Plan Generation for GUI Testing

- The 21st International Conference on Software Engineering
- The Fifth International Conference on Artificial Intelligence Planning and Scheduling
- IEEE Transactions on Software Engineering

Research Focus

Interactions between the GUI and the Underlying Code
**Why Planning for GUI Testing**

- GUIs are Event Driven
- Individual User Events
  - NOT ENOUGH!
  - Sequences of User Events lead to Different States
- Test Case: Sequence of User Events
- How to Generate Test Cases?
- Use Planning to Select Likely Test Cases

**Selecting Test Sequences**

- Infinitely Many
- Randomly Choose Sequences
- Expert Chooses Sequences
- Automatically Generate Events for COMMONLY USED TASKS
A Plan for a GUI Task

Outline

- Using Planning for Test Case Generation
  - Overall Approach
  - Exploiting GUI Structure
  - Generating Alternative Test Cases
- Experimental Results
- Related Research
- Concluding Remarks
Overview of Test Generation

<table>
<thead>
<tr>
<th>Phase</th>
<th>Step</th>
<th>Test Designer</th>
<th>Automatic Planning-based System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup</td>
<td>1</td>
<td>Derive Planning Operators from GUI</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Code Preconditions and Effects of Operators</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>Generate Test Cases</td>
<td></td>
</tr>
</tbody>
</table>

Straightforward Approach

- Define One Operator for each User Action

Operator :: CUT
Preconditions:
\[ \text{isCurrent(Menu2)}. \]

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

- Reduce the Number of Operators
  - System more Efficient
  - Easier for the Test Designer

Opening Modal Windows
Opening Menus

Interacting with the Underlying Software
Create Hierarchical Operators

Two Types of Abstractions
- Combine Buttons ⇒ Create System-Interaction Operators
- Decompose GUI Hierarchically ⇒ Create Abstract Operators

Create System-Interaction Operators

Sys-Interaction Operator:
File_SendTo_MailRecipient
= \langle \text{File} + \text{SendTo} + \text{MailRecipient} \rangle
Create Abstract Operators

Straightforward Approach
Main GUI's Operator Set

Using Abstraction
Language Window's Operator Set

Define Abstraction
SetLanguage() Abstract Operator

High Level Plan
... → SetLanguage() → ...

Sub Plan
SelectFromList("English(US)") → OK
Effects of Exploiting the GUI's Structure

- Reduction in Planning Operators
  - 325 operators $\Rightarrow$ 32 operators
  - Ratio 10:1 for MS WordPad
  - 20:1 for MS Word
- System Automatically Determines the System-interaction and Abstract Operators

Initial State

Goal State

This is the text.
Test Case

**INITIAL**

- **Primitive Operator**: `SelectText ("This")`
- **Abstract Operator**: `FormatFont ("This", 18pt)`

**GOAL**

- **Primitive Operator**: `SelectText ("text")`
- **Abstract Operator**: `FormatFont ("text", Underline)`

**Planner**

- **FormatFont**: 18pt → OK
- **FormatFont**: Underline → OK
Different from HTN Planning

No Interactions
Methods to Generate Alternative Test Cases

- Different Results from Planner
- Abstract Operator Decompositions
- Linearizations of the Partial-order Plan
Feasibility Study

• Purpose
  - To Determine whether Planning is a Feasible Approach for GUI Test Case Generation
  • Execution Time
  • Human Effort

• Experimental Design
  - GUI: MS WordPad
  - Planner: IPP [Koehler et al. ‘97]
  - Hardware Platform: 300 MHz Pentium based Machine, 200 MB RAM, Linux OS
  - 8 Tasks, Multiple Test Cases for each Task

Experimental Results

<table>
<thead>
<tr>
<th>(Task) Plan No.</th>
<th>Plan Time (sec.)</th>
<th>Sub Plan Time (sec.)</th>
<th>Total Time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.16</td>
<td>0</td>
<td>3.16</td>
</tr>
<tr>
<td>2</td>
<td>3.17</td>
<td>0</td>
<td>3.17</td>
</tr>
<tr>
<td>3</td>
<td>3.2</td>
<td>0.01</td>
<td>3.21</td>
</tr>
<tr>
<td>4</td>
<td>3.38</td>
<td>0.01</td>
<td>3.39</td>
</tr>
<tr>
<td>5</td>
<td>3.44</td>
<td>0.02</td>
<td>3.46</td>
</tr>
<tr>
<td>6</td>
<td>4.09</td>
<td>0.04</td>
<td>4.13</td>
</tr>
<tr>
<td>7</td>
<td>8.88</td>
<td>0.02</td>
<td>8.9</td>
</tr>
<tr>
<td>8</td>
<td>40.47</td>
<td>0.04</td>
<td>40.51</td>
</tr>
</tbody>
</table>
Related Work

- **GUI Testing**
  - *Genetic Algorithm* Technique [Kasik and George]
  - *Visual TDE for GUIs* [Foster, Goradia, Ostrand, and Szermer]

- **Planning for Testing**
  - [Adele Howe, Anneliese Von Mayrhauser, Richard Mraz in ASE ’97]

Concluding Remarks

- **Automatic Planning** is a Feasible Approach for GUI Test Case Generation

- **Automatic Generation of Preconditions and Effects from GUI Specifications**

- **Generate Expected Output** *(Automated Verification)*
Coverage Criteria for GUI Testing


Coverage Criteria

• Two purposes
  - Test data selection criteria
    • Rules used to select test cases
  - Test data adequacy criteria
    • Rules used to determine how much testing has been done

• Common Examples for Conventional Software
  - Statement coverage
  - Branch coverage
  - Path coverage

Structural Representation of the Code
Coverage Criteria for GUIs

- Cannot use code-based coverage
  - Source code not always available
  - Event-based input
    - Different level of abstraction
- Our Contribution
  - Hierarchical structure of the GUI in terms of events
  - Coverage criteria based on events

Outline

- GUI Definition
- Representation of GUIs
- Coverage Criteria
- Case Study
- Conclusions
GUI Definition

- Hierarchical
- Graphical Front-end
- Accepts User-generated and System-generated events
- Fixed sets of events
- Deterministic Output
- State of the GUI is the set of Objects and their Properties

GUI Representation

- Motivation
  - GUI testing needs a "Unit of Testing"
    - Manageable
    - Test the unit comprehensively
    - Test interactions among units
  - GUIs are created using library elements
    - Need to test these elements before packaging them for reuse
      - Certain level of confidence that the element has been adequately tested
    - User of these elements should be able to test the element in its context of use
Model GUI Hierarchically

- Hierarchy
  - GUIs are decomposed into a hierarchy of components
  - Hierarchical decomposition makes testing intuitive and efficient
  - Several hierarchical views of GUIs
  - We examine Modal Dialogs to create the hierarchical model

Modal Windows in GUIs
Modal Windows in GUIs

Invokes

Main

Print

Properties

Components

Modal Windows in GUIs
**Integration Tree**

**Definition:** Integration tree is a triple \( \langle N, R, B \rangle \)
- \( N \) is the set of components in the GUI
- \( R \in N \) is a designated component called the **Main** component
- \( B \) is the set of directed edges showing the invokes relation between components, i.e., \((C_x, C_y) \in B \) iff \( C_x \) invokes \( C_y \).

**Representing a Component**

**Definition:** Event \( e_x \) follows \( e_y \) iff \( e_x \) can be performed immediately after \( e_y \).
**Definition:** Event-flow graph is a 4-tuple \( V, E, B, I \)
- \( V \) is the set of vertices, representing events,
- \( E \) is the set of directed edges, showing the follows relationship,
- \( B \) is the set of events first available (shown in red),
- \( I \) is the set of events that invoke other components (dotted lines).

**Classifying Events**

**Classification**

-A new classification of events aids in creating the hierarchical model of the GUI

- Opening modal windows
  - Restricted-focus events
- Closing modal windows
  - Termination events
- Opening modeless windows
  - Unrestricted-focus events
- Opening menus
  - Menu-open events
- Interacting with underlying software
  - System-interaction events
Coverage Criteria

- Intuitively
  - Each component is a unit of testing
  - Test events within each component
    - Intra-component coverage criteria
  - Test events across components
    - Inter-component coverage criteria

Coverage Criteria

- Intra-component Coverage
  - Event coverage
    - Individual events
    - Each node in the event-flow graph
  - Event-interaction coverage
    - Each pair of events
    - Each edge in the event-flow graph
  - Length-n event sequence coverage
    - Sequences of events
    - Bounded by length
      - Length-1 event sequences
      - Length-2, length-6 event sequences
    - Paths in the event-flow graph
Coverage Criteria

- Inter-component Coverage
  - Invocation coverage
    - Invoke each component
    - Each restricted-focus event
  - Invocation-termination coverage
    - Invoke each component and terminate it
    - Restricted-focus event followed by a termination event
  - Inter-component length-n coverage
    - Longer sequences from one component to another
    - Bounded by length

Case Study

- Purpose
  - To determine:
    - How many test cases do we need to test WordPad
    - Correlation between event and code-based coverage

- Experimental design
  - GUI: our version of MS WordPad (36 modal windows, 362 events)
  - Hardware platform: 350 MHz Pentium based machine, 256 MB RAM
### Test Cases for WordPad

<table>
<thead>
<tr>
<th>Component Name</th>
<th>Event-sequence Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1' 2' 1 2 3 4 5 6</td>
</tr>
<tr>
<td>Main</td>
<td>56 791 14354 255720 4490626 78385288</td>
</tr>
<tr>
<td>FileOpen</td>
<td>10 80 640 5120 40960 327680</td>
</tr>
<tr>
<td>FileSave</td>
<td>10 80 640 5120 40960 327680</td>
</tr>
<tr>
<td>Print</td>
<td>12 108 972 8748 78732 708588</td>
</tr>
<tr>
<td>Properties</td>
<td>13 143 1573 17303 190333 2093663</td>
</tr>
<tr>
<td>PageSetup</td>
<td>11 88 704 5632 45056 360448</td>
</tr>
<tr>
<td>FormatFont</td>
<td>9 63 441 3087 21609 151263</td>
</tr>
<tr>
<td>Print+Properties</td>
<td>1 2 13 260 3913 52520 663013</td>
</tr>
<tr>
<td>Main+FileOpen</td>
<td>1 2 10 100 1180 17160 278760</td>
</tr>
<tr>
<td>Main+FileSave</td>
<td>1 2 10 100 1180 17160 278760</td>
</tr>
<tr>
<td>Main+PageSetup</td>
<td>1 2 11 110 1298 18876 306636</td>
</tr>
<tr>
<td>Main+FormatFont</td>
<td>1 2 9 81 909 13311 220509</td>
</tr>
<tr>
<td>Main+Print+Properties</td>
<td>12 145 1930 28987 466578</td>
</tr>
</tbody>
</table>

**Results**

**Correlation between Event-based & Code-based Coverage**

- **Code Instrumentation**
- **Generated all event sequences up to length 3. Total test cases: 21,659**
- **Executed all 21,659 cases and obtained execution traces**
- **Statement coverage**
Correlation between Event-based & Code-based Coverage

Results