Context-Aware Statistical Debugging
From Bug Predictors to Faulty Control Flow Paths

Lingxiao Jiang and Zhendong Su
University of California at Davis
Outline

- Introduction
- Context-Aware Statistical Debugging
- Empirical Evaluation
- Related Work and Conclusion
Why Need Debugging Aids?
Why Need Debugging Aids?

Debugging is time-consuming:
- Hard to reconstruct failure scenarios
- Hard to unveil root causes
Failure Analysis based on Feedback

Program

Instrumentation

if ( p==NULL )
call_p_null();

call_stmt_counter();

if ( i>array_bound )
call_access_vio();

Data Collection

Profiles

if ( value_is_invalid )
call_wrong_result();

Overhead on users
Statistical Debugging

Program

if ( p==NULL )
call_p_null();
call_stmt_counter();
if ( i>array_bound )
call_access_vio();
if ( value_is_invalid )
call_wrong_result();

Profiles

Instrumentation

call_stmt_counter();
if ( i>array_bound )
call_access_vio();
if ( value_is_invalid )
call_wrong_result();

Data Collection
Statistical Debugging

- Low-overhead instrumentation
  - Simple program predicates
  - Sampled instrumentation

- Statistics-based analysis
  - Which instrumented program predicates are more likely related with failures
Do these program predicates (i.e. **bug predictors**) directly indicate the locations of failure causes?
Do these program predicates (i.e. **bug predictors**) directly indicate the locations of failure causes?
Outline

- Introduction
- Context-Aware Statistical Debugging
- Empirical Evaluation
- Related Work and Conclusion
Context-Aware Statistical Debugging

Program

if (p1)
  inc_counter(p1);
if (p2)
  inc_counter(p2);
while (p3)
  inc_counter(p3);
if (p4)
  inc_counter(p4);

Profiles

if (predicate 1)
  while (predicate 3)
  if (predicate 4)
  END

Statistical Analysis

if (predicate 1)
while (predicate 3)

Predicate Clustering

if-predicate 1
if-predicate 2

.............

while-predicate 3
if-predicate 4

Predicate Correlation Clusters

Instrumentation

Data Collection

Control Flow Analysis

Function start
if (predicate 1)
  if (predicate 2)
while (predicate 3)
  if (predicate 4)
  END

BUG
Approach Overview

Machine Learning Modules

Approach Overview

Data pre-processing

Faulty Control Flow Paths with Bug Predictors

CFG Analysis (6)

Branch Prediction (5)

Instrumented Programs

Execution Profiles

(1)

Machine Learning Modules

Feature Selection (2/3)

Classification

Bug Predictors

Predicate Correlations

Clustering (4)
Approach Details (1/6)

- Profiling mechanism
  - Based on the *Cooperative Bug Isolation* project
  - Pre-selected instrumentation predicates
    - Branch conditions
    - Return values
    - Pair-wise comparisons among scalar-values

- Profile for each execution
  - A label: "success" or "failure"
  - A numerical vector: the number of times each predicate is observed to be true
Look for bug predictors (predicates)

- Establish relationship models between the predicates and the labels
  - \( \text{Label} = M(p_1, \ldots, p_n) \)
  - Based on machine learning algorithms. E.g. Support Vector Machines, Random Forests.

- Select the most failure-relevant predicates
  - Compute predicate impact scores based on the models

\((p_{11}, \ldots, p_{n1}, \text{label}_1), (p_{1i}, \ldots, p_{ni}, \text{label}_i), (p_{1m}, \ldots, p_{nm}, \text{label}_m)\)
Approach Details (3/6)

- A black box view of machine learning

\[ M(p_1, \ldots, p_n) \]

The bug predictors: the most failure-related predicates!
Look for relationships among the predicates

**Purpose**: Provide additional hints about the cause-failure transition chains
Approach Details (4/6)

- Look for relationships among the predicates
  - **Purpose**: Provide additional hints about the cause-failure transition chains
  - **Method**: Search for predicates of similar execution histories in *failed* runs using clustering

\[
(p_{11}, \ldots, p_{n1}, \text{label}_1) \quad (p_{11}, \ldots, p_{1}, \ldots, p_{1m}) \\
(p_{1i}, \ldots, p_{ni}, \text{label}_i) \quad \text{transposition} \quad (p_{j1}, \ldots, p_{ji}, \ldots, p_{jm}) \\
(p_{n1}, \ldots, p_{ni}, \ldots, p_{nm})
\]

"vertical" view of profiles

The predicate correlation clusters

"horizontal" view of profiles
Branch prediction

**Purpose:** Decide branch directions that may lead to failures

**Method:** Rely on feature selection and clustering. For each branch predicate,

- Traverse its corresponding branch, if the predicate is
  - Selected,
  - Or clustered with a selected predicate
  - Or, true in most failed runs, i.e., most $p_{ji} > 0$ in $(p_{j1}, \ldots, p_{ji}, \ldots, p_{jm})$

- "horizontal" profiles

- Otherwise, traverse both branches.
Control flow path generation

- Depth-first, inter-procedural graph traversal, until all selected predicates are met
- Directed by the branch predictions

- Return a set of faulty paths that may contain bug locations
- Linear time w.r.t. program sizes
- The more predicted branches, the faster
Outline

- Introduction
- Context-Aware Statistical Debugging
- Empirical Evaluation
- Related Work and Conclusion
Subject Programs

- Siemens test suite (STS)
  - 132 faulty versions for seven C programs
  - Hundreds of LoC in each version
  - \( \sim 43,400 \) LoC

- Rhythmbox 0.6.4
  - A multi-threaded music management program for GNOME
  - \( \sim 56,500 \) LoC
Result Summary for STS

- 79 bugs were localized in the faulty control flow paths that cover 9.2% of the code
- 38 bugs were localized by inspecting no more than 1% of the code
Result Summary for Rhythmbox (1/2)

- 5 bugs (2 unreported) localized
- A simple example:

```c
static gboolean alive = FALSE;

......

i < impl_array->len
monkey_media_is_alive() == FALSE
......
```

A sample predicate cluster

```c
void monkey_media_shutdown (void) {
  if (monkey_media_is_alive () == FALSE)
    return;
  alive = FALSE;
  ......
  for (...; i<impl_array->len; ...)  
    ...free memory...
}

......

gboolean monkey_media_is_alive (void) {
  return alive;
}
```

Failure due to race condition!
Result Summary for Rhythmbox (2/2)

Time Cost

<table>
<thead>
<tr>
<th>Instrumentation Predicates</th>
<th># of Predicates</th>
<th>Time of Predicate Selection (min)</th>
<th>Time of Predicate Clustering (min)</th>
<th>Time of Path Generation (min)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Branch conditions</td>
<td>6,863</td>
<td>41</td>
<td>30</td>
<td>&lt;1</td>
</tr>
<tr>
<td>Return values</td>
<td>25,287</td>
<td>45</td>
<td>770</td>
<td>&lt;1</td>
</tr>
</tbody>
</table>

Worthwhile w.r.t. many hours spent on traditional testing and debugging, especially for a real world application that we were not familiar with and that contains unknown bugs.
Outline

- Introduction
- Context-Aware Statistical Debugging
- Empirical Evaluation
- Related Work and Conclusion
Related Work (1/2)

- [Lal ESOP 2006]: Look for shortest paths that connect together as many bug predictors as possible, based on weighted pushdown systems
  - Exponential w.r.t. the number of given predicates
  - No consideration on predicate correlations

- [Brun ICSE 2004]: fault-localization with Daikon (a dynamic invariant detection tool) and machine learning.


- [Liu ESEC/FSE 2005]: statistical debugging with a different predicate selection strategy.

- Look for bug predictors only.
Related Work (2/2)

- [Jones ICSE 2002, ASE 2005]: fault-localization with a statement-level instrumentation and visualization
  - Rank and visualize statements in the order of their bug-relevancy
  - Use statements as bug predictors

- [Zeller FSE 2002, Cleve ICSE 2005]: delta debugging with known failure-inducing inputs
  - Focus in space on failure-related variables and in time on cause transitions
  - Know all program states within failed and succeeded runs
Conclusion

- A context-aware approach for statistical debugging based on the Cooperative Bug Isolation project
  - Identify bug predictors accurately
  - Cluster correlated program predicates
  - Construct faulty control flow paths

- As a result, more contextual information (predicate correlations, control flow paths) for debugging and less code inspection burden!
Thank you!

Questions?
jiangl@cs.ucdavis.edu