Today

- Test prioritization (Scout, formerly Echelon, from MSR)
- Delta debugging (Zeller)
  - [www.askigor.org](http://www.askigor.org) (for Linux binaries)

Regression testing

- Rerunning test cases which a program has previously executed correctly in order to detect errors spawned by changes or corrections made during software development and maintenance (FDA part 11 guide: glossary of terms, draft)
- “First, do no harm.”
- Needed in part due to “imperfect debugging” – Ohba and Chou, ICSE11

Definition

- Program $P'$ modified version of Program $P$
- $T$ is test suite for $P$
- How to validate $P'$ – specifically, those features of $P'$ that are also in $P$?
- What tests in $T$ should be run on $P'$?

Classic approach: retest-all

- Run all non-obsolete test cases in $T$
- Problem: expensive

Other approaches

- Analyze $P$, $P'$, and $T$ to select a subset of $T$
  - Focus on modified elements of $P$ in $P'$
- Define heuristics to remove redundant test cases from $T$ – with respect to a given coverage criterion
- Prioritize test cases
  - High rate of detecting faults
  - Cases that exercise most frequently used features
  - Round-robin (or such…)
Using program changes

- Source code differencing
  - F. Vokolos & P. Frankl, “Pythia: a regression test selection tool based on text differencing”, May 1997

- Data and control flow analysis

- Code entities

Analysis of various techniques

- Source code differencing
  - Simple and fast
  - Can be built using commonly available tools like “diff”
  - Will fail when macro definition changes
  - To avoid these pitfalls, static analysis is needed

- Data and control flow analysis
  - Flow analysis is difficult in languages like C/C++ with pointers, casts and aliasing
  - Interprocedural data flow techniques are extremely expensive and difficult to implement in complex environment

Our Solution

- Focus on change from previous version
  - Determine change at very fine granularity – basic block/instruction

- Operates on binary code
  - Easier to integrate in production environment
  - Scales well to compute results in minutes

- Simple heuristic algorithm to predict which part of code is impacted by the change

Test Effectiveness Infrastructure

Coverage Impact Analysis → Test Prioritization

Coverage Tools

- Old Build
- New Build
- Binary Off
- SMART/VULCAN

Magellan
Echelon: Test Prioritization

Leverage what has already been tested

Prioritized list of test cases

Block Change Analysis: Binary Matching

BMAT – Binary Matching [Wang, Pierce and McFarling JILP 2000]

Coverage Impact Analysis

- Terminology
  - Trace: collection of one or more test cases
  - Impacted Blocks: old modified and new blocks
- Compute the coverage of traces for the new build
  - Coverage for old (unchanged and modified) blocks are same as the coverage for the old build
  - Coverage for new nodes requires more analysis

Coverage Impact Analysis

- Limitations - New node may not be executed
  - If there is a path from successor to predecessor
  - If there are changes in control path due to data changes

Echelon: Test Case Prioritization

- Detects minimal sets of test cases that are likely to cover the impacted blocks (old changed and new blocks)
  - Input is traces (test cases) and a set of impacted blocks
  - Uses a greedy iterative algorithm for test selection
**Echelon: Test Selection**

- **Ordered List of Traces**
  - Trace T1
  - Trace T2
  - Trace T3
  - Trace T4
  - Trace T5
  - Trace T7
  - Trace T8
  - Trace Tm

- **Set 1**
  - T1, T2

- **Set 2**
  - T3, T5

- **Set 3**
  - T4

- Denotes that a trace T covers the impacted block

---

**Analysis of results**

**Three measurements of interest**

- How many sequences of tests were formed?
- How effective is the algorithm in practice?
- How accurate is the algorithm in practice?

---

**Details of BinaryE**

<table>
<thead>
<tr>
<th>Version 1</th>
<th>Version 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>12/11/2000</td>
</tr>
<tr>
<td>Functions</td>
<td>31,020</td>
</tr>
<tr>
<td>Blocks</td>
<td>668,068</td>
</tr>
<tr>
<td>Arcs</td>
<td>1,097,294</td>
</tr>
<tr>
<td>File size</td>
<td>8,880,128</td>
</tr>
<tr>
<td>PDB size</td>
<td>22,602,752</td>
</tr>
<tr>
<td>Impacted Blocks</td>
<td>0</td>
</tr>
<tr>
<td>Number of Traces</td>
<td>3128</td>
</tr>
<tr>
<td># Source Lines</td>
<td>~1.8 Million</td>
</tr>
</tbody>
</table>

**Echelon takes ~210 seconds for this 8MB binary**

---

**Number of Impacted Blocks in each set**

---

**Echelon: Test Selection Output**

- Ordered List of Traces
  - Trace T1
  - Trace T2
  - Trace T3
  - Trace T4
  - Trace T5
  - Trace T7
  - Trace T8
  - Trace Tm

- SET1

- SET2

- SET3

- SETn

- Each set contains test cases that will give maximum coverage of impacted nodes.

- **Gracefully handles the "main" modification case**

- If all the test can be run, tests should be run in this order to maximize the chances of detecting failures early.
Effectiveness of Echelon

- Important Measure of effectiveness is early defect detection
- Measured % of defects vs. % of unique defects in each sequence
- Unique defects are defects not detected by the previous sequence

Effectiveness of Echelon

<table>
<thead>
<tr>
<th>Sequence</th>
<th>% Defects detected</th>
<th>% Unique Defects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>80</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td>20</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

Blocks predicted hit that were not hit

Blocks predicted not hit that were actually hit

(Blocks were target of indirect calls are being predicted as not hit)
Summary

- Binary based test prioritization approach can effectively prioritize tests in large scale development environment
- Simple heuristic with program change in fine granularity works well in practice
- Currently integrated into Microsoft Development process

Coverage Impact Analysis

- Echelon provides a number of options
  - Control branch prediction
  - Indirect calls: if N is target of an indirect call a trace needs to cover at least one of its successor block
- Future improvements include heuristic branch prediction
  - Branch Prediction for Free [Ball, Larus]

Echelon: Test Selection

- Options
  - Calculations of weights can be extended, e.g. traces with great historical fault detection can be given additional weights
  - Include time each test takes into calculation
  - Print changed (modified or new) source code that may not be covered by any trace
  - Print all source code lines that may not be covered by any trace

Delta Debugging

Andreas Zeller
Shameless borrowing of material!
Mozilla Crash

What is relevant here?

Simplified Input

- Required 12 tests only
- 896 lines → 1 line

Simplify vs. Isolate

Simplification: 26 steps
Isolation: 7 steps

Error: <SELECT>