lots of false positives found in testing result when it comes to the static defect detection to large scale software, which make it necessary to introduce manual inspection to confirm the testing result and this process requires enormous time and effort[1]. Therefore, it is significant to analyze the relationship between alarms and reduce the number of manual confirmation alarms.

Le W et al.[2] proposed a path-sensitive fault association based on symbol execution and constraint solving techniques to generate fault correlation graphs for fault diagnosis. Zhao YS et al.[3] introduces symbolized three-valued logical abstraction domains on the basis of interval abstract domain to support the logical association between the expression variables. Jung et al.[4] use the Bayesian network to calculate the possibility of an alarm as a real defect. The range of associated alarm which based on symbol execution and constraint solving techniques is limited by the process.

This paper proposes a method to generate the dominant alarm, which can analyzes the feature function of the alarms and puts all the alarms with the same data source to equivalent class collection. Then, the dominant alarm in the equivalent class collection is determined by examining the contextual relationship of the codes where the alarm exits in the collection. Finally, the confirmation of the alarm in the collection can be accomplished by confirming the dominant alarm only.

The main contributions of this paper are as follows:
(1) Propose a method to generate the dominant alarm and give the definition of equivalent class collection and dominant alarm;
(2) Give the method to generate the dominant alarm which based on the feature function;
(3) Implement the method in DTSC, and prove it effectively improves the manual confirmation efficiency of 20%-30% through testing five open source projects.
Terminology

For a better understanding of this paper, in this section, we first introduce the concept of the feature function. Then, we give some definitions, such as the equivalent class collection, the dominant alarm and so on.

Feature Function

1. For alarm IP, the feature function of IP: \( D(IP) = D(v1) \& D(v2) \& \ldots \& D(vn) \);

   where \( D(vx) \) represents the feature information for the alarm of correlation variable \( vx \)

2. For variable \( v \), the feature information of \( v \): \( D(v) = E1(v) | E2(v) | \ldots | En(v) \);

   where \( Ex(v) \) represents the expression information for the \( x \)th setpoint of the variable \( v \)

3. For setpoint \( v \), the expression information of \( v \) is \( E(v) = <S, C, V, P> \):

   - Structure represents the structure of the setpoint, including the operator, library function, function entry parameters and function return value.
   - Constants represents all constants that appear in the setpoint.
   - Variable represents all variables that appear outside the custom function parameters in the setpoint.
   - Position represents the position information of the setpoint, contains \{<filename>, <method>, <beginline>, <endline>\}, and filename is the name of the file where the setpoint is located, method is the name of the function where the setpoint is located, beginline and endline are the starting and ending lines in the file for the setpoint.

Definition

Defect model refers to the features of grammatical or semantic defects that occur frequently in a program. The defect model in DTS is a description of the program, and a defect is caused if the attribute is violated.

Definition 1 (Defect model class). If the two defect model describe the same type of defect, they belong to a defect model class.

Definition 2 (Equivalent class collection). Assuming that \( D(IP) \) represents the feature function of the alarm IP. For the alarm collection \( S = \{IP1, IP2, \ldots, IPn\} \), if the alarm model of \( IP1 \) and \( IP2 \) belongs to a defect model class and \( D(IP1) = D(IP2) \), there is an equivalent relationship between \( IP1 \) and \( IP2 \). Therefore, \( IP1 \) and \( IP2 \) belong to an equivalent class collection. The alarm collection \( S \) may have multiple equivalent class collections, but an alarm \( IPx \) can only belong to an equivalent class collection.

Definition 3 (Dominant alarm). For an equivalent class collection \( S = \{IP1, IP2, \ldots, IPn\} \), the triggering time of the alarm in \( S \) is asynchronous, because there is a sequence of execution logically. Therefore, there must be an alarm IP is triggered firstly, we call \( IPx \) dominant alarm.

Definition 4 (Intra-procedural Dominate). For \( IP1 \) and \( IP2 \) in an equivalent class collection, if \( IP1 \) and \( IP2 \) are within the same process and \( IP1 \) is triggered before \( IP2 \) logically, \( IP1 \) intra-procedural dominate \( IP2 \).

Definition 5 (Inter-procedural Dominate). For \( IP1 \) and \( IP2 \) in an equivalent class collection, if \( IP1 \) is within the process \( fun1 \), \( IP2 \) is within the process \( fun2 \) and \( IP1 \) is triggered before \( IP2 \) logically, \( IP1 \) inter-procedural dominate \( IP2 \).

Definition 6 (The transmission of the dominant alarm). If \( IPa \) dominates \( IPb \) and \( IPb \) dominates \( IPc \), then \( IPa \) dominates \( IPc \).

Generation and Implementation

Based on the definition in the previous section, the method of generating the equivalent class collection and dominant alarm based on the feature function is given below.
Equivalent Class Collection

The equivalent class collection refers to the set of alarm which components has the same feature function. Therefore, the algorithm focuses on the traversal set to divide the alarm with the same feature function into the same equivalent class. The following is the pseudo-code and related description.

**Algorithm.** The Code of generating the equivalent class collection is shown in Figure 1:

![Figure 1. Code of generating the equivalent class collection.](image)

Line 3 to 4 traverse the test result list and combine two alarms in the list; Line 5 determines whether or not they belong to the same defect model class according to the feature information of the two alarms; On line 6 to 7, isSameDSCVP determines whether the feature function of the alarms are same by evaluating the feature function (S, C, V, P) in every setpoint in all layers. Then, addToEquivalentSet adds the alarms to the equivalent class collection. If any one of the alarms has been subordinate to an equivalent class collection A, the other will be added to A, otherwise a new collection which contains the alarms will be created.

Dominant Alarm

The dominant alarm refers to the alarm that is the first to be triggered in an equivalent class collection. According to the upper and lower relations of the alarm in the program code, the dominant relationship is divided into dominant intra-procedural and dominant inter-procedural. The following is the analytical method and algorithm.

**Intra-procedural Dominate.** The nodes in the CFG (control flow graph) contain assignment statements, input statements, output statements, and branch nodes. And the alarm trigger points are on those nodes. For alarm IPa and IPb, suppose that the triggered node of the alarm IPa(IPb) on the CFG is Na(Nb). If there is a path from Na to Nb, Na is executed prior to Nb in the program logic execution order. So IPa is triggered prior to the IPb, that is, IPa is the dominant alarm.

**Inter-procedural Dominate.** Each function in the program corresponds to a node in the CG (call graph). For the alarm IPa and IPb, we assume that function fa(fb) which contains the alarm IPa(IPb) is Na(Nb). If there is a path from Na to Nb, function fa calls fb. We can get the location of the call point by finding the foreground summary fa[5,6]. Finally, we can determine the dominant alarm by analyzing the logical execution order of Na and the node calling fb.

**Algorithm.** The Code of generating the dominant alarm is shown in Figure 2:
The algorithm consists of two methods (getDominantAlarm and dominantAlarmCFG). CFGHasPath (IPa, IPb) determines whether there is a directed path from IPa to IPb on CFG. If the path exists, it means that IPa is the dominant alarm, and IPa is the parent of the dominant alarm tree; otherwise, the function CFGHasPath (IPb, IPa) is called.

Line 13 to 14 calculate whether the two alarm triggers in the same process through withinProcedural. And if they are within the same procedural, dominantAlarmCFG will determine the dominant alarm. Lines 15 to 16 calculate whether the two alarm trigger between the process through outsideProcedural. And if they are in different procedural, CGHasPath(IPa.procedural, IPb.procedural) will be called to determine whether ProceduralIPa calls ProceduralIPb. Then, dominantAlarmCFG will determine the dominant alarm. Line 23 ~ 26 determine the return value of the algorithm.

### Experiment and Analysis

This paper proposes a method to generate dominant alarm based on the static defect detection, and gives the concrete implementation. In order to verify the effectiveness of the above method, it is implemented on DTSC. And five C open source project is selected for the generation experiment. These projects are uucp, aircrack, gsl, redis and nagios, which includes total 342708 lines of code. The test environment is shown in the table below.

<table>
<thead>
<tr>
<th>Hardware environment</th>
<th>Software environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor: Intel(R) Pentium(R) CPU G2030</td>
<td>OS: Microsoft Windows</td>
</tr>
<tr>
<td>Frequency: 3.00GHz.</td>
<td>System: 32-bit</td>
</tr>
<tr>
<td>Memory: 4.00GB.</td>
<td>JDK: JDK 1.7.0_02</td>
</tr>
<tr>
<td></td>
<td>Test tool: DTSC 9.0</td>
</tr>
</tbody>
</table>
Assuming that the number of test alarm is total, the number of alarms that equivalent class collections includes is equalNum alarms, and the number of dominant alarms is dominantNum. We define the Improving efficiency as:

Improving efficiency = (equalNum-dominantNum) / total * 100%

Table 2. Experimental results.

<table>
<thead>
<tr>
<th>project</th>
<th>code line</th>
<th>total num</th>
<th>dominant alarm num</th>
<th>equivalent class collections num</th>
<th>improving efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>uucp</td>
<td>63913</td>
<td>335</td>
<td>39</td>
<td>106</td>
<td>20.00%</td>
</tr>
<tr>
<td>aircrack</td>
<td>43154</td>
<td>350</td>
<td>34</td>
<td>114</td>
<td>22.86%</td>
</tr>
<tr>
<td>gsl</td>
<td>138102</td>
<td>1361</td>
<td>184</td>
<td>558</td>
<td>27.48%</td>
</tr>
<tr>
<td>redis</td>
<td>54284</td>
<td>440</td>
<td>58</td>
<td>161</td>
<td>23.41%</td>
</tr>
<tr>
<td>nagios</td>
<td>43255</td>
<td>277</td>
<td>51</td>
<td>197</td>
<td>52.71%</td>
</tr>
<tr>
<td>total</td>
<td>342708</td>
<td>2713</td>
<td>366</td>
<td>1136</td>
<td>28.28%</td>
</tr>
</tbody>
</table>

By analyzing the experimental results in the above table, we can find that the method of generating the dominant alarm can effectively improve the human conformation efficiency of 20%-30% on average.

Conclusion

In this paper, a generation method of dominant alarm based on static defect detection is proposed. Firstly, we give the definitions of equivalent class and dominant alarm by digging out the characteristic function relationship between alarms. Then we introduce the method of dominant alarm generation, and give the implementation process of the main algorithm. Finally, the feasibility of the proposed method is verified by experiments. The experimental results shed light on the generation of the dominant alarm can effectively reduce the number of manual confirmation of the alarm, and provide a strong support for the static defect detection tool to detect large-scale software.

The limitation of this paper is that we propose dominant alarm based on the equivalent class collection which refers to the set of alarm with the same data sources. But the data sources of quite a few alarms are not exactly same, but similar. How to reduce the number of manual confirmation based on similar data sources to improve the efficiency of confirmation is the direction of the next step.

References