DESIGN AND EMPIRICAL COMPARISON OF MULTIPLE TEST ORACLES FOR GUIs

Ishan Banerjee

Masters Thesis
08/01/2003

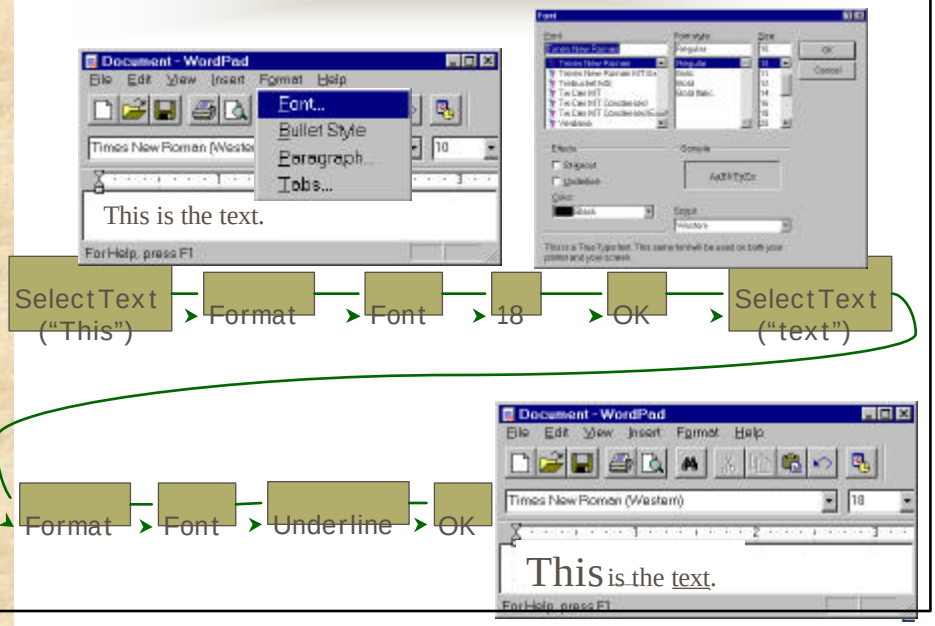
Advisor:
Atif M. Memon

Committee Members:
Adam Porter
Marvin V. Zelkowitz

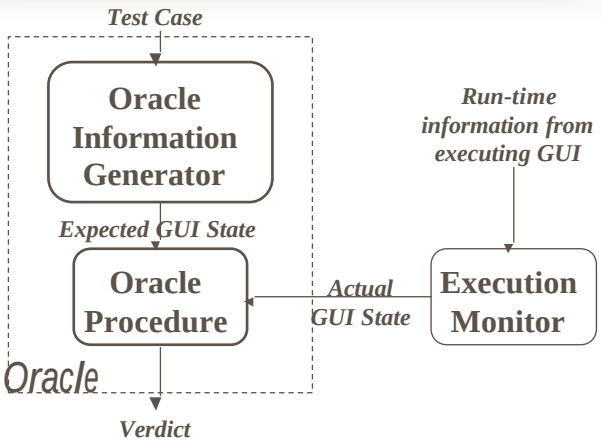
Department of Computer Science
University of Maryland

1

WHAT IS GUI TESTING ?



TEST ORACLE



- Oracle Information Generator - Expected state
- Execution Monitor - Actual state
- Oracle Procedure compares actual and expected state

3

GUI TEST ORACLE

- Components
  - Oracle Information – Expected State
  - Oracle Procedure – Compare
- Oracle Information Generator
  - Manual
  - Model Based – From Specifications
  - Execution Based
    - Screen Scraping
    - Execution Extraction <==
- Oracle Procedure
  - Actual Output <==> Oracle Information
  - May be a set of rules for checking actual output
  - May be an equality check

4

CHALLENGES OF THIS WORK

- Design GUI representation that can be tuned to create multiple test oracles for GUIs
- Design multiple oracle information for GUIs
- Develop compatible oracle procedure for GUIs
- Develop metrics for comparing different types of GUI test oracles

5

OUTLINE

- Related Work
- GUI State
- Oracle Information
- Oracle Procedure
- Combining Oracle Information and Procedure
- Experiments
  - Tool support
  - Fault seeding
- Results

6

RELATED WORK

- TAOS: Testing and Analysis with Oracle Support, D.J. Richardson, ISSTA 1994.
  - Manual
  - Specification based
- TOBAC: A Test Case Browser for Testing Object-Oriented Software, E. Siepman and A.R. Newton, ISSTA 1994.
  - Suggests 7 different oracles for object oriented programs

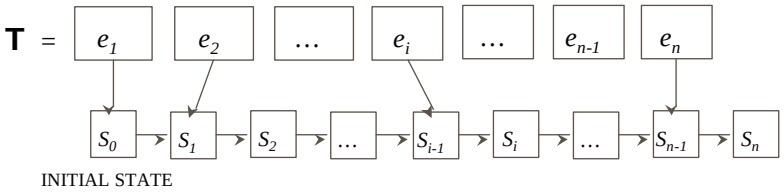
7

OUTLINE

- Related Work
- GUI State
- Oracle Information
- Oracle Procedure
- Combining Oracle Information and Procedure
- Experiments
  - Tool support
  - Fault seeding
- Results

8

TEST CASE AND GUI STATE



T = GUI test case of length n

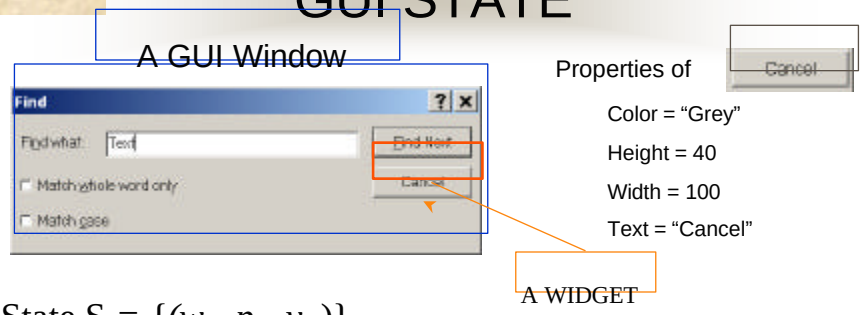
e_i = i^{th} GUI event of test case

S_0 = Initial State of the GUI

- A GUI test case consists of GUI events
- The state of the GUI changes as **T** is executed
- Event e_i is executed on state S_{i-1} to yield state S_i

9

GUI STATE



State S = $\{(w_i, p_j, v_k)\}$

Where

$w \in W$

$p \in P$

$v \in V$

S_{FIND} = { *("Cancel", "Color", "Grey"),*
("Cancel", "Height", 40),
("Cancel", "Width", "100"),
("Cancel", "Text", "Cancel"), …
}

- State of the entire GUI is the union of all window states
- Multiple types of oracle information can be created by selecting subset

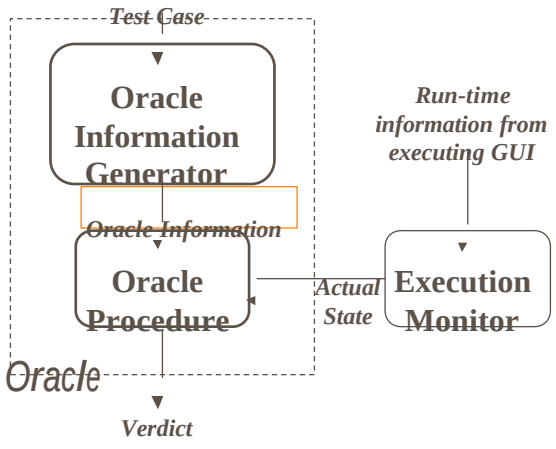
10

OUTLINE

- Related Work
- GUI State
- Oracle Information
- Oracle Procedure
- Combining Oracle Information and Procedure
- Experiments
  - Tool support
  - Fault seeding
- Results

11

GUI TEST ORACLE



- Oracle information for a test case is the expected state of the GUI when the test case is executed

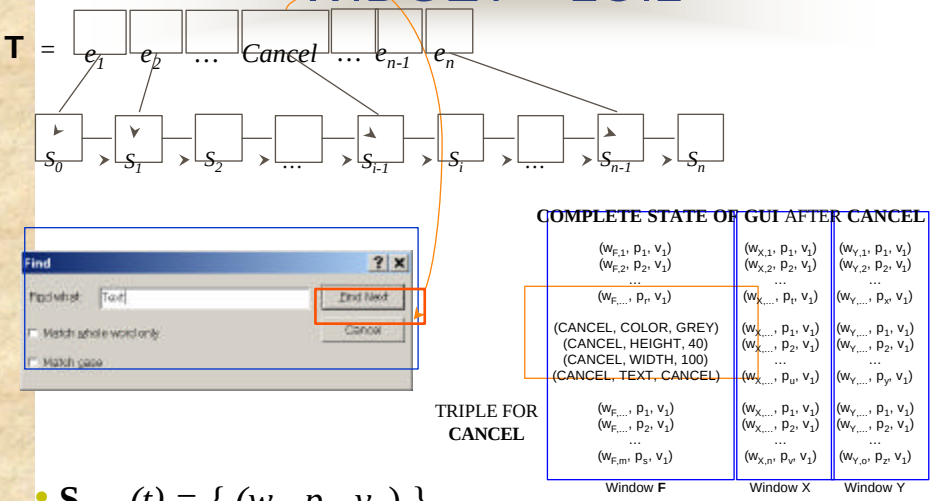
12

ORACLE INFORMATION

- Given a Test Case **T** = < S_0; e_1, e_2, …, e_n >
  - S_0 = *State of GUI before executing* **T**
  - e_i = i^{th} *event of test case*
- Oracle Information **OI** = < S_1 ,… S_n>
  - S_i = *State of GUI after event* e_i *is executed*
  - S_i *may be complete or partial state of GUI*

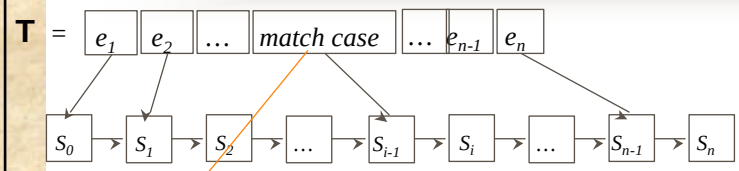
By varying the information in S_i, different *Levels of Oracle Information* (**LOI**) can be created

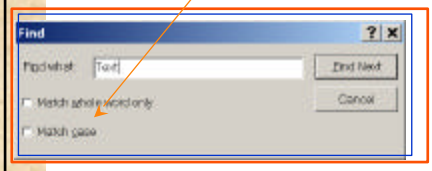
WIDGET – LOI1



- $\mathbf{S_{APP}}(t) = \{ (w_i , p_j , v_k) \}$
- After each event e_i, select triples for the active widget $w_{w,j}$
- $w_{w,j}$ is the widget on which event e_i is executed

ACTIVE WINDOW – LOI2





COMPLETE STATE OF GUI AFTER **MATCH CASE**

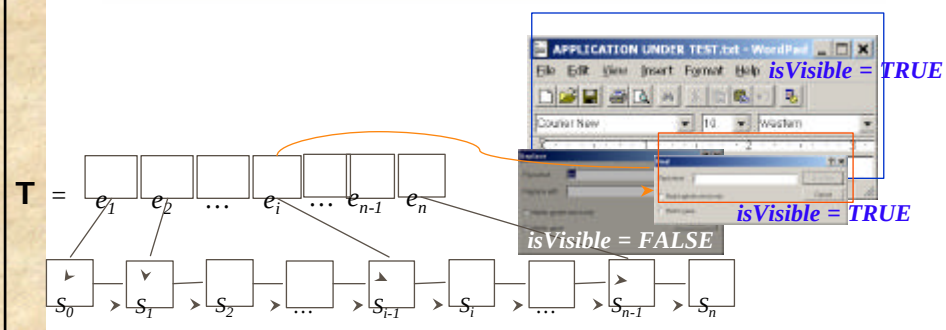
TRIPLE FOR **FIND** WINDOW

Window **F**	Window X	Window Y
$(w_{F,1}, p_1, v_1)$	$(w_{X,1}, p_1, v_1)$	$(w_{Y,1}, p_1, v_1)$
$(w_{F,2}, p_2, v_1)$	$(w_{X,2}, p_2, v_1)$	$(w_{Y,2}, p_2, v_1)$
…	…	…
$(w_{F,}, p_a, v_1)$	$(w_{X,...}, p_t, v_1)$	$(w_{Y,...}, p_x, v_1)$
$(w_{F,..}, p_1, v_1)$	$(w_{X,...}, p_1, v_1)$	$(w_{Y,...}, p_1, v_1)$
$(w_{F,..}, p_2, v_1)$	$(w_{X,...}, p_2, v_1)$	$(w_{Y,...}, p_2, v_1)$
…	…	…
$(w_{F,..}, p_b, v_1)$	$(w_{X,...}, p_u, v_1)$	$(w_{Y,...}, p_y, v_1)$
$(w_{F,..}, p_1, v_1)$	$(w_{X,...}, p_1, v_1)$	$(w_{Y,...}, p_1, v_1)$
$(w_{F,..}, p_2, v_1)$	$(w_{X,...}, p_2, v_1)$	$(w_{Y,...}, p_2, v_1)$
…	…	…
$(w_{F,m}, p_c, v_1)$	$(w_{X,n}, p_v, v_1)$	$(w_{Y,o}, p_z, v_1)$

- $\mathbf{S_{APP}}(t) = \{ (w_i, p_j, v_k) \}$
- After each event e_i, select triples for the active window
- The active window is W, on which event e_i is executed

15

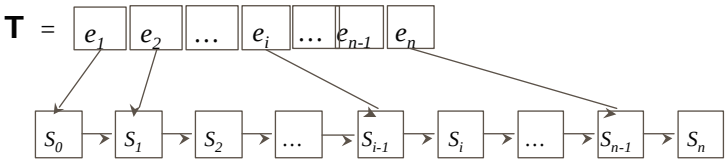
VISIBLE WINDOWS - LOI3



- $\mathbf{S_{APP}}(t) = \{ (w_i, p_j, v_k) \}$
- After each event e_i, select triples for all the visible windows

16

ALL WINDOWS – LOI4



- $\mathbf{S_{APP}}(t) = \{ (w_i, p_j, v_k) \}$
- After each event e_i, select all triples for all the windows
- All triples taken together is the complete state of the GUI

17

OUTLINE

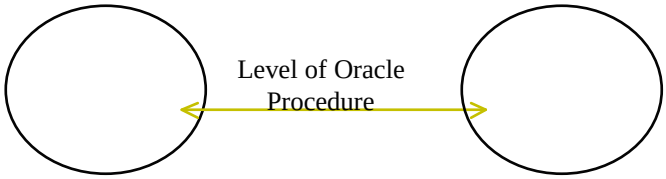
- Related Work
- GUI State
- Oracle Information
- Oracle Procedure
- Combining Oracle Information and Procedure
- Experiments
  - Tool support
  - Fault seeding
- Results

18

ORACLE PROCEDURE

- At specified times, while executing a test case, compares the actual state of the executing GUI with the Oracle Information (**OI**), using some constraints.

AS = *actual state* **OI** = *expected state*

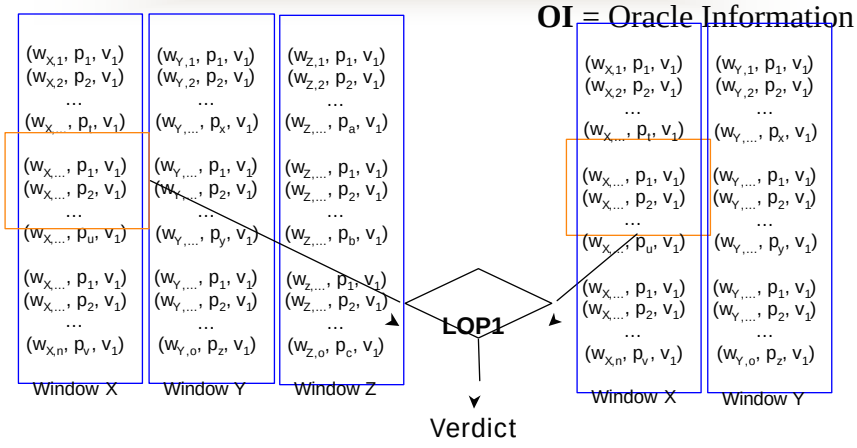


Different *Levels of Oracle Procedure* (**LOP**) are created by constraining

- the **AS** and **OI** used for comparing,
- the frequency of calling the Oracle Procedure

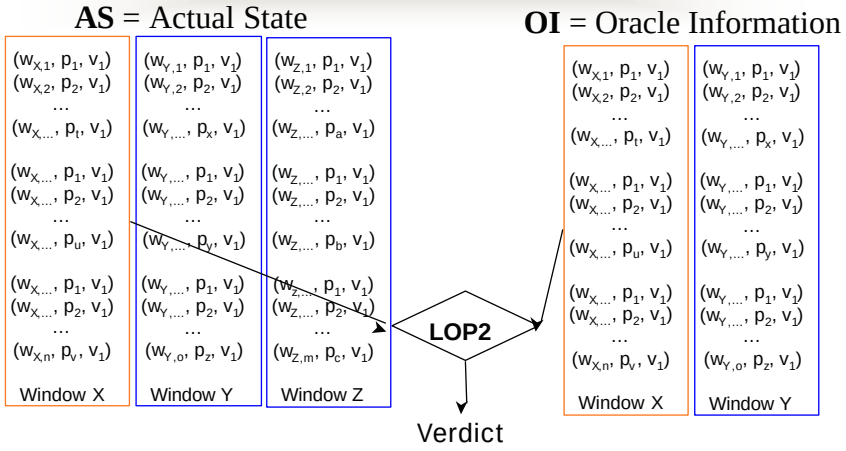
WIDGET – LOP1

OI = Oracle Information



- After <u>each</u> event e_i, select triples from **AS** and **OI** for the <u>active widget</u> $w_{w,j}$
- $w_{w,j}$ is the widget on which event e_i is executed

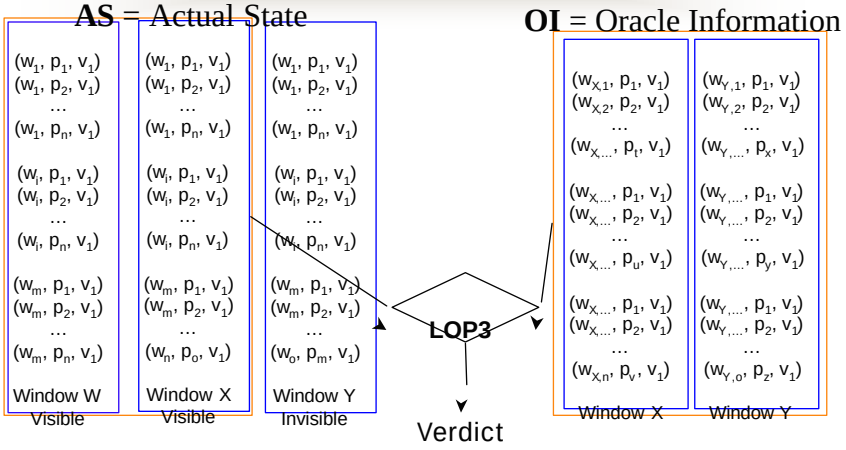
ACTIVE WINDOW – LOP2



- After <u>each</u> event e_i, select triples, from **AS** and **OI**, for the <u>active window</u>
- Active Window is W on which event e_i is executed

21

VISIBLE WINDOWS - LOP3



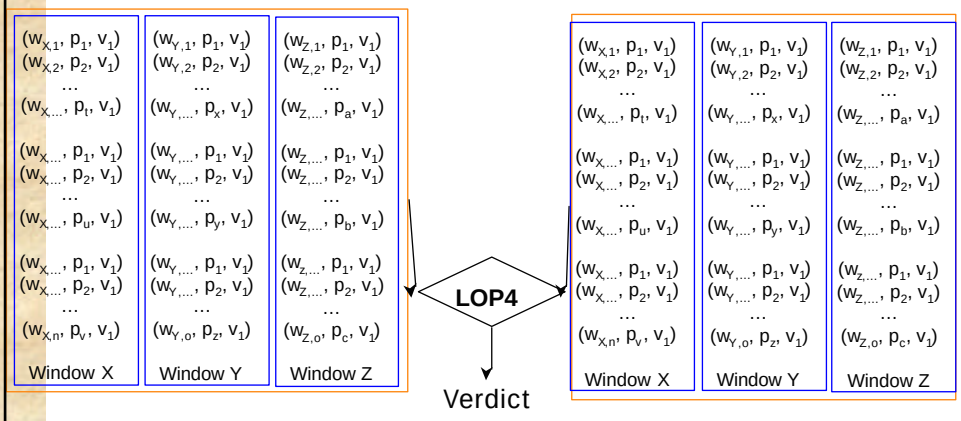
- After <u>each</u> event e_i, select triples, from **AS** and **OI**, for all <u>visible windows</u>

22

ALL WINDOWS – LOP4

AS = Actual State

OI = Oracle Information

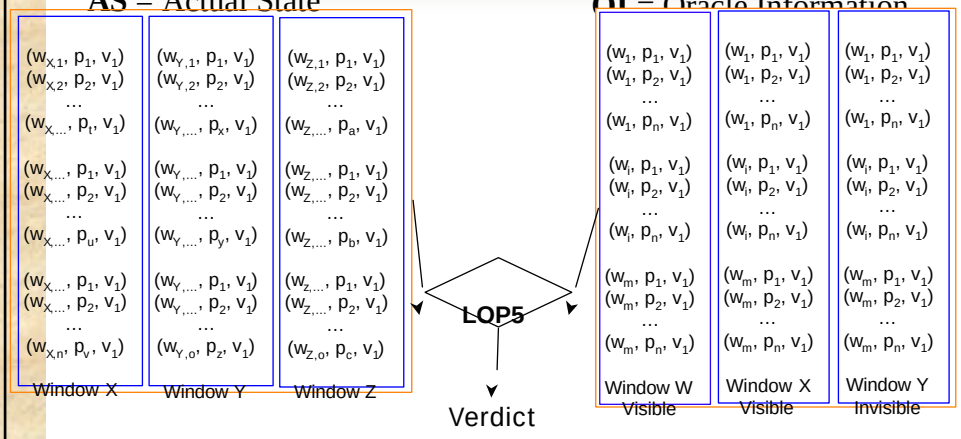


- After <u>each</u> event e_i, select all triples from **AS** and **OI** for all the windows

FINAL STATE – LOP5

AS = Actual State

OI = Oracle Information



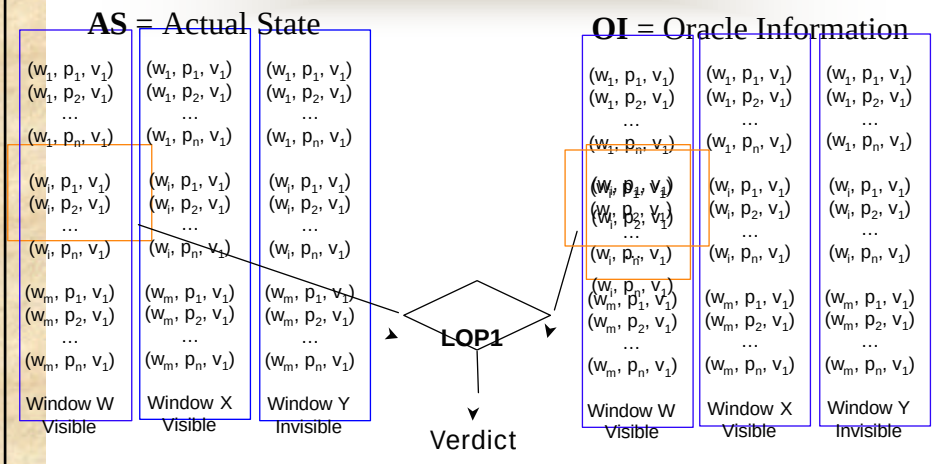
- After <u>last</u> event e_n, select all triples from **AS** and **OI**

OUTLINE

- Related Work
- GUI State
- Oracle Information
- Oracle Procedure
- Combining Oracle Information and Procedure
- Experiments
  - Tool support
  - Fault seeding
- Results

25

COMBINING LOI AND LOP



- LOP1 may be used with LOI1, LOI2, LOI3, LOI4

26

TYPES OF ORACLES

- An oracle procedure may use less than available oracle information (OI)
- The OI can be generated before running the test case
- Depending on resources available at OI collection, different OI is created
- Depending on resources available at test case execution time, the oracle procedure may use a subset of available OI
- The 4 LOI and 5 LOP can be combined into 11 test oracles

27

COMBINING LOI AND LOP

	LOP1	LOP2	LOP3	LOP4	LOP5
LOI1	X				
LOI2	X	X			
LOI3	X	X	X		
LOI4	X	X	X	X	X

- For the diagonal elements, all the Oracle Information is used by the Oracle Procedure
- For the sub-diagonal elements, the Oracle Procedure uses less than available Oracle Information

28

OUTLINE

- Related Work
- GUI State
- Oracle Information
- Oracle Procedure
- Combining Oracle Information and Procedure
- Experiments
  - Tool support
  - Fault seeding
- Results

29

QUESTIONS

- What is the cost incurred in using these oracles ?
- Do different types of oracles detect different number of faults for a given number of test cases ?
- Length of test case versus oracle types.

30

APPROACH

- 4 applications
- Seeded 100 faults to create 100 mutants
- Generated 600 test cases for each application
- Reported results of 4 x 100 x 600 = 240,000 test case runs

31

PROCESS

- Generate test cases
- Run test case on correct application
  - Collect oracle information (LOI1-LOI4)
- Run test case on mutants
  - Compare the actual state with stored oracle information using different levels of oracle procedure (LOP1-LOP5)

32

SUBJECT APPLICATIONS

- Subject Applications
  - TerpOffice – 4 Java applications
  - Developed by students of undergraduate software engineering (CMSC 435)

	LOC	Classes	Windows
TerpWord	1,747	9	8
TerpPresent	4,769	4	5
TerpPaint	9,287	42	8
TerpSpreadSheet	9,964	25	6
TOTAL	25,767	80	27

  - Fairly large programs

33

MUTANTS

- Seeded single fault in original code to create one mutant
  - Single fault avoids fault interaction
  - Easier to identify a fault when a mutant it killed
- Seeded 100 faults for each application
- Total 400 mutants

34

KILL MUTANTS

- A mutant is ‘killed’ when a test case distinguishes it from the original application
- Execute a test case on each mutant
- Compare actual output, **AS**, of mutant with expected output from **OI**
  - Use different types of oracle for comparing
  - Using different oracles, a mutant may or may not be killed for a test case
- Execute 600 test cases for each application

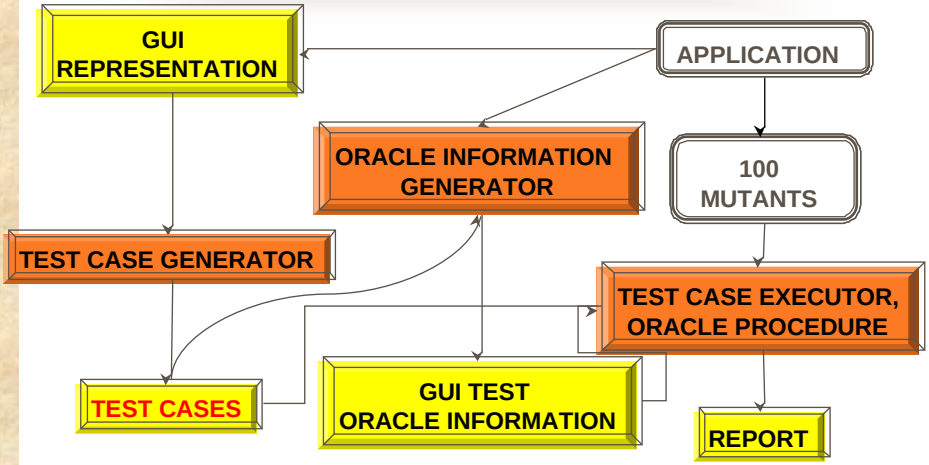
35

TOOL SUPPORT

- Manual process
  - Executing a test case with 20 events takes minute
  - Observing and recording Oracle Information is time consuming
  - Comparing actual state with expected state is time consuming
- Automated the entire process

36

TOOL SUPPORT



TESTING ARTIFACT Intermediate data generated by tool

TESTING TOOL Testing tool from GUITAR

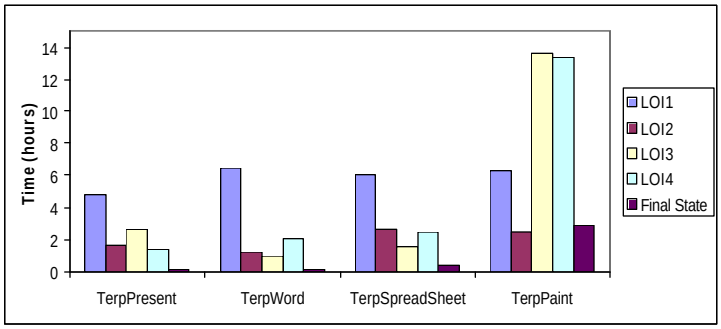
37

OUTLINE

- Related Work
- GUI State
- Oracle Information
- Oracle Procedure
- Combining Oracle Information and Procedure
- Experiments
  - Tool support
  - Fault seeding
- Results

38

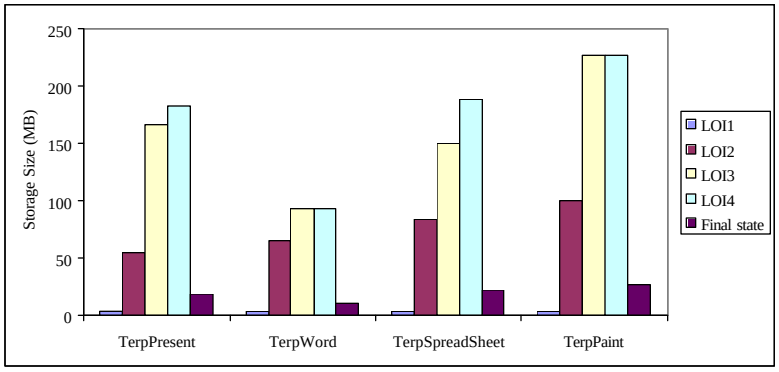
ORACLE INFORMATION GENERATION TIME



Final state required least time because only one state was queried

39

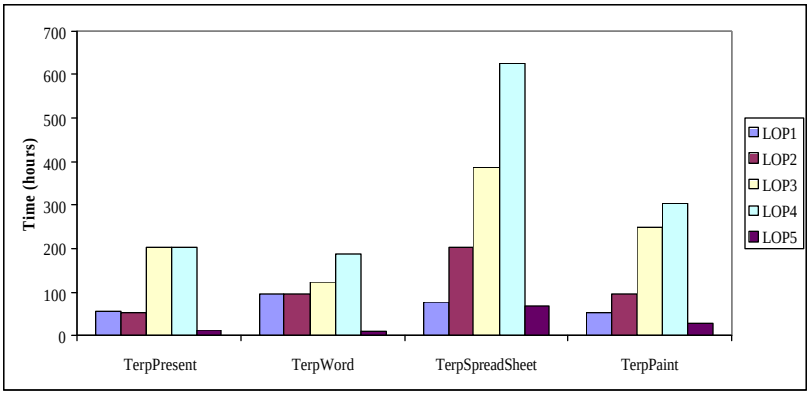
ORACLE STORAGE SPACE



- Space increases from LOI1 to LOI4
- Storing the final state requires less space than LOI2-LOI4

40

COMPARISON TIME



- Comparison time for LOP5 is least for all applications
- Only the final state was obtained

41

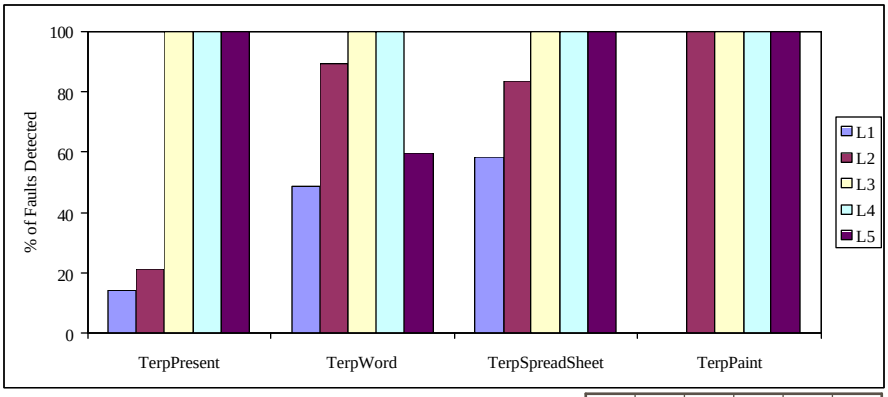
TYPES OF ORACLES

	LOP1	LOP2	LOP3	LOP4	LOP5
LOI1	X L1				
LOI2	X	X L2			
LOI3	X	X	X L3		
LOI4	X	X	X	X L4	X L5

- Diagonal entries labeled as L1…L5
- Faults detected for sub-diagonal entries is same as diagonal entry
- The LOP uses subset of LOI

42

FRACTION OF FAULTS

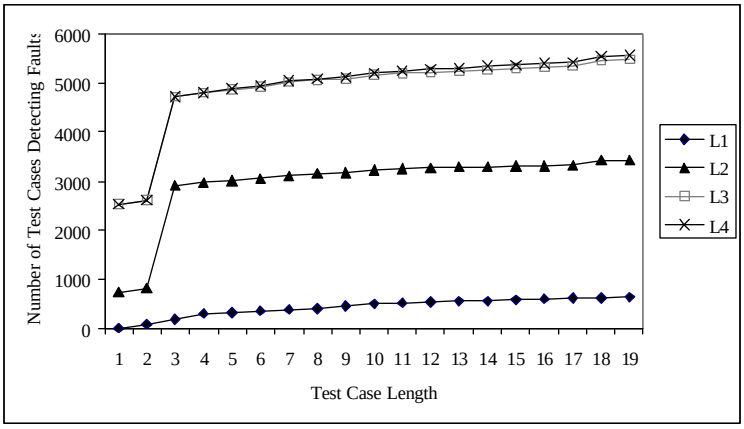


L3, L4, L5 found more faults than L1 and L2

	LOP 1	LOP 2	LOP 3	LOP 4	LOP 5
LOI1	X L1				
LOI2	X	X L2			
LOI3	X	X	X L3		
LOI4	X	X	X	X L4	X L5

43

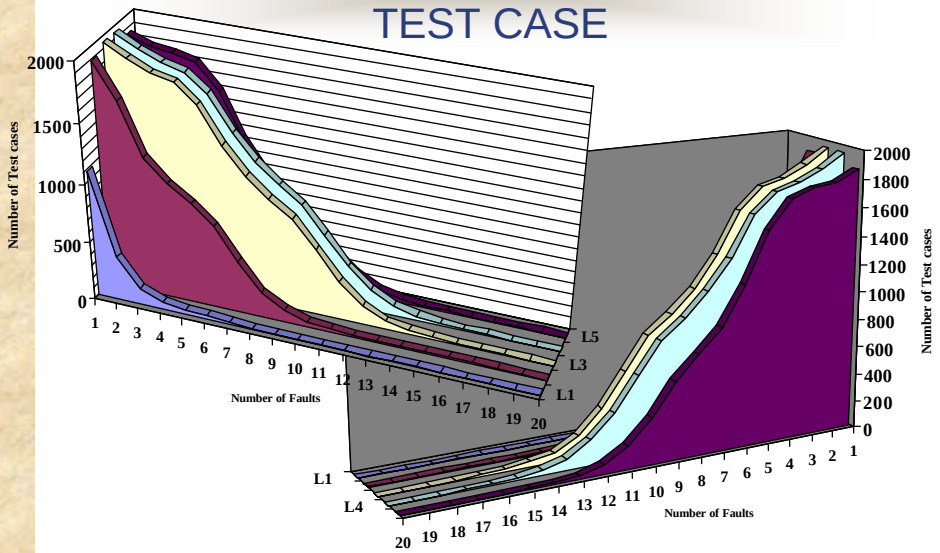
FIND FAULTS EARLY



- Defects found earlier with L3, L4

44

NUMBER OF FAULTS DETECTED BY A TEST CASE



- Larger number of test cases find faults using L3, L4, L5

45

LESSONS LEARNED

- Different oracles have different cost and fault detection ability
- Use of “Final State” is cost effective
  - Less storage and generation time
  - Found almost as many fault as “Visible Window” and “All Windows”
- If short test cases are available
  - Use an “Visible Windows” and “All Windows”
- “Final State” may miss faults that are masked, by the time the last event is executed

46

CONTRIBUTIONS

- Defined GUI state, so that is can be tuned to create multiple test oracles
- Developed GUI test oracles
  - Oracle information
  - Oracle procedure
- Developed tools for experiment
- Designed and conducted an experiment
- Developed guidelines for GUI testing

47

BROADER IMPACTS

- Applied multiple test oracles for effective regression testing (ICSM 2003)
- Developed reverse engineering tool for GUIs (WCRE 2003)
- Compared multiple test oracles for GUI testing (ASE 2003)
- Collaborated with Hughes Network Systems on developing testing tools

48

FUTURE WORK

- Metrics to evaluate and compare test oracles
- Use Windows applications as subjects
- Generate multiple types of test cases and observe the effect of different oracles

49