Model-based Testing

CSE 4495 - Lecture 8 - 30/08/2022

Instructor : Md. Mohaiminul Islam

mohaiminul@cse.uiu.ac.bd

Models and Software Analysis

- Before and while building products, engineers analyze models to address design questions.
- Software is no different.
- Software models capture different ways that the software *behaves* during execution.

Behavior Modeling

- **Abstraction** - simplify problem by identifying and focusing ***only*** on important aspects.
  - Solve a simpler problem, then apply to the big problem.
- A **model** is a simplified representation of an artifact.
  - Ignores all irrelevant elements of that artifact.

Software Models

- Model is an abstraction of system being developed.
  - By abstracting unnecessary details, powerful analyses can be performed.
- Can be extracted from specifications and design (or from code)
  - Illustrates *intended* behavior of the system.
  - Often **state machines**.
    - Events cause the system to react, changing its internal state.

Model-Driven Development

- Models often created during requirements analysis.
  - Allows refinement of requirements.
  - Prove that properties hold over model.
    - **Finite State Verification** (next class) - used to analyze requirements, plan development, create test cases.
- Can generate code from models.
- **Can create tests using model.**

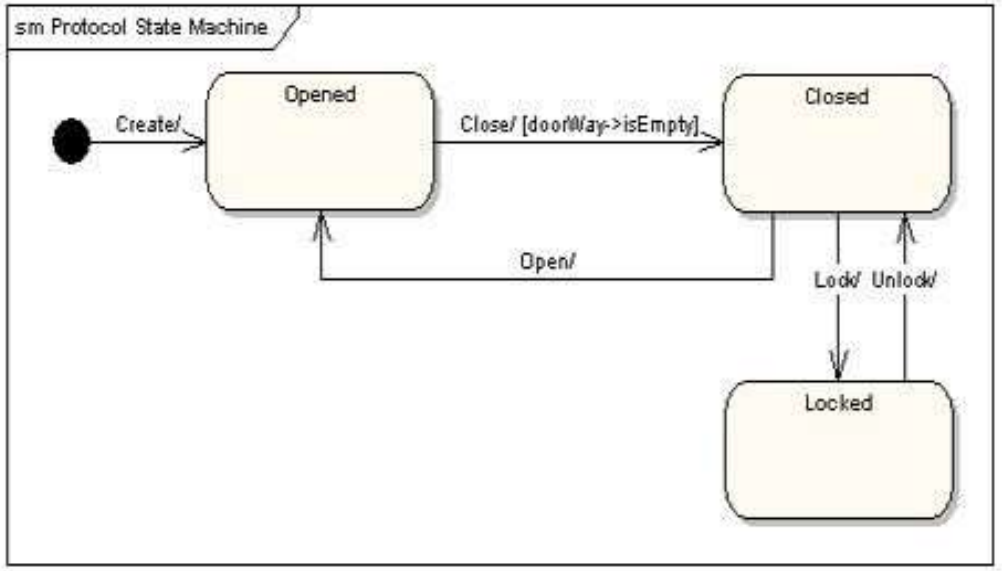
Finite State Machines

State Machines

- Program execution as sequence of states transformed by actions.
  - “Behavior” is state -> action -> state transitions.
- Set of all possible **real** behaviors is often infinite.
  - Called the “**state space**” of the program.
  - Models simplify a functionality or class state space into a finite set of states.

Finite State Machines

- Nodes represent states
  - Abstract description of the current value of an entity's attributes.
- Edges represent transitions
  - Events cause state to change.
  - Labeled **`event [guard] / activity`**
    - **`event`**:The event that triggered the transition.
    - **`guard`**: Conditions that must be true to transition.
    - **`activity`**: Output behavior when this transition is taken.



Terminology

- **Event** - An input that occurs at a defined time.
  - The user presses a self-test button.
  - The alarm goes off.
- **Condition** - Internal or external property describing a change over time.
  - The fuel level has risen over a threshold.
  - The alarm has been on for ten seconds.

Terminology

- **State** - Abstract description of the current value of the entity's attributes.
  - (ex: Not "X=5; Y=10", but "Normal Operating Mode")
  - The controller is in the "self-test" state after the self-test button has been pressed, and leaves it when the reset button has been pressed.
  - The tank is in the "too-low" state when the fuel level is below the set threshold for N seconds.

States, Transitions, and Guards

- States change in response to events (**transition**).
- When multiple transitions are possible, the choice is guided by the current conditions.
  - Also called the **guards** on a transition.
  - We take the transition that satisfies all guards.

State Transitions

Transitions are labeled in the form:

```
event [guard] / activity
```

- All three are optional.
  - Missing Activity: No output from this transition.
  - Missing Guard: Always take transition following event.
  - Missing Event: Take this transition immediately after entering preceding state (if guards met).

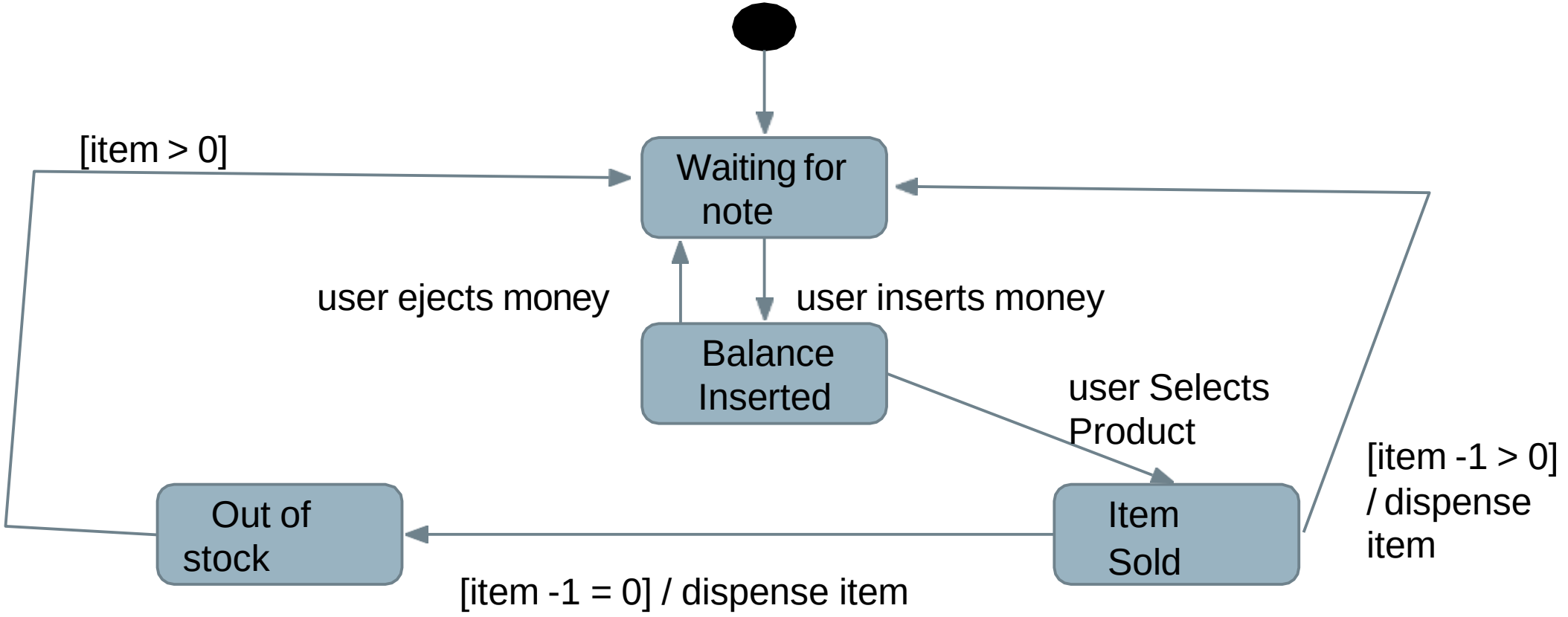
State Transition Examples

Transitions are labeled in the form:

```
event [guard] / activity
```

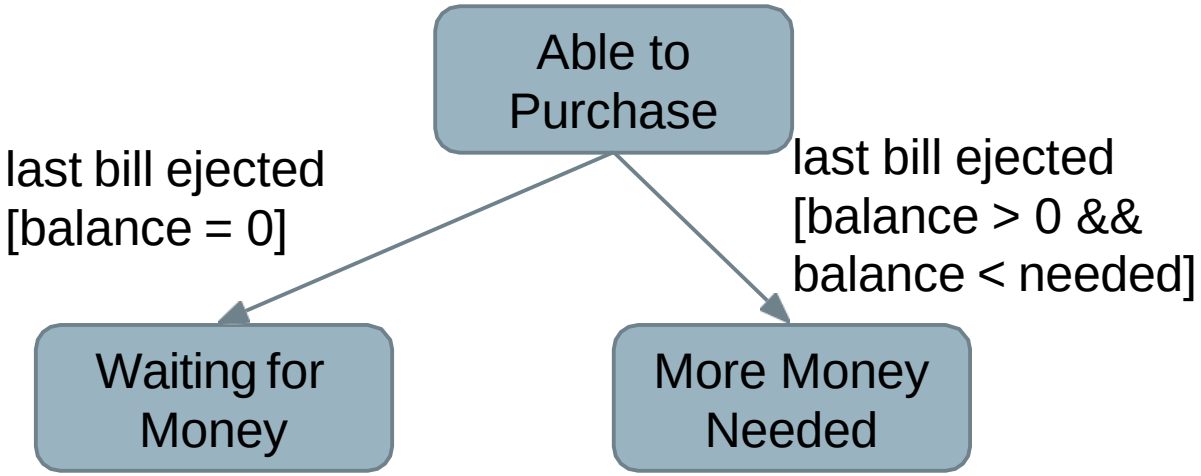
- Controller enters “self-test” mode after test button is pressed, leaves when reset button is pressed.
  - User pressing self-test, reset buttons are **`events.`**
- The tank enters “too-low” state when fuel level < threshold for N seconds.
  - Fuel level < threshold for N seconds is a **`guard`**.

Example: Simple Vending Machine



More on Transitions

Guards must be mutually exclusive



If event occurs and no transition is valid, then event is ignored.

last bill ejected [balance > 0 && balance >= needed]

Internal Activities

Can react to events and conditions without transitioning using internal activities.

Typing

entry / highlight all
exit / update field
character entered / add to field
help requested [verbose] / open help page
help requested [minimal] / update status bar

- Special events: **entry** and **exit**.
- Other activities occur each “time step”, until a transition occurs.
  - Entry and exit not re-triggered.

Example: Maintenance Tracking

- Customers send products for maintenance.
- **Maintenance tracking function** notes current stage of maintenance process for each customer.
  - Transition path determined by a set of rules.
  - States = stages of process.
  - Transitions shows paths through process.
- ***Model only what software tracks and controls!***

Example: Maintenance Tracking

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

No Maintenance

Waiting for Pick Up

If the maintenance is requested by web and the customer is a US resident, the item is picked up from the customer. Otherwise, the customer will ship the item.

Request - No Warranty

If the product is not covered by warranty or the warranty number is not valid, the item must be brought to a maintenance station. The station informs the customer of the estimated cost. Maintenance starts when the customer accepts the estimate. If the customer does not accept, the item is returned.

Wait for Acceptance

Wait for Returning

Example: Maintenance Tracking

Repair at Station

If the maintenance station cannot solve the problem, the product is sent to the regional headquarters (if in the US) or the main headquarters (otherwise). If the regional headquarters cannot solve the problem, the product is sent to main headquarters.

Repair at Regional HQ

Repair at Main HQ

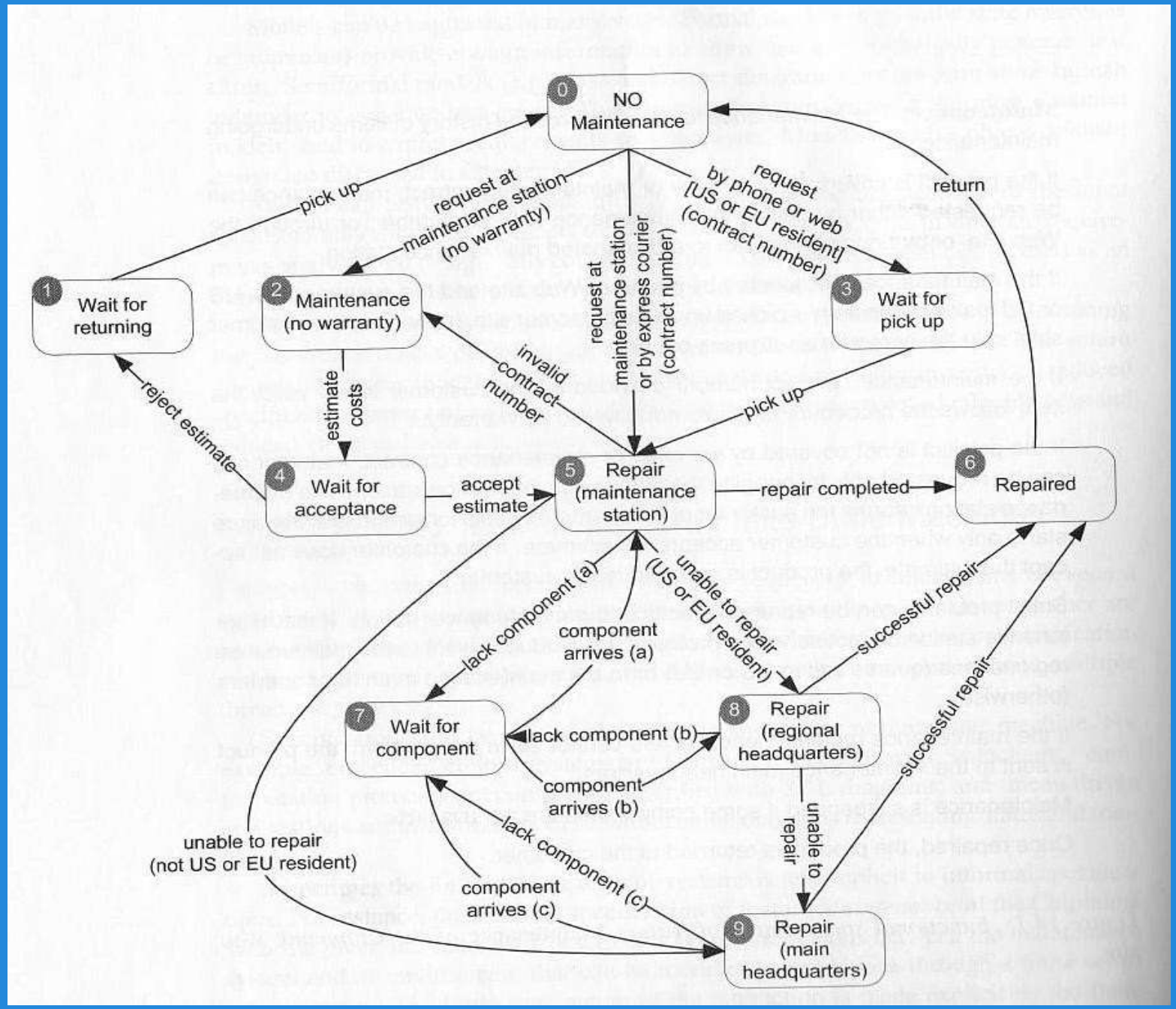
Maintenance is suspended if some components are not available.

Wait for Component

Once repaired, the product is returned to the customer.

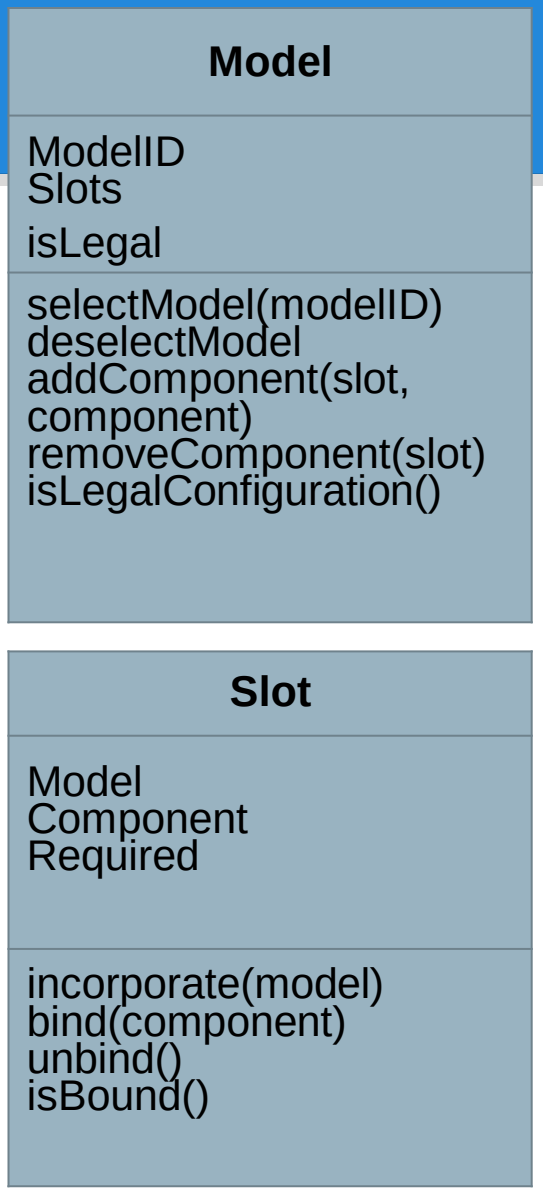
Repaired

Example: Maintenance Tracking



Example - Computer Model

- Many classes have stateful behavior.
  - States = class variables
  - Transitions = method calls
  - Derive model from class and create tests.
- Ex: We sell computers on our website. Model class represents a model of computer.

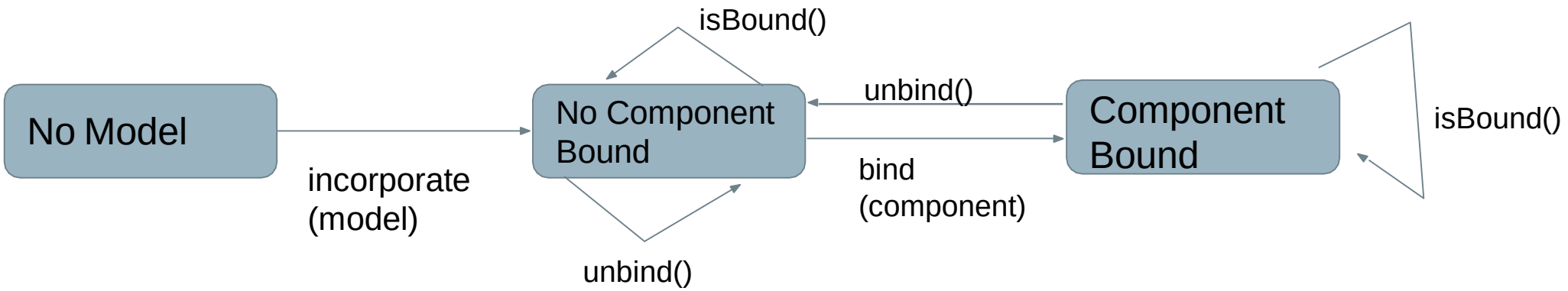


Slot Specification

Slot represents a configuration choice in all instances of a particular model of computer. A given model may have zero or more slots, each of which is marked as required or optional. If a slot is marked as required, it must be bound to a suitable component in all legal configurations. Slot offers the following methods:

- **Incorporate:** Make a slot part of a model, and mark it as either required or optional. All instances of a model incorporate the same slots.
- **Bind:** Associate a compatible component with a slot.
- **Unbind:** The unbind operation breaks the binding of a component to a slot, reversing the effect of a previous bind operation.
- **IsBound**: Returns true if a component is currently bound to a slot, or false if the slot is currently empty.

Slot State Machine



- Do not derive too many states.
  - Map variables to abstract values, not a state for each possible combination of values.
- Model how a method affects a class.
  - States only need to capture interactions between methods and the class state.

Example - Model

Model represents the current configuration of a model of computer.

- A given model may have zero or more slots, each of which is marked as required or optional.
- Each slot may contain a single component.
- To be a legal model, the model ID must exist in the ModelDB, each slot marked as required must be filled, the configuration must match that of the ModelDB entry for the model ID, and the optional components must match those allowed for that model in the ModelDB.

Example - Model

- **selectModel(modelID)**: Sets the model ID to the value passed in, as long as the model ID is set to “no model selected”. A model ID must be set before any other services are requested.
- **deselectModel():** Sets the model ID to “no model selected”. If the configuration was previously judged to be legal, it is no longer legal.
- **addComponent(slot, component):** Adds the selected component to the selected slot. If the configuration was previously judged to be legal, it is no longer legal.
- **removeComponent(slot)**: Removes the selected component to the selected slot. If the configuration was previously judged to be legal, it is no longer legal.
- **isLegalConfiguration():** Compares the current configuration to the entry in ModelDB. If the configuration is valid, the Model’s isLegal field is set to “true”.

Choosing States

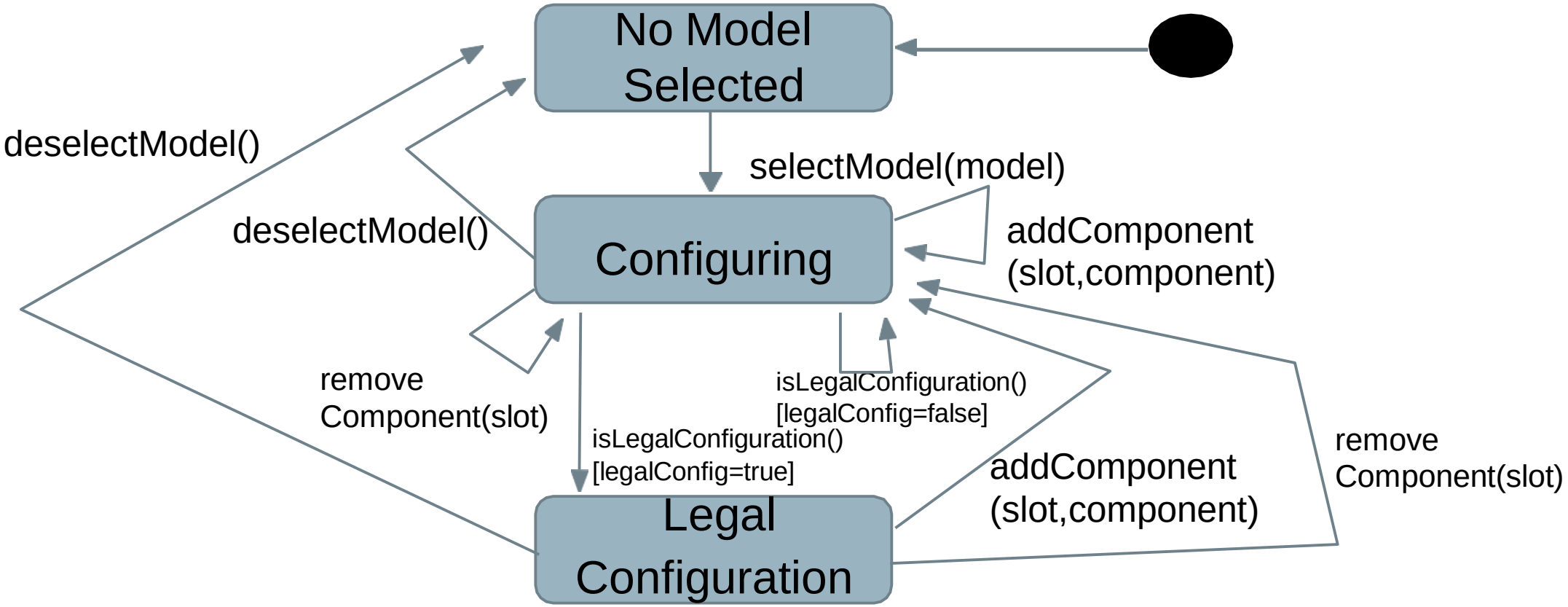
No Model Selected

Configuring

Legal Configuration

- What does the class represent?
  - e.g., a computer model.
- What causes method results to differ?
  - e.g., whether the model is legal or illegal.
- Can the class be in any other states?
  - e.g., we may not have set the model yet, we could still be making decisions and have not determined legality.

Choosing Transitions and Initial State



Model Coverage Criteria

Test Creation

- Tests created from models can be applied to the real program.
  - Events translated into method/API calls.
  - Program output (abstracted) should match model output.
- Model coverage maps to requirements coverage.
  - Tests should be effective for verification.
  - Exercises stateful behavior thoroughly.
  - Coverage criteria based on states, transitions, paths.

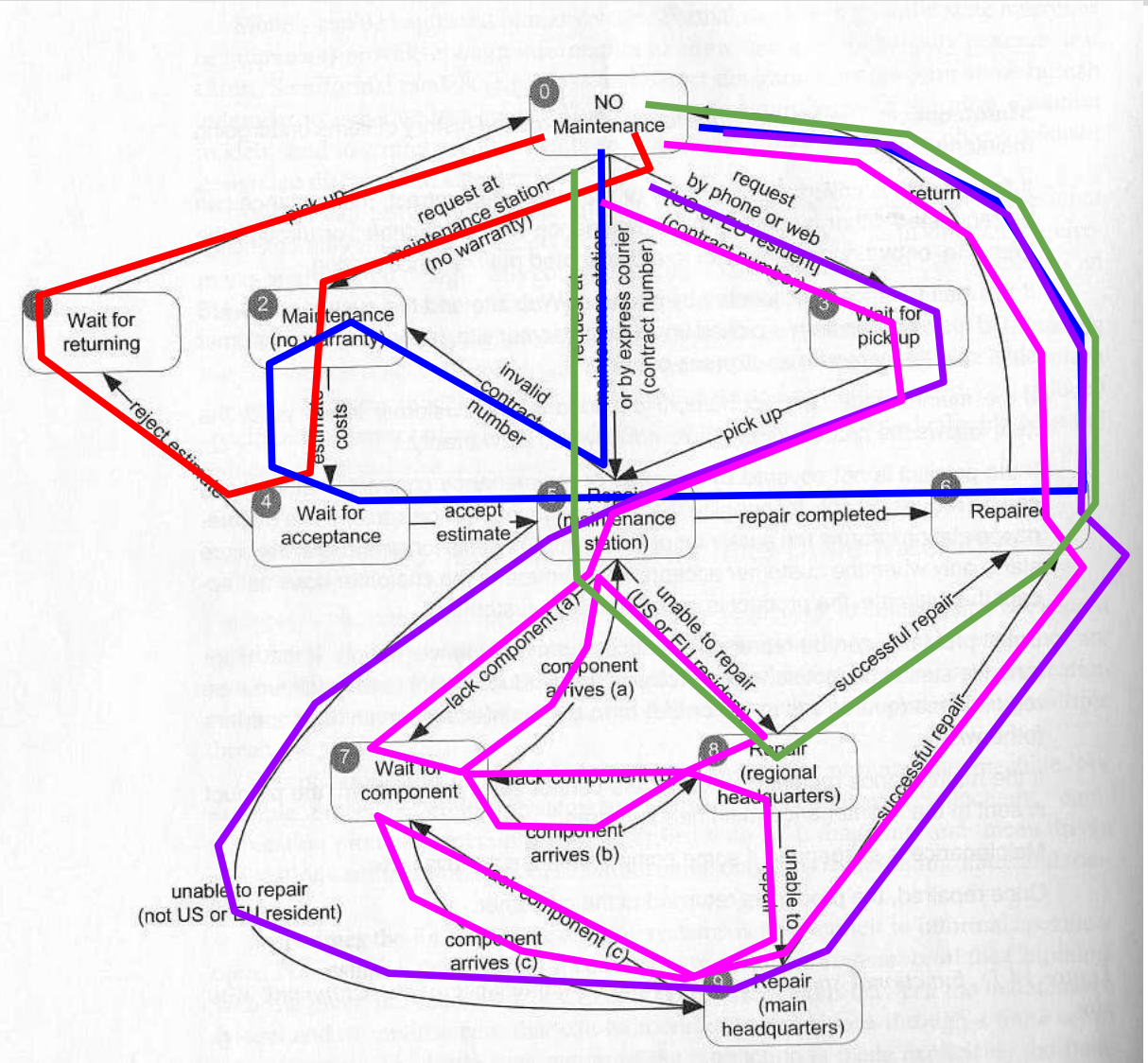
State Coverage

- **Each state must be reached by test cases.**
  - Unless model has been placed in each state, faults cannot be revealed.
  - **Num. of Covered States / Number of States**
- Easy to understand and obtain, but low fault-revealing power.
  - Software takes action during transitions
  - Most states can be reached through multiple transitions.

Transition Coverage

- A transition specifies a pre/post-condition.
  - “If system is in state S and sees event I, then after reacting to it, the system will be in state T.”
  - Faulty system could violate (pre, post-condition) pairs.
- Every transition must be covered by test cases.
  - **Num. Covered Transitions / Number of Transitions**

Example: Maintenance



- If no “final” states, we could achieve transition coverage with one large test case.
  - Smarter to target sections in different test cases.

Example Paths:

T1: request w/ no warranty (0->2) - estimate costs (2->4) - reject (4->1) - pick up (1->0)

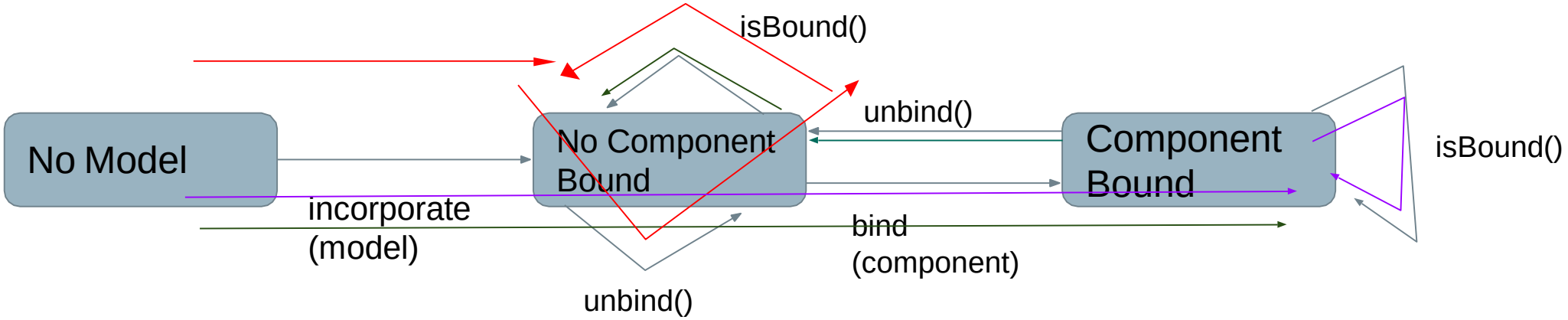
T2: 0->5->2->4->5->6->0

T3: 0->3->5->9->6->0

T4: 0->3->5->7->5->8->7->8->9->7->9->6->0

T5: 0->5->8->6->0

Example - Slot

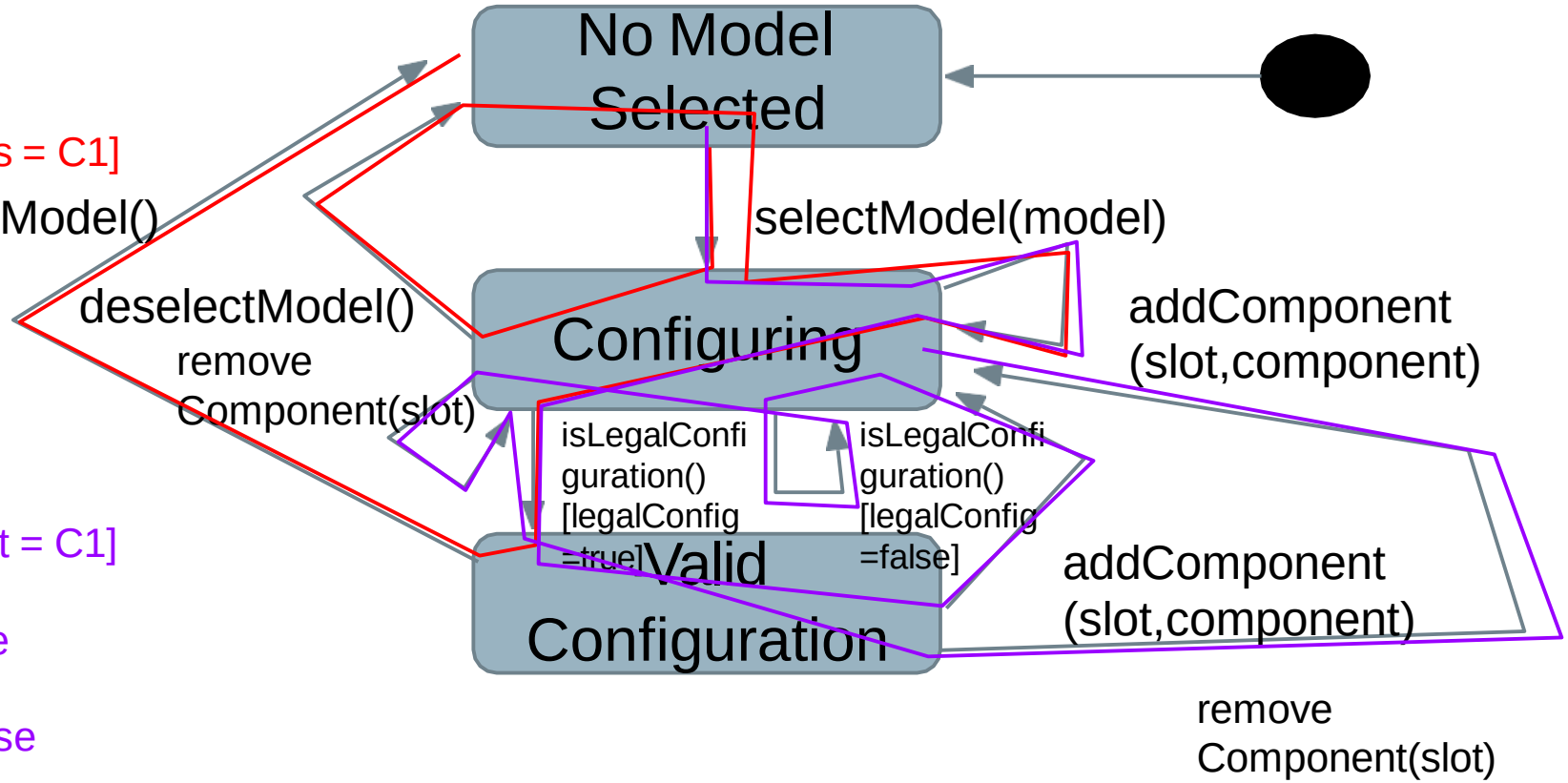


- Tests are series of method calls.
  - incorporate(model), isBound(), unbind()
  - incorporate(model), bind(component), isBound()
  - incorporate(model), bind(component), unbind(), isBound()

Example - Model

TC1:
selectModel(M1) [M1, 1 slots = C1]
deselectModel()
selectModel(M1)
addComponent(S1,C1)
isLegalConfiguration() //true
deselectModel()

TC2:
selectModel(M1) [M1, 1 slot = C1]
addComponent(S1,C1)
isLegalConfiguration() //true
addComponent(S2,C2)
isLegalConfiguration() // false
removeComponent(S2)
isLegalConfiguration() // true
removeComponent(S1)



Path Coverage Criteria

- Transition coverage based on assumption that transitions are independent.
- Many machines exhibit “history sensitivity”.
  - Transitions available depend on path taken.
    - “wait for component” in Maintenance Tracking example.
- Path-based metrics can cope with sensitivity.

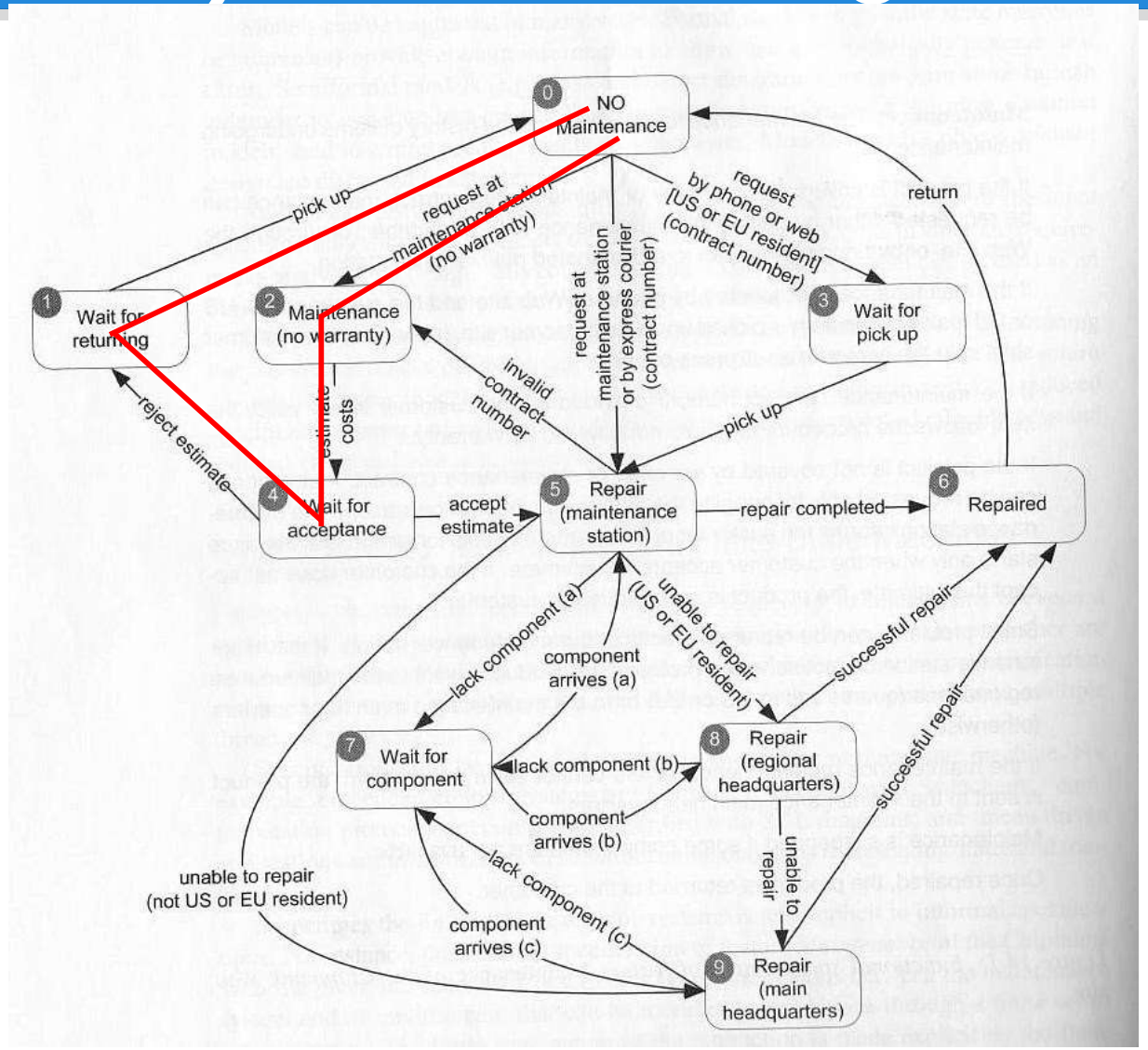
Path Coverage Metrics

- Single State Path Coverage
  - Requires that each subpath that traverses states at most once to be included in a path that is exercised.
- Single Transition Path Coverage
  - Requires that each subpath that traverses a transition at most once to be included in a path that is exercised.
- Boundary Interior Loop Coverage
  - Each distinct loop must be exercised minimum, an intermediate, and a large number of times.

Single State (Transition) Path Coverage

Single State (or Transition) Path Coverage

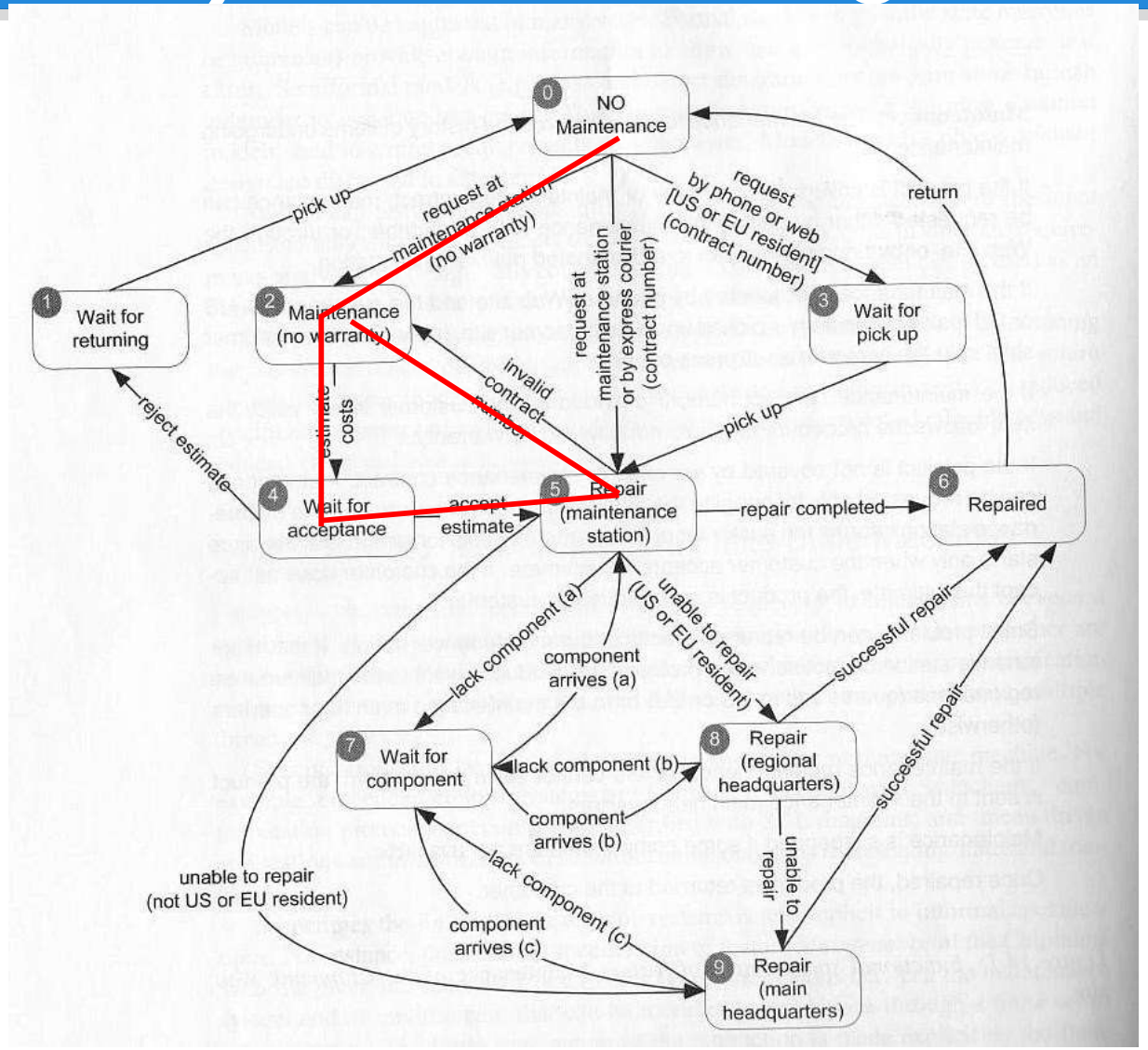
- Each subpath that traverses a state (or transition) **at most once** must be exercised.



Single State (Transition) Path Coverage

Single State (or Transition) Path Coverage

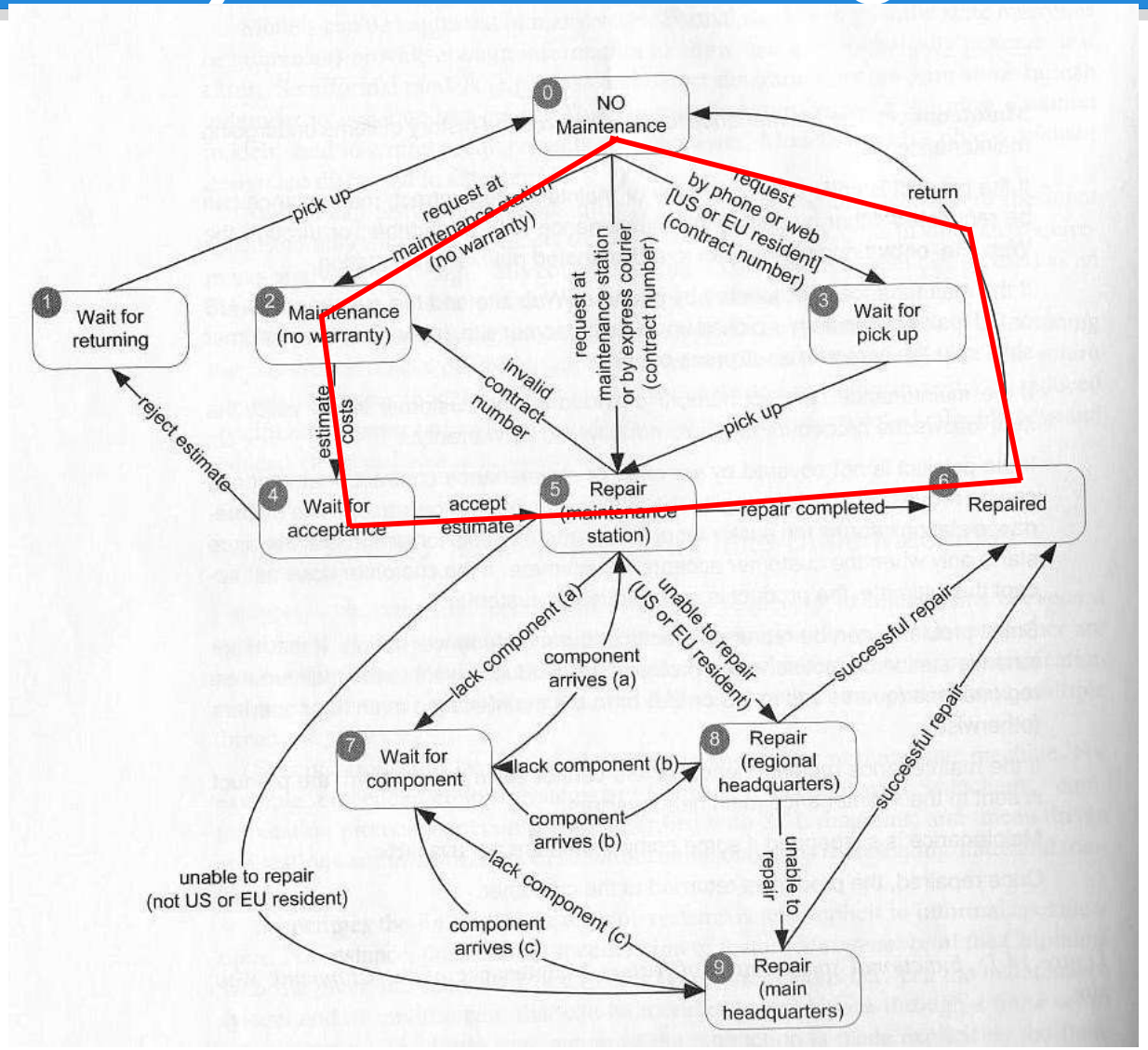
- Each subpath that traverses a state (or transition) **at most once** must be exercised.



Single State (Transition) Path Coverage

Single State (or Transition) Path Coverage

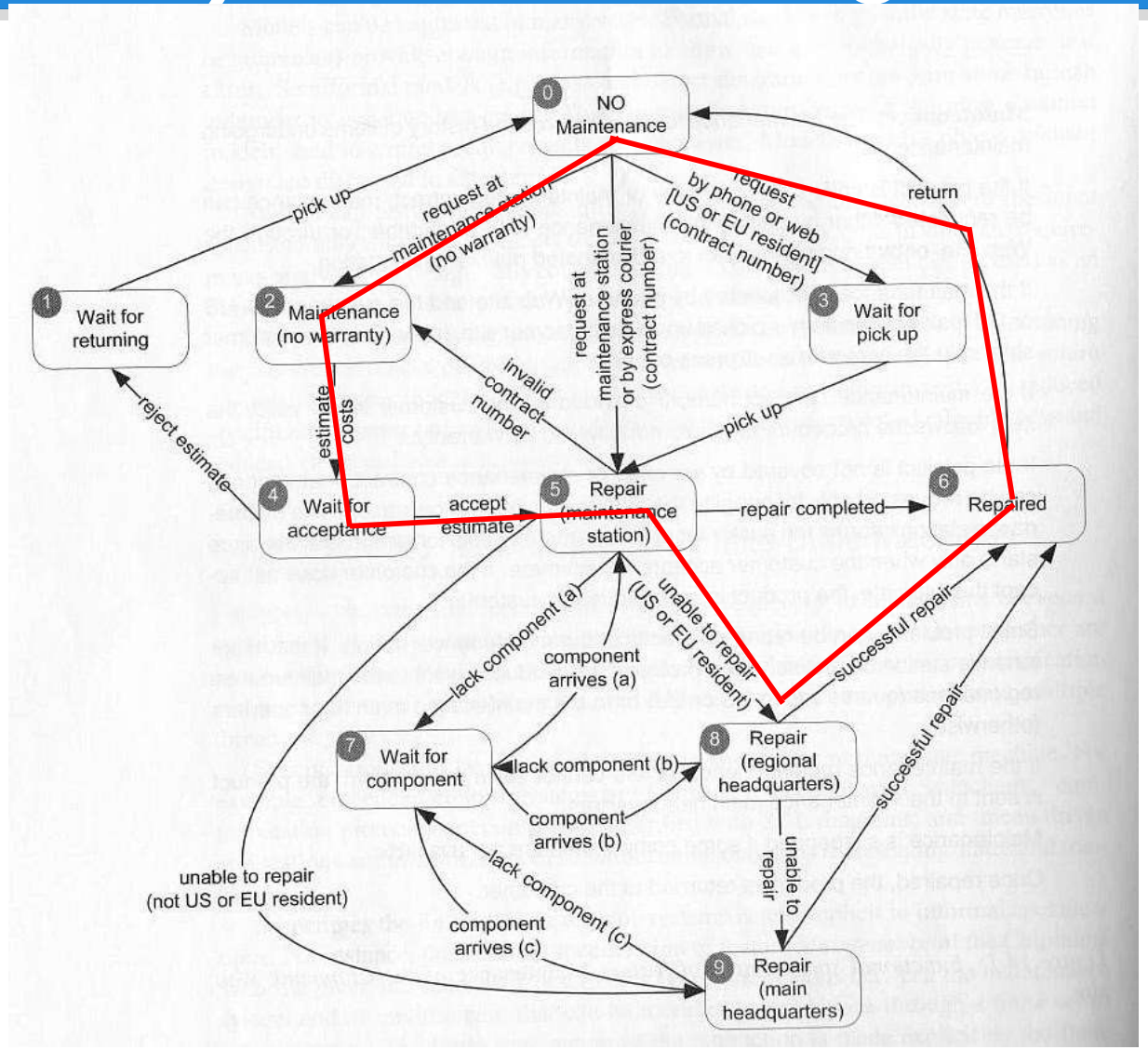
- Each subpath that traverses a state (or transition) **at most once** must be exercised.



Single State (Transition) Path Coverage

Single State (or Transition) Path Coverage

- Each subpath that traverses a state (or transition) **at most once** must be exercised.



We Have Learned

- Models can be used to systematically create tests.
  - Exercises stateful behavior of a class or functionality.
  - Maps well to requirements.
- State machines model expected behavior.
  - Cover states, transitions, non-looping paths, loops.
  - Can also verify properties over models as part of verification (next class).

Next Time

- Finite State Verification
  - Optional Reading - Pezze and Young, Chapter 8

Thank You