
# Develop Use Case

**Munawar, PhD**

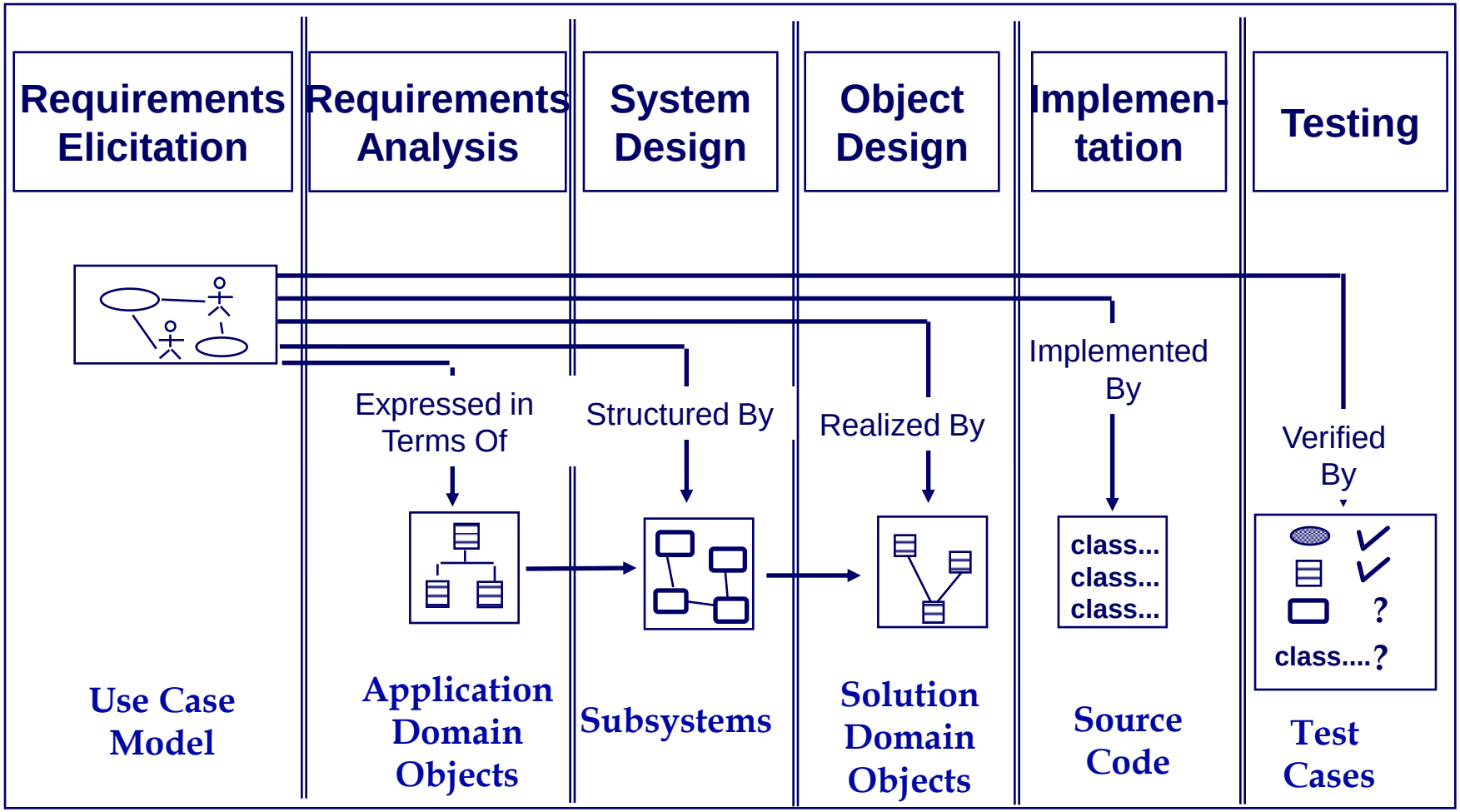
# Use Cases

- **A use case should describe the user's interaction with the system ...**
  - Not the computations the system performs
- **In general, a use case should cover the full sequence of steps from the beginning of a task until the en**
- **A use case should only include actions in which the actor interacts with the computer**
  - Some views differ on this one!!!

# Use Cases – Abstraction Level

- **A use case should be written so as to be as independent as possible from any particular implementation / user interface design**
- **Essential use cases (Constantine & Lockwood)**
  - Abstract, technology free, implementation independent
  - Defined at earlier stages
  - e.g., customer identifies herself
- **Concrete use cases**
  - Technology/user interface dependent
  - e.g., customer inserts a card, customer types a PIN

# Use Case-Driven Software Lifecycle Activities (1)

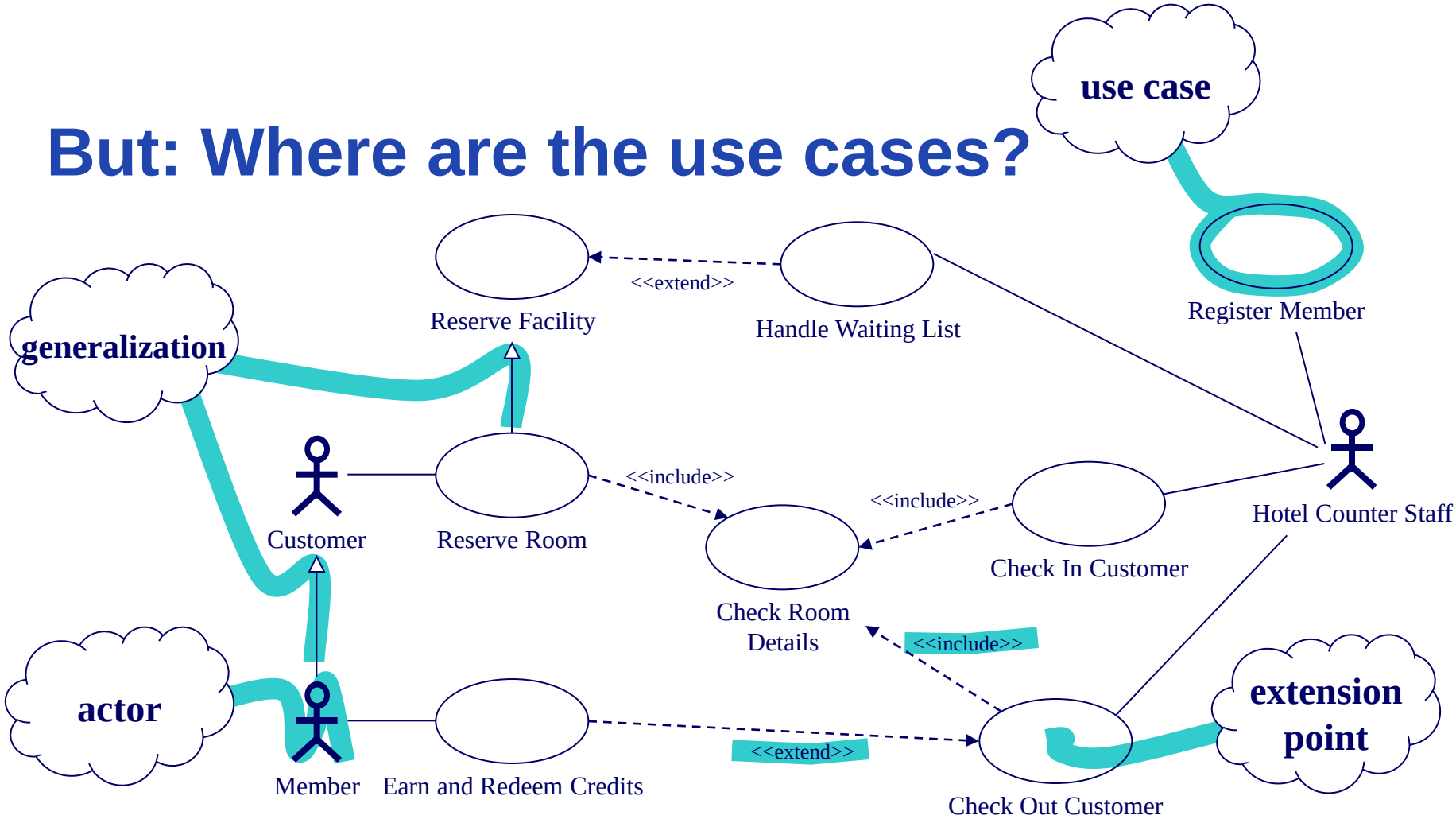


# Use Case-Driven Software Lifecycle Activities (2)

- **Scenarios guide elicitation, analysis, design, and testing**
  - There are many scenario-based approaches
  - E.g., XP employs user stories (scenarios) to directly generate tests that will guide software design and verification
- **Developers are often unable to speak directly to users**
- **Scenarios provide a good enough approximation of the "voice of the user"**

# Use Case Modeling with UML

## But: Where are the use cases?



# Use Case Diagrams

- **To define system boundary (subject), actors, and use cases**
  - Subject could be: a physical system, a component, a subsystem, a class
- **To structure and relate use cases**
  - Associate actors with use cases
  - Include relation
  - Extend relation
  - Generalization (of actors and use cases)

# Use Case Diagrams – <<include>> (Inclusions)

- **Inclusions allow one to express commonality between several different use cases**
- **Inclusions are included in other use cases**
  - Even very different use cases can share a sequence of actions (reuse)
  - Enable you to avoid repeating details in many use cases (consistency)
- **An inclusion represents the execution of a lower-level task with a lower-level goal (→ decomposition of complex tasks)**
- **Base use case references the included use case as needed**
- **Base use case cannot exist without the included use case**
- **When included use case is done, control returns to base use case**
- **Disadvantage: have to look in multiple places to understand use case**

# Use Case Diagrams – <<extend>> (Extensions)

- **Used to make optional interactions explicit or to handle exceptional cases**
- **By creating separate use case extensions, the description of the basic use case remains simple**
  - Note: the base use case can still be executed without the use case extension
- **Extension points must be created explicitly in the base use case**
- **Use sparingly: there is disagreement over the semantics**

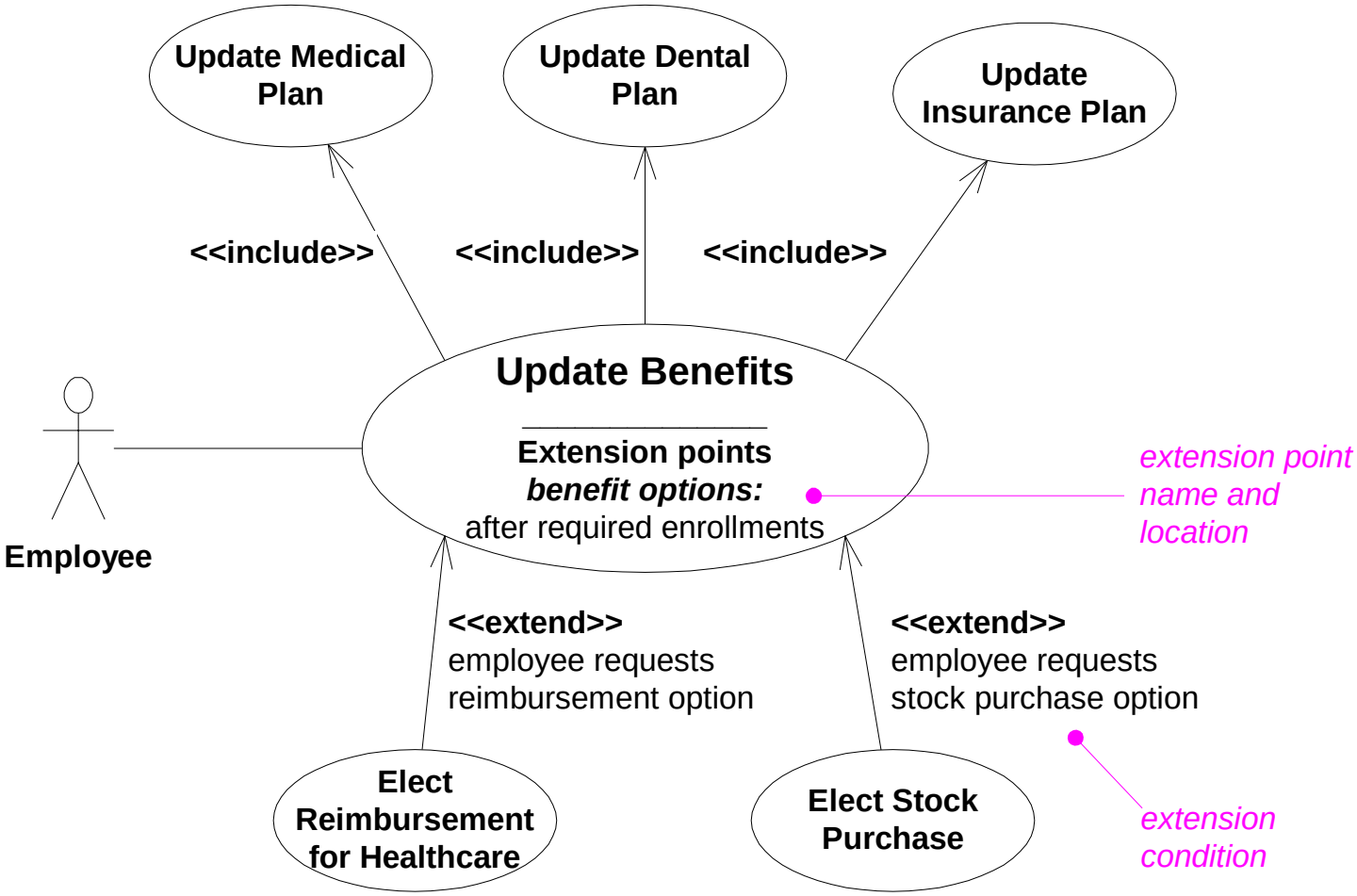
# Use Case Diagrams – Generalizations

- **Much like super-classes in a class diagram**
  - Need to satisfy “is-a” relation
- **Applies to use cases and to actors**
  - A generalized use case represents several similar use cases
  - One or more specializations provide details of the similar use cases
  - Inheriting use case can replace steps of inherited use case
  - Particularly useful for actors (clearer here, unlike use case generalization where the semantics are unclear – use generalization of use cases with caution)

# Use Case Diagrams – Generalizations

- **Much like super-classes in a class diagram**
  - Need to satisfy “is-a” relation
- **Applies to use cases and to actors**
  - A generalized use case represents several similar use cases
  - One or more specializations provide details of the similar use cases
  - Inheriting use case can replace steps of inherited use case
  - Particularly useful for actors (clearer here, unlike use case generalization where the semantics are unclear – use generalization of use cases with caution)

# Example: HR System



# Benefits of Use Case-Based Software Development

- **They can help to define the scope of the system**
- **They are often used to plan the development process**
- **They are used to both develop and validate the requirements**
  - Simple, easy to create
  - All stakeholders understand them
  - Often reflect user's essential requirements
  - Separates normal behavior from exceptional behavior
- **They can form the basis for the definition of test cases**
- **They can be used to structure user manuals**

# Lab Activities

- Based on your scenario in previous meeting, describe your use case diagram for your scenario !!!


# Thank You !

**Munawar, PhD**