OUTLINE

• Background
  • Automatic static analysis (ASA)
  • Actionable alert (AA) & Unactionable alert (UA)

• Motivation

• Related Work

• Method

• Experiment

• Summary
Automatic Static Analysis

- Static analysis is the process of evaluating a system or component based on its form, structure, content or documentation.

- ASA can identify common coding problems early in the development process via a tool that automates the inspection of source code.

- ASA tools: Findbugs, Lint, Checkstyle.
Alerts: potential source code anomalies reported by ASA.

- Null pointer dereference
- Buffer overflows
- Style inconsistencies
Alert

- Actionable Alerts: if a developer determines the alert is an important, fixable anomaly.

- Unactionable Alerts: When an alert is not an indication of an actual code anomaly or the alert is deemed unimportant to the developer.
STCAL: Sharing a single instance across thread boundaries without proper synchronization will result in erratic behaviour of the application.

7 times in revision 1497967 of Tomcat.
Alert

This method might ignore an exception.

```
1 try { socket.close (); }
2 catch (Exception ignore) {} 
3 try { reader.close (); }
4 catch (Exception ignore) {} 
```
Motivation

Alert density: 40 alerts/KLOC
35 - 91 % of alerts are UA.
Lots of UAs may lead developers and managers to reject ASA due to the overhead of alert inspection.

Suppose,
1000 alerts, 5 min/alert
need 10.4 workdays to inspect all alerts
identify UAs can save 3.6 - 9.5 days
Motivation

Actionable Alert Identification Techniques (AAIT): use the alerts with other information to classify or prioritize alerts.

classification: divide alerts into two groups, UA and AA.
prioritization: order alerts by the likelihood an alert is AA.
OUTLINE

- Background
- Motivation
- Related Work
- Method
- Experiment
- Summary
Related Work

Heckman and Williams use alert characteristics and machine learning to predict actionable FindBugs alerts. This is one of the most comprehensive actionable alert prediction studies today.

Bodden, Lam and Hendren use static analysis to deduce run-time properties of program. They use decision trees with code characteristics to decrease false positives.

Quinn et al. use alert characteristics and machine learning to find code patterns in static analysis alerts.
OUTLINE

- Background
- Motivation
- Related Work
- Method
  - Revision History & Build Project
  - Run ASA Tool
  - Alert Statement Slice
  - Tokenization
  - Extract Features
- Model
Method
Revision History

Generate subject revision history from source code repository, like CVS or SVN
Build Project

Build subject project for each revision. Delete the revisions which can not build successfully.
Run ASA Tool

Run Automatic Static Analysis Tool at each revision to get alert information. In our method, we use FindBug.
Alert Statement Slices

program slicer included in the IBM T.J. Watson Libraries for Analysis (WALA)

- We select the SA alerts as seed statements
- The slicer use the source code and seed statements to build a call graph and pointer analysis
- Construct backwards slices for each alert
Alert Statement Slices

Original:

```
int i;
int sum = 0;
int product = 1;
for(i=1; i<N; ++i){
    sum = sum + i;
    product = product * i;
}
write(sum);
write(product);
```

After slicing:

```
int i;
int sum = 0;
for(i=1; i<N; ++i){
    sum = sum + i;
}
write(sum);
```
Tokenization

Control flow information is important

Granularity: High level. consider method, control flow as tokens

Tool: Eclipse JDT Core
Extract N-Gram Features

Create a dictionary of N-gram from the program slice. Use information gain to reduce data dimensionality. Identify 320 features with most information value.
Other Features

There are three potential features for static analysis actionable alerts:
- Alert Information
- Source code metrics
- Subversion Information

The characteristic for each alert may be different for each alert type, so we need the alert information. Alert Information is retrieved from FindBug for each revision.
## Other Features

<table>
<thead>
<tr>
<th>Group</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alert Information(9)</td>
<td>Alert Category</td>
</tr>
<tr>
<td></td>
<td>Alert Type</td>
</tr>
<tr>
<td></td>
<td>Project Name</td>
</tr>
<tr>
<td></td>
<td>Package Name</td>
</tr>
<tr>
<td></td>
<td>File Name</td>
</tr>
<tr>
<td></td>
<td>Class Name</td>
</tr>
<tr>
<td></td>
<td>Method Signature</td>
</tr>
<tr>
<td></td>
<td>Priority</td>
</tr>
<tr>
<td></td>
<td>Total Alerts for Revision</td>
</tr>
</tbody>
</table>
Other Features

Code complexity metrics correlate with failure-prone modules. Previous work have used code size metrics to predict fault counts.

Use JavaNCSS to generate metrics at the file, package, and project levels.

These tools provide information about the size of source code by lines and the complexity of the programs.

Non commenting source statements (NCSS) counts the number of all statements excluding comments, empty statements, empty blocks, closing brackets or semicolons after closing brackets.
Other Features

<table>
<thead>
<tr>
<th>Group</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Metrics(7)</td>
<td>Classes in Package</td>
</tr>
<tr>
<td></td>
<td>Functions in Package</td>
</tr>
<tr>
<td></td>
<td>Functions in Class</td>
</tr>
<tr>
<td></td>
<td>Cyclomatic complexity in Function</td>
</tr>
<tr>
<td></td>
<td>Class NCSS</td>
</tr>
<tr>
<td></td>
<td>Function NCSS</td>
</tr>
<tr>
<td></td>
<td>Package NCSS</td>
</tr>
</tbody>
</table>
Other Features

Source code repository help determine how the set of alerts generated by static analysis and how the code base has changed over time.

We use the log files of the subversion repositories to analyze the code history.
## Other Features

<table>
<thead>
<tr>
<th>Subversion (15)</th>
<th>Alert Open Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Developers</td>
</tr>
<tr>
<td></td>
<td>File Creation Revision</td>
</tr>
<tr>
<td></td>
<td>File Last Modified Revision</td>
</tr>
<tr>
<td></td>
<td>File Age</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Project Added Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project Delete Lines</td>
</tr>
<tr>
<td>Project Growth</td>
</tr>
<tr>
<td>File Added Lines</td>
</tr>
<tr>
<td>File Deleted Lines</td>
</tr>
<tr>
<td>File Growth</td>
</tr>
<tr>
<td>Package Total Modified Lines</td>
</tr>
<tr>
<td>Package Percent Modified Lines</td>
</tr>
<tr>
<td>File Total Modified Lines</td>
</tr>
<tr>
<td>File Percent Modified Lines</td>
</tr>
</tbody>
</table>
Model

Take input from the 351 features
Add two fully-connected layer
one dropout layer
Final Layer is Sigmoid layer

Grid Search to tune hyperparameter
**OUTLINE**

- Related Work
- Method
- Experiment
  - Dataset
  - Configuration
  - Ground Truth
  - Baseline
  - Evaluation
- Results
- Summary
## Data Set

<table>
<thead>
<tr>
<th></th>
<th>JDOM</th>
<th>Log4j</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Data Format</td>
<td>Logging Library</td>
</tr>
<tr>
<td>Size(KLOC)</td>
<td>9-13</td>
<td>12-19</td>
</tr>
<tr>
<td>Time Frame</td>
<td>05/2000-12/2008</td>
<td>08/2001-06/2007</td>
</tr>
<tr>
<td># Built Revisions</td>
<td>30</td>
<td>11</td>
</tr>
<tr>
<td>Total Alerts</td>
<td>489</td>
<td>237</td>
</tr>
<tr>
<td>Actionable Alerts</td>
<td>200</td>
<td>97</td>
</tr>
<tr>
<td>Unactionable Alerts</td>
<td>254</td>
<td>112</td>
</tr>
<tr>
<td>Deleted Alerts</td>
<td>35</td>
<td>28</td>
</tr>
</tbody>
</table>
Configuration

• Gram Size - The size of an n-gram model. 3-gram model
• Minimum Token Occurrence - The minimum number of times a token must occur in the software to be included in an n-gram model. 3
Ground Truth

By definition, AA is an alert that a developer resolves by modifying the program. It will disappear from static analysis at some point.

If it is UA, the alert will never disappear. If an alert is removed because the file containing the alert is deleted, we consider the alert status as unknown and remove it from the list.
Ground Truth

FaultBench method by Heckman and Williams to accurately classify alerts as AA or UA.

- Generate revision history through the source code repository log.
- Data collection for each project. Download all files associated with a revision.
- Run ASA at each revision and determine which alerts were closed during the alert history.
- Create features for each alert.
Baseline

FindBugs assigns a priority measure to each alert. We assume high priority alerts are more actionable than low priority alerts.

Default FindBugs priority ranking:
Sort alerts according to the priority measure and randomize the order of alerts with the same priority.
Evaluation

Ten-fold cross validation to evaluate models.

randomly separate alerts into ten equal sets.
nine of the sets train the model and test the model use the last set.
repeat the process ten times.
Evaluation Metrics

Precision and Recall to evaluate how well our method performs.

Precision is the percentage of alerts classified as actionable that were actionable.

Recall is the percentage of alerts classified as actionable out of all actual actionable alerts.
## Results

<table>
<thead>
<tr>
<th>Project</th>
<th>Average Precision</th>
<th>Average Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDOM</td>
<td>90.3%</td>
<td>86.2%</td>
</tr>
<tr>
<td>Log4j</td>
<td>91.4%</td>
<td>85.0%</td>
</tr>
</tbody>
</table>
Example
In Log4j, when x=25, it means from the top 25% of the alerts, 74% of actionable alerts are found in our method, while 26% of actionable alerts are found using FindBugs priority ranking.
Results

Our method outperforms FindBugs priority ranking, it can help enhance alert ranking.
OUTLINE

▪ Background
▪ Motivation
▪ Related Work
▪ Method
▪ Experiment
▪ Summary
Summary

After introducing different methods of actionable alert detection, this project presents the following contributions:
1) present a deep learning method to detect actionable alerts.
2) use N-Gram feature combining with other alert features.
3) apply our method to JDOM and Log4j projects.

In our experiment, our method reach precision up to 91.4% and recall up to 86.2%. Our method outperforms FindBugs priority ranking in alert ranking.
THANKS FOR YOUR TIME!