WHAT IS GUI TESTING?

This is the text.

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

Format  Font  Underline  OK

This is the text.
TEST ORACLE

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

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
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

OUTLINE

• Related Work
• GUI State
• Oracle Information
• Oracle Procedure
• Combining Oracle Information and Procedure
• Experiments
  • Tool support
  • Fault seeding
• Results
RELATED WORK

  - Manual
  - Specification based
  - Suggests 7 different oracles for object oriented programs

OUTLINE

- Related Work
- GUI State
- Oracle Information
- Oracle Procedure
- Combining Oracle Information and Procedure
- Experiments
  - Tool support
  - Fault seeding
- Results
**TEST CASE AND GUI STATE**

\[ T = e_1 e_2 \ldots e_i \ldots e_{n-1} e_n \]

\[ S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_{i-1} \rightarrow S_i \rightarrow \ldots \rightarrow S_{n-1} \rightarrow S_n \]

**INITIAL STATE**

\[ T = \text{GUI test case of length } n \]

\[ e_i = \text{ith GUI event of test case} \]

\[ S_0 = \text{Initial State of the GUI} \]

- 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 \( S_i \)

**GUI STATE**

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

Where

\[ S_{\text{FIND}} = \{(\text{"Cancel"}, \text{"Color"}, \text{"Grey"}), \]

\[ (\text{"Cancel"}, \text{"Height"}, 40), \]

\[ (\text{"Cancel"}, \text{"Width"}, "100"), \]

\[ (\text{"Cancel"}, \text{"Text"}, \text{"Cancel"}), \ldots \} \]

- State of the entire GUI is the union of all window states
- Multiple types of oracle information can be created by selecting subset
- Oracle information for a test case is the expected state of the GUI when the test case is executed
**ORACLE INFORMATION**

- Given a Test Case \( T = < S_0, e_1, e_2, ..., e_n > \)
  - \( S_0 = \text{State of GUI before executing } T \)
  - \( e_i = i^{th} \text{ event of test case} \)
- **Oracle Information** \( OI = < S_1, ..., S_n > \)
  - \( S_i = \text{State of GUI after event } e_i \text{ is executed} \)
  - \( S_i \) 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**

- \( S_{\text{App}}(t) = \{ (w_i, p_j, v_k) \} \)
- 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**

\[ T = e_1, e_2, \ldots, \text{match case} \ldots e_{n-1}, e_n \]

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

**VISIBLE WINDOWS - LOI3**

\[ T = e_1, e_2, \ldots, e_{n-1}, e_n \]

- \( S_{\text{APP}}(t) = \{ \langle w_i, p_j, v_k \rangle \} \)
- After each event \( e_i \), select triples for all the visible windows
\textbf{ALL WINDOWS – LOI4}

\[ T = \{ e_1, e_2, \ldots, e_i, \ldots, e_n \} \]

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

\textbf{OUTLINE}

- Related Work
- GUI State
- Oracle Information
- \textbf{Oracle Procedure}
- Combining Oracle Information and Procedure
- Experiments
  - Tool support
  - Fault seeding
- Results
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.

\[ \text{AS} = \text{actual state} \quad \text{OI} = \text{expected state} \]

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

- After each event \( e_i \), select triples from AS and OI for the active widget \( w_{w,j} \)
- \( w_{w,j} \) is the widget on which event \( e_i \) is executed
ACTIVE WINDOW – LOP2

**AS** = Actual State

<table>
<thead>
<tr>
<th>Window X</th>
<th>Window Y</th>
<th>Window Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>(w₁, p₁, v₁)</td>
<td>(w₂, p₂, v₂)</td>
</tr>
<tr>
<td>(wₐ, pₐ, vₐ)</td>
<td>(wₐ₋₁, pₐ₋₁, vₐ₋₁)</td>
<td>...</td>
</tr>
</tbody>
</table>

**OI** = Oracle Information

<table>
<thead>
<tr>
<th>Window X</th>
<th>Window Y</th>
<th>Window Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>(w₁, p₁, v₁)</td>
<td>(w₂, p₂, v₂)</td>
</tr>
<tr>
<td>(wₐ, pₐ, vₐ)</td>
<td>(wₐ₋₁, pₐ₋₁, vₐ₋₁)</td>
<td>...</td>
</tr>
</tbody>
</table>

- After each event \( e_j \), select triples, from **AS** and **OI**, for the active window
- Active Window is \( W \) on which event \( e_j \) is executed

---

VISIBLE WINDOWS – LOP3

**AS** = Actual State

<table>
<thead>
<tr>
<th>Window W</th>
<th>Window X</th>
<th>Window Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible</td>
<td>(w₁, p₁, v₁)</td>
<td>Visible</td>
</tr>
</tbody>
</table>
| ... | ... | ...

**OI** = Oracle Information

<table>
<thead>
<tr>
<th>Window X</th>
<th>Window Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>(w₁, p₁, v₁)</td>
</tr>
<tr>
<td>(wₐ, pₐ, vₐ)</td>
<td>...</td>
</tr>
</tbody>
</table>

- After each event \( e_j \), select triples, from **AS** and **OI**, for all visible windows
• After each event $e_i$, select all triples from $AS$ and $OI$ for all the windows

• After last 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

COMBINING LOI AND LOP

\[ AS = \text{Actual State} \]
\[ OI = \text{Oracle Information} \]

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

COMBINING LOI AND LOP

<table>
<thead>
<tr>
<th></th>
<th>LOP1</th>
<th>LOP2</th>
<th>LOP3</th>
<th>LOP4</th>
<th>LOP5</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOI1</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOI2</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOI3</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOI4</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

- 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
OUTLINE

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

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.
**APPROACH**

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

**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)
SUBJECT APPLICATIONS

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

<table>
<thead>
<tr>
<th></th>
<th>LOC</th>
<th>Classes</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>TerpWord</td>
<td>1,747</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>TerpPresent</td>
<td>4,769</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>TerpPaint</td>
<td>9,287</td>
<td>42</td>
<td>8</td>
</tr>
<tr>
<td>TerpSpreadSheet</td>
<td>9,964</td>
<td>25</td>
<td>6</td>
</tr>
<tr>
<td>TOTAL</td>
<td>25,767</td>
<td>80</td>
<td>27</td>
</tr>
</tbody>
</table>

• Fairly large programs

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
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

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
TOOL SUPPORT

- GUI REPRESENTATION
- APPLICATION
- ORACLE INFORMATION GENERATOR
- 100 MUTANTS
- TEST CASE EXECUTOR, ORACLE PROCEDURE
- REPORT
- TEST CASE GENERATOR
- TEST CASES
- GUI TEST ORACLE INFORMATION

Intermediate data generated by tool

Testing tool from GUITAR

OUTLINE

- Related Work
- GUI State
- Oracle Information
- Oracle Procedure
- Combining Oracle Information and Procedure
- Experiments
  - Tool support
  - Fault seeding
- Results
Final state required least time because only one state was queried.

- Space increases from LOI1 to LOI4
- Storing the final state requires less space than LOI2-LOI4
• Comparison time for LOP5 is least for all applications
• Only the final state was obtained

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

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

FIND FAULTS EARLY

• Defects found earlier with L3, L4
NUMBER OF FAULTS DETECTED BY A TEST CASE

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

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
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

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
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