Test Oracles

- Discussion
  - Automation of oracle necessary
  - Expected behavior given
  - Necessary parts of an oracle
  - Name spaces

Test Oracle

- A test oracle determines whether a system behaves correctly for test execution

- Webster Dictionary - Oracle
  - a person giving wise or authoritative decisions or opinions
  - an authoritative or wise expression or answer

Purpose of Test Oracle

- Sequential Systems
  - Check functionality
- Reactive (event-driven) Systems
  - Check functionality
  - Timing
  - Safety

Reactive Systems

- Complete specification requires use of multiple computational paradigms
- Oracles must judge all behavioral aspects in comparison with all system specifications and requirements
- Hence oracles may be developed directly from formal specifications

Parts of an Oracle

- **Oracle information**
  - **Specifies what constitutes correct behavior**
    - Examples: input/output pairs, embedded assertions
- **Oracle procedure**
  - **Verifies the test execution results with respect to the oracle information**
    - Examples: equality
- **Test monitor**
  - **Captures the execution information from the run-time environment**
    - Examples
      - Simple systems: directly from output
      - Reactive systems: events, timing information, stimuli, and responses

Approach

- **Test class**
  - **Set of test data described by a condition that constrains input data and the initial system state**
- **Every test class will have an explicitly represented test oracle**
- **Results are monitored and verified against the oracle corresponding to all test classes satisfied for the test data**

Phases of the Approach

- **Oracle derivation**
  - **From specifications for each test class**
- **Monitoring test execution**
- **Mapping and applying the oracle procedure to the execution results**
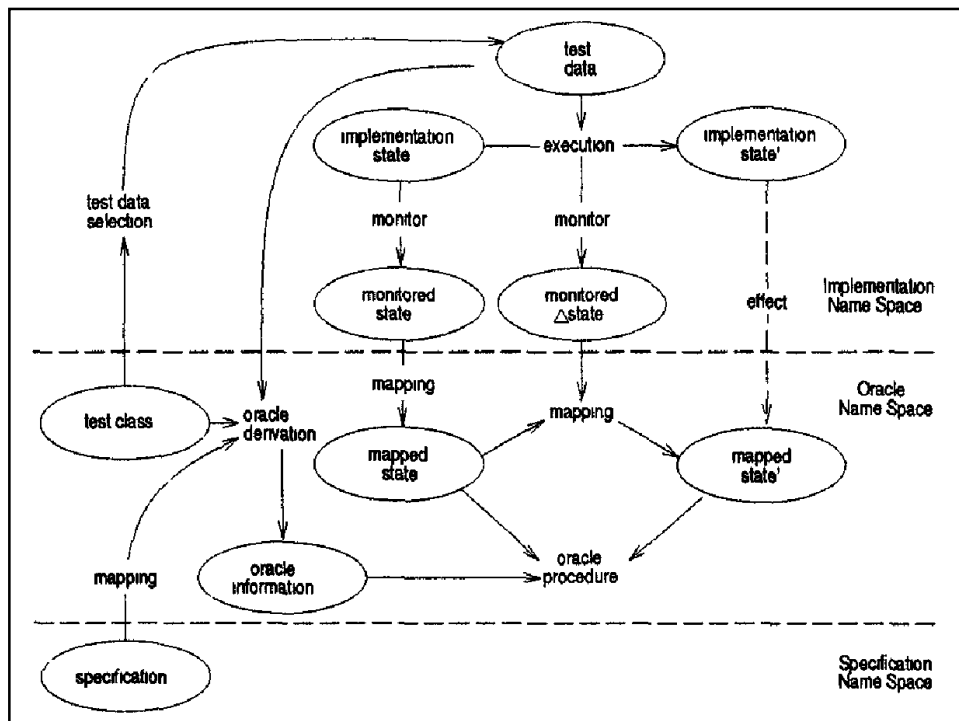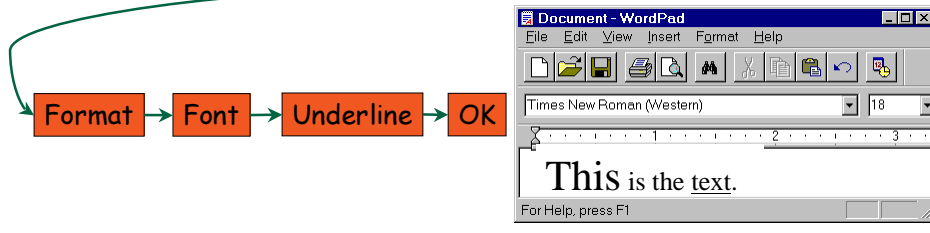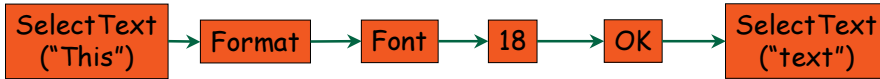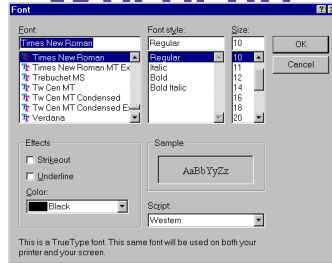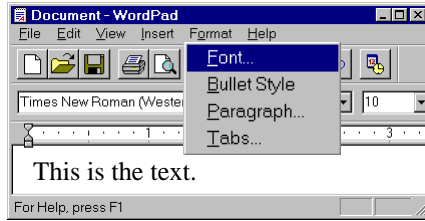
Automated Test Oracles
for GUIs

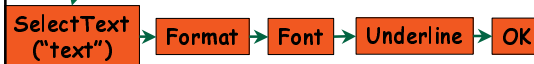## Foundations of Software Engineering, 2000

GUI Test Cases

GUI

- GUI Test Case
  - Sequence of events

A Test Case for WordPad

SelectText ("This") → Format → Font → 18 → OK → SelectText ("text")

Format → Font → Underline → OK

What Is Correct Behavior

SelectText ("This") → Format → Font → 18 → OK

SelectText ("text") → Format → Font → Underline → OK
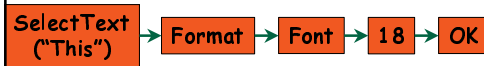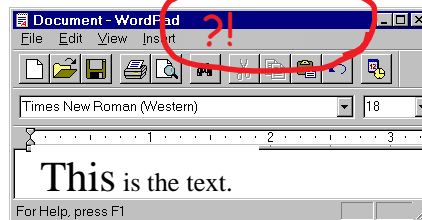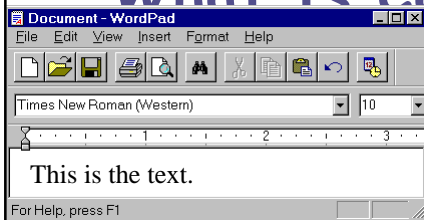
Check State, not only Output !!

Research Focus

- Goal
  - To check the GUI's state after each event
- Approaches
  - Manual
  - Automated

- Challenges
  - Generating expected state
  - Extracting actual state
  - Comparing expected & actual states
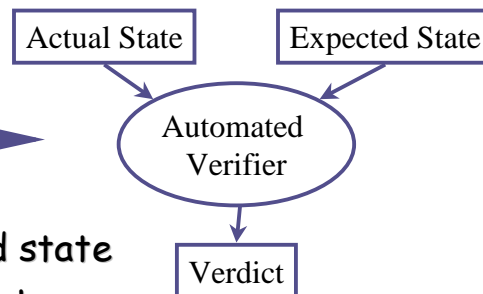
Outline

- Overview of GUI Oracle
- Generating Expected State
  - Modeling the GUI's State
    - Objects
    - Properties
  - Modeling the Events
- Obtaining Actual GUI's State
- Comparing Actual & Expected States

Overview of GUI Oracle

Test Case

Formal GUI Model → **Expected-state Generator**

Run-time information from executing GUI

Expected State

Verifier ← Actual State ← Execution Monitor
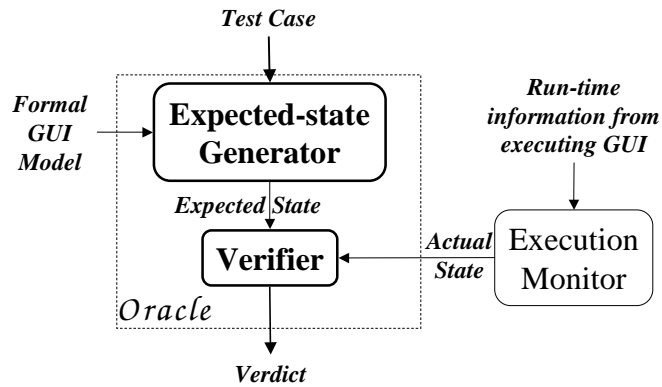
Oracle

Verdict

Modeling the GUI

A GUI consists of Objects
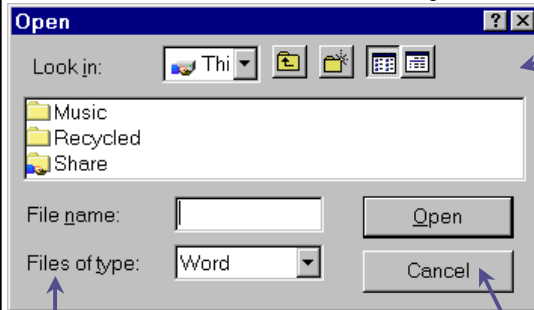
Form

Window State	*wsNormal*
Width	*1088*
AutoScroll	*TRUE*

Button

Caption	*Cancel*
Enabled	*TRUE*
Visible	*TRUE*
Height	*65*

Label

Align	*alNone*
Caption	*Files of type:*
Color	*clBtnFace*
Font	*(tFont)*

All Properties of Cancel
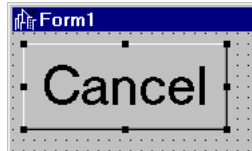
Object Inspector	
Button1: TButton	
Properties	Events
Cancel	true
Caption	Cancel
Cursor	crDefault
Default	false
DragCursor	crDrag
DragMode	dmManual
Enabled	true
+Font	(TFont)
Height	65
HelpContext	0
Hint	
Left	8
ModalResult	mrNone
Name	Button1
ParentFont	false
ParentShowHint	true
PopupMenu	
ShowHint	false
TabOrder	0
TabStop	true
Tag	0
Top	8
Visible	true
Width	153

Determining Properties

- **Manual Examination of GUI**
- **Specifications (Reduced Set)**
  - **GUI being tested**
- **Toolkit/Language (Complete Set)**
  - **All available properties**

Now we know how to represent the GUI's state

Modeling Events

- **Events are State Transducers**
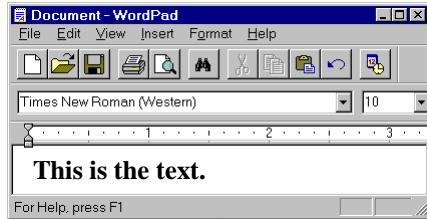
State: S_i

SelectText ("This")

Event: e
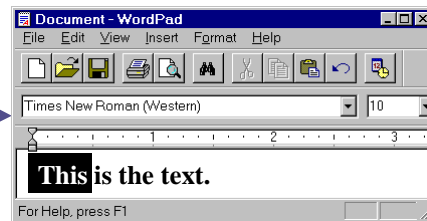
State: S_j

Notation: $S_j = [S_i, e]$

Representing Events

- **We define an event as:**

$$State_j = [State_i, event]$$

- **For example:**

$$State_j = [State_i, cut]$$

- **Need a compact representation**

Operators

Menu1

Menu2

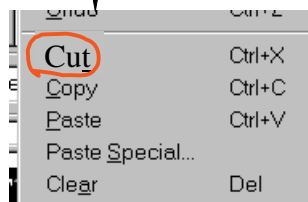|	
Cut	Ctrl+X
Copy	Ctrl+C
Paste	Ctrl+V
Paste Special...	
Clear	Del

Operator :: *CUT*
Preconditions:
       isCurrent(Menu2).

Effects:
       FORALL *Obj* in *Objects*
            Selected(*Obj*) $\Rightarrow$
                ADD *inClipboard*(*Obj*)
                DEL *onScreen*(*Obj*)
                DEL *Selected*(*Obj*)
       ADD *isCurrent*(Menu1)
       DEL *isCurrent*(Menu2).

Obtaining next state

Deriving Expected State

- **Given S_0, the initial state,**
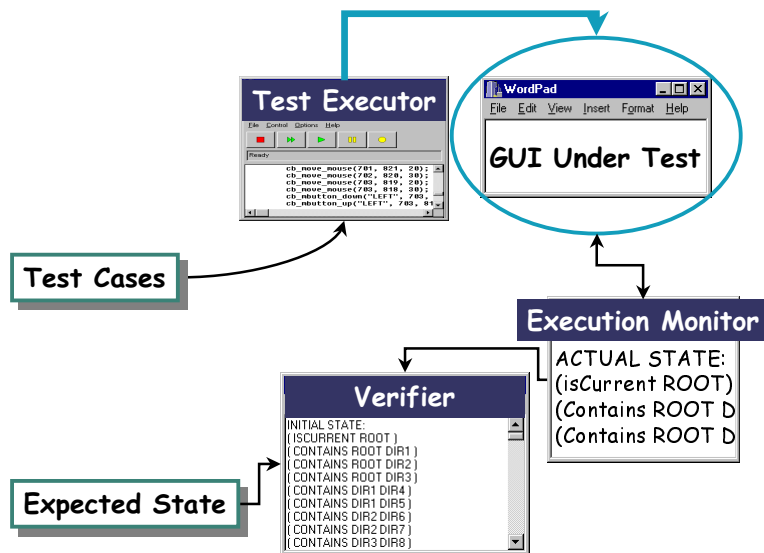- **A sequence of events**

$$e_1 \rightarrow e_2 \rightarrow e_3 \dashrightarrow e_n$$

- **Obtain $S_1 = [S_0, e_1]$**
- **And $S_i = [S_{i-1}, e_i]$**

$$S_0 \xrightarrow{e_1} S_1 \xrightarrow{e_2} S_2 \xrightarrow{e_3} S_3 \dashrightarrow{e_n} S_n$$
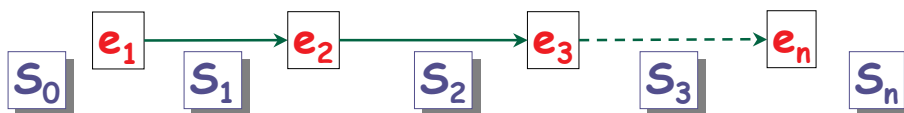
Obtaining Actual GUI's State

- **Execution Monitor**
  - **Screen Scraping**
  - **Queries**
  - **Compatible with Expected State**
  - **Returns <Object, Property, Value>**

    <Button1, "Caption", "Cancel">

Automated Execution

Test Executor

File Control Options Help

Ready

cb_move_mouse(701, 821, 20);
cb_move_mouse(702, 820, 30);
cb_move_mouse(703, 819, 20);
cb_move_mouse(703, 818, 30);
cb_mbutton_down("LEFT", 703,
cb_mbutton_up("LEFT", 703, 81

WordPad

File Edit View Insert Format Help

GUI Under Test

Test Cases

Execution Monitor

ACTUAL STATE:
(isCurrent ROOT)
(Contains ROOT D
(Contains ROOT D

Verifier

INITIAL STATE:
(ISCURRENT ROOT)
(CONTAINS ROOT DIR1)
(CONTAINS ROOT DIR2)
(CONTAINS ROOT DIR3)
(CONTAINS DIR1 DIR4)
(CONTAINS DIR1 DIR5)
(CONTAINS DIR2 DIR6)
(CONTAINS DIR2 DIR7)
(CONTAINS DIR3 DIR8)

Expected State

Comparing
Actual and Expected States

- Verifier
- Three Levels of Testing
  - Changed Property Set *(Operators)*
  - GUI Relevant Property Set *(Specifications)*
  - Complete Property Set *(Toolkit/Language)*
- Hybrid Approach
  - Use all 3