Activity Diagrams

- Describe the behavior of a system in terms of activities
- Represent the sequencing and coordination of actions or steps, similar to a control flow graph.

- Activity: Rounded rectangles represent actions called activities.
- Edges between activities represent control flow.
  - branching, looping, concurrency
- Activity diagrams can be hierarchical:
  - A given activity in a rounded rectangle could be further detailed in its own separate activity diagram.

Activity Diagrams: Branching

- Decisions (branches, alternates)
  - Branch Node: diamond with one incoming arrow two or more outgoing arrows.
  - Outgoing edges are labeled with guards (conditions in square brackets) that select that arrow when the condition is true.
  - [else] can be used as a guard.
  - Merge nodes (diamond with many incoming, one outgoing arrow) to mark the end of the branching.
  - The diamonds are sometimes omitted, but should be included for clarity.
Decision in the Handle Incident process.

Activity Diagrams: Concurrency

- Fork nodes and Join nodes (concurrency)
  ✤ The fork is a line with one incoming edge and several outgoing edges.
  ✤ Fork: denotes splitting control into multiple threads, representing the fact that each outgoing edge can be done in parallel.
  ✤ The join is a line with many incoming edges and one outgoing edge.
  ✤ Join: denotes synchronizing threads back into one (waiting until all of the incoming activities are completed before moving forward).
  ✤ Fork and Join denote activities that may be done in any order (they are not required to be done concurrently).

Concurrency in incident management process.

Activity Diagrams: swimlanes

- Swimlanes (activity partitions)
  ✤ Rectangles enclosing a group of activities
  ✤ Used to denotes responsibilities of objects or actors that carry out the activities in the given rectangle.
  ✤ Edges may cross swimlane boundaries
Swimlanes in incident management process.

State diagrams

- Describe the dynamic behavior of an individual object (or subsystem).
- A state diagram describes the sequence of states an object goes through in response to external events
  - A graph: states are nodes, transitions are directed edges
- Transitions from one state to another occur as a result of external events

When and how to use Activity Diagrams

- When developing use cases
  - Activity diagrams are good at capturing business (and other) processes (also called workflows).
- During Object-Oriented design
  - Deciding what objects perform which activities (once you already have an activity diagram).
- When designing complicated operations/methods.
  - Use to model the control flow through a single method (like a flowchart or control flow diagram).
- When dealing with multithreaded applications.

State diagram for the watch display

- Edges are labeled with the event that triggers them
- Small black circle: start state
- Small black circle inside another circle: finish state
States

- A state is (often) represented as a value of an attribute of an object that is changed by an external event.
  - An Incident can exist in four states: Active, Inactive, Closed and Archived
- A state is a node in the graph
- The node can specify some activity that is performed when the node is entered.
  - This is denoted inside a component of the state using the following syntax, where “activity” is replaced by a description of that activity:
    ```
    do/activity
    ```
  - The activity could be interrupted by some external event.

Transitions

- A transition represents a change of state triggered by events, conditions, or time.
  - Transitions are directed edges in the graph
  - Edges are labelled by the event causing the transition:
    ```
    Event [Guard] / Action
    ```
    Each part is optional, Guard must be true to transition, Action is performed when transition occurs.
  - If the Event is omitted, the transition occurs as soon as the activity in the given source state is completed.
  - An event can represent the passing of a period of time:
    ```
    after 20 minutes
    ```

State Machine diagram for 2Bwatch

State diagram with nested state and guards
When and how to use State Diagrams

- When designing a class that has an attribute that responds to external events (and determining which state the object is in is not trivial)
  - Use the state diagram to document the transitioning behavior
- During testing
  - If you have a state diagram, you can develop tests that perform a sequence of events and then verify that the object is in the correct state with respect to the diagram
  - If your object (or system) does not have an attribute that responds to external events, do not use state diagrams.
- User Interface objects often have behavior that is useful to depict with a state diagram