Using Execution Feedback in Test Case Generation

CMSC 737 – Presentation
Bao Nguyen
baonn@cs.umd.edu

Two Strategies
- **Static plan**
  - Category Partition
  - Data flow analysis (path, branch, def-use, etc)
  - Predicate based: BOR – BRO
  - Try to guess ahead!!!
- **Dynamic plan**
  - Execution information as feedback
  - Generating test cases on the fly

What’s next…
- Test case generation based on execution feedback
- Case study: two recent papers in ICSE’07
- What I’m looking at
- Conclusion

Why Static Plans Are Not Enough?
- Software is so complex:
  - Empirical products => Difficult to formalize
  - Subjective solutions => Difficult to use
  - Human actuators => Difficult to manage
  - Intelligent products => Difficult to predict
  - Dynamic environments => Difficult to address

  Hard to predict ahead !!!
  Hard to exhaustedly test !!!

General Framework

(Adapted from "Software Cybernetics: Progress and Challenges" - Aditya P. Mathur)
What’s next…

- Execution feedback based test case generation
- Case study: two recent papers in ICSE’07
  - GUI Testing: Yuan and Memon. “Using GUI Run-Time State as Feedback to Generate Test Cases”
- What I’m looking at
- Conclusion

Motivations of the paper

- Previous work
  - 1-way: Crash Test
  - 2-way: Smoke Test
- Longer test cases detected additional faults
- Unable to run multi-way test coverage
  - 2-way run for months [TSE’05]
  - Try to prune edge
Key idea
- Use GUI states as feedback to identify “important” edges
  - Called Event Semantic Interaction Edges
  - Generate new longer test cases covering those edges

Event Semantic Interaction
- **Heuristic**: Two events executed together results differently than executed in isolation → semantic interaction

Three contexts for events wrt windows
- **Context 1**: events in modeless window
  - $ea(S)$: the GUI state after executing
    - $e2; \text{TERM}$: $x=1,2$
  - $ea(eS)$: the GUI state after executing sequence $e1; e2; \text{TERM}$
- **Context 2**: events in same modal window
  - $ea(S)$: the GUI state after executing $e1; \text{TERM}$
  - $ea(eS)$: the GUI state after executing sequence $e1; e2; \text{TERM}$
- **Context 3**: events in parent and child modal window
  - $ea(S)$: the GUI state after executing $e1; \text{TERM}$
  - $ea(eS)$: the GUI state after executing sequence $e1; e2; \text{TERM}$

Six predicates for modeless windows
- **Predicate 1**:
  \[ 3w \in W, p \in P, r \in R, v \in V, e \in E : (w, p, v) \in ((w, p, v) \cup (w, p, v')) \quad \text{where} \quad v' \notin S_0 \quad \text{and} \quad v' \neq (w, p, v) \]
- **Predicate 2**:
  \[ 3w \in W, p \in P, r \in R, v \in V, e \in E : (w, p, v) \in ((w, p, v) \cup (w, p, v')) \quad \text{where} \quad v' \in S_0 \quad \text{and} \quad v' \neq (w, p, v) \]
- **Predicate 3**:
  \[ 3w \in W, p \in P, r \in R, v \in V, e \in E : (w, p, v) \in ((w, p, v) \cup (w, p, v')) \quad \text{where} \quad v' \notin S_0 \quad \text{and} \quad v' \neq (w, p, v) \]
- **Predicate 4**:
  \[ 3w \in W, p \in P, r \in R, v \in V, e \in E : (w, p, v) \in ((w, p, v) \cup (w, p, v')) \quad \text{where} \quad v' \in S_0 \quad \text{and} \quad v' \neq (w, p, v) \]
- **Predicate 5**:
  \[ 3w \in W, p \in P, r \in R, v \in V, e \in E : (w, p, v) \in ((w, p, v) \cup (w, p, v')) \quad \text{where} \quad v' \notin S_0 \quad \text{and} \quad v' \neq (w, p, v) \]
- **Predicate 6**:
  \[ 3w \in W, p \in P, r \in R, v \in V, e \in E : (w, p, v) \in ((w, p, v) \cup (w, p, v')) \quad \text{where} \quad v' \in S_0 \quad \text{and} \quad v' \neq (w, p, v) \]

(More details refer to “A comprehensive framework for testing graphical user interfaces”
Atif M. Memon, Ph.D. dissertation, 2001)

Experiments
- **Subject applications**: three OSS
  - CrosswordSage 0.3.5
  - FreeMind 0.9.9b2
  - JMSN 0.9.9b2
- **Test oracle**
  - Program crashes
Result - Test case reduction

<table>
<thead>
<tr>
<th>2-way</th>
<th>3-way</th>
<th>4-way</th>
<th>5-way</th>
</tr>
</thead>
<tbody>
<tr>
<td>99.78%</td>
<td>99.97%</td>
<td>99.99%</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Percentages of test case reduction

Result - Faults detection

<table>
<thead>
<tr>
<th>All-edges Covering Test Cases</th>
<th>3,4,5-way Covering Test Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>CrosswordSage</td>
<td>FreeMind</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Lessons learned

- Event handlers are implemented in multiple classes
- A large input space is needed
- Crash because the permutations of events
  
  => Need longer test cases???

Conclusion of this paper

- Contributions
  - A new GUI model for test-case generation
  - A new relationships among GUI events (i.e. context)
  - A utilization of GUI state as feedback
  - A fully automatic end-to-end GUI testing process
  - A demonstration
- Future work
  - Simplify 6 predicates and 3 contexts
  - Identify and classify events dominating ESI
  - Minimize number of test cases
  - Apply feedback technique to objects outside GUI

What’s next…

- Test case generation based on execution feedback
- Case study: two recent papers in ICSE’07
- What I’m looking at
- Conclusion

What I’m looking at

- Push test case generation and test case execution closer
- Generate new test cases during the execution
- Utilize the feedback immediately
A case study

- **Adaptive test oracles**: the QoS idea

![Diagram of a test case and oracle information]

**Figure 1. Test Case vs. Oracle Information**

---

**Conclusion**

- Software is dynamic so we need a dynamic approach
- Using feedback in software testing is feasible
- Somewhat related to control theories (i.e. software cybernetic)
- Drawback: Like hill climbing
  => local optimization
  - Can mutants (like in GA) overcome this?
  - Systematically vs. Randomly

---

**Questions**

- What does “**Event Semantic Interaction**” in section 4 mean?
- What are the threats to validity and what are the weaknesses in Xun’s experiments?