

Test Prioritization

17-654/17-765
Analysis of Software Artifacts
Jonathan Aldrich

Test Prioritization: Motivation

- Goal: find and eliminate newly introduced defects
- Regression Testing for Windows
 - Many tests
 - Many platform configurations to run them on
 - Full tests take weeks to run
- Test Prioritization
 - Want to run tests likely to fail first
 - Day 1 after internal release, not day 21!
- Test Selection
 - What tests should I run before checking in code?
 - What tests should be run before releasing a critical fix?
 - Special case of prioritization

Observation: New defects are introduced from changed code

3/29/2005

Slide adapted from ICFEM talk by Amitabh Srivastava

2

Challenges in Test Prioritization

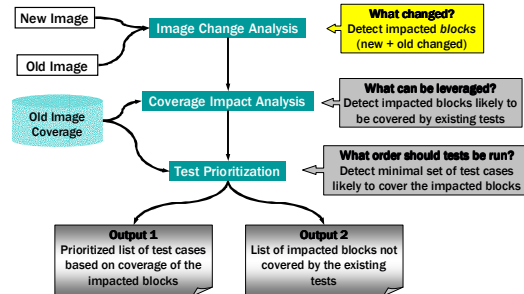
- Detecting change and affected parts of the program
- Scalability to handle complex systems
 - Tens of millions of tests
 - Thousands of developers and testers
 - Tens of millions lines of source code
 - Acceptable response times
- Integrating seamlessly into development process

3/29/2005

Slide adapted from ICFEM talk by Amitabh Srivastava

3

Scout: Test Prioritization System



3/29/2005

Slide adapted from ICFEM talk by Amitabh Srivastava

4

BMAT – Binary Matching

- Goal: detect corresponding blocks in old and new versions of a program
 - [Wang, Pierce, and McFarling JILP 2000]
- Matches basic blocks in binary code
 - + don't need source code
 - must ignore changes in address space
- Algorithm considers similarities in code and in its uses

3/29/2005

Slide adapted from ICFEM talk by Amitabh Srivastava

5

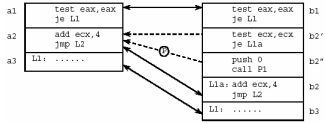
BMAT – Matching Procedures

- Match procedures if names match
 - Qualified by package, scope, etc.
 - If ambiguous, extend to include argument types
- Check for similar names
 - Verify match if blocks are similar (see below)
- Look for function bodies hashing the same
- Pairwise compare blocks otherwise
- If no match, conclude function is new

3/29/2005

6

BMAT – Matching Blocks



- Match blocks based on hash of contents
 - Look for exact match first, then apply fuzzy hashing algorithm
 - Fuzzy algorithms ignore information that is likely to change due to innocuous changes: offsets, registers, block addr, opcodes
- Control-flow match
 - Build CFG, look for node pairs with the same connectivity
 - May match many new blocks to one old block
 - Partial match: new block not always executed (e.g. b2*)

3/29/2005

7

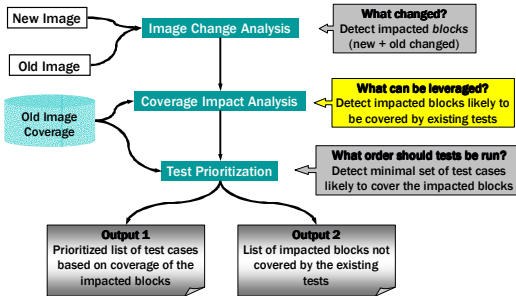
Detecting Impacted Blocks

- Old blocks
 - Identical (modulo address changes)
- Impacted blocks
 - Old modified blocks
 - New blocks

3/29/2005

8

Scout: Test Prioritization System



3/29/2005

Slide adapted from ICFEM talk by Amitabh Srivastava

9

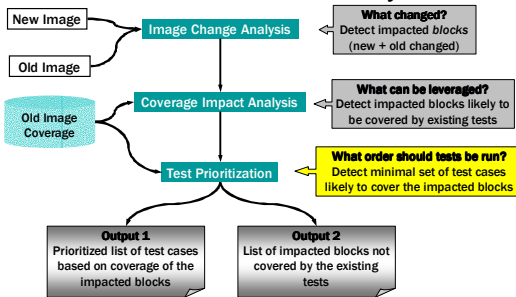
Computing Coverage

- Computed for each test T
- Old block b
 - Covered if T covered b in old binary
- New block
 - Covered if at least one predecessor and successor were covered in old binary
 - Heuristic: predict edges taken
 - Heuristic: don't check predecessors for indirect call targets

3/29/2005

10

Scout: Test Prioritization System



3/29/2005

Slide adapted from ICFEM talk by Amitabh Srivastava

11

Prioritization Algorithm

Input:
 TestList: set of tests
 Coverage(t): set of blocks covered by test t
 ImpactedBlockSet: set of new and old modified blocks

Output: a set of sequences Seq

Algorithm:
 while (any t in TestList covers any block in ImpactedBlockSet)
 {
 CurrBlockSet = ImpactedBlockSet
 Start a new sequence Seq
 while (any t in TestList covers any block in CurrBlockSet)
 {
 for each t in TestList compute
 {
 Weight(t) = count[CurrBlockSet ∩ Coverage(t)]
 }
 Select test t in TestList with maximum weight
 Add t to current sequence Seq
 Remove t from TestList
 CurrBlockSet = CurrBlockSet - Coverage(t)
 }
 }
 Put all remaining tests in TestList in a new sequence Seq

| Test | Blocks |
|------|-------------|
| t1 | b2,b7 |
| t2 | b1,b2,b3,b8 |
| t3 | b7 |
| t4 | b6 |
| t5 | b1,b2,b5 |
| t6 | b4,b5 |

Impacted: b1,b2,b4,b7,b8

Seq1:
 Seq2:
 Seq3:

3/29/2005

12

Echelon Performance: ProductX.EXE

| Image Info | | | Results | |
|------------------|--------------|--------------|---------------------------------------|--------------------------|
| | Build 2411.1 | Build 2529.0 | Impacted Blocks | 378 (220 New, 158 OC) |
| Date | 12/11/2000 | 01/29/2001 | Likely Covered by existing tests (LC) | 176 Blocks |
| Functions | 31,020 | 31,026 | Traces needed to cover LC (Set 1) | 16 Traces |
| Blocks | 668,068 | 668,274 | Number of sets in prioritized list | 1,225 |
| File size | 8,880,128 | 8,880,128 | | |
| PDB size | 22,602,752 | 22,651,904 | | |
| Number of Traces | 3,128 | 3,128 | | |

1.8 million lines of source code

Scout took about 210 seconds

3/29/2005

Slide adapted from ICFEM talk by Amitabh Srivastava

13

Test Sequence Characteristics

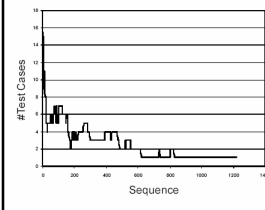


Figure 3. Number of tests in each sequence

3/29/2005

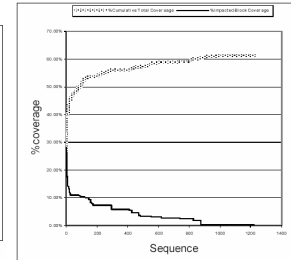


Figure 5. Cumulative coverage and impacted coverage

14

Prediction Errors

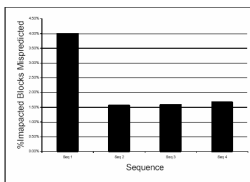


Figure 6. Predicted blocks not covered

1-4% False Positives

3/29/2005

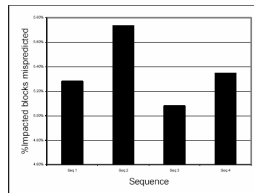


Figure 7. Blocks covered but not predicted

4-5% False Negatives

15

Defect Detection

Program A

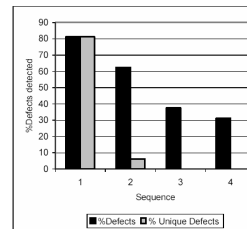


Figure 12. Defects detected in each sequence

3/29/2005

Program B

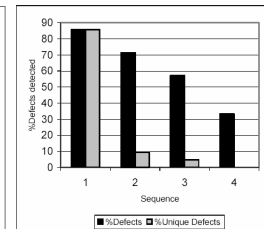


Figure 13. Defects detected in each sequence

16

Summary: Test Prioritization

- Effectively being used in MS Windows, SQL, and Exchange development process
 - Quickly identifies tests most likely to detect errors
- Scales to production environments - millions of tests and thousands of binaries
- Combination of approximations and static analysis to eliminate manual methods
- Collect information about development process

3/29/2005

Slide adapted from ICFEM talk by Amitabh Srivastava

17