A Comprehensive Framework for Testing Graphical User Interfaces

Atif M. Memon
(with minor edits from original)
Challenges of GUI Testing

- What is the test input to the GUI?
  - (Test case generation)
- How much testing is enough?
  - (Test coverage)
- Is the GUI executing correctly during testing?
  - (Test oracles)
- What test results can be salvaged from the previous test runs to test a new version?
  - (Regression testing)
- How to represent the GUI to handle all the above?
  - (Representation)

Overview of the Framework

- GUI representation
- Test case generation
- Test Oracles
Creating the GUI Model

- **Modeling**
  - GUI’s state in terms of objects & their properties
  - Events as state transducers
  - GUI’s hierarchical structure
    - Classifying events

Modeling the GUI’s State

- A GUI at time T is modeled using:
  - Objects $O = \{o_1, o_2, o_3, ..., o_m\}$
  - Properties $P = \{p_1, p_2, p_3, ..., p_l\}$, where $p_i$ is an $n_i$-ary ($n_i \geq 1$) **Boolean** relation of the form:

  $\text{Property}(o_{i_1}, o_{a_1}, o_{b_1}, ..., o_{x_1}, \text{value})$

  $\text{Caption}(\text{Button, “Cancel”})$

  **GUI’s state:** $S = p_1 \wedge p_2 \wedge p_3 \wedge ... \wedge p_n$
Example: Modeling the GUI’s State

```
Form1
  WState(Form1, wsNormal)
  Width(Form1, 1088)
  Scroll(Form1, TRUE)

Label1
  Align(Label1, alNone)
  Caption(Label1, “Files of type:”)
  Color(Label1, clBtnFace)
  Font(Label1, (tfont))

Button1
  Caption(Button1, Cancel)
  Enabled(Button1, TRUE)
  Visible(Button1, TRUE)
  Height(Button1, 65)
```

All Properties of Button1

```
Caption Cancel
```

```
<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>false</td>
</tr>
<tr>
<td>DragCursor</td>
<td>cICancel</td>
</tr>
<tr>
<td>DragMode</td>
<td>dmManual</td>
</tr>
<tr>
<td>Enabled</td>
<td>true</td>
</tr>
<tr>
<td>Font</td>
<td>(TFont)</td>
</tr>
<tr>
<td>Height</td>
<td>65</td>
</tr>
<tr>
<td>HelpContext</td>
<td>0</td>
</tr>
<tr>
<td>Hint</td>
<td>0</td>
</tr>
<tr>
<td>Left</td>
<td>0</td>
</tr>
<tr>
<td>ModalResult</td>
<td>mNone</td>
</tr>
<tr>
<td>Name</td>
<td>Button1</td>
</tr>
<tr>
<td>ParentFont</td>
<td>false</td>
</tr>
<tr>
<td>ParentShowHint</td>
<td>false</td>
</tr>
<tr>
<td>PopupMenu</td>
<td>true</td>
</tr>
<tr>
<td>TabOrder</td>
<td>0</td>
</tr>
<tr>
<td>Tag</td>
<td>0</td>
</tr>
<tr>
<td>Top</td>
<td>0</td>
</tr>
<tr>
<td>Visible</td>
<td>true</td>
</tr>
<tr>
<td>Width</td>
<td>153</td>
</tr>
</tbody>
</table>
```
Determining Objects & Properties

- Specifications (reduced set)
  - GUI being tested
- Toolkit/language (complete set)
  - All available properties

Now we know how to represent the GUI’s state

GUI Events

- GUI’s state is not static
- Events change the GUI’s state
- Events $E = \{e_1, e_2, e_3, \ldots, e_n\}$, associated with a GUI are functions from one GUI state $S_i$ to another state $S_j$
- Notation: $S_j = e_1(S_i)$
Representing Events

- Infeasible to give exhaustive specifications of the state mapping for each event
- No set limit to the number of objects a GUI can contain at any point in time
- There can be infinitely many states of the GUI
- Model the GUI events using operators, which specify their preconditions and effects
Operators

- An Operator is a 3-tuple
  - \(\langle\text{Name}, \text{Precondition}, \text{Effects}\rangle\)
  - Name identifies an event and its parameters
  - Precondition is a boolean expression
  - Effects is a sequence of ADD and DEL commands on properties
- Operator Op is applicable in any state \(S_i\) in which:
  - The Precondition\( (Op)\) holds
- The resulting state \(S_j\) is determined by using Effects\( (Op)\)
  - ADD commands add their properties, and
  - DEL commands delete properties,
  - in the specified order

Operator Example

Now we know how to represent the events of the GUI
**Model GUI Hierarchically**

- Hierarchy
  - GUIs are designed as a hierarchy of objects
  - Objects are reused from libraries
  - Hierarchical model makes testing efficient
- Classification
  - A new classification of events aids in creating the hierarchical model of the GUI

---

**Classifying Events**

- Opening menus
  - Menu-open events
- Opening modal windows
  - Restricted-focus events
- Opening modeless windows
  - Unrestricted-focus events
- Interacting with underlying software
  - System-interaction events
GUI Modeling Steps

- From the GUI's specifications (formal/informal),
  - Identify objects and properties
  - Create operators for GUI events
  - Using the event classification, create hierarchical and system-interaction operators
    - Details later

Generating GUI Test Cases

- Individual user events are not enough
  - Sequences of user events lead to different states
- Test case: sequence of user events
**Definition: GUI Test Case**

- Event is defined as:
  - \( S_j = e(S_i) \)
- **legal event sequence**
  - \( e_1; e_2; e_3; \ldots; e_n \) is a legal event sequence
  - for state \( S_0 \)
  - iff Exists \( S_0; S_1; S_2; \ldots; S_n \) such that \( S_i = e_i(S_{i-1}), 1 \leq i \leq n \)
- **A GUI test case is a pair**
  - \( (S_0, e_1; e_2; e_3; \ldots; e_n) \)
  - \( S_0 \) is any state, and
  - \( e_1; e_2; e_3; \ldots; e_n \) is a legal event sequence

---

**A Test Case for WordPad**

\( S_0 \)

This is the text.

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

Format — Font — Underline — OK

This is the text.
Selecting Test Sequences

- Infinitely many
- Randomly choose sequences
- Expert chooses sequences
- Automatically generate events for COMMONLY USED TASKS

A Plan for a GUI Task
Overview of Test Generation

<table>
<thead>
<tr>
<th>Phase</th>
<th>Step</th>
<th>Test Designer</th>
<th>Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup</td>
<td>1</td>
<td></td>
<td>Obtain planning operators from GUI model</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Code preconditions and effects of operators in model</td>
<td></td>
</tr>
<tr>
<td>Test Case Generation</td>
<td>3</td>
<td>Specify a task (Initial and Goal States)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4</td>
<td></td>
<td>Generate Test Cases</td>
</tr>
</tbody>
</table>

Using Primitive Operators

- One operator for each event

Operator :: \textit{CUT}

Precondition:
\[ \text{isCurrent}(\text{Menu2}). \]

Effects:
\[ \text{FORALL} \text{ Obj in Objects} \]
\[ \text{Selected}(\text{Obj}) \Rightarrow \]
\[ \text{ADD}\ \text{inClipboard}(\text{Obj}) \]
\[ \text{DEL}\ \text{onScreen}(\text{Obj}) \]
\[ \text{DEL}\ \text{Selected}(\text{Obj}) \]
\[ \text{ADD}\ \text{isCurrent}(\text{Menu1}) \]
\[ \text{DEL}\ \text{isCurrent}(\text{Menu2}). \]
Exploit the GUI’s Structure

- Reduce the number of operators
  - System more efficient
  - Easier for the test designer

Operator Abstractions

Two types of abstractions
- Combine buttons ⇒ create system-interaction operators
- Decompose GUI hierarchically ⇒ create hierarchical operators
Create System-Interaction Operators

Sys-Interaction Operator:
File_SendTo_MailRecipient
= <File + SendTo + MailRecipient>

Create Hierarchical Operators

Using Primitive Operators Only
Main GUI’s Operator Set
... Set Language
SelectFromList()
Default
OK
Cancel
...

Using Abstraction
Main GUI’s Operator Set
... Set Language
...

Language Window’s Operator Set
SelectFromList()
Default
OK
Cancel
**Effects of Exploiting the GUI’s Structure**

- **Reduction in planning operators**
  - 362 operators $\Rightarrow$ 32 operators
  - Ratio 10:1 for MS Wordpad
  - 20:1 for MS Word

- **Efficiency**

<table>
<thead>
<tr>
<th>Task No.</th>
<th>Plan Length</th>
<th>Time (sec.)</th>
<th>Hier Plan Length</th>
<th>Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18</td>
<td>8.93</td>
<td>3</td>
<td>0.11</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>47.62</td>
<td>4</td>
<td>0.18</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
<td>189.87</td>
<td>5</td>
<td>0.14</td>
</tr>
<tr>
<td>4</td>
<td>26</td>
<td>3312.72</td>
<td>6</td>
<td>7.18</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>-</td>
<td>3</td>
<td>0.10</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>-</td>
<td>4</td>
<td>13.01</td>
</tr>
</tbody>
</table>
This is the text.

---

**Test Case**

```
Initial State

Primitive Operator
SelectText ("This")

Hierarchical Operator
FormatFont ("This", 18pt)

Planner
FormatFont 18 OK

Mapping
Format Font

Goal State

Primitive Operator
SelectText ("text")

Hierarchical Operator
FormatFont ("text", Underline)

Planner
FormatFont Underline OK

Mapping
Format Font

SelectText ("text")

Format Font Underline OK
```

For Help, press F1
Different from HTN Planning

Alternative Test Case
Methods to Generate Alternative Test Cases

- Different results from planner
- Hierarchical operator decompositions
- Linearizations of the partial-order plan

Experiments

- **Purpose**
  - To determine whether planning is a feasible approach for GUI test case generation
    - Execution time
    - Human effort

- **Experimental design**
  - **GUI**: our version of MS Wordpad (36 modal windows, 362 events)
  - **Tasks**: 50 tasks (initial & goal states)
  - **Test cases**: generated 290 test cases (6-56 events) using the IPP AI planner
  - **Hardware platform**: 350 MHz Pentium based machine, 256 MB RAM
Test Case Generation

Generating Test Cases

Automated Execution

Executing Test Cases
Test Oracles

GUI Specifications

Test Designer

GUI Model

Oracle

Expected-state Generator

Expected-state sequence

Verifier

Verdict

Execution Generator

GUI State (run-time)

Execution Monitor

Actual State Information

Actual State Information

Orchestrator

Test Case

Test Executor

Regression Tester

Test Coverage Evaluator

Test Oracles: What is the Correct Behavior

Check State, not only Output!!
Determine Correct Behavior

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

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

Overview of GUI Oracle

- Test Case
- GUI Model
- Expected-State Generator
- Verifier
- Oracle
- Expected State
- Actual State
- Verdict
- Execution Monitor
- Run-time information from executing GUI
**Expected State**

- **Obtaining Next State**
  - Given a test case $T$ with $S_0$, the initial state, and
  - A sequence of events
    
    $\begin{array}{c}
    e_1 \\
    e_2 \\
    e_3 \\
    \vdots \\
    e_n
    \end{array}$
  - The next state $S_y$ is obtained from the current state $S_x$ and operator $Op$
    - Delete property $P$ from $S_x$ if $\text{Effects}(Op)$ contains the command “DEL $P$”
    - Add property $P$ in $S_y$ if $\text{Effects}(Op)$ contains the command “ADD $P$”

**Deriving Expected State**

- Obtain $S_1 = e_1(S_0)$
- And $S_i = e_i(S_{i-1})$

- Expected state is a conjunction of properties of the form
  - $\text{Property}(\text{Objects}..., \text{Value})$
Obtaining Actual GUI's State

- **Execution monitor**
  - Compatible with expected state
  - Returns property(objects..., value)
    - e.g., Caption(button1, "cancel")
  - Actual state can be obtained by
    - Screen scraping
    - Queries

Automated Execution

- Test Executor
- GUI Under Test
- Test Cases
- Execution Monitor
- Verifier
- Expected State

ACTUAL STATE:
(isCurrent ROOT)
(Contains ROOT D)
(Contains ROOT D)
Comparing Actual and Expected States

- **Verifier**
- **Three levels of checking**
  - Changed property set *(operators)*
  - GUI relevant property set *(specifications)*
  - Complete property set *(toolkit/language)*
- **Hybrid approach**
  - Use all 3
Experiments

- **Purpose:** determine
  - Time to derive expected state
  - Time to execute monitor and verifier
- **Experimental design**
  - **GUI**: our version of MS Wordpad (36 modal windows, 362 events)
  - **Test cases**: generated 290 test cases (6-56 events) using an AI planner
  - **Hardware platform**: 350 mhz pentium based machine, 256 MB RAM
  - **Properties**: reduced set
  - **Level of checking**: GUI relevant property set

---

Deriving Expected State

![Graph showing the relationship between test case length and time](image)

*Total CPU time (all 290 test cases and expected states)*

75.84 sec.
GUI Relevant-properties checking
Total running time < 10 minutes

Putting it All Together

GUI Specifications
Test Designer
GUI Model
Oracle
Expected-state Generator
Expected-state sequence
Verifier
Verdict

Executing GUI
GUI State (run-time)
Task
Test-Case Generator
Actual State Information
Test Case
Test Executor

GUI Implementation:
Tools (Languages/Toolkits)

Regression Testing Algorithms
Coverage Evaluation Algorithms
Coverage Criteria
Coverage Report