On Test Suite Composition and Cost-Effective Regression Testing

Ping Chen
Nov 23, 2004

Outline
- Introduction
- Background and Related Work
- Experiments
- Discussion
- Conclusion
Introduction

- Why regression testing?
  - Software evolves
  - Validate new features
  - Detect whether new faults have been introduced
- Why study regression testing?
  - Important, but expensive
  - Improve cost-effectiveness

Known Methodologies

- Regression Test Methodologies
  - Retest-all
  - Regression test selection
  - Regression test reduction
  - Regression test prioritization
- Cost-effectiveness of specific technique varies with characteristics of test suites
Test Suite Composition

- How to compose test inputs into test cases within a test suite
  - How many test cases within a test suite?
  - Large or small test cases?
- Focus in this paper
  - Test suite granularity
  - Test input grouping

Some Definitions

- Test case
  - Consists of a pre-test state of the system under test, a sequence of test inputs, a statement of expected test results
- Test Suite
  - A set of test cases
Practical Questions when design test cases

- Practical questions test engineers face
  - Word processor – how many and which editing commands to include per sequence
  - Compiler – how many and which constructs to include in each target-language program
  - Test a class library – how many and which methods to invoke per driver

Practical Choices

- Answers involve many factors
  - Setup cost?
  - Large or small test cases?
  - Interaction of inputs?
- Test case size
- Number of test cases
Practical Definitions

- Test suite granularity
  - Describes a partition on a set of test inputs into a test suite containing test cases of a given size

- Test input grouping
  - Involves the relationship between the test inputs that are assembled into individual test case

Methodologies Revisited

- Program P, P’ is the modified version of P, T is a test suite set for P
- Regression testing is concerned with validating P’
- Reuse T or create new test cases
- Here, we focus on reuse of T
Retest-all

- Simply reuse all non-obsolete test cases in T to test P'
- Obsolete test cases in T
  - no longer apply to P'
  - must be reformulated or discarded
- Expensive

Regression Test Selection

- Use information about P, P' and T to select a subset of T
- Some techniques
  - Modified entity technique
  - Modified non-core entity technique
  - Minimization technique
Regression Test Reduction

- T may contain redundant test cases
- GHS reduction technique used
  - A heuristic proposed by Gupta, Harrold and Soffa
  - Attempts to produce suites that are minimal for a given coverage criterion
  - In this paper, function coverage criterion is used

Test Case Prioritization

- Schedule test cases according to some criterion
- Test cases with higher priority will be executed first
- Some commonly used criteria
  - High rate of detecting faults
  - Exercise features with most frequent use first
Related Work

- Cost and benefits of all mentioned methodologies have been well examined in many articles.
- Test suite granularity has been treated as a factor in some studies of regression test
  - only consider safe RTS techniques
  - did not consider test input grouping

Experiments

- How do test suite granularity and test input grouping affect the costs and benefits of regression testing methodologies?
- 4 null hypotheses
  - H1 (test suite granularity): no significant impact
  - H2 (test input grouping): no significant impact
  - H3 (technique): not perform significantly differently
  - H4 (interaction of two factors): not significantly differ
Objects of Analysis

- **Emp-server**
  - Server component of client-server interaction game *Empire*
  - Main routine
    - Initialization
    - An event loop
      - Waits for a command
      - Invokes routines to process received command

- **Bash**
  - Short for “Bourne Again SHell”
  - Popular open-source application
  - Provides a command line interface to multiple Unix services
Variables and Measurements

- Independent Variables
  1. Regression Testing Techniques
     - Retest-all
     - Selection: modified, modified non-core, minimization
     - Reduction: GHS
     - Prioritization
       - Additional function coverage
       - Additional modified-function coverage
       - Optimal

<table>
<thead>
<tr>
<th>Program</th>
<th>Version</th>
<th>Functions</th>
<th>Changed Functions</th>
<th>Lines of Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>emp-server</td>
<td>4.2.0</td>
<td>1,188</td>
<td>—</td>
<td>63,014</td>
</tr>
<tr>
<td>emp-server</td>
<td>4.2.1</td>
<td>1,188</td>
<td>51</td>
<td>63,014</td>
</tr>
<tr>
<td>emp-server</td>
<td>4.2.2</td>
<td>1,197</td>
<td>245</td>
<td>63,658</td>
</tr>
<tr>
<td>emp-server</td>
<td>4.2.3</td>
<td>1,196</td>
<td>157</td>
<td>63,937</td>
</tr>
<tr>
<td>emp-server</td>
<td>4.2.4</td>
<td>1,197</td>
<td>9</td>
<td>63,988</td>
</tr>
<tr>
<td>emp-server</td>
<td>4.2.5</td>
<td>1,197</td>
<td>101</td>
<td>64,063</td>
</tr>
<tr>
<td>emp-server</td>
<td>4.2.6</td>
<td>1,197</td>
<td>32</td>
<td>64,108</td>
</tr>
<tr>
<td>emp-server</td>
<td>4.2.7</td>
<td>1,197</td>
<td>156</td>
<td>64,439</td>
</tr>
<tr>
<td>emp-server</td>
<td>4.2.8</td>
<td>1,189</td>
<td>52</td>
<td>64,381</td>
</tr>
<tr>
<td>emp-server</td>
<td>4.2.9</td>
<td>1,189</td>
<td>12</td>
<td>64,396</td>
</tr>
<tr>
<td>bash</td>
<td>2.0</td>
<td>1,494</td>
<td>—</td>
<td>48,292</td>
</tr>
<tr>
<td>bash</td>
<td>2.01</td>
<td>1,537</td>
<td>238</td>
<td>49,555</td>
</tr>
<tr>
<td>bash</td>
<td>2.01.1</td>
<td>1,538</td>
<td>40</td>
<td>49,666</td>
</tr>
<tr>
<td>bash</td>
<td>2.02</td>
<td>1,678</td>
<td>197</td>
<td>58,090</td>
</tr>
<tr>
<td>bash</td>
<td>2.02.1</td>
<td>1,678</td>
<td>12</td>
<td>58,103</td>
</tr>
<tr>
<td>bash</td>
<td>2.03</td>
<td>1,703</td>
<td>152</td>
<td>59,010</td>
</tr>
<tr>
<td>bash</td>
<td>2.04</td>
<td>1,890</td>
<td>267</td>
<td>63,648</td>
</tr>
<tr>
<td>bash</td>
<td>2.05a</td>
<td>1,942</td>
<td>411</td>
<td>65,319</td>
</tr>
<tr>
<td>bash</td>
<td>2.05b</td>
<td>1,949</td>
<td>34</td>
<td>65,433</td>
</tr>
<tr>
<td>bash</td>
<td>2.05</td>
<td>1,950</td>
<td>20</td>
<td>65,474</td>
</tr>
</tbody>
</table>
Variables and Measurements

- Independent Variables

  2. Test Suite Granularity
  - Construct test suites of varying granularities by sampling a single "pool" or "universe" of test grains
  - Test grain – smallest input that could be used as a test case
  - A test case of size $s$ consists of $s$ test grains
  - A universe of $n$ test grains produces $\lceil n/s \rceil$ test cases of size $s$
  - In this way, we have G1, G2, G4, G8, G16, G32, G64 – 7 levels of granularities

- Independent Variables

  3. Test input grouping
  - Random grouping
    - Sampling test grains randomly
    - “left-over”s form a group with size may smaller than $s$
  - Functional grouping
    - Separate test grains into “buckets” according to functionalities
    - Repeatedly taking $s$ test grains from each bucket and forming a test case
    - “left-over”s from each “bucket” form another “pool”
    - Randomly sample test cases in this small “pool”
    - May create a certain number of functionally nonhomogeneous test cases
Variables and Measurements

- Dependent Variables and Measurements
  - Savings in Test Execution Time
  - Costs in Fault Detection Effectiveness
    - Which test cases reveal which faults
    - In this study, program has multiple faults
    - Activate each fault in P’ individually
    - May overestimate faults due to faults masking
      - However, it’s a nuisance variable: 0.48% for empserver; 0.012% for bash

Variables and Measurements

- Dependent Variables and Measurements
  - Savings in Rate of Fault Detection
    - APFD – weighted average of the percentage of faults detected

\[
APFD = 1 - \frac{\sum_{i=1}^{m} T_{fi}}{mn} + \frac{1}{2n}
\]
Experiment Setup

- Test Cases and Test Automation
  - Emp-server
    - Use the *Empire* information files (informal specification-based)
    - Category partition method
    - Create smallest test cases possible
      - Sequence of 1-6 lines of characters (average 1.2 lines per test case)
      - Each command of 1-38 test cases
      - 1985 test grains
  - Functional Grouping
    - 196 buckets (196 commands)
    - Average size of 12 test case per bucket
    - No bucket larger than 64, few greater than 16

- Test Cases and Test Automation
  - Emp-server
    - Test script: execute and validate automatically
  - Bash
    - Partition each large test case that came with release 2.0 into the smallest possible test grains
      - They all function across all the 10 releases
      - Each exercise whole functional components
    - Create new test cases by using reference documentation to cover those (67%) uncovered functions
    - 1168 test cases
      - 18 buckets
      - Average bucket size 64
Information of Generated Test Cases

Table II. Test Cases per Granularity Level

<table>
<thead>
<tr>
<th>Granularity Level</th>
<th>Emp-Server</th>
<th>Bash</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1</td>
<td>1985</td>
<td>1168</td>
</tr>
<tr>
<td>G2</td>
<td>903</td>
<td>584</td>
</tr>
<tr>
<td>G4</td>
<td>497</td>
<td>2992</td>
</tr>
<tr>
<td>G8</td>
<td>249</td>
<td>146</td>
</tr>
<tr>
<td>G16</td>
<td>125</td>
<td>73</td>
</tr>
<tr>
<td>G32</td>
<td>63</td>
<td>37</td>
</tr>
<tr>
<td>G64</td>
<td>32</td>
<td>19</td>
</tr>
</tbody>
</table>

Table III. Percentages of Purely Homogeneous Test Cases Present in Functional Groupings

<table>
<thead>
<tr>
<th>Program</th>
<th>G2</th>
<th>G4</th>
<th>G8</th>
<th>G16</th>
<th>G32</th>
<th>G64</th>
</tr>
</thead>
<tbody>
<tr>
<td>emp-server</td>
<td>95.0</td>
<td>89.0</td>
<td>72.0</td>
<td>35.0</td>
<td>12.0</td>
<td>0.0</td>
</tr>
<tr>
<td>bash</td>
<td>95.0</td>
<td>93.0</td>
<td>95.5</td>
<td>90.4</td>
<td>78.4</td>
<td>63.2</td>
</tr>
</tbody>
</table>

Experiment Setup

- **Faults**
  - Seeded: associated with variables, control flow and memory allocation
  - Reject
    - Those could not be detected by any test case
    - Those were detected by more than 80% of the test case at every granularity level
Experiment Design an Analysis Strategy

- Design four sets of experiments for each program, each with same format
- Randomized Factorial (RF) design
  - 2 levels of grouping
  - 7 levels of granularities
  - Varying number of techniques
- Why RF?
  - Main factors
  - Interaction between them
- Analyze emp-server and bash separately

Threats to Validity

- Internal Validity
  - Human factor, e.g fault seeding
  - Only 1 test suite, however, multiple test suite is costly
  - Masking effect: infrequently
  - Magnitude and distribution of changes between versions
- External Validity
  - Quantity and quality of programs studied
  - Fault representativeness
Threats to Validity

- Construct
  - Three measurements are not only possible ones
  - Ignore some other costs
    - Human costs of executing, auditing, managing
    - Debugging costs
    - Analysis time to select, prioritize and reduce

- Conclusion
  - Number of programs and version

Data and Analysis

- Measurements
  - Savings in execution time
  - Faults detection effectiveness
  - Savings in rate of detecting faults (used in prioritization)

- Techniques
  - Retest-all
  - Selection
  - Reduction
  - Prioritization
Data and Analysis – Retest-all

- Fault detect effectiveness remain constant, independent with both factors
  - At higher granularity lever, some test cases do fail to detect faults detectable at lower level
  - Too few to be visible in the graph
- Test suites are powerful enough
  - Detect all of the faults in the programs
- Execution time decreases when granularity level increase, independent with grouping
Analysis of Variance (ANOVA)

Table IV. Retest-All ANOVA

Technique: Retest-All

<table>
<thead>
<tr>
<th>Source</th>
<th>Opport.</th>
<th>MS</th>
<th>F</th>
<th>p</th>
<th>Bash</th>
<th>DF</th>
<th>MS</th>
<th>F</th>
<th>p</th>
</tr>
</thead>
<tbody>
<tr>
<td>Granularity</td>
<td>3288098</td>
<td>6</td>
<td>544683</td>
<td>5693.41</td>
<td>0.00</td>
<td>4635964</td>
<td>6</td>
<td>4772661</td>
<td>19.13</td>
</tr>
<tr>
<td>Grouping</td>
<td>199</td>
<td>1</td>
<td>199</td>
<td>2.08</td>
<td>0.15</td>
<td>131338</td>
<td>1</td>
<td>131338</td>
<td>3.25</td>
</tr>
<tr>
<td>Gran. Group.</td>
<td>367</td>
<td>6</td>
<td>61</td>
<td>0.64</td>
<td>0.70</td>
<td>70586</td>
<td>6</td>
<td>11764</td>
<td>0.29</td>
</tr>
<tr>
<td>Error</td>
<td>10715</td>
<td>112</td>
<td>96</td>
<td>452262</td>
<td>112</td>
<td>46381</td>
<td>112</td>
<td>46381</td>
<td>1</td>
</tr>
</tbody>
</table>

Data and Analysis - Selection

- Compare each techniques with retest-all
  - Modified entity
  - Modified non-core entity
  - Minimization
Data and Analysis - Selection

- Modified entity: same trends as retest-all
- Modified non-core entity
  - faults left undetected
  - FDE↑ when granularity↑ for both grouping
    - exception: FDE↓ when granularity↑ for functional grouping
  - Execution time ↓ when granularity ↑
Data and Analysis - Selection

- **Minimization**
  - Greater variance in FDE, less consistently increasing
    - Due to larger number of relatively hard-to-detect faults
  - **Execution time**
    - Emp-server: no much savings
    - Bash: time increases when granularity increases
      - Larger and varied functions -> redundancy at coarse granularity lever
      - Shows a negative effect of higher granularity
      - Depend on program and coverage pattern

Data and Analysis – Selection
Execution Time (Minimization)
### Analysis of Variance (ANOVA)

#### Table V. Retest-All and Modified Non-core Entity ANOVAs

**Variable: Fault-Detection Effectiveness**

<table>
<thead>
<tr>
<th>Source</th>
<th>Emp-Server</th>
<th>Bash</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SS</td>
<td>DF</td>
</tr>
<tr>
<td>Granularity</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>Grouping</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Technique</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>Granularity*Grouping</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Granularity*Technique</td>
<td>11</td>
<td>6</td>
</tr>
<tr>
<td>Grouping*Technique</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Gran.*Group.*Tech.</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>Error</td>
<td>92</td>
<td>224</td>
</tr>
</tbody>
</table>

**Variable: Test Execution Time**

<table>
<thead>
<tr>
<th>Source</th>
<th>Emp-Server</th>
<th>Bash</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SS</td>
<td>DF</td>
</tr>
<tr>
<td>Granularity</td>
<td>2851676</td>
<td>6</td>
</tr>
<tr>
<td>Grouping</td>
<td>3944</td>
<td>1</td>
</tr>
<tr>
<td>Technique</td>
<td>402157</td>
<td>1</td>
</tr>
<tr>
<td>Granularity*Grouping</td>
<td>4837</td>
<td>6</td>
</tr>
<tr>
<td>Granularity*Technique</td>
<td>75987</td>
<td>6</td>
</tr>
<tr>
<td>Grouping*Technique</td>
<td>1835</td>
<td>1</td>
</tr>
<tr>
<td>Gran.*Group.*Tech.</td>
<td>5121</td>
<td>6</td>
</tr>
<tr>
<td>Error</td>
<td>451100</td>
<td>224</td>
</tr>
</tbody>
</table>

#### What we learn from ANOVA

- **Emp-server**
  - Granularity and techniques have statistically significant impact

- **Bash**
  - Only technique had significant impact
  - How about interaction between granularity?
    - Bonferroni multiple comparison analysis (skipped)
What we learn from Bonferroni

- FDE:
  - granularity has no impact for retest-all technique
  - FDE decreases as granularity level increases for modified non-core technique
- Execution time
  - All three factors have significant impact
- Did NOT reveal significant effects by grouping
- Similarly, Bonferroni comparison analysis done with minimization and retest-all (skipped)

Data and Analysis - Reduction
Data and Analysis - Reduction

- GHS reduction produces similar results of modified non-core and minimization
- Overall trend: Effectiveness increases when granularity increases
  - Exception (Emp-server): at G8, effectiveness drop due to non-homogeneity of test cases
- Execution Time
  - Emp-server: decrease
  - Bash: increase (same reason as mentioned)

Results from Bonferroni

- Impact of grouping only occur on GHS reduction
- Grouping may significantly impact effects of other factors
Data and Analysis - Prioritization

- Optimal
  - Gives an upper bound
- Additional function coverage
  - Run those test cases which cover most number of functions first
- Additional modified function coverage
  - Run those test cases covering most number of modified function first
  - For those covering not-modified function using the normal function coverage prioritization

Fig. 6. APFD values for test case prioritization.
Data and Analysis - Prioritization

- Decrease in APFD when granularity increased, independent of grouping
- APFD values for bash lower than emp-server
  - The paper switched notation between emp-server and bash
- On bash, at higher granularity level, random grouping produces higher APFD values than that by functional grouping

Results from ANOVA

- Both programs
  - Granularity level and technique: significant effect on APFD
  - Interaction of these two: significant too
- Bash
  - Grouping is significant on APFD
### Summary for emp-server

#### Table XVI. Summary of Significant Effects for Emp-Server

<table>
<thead>
<tr>
<th></th>
<th>Re-test-All</th>
<th>Selection</th>
<th>Reduction</th>
<th>Prioritization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mod’d Noncore vs Re-test-All</td>
<td>Minimization vs Re-test-All</td>
<td>GHS vs Re-test-All</td>
<td>Coverage vs Optimal vs Optimal</td>
</tr>
<tr>
<td>exec</td>
<td>file</td>
<td>exec</td>
<td>file</td>
<td>exec</td>
</tr>
<tr>
<td>granularity</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>grouping</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>technique</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>gran^grp</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>gran^tech</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>grp^tech</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Columns headed “exec” pertain to execution time, and columns headed “file” pertain to fault detection effectiveness. “*” entries indicate cases in which the source of variation or interaction listed in column 1 was statistically significant, and “–” entries indicate cases where significance was not found.

### Summary for bash

#### Table XVII. Summary of Significant Effects for Bash

<table>
<thead>
<tr>
<th></th>
<th>Re-test-All</th>
<th>Selection</th>
<th>Reduction</th>
<th>Prioritization</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mod’d Noncore vs Re-test-All</td>
<td>Minimization vs Re-test-All</td>
<td>GHS vs Re-test-All</td>
<td>Coverage vs Optimal vs Optimal</td>
</tr>
<tr>
<td>exec</td>
<td>file</td>
<td>exec</td>
<td>file</td>
<td>exec</td>
</tr>
<tr>
<td>granularity</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>grouping</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>technique</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>gran^grp</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>gran^tech</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>grp^tech</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>gran^grp^tech</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Columns headed “exec” pertain to execution time, and columns headed “file” pertain to fault detection effectiveness. “*” entries indicate cases in which the source of variation or interaction listed in column 1 was statistically significant, and “–” entries indicate cases where significance was not found.
Discussion

- H1 (granularity): results supports significant impact
- H2 (grouping): results not able to reject this hypothesis
  - Grouping only has significant impact on
    - GHS reduction, diff-coverage prioritization
- H3 (techniques): not significantly different performance
- H4 (interactions): discover frequent interaction between granularity and technique

Implications for common practice (retest-all)

- Use of coarse granularity test suites increases efficiency for retest-all
- Some qualification on this conclusion
  - Savings on execution time may due to overhead and clear-up
  - Fine granularity -> prioritization and localize faults
  - Program complexity relative to input size
  - Test oracle accuracy
Implication for Selection

- Safe RTS = retest-all in our experiments
- Non-safe RTS
  - Aggressive minimization technique
    - Granularity has less influence than other factors
  - Less aggressive modified non-core entity
    - Finer granularity
- Results show fault difficulty can influence granularity

Implication for Reduction

- Difference between minimization and reduction
  - Larger test suite size
- Same implication on test suite granularity as for minimization RTS
  - Mixed
  - No suggestion yet
- Grouping
  - Functional grouping better
  - The above doesn’t hold w.r.t. hard-to-detect faults
Implication for Prioritization

- Finer granularity preferred
  - Greater opportunities for prioritization
  - Higher APFD values
- Significant granularity accompanied by significance in techniques

Conclusion

- Success in finding faults can vary
  - Test suite granularity
  - Test input grouping
- This paper highlights cost-benefits associated with these two factors
- Granularity: significant impact
- Grouping: limited impact