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 contains redundant test cases
- GHS reduction technique used
  - A heuristic proposed by Gupta, Harrold and Soffa
  - Attempts to produce suites that are minimal for a give 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

Objects of Analysis

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

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

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

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

Variables and Measurements

- 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 emp-server; 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

$$APFD = 1 - \frac{\sum_{i=1}^{m} Tfi}{mn} + \frac{1}{2n}$$

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

Experiment Setup

- 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

	Emp-Server	Bash
G1	1985	1168
G2	993	584
G4	497	292
G8	249	146
G16	125	73
G32	63	37
G64	32	19

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

Program	G2	G4	G8	G16	G32	G64
emp-server	95.0	89.0	72.0	35.0	12.0	00.0
bash	95.0	98.0	95.5	90.4	78.4	63.2

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

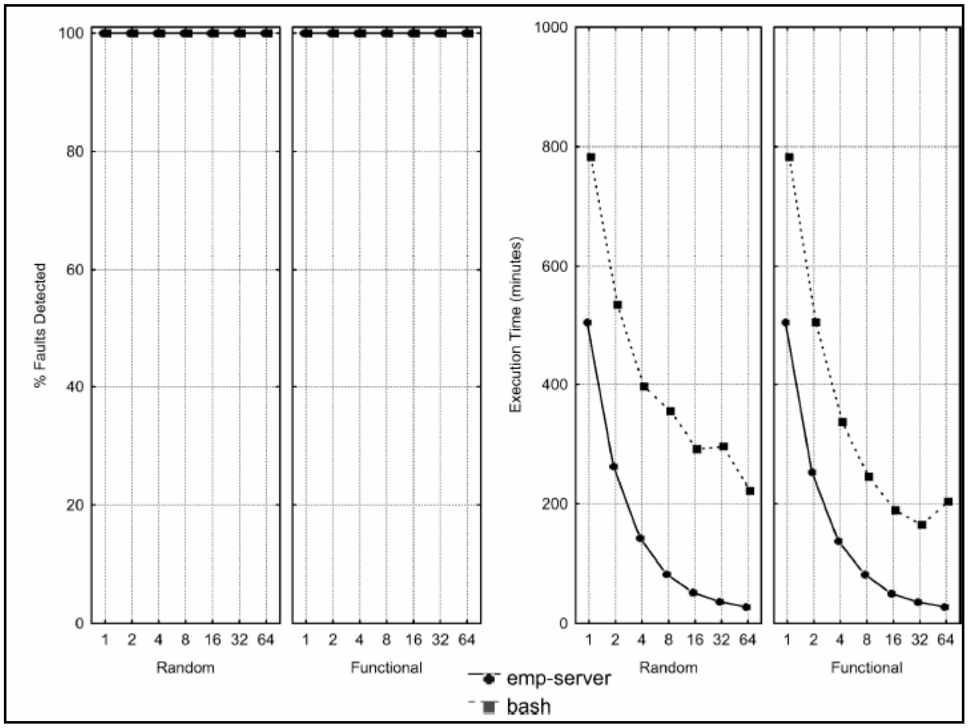
- Internal Validity
  - Human factor, e.g fault seeding
  - Only 1 test suite, however, multiple test suite is costy
  - Masking effect: infrequently
  - Magnitude and distribution of changes between versions
- External Validity
  - Quantity and quality of programs studied
  - Fault representativeness

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

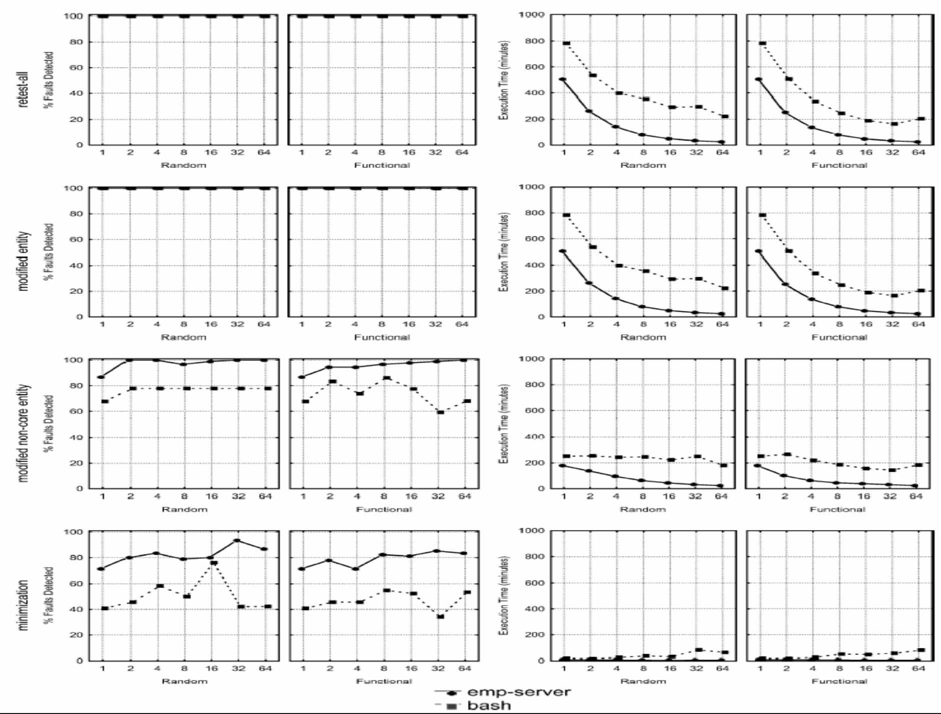
Analysis of Variance (ANOVA)

Table IV. Retest-All ANOVA

Technique: Retest-All										
Variable: *Test Execution Time.*										
	Emp-Server					Bash				
Source	SS	DF	MS	F	p	SS	DF	MS	F	p
Granularity	3268098	6	544683	5693.41	*0.00*	4635964	6	772661	19.13	*0.00*
Grouping	199	1	199	2.08	0.15	131338	1	131338	3.25	0.07
Gran.* Group.	367	6	61	0.64	0.70	70586	6	11764	0.29	0.94
Error	10715	112	96			4522624	112	40381		

Data and Analysis - Selection

- Compare each techniques with retest-all
  - Modified entity
  - Modified non-core entity
  - Minimization



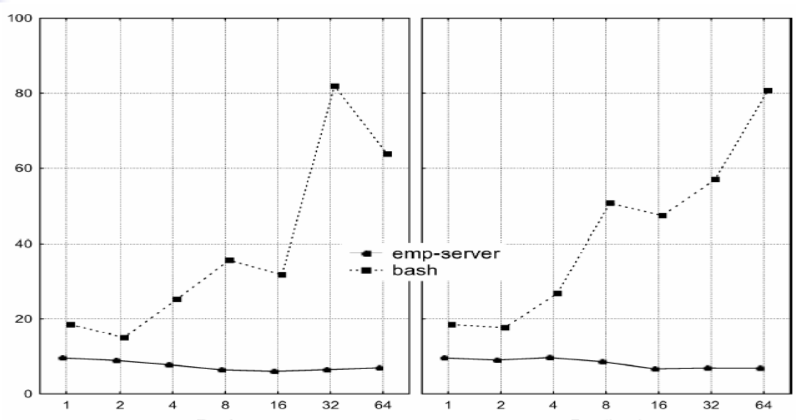
Data and Analysis - Selection

- 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 Noncore Entity ANOVAs

Techniques: Modified Non-Core Entity and Retest-All										
Variable: *Fault-Detection Effectiveness*										
	Emp-Server					Bash				
Source	SS	DF	MS	F	p	SS	DF	MS	F	p
Granularity	9	6	1	3.97	*0.00*	28	6	5	0.53	0.78
Grouping	0	1	0	1.08	0.30	12	1	12	1.37	0.24
Technique	8	1	8	20.87	*0.00*	70	1	70	8.00	*0.01*
Granularity*Grouping	3	6	0	1.23	0.29	35	6	6	0.67	0.67
Granularity*Technique	11	6	2	4.97	*0.00*	5	6	1	0.09	1.00
Grouping*Technique	1	1	1	1.55	0.21	1	1	1	0.08	0.78
Gran.*Group.*Tech.	1	6	0	0.40	0.88	3	6	0	0.05	1.00
Error	82	224	0			1967	224	9		
Variable: *Test Execution Time*										
	Emp-Server					Bash				
Source	SS	DF	MS	F	p	SS	DF	MS	F	p
Granularity	2851876	6	475313	236.02	*0.00*	2959544	6	493257	12.55	*0.00*
Grouping	3944	1	3944	1.96	0.16	154812	1	154812	3.94	0.05
Technique	402157	1	402157	199.70	*0.00*	1629161	1	1629161	41.44	*0.00*
Granularity*Grouping	4837	6	806	0.40	0.88	119699	6	19950	0.51	0.80
Granularity*Technique	758867	6	126478	62.80	*0.00*	1779040	6	296507	7.54	*0.00*
Grouping*Technique	1835	1	1835	0.91	0.34	14175	1	14175	0.36	0.55
Gran.*Group.*Tech.	2121	6	353	0.18	0.98	3533	6	589	0.02	1.00
Error	451100	224	2014			8807271	224	39318		

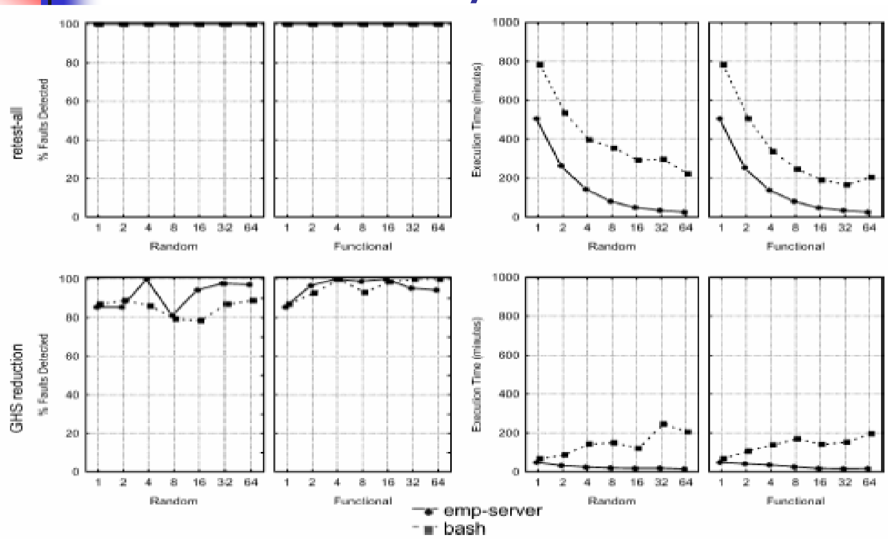
What we learn from ANOVA

- 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

Data and Analysis - Prioritization

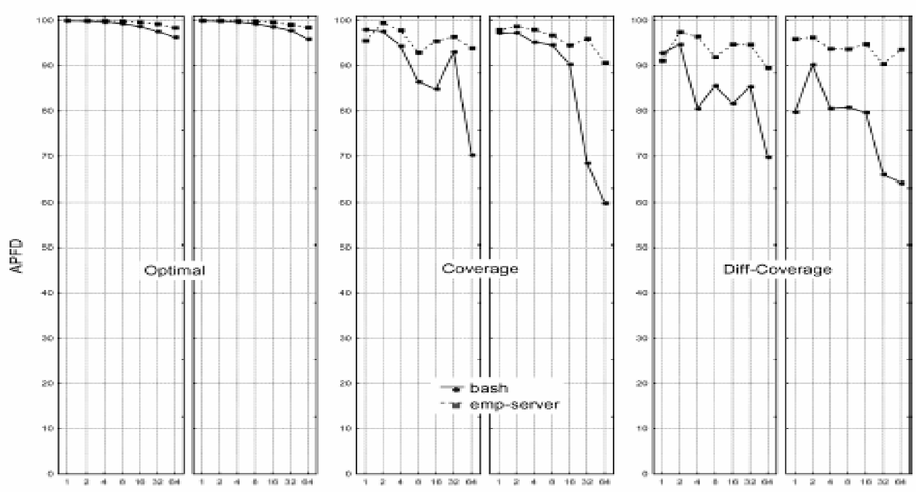


Fig. 6. APFD values for test case prioritization.

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

	Retest-All		Selection				Reduction		Prioritization	
			Mod'd Noncore vs Retest-All		Minimization vs Retest-All		GHS vs Retest-All		Coverage vs Optimal	Diff-Coverage vs Optimal
	exec	fde	exec	fde	exec	fde	exec	fde	apfd	apfd
granularity	*	–	*	*	*	–	*	*	*	*
grouping	–	–	–	–	–	–	–	*	–	*
technique			*	*	*	*	*	*	*	*
gran*grp	–	–	–	–	–	–	–	*	–	*
gran*tech			*	*	*	–	*	*	*	*
grp*tech			–	–	–	–	*	*	–	–
gran*grp*tech			–	–	–	–	*	*	–	*

Columns headed "exec" pertain to execution time, and columns headed "fde" 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

	Retest-All		Selection				Reduction		Prioritization	
			Mod'd Noncore vs Retest-All		Minimization vs Retest-All		GHS vs Retest-All		Coverage vs Optimal	Diff-Coverage vs Optimal
	exec	fde	exec	fde	exec	fde	exec	fde	apfd	apfd
granularity	*	–	*	–	*	–	*	–	*	*
grouping	–	–	–	–	–	–	–	–	–	*
technique			*	*	*	*	*	*	*	*
gran*grp	–	–	–	–	–	–	–	–	–	–
gran*tech			*	–	*	–	*	–	–	*
grp*tech			–	–	–	–	–	–	–	*
gran*grp*tech			–	–	–	–	–	–	–	–

Columns headed "exec" pertain to execution time, and columns heade "fde" 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