Test Generation and Coverage

ICSE 2013 Tutorial

Automated Testing of GUI Applications
Models, Tools and Controlling Flakiness

Atif M. Memon and Myra B. Cohen

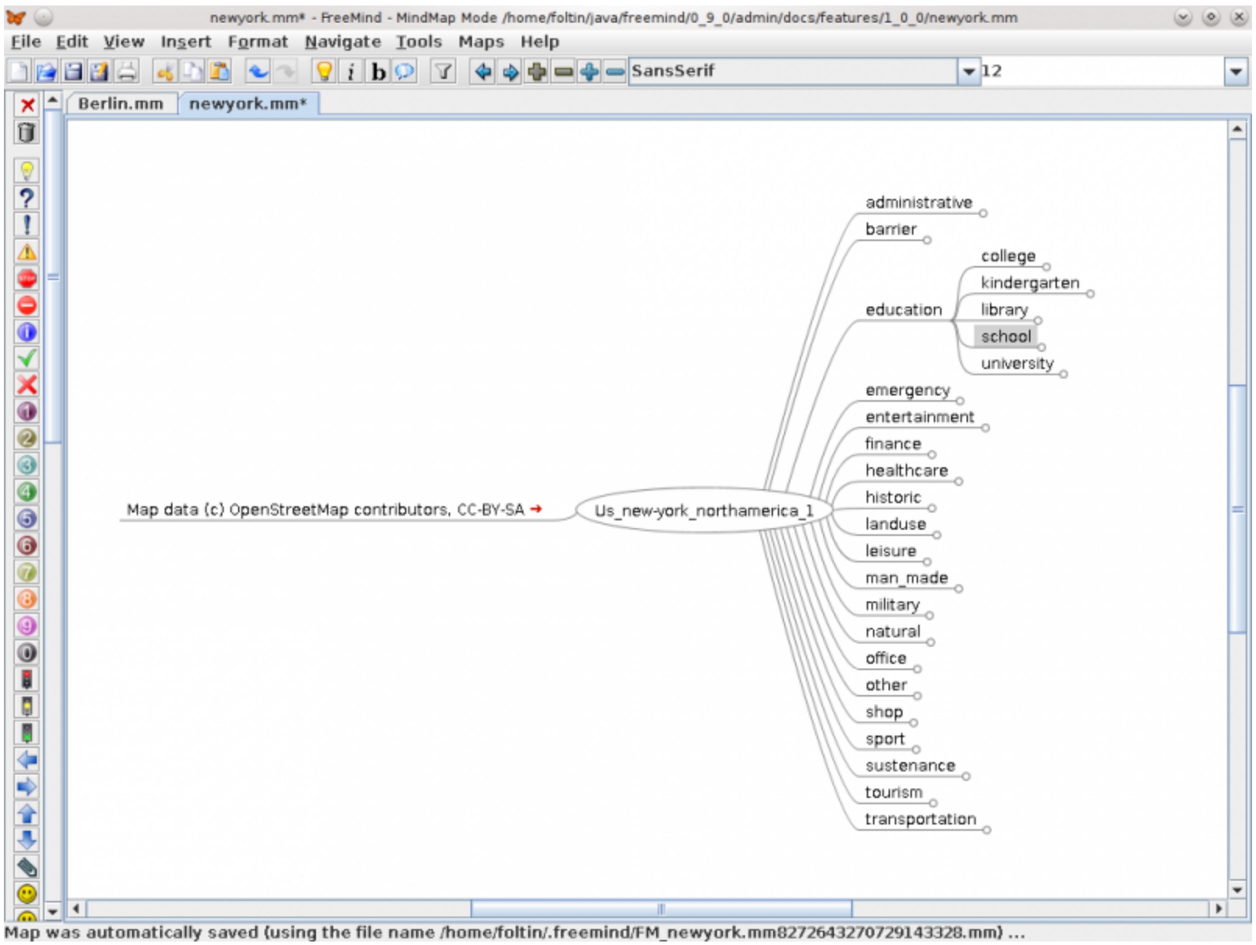
University of Maryland

University of Nebraska Lincoln

Adding Context

Lets return to our fault example from the beginning…

A Concrete Example



Adding Context

Setup:

In the Freemind Application there are four events: e_1, e_2, e_3, e_4, that correspond to {*SetNodeToCloud*, *NewChildNode*, *NewParentNode*, *Undo*}

A Fault:

Execution of either <e_1,e_2,e_3> or <e_2,e_1, e_3> throws an *ArrayIndexOutOfBounds* exception

Adding Context

Context Matters:

The combination $<e_2, e_3>$ triggers this failure, but it needs the context of a cloud node (e_1).

If we test $<e_2, e_3, e_1>$ the fault is not detected

Consecutive Events:

While $<e_1, e_2, e_3>$ triggers the fault, but inserting e_4 (undo), at points in the sequence (e.g. $<e_1, e_2, e_4, e_3>$ causes the failure to be missed

Adding Context

Position is Important:

If the sequence <$e_{1,}e_2$> appears at the start of the test case, we have no chance of detecting this fault

However, if it appears later then a sequence such as <…e_1,…, $e_{2,..}$> may trigger it

Coverage of Event Graphs

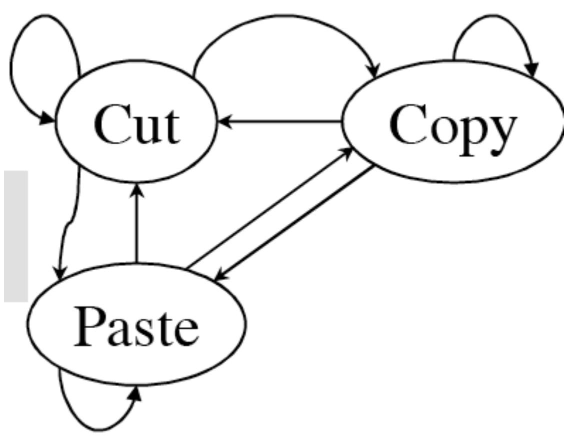
- Different types of coverage have been defined to reduce the complexity of EIG and EFG relationships
  - Cover all edges of EIG (length 2 test cases)
  - Cover all k-paths in graph, k > 2
  - Randomly select longer paths in graph

Example
(3 interacting events)

<Cut>

<Copy>

<Paste>



9 length 2 sequences

27 length 3 sequences

59,049 length 10 sequences

14 Million + length 15 sequences!

Possible Options

- Exhaustively generate all short paths:
  - length 2 (smoke tests)
  - or length 3
- Randomly generate longer paths
- Systematically sample paths

Shorter Sequences

Length 2 (Smoke Tests)
1.<Cut, Cut>
2.<Cut, Copy>
3.<Cut, Paste>
4.<Copy, Copy>
5.<Copy, Paste>
6.<Copy, Cut>
7.<Paste, Paste>
8.<Paste, Cut>
9.<Paste, Copy>

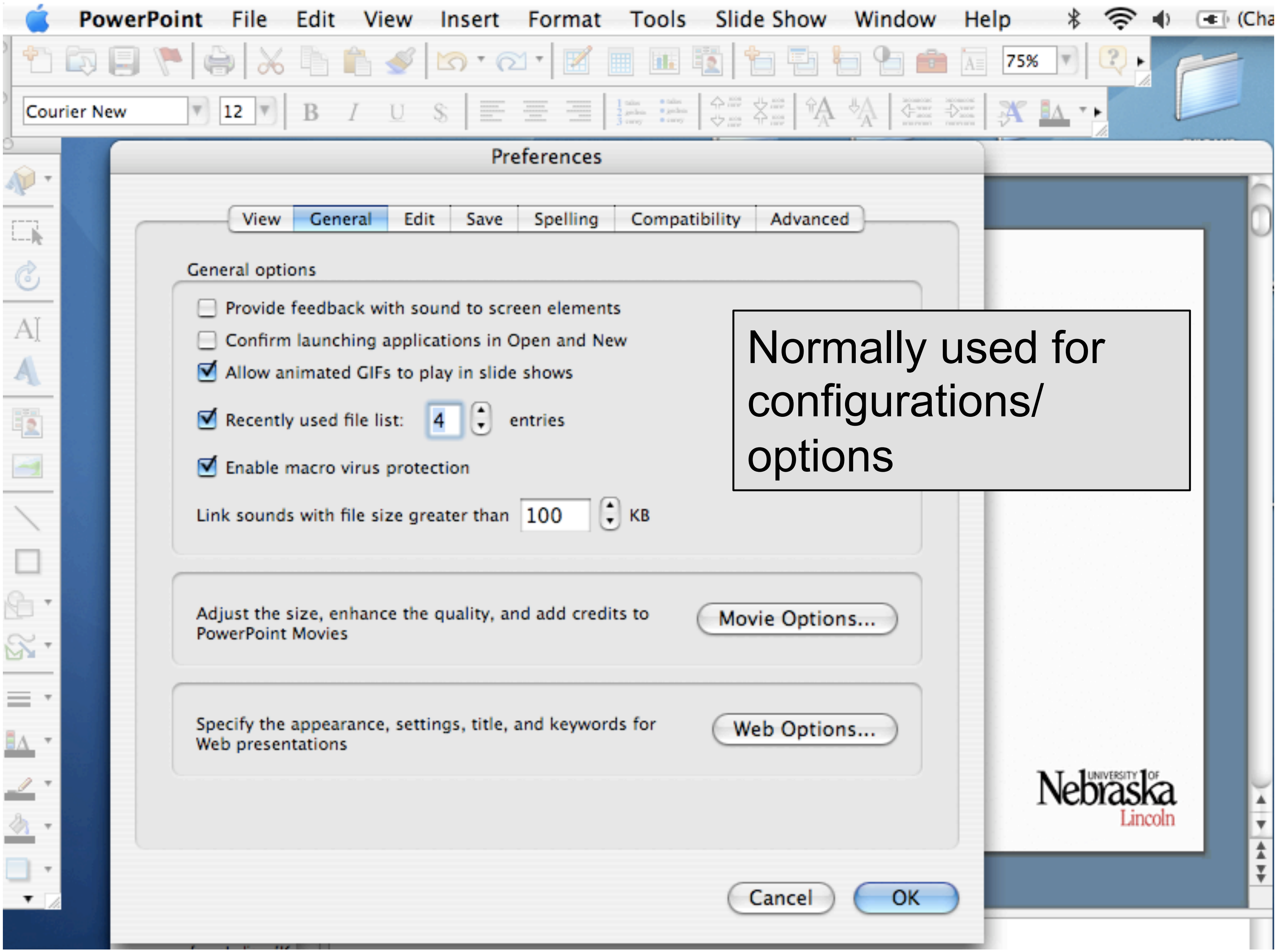
Length 3-way
1.<Cut, Cut, Cut>
2.<Cut, Cut, Copy>
3.<Cut, Cut, Paste>
4.<Cut, Copy, Cut>
5.<Cut, Copy, Paste>
.
.
.
27.<Paste, Paste, Paste>

But!

- Research has shown that longer sequences find more faults
- The Freemind fault would go undetected with only length 2 sequences

Sampling Technique

- We can modify a technique from functional input testing:
  - *Combinatorial interaction testing* (CIT)
  - Covers all pairs or t-way combinations, but does not consider position
  - Used extensively to sample inputs, configurations, protocols
  - Tools exist to automatically generate samples



Normally used for configurations/ options

Combinatorial Testing (CIT)

	1	2	3	4
1	*Cut*	*Cut*	*Cut*	*Cut*
2	*Cut*	*Paste*	*Paste*	*Copy*
3	*Cut*	*Copy*	*Copy*	*Paste*
4	*Copy*	*Cut*	*Paste*	*Paste*
5	*Copy*	*Copy*	*Cut*	*Copy*
6	*Copy*	*Paste*	*Copy*	*Cut*
7	*Paste*	*Cut*	*Copy*	*Copy*
8	*Paste*	*Paste*	*Cut*	*Paste*
9	*Paste*	*Copy*	*Paste*	*Cut*

Combine all pairs between locations (2-way)

Combinatorial Testing (CIT)

1	*Cut*	*Cut*	*Cut*	*Cut*
2	*Cut*	*Paste*	*Paste*	*Copy*
3	*Cut*	*Copy*	*Copy*	*Paste*
4	*Copy*	*Cut*	*Paste*	*Paste*
5	*Copy*	*Copy*	*Cut*	*Copy*
6	*Copy*	*Paste*	*Copy*	*Cut*
7	*Paste*	*Cut*	*Copy*	*Copy*
8	*Paste*	*Paste*	*Cut*	*Paste*
9	*Paste*	*Copy*	*Paste*	*Cut*

Combinatorial Testing (CIT)

1	Cut	Cut	Cut	Cut
2	Cut	Paste	Paste	Copy
3	Cut	Copy	Copy	Paste
4	Copy	Cut	Paste	Paste
5	Copy	Copy	Cut	Copy
6	Copy	Paste	Copy	Cut
7	Paste	Cut	Copy	Copy
8	Paste	Paste	Cut	Paste
9	Paste	Copy	Paste	Cut
	…..	… 3-way CIT	…	…
27				

Relaxed CIT Coverage

e_1	e_1
e_1	e_2
e_2	e_1
e_2	e_2

2-cover

e_1	e_2	*	*	*
e_1	*	e_2	*	*
e_1	*	*	e_2	*
e_1	*	*	*	e_2
*	e_1	e_2	*	*
*	e_1	*	e_2	*
*	e_1	*	*	e_2
*	*	e_1	e_2	*
*	*	e_1	*	e_2
*	*	*	e_1	e_2

<u>event-2-sequences</u>

<(e_1,1), (e_2,2)>
<(e_1,1), (e_2,3)>
<(e_1,1), (e_2,4)>
.
.
.
<(e_1,4), (e_2,5)>

All possible *event-2-sequences* for *event-2-tuple* (e1,e2)

Three Length 5 Test Sequences

e_1	e_2	e_1	e_2	e_2
e_1	e_1	e_2	e_1	e_1
e_2	e_2	e_2	e_1	e_2

Definitions

Definition

An *event-t-tuple* $(e_i, e_j, \ldots, e_t)$ is an ordered tuple of size t of events from E. A set of events E gives rise to $|E|^t$ *event-t-tuples*, *i.e.*, all possible permutations of events.

To account for context, we need to associate positions within a sequence to specific events.

Definition

An *event-position* in a length-k sequence S is an ordered pair (e, p), where event $e \in E$ is at position p $(1 \leq p \leq k)$.

Definitions

Definition

An *event-t-sequence* is a vector of *event-positions* of length t, $<(e_i, p_1), (e_j, p_2), ..., (e_n, p_t)>$, where $k \geq t$, $1 \leq p_x \leq k$ for all x, $1 \leq x \leq t$, $p_1 < p_2 < ... < p_t$, and $e_i, e_j, ...e_n \in E$.

Definition

An *event-consecutive-t-sequence* is an *event-t-sequence* $<(e_i, p_1), (e_j, p_2), ..., (e_k, p_t)>$ such that $p_x = p_{x-1} + 1$, for all $1 < x \leq t$.

Definition

An *event-non-consecutive-t-sequence* is an *event-t-sequence* $<(e_i, p_1), (e_j, p_2), ...(e_n, p_t)>$, where $p_1 < p_2 < ... < p_t$, such that at least one interval $(p_2 - p_1), ..., (p_t - p_{t-1})$ is greater than 1.

Coverage Adequacy

Definition

A test suite is *t-cover* adequate if it executes all possible *event-t-tuples* in the form of an *event-consecutive-t-sequence* at least once.

Definition

A test suite is *t^+-cover* adequate if it executes all possible *event-t-tuples* in the form of an *event-non-consecutive-t-sequence* at least once. Adequacy is zero when $t = k$.

Coverage Adequacy

Definition

A test suite is *t^*-cover* adequate if it is both t-*cover* adequate and t^+- *cover* adequate for a common set of events E.

Definition

A test suite with test cases of length k is *t-k-covering array* adequate when $t < k$ and it contains all possible *event-t-sequences* at least once.

Example

Three events:
{ClearCanvas, DrawCircle, Refresh}

Example

t-**Cover**			
ClearCanvas	ClearCanvas	Refresh	DrawCircle
DrawCircle	Refresh	ClearCanvas	Refresh
Refesh	Refresh	DrawCircle	ClearCanvas
ClearCanvas	DrawCircle	DrawCircle	Refresh

Missing t^+-cover tuples:
$\{ClearCanvas, ClearCanvas\}, \{Refresh, Refresh\}$

Example

t^+**-Cover**			
ClearCanvas	Refresh	DrawCircle	Refresh
Refresh	ClearCanvas	DrawCircle	ClearCanvas
DrawCircle	Refresh	ClearCanvas	Refresh
DrawCircle	ClearCanvas	Refresh	DrawCircle

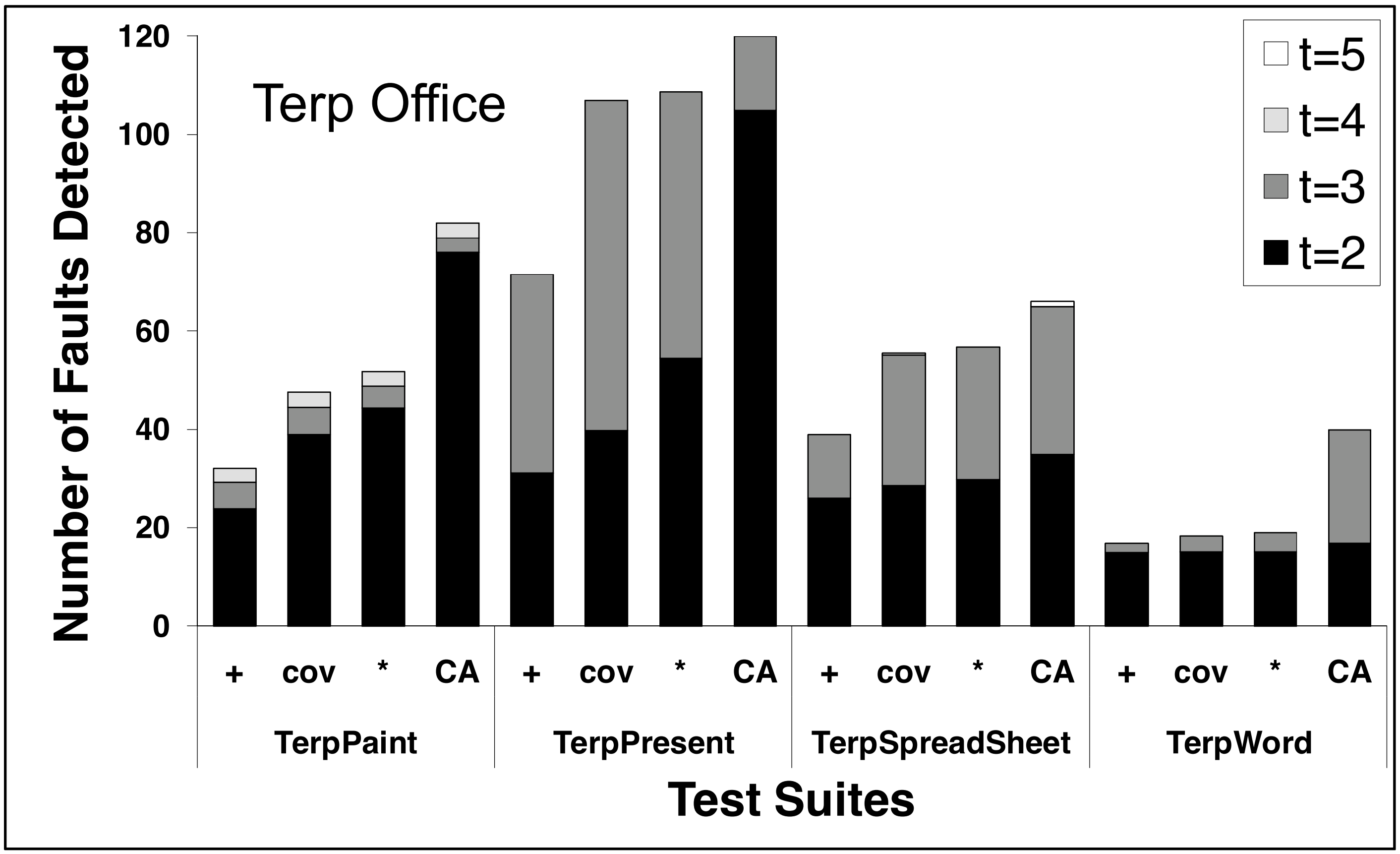
Missing t-cover tuples:
{*ClearCanvas,ClearCanvas*},{*DrawCircle,DrawCircle*},
{*Refresh,Refresh*}

Study Overview

- Conducted an empirical study on 4 TerpOffice Applications[1] and 4 Open Source Applications[2]
- Generated all 2,3,4,5 - CIT samples of length 10 (until TS size > 20,000)
- TerpOffice - seeded faults
- Open Source - detected crashes
- All faults undetected by 2-way sequences
- Compared size and fault detection at each strength

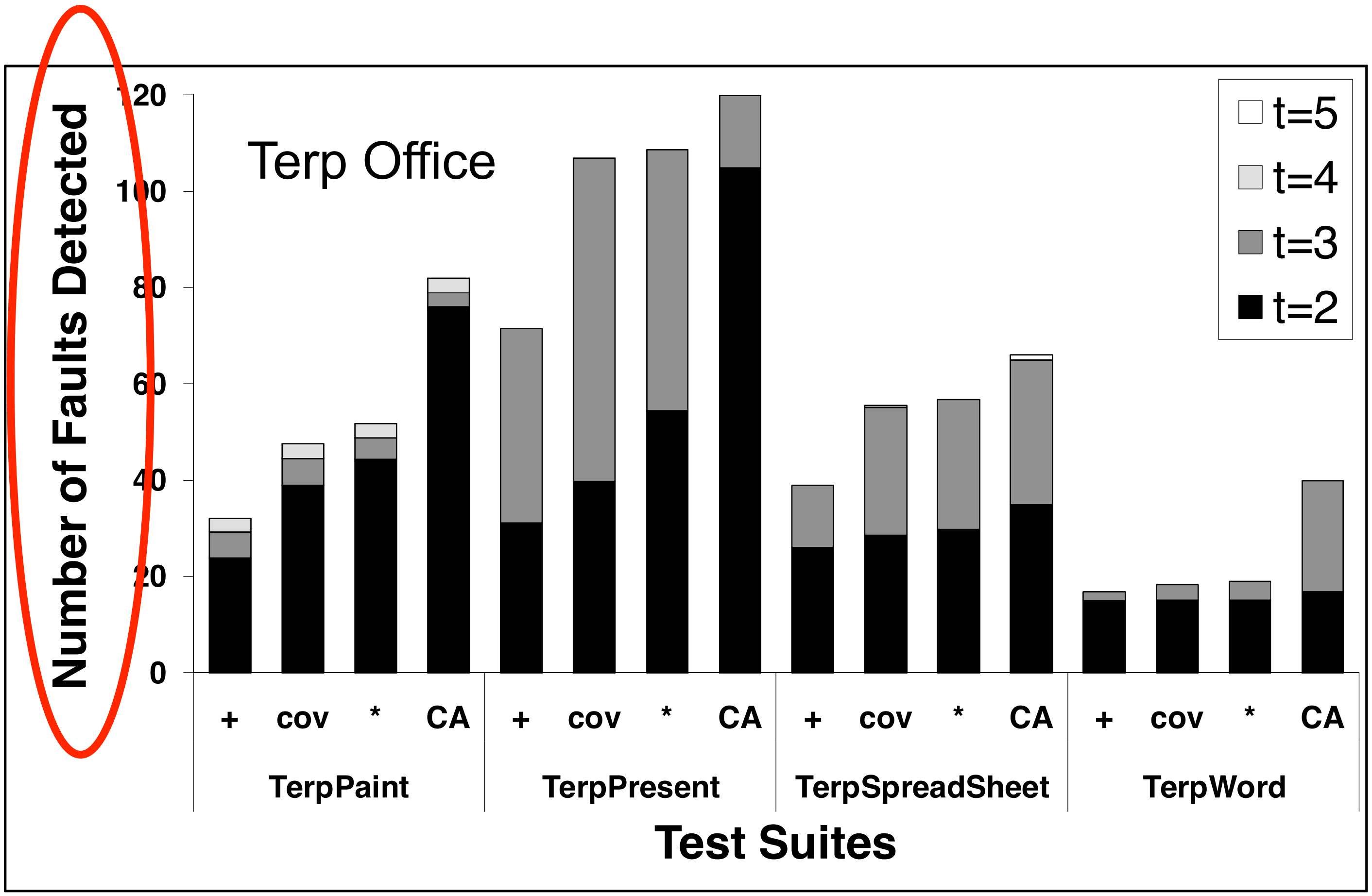
1. University of Maryland
2. SourceForge

Results



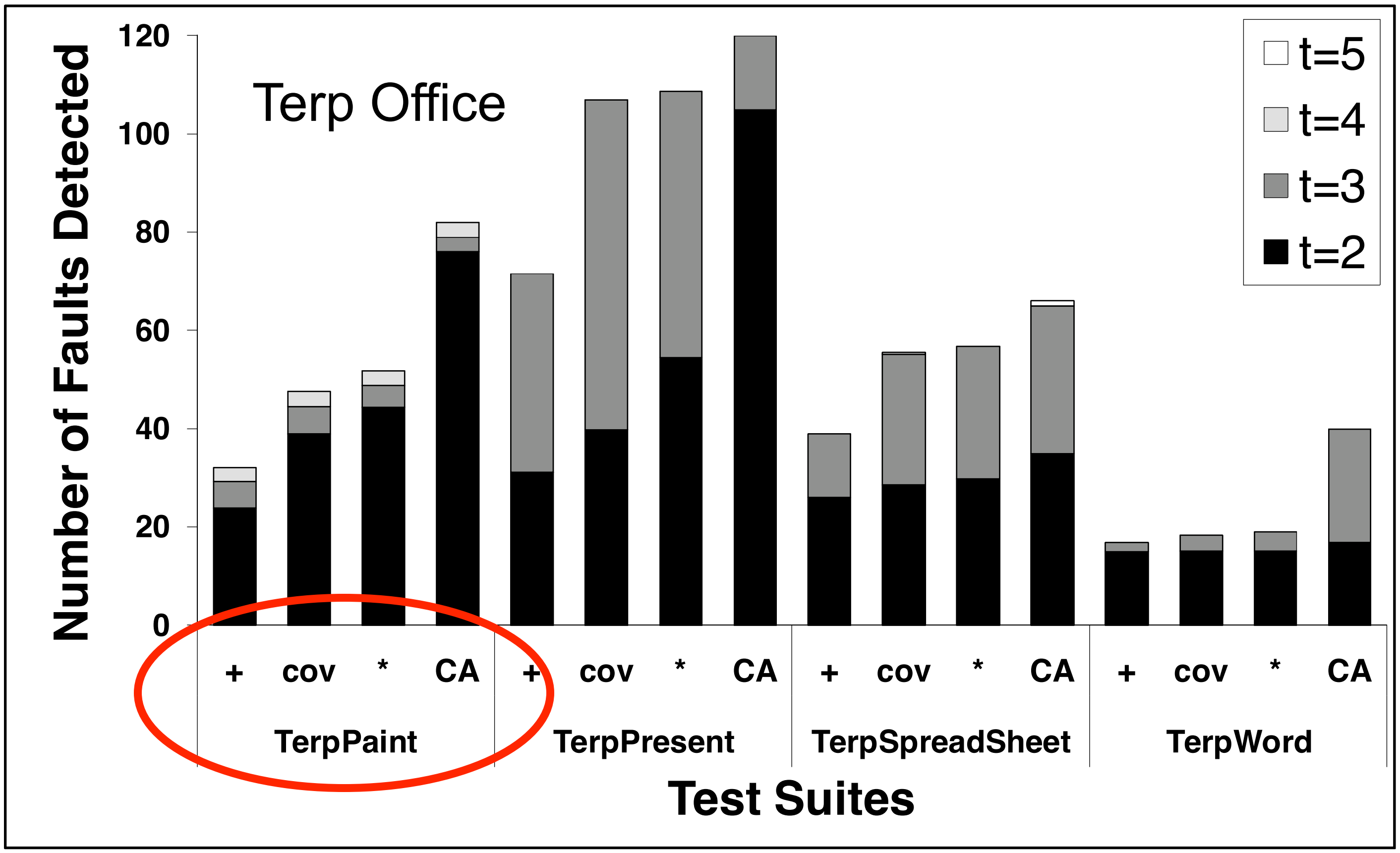
[Yuan et al.TSE 11]

Results

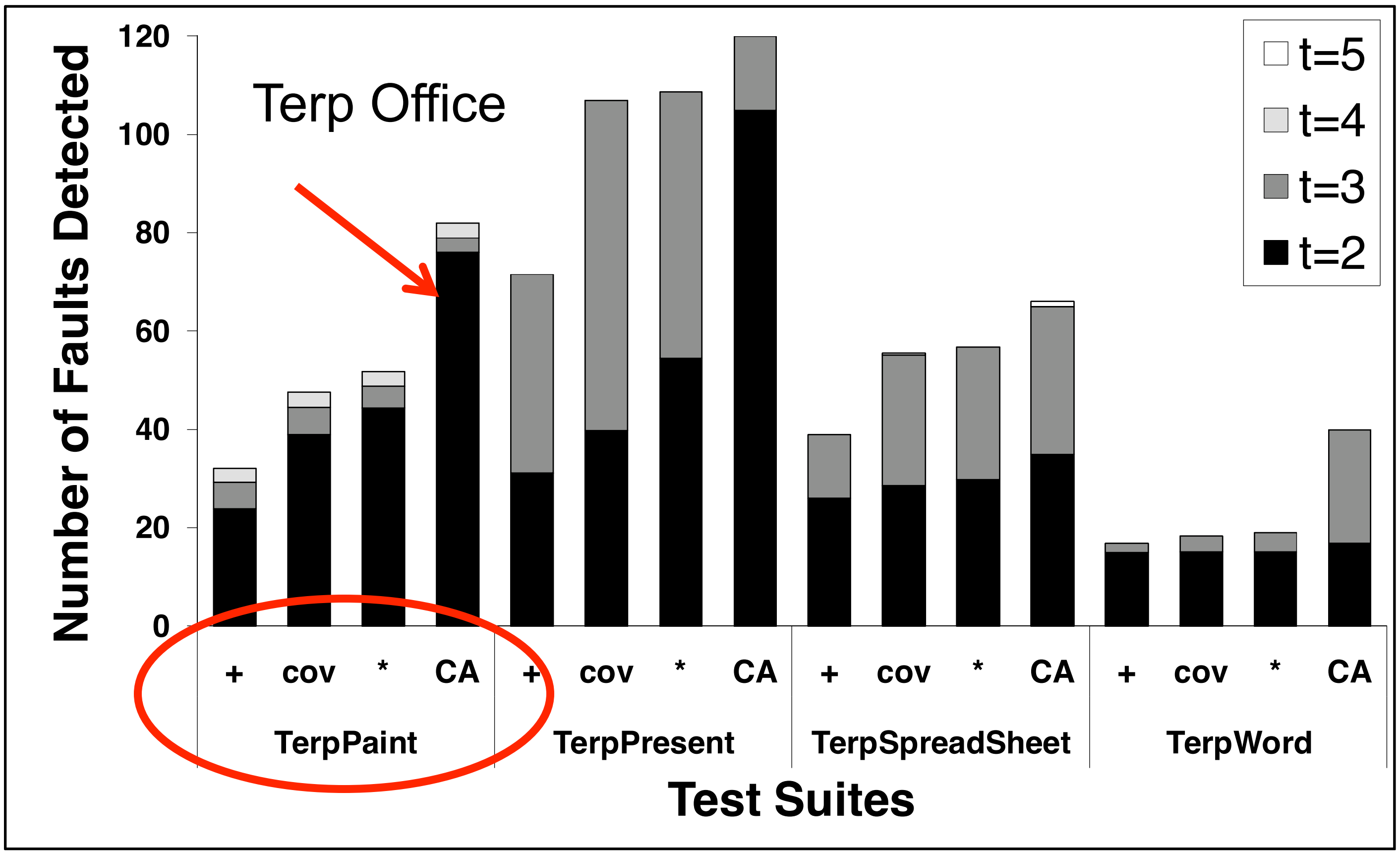


Terp Office
Number of Faults Detected
120
100
80
60
40
20
0
t=5
t=4
t=3
t=2
+
cov
*
CA
TerpPaint
TerpPresent
TerpSpreadSheet
TerpWord
Test Suites

Results

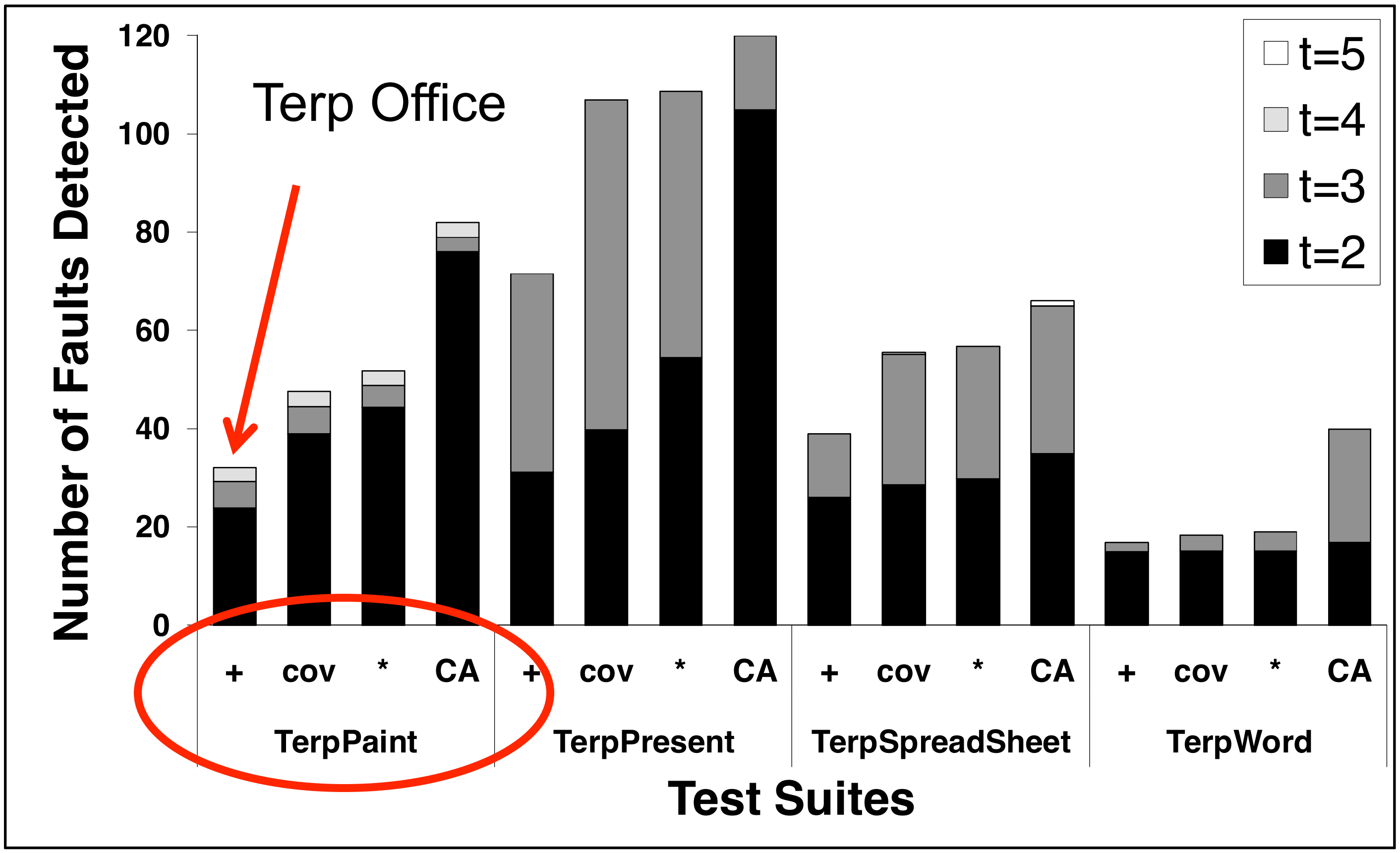

Terp Office
Number of Faults Detected
120
100
80
60
40
20
0
t=5
t=4
t=3
t=2
+
cov
*
CA
TerpPaint
TerpPresent
TerpSpreadSheet
TerpWord
Test Suites

Results



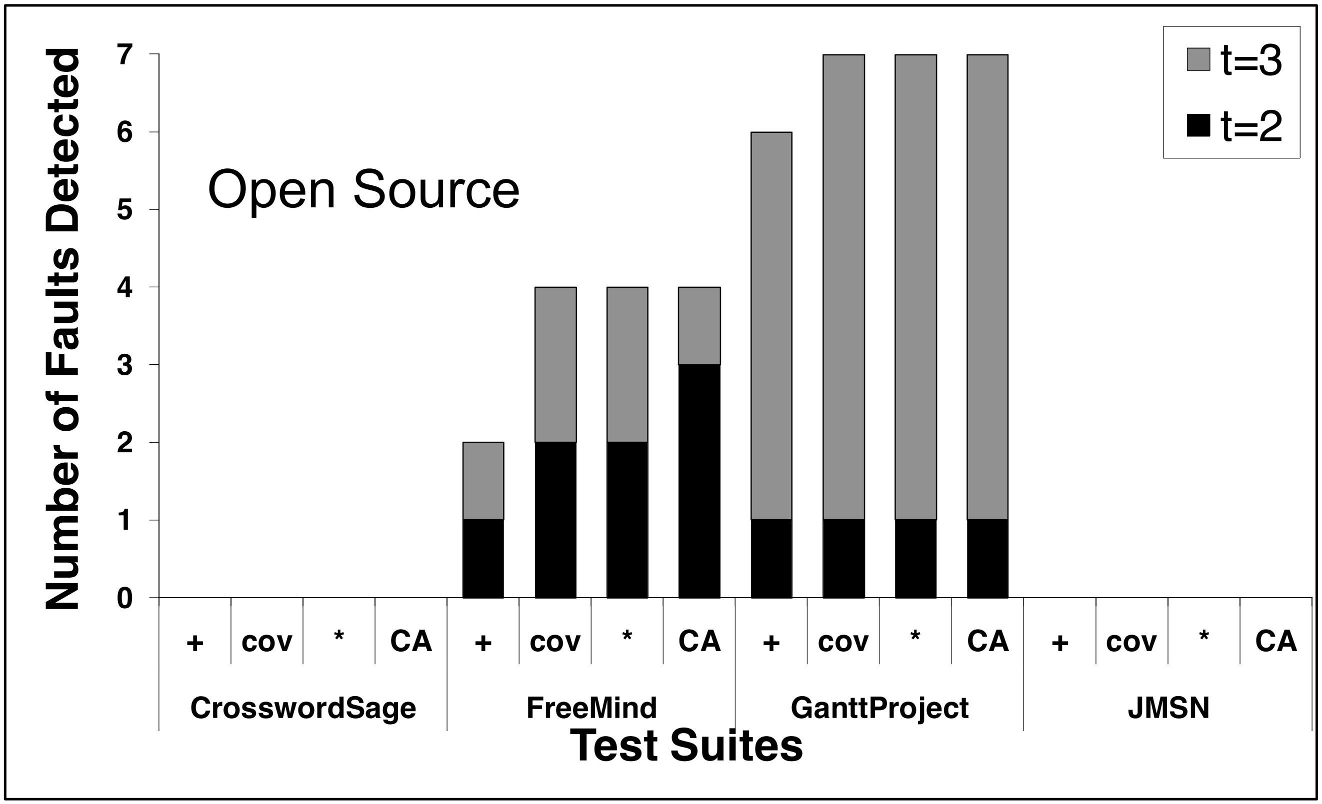
Number of Faults Detected
120
100
80
60
40
20
0
Terp Office
t=5
t=4
t=3
t=2
+
cov
*
CA
+
cov
*
CA
+
cov
*
CA
+
cov
*
CA
TerpPaint
TerpPresent
TerpSpreadSheet
TerpWord
Test Suites

Results



Terp Office
t=5
t=4
t=3
t=2
Number of Faults Detected
0
20
40
60
80
100
120
+
cov
*
CA
TerpPaint
TerpPresent
TerpSpreadSheet
TerpWord
Test Suites

Results



Open Source
t=3
t=2
Number of Faults Detected
7
6
5
4
3
2
1
0
+
cov
*
CA
+
cov
*
CA
+
cov
*
CA
+
cov
*
CA
CrosswordSage
FreeMind
GanttProject
JMSN
Test Suites

Cost Trade-off

- The criteria with fewer test cases find fewer faults

Group	Strength	t^{+}-*cover*	t-*cover*	t^{*}-*cover*	t-10-CA
TerpPaint					
Group 1	t=2	19.0/43.8	33.0/123.4	38.0/161.8	69/1055
Group 2	t=2	3.4/73.6	4.0/208.6	4.0/273.4	4/1783
Group 3	t=2	1.4/8.8	2.4/22.6	2.8/28.8	5/171
	t=3	6.8/40.2	8.0/255.8	8.0/291.8	8/2870
Group 5	t=4	5.0/22.0	5.0/81.4	5.0/84.2	5/3428

Avg. # Faults detected

Avg. Size of Test Suite

Cost Trade-off

- The criteria with fewer test cases find fewer faults

Group	Strength	t^+-*cover*	t-*cover*	t^*-*cover*	t-10-CA
TerpPaint					
Group 1	t=2	19.0/43.8	33.0/123.4	38.0/161.8	69/1055
Group 2	t=2				4/1783
Group 3	t=2				5/171
	t=3				8/2870
Group 5	t=4	5.0/22.0	5.0/81.4	5.0/84.2	5/3428

19 faults found: 44 test cases
69 faults found: 1055 test cases

Avg. # Faults detected

Avg. Size of Test Suite

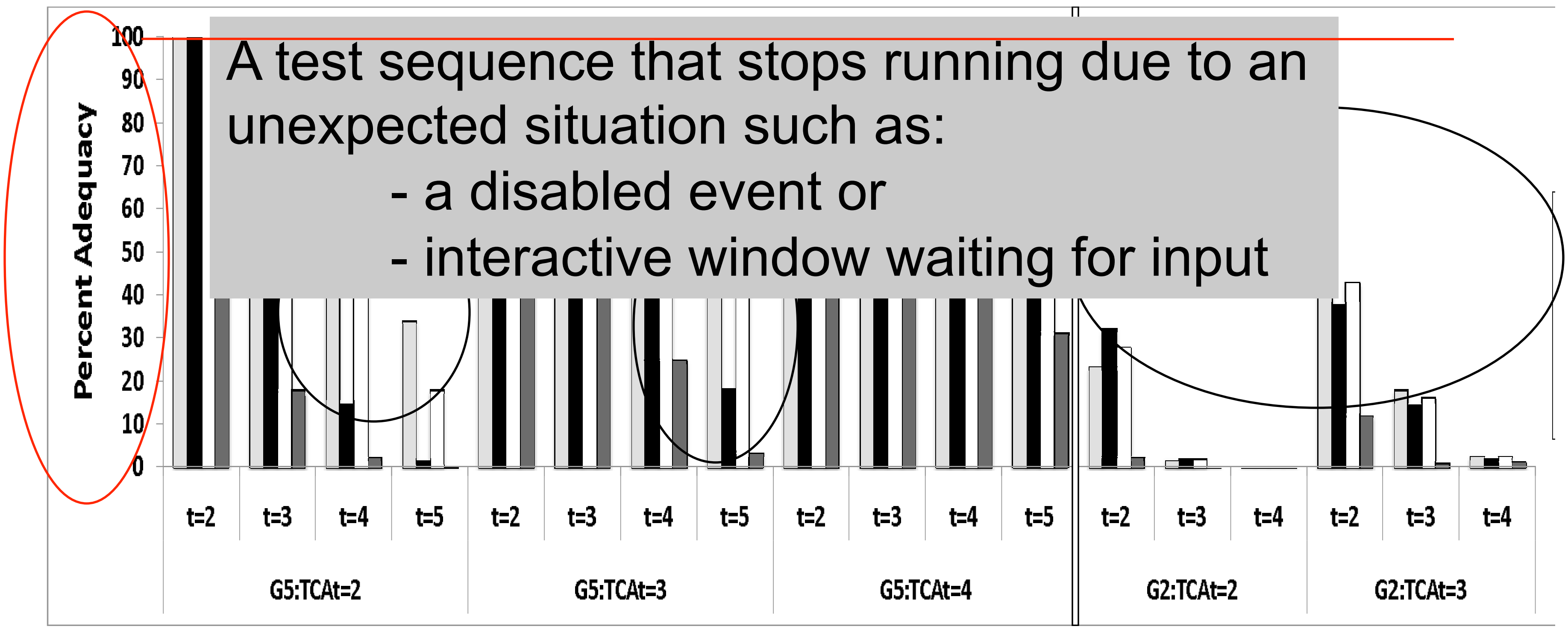
Cost Trade-off

- The criteria with fewer test cases find fewer faults

3 faults found: 73 test cases
4 faults found: 1783 test cases

Group	Strengt				t-10-CA
Group 1	t=2				69/1055
Group 2	t=2	3.4/73.6	4.0/208.6	4.0/273.4	4/1783
Group 3	t=2	1.4/8.8	2.4/22.6	2.8/28.8	5/171
	t=3	6.8/40.2	8.0/255.8	8.0/291.8	8/2870
Group 5	t=4	5.0/22.0	5.0/81.4	5.0/84.2	5/3428

Many Infeasible Test Sequences



Problem with Infeasible Sequences

- Causes test harness to hang
  - Entire test suite may not run
- Reduces CIT coverage
- May reduce fault detection ability
- Requires manual intervention

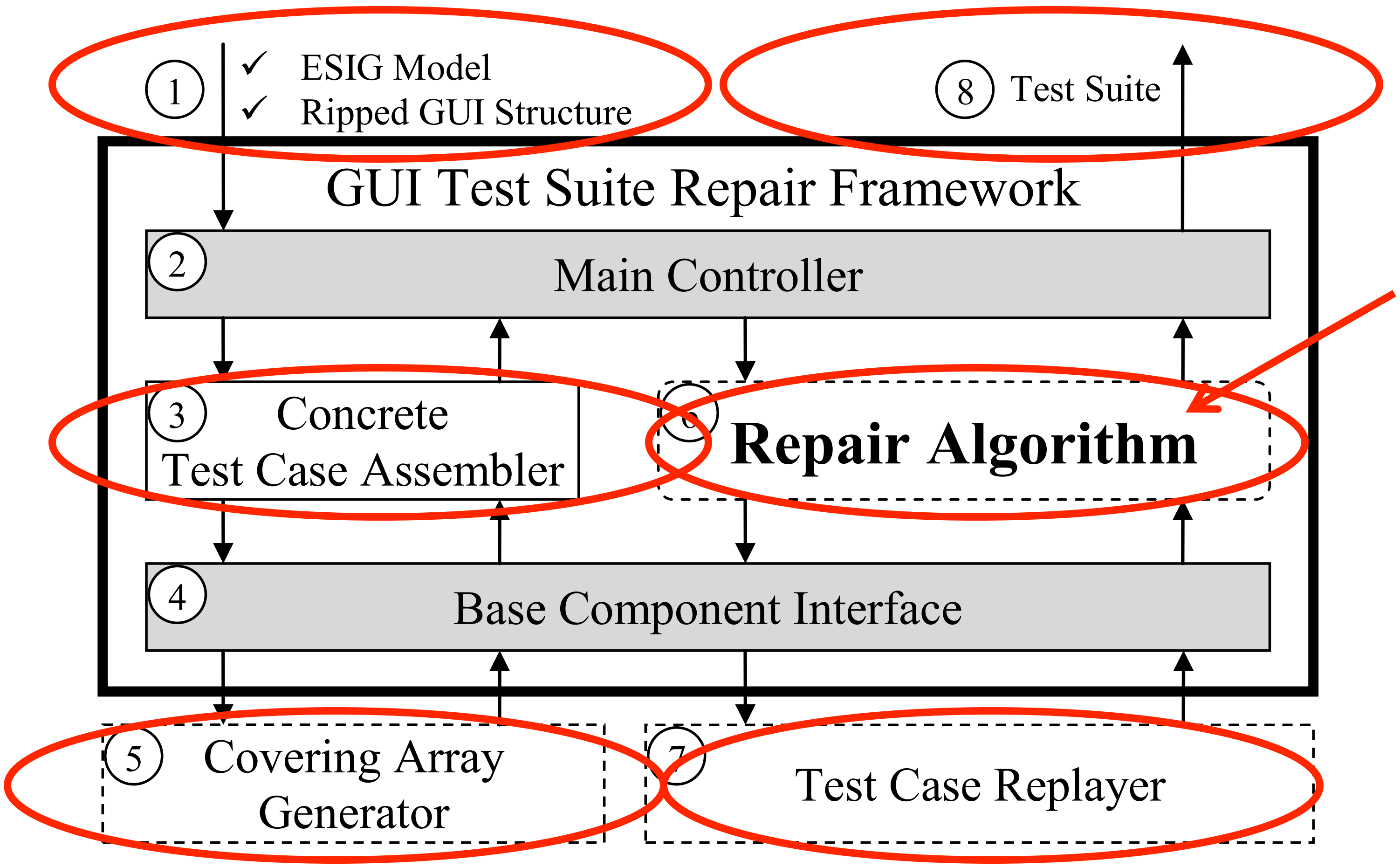
Can This be Avoided?

- Perfect state machine model could provide us with exact feasibility
  - Hard to create
  - But this won't scale which is why we use the stateless EFG/EIG

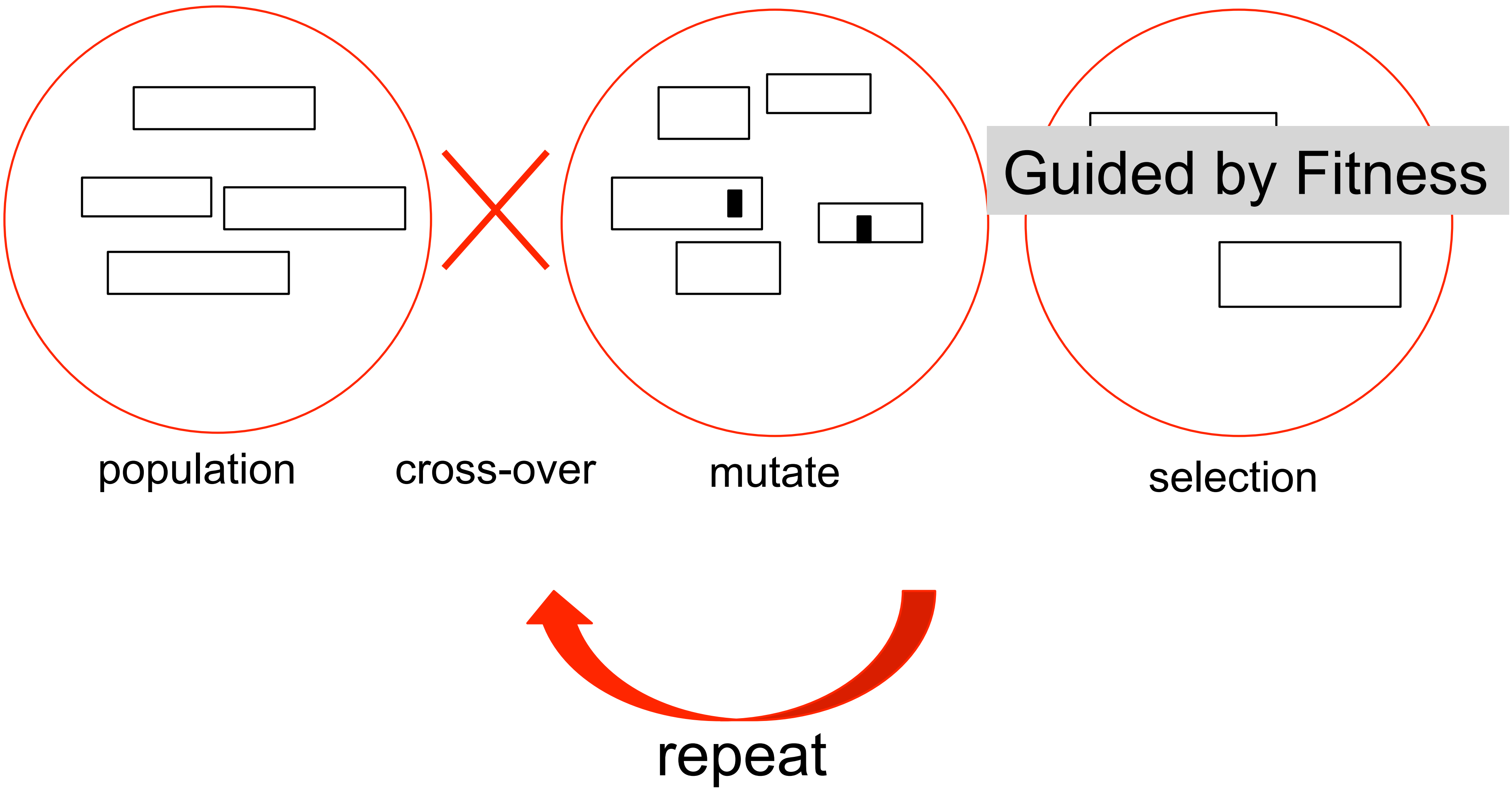
Handling Infeasibility

- Examined the problem of infeasibility in generating longer sequences
- Developed a framework to automatically repair test suites
- Uses a genetic algorithm to iteratively increase coverage while avoiding infeasible sequences

Repair Framework



Genetic Algorithm



Fitness

- Combines coverage and feasibility

$$fitness(s) = b * \mathrm{cov}_s - p(l - f_s)$$

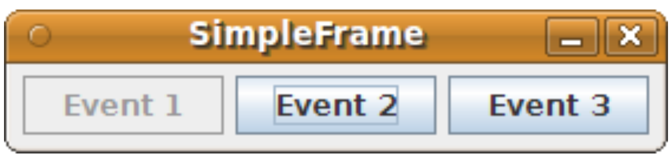
- cov_s= newly covered t–sets in test case
- l = length of test case
- f_s = point of failure
- *p* and *b* are constants
  - 100,000 and 10 in our implementation

Case Study

- Seven synthetic subjects
  - Helps with determinism
  - Can isolate types of infeasible patterns
  - Do not need to worry about faults/other causes of infeasibility
- Programs and artifacts available on COMET

Case Study

- 7 Subjects with 3 to 5 events
  - Disabled, Requires, Event Consecutive (2 and 3-way), Excludes (2 and 3-way), Compound



- Metrics:
  – Coverage of test suites after repair
  – Size of repaired test suites
  – Time to execute

Results(RQ1): Effectiveness

Coverage Strength	% Size Increase	% Final Coverage	% Coverage Increase
2-way Average	-46 to 31 13	100	5.3 to 233 89
3-way Average	-4 to 22 8.6	100 (99.8 cmp) 100	4 to 65 34

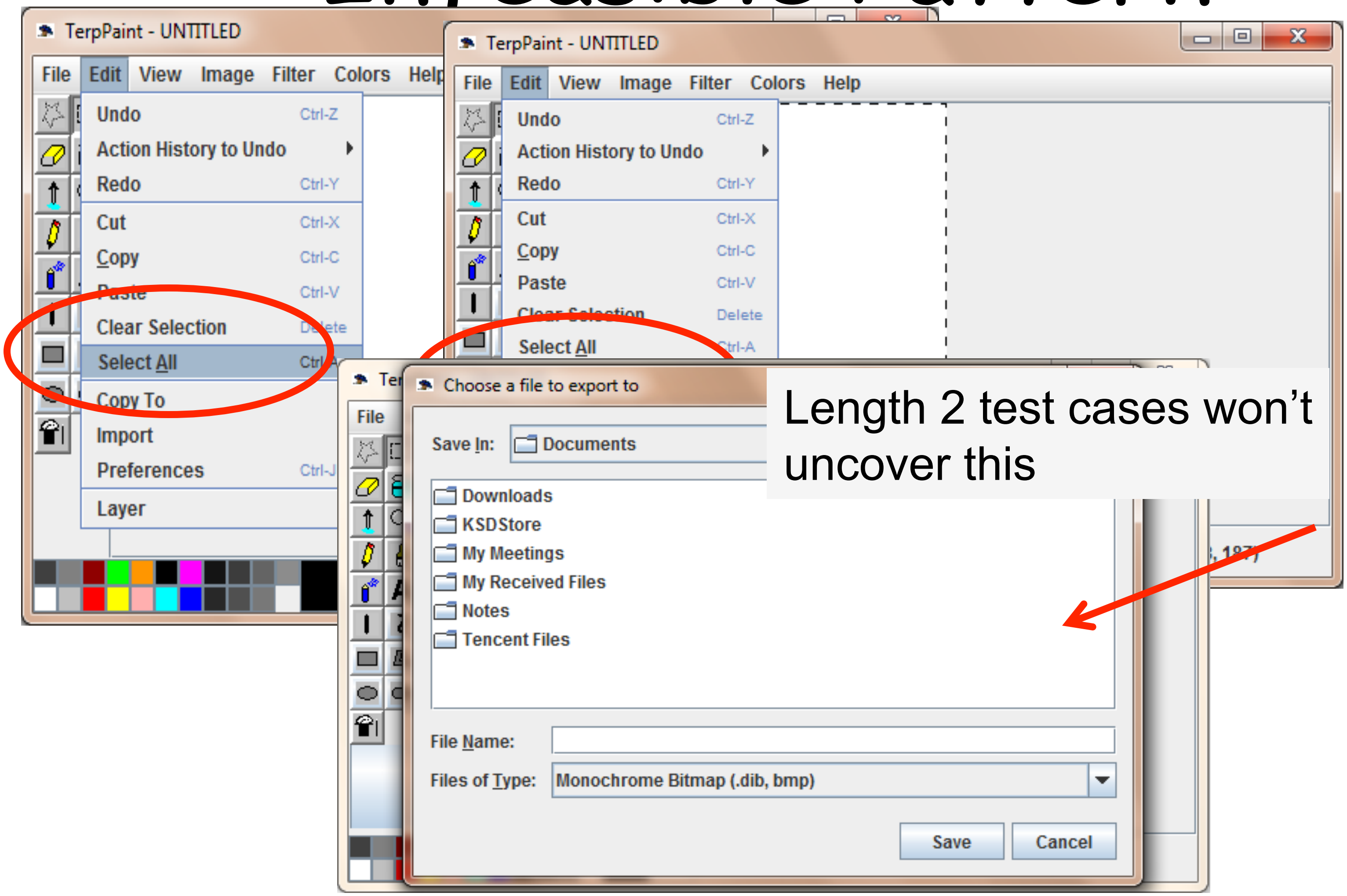
Results(RQ2): vs. Random

Coverage Strength	Avg. Random % Coverage	Avg. GA % Coverage
2-way Length 5 Length10	 95 (80-100) 85 (70-99)	 100 100
3-way Length 5 Length 10	 92 (80-99) 90 (74-99)	 100 100

Applying to Real Programs

- Length 10 test suite for TerpPaint used in fault detection study
- Well understood program
  - long test sequences did not achieve 100% coverage
- Our GA achieved 100 percent coverage except:
  - <*Select All*, *Copy To*>

A 2-way Consecutive Infeasible Pattern



Length 2 test cases won't uncover this

Summary

- We have shown some basic test generation techniques that utilize the EFG/EIGs
- We have presented several types of coverage criteria that have cost/ effectiveness tradeoffs
- We have discussed the need for test suite repair in this scenario

References

1. X. Yuan, M.B. Cohen and A.M. Memon, Covering array sampling of input event sequences for automated GUI testing, *Proceedings of the IEEE International Conference on Automated Software Engineering (ASE)* (short paper), Atlanta, GA, November 2007, pp. 405-408.
2. X. Yuan, M.B. Cohen and A.M. Memon, GUI Interaction Testing: Incorporating Event Context, *IEEE Transactions on Software Engineering* , 37(4), 2011, pp. 559-574.
3. S. Huang, M.B. Cohen and A.M. Memon, Repairing GUI Test Suites Using a Genetic Algorithm, *International Conference on Software Testing, Verification and Validation (ICST)*, April 2010, pp. 245-254.

This material is based upon work supported by the National Science Foundation under Grant No. CNS-0855139 and CNS-0855055. CNS-1205472 and CNS-1205501.
Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

ICSE 2013 Tutorial

Automated Testing of GUI Applications
Models, Tools and Controlling Flakiness

Atif M. Memon and Myra B. Cohen