

# Revision

Lecture #11  
Software Engineering and  
Project Management

Instructed by Steven Choy on Dec 11, 2006



1

## Revisions on

- Chapter 1: Introduction to Software Engineering
- Chapter 2: Modeling with UML
- Chapter 4: Requirements Elicitation
- Chapter 5: Requirements Analysis



2

## Chapter 1: Introduction to Software Engineering



3

## What is Software Engineering?

- The objective of software engineering activities is to develop quality software on time and within budget
- Complexity and change cause SE failure
  - Break down complexity
  - Manage changes



4

## Aspects of Software Engineering



- Four aspects of software engineering (ch.1 section 1.2)
  - 1. Modeling
  - 2. Problem solving
  - 3. Knowledge acquisition
  - 4. Rationale

5

## 1. Modeling



- Also known as “Abstraction”
- Consist of application domain model and solution domain model
- Both model could be built using object-oriented methods

6

## 2. Problem Solving



- Formulate the problem
- Analyze the problem
- Search for solutions
- Decide on the appropriate solution
- Specify the solution

7

## 3. Knowledge acquisition



- Understand the client requirements and the business environment that the system will operate.
- Keep learning new development tools (e.g. programming languages and environment) and methodologies
- A non-linear process

8

## 4. Rationale

- The reason behind a design decision
- Keep record of rationale of the system during development and changes.

9

## Software Engineering Activities

- SE development activities (ch.1 section 1.4, Figure 1-2 on p.17)

Activities	Description
Requirement elicitation	Purpose and requirement of the system
Analysis	System models with objects (understandable by clients)
System design	Completed system models
Object design	Define object details
Implementation	Write the program code for each objects
Testing	Determine conformance to the requirements

10

## SE Management Activities

- SE management activities (ch.1 section 1.5)

Activities	Description
Communication	Understanding and collaboration between project participants
Rationale management	Keep track of decision justifications
Software configuration management	Software versions tracking (code and documentation)
Project management	Activities co-ordinations
Software life cycle	Provides ordering for carry out complex software development process

11

## Additional SE Concepts

- Addition SE concepts (ch.1 section 1.3)
  - Participants and roles
    - Client, user, developer, etc.
  - Work products
    - Specification, manuals, reports, etc.
  - Activities, tasks and resources
    - Requirement elicitation, database, etc.
  - Functional and non-functional requirements
    - System constraint (non-functional)
  - Notation, methods and methodologies

12



## Chapter 2: Modeling with UML

13

## UML Overview

- Stands for **Unified Modeling Language**
- For graphical object-oriented modeling/abstraction
- Accepted by OMG (Object Management Group) as standard modeling notation
- Five most commonly used UML diagrams
  - Use case diagram
  - Class diagram
  - Interaction diagram
  - Statechart diagram
  - Activity diagram





14



## Use Case Diagrams

15

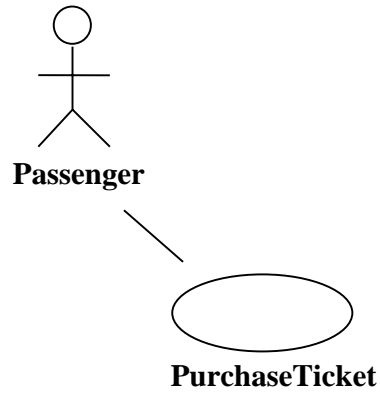
## Use Case Diagram

- Represent **functionality** of the system from user's point of view
- Functional model
- Define boundaries of the system
- Use case description (page 45, figure 2-14) 
- Scenario – instance of use case (page 47, figure 2-15) 
  - Three fields: Name, participating actor instances, flow of events



16

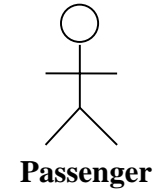
## Use Case Diagrams



- Used during requirements elicitation to represent external behavior
- **Actors** represent roles, that is, a type of user of the system
- **Use cases** represent a sequence of interaction for a type of functionality
- The use case model is the set of all use cases. It is a complete description of the functionality of the system and its environment

17

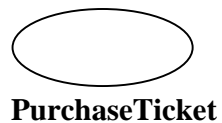
## Actors



- An actor models an external entity which communicates with the system:
  - User
  - External system
  - Physical environment
- An actor has a unique name and an optional description.

18

## Use Case



A use case represents a class of functionality provided by the system as an event flow.

A use case consists of:

- Unique name
- Participating actors
- Entry conditions
- Flow of events
- Exit conditions
- Special requirements

19

## Class Diagrams, Object Diagrams

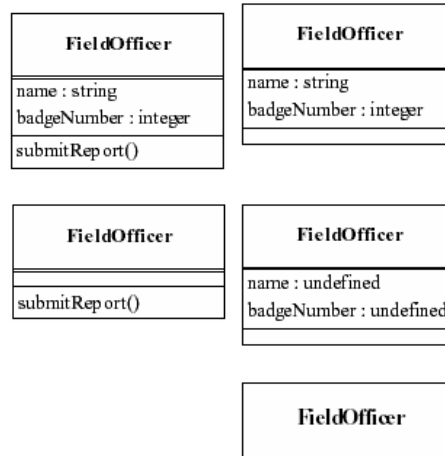


20

## Class Diagram



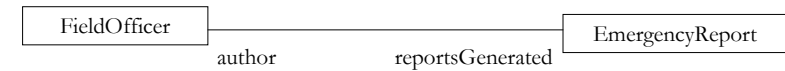
- Class diagram
  - Represent **structure** of a system
  - Three compartment
    - Class name
    - Attributes
    - Operations



## Class Diagram



- Association
  - Relationship between classes
- Role
  - Give meaning to an association

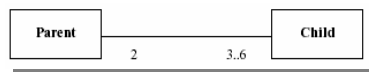
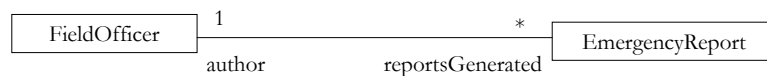


22

## Class Diagram



- Multiplicity
  - Indicates how many objects participate in a given relationship
  - One-to-one, one-to-many, many-to-many
  - 0 1 n 0..n 1..n 0..\* 1..\* \* etc.



## Class Diagram



- Aggregation
  - Captures "Whole/part" relationship



- Plain association is acceptable, but not a complete description then

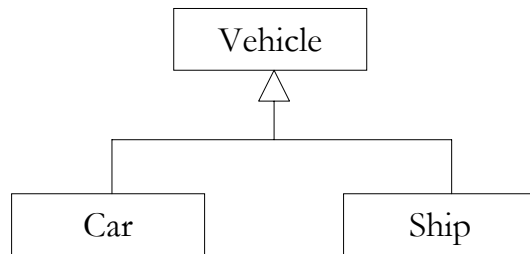


24

## Class Diagram



- Generalization
  - Captures “General/specific” relationship



25

## Class Diagram



- Navigation arrow
  - Associations can be bi-directional or uni-directional
  - Navigation arrow indicates the direction of associations

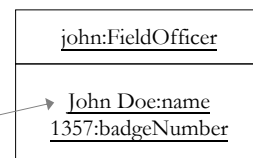


26

## Object Diagram



- Just a modified form a class diagram
- Object – instance of class
- Underline the object name to represent object
  - Example
    - john
    - john:FieldOfficer
    - mikePC:Computer
- Attribute value for an object
  - Also an object
  - Underline it as well



27

## Object Diagram



- Associations (in class diagram) becomes actual links (in object diagram)
- Multiplicity does not apply in object diagram

28

## More on classes and objects



- Abstract class
  - Class which cannot be instantiate
  - Cannot create object using abstract class
- Concrete class
  - Class which can be instantiate
- Method of a class
  - An implementation of an operation
  - Call a method of an object
    - Means sends a message to an object

29

## Interaction Diagram



- Represent system's behavior in terms of interactions among a set of objects.
- Two types of diagrams
  - Sequence diagram
  - Collaboration diagram

30

## Interaction Diagrams



31

## Sequence Diagrams

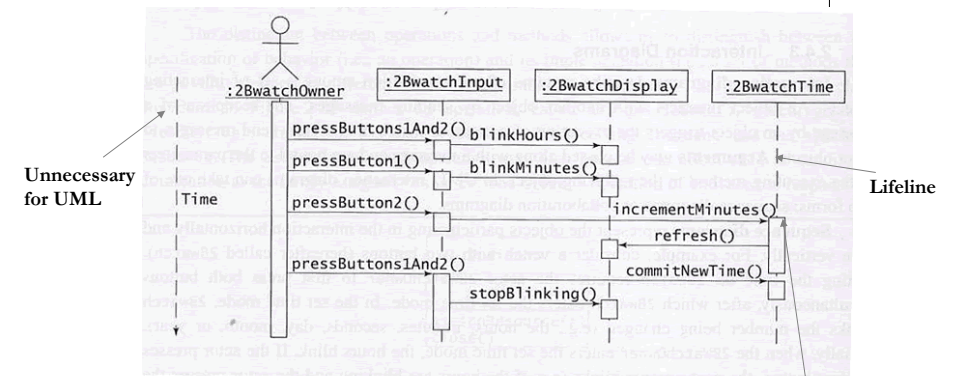


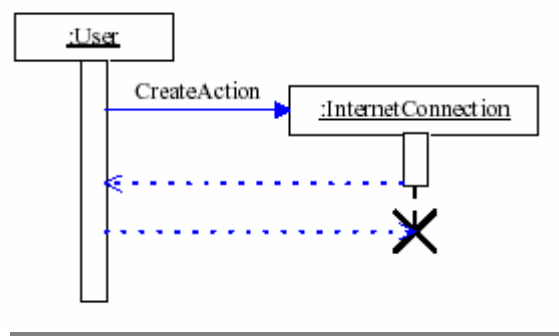
Figure 2-34 Example of a sequence diagram: setting the time on 2Bwatch.

Activation of object

32

# Sequence Diagrams

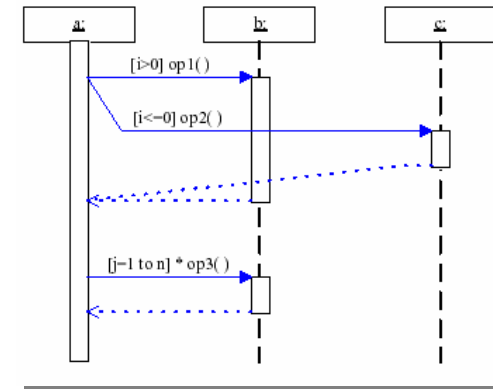
- Object creation and destruction



33

# Sequence Diagram

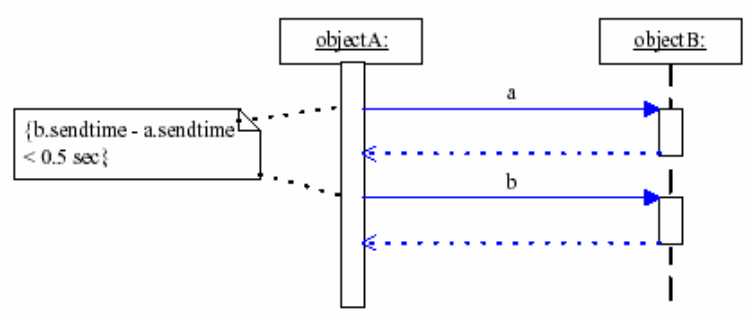
- Condition and iteration



34

# Sequence Diagrams

- Timing constraint
  - Sending time, receive time, elapsed time, queue time, etc.



35

# Collaboration Diagram

- Numbering the message

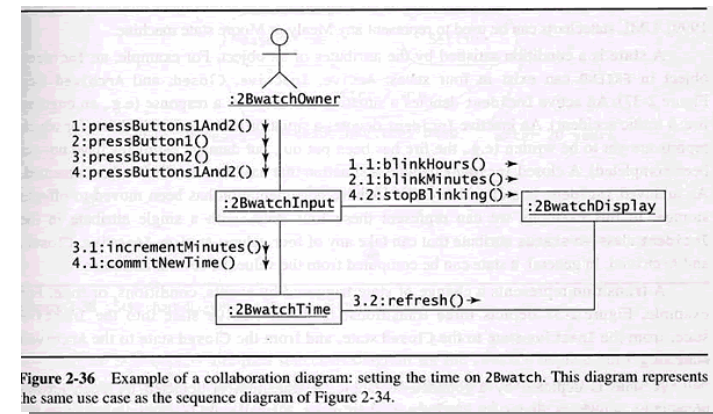


Figure 2-36 Example of a collaboration diagram: setting the time on 2Bwatch. This diagram represents the same use case as the sequence diagram of Figure 2-34.

36

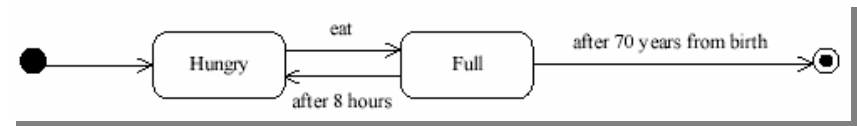


## Statechart Diagrams

37

## Startchart Diagram

- Shows the behavior of a single object in terms of states and transition between states
- Represents different information than sequence diagram
- State transitions – result of external events for a single object

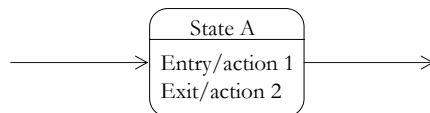


38

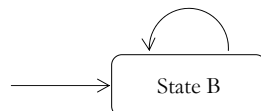
## Statechart Diagram

- Represent the behavior of nontrivial objects

- State
- Transition
  - Internal transition

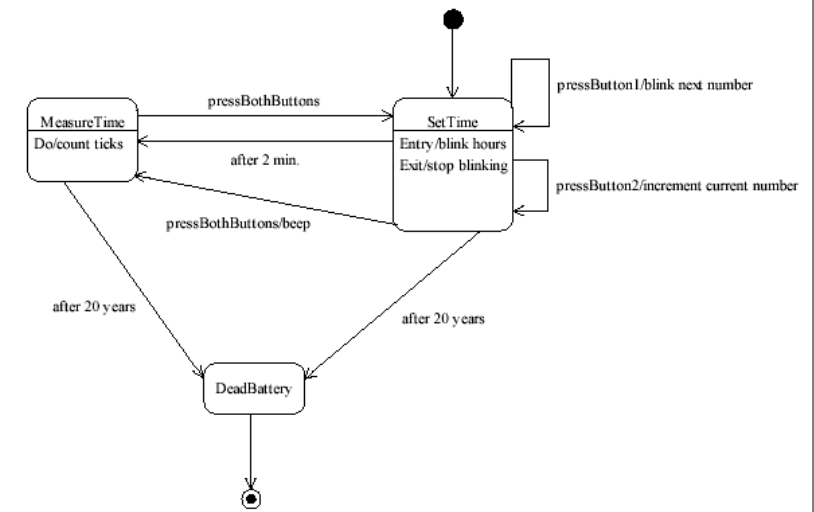


- Self-transition



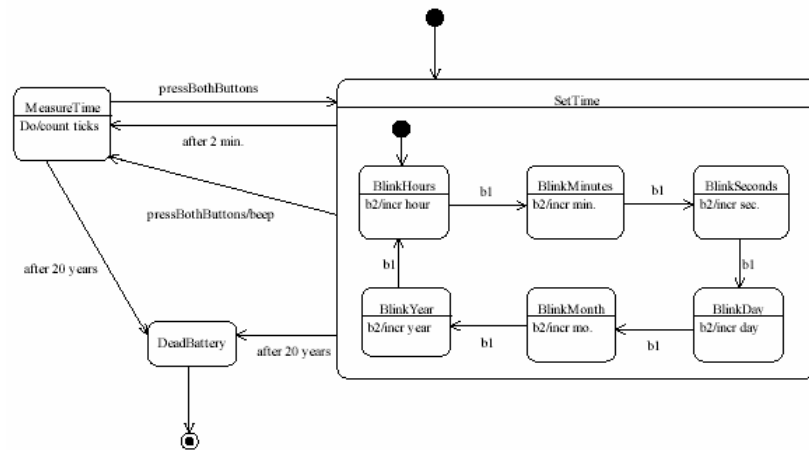
39

## Statechart Diagram



# Statechart Diagram

- Nested state

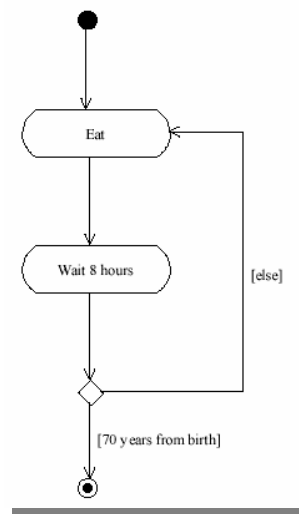


# Activity Diagrams



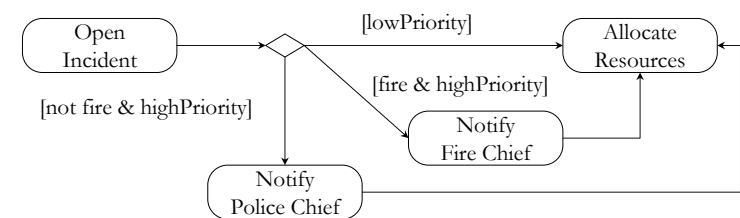
# Activity Diagrams

- Shows system behavior by the flow of activity
- Control flow and data flow
- Activities – modeling elements, execution of a set of operations
- Synchronization of control flow (represented by thick bar)



# Activity Diagram

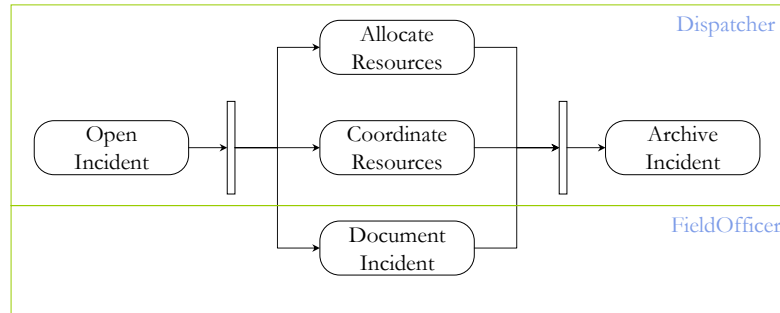
- Represent **data flow** or **control flow** through a system
- Initial and stop states
  - Not mandatory but increase readability
- Decision





## Activity Diagram

- Synchronization bar
  - Fork, Join, Swimline



45



## Chapter 4: Requirement Elicitation

46

## Requirement Elicitation and Analysis

- How do we identify the purpose of a system?
- Requirements Process
  - Requirements Elicitation
    - Definition of the system in terms understood by the customer: Problem Description
  - Requirements Analysis
    - Technical specification of the system in terms understood by the developer: Problem Specification

47



## Requirement Elicitation

- Very challenging activity in real world situation
- Requires collaboration of people with different backgrounds
  - **Users** with application domain knowledge
  - **Developer** with solution domain knowledge (design knowledge, implementation knowledge)
- Bridging the gap between user and developer
  - **Scenarios**: Example of the use of the system in terms of a series of interactions with between the user and the system
  - **Use cases**: Abstraction that describes a class of scenarios

48



## Requirement Elicitation



- When will we perform requirement elicitation?
  - When there is a problem in current situation that we can address
  - Why we can address the problem?
    1. Because something has changed...
    2. Change in application domain
      - New function (business process) is introduced into the business
    3. Change in solution domain
      - New solution (new technologies) is introduced

49

## Requirements Elicitation



- Types of Requirements Elicitation
  - **Greenfield engineering**
    - Development start from scratch
  - **Reengineering**
    - Development based on an existing system
    - With intention to replace the existing system
  - **Interface engineering**
    - Reworks the user interface of an existing system

50

## Requirement Elicitation



- Start with **Problem Statement**
- Focus on the requirements from the user's view of the system.
- Some more output ...
  - **System Specification**
    - derived from the *problem statement*
    - uses natural language
  - **Analysis Model**
    - uses formal or semi-formal notation (e.g.UML)

51

## Requirement Elicitation



- Problem
  - Describes the problem addressed by the system
    - Current situation
    - Desirable situation
    - Brief description of proposed system
    - Scenarios of using proposed system
    - Significant system's benefits
      - Well founded and not exaggerated

52

## Requirement Elicitation



- Problem Statement contains :
  - Problem
  - Objectives
  - Functional requirements
  - Non-functional requirements
  - Target environment

53

## Requirements Elicitation



- **Functional vs Non-functional requirements**
  - **Functional requirements**
    - Describe external behavior (not internal structure)
  - **Non-functional requirements**
    - Additional constraints
    - FURPS+
      - Functional
      - Usability
      - Reliability
      - Performance
      - Supportability
      - + miscellaneous requirements

54

## Requirement Elicitation



- What is usually not in the requirements?
  - System structure, implementation technology
  - Development methodology
  - Development environment
  - Implementation language
  - Reusability

55

## Requirements Elicitation



- Requirements validation criteria (1/3)
  - **Completeness**
    - Contains all scenario the system has to deal with
  - **Consistency**
    - Internal consistency
      - Keep related information closed together
      - Prevent duplicated information
      - Avoid calling same concept by different names
    - External consistency

56

## Requirements Elicitation



- Requirements validation criteria (2/3)

- **Clarity**
  - Be unambiguous
  - Use formal specification (e.g. OCL)
- **Correctness**
  - Requirement specifications in line with system's intended business objective

57

## Requirements Elicitation



- Requirements validation criteria (3/3)

- **Realistic**
  - Requires experience
- **Verifiable**
  - Quantifying the requirement
- **Traceable**
  - Every requirement can be traced to the appropriate function(s) in the system

58

## Requirements Elicitation Activities



## Requirements Elicitation Activities



- Identifying stakeholders
- Creating problem statement
- Identifying actors
- Identifying scenarios
- Identifying use cases
- Refining use cases
- Identifying relationships among actors and use cases
- Identifying initial analysis objects
- Identifying non-function requirements

59

60

## Requirements Elicitation Activities



- Identifying Stakeholders
  - Users of the system from within / outside the company
  - Payer for the system
  - People who will benefit / suffer by the system
  - Maintainer of the system, etc.
- Identifying Actors
  - Primary actors (*receive services*)
  - Supporting actors (*provides services*)
  - Actor profile table / Persona
    - Actor name, background, skills and operations

61

## Requirements Elicitation Activities



- Identifying Scenarios
  - Question to ask yourself or the clients
    - What are the primary tasks that the system needs to perform?
    - What data will the actor create, store, change, remove or add in the system?
    - What external changes does the system need to know about?

62

## Requirements Elicitation Activities



- Identifying Use Cases
  - A use case captures all the scenarios with regard to a single purpose
  - Textual representation consist of following items
    - Use case name
      - Begin with a verb (e.g. BuyLunchSet)
    - Actors
      - Indicate who is the initiating actor, participating actor
    - Flow of events
    - Entry condition
    - Exit condition
    - Quality requirement

63

## Requirements Elicitation Activities



- Refining Use Cases
  1. Expand the use case to include detailed behavior of other related scenarios
  2. Capturing exceptional scenarios in a detailed extending use case
    - Reduce the need for if-statements
    - Prevent the main flow from becoming cluttered
  3. Add and drop use cases
  4. But don't get into user interface design
    - Not in this stage

64

## Requirements Elicitation Activities



- Identifying relationships among actors and use cases
  - Relationship between an actor and a use cases
    - <<initiate>> and <<participate>>
  - Shared behavior between use cases
    - <<include>>
      - Functional decomposition
      - Reuse of existing functionality
    - <<extend>>
      - Association for use cases
      - Exceptional handling

65

## Requirements Elicitation Activities



- Identifying relationships among actors and use cases (cont.)
  - Three levels of use cases
    - Top level
      - High-level use cases
      - Use of <<initiate>> and <<participate>> stereotypes
    - 2<sup>nd</sup> level
      - High-level use cases with its included detailed uses cases
      - Use of <<include>> stereotype
    - 3<sup>rd</sup> level
      - Use cases and related use cases with <<extend>> stereotype

66

## Requirements Elicitation Activities



- Identifying initial analysis objects
  - Creating a [glossary](#) / [data dictionary](#)
    - Help to identify and eliminate the use of the same term for different meanings
    - Try to stick with one term for a concept in the requirement specification

67

## Chapter 5: Requirements Analysis



68

## Requirements Analysis



- Requirements elicitation produces
  - *Functional requirements*
  - *Non-functional requirements*
- Requirement analysis consume
  - *Functional* and *non-functional requirements*
- Requirement analysis produce
  - *Analysis object model*
  - *Dynamic model*

69

## Requirements Analysis



- Analysis model
  - *Functional model*
    - Use cases, scenarios
  - *Analysis object model*
    - Class and object diagrams (express system structure)
  - *Dynamic model*
    - Statechart and sequence diagrams (express system behavior)

**Note: Analysis – user level only (don't include system level materials such as database, network, etc)**

70

## Requirements Analysis



- Analysis object model
  - From use cases to objects
  - *Entity*
    - Application domain concept
    - Seek in glossary / data dictionary
  - *Boundary*
    - Interface between an actor and the system
    - Prevent detailed screen elements (too detailed at this stage)
  - *Control*
    - Coordinating the entity and boundary objects

71

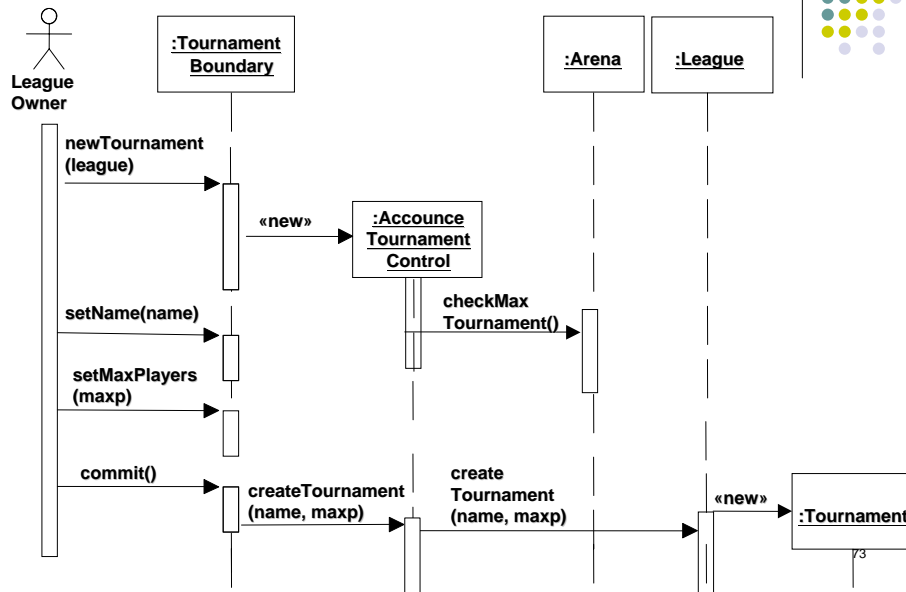
## Requirements Analysis



- Mapping use case to objects with sequence diagrams
  - Help us to find missing objects or grey areas in the requirements specification
  - 1<sup>st</sup> column – *Actor*
  - 2<sup>nd</sup> column – *Boundary* (Actor used to initiate the use case)
  - 3<sup>rd</sup> column – *Control* (manages rest of use case)
  - Control objects are created by boundary objects initiating the use cases
  - Entity objects are accessed by control and boundary objects.
  - Entity objects *never* access boundary or control objects.

72

## Mapping use case to objects with sequence diagrams



## Requirements Analysis

### • Modeling Classes

- Identifying Associations
  - If an instance of one class **creates** / **destroys** / **uses** / **owns** an instance of another class
- Identifying Aggregations
  - Look for keywords “has”, “consists of”, “includes”, etc. in the statements.
  - Aggregation / shared aggregation (◊)
    - The whole and the part can exist independently
  - Composition / composite aggregation (◆)
    - The existence of the parts depends on the whole
- Identifying Attributes
  - Can change during the software development process

74

## Requirements Analysis

### • Modeling Classes (cont.)

- Identifying state-based behavior
  - Use statecharts to model entity objects with extended life spans. (beyond a single scenario)
- Identifying inheritance
  - Look for keywords “is a”, “is a kind of”, “is one of either”
  - Help to clarify any misunderstanding of the application domain
- Capture association, aggregation and inheritance in three separate class diagrams

75

## Requirements Analysis

### • Reviewing the analysis model

- For each object
  - Check which use case it is **created** / **used** / **destroyed** and **modified**.
  - If object is created but never destroyed, use case is missing.
  - If the object doesn't fill in any of the four areas (**created** / **used** / **destroyed** / **modified**), maybe we forgotten a use case for the object.

76