Model based testing

Automated testing and verification
Model based testing

• We already know how it works... right?

• At least partially...
Model based testing

- We already know how it works... right?

- At least partially.

Control flow and data flow graphs are models, too.
Why model-based testing?

• Models used in specification or design have structure
  • Useful information for selecting representative classes of behavior;
  • Difficult to capture that structure clearly and correctly in constraints in combinatorial testing.
• We can devise test cases to check actual behavior against behavior specified by the model
  • “Coverage” similar to structural testing, but applied to specification and design models
Deriving test cases from finite state machines

A common kind of model for describing behavior that depends on sequences of events
Finite State Machines

• Good at describing interactions in systems with a small number of modes.

• Good at describing transducers (via finite state machines).

• Widely used in industry (Statecharts).

• Most systems are “infinite state” (or effectively so), but many systems are finite state + parameters – there are a finite set of states that control the way data is moved around.

• Good examples are systems like communication protocols or many classes of control systems (e.g. automated braking, flight control systems).

• Good for describing interactive systems that rarely reach a final state
From an informal specification...

**Maintenance:** The Maintenance function records the history of items undergoing maintenance.

If the product is covered by warranty or maintenance contract, maintenance can be requested either by calling the maintenance toll free number, or through the web site, or by bringing the item to a designated maintenance station.

If the maintenance is requested by phone or web site and the customer is a US or EU resident, the item is picked up at the customer site, otherwise, the customer shall ship the item with an express courier.

If the maintenance contract number provided by the customer is not valid, the item follows the procedure for items not covered by warranty.

If the product is not covered by warranty or maintenance contract, maintenance can be requested only by bringing the item to a maintenance station. The maintenance station informs the customer of the estimated costs for repair. Maintenance starts only when the customer accepts the estimate.

If the customer does not accept the estimate, the product is returned to the customer.

Small problems can be repaired directly at the maintenance station. If the maintenance station cannot solve the problem, the product is sent to the maintenance regional headquarters (if in US or EU) or to the maintenance main headquarters (otherwise).

If the maintenance regional headquarters cannot solve the problem, the product is sent to the maintenance main headquarters.

Maintenance is suspended if some components are not available.

Once repaired, the product is returned to the customer.
From an informal specification...

**Maintenance:** The Maintenance function records the history of items undergoing maintenance.

If the product is covered by warranty or maintenance contract, maintenance can be requested either by calling the maintenance toll free number, or through the web site, or by bringing the item to a designated maintenance station.

If the maintenance is requested by phone or web site and the customer is a US or EU resident, the item is picked up at the customer site, otherwise, the customer shall ship the item with an express courier.

If the maintenance contract number provided by the customer is not valid, the item follows the procedure for items not covered by warranty.

If the product is not covered by warranty or maintenance contract, maintenance can be requested only by bringing the item to a maintenance station. The maintenance station informs the customer of the estimated costs for repair. Maintenance starts only when the customer accepts the estimate.

If the customer does not accept the estimate, the product is returned to the customer.

Small problems can be repaired directly at the maintenance station. If the maintenance station cannot solve the problem, the product is sent to the maintenance regional headquarters (if in US or EU) or to the maintenance main headquarters (otherwise).

If the maintenance regional headquarters cannot solve the problem, the product is sent to the maintenance main headquarters.

Maintenance is suspended if some components are not available.

Once repaired, the product is returned to the customer.
...to a finite state machine...
...to a test suite

| TC1 | 0 | 2 | 4 | 1 | 0 |
| TC2 | 0 | 5 | 2 | 4 | 5 | 6 | 0 |
| TC3 | 0 | 3 | 5 | 9 | 6 | 0 |
| TC4 | 0 | 3 | 5 | 7 | 5 | 8 | 7 | 8 | 9 | 6 | 0 |

Meaning: From state 0 to state 2 to state 4 to state 1 to state 0

Is this a thorough test suite? How can we judge?

(c) 2007 Mauro Pezzè & Michal Young
TC1  0  2  4  1  0
TC2  0  5  2  4  5  6  0
TC3  0  3  5  9  6  0
TC4  0  3  5  7  5  8  7  8  9  6  0
“Covering” finite state machines

• **State coverage:**
  - Every state in the model should be visited by at least one test case.

• **Transition coverage**
  - Every transition between states should be traversed by at least one test case.
    - *This is the most commonly used criterion*
  - A transition can be thought of as a (precondition, postcondition) pair.
Path sensitive criteria?

• Basic assumption: States fully summarize history
  • No distinction based on how we reached a state; this should be true of well-designed state machine models

• If the assumption is violated, we may distinguish paths and devise criteria to cover them
  • **Single state path coverage:**
    • traverse each subpath that reaches each state at most once
  
  • **Single transition path coverage:**
    • traverse each subpath that reaches each transition at most once

  • **Boundary interior loop coverage:**
    • each distinct loop of the state machine must be exercised the minimum, an intermediate, and the maximum or a large number of times
    • *Of the path sensitive criteria, only boundary-interior is common*
Completeness of the model

• Finite State Machines are usually incomplete
  • *don’t care* transitions (do anything or nothing)
  • *error* transitions (trigger error-handling procedure)
  • *self* transitions (remain in the same state)

• Implicit transitions usually represent impossible or irrelevant transitions.

• Not necessary to test them
Testing decision structures

Some specifications are structured as decision tables, decision trees, or flow charts. We can exercise these as if they were program source code.
Pricing: The pricing function determines the adjusted price of a configuration for a particular customer.

The scheduled price of a configuration is the sum of the scheduled price of the model and the scheduled price of each component in the configuration. The adjusted price is either the scheduled price, if no discounts are applicable, or the scheduled price less any applicable discounts.

There are three price schedules and three corresponding discount schedules, Business, Educational, and Individual.

• Educational prices: The adjusted price for a purchase charged to an educational account in good standing is the scheduled price from the educational price schedule. No further discounts apply.

• Special-price non-discountable offers: Sometimes a complete configuration is offered at a special, non-discountable price. When a special, non-discountable price is available for a configuration, the adjusted price is the non-discountable price or the regular price after any applicable discounts, whichever is less.
Boolean expressions as outputs

(individual account

AND NOT current purchase > tier1 individual threshold
AND NOT special offer price < individual scheduled price )

OR ( business account

AND NOT current purchase > tier1 business threshold
AND NOT current purchase > tier1 business yearly threshold
AND NOT special offer price < business scheduled price )

-> no discounts
...to a decision table...

<table>
<thead>
<tr>
<th></th>
<th>edu</th>
<th>individual</th>
</tr>
</thead>
<tbody>
<tr>
<td>EduAc</td>
<td>T T</td>
<td>F F F F F F</td>
</tr>
<tr>
<td>BusAc</td>
<td>- -</td>
<td>F F F F F F</td>
</tr>
<tr>
<td>CP &gt; CT1</td>
<td>- -</td>
<td>F F T T - -</td>
</tr>
<tr>
<td>YP &gt; YT1</td>
<td>- -</td>
<td>- - - - - -</td>
</tr>
<tr>
<td>CP &gt; CT2</td>
<td>- -</td>
<td>- - F F T T</td>
</tr>
<tr>
<td>YP &gt; YT2</td>
<td>- -</td>
<td>- - - - - -</td>
</tr>
<tr>
<td>SP &lt; Sc</td>
<td>F T</td>
<td>F T - - - -</td>
</tr>
<tr>
<td>SP &lt; T1</td>
<td>- -</td>
<td>- - F T - -</td>
</tr>
<tr>
<td>SP &lt; T2</td>
<td>- -</td>
<td>- - - - F T</td>
</tr>
<tr>
<td>out</td>
<td>Edu SP ND SP T1 SP T2 SP</td>
<td></td>
</tr>
</tbody>
</table>

(c) 2007 Mauro Pezzè & Michal Young
<table>
<thead>
<tr>
<th></th>
<th>EduAc</th>
<th>BusAc</th>
<th>CP &gt; CT1</th>
<th>YP &gt; YT1</th>
<th>CP &gt; CT2</th>
<th>YP &gt; YT2</th>
<th>SP &gt; Sc</th>
<th>SP &gt; T1</th>
<th>SP &gt; T2</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>-</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>-</td>
<td>-</td>
<td>F</td>
<td>-</td>
<td>-</td>
<td>ND</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>-</td>
<td>-</td>
<td>F</td>
<td>-</td>
<td>-</td>
<td>SP</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td></td>
<td>-</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
</tbody>
</table>

(c) 2007 Mauro Pezzè & Michal Young

Thursday, January 17, 13
...with constraints...

at-most-one (EduAc, BusAc)

at-most-one (YP < YT1, YP > YT2)

YP > YT2 -> YP > YT1

at-most-one (CP < CT1, CP > CT2)

CP > CT2 -> CP > CT1

at-most-one (SP < T1, SP > T2)

SP > T2 -> SP > T1
...to test cases

- **Basic condition coverage**
  - a test case specification for each column in the table

- **Compound condition adequacy criterion**
  - a test case specification for each combination of truth values of basic conditions

- **Modified condition/decision adequacy criterion (MC/DC)**
  - each column in the table represents a test case specification.
  - we add columns that differ in one input row and in outcome, then merge compatible columns
# Example MC/DC

<table>
<thead>
<tr>
<th></th>
<th>C.1</th>
<th>C.1a</th>
<th>C.1b</th>
<th>C.10</th>
</tr>
</thead>
<tbody>
<tr>
<td>EduAc</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>-</td>
</tr>
<tr>
<td>BusAc</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>T</td>
</tr>
<tr>
<td>CP &gt; CT1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>F</td>
</tr>
<tr>
<td>YP &gt; YT1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>F</td>
</tr>
<tr>
<td>CP &gt; CT2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>YP &gt; YT2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SP &gt; Sc</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>SP &gt; T1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SP &gt; T2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>out</td>
<td>Edu</td>
<td>*</td>
<td>*</td>
<td>SP</td>
</tr>
</tbody>
</table>

(c) 2007 Mauro Pezzè & Michal Young
### Example MC/DC

<table>
<thead>
<tr>
<th></th>
<th>C.1</th>
<th>C.1a</th>
<th>C.1b</th>
</tr>
</thead>
<tbody>
<tr>
<td>EduAc</td>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>BusAc</td>
<td>-</td>
<td>-</td>
<td>F</td>
</tr>
<tr>
<td>CP &gt; CT1</td>
<td>-</td>
<td>-</td>
<td>T</td>
</tr>
<tr>
<td>YP &gt; YT1</td>
<td>-</td>
<td>-</td>
<td>F</td>
</tr>
<tr>
<td>CP &gt; CT2</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>YP &gt; YT2</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SP &gt; Sc</td>
<td>F</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>SP &gt; T1</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>SP &gt; T2</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>out</td>
<td>Edu</td>
<td>*</td>
<td>SP</td>
</tr>
</tbody>
</table>

*Generate C.1a and C.1b by flipping one element of C.1*

*C.1b can be merged with an existing column (C.10) in the spec*

*Outcome of generated columns must differ from source column*
Flowgraph based testing

If the specification or model has both decisions and sequential logic, we can cover it like program source code.
from an informal spec (i/iii)...

• **Process shipping order**: The Process shipping order function checks the validity of orders and prepares the receipt

A valid order contains the following data:

• cost of goods: If the cost of goods is less than the minimum processable order (MinOrder) then the order is invalid.

• shipping address: The address includes name, address, city, postal code, and country.

• preferred shipping method: If the address is domestic, the shipping method must be either land freight, expedited land freight, or overnight air; If the address is international, the shipping method must be either air freight, or expedited air freight.
a shipping cost is computed based on

- address and shipping method.

- type of customer which can be individual, business, educational

- preferred method of payment. Individual customers can use only credit cards, business and educational customers can choose between credit card and invoice

- card information: if the method of payment is credit card, fields credit card number, name on card, expiration date, and billing address, if different than shipping address, must be provided. If credit card information is not valid the user can either provide new data or abort the order
The outputs of Process shipping order are

- **validity**: Validity is a boolean output which indicates whether the order can be processed.

- **total charge**: The total charge is the sum of the value of goods and the computed shipping costs (only if validity = true).

- **payment status**: if all data are processed correctly and the credit card information is valid or the payment is invoice, payment status is set to valid, the order is entered and a receipt is prepared; otherwise validity = false.
...to a flowgraph

Process shipping order

1. CostOfGoods < MinOrder
   - no
     - international shipping address
     - preferred shipping method = air freight OR expedited air freight
       - calculate international shipping charge
       - total charge = goods + shipping
       - individual customer
         - yes
           - method of payment
             - credit card
               - yes
                 - obtain credit card data: number, name on card, expiration date
                 - billing address = shipping address
                   - yes
                     - payment status = valid
                       - enter order
                         - prepare receipt
                       - invalid order
                     - no
                       - abort order?
                         - yes
                           - invalid order
                         - no
                           - obtain billing address
                             - valid credit card information
                               - yes
                                 - payment status = valid
                                   - enter order
                                     - prepare receipt
                                   - invalid order
                                 - no
                               - no
                           - abort order?
                             - yes
                               - invalid order
                             - no
...from the flow graph to test cases

<table>
<thead>
<tr>
<th>Case</th>
<th>Too Small</th>
<th>Ship Where</th>
<th>Ship Method</th>
<th>Cust Type</th>
<th>Pay Method</th>
<th>Same Address</th>
<th>CC valid</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC-1</td>
<td>No</td>
<td>Int</td>
<td>Air</td>
<td>Bus</td>
<td>CC</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>TC-2</td>
<td>No</td>
<td>Dom</td>
<td>Land</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TC-3</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TC-4</td>
<td>No</td>
<td>Dom</td>
<td>Air</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TC-5</td>
<td>No</td>
<td>Int</td>
<td>Land</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TC-6</td>
<td>No</td>
<td>-</td>
<td>-</td>
<td>Edu</td>
<td>Inv</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>TC-7</td>
<td>No</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>CC</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>TC-8</td>
<td>No</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>CC</td>
<td>-</td>
<td>No (abort)</td>
</tr>
<tr>
<td>TC-9</td>
<td>No</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>CC</td>
<td>-</td>
<td>No (no abort)</td>
</tr>
</tbody>
</table>

Branch testing: cover all branches

(c) 2007 Mauro Pezzè & Michal Young

Thursday, January 17, 13
Grammar-based testing

Complex input is (or can) often be described by a context-free grammar
Grammars in specifications

• Grammars are good at:
  • Representing inputs of varying and unbounded size
  • With recursive structure
  • And boundary conditions

• Examples:
  • Complex textual inputs
  • Trees (search trees, parse trees, ...)
    • Note XML and HTML are trees in textual form
  • Program structures
    • Which are also tree structures in textual format!
Grammar-based testing

• Test cases are strings \textit{generated} from the grammar

• Coverage criteria:
  
  • \textbf{Production coverage}: each production must be used to generate at least one (section of) test case

  • \textbf{Boundary condition}: annotate each recursive production with minimum and maximum number of application, then generate:

    • Minimum
    • Minimum + 1
    • Maximum - 1
    • Maximum
from an informal specification (i/iii)... 

• The Check-configuration function checks the validity of a computer configuration.

• The parameters of check-configuration are:
  
  • Model

  • Set of components
• **Model**: A model identifies a specific product and determines a set of constraints on available components. Models are characterized by logical slots for components, which may or may not be implemented by physical slots on a bus. Slots may be required or optional. Required slots must be assigned with a suitable component to obtain a legal configuration, while optional slots may be left empty or filled depending on the customers' needs.

• Example: The required "slots" of the C20 laptop computer include a screen, a processor, a hard disk, memory, and an operating system. (Of these, only the hard disk and memory are implemented using actual hardware slots on a bus.) The optional slots include external storage devices such as a CD/DVD writer.
• Set of Components: A set of [slot,component] pairs, which must correspond to the required and optional slots associated with the model. A component is a choice that can be varied within a model, and which is not designed to be replaced by the end user. Available components and a default for each slot is determined by the model. The special value empty is allowed (and may be the default selection) for optional slots. In addition to being compatible or incompatible with a particular model and slot, individual components may be compatible or incompatible with each other.

• Example: The default configuration of the C20 includes 20 gigabytes of hard disk; 30 and 40 gigabyte disks are also available. (Since the hard disk is a required slot, empty is not an allowed choice.) The default operating system is Ubuntu 12, personal edition, Ubuntu 10 edition may also be selected. The 12 edition requires at least 30 gigabytes of hard disk.
...to a grammar

⟨Model⟩ ::= ⟨modelNumber⟩ ⟨compSequence⟩
    ⟨optCompSequence⟩
⟨compSequence⟩ ::= ⟨Component⟩ ⟨compSequence⟩ | empty
 ⟨optCompSequence⟩ ::= ⟨OptionalComponent⟩ ⟨optCompSequence⟩ | empty
 ⟨Component⟩ ::= ⟨ComponentType⟩ ⟨ComponentValue⟩
 ⟨OptionalComponent⟩ ::= ⟨ComponentType⟩
 ⟨modelNumber⟩ ::= string
 ⟨ComponentType⟩ ::= string
 ⟨ComponentValue⟩ ::= string
...to a grammar with limits

Model

compSeq1 [0, 16]
compSeq2
optCompSeq1 [0, 16]
optCompSeq2
Comp
OptComp
modNum
CompTyp
CompVal

::= <modelNumber> <compSequence>
  <optCompSequence>
  ::= <Component> <compSequence>
    ::= empty
  <optCompSequence>
    ::= <OptionalComponent>
      <optCompSequence>
        ::= empty
      <compSequence>
        ::= <ComponentType> <ComponentValue>
          ::= <ComponentType>
            ::= string
          ::= string
          ::= string
…to test cases

- “Mod000”
  - Covers Model, compSeq1[0], compSeq2, optCompSeq1[0], optCompSeq2, modNum

- “Mod000 (Comp000, Val000) (OptComp000)”
  - Covers Model, compSeq1[1], compSeq2, optCompSeq2[0], optCompSeq2, Comp, OptComp, modNum, CompTyp, CompVal
Boundary condition grammar-based criterion

- \text{compSeq1}[0] (0 times)
- \text{compSeq1}[1] (1 time)
- \text{compSeq1}[15] (n-1 times)
- \text{compSeq1}[16] (n times)
- \text{compSeq1}[17] (n+1 times)
Grammar vs. Combinatorial Testing

• Combinatorial specification-based testing is good for “mostly independent” parameters
  
  • We can incorporate a few constraints, but complex constraints are hard to represent and use
  
  • We must often “factor and flatten”
    
    • E.g., separate “set of slots” into characteristics “number of slots” and predicates about what is in the slots (all together)
  
  • Grammar describes sequences and nested structure naturally
    
    • But some relations among different parts may be difficult to describe and exercise systematically, e.g., compatibility of components with slots
Probabilistic grammar-based criteria

- Assign probabilities to productions, indicating which production to select at each step to generate test cases.

- Probabilities as interpreted as weights that determine how frequent each production is used to generate a test case.
Summary: The big picture

• Models are useful abstractions

  • In specification and design, they help us think and communicate about complex artifacts by emphasizing key features and suppressing details

  • Models convey structure and help us focus on one thing at a time

• We can use them in systematic testing

  • If a model divides behavior into classes, we probably want to exercise each of those classes!

  • Common model-based testing techniques are based on state machines, decision structures, and grammars

    • but we can apply the same approach to other models
Model based testing for Object Oriented Software
Characteristics of OO Software

Typical OO software characteristics that impact testing

• State dependent behavior

• Encapsulation

• Inheritance

• Polymorphism and dynamic binding

• Abstract and generic classes

• Exception handling
Intraclass State Machine Testing

• Basic idea:
  - The state of an object is modified by operations
  - Methods can be modeled as state transitions
  - Test cases are sequences of method calls that traverse the state machine model

• State machine model can be derived from specification (functional testing), code (structural testing), or both
Informal state-full specifications

**Slot:** represents a slot of a computer model.

... slots can be bound or unbound. Bound slots are assigned a compatible component, unbound slots are empty. Class slot offers the following services:

- **Incorporate:** slots can be installed on a model as *required* or *optional*.
  
  ...

- **Bind:** slots can be bound to a compatible component.
  
  ...

- **Unbind:** bound slots can be unbound by removing the bound component.

- **IsBound:** returns the current binding, if bound; otherwise returns the special value *empty*.
Identifying states and transitions

• From the informal specification we can identify three states:
  - Not_present
  - Unbound
  - Bound

• and four transitions
  - incorporate: from Not_present to Unbound
  - bind: from Unbound to Bound
  - unbind: ...to Unbound
  - isBound: does not change state
Deriving an FSM and test cases

- TC-1: incorporate, isBound, bind, isBound
- TC-2: incorporate, unBind, bind, unBind, isBound
Testing with State Diagrams

- A statechart (called a "state diagram" in UML) may be produced as part of a specification or design
  
  - May also be implied by a set of message sequence charts (interaction diagrams), or other modeling formalisms

- Two options:
  
  - Convert ("flatten") into standard finite-state machine, then derive test cases
  
  - Use state diagram model directly

(c) 2007 Mauro Pezzè & Michal Young
Statecharts specification

class model

noModelSelected

selectModel(model)

send modelDB: getModel(modelID, this)

deselectModel()

modelSelected

addComponent(slot, component)

send modelDB: findComponent()

send slot: bind()

addComponent(slot, component)

send Component_DB: get_component()

send slot: bind

workingConfiguration

isLegalConfiguration()

[legalConfig = true]

removeComponent(slot)

send slot: unbind()

validConfiguration

removeComponent(slot)

send slot: unbind()
Statecharts specification

class model

super-state or “OR-state”

modelSelected

selectModel(model)

modelDB: getModel(modelID, this)

noModelSelected

deselectModel()

addComponent(slot, component)

send modelDB: findComponent()

send slot: bind()

addComponent(slot, component)

send Component_DB: get_component()

send slot: bind

workingConfiguration

isLegalConfiguration()

[legalConfig = true]

removeComponent(slot)

send slot: unbind()

removeComponent(slot)

send slot: unbind()

validConfiguration

boolean super-state or “OR-state”
Statecharts specification

**class model**

**super-state or “OR-state”**

- noModelSelected
  - selectModel(model)
  - deselectModel()

- modelSelected
  - addComponent(slot, component)
    - send modelDB: findComponent()
      - send slot:bind()
  - deselectModel()

- workingConfiguration
  - isLegalConfiguration()
    - legalConfig = true
  - removeComponent(slot)
    - send slot:unbind()

- validConfiguration
  - addComponent(slot, component)
    - send Component_DB: get_component()
      - send slot:bind

(c) 2007 Mauro Pezzè & Michal Young
Statecharts specification

class model

super-state or “OR-state”

modelSelected

addComponent(slot, component)

send modelDB: findComponent()
  send slot:bind()

addComponent(slot, component)

send Component_DB: get_component()
  send slot:bind

validConfiguration

isLegalConfiguration()
[legalConfig = true]

workingConfiguration

selectModel(model)

deselectModel()

removeComponent(slot)

send slot:unbind()

removeComponent(slot)

deselectModel()

send slot:unbind()

selectModel(model)

deselectModel()

addComponent(slot, component)

send modelDB: getModel(modelID, this)

removeComponent(slot)

deselectModel()

send slot:unbind()

(c) 2007 Mauro Pezzè & Michal Young

Thursday, January 17, 13
From Statecharts to FSMs

- noModelSelected
  - selectModel(model)
  - deselectModel()
  - addComponent(slot, component)

- workingConfiguration
  - removeComponent(slot)
  - isLegalConfiguration()
    - [legalConfig=true]
  - deselectModel()
  - addComponent(slot, component)

- validConfiguration
  - removeComponent(slot)
Statechart based criteria

- In some cases, “flattening” a Statechart to a finite-state machine may cause “state explosion”
  - Particularly for super-states with “history”
- Alternative: Use the statechart directly
- Simple transition coverage: execute all transitions of the original Statechart
  - incomplete transition coverage of corresponding FSM
  - useful for complex statecharts and strong time constraints (combinatorial number of transitions)