

Chapter 4

Requirements Analysis and Validation

Object-Oriented
Software Construction

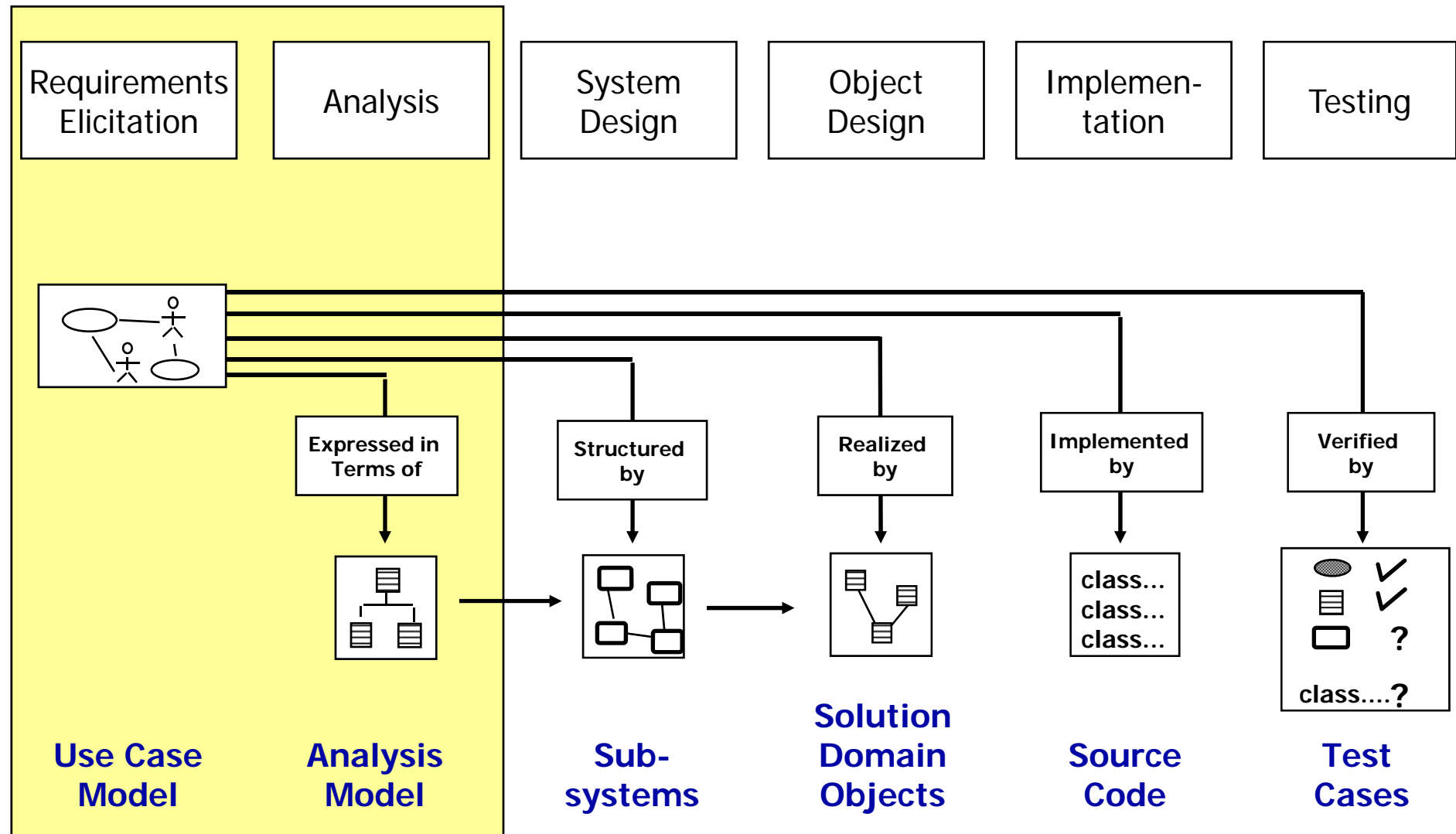
Armin B. Cremers, Tobias Rho, Daniel Speicher, Holger Mügge
(based on Bruegge & Dutoit)



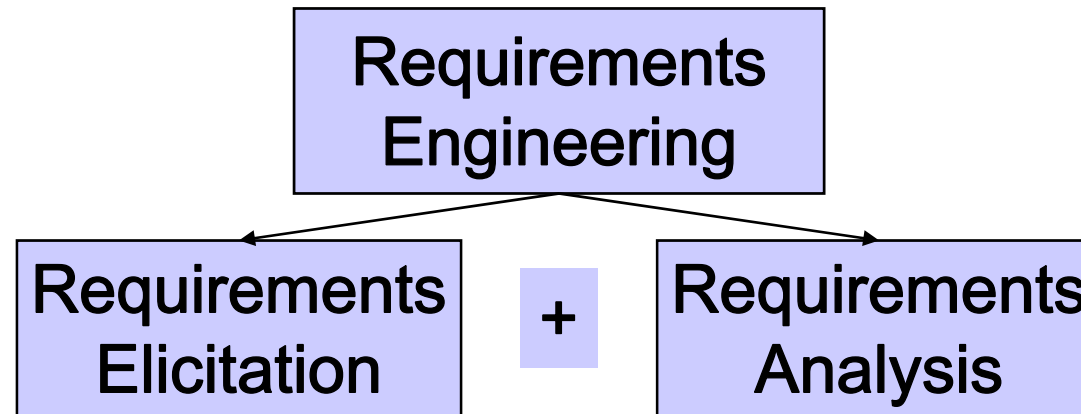
Overview

- ◆ Introduction to Requirements Analysis
- ◆ Class and Object Identification
- ◆ The Abbott's Method
- ◆ The Analysis Object Model
- ◆ The Dynamic Model
- ◆ Boundaries, Entities & Controller
- ◆ Requirements Verification and Negotiation

Software Development Process: A Brief Overview



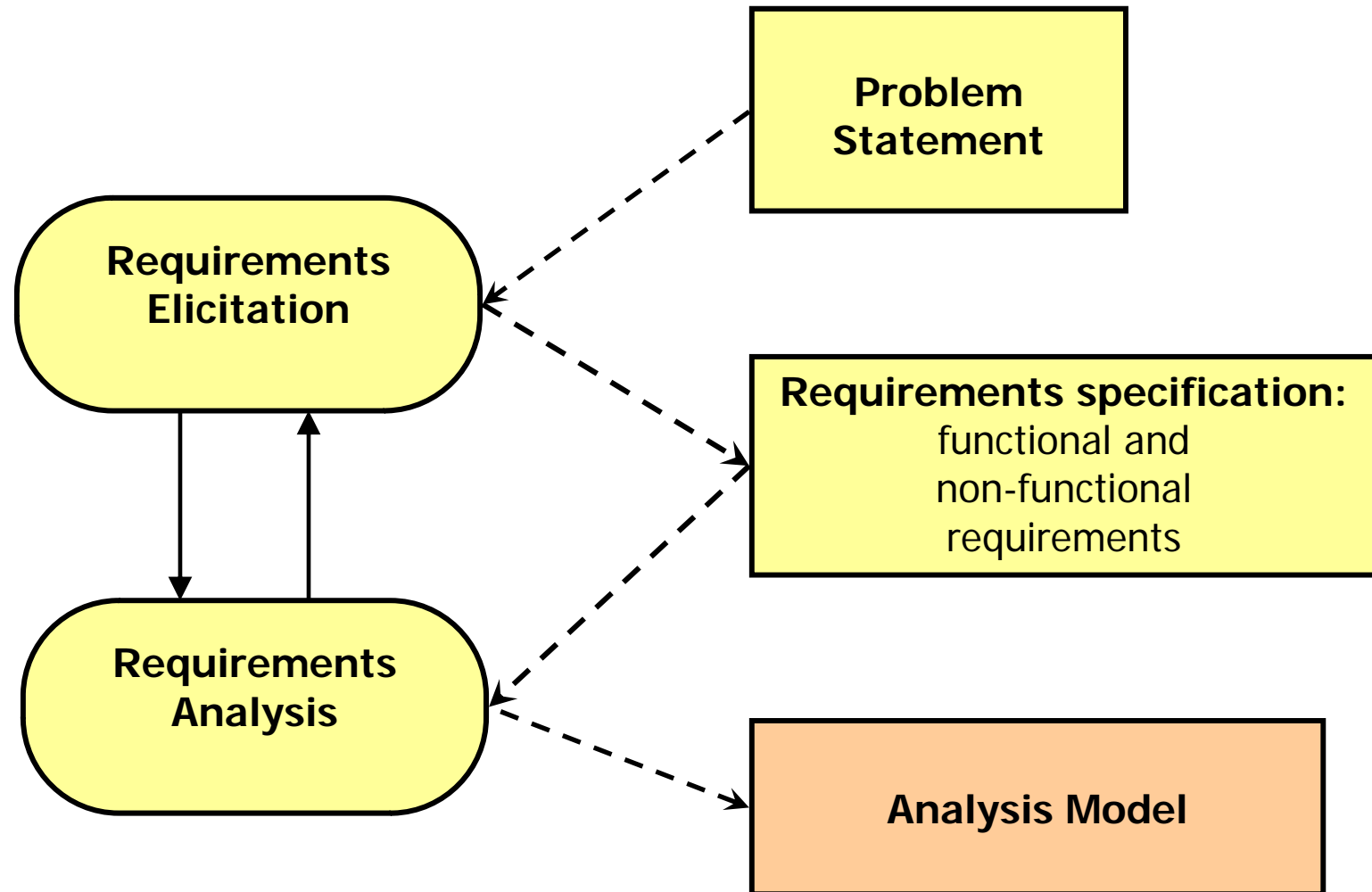
What is Requirements Engineering?



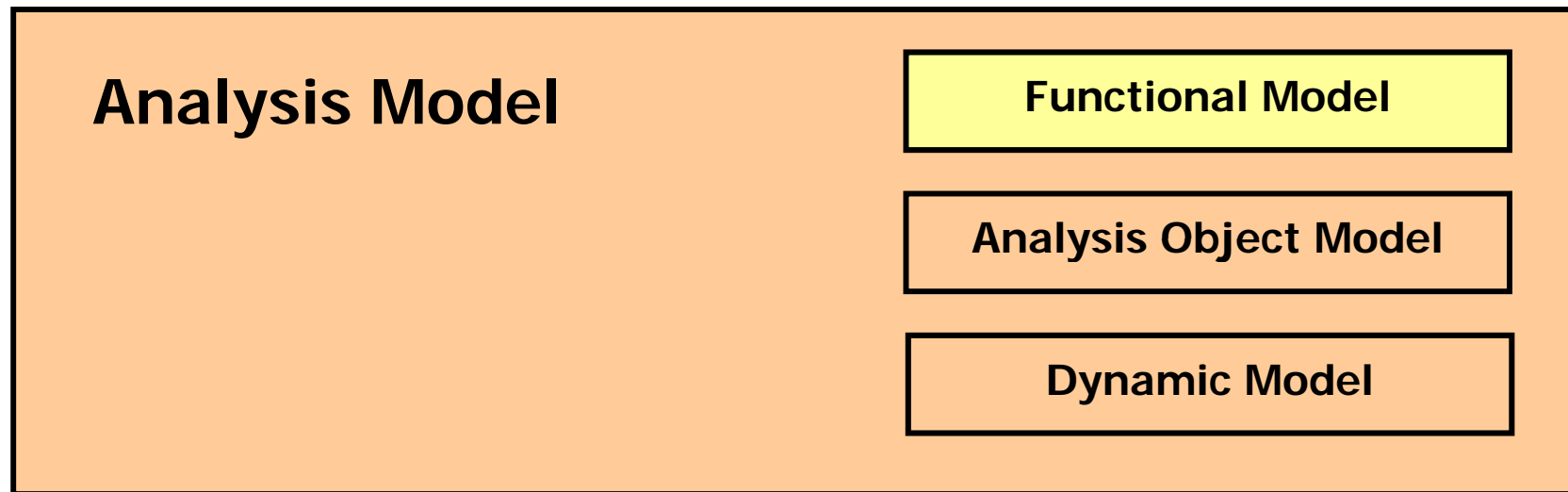
- Basis for Discussions with customer
- Use Customer's language to define system
- Problem Description
- Focused, informal

- Design basis for developers
- Use developer's language to define systems
- Problem Specification
- Technical, precise

Process of Requirements Analysis



The Goal: The Analysis Model



The functional model

- contains Scenarios & Use Cases
- has been elaborated during Requirements Elicitation

Analysis and Analysis Model

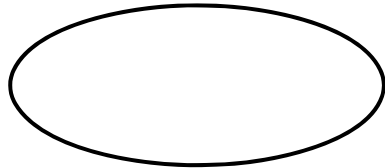
Some general aspects

- ◆ The *analysis model* is the basis for further development
 - ◆ Analysis Object Model (application domain objects, participating objects)
 - ◆ Dynamic Model (Behavioral aspects of these objects)
- ◆ Analysis focus on producing a model of the system (analysis model) which is:
 - ◆ Correct
 - ◆ Complete
 - ◆ Consistent
 - ◆ Unambiguous
 - ◆ Verifiable
- ◆ Structuring and formalizing the requirements
 - ◆ New details are added, insights and errors may be discovered
 - ◆ → Update of System Specification often necessary

Use Case Diagrams

Use Case

- ◆ An use case represents a class of functionality provided by the system as a sequence of interaction (event flow)



PurchaseTicket

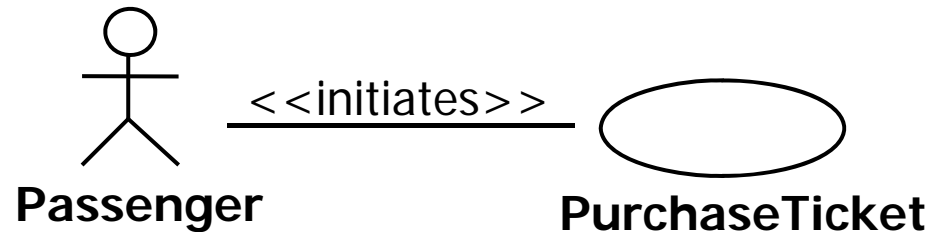
A use case consists of

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

→ Hard to infer from the graphical representation

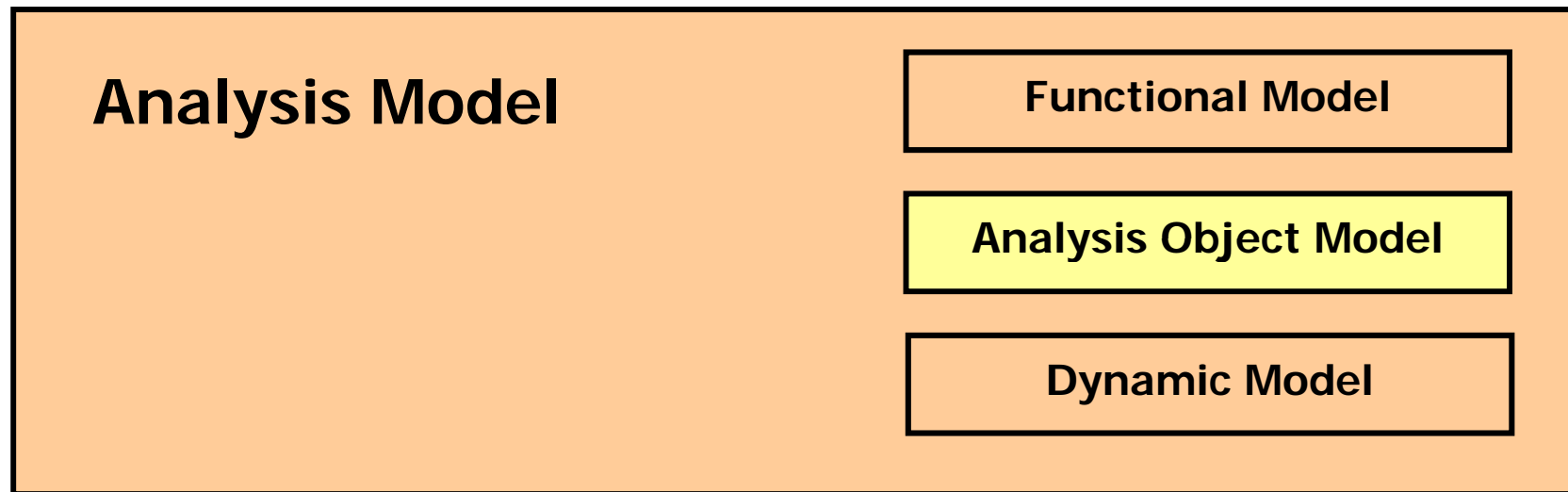
Use Case Diagrams

Textual Representation



<i>Use case name</i>	Purchase ticket
<i>Participating Actors</i>	Initiated by Passenger
<i>Flow of Events</i>	<ol style="list-style-type: none">1. Passenger selects the number of zones to be traveled.2. SYSTEM displays the amount due.3. Passenger inserts money, of at least the amount due (Extension Point: handleException).4. SYSTEM returns change.5. SYSTEM issues ticket.
<i>Entry Condition</i>	Passenger standing in front of ticket distributor.
<i>Exit Condition</i>	Passenger has a correct ticket.
<i>Quality Requirements</i>	<ul style="list-style-type: none">• The Passenger's ticket is issued within 15 seconds.

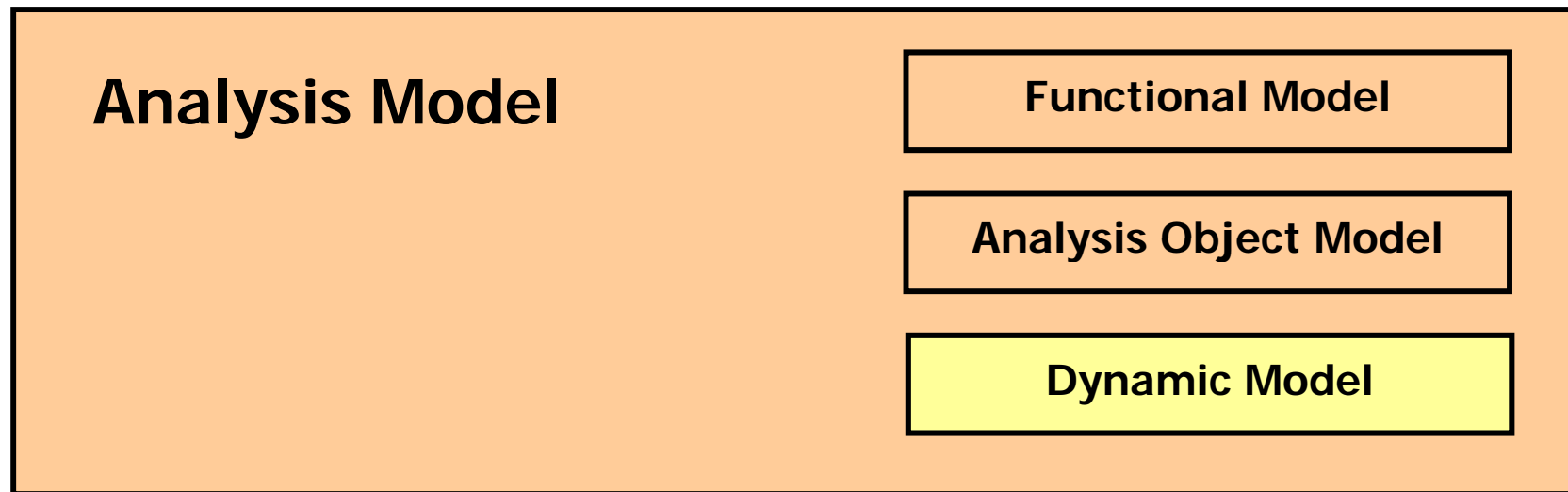
The Goal: The Analysis Model



The analysis object model

- preserves knowledge of the system structure and domain
- represents static view on the system
- uses Class and Object Diagrams

The Goal: The Analysis Model

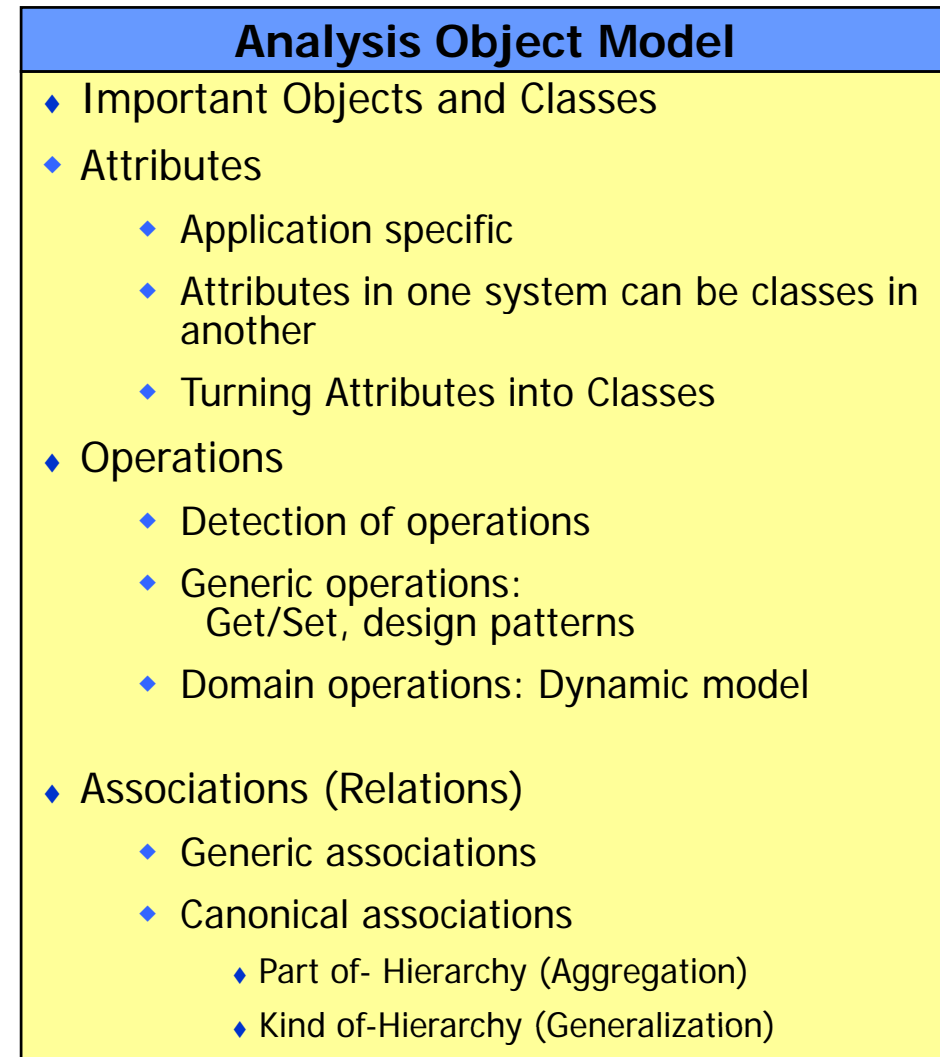
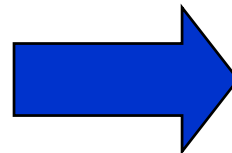
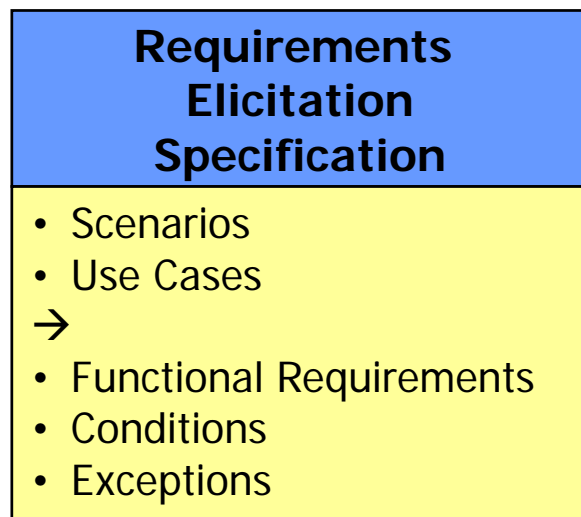


The dynamic model

- preserves knowledge of the system behavior
- uses Sequence diagrams and Statecharts

- ◆ Analysis Object Model
 - ◆ Static view of the system
 - ◆ Classes and Objects (User-Level Concepts, no software classes!)
 - ◆ Attributes
 - ◆ Operations
 - ◆ Associations
 - ◆ → can serve as visual dictionary of application domain
- ◆ Dynamic Model
 - ◆ Models the behavior of the system
 - ◆ Responsibilities of classes, Flow of Control
 - ◆ In general using
 - ◆ Statechart diagrams (1 Object)
 - ◆ Sequence diagram (Flow of control, relationships)

From Requirements Specification to the Analysis Object Model



Analysis Object Model: Class Identification (1/2)

- ◆ Identify and describe all important entities in the system
- ◆ Steps during object modeling
 - ◆ 1. Class identification
 - ◆ Fundamental assumption: we can find abstractions
 - ◆ 2. Find the attributes
 - ◆ 3. Find the methods
 - ◆ 4. Find the associations between classes
- ◆ What happens if we find the wrong abstractions?
 - ◆ Iterate and correct the model
 - ◆ How do we know that something is wrong? Ask the customer!
- ◆ Order of steps secondary, only a heuristic (Iteration is important)

Analysis Object Model: Class Identification (2/2)

- ◆ Objects are not just found by taking a picture of a scene or domain
- ◆ The application domain has to be analyzed
- ◆ Depending on the purpose of the system different objects might be found
 - ◆ How can we identify the purpose of a system?
- ◆ **Take the Scenarios and use cases as foundation!**
- ◆ Another important problem: Define system border
 - ◆ What object is inside, what object is outside?

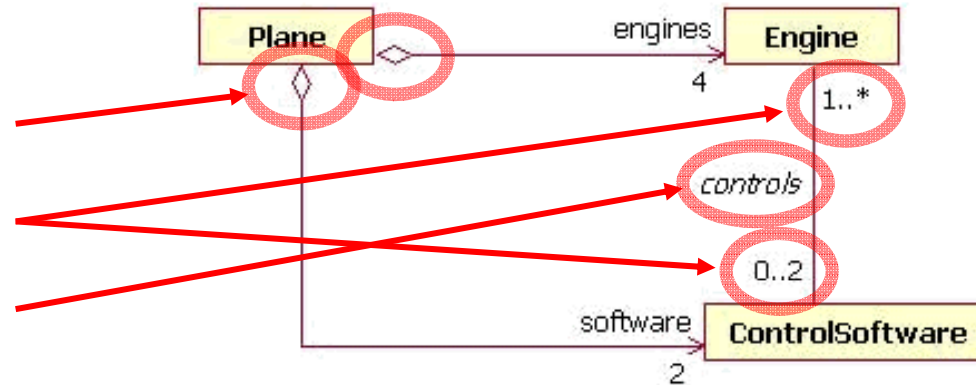
Analysis Object Model: Object vs. Class

- ◆ Object (instance): Exactly one thing
 - ◆ This lecture on OOSC on April, 28rd 2008 from 16:00 - 17:30
- ◆ A class describes a group of objects with similar properties
 - ◆ Lectures, games, tournaments, mechanics, cars, databases
- ◆ UML provides graphic notations for modeling objects, classes and their relationships
 - ◆ Class diagram
 - ◆ Template for describing many instances of data
 - ◆ Useful for taxonomies, patterns, schema...
 - ◆ Object diagram
 - ◆ A particular set of objects relating to each other.
 - ◆ Useful for discussing scenarios, test cases and examples

Class and Object Diagrams

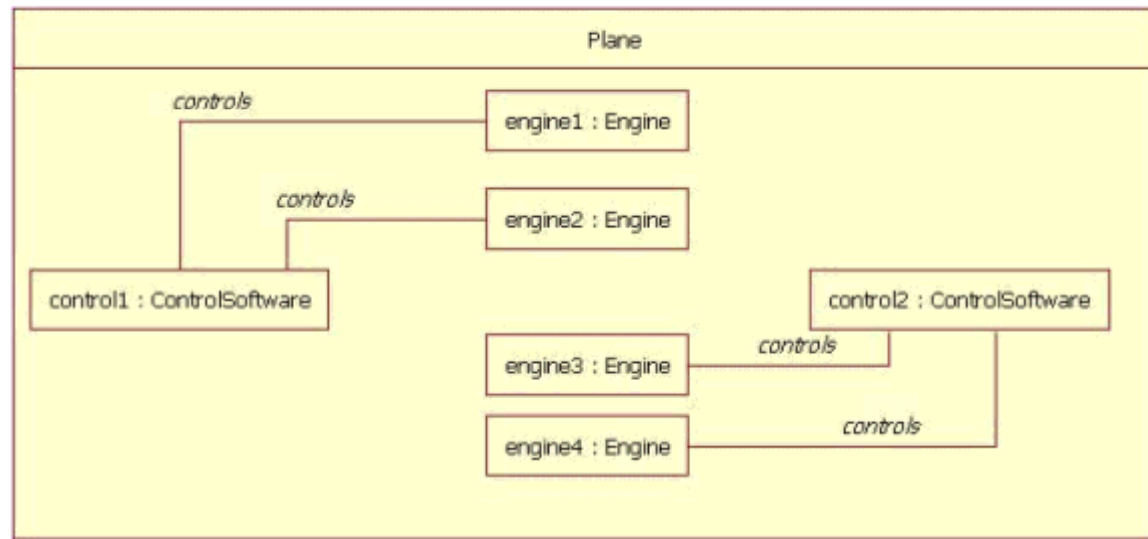
- ◆ Classes

- ◆ containment
- ◆ cardinalities
- ◆ associations



- ◆ Objects

- ◆ containment
- ◆ cardinalities
- ◆ associations



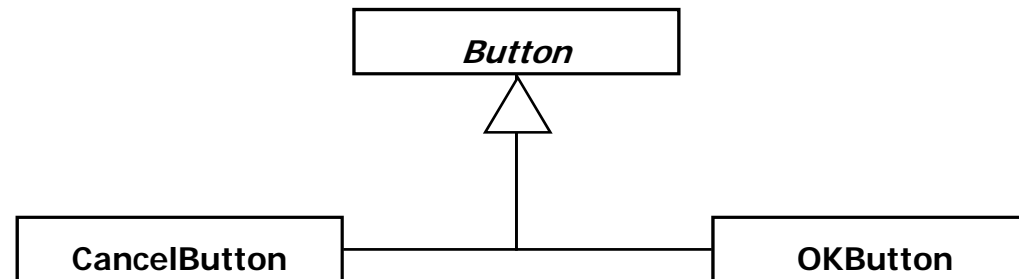
Identify Dependencies: Associations and Aggregates

- ◆ An Association consists of
 - ◆ a name
 - ◆ roles (2 or more)
 - ◆ multiplicities

- ◆ Look out for Aggregations
 - ◆ Note the difference between
 - ◆ Composition (Lifetime is dependent) ◆
 - ◆ (Shared) aggregation ◆

Identify Dependencies: Generalization and Specialization

- ◆ Inheritance as the guiding concept
- ◆ Generalization
 - ◆ Abstract activity or concept out of low-level ones
- ◆ Specialization
 - ◆ Start with the general case + refinement



Analysis Object Model: Abbott Textual Analysis [Abbott, 1983]

- ◆ Set of heuristics for identifying objects, attributes and associations from a requirements specification
- ◆ Mapping of parts of speech to model components
 - ◆ Nouns are good candidates for classes
 - ◆ Verbs are good candidates for operations
 - ◆ ... and further more specific mappings (next slide)
- ◆ In the problem statement (originally proposed, but rarely works if the problem statement is large - more than 5 pages)
 - ◆ Quality depends on style of writing
 - ◆ Too many nouns
- ◆ In the flow of events of use cases (preferable)
 - ◆ Small list of candidate objects

Mapping parts of speech to object model components

<i>Part of speech</i>	<i>Model component</i>	<i>Example</i>
Proper noun	object	Jim Smith
Common noun	class	Toy, Doll
Doing verb	method	buy, recommend
being verb	inheritance	is-a, either .. or
having verb	aggregation	has an
modal verb	constraint	must be
adjective	attribute	3 years old
transitive verb	method	enter

Analysis Object Model: Identifying Participating Objects

- ◆ Pick a use case and look at its flow of events
 - ◆ Find terms that developers or users need to clarify in order to understand the flow of events
 - ◆ Look for recurring nouns (e.g., Incident),
 - ◆ Identify real world entities that the system needs to keep track of (e.g., FieldOfficer, Dispatcher, Resource),
 - ◆ Identify real world procedures that the system needs to keep track of (e.g., EmergencyOperationsPlan),
 - ◆ Identify data sources or sinks (e.g., Printer)
- ◆ Be prepared that some objects are still missing and need to be found:
 - ◆ Model the flow of events with a sequence diagram (later..)
- ◆ Always use the user's terms

Analysis Object Model: Example – Analyzing The Flow of events

- ◆ The customer enters the store to buy a toy
- ◆ It has to be a toy that his daughter likes and it must cost less than 50 Euro.
- ◆ An assistant helps him.
- ◆ The suitability of the toy depends on the age of the child.
- ◆ His daughter is 10 years old
- ◆ The assistant recommends another type of toy, namely the board game “Monopoly”. For this game, the players must be older than 8.
- ◆ The assistant also suggests a game collection which is also a toy that consists of many separate game like chess and card-games. For this collection, the players must be older than 5
- ◆ *Not a very good description, but okay as example ;-)*

Analysis Object Model: Analyzing the nouns (selection)

- ◆ The **customer** enters the **store** to buy a **toy**
- ◆ It has to be a toy that his daughter likes and it must cost less than 50 Euro.
- ◆ An assistant helps him.
- ◆ The suitability of the toy depends on the age of the child.
- ◆ His daughter is 10 years old
- ◆ The assistant recommends another type of toy, namely the **board game** "Monopoly". For this game, the players must be older than 8.
- ◆ The assistant also suggests a **game collection** which is also a toy that consists of many separate game like chess, card-game. For this collection, the players must be older than 5

Analysis Object Model: Analyzing the verbs (selection)

- ◆ The **customer** **enters** the **store** to buy a **toy**
- ◆ It has to be a toy that his daughter likes and it must cost less than 50 Euro.
- ◆ An assistant helps him.
- ◆ The suitability of the toy depends on the age of the child.
- ◆ His daughter is 10 years old
- ◆ The assistant **recommends** another type of toy, namely the **board game** "Monopoly". For this game, the players must be older than 8.
- ◆ The assistant also suggests a **game collection** which is also a toy that consists of many separate game like chess, card-game. For this collection, the players must be older than 5

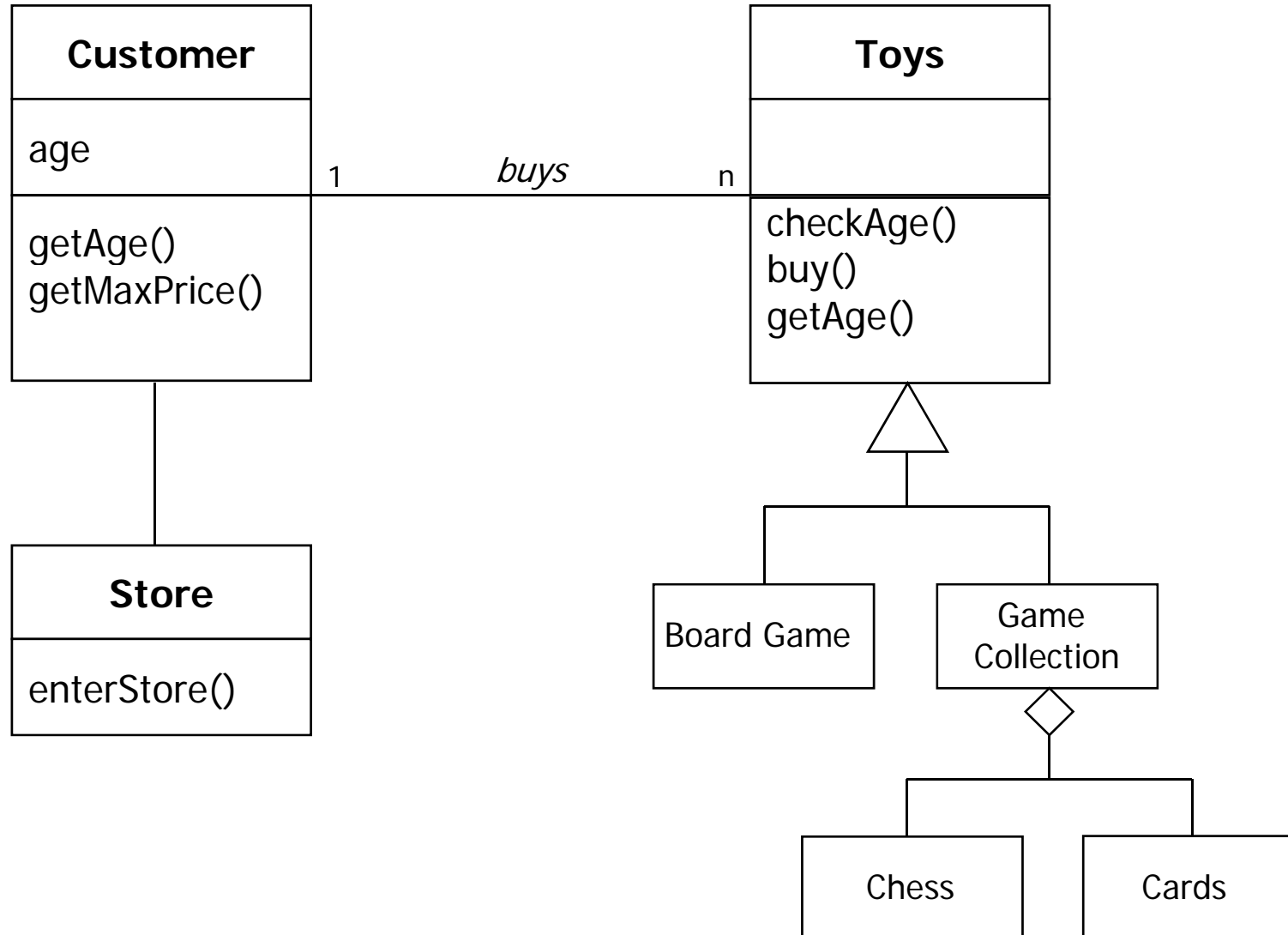
Analysis Object Model: Hierarchy and Inheritance

- ◆ The **customer** **enters** the **store** to buy a **toy**
- ◆ It has to be a toy that his daughter likes and it must cost less than 50 Euro.
- ◆ An assistant helps him.
- ◆ The suitability of the toy depends on the age of the child.
- ◆ His daughter is 10 years old
- ◆ The assistant **recommends** another **type of toy**, namely the **board game** "Monopoly". For this game, the players must be older than 8.
- ◆ The assistant also suggests a **game collection** which is also a toy that **consists** of many separate game like chess, card-game. For this collection, the players must be older than 5

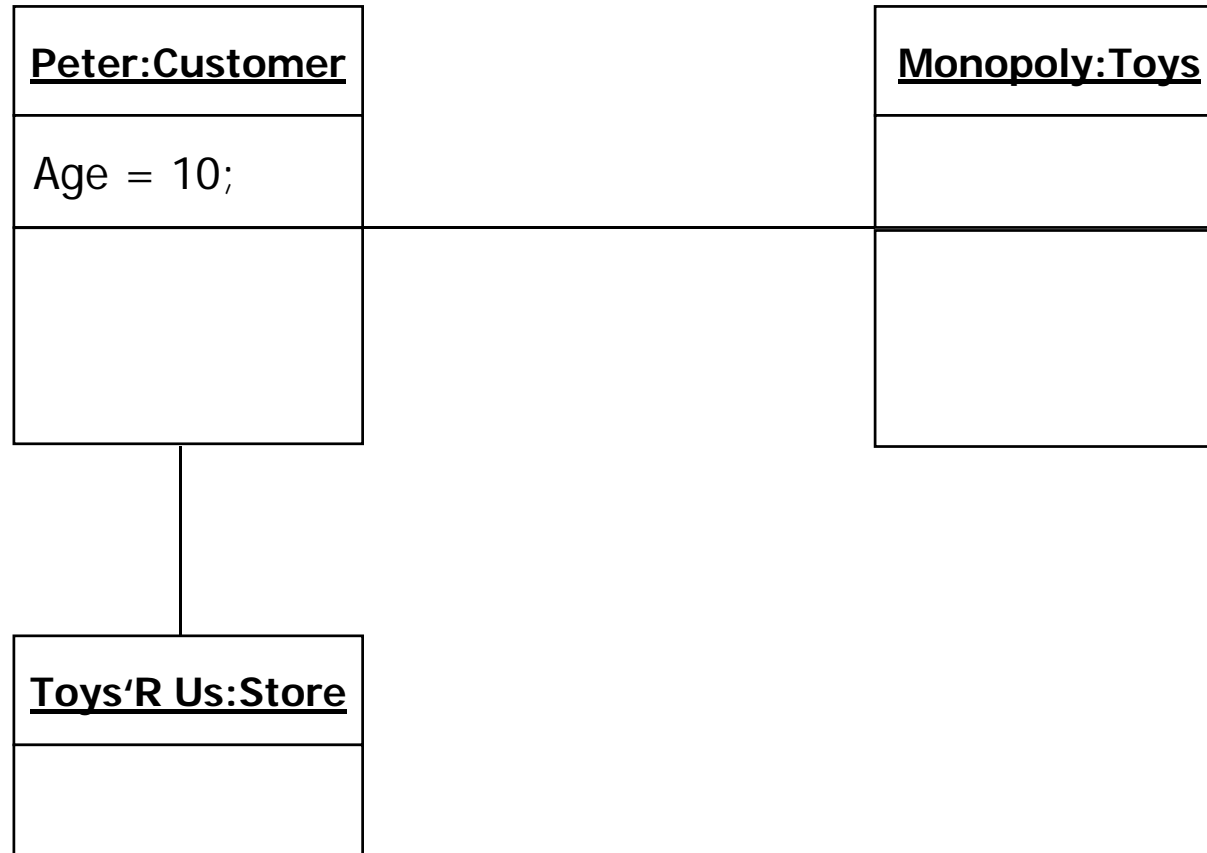
Analysis Object Model: Overview of Analysis

<i>Example</i>	<i>Grammatical construct</i>	<i>UML Component</i>
"Monopoly"	Proper noun	Object
"toy", "board game", customer	Common noun	class
"10 years old", "50 Euro"	Adjective	Attribute
"enters"	Verb	Method
"recommends...."	Doing verb	Method
"is a" , "type of..."	Classifying verb	Inheritance
"Has a " , "consists of"	Having Verb	Aggregation
"must be" , "less than"	modal Verb	Constraint

Analysis Object Model: Class Diagram (Sketch)



Analysis Object Model: Object Diagram (Sketch)



Analysis Object Model: Three Object Types

- ◆ **Problem:** What to do with participating objects?
- ◆ **Suggestion:** Partition into three types
- ◆ Entity Objects
 - ◆ Represent the persistent information tracked by the system (Application domain objects, “Business objects”)
- ◆ Boundary Objects
 - ◆ Represent the interaction between the user and the system
- ◆ Control Objects
 - ◆ Represent the control tasks performed by the system
- ◆ Having three types of objects leads to models that are more resilient to change
 - ◆ The interface of a system changes more likely than the control
- ◆ Standard UML does not distinguish among these types

Analysis Object Model: Example - Breakout Game

Display

LeftButton

RightButton

Boundary Objects

GameControl

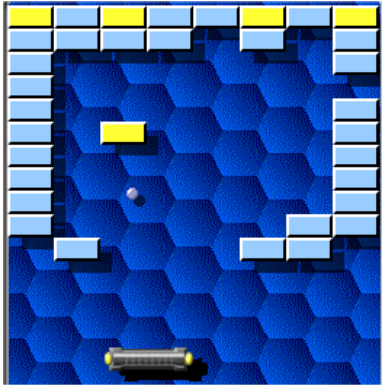
Control Objects

Points

HighScore

GameState

Entity Objects



Analysis Object Model: Example - Breakout Game (Naming in UML)

- ◆ UML provides several mechanisms to extend the language
- ◆ UML provides the stereotype mechanism to present new modeling elements

<< Boundary >>
Display

<<Control>>
GameControl

<< Entity >>
Points

<< Boundary >>
LeftButton

<< Entity >>
HighScore

<< Boundary >>
RightButton

<< Entity >>
GameState

Identifying Key Objects: Entity Objects

- ◆ Represents persistent or long-lived information tracked by the system
- ◆ Real-World entities from application domain (WishList, Receipt)
- ◆ Real-World activities the system needs to track (Order, EmergencyPlan)

- ◆ Recurring Nouns that are used in the Use Cases

Represent the persistent information tracked by the system (Application domain objects, "Business objects")

Identifying Key Objects: Boundary Objects

- ◆ Represent the system interface for the actors
- ◆ Forms, Messages, Terminals for different users
 - ◆ Identify Forms the user needs to enter data into the system
 - ◆ Identify messages the system uses to respond to the user
- ◆ Also represent interface for external systems (e.g. online payment)
- ◆ Described in End User Terms
 - ◆ Good: EmergencyReportForm, PlaceOrderButton
 - ◆ Bad: MenuItem, ScrollBar (lacks semantics, just technical)

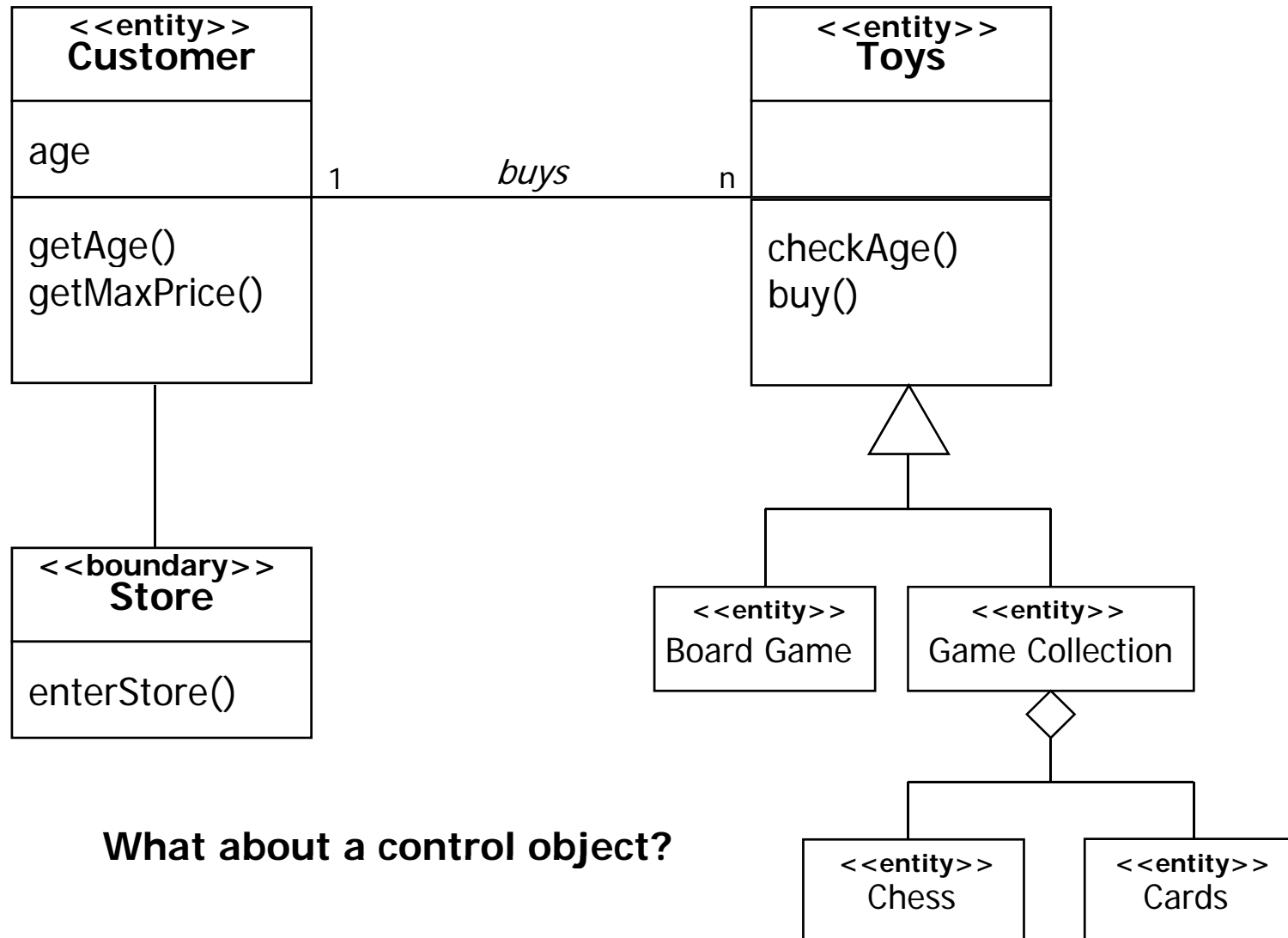
Represent the interaction between
the user and the system

Identifying Key Objects: Control Objects

- ◆ Responsible for coordinating boundary and entity objects
 - ◆ Collecting information from boundary objects
 - ◆ Dispatching it to entity objects
 - ◆ Control of distributed systems
 - ◆ The internal workflow of a system
- ◆ One Control Object per Use Case (no counterpart in real world)
 - ◆ Life span covers the extent of a use case (user session)

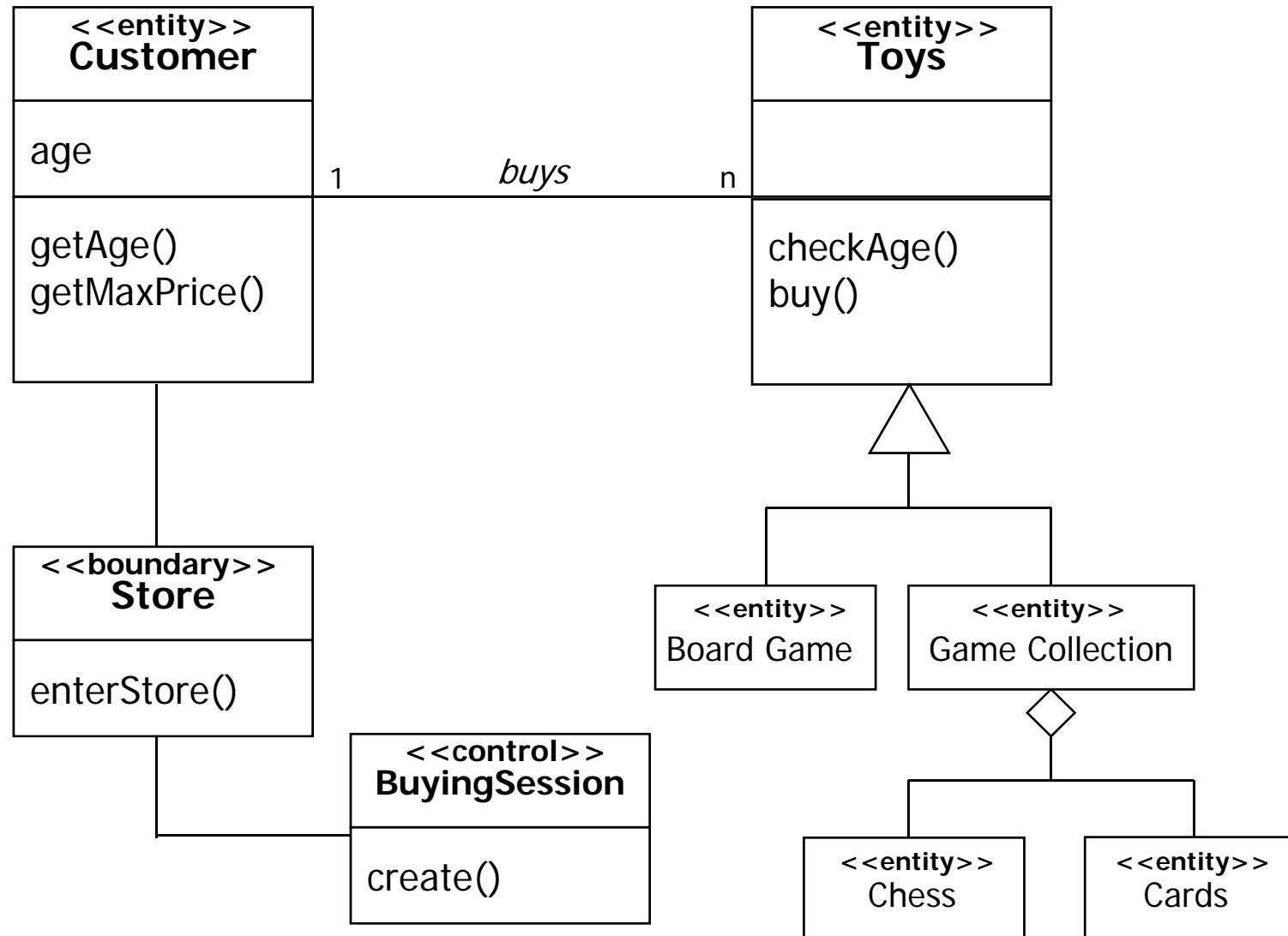
Represent the control tasks
performed by the system

Analysis Object Model: Extension to example Class Diagram



What about a control object?

Analysis Object Model: Extension to example Class Diagram



Steps to Analysis (1/3)

- ◆ Interview end-users or domain experts and gather scenarios
- ◆ Extract use cases from the scenarios, with the help of application domain expert
- ◆ Analyze the flow of events with Abbot's textual analysis
- ◆ Generate the class diagrams, which include the following steps:
 - ◆ Class identification (textual analysis, domain experts) across all Use Cases (→ sharing of classes and objects)
 - ◆ Identification of attributes and operations (sometimes before the classes are found!)
 - ◆ Identification of associations between classes
 - ◆ Identification of multiplicities
 - ◆ Identification of roles
 - ◆ Identification of constraints

Steps to Analysis – Management (2/3)

- ◆ Explicitly schedule meetings for object identification
- ◆ First just find objects
- ◆ Then try to differentiate them between entity, boundary and control objects
- ◆ Find associations and their multiplicity
 - ◆ Unusual multiplicities usually lead to new objects or categories
- ◆ Identify Inheritance: Look for a Taxonomy, Categorize
- ◆ Identify Aggregation

- ◆ Allow time for brainstorming
- ◆ Iterate, iterate, iterate ...

Steps to Analysis – Problems (3/3)

- ◆ Do not defer important decisions
- ◆ Don't insist on finding all objects
- ◆ Don't be too detailed
 - ◆ More details are added during object design
 - ◆ What is easier: adding a detail early or removing it later?
- ◆ Requirements are still incomplete
 - ◆ Check list (later)
 - ◆ Requirements validation (later)
- ◆ Feedback with Customers & other Stakeholders very valuable

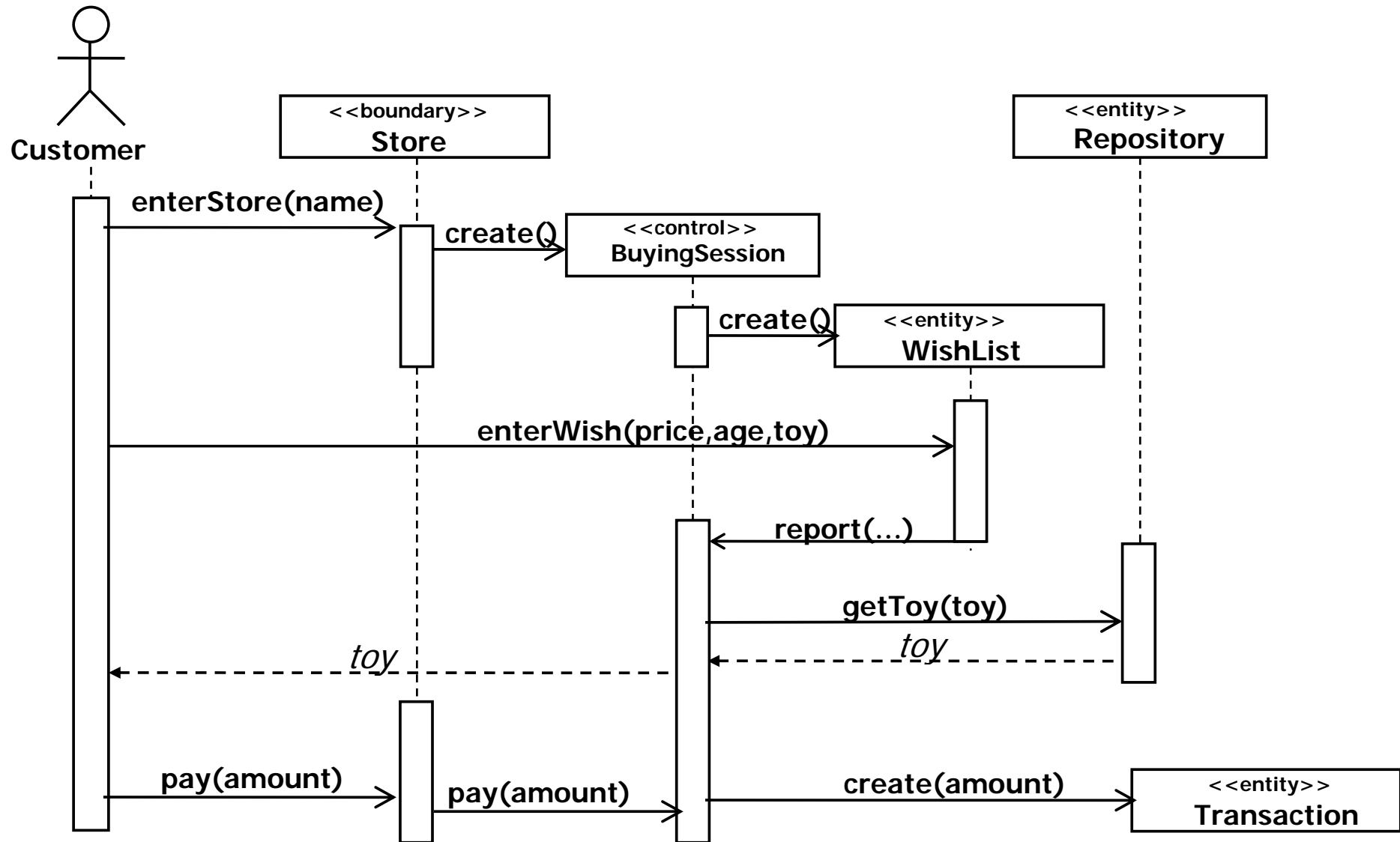
- ◆ Modeling of behavioral aspects among the participating objects
- ◆ Tie use cases with objects
 - ◆ Shows how the behavior of a use case (or scenario) is distributed among its participating objects
 - ◆ Identify new objects and classes
 - ◆ Helps to determine how decentralized the system is.
- ◆ Purpose:
 - ◆ Detect and supply methods for the object model
- ◆ How do we do this?
 - ◆ Start with use case or scenario
 - ◆ Model interaction between objects → sequence diagram, CRC
 - ◆ Model dynamic behavior of a single object → state chart diagram (not handled in this lecture)

- ◆ Sequence Diagram
 - ◆ Can be read out of use cases
 - ◆ Dynamic behavior of a set of objects arranged in time sequence
 - ◆ Good for real-time specifications and complex scenarios
 - ◆ Information of the life time of objects
 - ◆ Relationships and cooperation between objects

- ◆ State Chart Diagram
 - ◆ describes the dynamic behavior of a single object
 - ◆ A state machine that describes the response of an object of a given class to the receipt of outside stimuli (Events).

Drawing Sequence diagrams

Example



Drawing Sequence diagrams

Heuristics

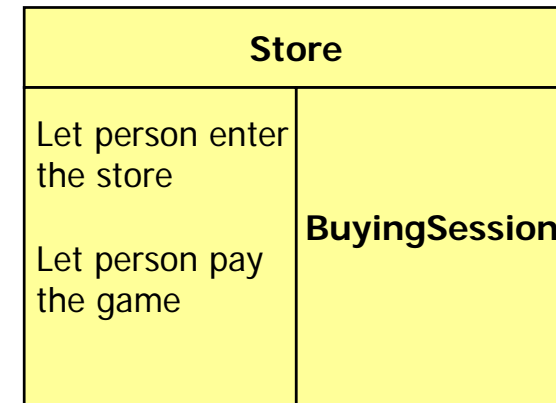
- ◆ The first column should correspond to the actor who initiated the use case
- ◆ The second column should be an (initial) boundary object
- ◆ The third column should be the control object that manages the rest of the use case (only boundary objects create control objects)
- ◆ Only control objects create further boundary objects that are presented to actors
- ◆ Entity objects must not be accessed by actors, only through control and boundary objects.

Dynamic Modeling with CRC (Classes, Responsibilities, Collaborators)

- ◆ Originally developed for teaching purposes [Beck and Cunningham 1989]

- ◆ Card Game

- ◆ One Card represents one class (Store)
- ◆ Describe the responsibilities of that class (left column)
- ◆ Describe the needed classes (collaborators) (right column)






- ◆ Good method for a group of developers during a brainstorming session (easy to create)
- ◆ Sequence Diagrams: good for one developer (more precise but hard to create)

Practical Tips for Dynamic Modeling

- ◆ Construct dynamic models only for classes with significant dynamic behavior
- ◆ Consider only relevant attributes
 - ◆ Use abstraction if necessary
- ◆ Look at the granularity of the application when deciding on actions and activities

Summary: Requirements Analysis

1. What are the functions of the system?  **Functional Modeling**
 - ◆ Create scenarios and use case diagrams
 - ◆ Talk to client, observe, get historical records, do thought experiments
2. What is the structure of the system?  **Object Modeling**
 - ◆ Create class diagrams
 - ◆ Identify objects.
 - ◆ What are the associations between them? What is their multiplicity?
 - ◆ What are the attributes of the objects?
 - ◆ What operations are defined on the objects?
3. What is its behavior?  **Dynamic Modeling**
 - ◆ Create sequence diagrams
 - ◆ Identify senders and receivers
 - ◆ Show sequence of events exchanged between objects. Identify event dependencies and event concurrency.
 - ◆ Create state diagrams
 - ◆ Only for the dynamically interesting objects.

Requirements Analysis Document Template

- 1. Introduction**
- 2. Current system**
- 3. Proposed system**
 - 3.1 Overview*
 - 3.2 Functional requirements*
 - 3.3 Nonfunctional requirements*
 - 3.4 Constraints ("Pseudo requirements")*
 - 3.5 System models*
 - 3.5.1 Scenarios
 - 3.5.2 Use case model
 - 3.5.3 Object model
 - 3.5.3.1 Data dictionary
 - 3.5.3.2 Class diagrams
 - 3.5.4 Dynamic models
 - 3.5.5 User interface
- 4. Glossary**

- ◆ Goal
 - ◆ Discover problems, incompleteness and inconsistencies in the elicited requirements
 - ◆ Feed back to the stakeholders to resolve them through the negotiation process
- ◆ Interleaved with elicitation as problems are discovered when the requirements are elicited
- ◆ A problem checklist may be used to support analysis. Each requirement may be assessed against the checklist

- ◆ Premature design
 - ◆ Is premature design or implementation information included?
- ◆ Combined requirements
 - ◆ Is always a single requirement described or could it be broken down into several different requirements?
- ◆ Unnecessary requirements
 - ◆ Is the requirement 'gold plating'? That is, is the requirement a cosmetic addition to the system which is not really necessary.
- ◆ Use of non-standard hardware
 - ◆ Does the requirement mean that non-standard hardware or software must be used?
- ◆ Conformance with business goals
 - ◆ Is the requirement consistent with the business goals defined in the introduction to the requirements document?
- ◆ Requirements ambiguity
 - ◆ Is the requirement ambiguous i.e. could it be read in different ways by different people? What are the possible interpretations of the requirement?
- ◆ Requirements realism
 - ◆ Is the requirement realistic given the technology which will be used to implement the system?
- ◆ Requirements testability
 - ◆ Is the requirement testable?

- ◆ High priority (“Core requirements”)
 - ◆ Must be addressed during analysis, design, and implementation.
 - ◆ A high-priority feature must be demonstrated successfully during client acceptance.
- ◆ Medium priority (“Optional requirements”)
 - ◆ Must be addressed during analysis and design.
 - ◆ Usually implemented and demonstrated in the second iteration of the system development.
- ◆ Low priority (“Fancy requirements”)
 - ◆ Must be addressed during analysis (“very visionary scenarios”).
 - ◆ Illustrates how the system is going to be used in the future if not yet available technology enablers are available

- ◆ An information stage where the nature of the problems associated with a requirement is explained.
- ◆ A discussion stage where the stakeholders involved discuss how these problems might be resolved.
 - ◆ All stakeholders with an interest in the requirement should be given the opportunity to comment. Priorities may be assigned to requirements at this stage.
- ◆ A resolution stage where actions concerning the requirement are agreed.
 - ◆ These actions might be to *delete the requirement*, to suggest specific *modifications* to the requirement or to *elicit further information* about the requirement.

Requirements Validation



Requirements Validation

Overview

- ◆ Goal: Certify that the requirements document is an acceptable description of the system to be implemented
- ◆ Requirements validation is a critical step in the development process, usually after requirements analysis.
- ◆ Also at delivery (client acceptance test).

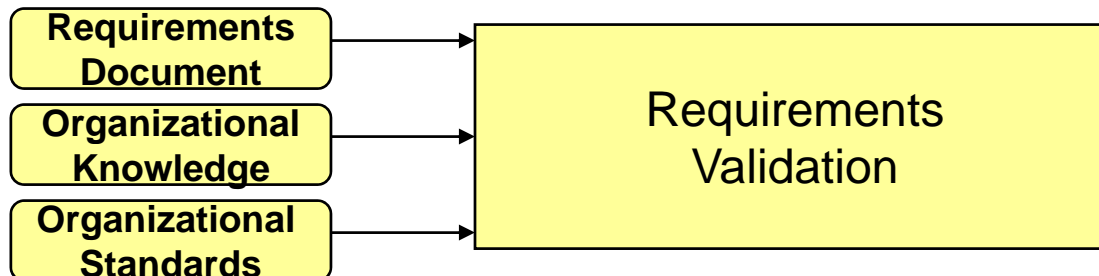
- ◆ Problem
 - ◆ No existing document which can be a basis for validation
 - ◆ Requirements change very fast during requirements elicitation

Analysis vs. Validation Differentiation

- ◆ Analysis (Verification) works with raw requirements as elicited from the system stakeholders
 - ◆ Usually incomplete and expressed in an informal and unstructured way
 - ◆ Incorporation of stakeholder is important
 - ◆ “Have we got the right requirements”
- ◆ Validation works with a final draft of the requirements document i.e. with negotiated and agreed requirements
 - ◆ All known incompleteness and inconsistency are removed
 - ◆ Validation process is more concerned with the way in which the requirements are described
 - ◆ Stakeholders not necessarily important
 - ◆ “Have we got the requirements right”

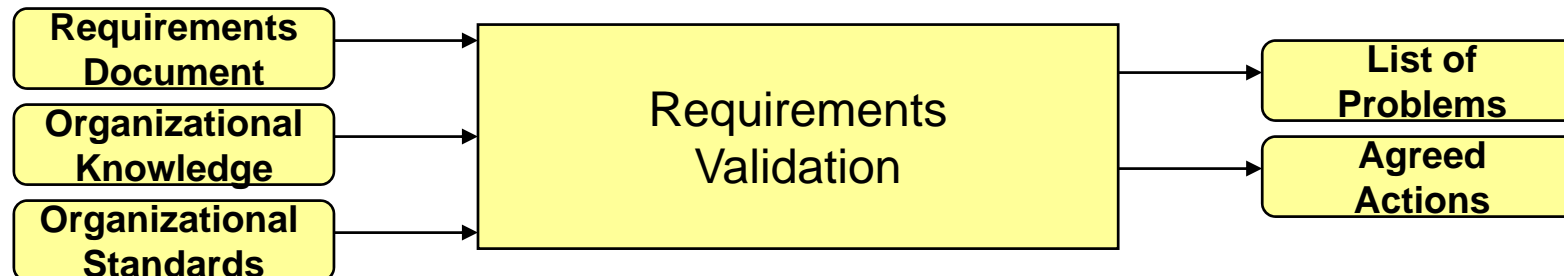
Validation inputs

- ◆ Requirements document
 - ◆ Should be a complete version of the document, not an unfinished draft
 - ◆ Formatted and organized according to organizational standards
- ◆ Organizational knowledge
 - ◆ Knowledge, often implicit, of the organization which may be used to judge the realism of the requirements
- ◆ Organizational standards
 - ◆ Local standards e.g. for the organization of the requirements document



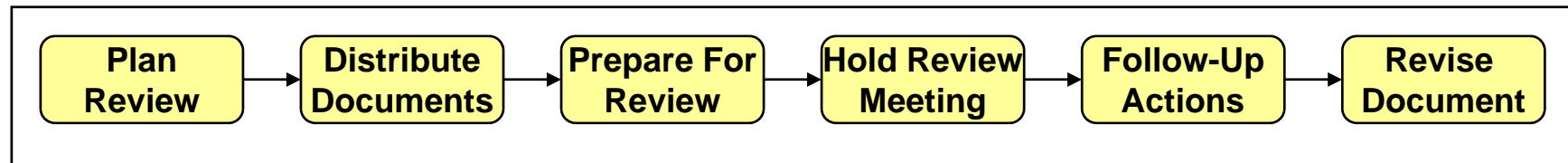
Validation outputs

- ◆ Problem list
 - ◆ List of discovered problems in the requirements document
- ◆ Agreed actions
 - ◆ List of agreed actions in response to requirements problems
 - ◆ Some problems may have several corrective actions; some problems may have no associated actions

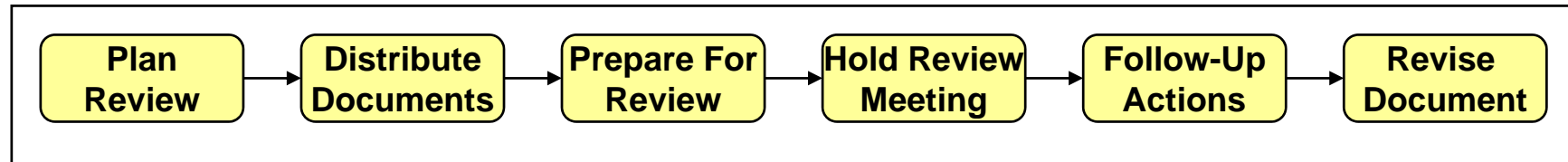


Requirements review process

- ◆ A group of people read and analyze the requirements, look for problems, meet and discuss the problems and agree on actions to address these problems



Requirements review process



- ◆ Plan review
 - ◆ The review team is selected and a time and place for the review meeting is chosen.
- ◆ Distribute documents
 - ◆ The requirements document is distributed to the review team members
- ◆ Prepare for review
 - ◆ Individual reviewers read the requirements to find conflicts, omissions, inconsistencies, deviations from standards and other problems.
- ◆ Hold review meeting
 - ◆ Individual comments and problems are discussed and a set of actions to address the problems is agreed.
- ◆ Follow-up actions
 - ◆ The chair of the review checks that the agreed actions have been carried out.
- ◆ Revise document
 - ◆ The requirements document is revised to reflect the agreed actions. At this stage, it may be accepted or it may be re-reviewed

Requirements Validation Metrics

- ◆ Completeness
 - ◆ All possible scenarios through the system are described including exceptional cases
- ◆ Consistency
 - ◆ The system does not contradict itself
- ◆ Ambiguity
 - ◆ Exactly one system is specified
- ◆ Correctness
 - ◆ Specification represents accurately the system that the client needs
- ◆ Realism
 - ◆ The system can be implemented within constraints
- ◆ Verifiability
 - ◆ Tests can be designed to demonstrate that the system fulfills requirements specification

Requirements review process

Review checklists

- ◆ Understandability
 - ◆ Can readers of the document understand what the requirements mean?
- ◆ Redundancy
 - ◆ Is information unnecessarily repeated in the requirements document?
- ◆ Completeness
 - ◆ Does the checker know of any missing requirements or is there any information missing from individual requirement descriptions?
- ◆ Ambiguity
 - ◆ Are the requirements expressed using terms which are clearly defined?
 - ◆ Do you use synonyms or homonyms?
- ◆ Organization
 - ◆ Is the document structured in a sensible way?
 - ◆ Are the descriptions of requirements organized so that related requirements are grouped?
- ◆ Consistency
 - ◆ Do the descriptions of different requirements include contradictions?
 - ◆ Are there contradictions between individual requirements and overall system requirements?
- ◆ Conformance to standards
 - ◆ Does the requirements document and individual requirements conform to defined standards?
 - ◆ Are departures from the standards, justified?
- ◆ Traceability
 - ◆ Are requirements unambiguously identified, include links to related requirements and to the reasons why these requirements have been included?

Requirements review process

Pre-review checking

- ◆ Reviews are expensive
 - ◆ involve a number of people spending time reading and checking the requirements document
- ◆ This expense can be reduced by using pre-review checking
 - ◆ person checks the document and looks for
 - ◆ straightforward problems such as missing requirements
 - ◆ lack of conformance to standards
 - ◆ typographical errors, etc.
- ◆ Document may be returned for correction or the list of problems distributed to other reviewers

Requirements review process

Review team membership

- ◆ Reviews should involve a number of stakeholders drawn from different backgrounds
 - ◆ People from different backgrounds bring different skills and knowledge to the review
 - ◆ Stakeholders feel involved in the RE process and develop an understanding of the needs of other stakeholders
- ◆ Review team should always involve at least a domain expert and an end-user

User manual development

- ◆ Rewriting requirements in a different way is a very effective validation technique.
- ◆ Writing a user manual from the requirements forces a detailed requirements analysis and thus can reveal problems with the document
- ◆ Information in the user manual
 - ◆ Description of the functionality and how it is implemented
 - ◆ Which parts of the system have not been implemented
 - ◆ How to get out of trouble
 - ◆ How to install and get started with the system
- ◆ Written in a language understandable for end-users

- ◆ Each requirement should be testable i.e. it should be possible to define tests to check whether or not that requirement has been met.
- ◆ Inventing requirements tests is an effective validation technique as missing or ambiguous information in the requirements description may make it difficult to formulate tests
- ◆ Each functional requirement should have an associated test

- ◆ What usage scenario might be used to check the requirement?
- ◆ Does the requirement, on its own, include enough information to allow a test to be defined?
- ◆ Is it possible to test the requirement using a single test or are multiple test cases required?
- ◆ Could the requirement be re-stated to make the test cases more obvious?

- ◆ The requirement's identifier
 - ◆ There should be at least one for each requirement.
- ◆ Related requirements
 - ◆ These should be referenced as the test may also be relevant to these requirements.
- ◆ Test description
 - ◆ A brief description of the test and why this is an objective requirements test. This should include system inputs and corresponding outputs.
- ◆ Requirements problems
 - ◆ A description of problems which made test definition difficult or impossible.
- ◆ Comments and recommendations
 - ◆ These are advice on how to solve requirements problems which have been discovered.

- ◆ Exclusive requirements
 - ◆ These are requirements which exclude specific behavior. For example, a requirement may state that system failures must never corrupt the system database. It is not possible to test such a requirement exhaustively.
- ◆ Some non-functional requirements
 - ◆ Some non-functional requirements, such as reliability requirements, can only be tested with a large test set.
 - ◆ The efficiency of adaptation, modification, maintenance can hardly be tested (consider interviews, observation etc. with users)

Summary

- ◆ From use cases to class diagrams
- ◆ Object and Class Identification
 - ◆ entity, boundary and control objects
 - ◆ Abbott's technique helps in object identification
- ◆ Users of class diagrams
- ◆ Dynamic modeling
 - ◆ Sequence diagrams
 - ◆ State diagrams
- ◆ Non-functional requirements analysis
- ◆ Requirements analysis model validation