# CSE503: Software Engineering

David Notkin
University of Washington
Department of Computer Science & Engineering
Spring 2006

## Today

- Test prioritization (Scout, formerly Echelon, from MSR)
- Delta debugging (Zeller)
  - 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…)

"Borrowed" with only minor reduction – thank you, Amitabh and Jay!

## Effectively Prioritizing Tests in Development Environment

ISSTA 2002

Amitabh Srivastava
Jay Thiagarajan

PPRC, Microsoft Research

## Using program changes

- Source code differencing
  - S. Elbaum, A. Malishevsky & G. Rothermel "Test case prioritization: A family of empirical studies", Feb. 2002
  - S. Elbaum, A. Malishevsky & G. Rothermel "Prioritizing test cases for regression testing" Aug. 2000
  - F. Vokolos & P. Frankl, "Pythia: a regression test selection tool based on text differencing", May 1997

## Using program changes

- Data and control flow analysis
  - T. Ball, "On the limit of control flow analysis for regression test selection" Mar. 1998
  - G. Rothermel and M.J. Harrold, "A Safe, Efficient Regression Test Selection Technique" Apr. 1997
- Code entities
  - Y. F. Chen, D.S. Rosenblum and K.P. Vo "TestTube: A System for Selective Regression Testing" May 1994

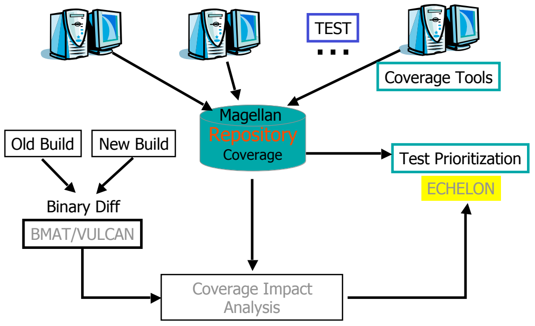
## Analysis of various techniques

- Source code differencing
  - Simple and fast
  - Can be built using commonly available tools like "diff"
  - Simple renaming of variable will trip off
  - 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

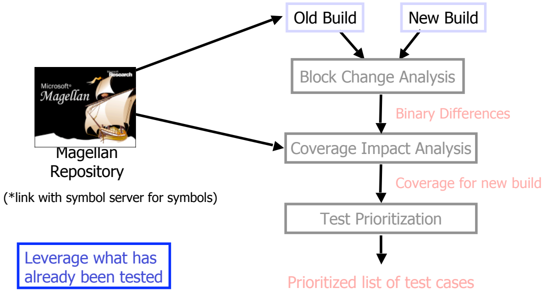
## Our Solution

- 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

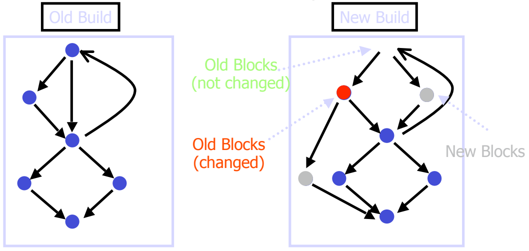
## Test Effectiveness Infrastructure



## Echelon : Test Prioritization

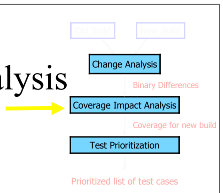


## Block Change Analysis: Binary Matching



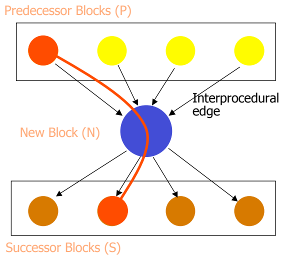
BMAT – Binary Matching [Wang, Pierce and McFarling JILP 2000]

## 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



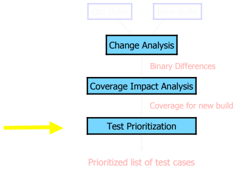
- A Trace may cover a new block N if it covers at least one Predecessor block and at least one Successor Block
- If P or S is a new block, then its Predecessors or successors are used (iterative process)

## Coverage Impact Analysis

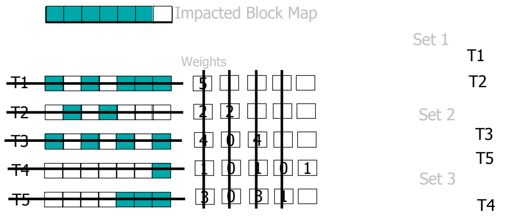
- Limitations - New node may not be executed
  - If there is a path from successor to predecessor
  - If there are changes in control path due to data changes

## 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



Set 1
T1
T2

Set 2
T3
T5

Set 3
T4

🟩 Denotes that a trace T covers the impacted block

## Echelon: Test Selection Output

Ordered List of Traces

Trace T1
Trace T2
Trace T3
SET1

Trace T4
Trace T5
SET2

Trace T7
Trace T8
SET3

Trace Tm
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

## Analysis of results

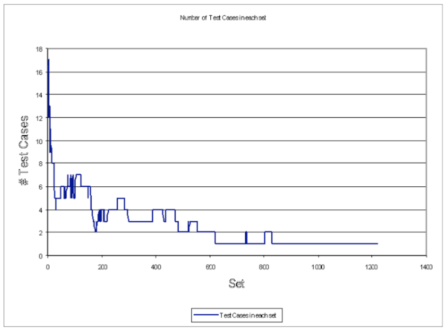
Three measurements of interest

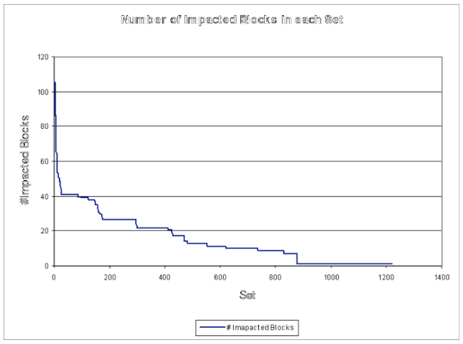
- How many sequences of tests were formed ?
- How effective is the algorithm in practice ?
- How accurate is the algorithm in practice ?

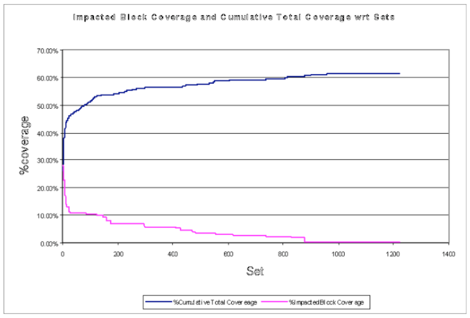
## Details of BinaryE

| | Version 1 | Version 2 |
|---|---|---|
| Date | 12/11/2000 | 01/29/2001 |
| Functions | 31,020 | 31,026 |
| Blocks | 668,068 | 668,274 |
| Arcs | 1,097,294 | 1,097,650 |
| File size | 8,880,128 | 8,880,128 |
| PDB size | 22,602,752 | 22,651,904 |
| Impacted Blocks | 0 | 378 (220 N, 158 OC) |
| Number of Traces | 3128 | 3128 |
| # Source Lines | ~1.8 Million | ~1.8 Million |

Echelon takes ~210 seconds for this 8MB binary







25

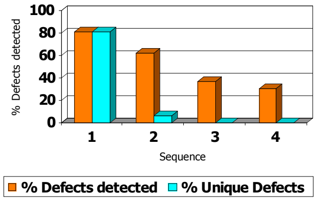
## Effectiveness of Echelon

- 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

26

## Effectiveness of Echelon

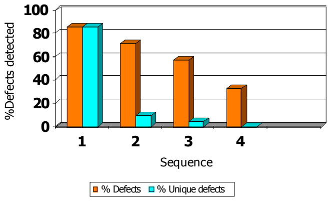
**Defects detected in each sequence**



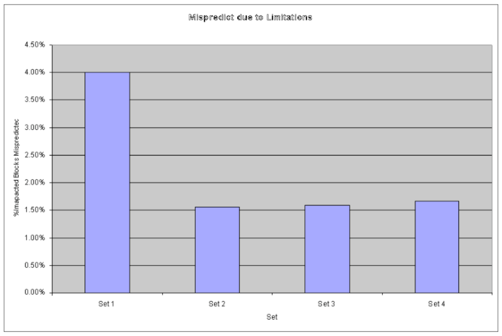
27

## Effectiveness of Echelon

**Defects detected in each sequence**

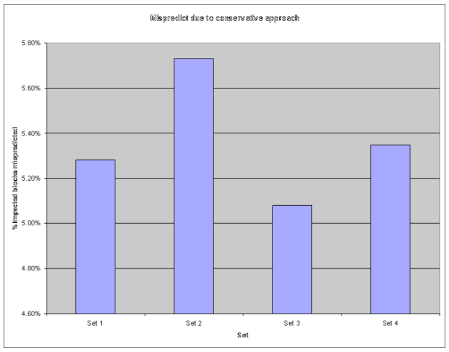


28



Blocks predicted hit that were not hit

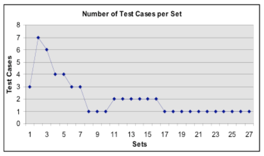
29

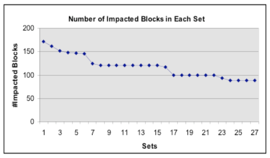


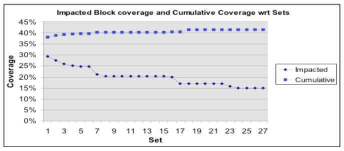
Blocks predicted not hit that were actually hit
(Blocks were target of indirect calls are being predicted as not hit)

30

# Echelon Results: BinaryK

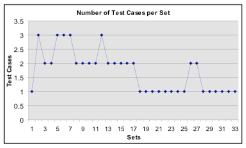


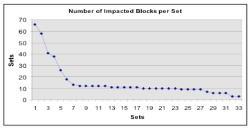


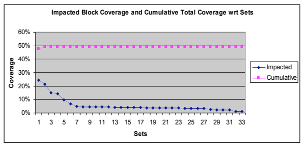


| | Build 2470 | Build 2480 |
|---|---|---|
| Date | 05/01/2001 | 05/23/2001 |
| Functions | 1,761 | 1,774 |
| Blocks | 32,012 | 32,135 |
| Arcs | 47,131 | 47,323 |
| File size | 882,688 | 894,464 |
| Impacted Blocks | 0 | 589 (350 N, 239 OC) |
| Traces | 56 | 56 |

# Echelon Results: BinaryU







| | Build 2470 | Build 2480 |
|---|---|---|
| Date | 05/01/2001 | 05/23/2001 |
| Functions | 1,967 | 1,970 |
| Blocks | 30,916 | 31,003 |
| Arcs | 46,638 | 46,775 |
| File size | 528,384 | 528,896 |
| Impacted Blocks | 0 | 270 (190 N, 80 OC) |
| Traces | 56 | 56 |

# 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!
See
http://www.st.cs.uni-sb.de/papers/fse2002/ms2003.pdf

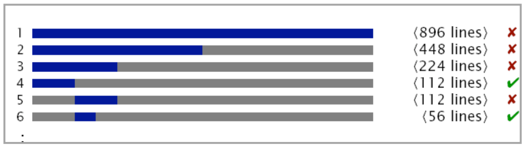
# Mozilla Crash

```
<td align=left valign=top>
<SELECT NAME="op_sys" MULTIPLE SIZE=7>
<OPTION VALUE="All">All<OPTION VALUE="Windows 3.1">Windows 3.1<OPTION VALUE="Windows 95">Windows 95<OPTION VALUE="Windows
98">Windows 98<OPTION VALUE="Windows ME">Windows ME<OPTION VALUE="Windows 2000">Windows 2000<OPTION VALUE="Windows
NT">Windows NT<OPTION VALUE="Mac System 7">Mac System 7<OPTION VALUE="Mac System 7.5">Mac System 7.5<OPTION VALUE="Mac
System 7.6.1">Mac System 7.6.1<OPTION VALUE="Mac System 8.0">Mac System 8.0<OPTION VALUE="Mac System 8.5">Mac System
8.5<OPTION VALUE="Mac System 8.6">Mac System 8.6<OPTION VALUE="Mac System 9.x">Mac System 9.x<OPTION VALUE="MacOS X">MacOS
X<OPTION VALUE="Linux">Linux<OPTION VALUE="BSDI">BSDI<OPTION VALUE="FreeBSD">FreeBSD<OPTION VALUE="NetBSD">NetBSD<OPTION
VALUE="OpenBSD">OpenBSD<OPTION VALUE="AIX">AIX<OPTION VALUE="BeOS">BeOS<OPTION VALUE="HP-UX">HP-UX<OPTION
VALUE="IRIX">IRIX<OPTION VALUE="Neutrino">Neutrino<OPTION VALUE="OpenVMS">OpenVMS<OPTION VALUE="OS/2">OS/2<OPTION
VALUE="OSF/1">OSF/1<OPTION VALUE="Solaris">Solaris<OPTION VALUE="SunOS">SunOS<OPTION VALUE="other">other</SELECT>
</td>
<td align=left valign=top>
<SELECT NAME="priority" MULTIPLE SIZE=7>
<OPTION VALUE="--">--<OPTION VALUE="P1">P1<OPTION VALUE="P2">P2<OPTION VALUE="P3">P3<OPTION VALUE="P4">P4<OPTION
VALUE="P5">P5</SELECT>
</td>
<td align=left valign=top>
<SELECT NAME="bug_severity" MULTIPLE SIZE=7>
<OPTION VALUE="blocker">blocker<OPTION VALUE="critical">critical<OPTION VALUE="major">major<OPTION
VALUE="normal">normal<OPTION VALUE="minor">minor<OPTION VALUE="trivial">trivial<OPTION VALUE="enhancement">enhancement</SELECT>
</tr>
</table>
```

What is relevant here?

# Simplified Input

- Required 12 tests only
- 896 lines → 1line



<SELECT NAME="priority" MULTIPLE SIZE=7>

# Simplify vs. Isolate

```
 1 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✘
 2 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
 3 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
 4 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
 5 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✘
 6 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✘
 7 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
 8 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
 9 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
10 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✘
11 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
12 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
13 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
14 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
15 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
16 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✘
17 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✘
18 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✘
19 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
20 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
21 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
22 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
23 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
24 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
25 <SELECT NAME="priority"␣MULTIPLE␣SIZE=7> ✔
26 <SELECT NAME="priority"␣MULTIPLE␣SIZE=7> ✘
```

Simplification: 26 steps

```
2 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✘
4 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✘

7 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
6 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
5 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
3 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
1 <SELECT␣NAME="priority"␣MULTIPLE␣SIZE=7> ✔
```

Isolation: 7 steps

Error: <SELECT>