Object-oriented Software

Class: The fundamental unit of abstraction
- Instance variables
- Methods

Component

Objects: Instances of class
- State
- Behavior

Messages

Jeff Offutt
Information & Software Engineering
George Mason University
Fairfax, VA USA
www.ise.gmu.edu/~ofut/
ofut@ise.gmu.edu

Len Gallagher
Tony Cincotta
Julie Zanon
National Institute of Standards and Technology

Supported by NIST and NSF.
Modeling Classes as FSMs

Single Class Example – Engine

- States : \{ S_0, S_f, ON, OFF \}
- Variables : \{ int speed, boolean KeyOn \}
- Methods : \{ Engine(), \sim Engine(), setKeyOn (boolean in), Start (int S), Stop(), setSpeed (int S), int getSpeed() \}
- Transitions : \((source \rightarrow target, trigger, guard, \{actions\})\)
  - t1 : \((S_0 \rightarrow OFF, Engine(), true, \{speed=0, KeyOn=false\})\)
  - t2 : \((OFF \rightarrow ON, Start(), KeyOn=true \& 0\leq S \leq 110, \{speed=S\})\)
  - t3 : \((ON \rightarrow OFF, Stop(), true, \{speed=0\})\)
  - t4 : \((OFF \rightarrow OFF, getSpeed(), true, \{return speed\})\)

Finite state machine for class Engine
Testing OO Software

1) **Intra-method testing**: Testing individual methods within classes
2) **Inter-method testing**: Pairs of methods within a class are tested in concert
3) **Intra-class testing**: Testing a single class, usually using sequences of calls to methods within the class
4) **Inter-class testing**: More than one class is tested at the same time (integration)

This research is focused on inter-class testing.

Inter-class FSM-based Testing

- Model each class as a finite class state machine (CSM)
- Create combined class state machine from all CSMs
  - Identify mutator methods and their parameters
  - Identify accessor methods
  - Inter-class method calls are messages
  - Add messages as edges to the CSM to create the CSM

- CCSM is stored in a relational database
**Component Flow Graph**

- A *component flow graph* (CFG) represents control and data flows among classes and state variables
- Create a CFG
  - Nodes are:
    - States in CCSM that are sources or targets of messages
    - Transitions in the CCSM
    - Guards on transitions
  - Edges represent control flow and data definition-use pairs
    - Intra-class
    - Inter-class
- CFG represents synchronous and asynchronous interactions among classes

---

**Generating Test Requirements**

- **All-uses** on the CFG: Test every definition to every use in the CFG
- **Candidate Test Paths**: A path from a definition to a use in the CFG
  - must be def-clear: state variable is not changed directly or indirectly (remember asynchronous interactions)
- Candidate test paths are our *test requirements*
- **Executable Test Path** is a CTP with parameter values
  - feasible: an input exists that will cause it to be executed
- **Executable test cases** are user-level inputs that satisfy executable test paths
Classes, Objects, State Variables, Methods

<table>
<thead>
<tr>
<th>ClassAlias</th>
<th>ClassId</th>
<th>StateId</th>
<th>StateName</th>
<th>DefnPredicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throttle</td>
<td>c10</td>
<td>s00</td>
<td>Initial</td>
<td>Undefined</td>
</tr>
<tr>
<td>Throttle</td>
<td>c10</td>
<td>s01</td>
<td>Idle</td>
<td>Position=fconst</td>
</tr>
<tr>
<td>Throttle</td>
<td>c10</td>
<td>s02</td>
<td>Manual</td>
<td>fconst&lt;Position=gconst &amp; Position&gt;Floor</td>
</tr>
<tr>
<td>Throttle</td>
<td>c10</td>
<td>s03</td>
<td>Automatic</td>
<td>fconst&lt;Position=gconst &amp; Position=Floor</td>
</tr>
<tr>
<td>Throttle</td>
<td>c10</td>
<td>s04</td>
<td>Danger</td>
<td>GasPedal=gconst</td>
</tr>
</tbody>
</table>

Define State Predicates

<table>
<thead>
<tr>
<th>ClassAlias</th>
<th>StateId</th>
<th>StateName</th>
<th>DefnPredicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gauges</td>
<td>s00</td>
<td>Initial</td>
<td>Undefined</td>
</tr>
<tr>
<td>Gauges</td>
<td>s01</td>
<td>Normal</td>
<td>Speed&lt;180 &amp; OilLight=Off &amp; WaterTemp&lt;100</td>
</tr>
<tr>
<td>Gauges</td>
<td>s02</td>
<td>Danger</td>
<td>Speed&gt;=180 OR OilLight=On OR WaterTemp&gt;=100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ClassAlias</th>
<th>ClassId</th>
<th>StateId</th>
<th>StateName</th>
<th>DefnPredicate</th>
</tr>
</thead>
<tbody>
<tr>
<td>CruiseUnit</td>
<td>c05</td>
<td>s00</td>
<td>Initial</td>
<td>Undefined</td>
</tr>
<tr>
<td>CruiseUnit</td>
<td>c05</td>
<td>s01</td>
<td>Off</td>
<td>UserSwitch=Off</td>
</tr>
<tr>
<td>CruiseUnit</td>
<td>c05</td>
<td>s02</td>
<td>Inactive</td>
<td>UserSwitch=On &amp; Gauges.Cruise()=Off &amp; TargetSpeed=0 &amp; UserMode=NULL</td>
</tr>
<tr>
<td>CruiseUnit</td>
<td>c05</td>
<td>s03</td>
<td>Cruise</td>
<td>UserSwitch=On &amp; UserMode=N &amp; Gauges.Cruise()=On &amp; SlowCutoff=TargetSpeed=FastCutoff</td>
</tr>
<tr>
<td>CruiseUnit</td>
<td>c05</td>
<td>s04</td>
<td>Accelerate</td>
<td>UserSwitch=On &amp; UserMode=RA &amp; Gauges.Cruise()=On</td>
</tr>
<tr>
<td>CruiseUnit</td>
<td>c05</td>
<td>s05</td>
<td>Decel</td>
<td>UserSwitch=On &amp; UserMode=SD &amp; Gauges.Cruise()=On</td>
</tr>
<tr>
<td>CruiseUnit</td>
<td>c05</td>
<td>s06</td>
<td>Override</td>
<td>UserSwitch=On &amp; Gauges.Cruise()=Off &amp; SlowCutoff=TargetSpeed=FastCutoff</td>
</tr>
</tbody>
</table>
Finite Class State Machine Model of Each Object
(sourceState, targetState, method, guard, action)

Determine Guards and Actions for Each Transition

<table>
<thead>
<tr>
<th>Class</th>
<th>State</th>
<th>Transition</th>
<th>TriggerState</th>
<th>Function</th>
<th>Guard</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>c05</td>
<td>t01</td>
<td>Inactive</td>
<td>Cruise</td>
<td>UserMode()</td>
<td>x=NT &amp; UserMode = RA &amp; (SlowCutoff &lt; Gauges.Speed() &lt; FastCutoff) &amp; AutoSystem.BrakeActive() = false &amp; AutoSystem.ClutchActive() = false</td>
<td>Call Throttle.Floor(TargetThrottle); Call Gauges.Cruise(On); UserMode := NT; Pause; CurrentSpeed := Gauges.Speed(); Put CheckState() on Call Queue;</td>
</tr>
<tr>
<td>c05</td>
<td>t02</td>
<td>Cruise</td>
<td>CheckState()</td>
<td></td>
<td>X = NT &amp; UserMode = RA &amp; (SlowCutoff &lt; Gauges.Speed() &lt; FastCutoff) &amp; AutoSystem.BrakeActive() = false &amp; AutoSystem.ClutchActive() = false</td>
<td>Call Throttle.Floor(TargetThrottle); Call Gauges.Cruise(On); UserMode := NT; Pause; CurrentSpeed := Gauges.Speed(); Put CheckState() on Call Queue;</td>
</tr>
<tr>
<td>c05</td>
<td>t03</td>
<td>Override</td>
<td>Override</td>
<td></td>
<td>x&lt;&gt;NT OR Gauges.Speed() &lt;= SlowCutoff OR Gauges.Speed() &gt;= FastCutoff</td>
<td>Call Throttle.Floor(TargetThrottle); Call Gauges.Cruise(Off); UserMode := x;</td>
</tr>
<tr>
<td>c05</td>
<td>t04</td>
<td>Override</td>
<td>Override</td>
<td></td>
<td>x=NT &amp; UserMode = RA &amp; (SlowCutoff &lt; Gauges.Speed() &lt; FastCutoff) &amp; AutoSystem.BrakeActive() = false &amp; AutoSystem.ClutchActive() = false</td>
<td>Call Throttle.Floor(TargetThrottle); Call Gauges.Cruise(Off); UserMode := NT; Pause; CurrentSpeed := Gauges.Speed(); Put CheckState() on Call Queue;</td>
</tr>
</tbody>
</table>
Size of Engine System

- 10 classes
- 46 class variables
- 97 methods
- 76 states
- 143 transitions

Component Flow Graph

- 208 Nodes
- 551 Edges
- 3433 Def-use pairs

CFG is generated in a few seconds

Finding Candidate Test Paths

<table>
<thead>
<tr>
<th>Step</th>
<th>New Paths</th>
<th>New DU-pairs</th>
<th>Unsolved DU-Pairs</th>
<th>Not Def-clear</th>
<th>Partial Paths</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>18</td>
<td>18</td>
<td>3433</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3316</td>
<td>0</td>
<td>3316</td>
</tr>
<tr>
<td>3</td>
<td>363</td>
<td>363</td>
<td>2948</td>
<td>5</td>
<td>4174</td>
</tr>
<tr>
<td>4</td>
<td>69</td>
<td>69</td>
<td>2879</td>
<td>0</td>
<td>15,077</td>
</tr>
<tr>
<td>5</td>
<td>291</td>
<td>287</td>
<td>2564</td>
<td>28</td>
<td>49,664</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>231</td>
<td>214</td>
<td>445</td>
<td>6</td>
<td>46,509</td>
</tr>
<tr>
<td>12</td>
<td>26</td>
<td>17</td>
<td>428</td>
<td>0</td>
<td>50,822</td>
</tr>
</tbody>
</table>
Object-oriented Inter-class Test Process

Test Suite Generator

Executable Test Sequences

Test Harness

Graphical User Interface

Java Imp of Spec

Java Rapid Prototype Machine

DB Rep of Spec

Test Suite Generation – Integration Testing

Test Suite Generator

Candidate Test Paths

Def/Use Nodes & Edges

Component Flow Graph

Identify Relevant Transitions

Identify Component To Test

Executable Test Sequences

Not automated yet
Rapid Prototype Machine

Java Rapid Prototype Machine

Java Object for each Spec object

Message Queue for each Object

Simulated Parallel Processing

Graphical User Interface

Screenshot of Auto System
Summary of Results

• Inter-class testing technique
• Test process for systems comprised of
  – Components that run as separate processes
  – Communication via message passing
  – Object-oriented software
• Tests based on class descriptions
  – Methods
  – State variables
  – Defs and uses of state variables by methods
• Database representation of object state behavior

Current and Future Work

• Complete the automation of the executable tests
• Mapping problem
  – What methods must be called to trigger each transition?
  – What user-level actions must be carried out
• Web application interface
• Conveniently capture class information into database
  – Accept and translate UML diagrams
  – Form-based graphical user interface
• Full experimentation with fault study