What Causes My Test Alarm?
Automatic Cause Analysis for Test Alarms in System and Integration Testing

Authors: He Jiang¹, Xiaochen Li¹, Zijiang Yang², Jifeng Xuan³
¹Dalian University of Technology, ²Western Michigan University, ³Wuhan University
Background

System and integration testing (SIT)

- Continuous integration increases SIT’s frequency.
  - DevOps: faster time to market
  - Cloud-based system: run 1,000 test scripts in 25 minutes

- Running test scripts in SIT may fail.
  - We find 6000+ failures in a single month in one product

- Testers need to figure out the failure causes
  - Require the stakeholders to fix them
Background

Test software in SIT

- To test software
  - Many artifacts and stakeholders are involved
  - Any artifact may have defects
Test alarms in SIT

- Test scripts may fail for various causes
  - A test alarm is an alarm to warn the test script failure

<table>
<thead>
<tr>
<th>ID</th>
<th>Type of cause</th>
<th>Testers’ solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>Obsolete test</td>
<td>update test scripts</td>
</tr>
<tr>
<td>C2</td>
<td>Product code defect</td>
<td>submit bugs to developers</td>
</tr>
<tr>
<td>C3</td>
<td>Configuration error</td>
<td>correct configuration files</td>
</tr>
<tr>
<td>C4</td>
<td>Test script defect</td>
<td>debug test scripts</td>
</tr>
<tr>
<td>C5</td>
<td>Device anomaly</td>
<td>submit bugs to instrument suppliers</td>
</tr>
<tr>
<td>C6</td>
<td>Environment issue</td>
<td>diagnose the environment</td>
</tr>
<tr>
<td>C7</td>
<td>Software problem</td>
<td>ask site reliability engineers to diagnose</td>
</tr>
</tbody>
</table>
Related Work

Classify test alarms (academic)

- Product code defect or Test script defect [Rogstad et al. 15]
  - For database applications

- Product code defect or Obsolete test [Hao et al. 13]
  - Unit testing
  - First decision tree

- Product code defect or others [Herzig & Nagappan 15]
  - Association rules / Binary Classification

REF:
2. D. Hao, T. Lan, H. Zhang, C. Guo, and L. Zhang. Is this a bug or an obsolete test? In ECOOP
Related Work

Classify test alarms (academic)

- Product code defect or Test script defect [Rogstad et al. 15]
  - For database applications

- Product code defect or Obsolete test [Hao et al. 13]
  - Unit testing
  - First decision tree

- Product code defect or others
  - Association rules / Binary Classification

The causes are more complex than binary classification

REF:
2. D. Hao, T. Lan, H. Zhang, C. Guo, and L. Zhang. Is this a bug or an obsolete test? In ECOOP
Related Work

Classify test alarms (industry)

- A survey to industry testers
  - They collect test logs of failed test scripts
  - They manually build regular expressions for classification
  - Accuracy is 20%-30% over distinct projects

<table>
<thead>
<tr>
<th>#</th>
<th>Regular expression</th>
<th>Cause type</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>topomatch fail</td>
<td>Environment issue</td>
<td>normal</td>
</tr>
<tr>
<td>2</td>
<td>Info: GEN_ERROR_FILE_OPEN</td>
<td>Environment issue</td>
<td>normal</td>
</tr>
<tr>
<td>3</td>
<td>Error: The current mode is unframed mode. Please delete it first</td>
<td>Test script defect</td>
<td>normal</td>
</tr>
<tr>
<td>4</td>
<td>Error: Operation abnormal</td>
<td>Product code defect</td>
<td>severe</td>
</tr>
</tbody>
</table>
The Problem

Test alarm analysis

- Analyze the cause of test alarms
  - Test logs are easy to get
  - Testers also read test logs to analyze the alarms

Classification before bug location, bug fixing etc.

Test script fails

Test alarm

Test log

Algorithm

Failure cause

Find out

collect
The Problem

A test log

- Bilingual documents: English & Chinese
- Long: more than 1000 lines, more than 10GB (14,000 logs)
Cause Analysis Model (CAM)

Framework

• CAM’s Idea

➢ Search the test logs of historical test alarms that may have the same failure cause with the new test log
A test log snippet of function point “AUTO UPDATE SCHEMA” (AUS)

- Each test script is associated with a func. point
- Func. points are functional requirements for the software
- A test script verifying function “configure network proxy” may add "NETCONF_PROXY_FUNC" as the func. point

New test log snippet with function point “AUTO UPDATE SCHEMA (AUS)”
E [exception happens continuously for more than 20 times] [2015-06-28 02:10:52.964] timed out while waiting for more data
Cause Analysis Model (CAM)

Test log preprocess

- Language Detection

New test log snippet with function point “AUTO UPDATE SCHEMA (AUS)"
E [exception happens continuously for more than 20 times]
[2015-06-28 02:10:52.964] timed out while waiting for more data
Cause Analysis Model (CAM)

Test log preprocess

- Language Detection
- English NLP
  - Tokenization,
  - Stop words removal
    (single letters, punctuation marks, and numbers ),
  - Stemming

New test log snippet with function point “AUTO UPDATE SCHEMA (AUS)”
E [exception happens continuously for more than 20 times]
[2015-06-28 02:10:52.964] timed out while waiting for more data
Cause Analysis Model (CAM)

Test log preprocess

- Language Detection
- English NLP
  - Tokenization,
  - Stop words removal
    (single letters, punctuation marks, and numbers ),
  - Stemming
- Chinese NLP
  - Word segmentation

New test log snippet with function point “AUTO UPDATE SCHEMA (AUS)”
E [exception happens continuously for more than 20 times]
[2015-06-28 02:10:52.964] timed out while waiting for more data

E [] [2015-06-28 02:10:52.964] \timed \out \while \waiting \for \more \data

exception \happens \continuously \for more than \20 \times
Test log preprocess

- Language Detection
- English NLP
  - Tokenization,
  - Stop words removal
    (single letters, punctuation marks, and numbers),
  - Stemming
- Chinese NLP
  - Word segmentation
- Term Integration

New test log snippet with function point “AUTO UPDATE SCHEMA (AUS)”
E [exception happens continuously for more than 20 times]
[2015-06-28 02:10:52.964] timed out while waiting for more data

E [ ] [2015-06-28 02:10:52.964] \ timed \ out \ while \ waiting \ for \ more \ data
• Select historical test logs by func. point
  ➢ Select all, if no matched func. point

### Historical test log selection

<table>
<thead>
<tr>
<th>Logs</th>
<th>Func. Point</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>his3</td>
<td>AUS</td>
<td>C2</td>
</tr>
<tr>
<td>his4</td>
<td>AUS</td>
<td>C3</td>
</tr>
<tr>
<td>his1</td>
<td>AUS</td>
<td>C3</td>
</tr>
<tr>
<td>his2</td>
<td>AUS</td>
<td>C3</td>
</tr>
<tr>
<td>his5</td>
<td>AUS</td>
<td>C2</td>
</tr>
<tr>
<td>his6</td>
<td>NPF</td>
<td>C1</td>
</tr>
<tr>
<td>his7</td>
<td>NPF</td>
<td>C3</td>
</tr>
</tbody>
</table>

New test log snippet with function point “AUTO UPDATE SCHEMA” (AUS)
E [exception happens continuously for more than 20 times] [2015-06-28 02:10:52.964] timed out while waiting for more data
**Cause Analysis Model (CAM)**

**Cause prediction**

- Log similarity with selected logs
  - 2-shingling terms (successfully applied in information retrieval)
  - TF-IDF based cosine similarity

<table>
<thead>
<tr>
<th>Logs</th>
<th>Func. Point</th>
<th>$\text{Sim}_{\text{log}}$</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>his3</td>
<td>AUS</td>
<td>0.586</td>
<td>C2</td>
</tr>
<tr>
<td>his4</td>
<td>AUS</td>
<td>0.472</td>
<td>C3</td>
</tr>
<tr>
<td>his1</td>
<td>AUS</td>
<td>0.322</td>
<td>C3</td>
</tr>
<tr>
<td>his2</td>
<td>AUS</td>
<td>0.320</td>
<td>C3</td>
</tr>
<tr>
<td>his5</td>
<td>AUS</td>
<td>0.134</td>
<td>C2</td>
</tr>
</tbody>
</table>
Cause Analysis Model (CAM)

Cause prediction

- Predict by k-Nearest Neighbor
  - Case 1: the similarity of top 1 log (his3) exceeds a threshold
  - Case 2: the similarity of top 1 log (his3) is lower than a threshold
    - \( C_2 = 0.586 + 0.134; C_3 = 0.472 + 0.311 + 0.320 \)

<table>
<thead>
<tr>
<th>Logs</th>
<th>Func. Point</th>
<th>Sim(_{\text{log}})</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>his3</td>
<td>AUS</td>
<td>0.586</td>
<td>C_2</td>
</tr>
<tr>
<td>his4</td>
<td>AUS</td>
<td>0.472</td>
<td>C_3</td>
</tr>
<tr>
<td>his1</td>
<td>AUS</td>
<td>0.322</td>
<td>C_3</td>
</tr>
<tr>
<td>his2</td>
<td>AUS</td>
<td>0.320</td>
<td>C_3</td>
</tr>
<tr>
<td>his5</td>
<td>AUS</td>
<td>0.134</td>
<td>C_2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Logs</th>
<th>Func. Point</th>
<th>Sim(_{\text{log}})</th>
<th>Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>his3</td>
<td>AUS</td>
<td>0.586</td>
<td>C_2</td>
</tr>
<tr>
<td>his4</td>
<td>AUS</td>
<td>0.472</td>
<td>C_3</td>
</tr>
<tr>
<td>his1</td>
<td>AUS</td>
<td>0.322</td>
<td>C_3</td>
</tr>
<tr>
<td>his2</td>
<td>AUS</td>
<td>0.320</td>
<td>C_3</td>
</tr>
<tr>
<td>his5</td>
<td>AUS</td>
<td>0.134</td>
<td>C_2</td>
</tr>
</tbody>
</table>
• Present differences between the new log and the most similar test log of the same cause
  ➢ Testers are familiar with historical test logs
  ➢ Comparison may be more easier

`new test log`

```
  cd/opt/VNFP/0
  -bash: cd
  imageVMNPSO-001
  assertion fails
```

`historical test log`

```
  rm /opt/VNFP/0
  imageVMNPSO-001
  assertion fails
```
Experimental Setup

Dataset

- Two industrial testing projects at Huawei-Tech Inc.
- Logs about one month per project
- More than 14,000 test logs
- Focus on one failure cause per test log

<table>
<thead>
<tr>
<th>#</th>
<th>Info</th>
<th>Dataset 1</th>
<th>Dataset 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td># Test logs</td>
<td>7663</td>
<td>6977</td>
</tr>
<tr>
<td>2</td>
<td>Size</td>
<td>4.72GB</td>
<td>6.06GB</td>
</tr>
<tr>
<td>4</td>
<td># Testing day</td>
<td>40 day</td>
<td>22 day</td>
</tr>
<tr>
<td>5</td>
<td># Test logs per day</td>
<td>192</td>
<td>317</td>
</tr>
<tr>
<td>6</td>
<td># Avg. lines</td>
<td>942 lines</td>
<td>1375 lines</td>
</tr>
<tr>
<td>7</td>
<td># Avg. test steps</td>
<td>247 test steps</td>
<td>344 test steps</td>
</tr>
<tr>
<td>8</td>
<td># Obsolete test (C1)</td>
<td>1185</td>
<td>*</td>
</tr>
<tr>
<td>9</td>
<td># Product code defect (C2)</td>
<td>4459</td>
<td>1963</td>
</tr>
<tr>
<td>10</td>
<td># Configuration error (C3)</td>
<td>761</td>
<td>345</td>
</tr>
<tr>
<td>11</td>
<td># Test script defect (C4)</td>
<td>892</td>
<td>3259</td>
</tr>
<tr>
<td>12</td>
<td># Device anomaly (C5)</td>
<td>335</td>
<td>298</td>
</tr>
<tr>
<td>13</td>
<td># Environment issue (C6)</td>
<td>19</td>
<td>168</td>
</tr>
<tr>
<td>14</td>
<td># Software problem (C7)</td>
<td>12</td>
<td>944</td>
</tr>
<tr>
<td>15</td>
<td># Avg. type of causes per day</td>
<td>3.85 per day</td>
<td>3.86 per day</td>
</tr>
</tbody>
</table>
Experimental Setup

- Evaluation method
  - Accuracy, Area-Under-Curve
  - Running time, memory consumption
  - Incremental framework (simulate testers’ daily work)

- Baseline Algorithms: bag-of-words
  - Lazy Associative Classifier (LAC)
  - Best First Tree (BFT).
  - Topic Model (TM)
Evaluate CAM’s hypothesis

- Are the test logs with the same causes more similar than those with different causes?

- As the similarity grows, more and more test logs are in the same failure cause
- Test logs with the same causes are more similar
Experimental Results

Overall performance

- How does CAM perform against baseline algorithms?

- Outperform the baseline algorithms (p<0.05)

Fig. 1 Accuracy for algorithms on two datasets
**Experimental Results**

### Overall performance

- How does CAM perform against baseline algorithms?

    | Algorithm | Cause | C1   | C2   | C3   | C4   | C5   | C6   | C7   |
    |-----------|-------|------|------|------|------|------|------|------|
    | DS1       | LAC   | 0.61 | 0.57 | 0.48 | 0.52 | 0.50 | 0.33 | 0.51 |
    |           | BFT   | 0.73 | 0.65 | 0.66 | 0.60 | 0.77 | 0.40 | 0.70 |
    |           | TM    | 0.68 | 0.67 | 0.56 | 0.58 | 0.62 | 0.50 | 0.54 |
    |           | CAM   | 0.77 | 0.71 | 0.59 | 0.61 | 0.62 | 0.50 | 0.62 |
    | DS2       | LAC   | -    | 0.60 | 0.53 | 0.64 | 0.63 | 0.83 | 0.73 |
    |           | BFT   | -    | 0.67 | 0.65 | 0.70 | 0.60 | 0.77 | 0.86 |
    |           | TM    | -    | 0.62 | 0.51 | 0.68 | 0.52 | 0.77 | 0.78 |
    |           | CAM   | -    | 0.68 | 0.66 | 0.81 | 0.51 | 0.74 | 0.87 |

- Outperform the baseline algorithms ($p<0.05$)
- Superior over the majority of cause types
Experimental Results

Overall performance

- How does CAM perform against baseline algorithms?

- Outperform the baseline algorithms ($p<0.05$)
- Superior over the majority of cause types
- Resources saving, take about 0.1s and less than 4GB memory to process a test log.

Fig. 3 Comparison on computation resources consumption
Experimental Results

Historical test log selection

- How does historical test log selection work?
  - CAM-FP: CAM without historical test log selection

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>DS1</th>
<th>DS2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Accuracy</td>
<td>Total time</td>
</tr>
<tr>
<td>CAM-FP</td>
<td>0.555</td>
<td>39.2 min</td>
</tr>
<tr>
<td>CAM</td>
<td>0.583</td>
<td>6.9 min</td>
</tr>
</tbody>
</table>

Fig. 4 Accuracy, total time, and memory for CAM and CAM-FP

- Selection reduces noisy and shortens running time
Experimental Results

Historical test log selection

- How does historical test log selection work?
  - CAM-FP: CAM without historical test log selection
  - Selection reduces noisy and shortens running time
  - Without selection, CAM-FP still achieves competitive performance

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>DS1 Accuracy</th>
<th>DS2 Accuracy</th>
<th>DS1</th>
<th>C1</th>
<th>C2</th>
<th>C3</th>
<th>C4</th>
<th>C5</th>
<th>C6</th>
<th>C7</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAM-FP</td>
<td>0.555</td>
<td>0.555</td>
<td>CAM-FP</td>
<td>0.73</td>
<td>0.70</td>
<td>0.59</td>
<td>0.57</td>
<td>0.59</td>
<td>0.50</td>
<td>0.62</td>
</tr>
<tr>
<td>CAM</td>
<td>0.583</td>
<td>0.695</td>
<td>CAM</td>
<td>0.77</td>
<td>0.71</td>
<td>0.59</td>
<td>0.61</td>
<td>0.62</td>
<td>0.50</td>
<td>0.62</td>
</tr>
</tbody>
</table>

Fig. 4 Accuracy, total time, and memory for CAM and CAM-FP

Fig. 5 AUC values for CAM and CAM-FP
Experimental Results

Evaluation in real scenario

● How does CAM perform in a real development scenario?
  ➢ 72% accuracy after running for two months.

● Feedback
  ➢ CAM is better than manually building regular expressions.
  ➢ Actually, I will not believe in an automatic tool. However, after presenting the historical test logs, I can quickly decide whether the prediction is correct. **CAM accelerates my work.**
  ➢ **Suggestions:** labeling the defect-related snippets, provide suggestions on how to fix defects
Conclusion

In this paper, we

- Propose a new approach to address automatically analyzing the test alarm causes in SIT.

- Construct two industrial datasets [http://oscar-lab.org/cam/]. The failure causes are manually labeled and verified by testers.

- Conduct a series of experiments to investigate CAM. CAM is both effective and efficient.

- Deploy and evaluate CAM in a real development scenario.
Thanks

What Causes My Test Alarm?
Automatic Cause Analysis for Test Alarms in System and Integration Testing

Reporter: Xiaochen Li
Dalian University of Technology, China

Authors: He Jiang\textsuperscript{1}, Xiaochen Li\textsuperscript{1}, Zijiang Yang\textsuperscript{2}, Jifeng Xuan\textsuperscript{3}
\textsuperscript{1}Dalian University of Technology, \textsuperscript{2}Western Michigan University, \textsuperscript{3}Wuhan University