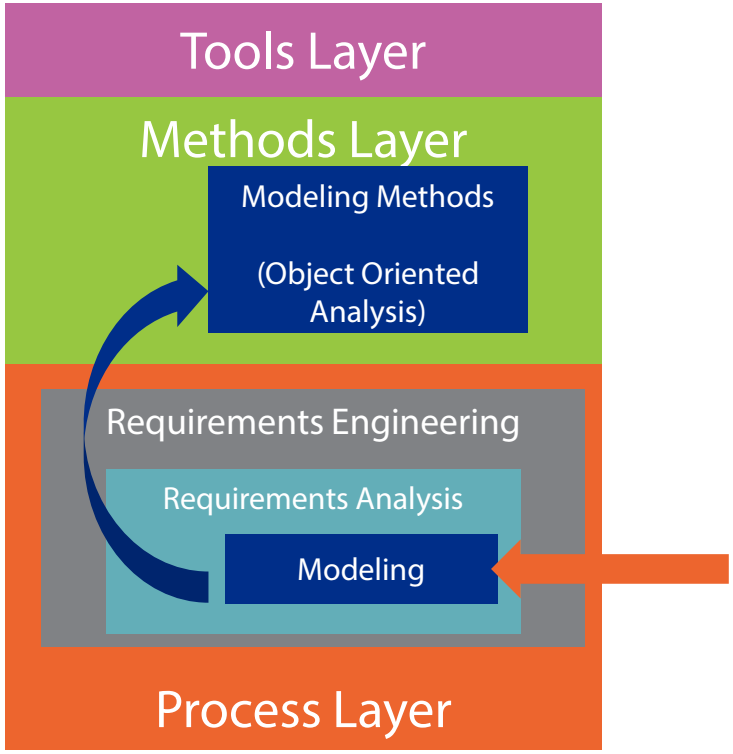
Requirements Modeling – Object Oriented Analysis

Mohamad Halabi
@mohamadhalabi

pluralsight
hardcore dev and IT training

Introduction



Structured Analysis Criticism

1. **Analysis is centered on data**
    a) Data is not good at modeling a system's building blocks
2. **Behavioral modeling is a second-thought**
    a) CFDs are extensions
    b) Still STDs and CFDs do not provide rich behavioral modeling

The Move Towards Modern Methods

- **We live in a world of objects!**
- **We think of systems in terms of objects rather than data**
- **Objects encapsulate data (attributes) and behavior (operations)**
- **Object Oriented Analysis methods are centered on objects**
  - Modeling is based on objects, their relationships, and their interactions
  - This covers both the static and dynamic (behavioral) aspects

Various Approaches to Object-Oriented Analysis and Design

- Object Oriented Analysis is typically followed by Object Oriented Design
- Method widely known as Object Oriented Analysis and Design (OOAD)
- No one standard approach to OOAD
- All approaches do share the same principles

Various Approaches to Object-Oriented Analysis and Design

- **We'll cover a generic approach based on 4+1 View Model**
- **RUP's approach will be briefly described**
- **All Object Oriented Analysis methods share the same principles:**
  - Functionality is modeled through Use Cases
  - Classes and relationships model the static aspect
    - What are the system parts that enable Use Case functionalities?
  - Class interactions model the dynamic (behavioral) aspect
    - What are the relationships between parts that enable Use Case functionalities?

The Role of UML in Object Oriented Analysis

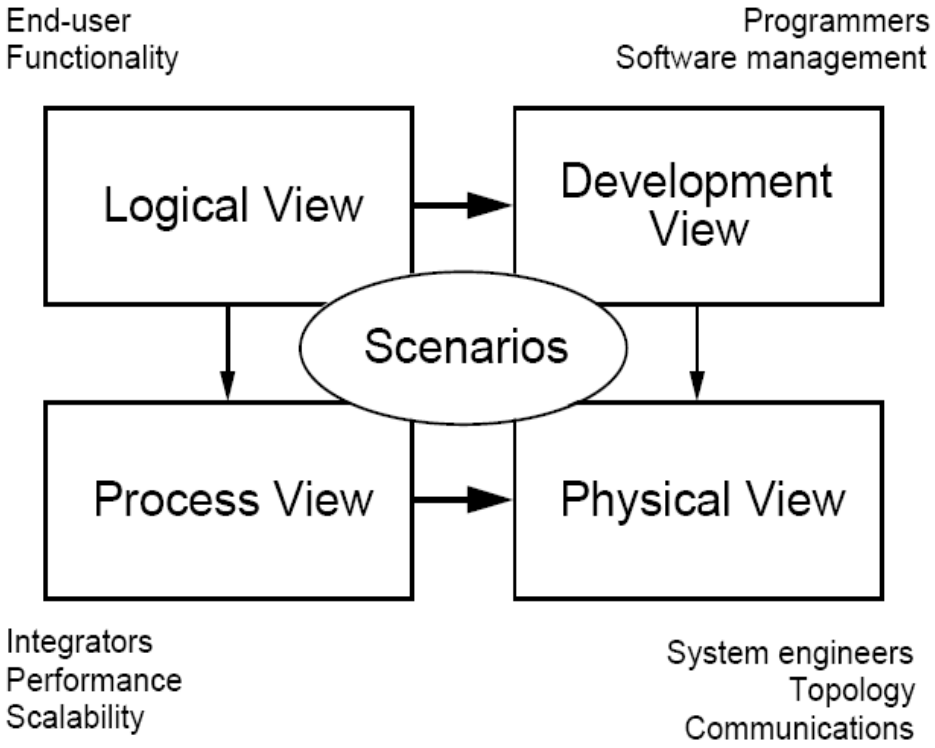
- **Unified Modeling Language (UML) is controlled by an open standards group (OMG)**
- **UML is created by OOAD thought leaders**
  - So UML became the standard modeling approach for all OOAD methods

The Role of UML in Object Oriented Analysis

- **What UML diagrams to use for Object Oriented Analysis?**
  - We need to know the system views (perspectives) to model
- **4+1 View Model partitions UML diagrams into various views**

The 4+1 View Model

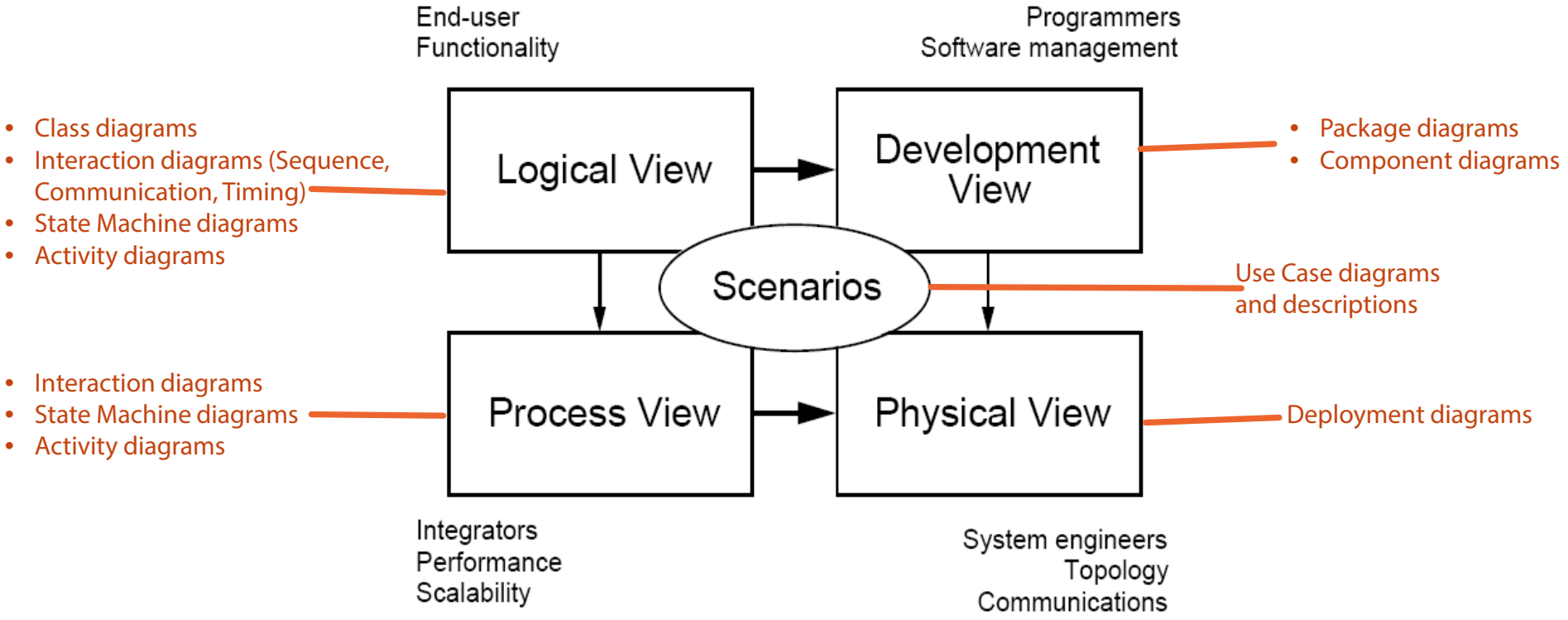
- The 4+1 View Model of Software Architecture
- A software's architecture is divided into four views
- Each view models the system's architecture from a different perspective



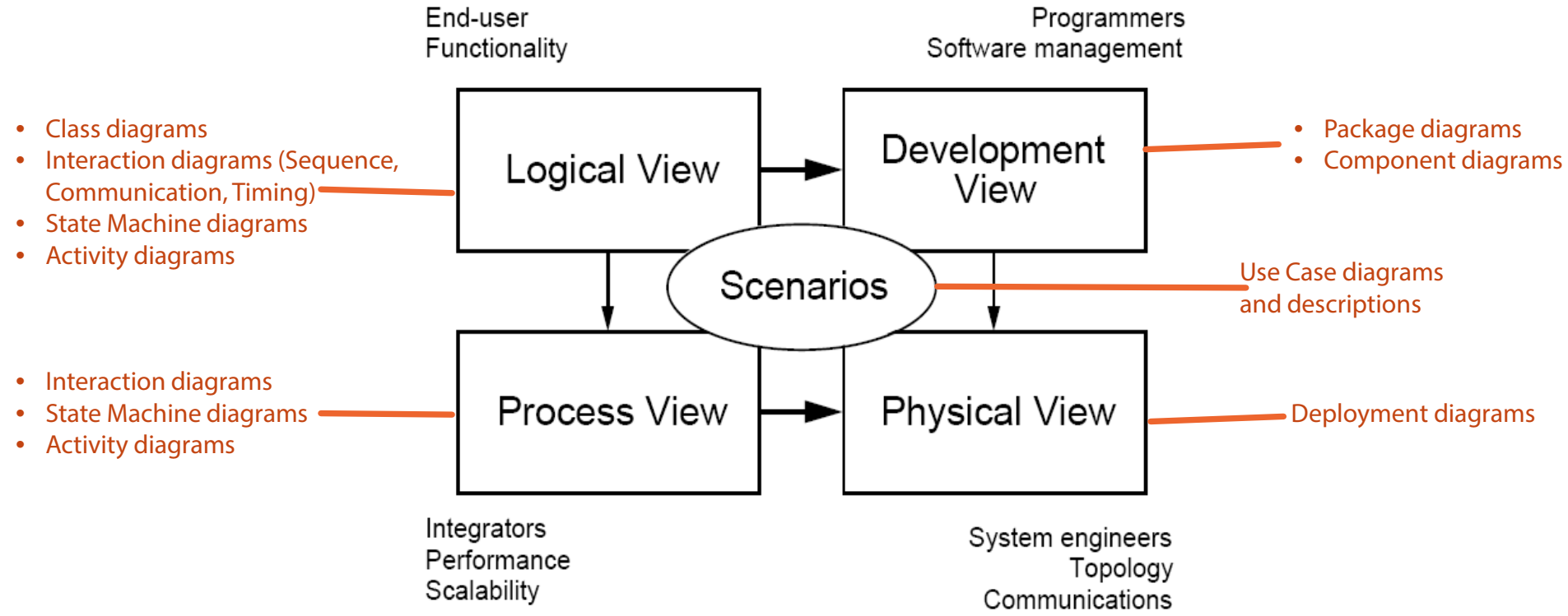
Source: Philippe Kruchten – IEEE paper

UML Mappings

- Originally, the 4+1 View Model has no mention of UML
- UML mappings were later proposed
- There is no one agreed-upon standard for mapping



Analysis Views



- **Analysis: Scenarios and Logical views**
- **Design: Process, Development, and Physical views**

How Come an Architectural Model Is Used for Analysis?

- **The 4+1 View Model is for system architecture**
- **…However, we have just used the Scenarios and Logical views for analysis modeling**
- **Q: How come? Doesn't architecture come after analysis?**
  - In the next module I will talk about analysis vs. design
- **A: Models are all about abstraction:**
  - Same diagram can be used to understand the problem (i.e. analysis) or the solution (i.e. design) depending on its level of detail

Static/Dynamic Modeling

Static Models

The parts that make up the system

The relationships between these parts

In Structured Analysis, ERDs represent static modeling

Dynamic (Behavioral) Models

The interaction between parts

The change of state of the parts

In Structured Analysis, STDs and CFDs represent dynamic modeling

What About 4+1 Models?

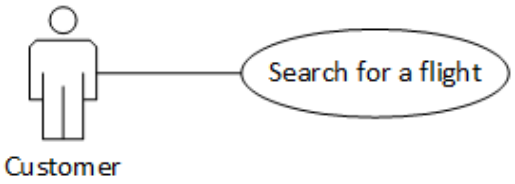
- **Scenarios view:**
  - Use Case diagrams and descriptions are static models
- **Logical view:**
  - Class diagrams are static models
  - Interaction (Sequence, Communication, Timing), State Machine, and Activity diagrams are dynamic models
- **Process view:**
  - Interaction, State Machine, and Activity diagrams are dynamic models
- **Development view:**
  - Component and Package diagrams are static models
- **Physical view:**
  - Deployment diagrams are static models

Use Cases

- A Use Case is a set of scenarios that describes system's interaction with actors
- Actors are black boxes: people, applications, or any entity
- Each Use Case captures a system's functionality
- In the 4+1 View Model, Use Case view drives all other views

Notation

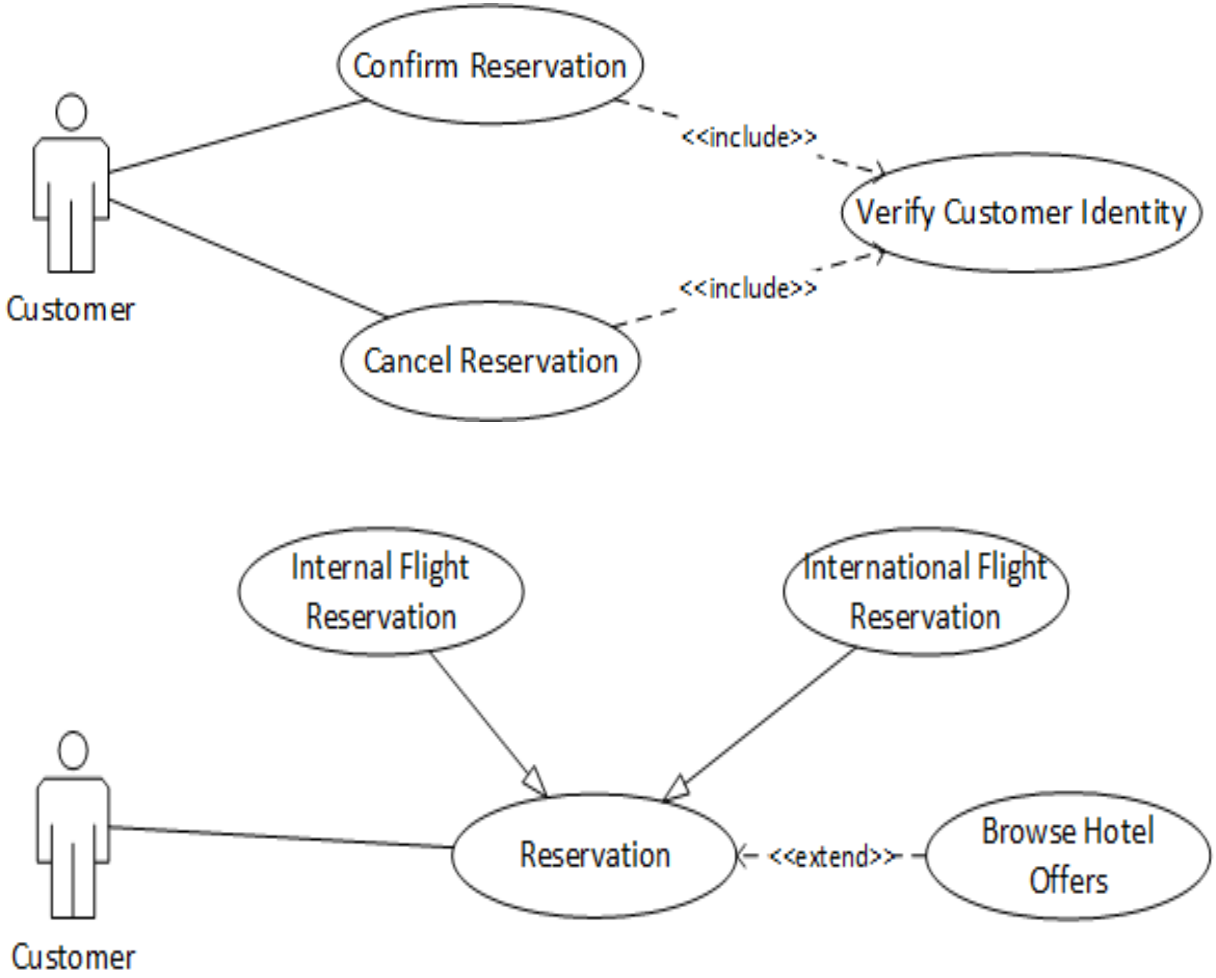
- **Use Case diagram:**



- **Use Case description:**

Use Case: Search for a flight	
Use Case ID	U-004
Related Requirements	R1, R23
Description	An online customer searches for a flight
Preconditions	Customer has internet access
Success Condition	Search results are displayed
Failed Condition	Search transaction is not completed
Actors	Customer
Flow of Activities	1. Customer visits the search page of the system 2. Customer supplies the mandatory search information 3. Customer submits search form 4. The system verifies supplied information 5. System performs search against flights database 6. A grid of the search results is displayed to the customer
Alternate Flow	4. The system fails to verify supplied information 1.1. System displays a message to customer showing unverifiable entries 1.2. Customer fixes issues and submits search form

Relationships

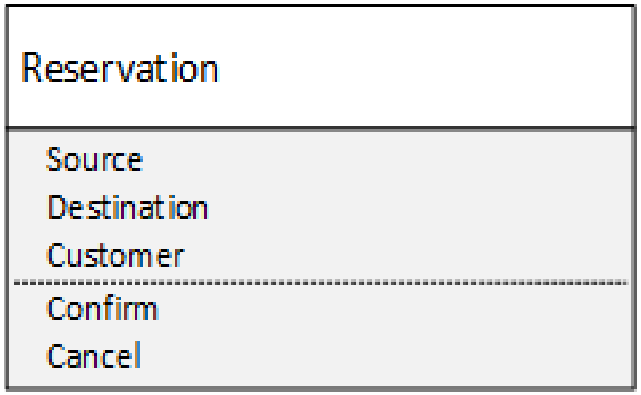


Class Diagrams

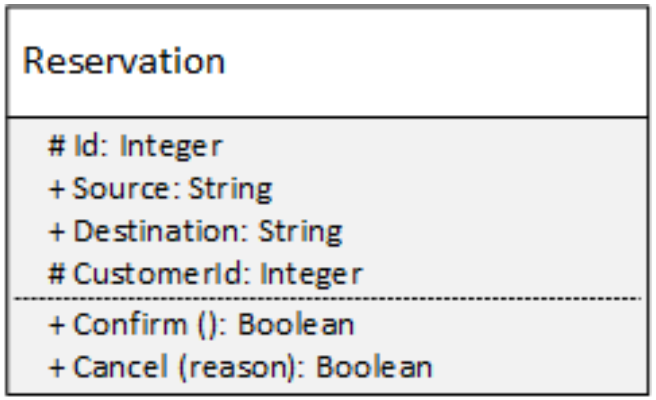
- **Use Cases describe a system's functionality**
- **…Classes describe what system parts (objects) achieve this functionality**
- **Warning: We're still talking about the "what", not the "how"**
  - In analysis, Class diagrams help discover the parts and their relationships at a conceptual level
  - In design, conceptual classes are refined to show details (ex: members visibility and types and operations parameters)

Analysis vs. Design Class Diagram

Analysis Class Diagram



Design Class Diagram



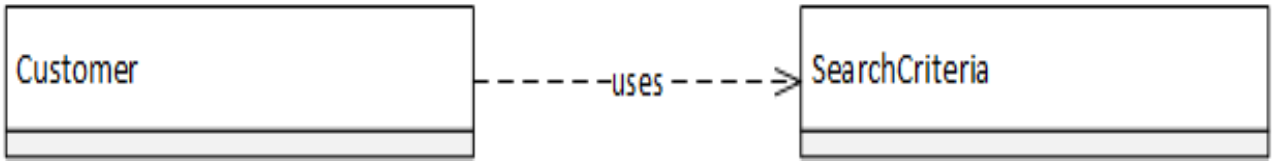
- **Some argue that Analysis classes should only list names:**
- **Disagree! More into the analyst role that just listing the classes**
  - Without conceptually modeling members, Interaction and Activity diagrams won't be modeled properly

Candidate Classes

- **An analysis class:**
  - Must help the system achieve a functionality modeled in Use Cases
  - Must hold information (i.e. attributes) needed by problem domain analysis
  - Must have operations that model problem domain functionality
    - They typically change attribute values
  - Collaborates with other classes to achieve a certain functionality
  - Might represent internal or external entity
    - External entities guide the design of interface design

Relationship: Dependency

- A class uses the functionality of another class



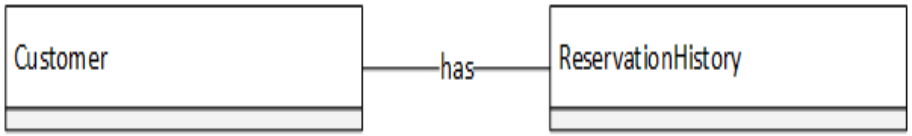
- Code representation:

```
public class Customer
{
    void SearchForFlight(SearchCriteria crt)
    {
        //use crt to perform search
    }
```

Relationships: Association and Aggregation

Association

A class has a reference to another class

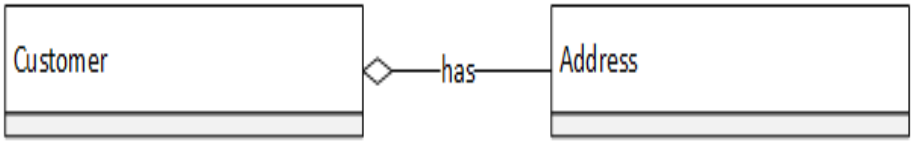


Weaker relationship

Customer cannot delete his reservation history
- History controlled by the system

Aggregation

A class has a reference to another class



More powerful relationship

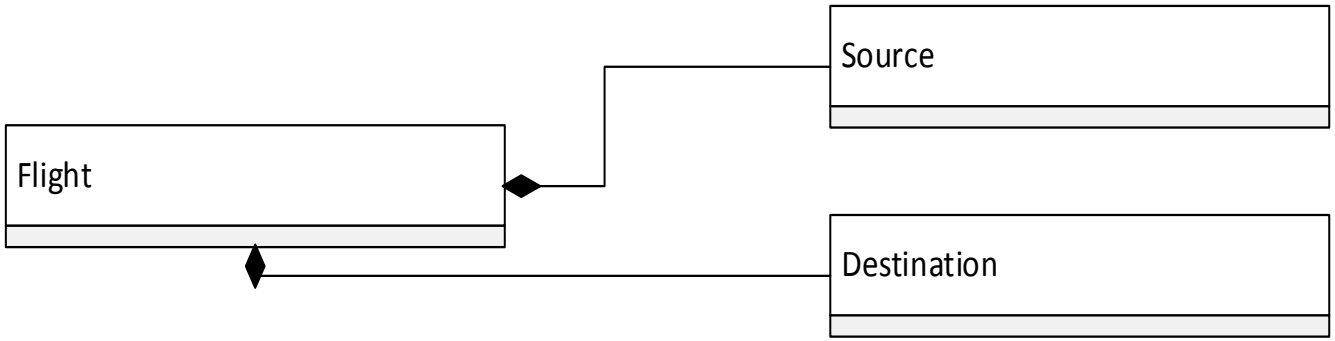
Customer can delete her address
- Address controlled by the customer

- **Code representation:**

```
public class Customer
{
    ReservationHistory resHistory;
    Address[] adrs;
```

Relationship: Composition

- Strongest type
- Models the constituent parts of a class



- Code representation:

```
public class Flight
{
    Source src;
    Destination dest;
```

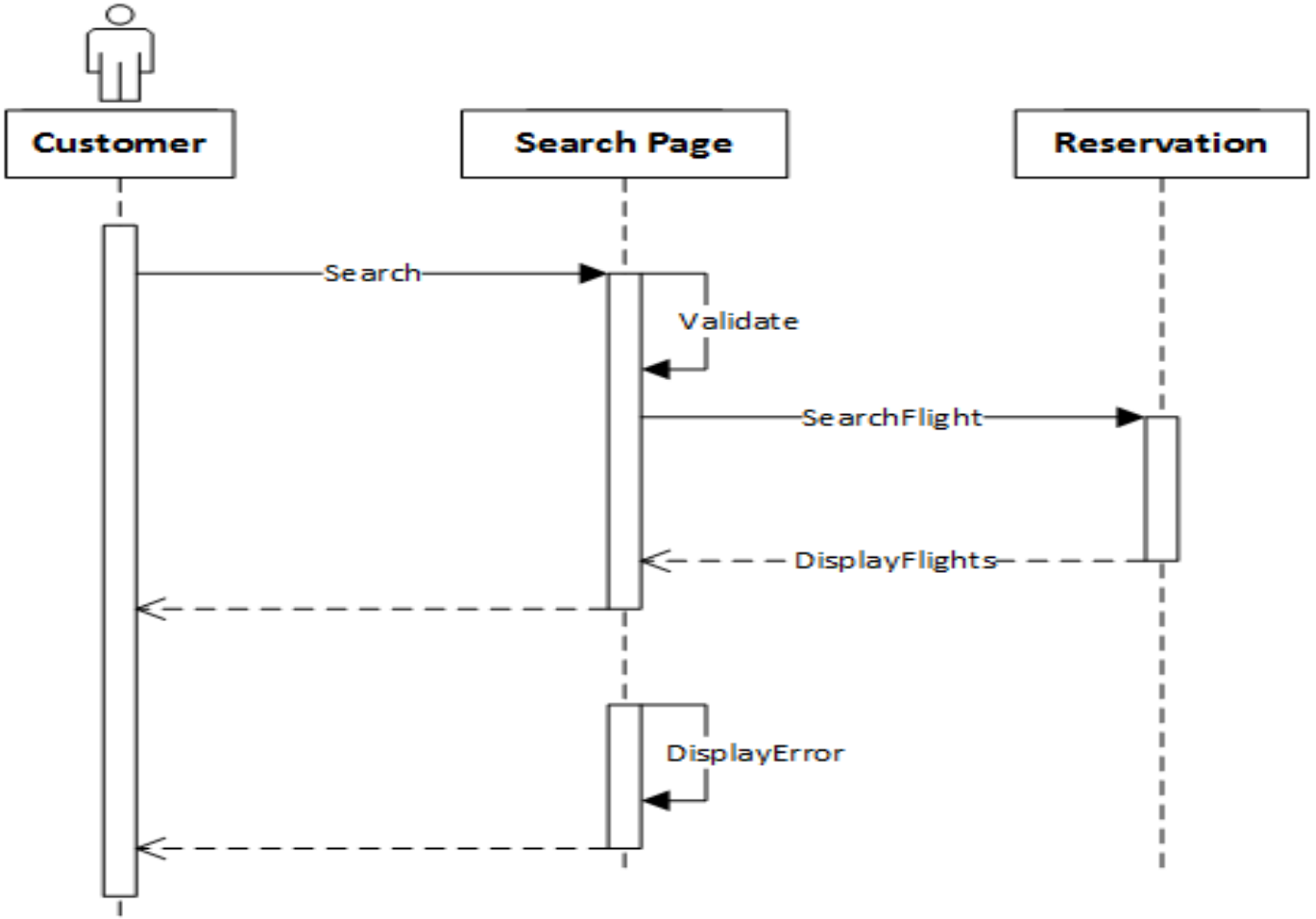
Interaction Diagrams

- **Use Cases describe functionality**
- **Classes show what are the parts that enable this functionality**
- **Interaction diagrams show the interaction between these parts**
- **Types of Interaction diagrams:**
  - Sequence
  - Communication
  - Timing

Sequence Diagrams

- **Capture the order of interactions between system parts**
- **4 key parts:**
  - Participant is an object or an entity
  - Participants communicate via messages or signals
  - Horizontal axis shows current active (i.e. doing something) participant
  - Vertical axis indicates time order (note: not duration)

Sequence Diagram



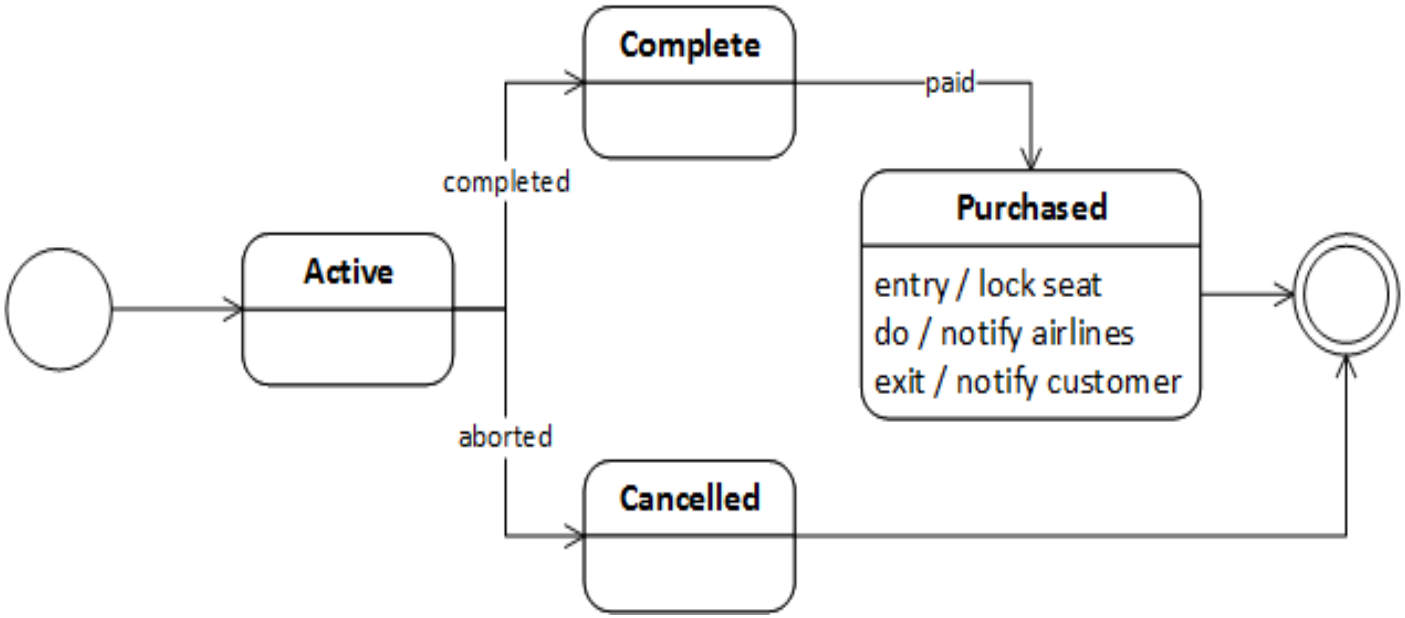
Communication and Timing Diagrams

- **Communication diagrams are similar to Sequence diagrams**
  - They do not show the order of interactions
- **Timing diagrams model detailed timing information**
  - Ex: An interaction must take no longer than 5 seconds
  - Ex: An interaction must take no more than half the time of other interaction

State Machine Diagrams

- **In Structured Analysis, State Transition Diagrams (STDs) model system's change of states**
- **UML's State Machine diagrams (also called State or Statechart diagrams) are variation of STDs**
  - State Machine diagrams show object state changes and events causing the change

State Machine Diagram: Reservation Object

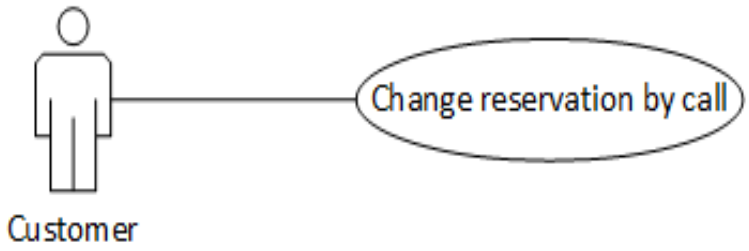


Activity Diagrams

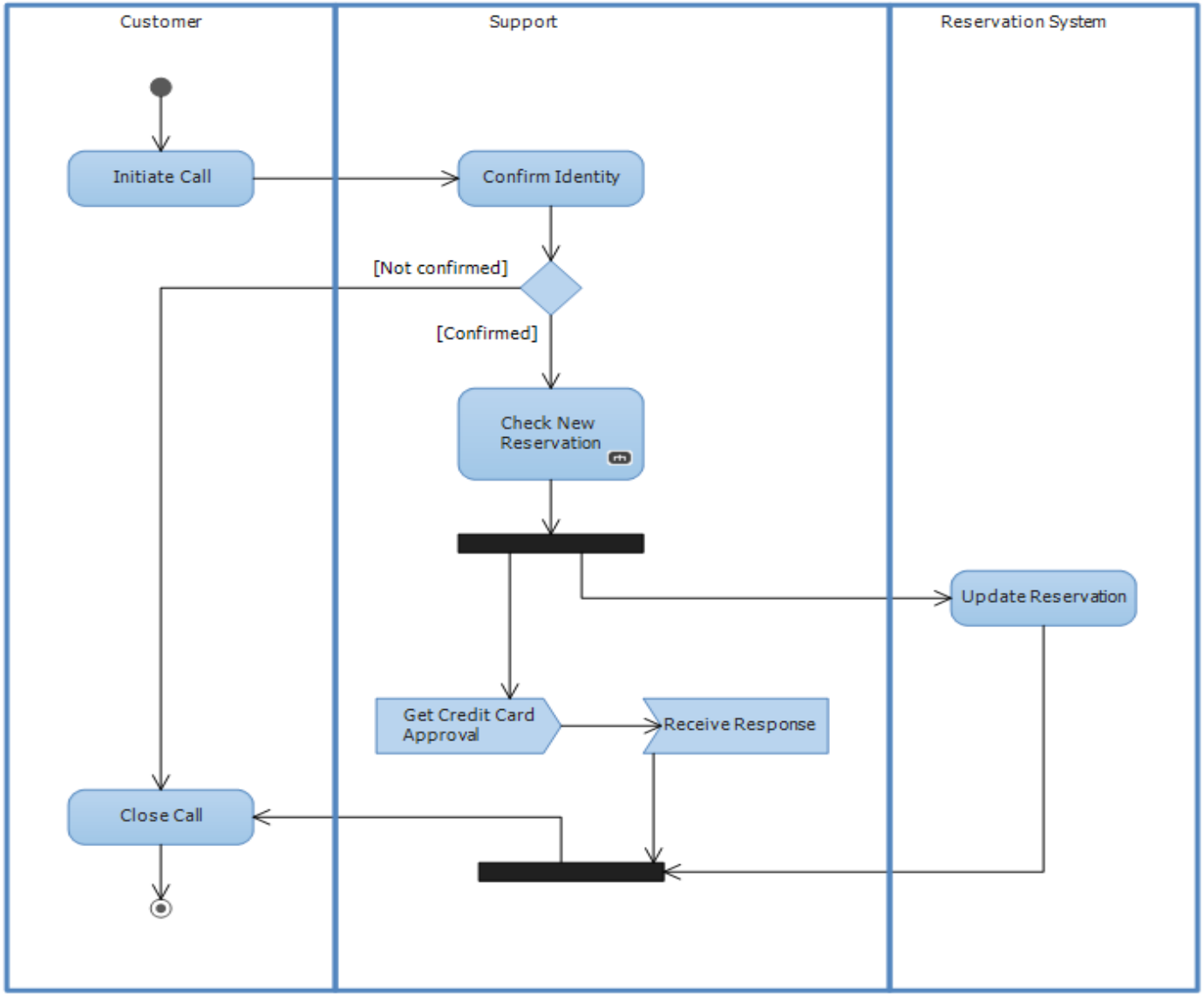
- **Show how high-level actions are chained into business processes**
- **They show:**
  - Actions
  - Decision paths
  - Parallel processing
  - Swimlanes
- **Model in more details Use Cases functionality**

Activity Diagram Example

- **Consider the following Use Case:**

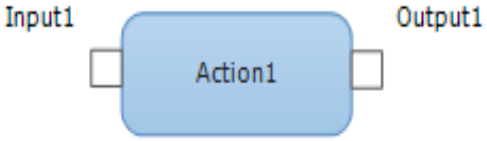


Activity Diagram Example



Other Details

Objects flowing between actions



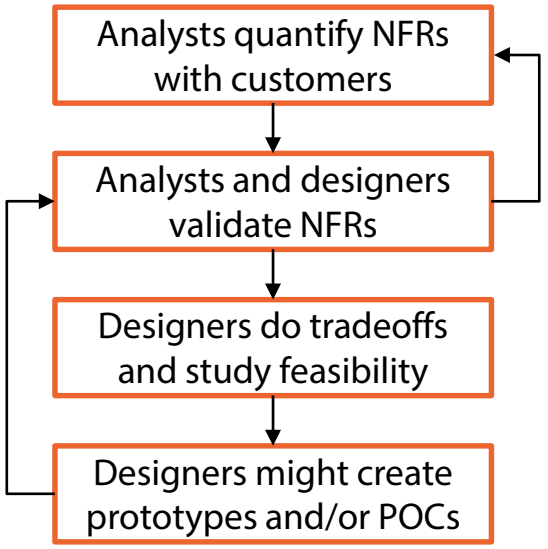
Inputs/outputs to actions

- **These are typically left to the Design phase**

What About Modeling Non-functional Requirements?

- **Structured Analysis and Object Oriented Analysis model functional requirements**
- **What about non-functional requirements (NFRs)?**
- **NFRs are cross-cutting:**
  - Not specific to a single Use Case
  - Span multiple parts and processes
- **In analysis NFRs are modeled in clear text or tabular forms**
  - The important thing is that they are measurable
- **NFRs become the focus of Design phase as a solution is sought**

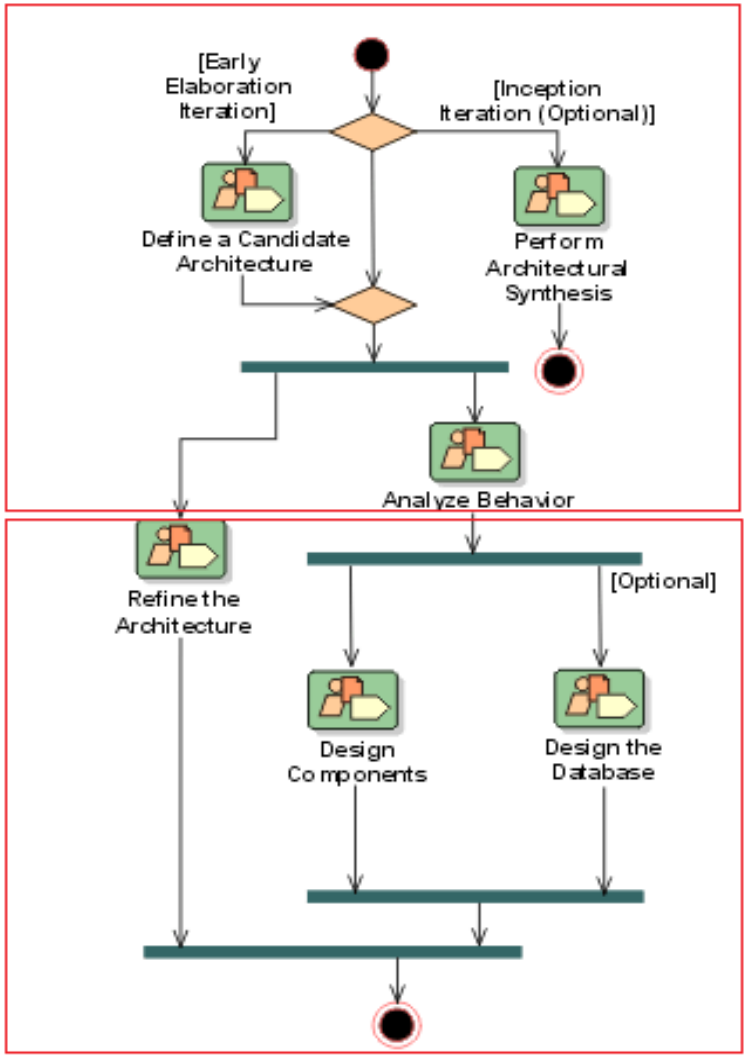
NFRs in the Analysis Phase



- **Usability requirement: "*Users must be able to retrieve a customer's activity log in less than 5 seconds*"**
- **Performance requirement : "*The response time of search transactions must not exceed 3 s under peak load of 1000 users*"**

RUP OOAD

- **IBM's RUP has its own specialization of OOAD (based on 4+1 Model)**



Analysis and Design are part of a single discipline

Collaboration between analysts, designers, and architects

Source: IBM Developer

RUP OOAD

- **A Use Case model is generated in the Requirements discipline**

Analysis

Architectural Analysis

Define high-level candidate architecture

Define architecture patterns and mechanisms
- Mechanisms include NFRs solution

Create a Use Case Realization for each Use Case
- A Realization describes a Use Case in terms of collaborating objects

Use Case Analysis

One iteration per Realization

Use Case descriptions are refined

For each realization define Analysis Classes and distribute behavior
- Analysis Classes tackle NFRs

Map Analysis Mechanisms to Analysis Classes

Design

Identify:
- System
- Subsystems
- Classes
- Packages

Identify Design Mechanisms

Create Process and Deployment views

Common Principles of OOAD Methods

- **OOAD methods share common principles**
  - Analysis and design are centered on objects
  - UML diagrams model the system from different perspectives

What About Business Process Analysis?

- **Both Structured Analysis (SA) and Object Oriented Analysis (OOA) provide some form of process modeling**
  - STDs and CFDs model control flow in SA
  - State Machines and Interaction diagrams model control flow in OOA
  - Activity diagrams is a powerful process modeling notation in OOA
- **However, neither SA nor OOA are centered on processes**
  - SA is centered on data
  - OOA is centered on objects

Business Process Analysis (BPA)

- **BPA is another analysis method that is centered on business process improvement**
  - Reduce cost and waste
  - Make efficient use of resources
- **Process modeling is centered on:**
  - Activities
  - Relationships between activities
  - Resources
  - Events
- **Business Process Modeling Notation (BPMN)**

Business Process Analysis (BPA)

- **BPA leads to different implementation style**
  - OOA → Object Oriented programming
  - BPA → WF applications or Business Process Management (BPM) engines

More Resources

- **More about the 4+1 View Model:**
  - *Architectural Blueprints—The "4+1" View Model of Software Architecture* by Philippe Kruchten (IEEE, November 1995)
- **More about UML:**
  - Search "UML" in Pluralsight library
- **More about OOAD in RUP:**
  - *Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and Iterative Development* by Craig Larman

Summary

- **OOA is typically followed by OOD – the method is called OOAD**
- **Perspectives are divided based on the 4+1 View Model**
  - UML notations are mapped into these perspectives
- **Modeling notations are static and dynamic (behavioral)**
  - Static notations model system parts and their relationships
  - Dynamic notations model interactions of system parts
- **Static: Use Cases (Scenarios view) and Class diagrams (Logical view)**
  - Use Cases model system functionalities
  - Class diagrams model system parts and relationships
- **Dynamic: Interaction, State Machine, and Activity diagrams (Logical View)**
  - Model how system parts interact with each other

What's Next?