CLUSTERING STATIC ANALYSIS DEFECT REPORTS TO REDUCE MAINTENANCE COSTS

Zachary P. Fry and Westley Weimer
University of Virginia
Static Analysis-based Bug Finders

• Use known-faulty semantic patterns to find suspected bugs \textit{statically}
  • Generally with minimal human intervention

• Valgrind, Fortify, SLAM, ConQAT, CodeSonar, PMD, Findbugs, Coverity SAVE, etc.

• Influential in both academia and industry
  • \textit{Many} academic tools spanning various languages
  • Coverity boasts over 300 employees and over 1,100 customers, with extremely high growth
Static Analysis-based Bug Finders

- Produce many defect reports in practice

<table>
<thead>
<tr>
<th>Program</th>
<th>KLOC</th>
<th>Reports</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eclipse</td>
<td>3,618</td>
<td>4,345</td>
</tr>
<tr>
<td>Linux (sound)</td>
<td>420</td>
<td>869</td>
</tr>
<tr>
<td>Blender</td>
<td>996</td>
<td>827</td>
</tr>
<tr>
<td>GDB</td>
<td>1,689</td>
<td>827</td>
</tr>
<tr>
<td>MPlayer</td>
<td>845</td>
<td>500</td>
</tr>
</tbody>
</table>

- Difficult to adapt to particular styles or idioms
- Regardless of true or false positives, groups of defect reports exhibit similarity in practice
Structurally Similar Defects

• Some defect reports are obviously similar or different
• Some are not:

printk(KERN_DEBUG "Receive CCP frame from peer slot(%d)",
lp->ppp_slot);
if (lp->ppp_slot < 0 ||
lp->ppp_slot > ISDN_MAX) {
printk(KERN_ERR "%s:
lp->ppp_slot (%d) out of range", _FUNCTION_,
lp->ppp_slot);
return;
}
is = ippp_table[lp->ppp_slot];
isdn_ppp_frame_log("ccp-rcv",
skb->data, skb->len, 32,

if (!lp->master)
qdisc_reset(lp->netdev->
dev.qdisc);
lp->dialstate = 0;
dev->st_netdev[isdn_dc2minor(
lp->isdn_device
lp->isdn_channel)] = NULL;
isdn_free_channel(
lp->isdn_device,
lp->isdn_channel,
ISDN_USAGE_NET);
lp->flags &=
ISDN_NET_CONNECTED;

sidx = isdn_dc2minor(di, 1);
#ifdef ISDN_DEBUG_NET_ICALL
printk(KERN_DEBUG “n_fi:ch=0
”);
#endif

if (USG_NONE(dev->usage[sidx])){
if (dev->usage[sidx] &
ISDN_USAGE_EXCLUSIVE) {
printk(KERN_DEBUG “n_fi: 2nd channel is down and bound
”);
if ((lp->pre_device == di) &
(lp->pre_channel == 1)) {

}}}
Determining Defect Report Similarity

• Some defect reports are obviously similar or different
• Some are not:

```c
printk(KERN_DEBUG "Receive CCP frame from peer slot(%d)",
lp->ppp_slot);
if (lp->ppp_slot < 0 ||
    lp->ppp_slot > ISDN_MAX) {
    printk(KERN_ERR "%s: lp->ppp_slot (%d) out of range",
        _FUNCTION_,
    lp->ppp_slot);
    return;
}

is = ippp_table[lp->ppp_slot];
isdn_ppp_frame_log("ccp-rcv",
    skb->data, skb->len, 32,
```

```c
if (!lp->master) {
    qdisc_reset(lp->netdev->
        dev.qdisc),
    lp->dialstate = 0;
    dev->st_netdev(isdn_dc2minor(
        lp->isdn_device
    lp->isdn_channel)
        ) = NULL;
    isdn_free_channel(
        lp->isdn_device,
    lp->isdn_channel,
        ISDN_USAGE_NET);
    lp->flags &=
        ISDN_NET_CONNECTED;
```

```c
sidx = isdn_dc2minor(di, 1);
#ifndef ISDN_DEBUG_NET_ICALL
    printk(KERN_DEBUG "n_fi:ch=0\n");
#endif

if (USG_NONE(dev->usage[sidx])){
    if (dev->usage[sidx] &
        ISDN_USAGE_EXCLUSIVE) {
        printk(KERN_DEBUG "n_fi: 2nd channel is down and bound\n");
        if (!lp->pre_device == di) &
            lp->pre_channel == 1)) {
```
Determining Defect Report Similarity

- Some defect reports are obviously similar or different
- Some are not:

```c
printk(KERN_DEBUG "Receive CCP frame from peer slot(%d)",
       lp->ppp_slot);
if (lp->ppp_slot < 0 ||
    lp->ppp_slot > ISDN_MAX) {
    printk(KERN_ERR "%s:
                lp->ppp_slot (%d) out of
              range", _FUNCTION_,
                lp->ppp_slot);
    return;
}

is = ippp_table[lp->ppp_slot];
isdn_ppp_frame_log("ccp-rcv",
                   skb->data, skb->len, 32,

if (!lp->master)
    qdisc_reset(lp->netdev->
               dev.qdisc);
lp->dialstate = 0;

if (USG_NONE(dev->usage[sidx])){
    if (dev->usage[sidx] &
        ISDN_USAGE_EXCLUSIVE) {
        printk(KERN_DEBUG "n_fi: 2nd
                      channel is down and bound\n" );
        if ((lp->pre_device == di) &&
            (lp->pre_channel == 1)) {
        ```
Determining Defect Report Similarity

• Some defect reports are obviously similar or different
• Some are not:

```c
printk(KERN_DEBUG "Receive CCP frame from peer slot(%d)", lp->ppp_slot);
if (lp->ppp_slot < 0 ||
    lp->ppp_slot > ISDN_MAX) {
    printk(KERN_ERR "%s:
          lp->ppp_slot (%d) out of range", _FUNCTION_,
          lp->ppp_slot);
    return;
}

if (!lp->master)
    qdisc_reset(lp->netdev->
          dev.qdisc);
lp->dialstate = 0;

dev->st_netdev[isdn_dc2minor(
    lp->isdn_device,
    lp->isdn_channel)] = NULL;
isdn_free_channel(
    lp->isdn_device,
    lp->isdn_channel,
    ISDN_USAGE_NET);
lp->flags &=
    ISDN_NET_CONNECTED;
```
Goals

• To both aid in triage of real defects and facilitate the elimination of false positives, we desire a technique for clustering automatically-generated, static analysis-based defect reports.

• The technique should be flexible to meet the needs of different systems and development teams.

• The resulting clusters should be more accurate than those produced by existing baselines and also congruent with human notions of related defect reports.
High Level Approach

<table>
<thead>
<tr>
<th></th>
<th>R1 x R2</th>
<th>R1 x R3</th>
<th>R2 x R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td></td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>R2</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3</td>
<td></td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

- R1 x R2: Failed
- R1 x R3: Failed
- R2 x R3: Passed
High Level Approach

<table>
<thead>
<tr>
<th></th>
<th>R1 x R2</th>
<th>R1 x R3</th>
<th>R2 x R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Clustering
High Level Approach

R1  R2  R3

<table>
<thead>
<tr>
<th></th>
<th>R1 x R2</th>
<th>R1 x R3</th>
<th>R2 x R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Clustering

C1: {R1}
C2: {R2, R3}
Approach – Types of Information

• Gathered or synthesized from structured defect reports
  • Type of defect
  • Suspected faulty line
  • Set of lines on static execution path to suspected fault
  • The enclosing function of the suspected fault
  • Three-line window of context around faulty line
  • Macros
  • File system path of suspected faulty file
  • Additional meta-information

• These categories conform to many state-of-the-art static analysis tools’ output format
  • For instance, Coverity’s SAVE tool and Findbugs
Approach – Types of Similarity Metrics

- Structured Similarity Metrics
  - Exact equality

Component $\text{comp} = \text{myGraph.subcomponent(size, false)}$;
Component $\text{comp} = \text{g.subcomponent(getSize(), false)}$;
Approach – Types of Similarity Metrics

• **Structured Similarity Metrics**
  • Exact equality
  • Strict pair-wise comparison

```java
Component comp = myGraph.subComponent(size, false);
Component comp = g.subComponent(getSize(), false);
```
Approach – Types of Similarity Metrics

• Structured Similarity Metrics
  • Exact equality
  • Strict pair-wise comparison
  • Levenshtein edit distance

```
Component comp = myGraph.subcomponent(size, false);
Component comp = g.subcomponent(getSize(), false);
```
Approach – Types of Similarity Metrics

• Structured Similarity Metrics
  • Exact equality
  • Strict pair-wise comparison
  • Levenshtein edit distance
  • TF-IDF

Component comp = myGraph.subcomponent(size, false);
Component comp = g.subcomponent(getSize(), false);
Approach – Types of Similarity Metrics

- Structured Similarity Metrics
  - Exact equality
  - Strict pair-wise comparison
  - Levenshtein edit distance
  - TF-IDF
  - Largest common pair-wise prefix

```java
Component comp = myGraph.subcomponent(size, false);
Component comp = g.subcomponent(getSize(), false);
```
Approach – Types of Similarity Metrics

- Structured Similarity Metrics
  - Exact equality
  - Strict pair-wise comparison
  - Levenshtein edit distance
  - TF-IDF
  - Largest common pair-wise prefix
  - Punctuation edit distance

Component comp = myGraph.subcomponent(size, false);
Component comp = g.subcomponent(getSize(), false);
Approach – Similarity and Clusters

- Learn a linear regression model for all relevant information-metric pairs with similarity cutoff.
- Traditional clustering (e.g. k-medoid) assumes equal feature weights and real-valued properties measured for individual entities.
- Recursively find maximum cliques (clusters) and remove them from similarity graph.
Evaluation

• Research Questions
  1. How effective is our technique at accurately clustering automatically-generated defect reports?
  2. Does our approach outperform existing baseline techniques?
  3. Do humans agree with the clusters produced by our technique?
Evaluation

- **Static analysis defect finding tools**
  - Coverity SAVE (commercial) and Findbugs (open source)

- **Benchmarks**
  - Seven C and four Java open source programs totaling more than 14 million lines of code, yielding 8,948 defect reports

- **Metrics – competing**
  - Cluster accuracy
  - Cluster size

- **Baseline techniques**
  - Code Clone tools – Checkstyle, ConQAT, PMD
  - Well-established tools that solve a similar problem
Results

• Pareto frontier representing parametric choice between accuracy and cluster size
• Split between languages
Results

Larger clusters at most levels of accuracy

Pareto Frontier - All C Benchmark Programs

Larger clusters at all levels of accuracy

Pareto Frontier - All Java Benchmark Programs
Results

Capable of near perfect accuracy

Capable of perfect accuracy
Cluster Quality

• Clusters ultimately should agree with humans’ intuition of defect report similarity
• Given highly accurate (>90%) and highly inaccurate (<10%) clusters of actual defect reports, we asked humans if they thought the defect reports described the same or highly related bugs

• Results
  • “Accurate” clusters: 99% of humans think reports are related
  • “Inaccurate” clusters: 44% of humans think reports are related

• Humans do not overwhelmingly agree on inaccurate clusters
  • Motivates a parametric approach
Conclusion

- Defect reports from static analyses are prevalent and can be readily clustered.
- Our technique is effective at clustering such reports – it is capable of nearly perfect accuracy.
- Our technique outperforms the nearest baselines – with almost unanimously bigger clusters at all accuracy levels.
- Our technique produces accurate clusters – and humans agree with those clusters.