Test Prioritization

Related reading: *Effectively Prioritizing Tests in Development Environment*

15-413 Introduction to Software Engineering
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
**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

---

**Scout: Test Prioritization System**

```
New Image

Old Image

Old Image Coverage

Image Change Analysis

Coverage Impact Analysis

Test Prioritization

Output 1
Prioritized list of test cases based on coverage of the impacted blocks

Output 2
List of impacted blocks not covered by the existing tests

What changed?
Detect impacted blocks (new + old changed)

What can be leveraged?
Detect impacted blocks likely to be covered by existing tests

What order should tests be run?
Detect minimal set of test cases likely to cover the impacted blocks
```
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

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
Detecting Impacted Blocks

- **Old blocks**
  - Identical (modulo address changes)
- **Impacted blocks**
  - Old modified blocks
  - New blocks
Scout: Test Prioritization System

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 branches taken
  - Heuristic: don’t check predecessors for indirect call targets
Scout: Test Prioritization System

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
Echelon Performance: ProductX.EXE

<table>
<thead>
<tr>
<th>Image Info</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Build 2411.1</td>
</tr>
<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>File size</td>
<td>8,880,128</td>
</tr>
<tr>
<td>PDB size</td>
<td>22,602,752</td>
</tr>
<tr>
<td>Number of Traces</td>
<td>3,128</td>
</tr>
</tbody>
</table>

1.8 million lines of source code
Scout took about 210 seconds

Test Sequence Characteristics

![Figure 3. Number of tests in each sequence](image1.png)

![Figure 5. Cumulative coverage and impacted coverage](image2.png)
Prediction Errors

1-4% False Positives

4-5% False Negatives

Defect Detection

Program A

Program B

16 November 2005
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