Chapter 6  Requirements Modeling: Scenarios, Information, and Analysis Classes

Slides in this presentation were taken from two sources:

*Software Engineering: A Practitioner’s Approach, 7/e*
by Roger S. Pressman


*Object-Oriented Software Engineering: Using UML, Patterns, and Java, 2/e, Chapters 4 and 5*
by Bernd Bruegge and Allen H. Dutoit

The footer information identifies the source for each slide.
Requirements Analysis

- Requirements analysis
  - specifies software’s operational characteristics
  - indicates software’s interface with other system elements
  - establishes constraints that software must meet

- Requirements analysis allows the software engineer (called an analyst or modeler in this role) to:
  - elaborate on basic requirements established during earlier requirement engineering tasks
  - build models that depict user scenarios, functional activities, problem classes and their relationships, system and class behavior, and the flow of data as it is transformed.
A Bridge

These slides are designed to accompany *Software Engineering: A Practitioner’s Approach, 7/e* (McGraw-Hill, 2009). Slides copyright 2009 by Roger Pressman.
Problem Statement

- The problem statement is developed by the client as a description of the problem addressed by the system.
- Other words for problem statement:
  - Statement of Work
- A good problem statement describes
  - The current situation
  - The functionality the new system should support
  - The environment in which the system will be deployed
  - Deliverables expected by the client
  - Delivery dates
  - A set of acceptance criteria
Ingredients of a Problem Statement

♦ Current situation: The Problem to be solved
♦ Description of one or more scenarios
♦ Requirements
  ✷ Functional and Nonfunctional requirements
  ✷ Constraints (“pseudo requirements”)
♦ Project Schedule
  ✷ Major milestones that involve interaction with the client including deadline for delivery of the system
♦ Target environment
  ✷ The environment in which the delivered system has to perform a specified set of system tests
♦ Client Acceptance Criteria
  ✷ Criteria for the system tests
Current Situation: The Problem To Be Solved

♦ There is a problem in the current situation
  ♦ Examples:
    ♦ The response time when playing letter-chess is far too slow.
    ♦ I want to play Go, but cannot find players on my level.

♦ What has changed? Why address the problem now?
  ♦ There has been a change, either in the application domain or in the solution domain
  ♦ Change in the application domain
    ♦ A new function (business process) is introduced into the business
    ♦ Example: We can play highly interactive games with remote people
  ♦ Change in the solution domain
    ♦ A new solution (technology enabler) has appeared
    ♦ Example: The internet allows the creation of virtual communities.
Types of Requirements

♦ Functional requirements:
  ♦ Describe the interactions between the system and its environment independent from implementation
  ♦ Examples:
    ♦ An ARENA operator should be able to define a new game.

♦ Nonfunctional requirements:
  ♦ User visible aspects of the system not directly related to functional behavior.
  ♦ Examples:
    ♦ The response time must be less than 1 second
    ♦ The ARENA server must be available 24 hours a day

♦ Constraints (“Pseudo requirements”):
  ♦ Imposed by the client or the environment in which the system operates
    ♦ The implementation language must be Java
    ♦ ARENA must be able to dynamically interface to existing games provided by other game developers.
Rules of Thumb

- The model should focus on requirements that are visible within the problem or business domain. The level of abstraction should be relatively high.
- Each element of the analysis model should add to an overall understanding of software requirements and provide insight into the information domain, function and behavior of the system.
- Delay consideration of infrastructure and other non-functional models until design.
- Minimize coupling throughout the system.
- Be certain that the analysis model provides value to all stakeholders.
- Keep the model as simple as it can be.
Domain Analysis

Software domain analysis is the identification, analysis, and specification of common requirements from a specific application domain, typically for reuse on multiple projects within that application domain . . . [Object-oriented domain analysis is] the identification, analysis, and specification of common, reusable capabilities within a specific application domain, in terms of common objects, classes, subassemblies, and frameworks . . .

Donald Firesmith
Domain Analysis

- Define the domain to be investigated.
- Collect a representative sample of applications in the domain.
- Analyze each application in the sample.
- Develop an analysis model for the objects.
Elements of Requirements Analysis

- Scenario-based models
  e.g., use cases user stories

- Class models
  e.g., class diagrams collaboration diagrams

- Behavioral models
  e.g., state diagrams sequence diagrams

- Flow Models
  e.g., DFDs data models

These slides are designed to accompany Software Engineering: A Practitioner’s Approach, 7/e (McGraw-Hill, 2009). Slides copyright 2009 by Roger Pressman.
Scenario-Based Modeling

“[Use-cases] are simply an aid to defining what exists outside the system (actors) and what should be performed by the system (use-cases).” Ivar Jacobson

(1) What should we write about?
(2) How much should we write about it?
(3) How detailed should we make our description?
(4) How should we organize the description?
What to Write About?

- **Inception and elicitation**—provide you with the information you’ll need to begin writing use cases.
- **Requirements gathering meetings, QFD, and other requirements engineering mechanisms** are used to:
  - identify stakeholders
  - define the scope of the problem
  - specify overall operational goals
  - establish priorities
  - outline all known functional requirements, and
  - describe the things (objects) that will be manipulated by the system.
- To begin developing a set of use cases, list the functions or activities performed by a specific actor.
How Much to Write About?

- As further conversations with the stakeholders progress, the requirements gathering team develops use cases for each of the functions noted.
- In general, use cases are written first in an informal narrative fashion.
- If more formality is required, the same use case is rewritten using a structured format similar to the one proposed.
Use-Cases

- a scenario that describes a “thread of usage” for a system
- **actors** represent roles people or devices play as the system functions
- **users** can play a number of different roles for a given scenario
Developing a Use-Case

- What are the main tasks or functions that are performed by the actor?
- What system information will the actor acquire, produce or change?
- Will the actor have to inform the system about changes in the external environment?
- What information does the actor desire from the system?
- Does the actor wish to be informed about unexpected changes?
Another Use Case Example: Allocate a Resource

♦ **Actors:**
  - *Field Supervisor:* This is the official at the emergency site....
  - *Resource Allocator:* The Resource Allocator is responsible for the commitment and decommitment of the Resources managed by the FRIEND system. ...
  - *Dispatcher:* A Dispatcher enters, updates, and removes Emergency Incidents, Actions, and Requests in the system. The Dispatcher also closes Emergency Incidents.
  - *Field Officer:* Reports accidents from the Field
Another Use Case Example: Allocate a Resource

♦ Use case name: AllocateResources
♦ Participating Actors:
  ♦ Field Officer (Bob and Alice in the Scenario)
  ♦ Dispatcher (John in the Scenario)
  ♦ Resource Allocator
  ♦ Field Supervisor
♦ Entry Condition
  ♦ The Resource Allocator has selected an available resource.
  ♦ The resource is currently not allocated
♦ Flow of Events
  ♦ The Resource Allocator selects an Emergency Incident.
  ♦ The Resource is committed to the Emergency Incident.
♦ Exit Condition
  ♦ The use case terminates when the resource is committed.
  ♦ The selected Resource is now unavailable to any other Emergency Incidents or Resource Requests.
♦ Special Requirements
  ♦ The Field Supervisor is responsible for managing the Resources
Order of steps when formulating use cases

♦ First step: name the use case
  ♦ Use case name: ReportEmergency

♦ Second step: Find the actors
  ♦ Generalize the concrete names (“Bob”) to participating actors (“Field officer”)
  ♦ Participating Actors:
    ♦ Field Officer (Bob and Alice in the Scenario)
    ♦ Dispatcher (John in the Scenario)

♦ Third step: Then concentrate on the flow of events
  ♦ Use informal natural language
Use Case Associations

♦ A use case model consists of use cases and use case associations
  ♦ A use case association is a relationship between use cases
♦ Important types of use case associations: Include, Extends, Generalization
♦ Include
  ♦ A use case uses another use case (“functional decomposition”)
♦ Extends
  ♦ A use case extends another use case
♦ Generalization
  ♦ An abstract use case has different specializations
<<Include>>: Functional Decomposition

♦ Problem:
  ♦ A function in the original problem statement is too complex to be solvable immediately

♦ Solution:
  ♦ Describe the function as the aggregation of a set of simpler functions. The associated use case is decomposed into smaller use cases
<<Include>>: Reuse of Existing Functionality

- Problem:
  - There are already existing functions. How can we *reuse* them?
- Solution:
  - The *include association* from a use case A to a use case B indicates that an instance of the use case A performs all the behavior described in the use case B (“A delegates to B”)
- Example:
  - The use case “ViewMap” describes behavior that can be used by the use case “OpenIncident” (“ViewMap” is factored out)

Note: The base case cannot exist alone. It is always called with the supplier use case
**<Extend>> Association for Use Cases**

- **Problem:**
  - The functionality in the original problem statement needs to be extended.

- **Solution:**
  - An *extend association* from a use case A to a use case B indicates that use case B is an extension of use case A.

- **Example:**
  - The use case “ReportEmergency” is complete by itself, but can be extended by the use case “Help” for a specific scenario in which the user requires help.

Note: The base use case can be executed without the use case extension in extend associations.
Figure 4-12, Example of use of extend relationship.
Generalization association in use cases

- Problem:
  - You have common behavior among use cases and want to factor this out.
- Solution:
  - The generalization association among use cases factors out common behavior. The child use cases inherit the behavior and meaning of the parent use case and add or override some behavior.
- Example:
  - Consider the use case “ValidateUser”, responsible for verifying the identity of the user. The customer might require two realizations: “CheckPassword” and “CheckFingerprint”
How to Specify a Use Case (Summary)

♦ Name of Use Case
♦ Actors
  ♦ Description of Actors involved in use case)
♦ Entry condition
  ♦ “This use case starts when…”
♦ Flow of Events
  ♦ Free form, informal natural language
♦ Exit condition
  ♦ “This use cases terminates when…”
♦ Exceptions
  ♦ Describe what happens if things go wrong
♦ Special Requirements
  ♦ Nonfunctional Requirements, Constraints)
**ARENAs: The Problem**

- The Internet has enabled virtual communities
  - Groups of people sharing common of interests but who have never met each other in person. Such virtual communities can be short lived (e.g., people in a chat room or playing a multi-player game) or long lived (e.g., subscribers to a mailing list).
- Many multi-player computer games now include support for virtual communities.
  - Players can receive news about game upgrades, new game levels, announce and organize matches, and compare scores.
- Currently each game company develops such community support in each individual game.
  - Each company uses a different infrastructure, different concepts, and provides different levels of support.
- This redundancy and inconsistency leads to problems:
  - High learning curve for players joining a new community,
  - Game companies need to develop the support from scratch
  - Advertisers need to contact each individual community separately.
ARENA: The Objectives

♦ Provide a generic infrastructure for operating an arena to
  ♦ Support virtual game communities.
  ♦ Register new games
  ♦ Register new players
  ♦ Organize tournaments
  ♦ Keeping track of the players scores.

♦ Provide a framework for tournament organizers
  ♦ to customize the number and sequence of matchers and the accumulation of expert rating points.

♦ Provide a framework for game developers
  ♦ for developing new games, or for adapting existing games into the ARENA framework.

♦ Provide an infrastructure for advertisers.
Example Scenario

<table>
<thead>
<tr>
<th>Scenario name</th>
<th>organizeTicTacToTournament</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participating actor instances</td>
<td>alice:Operator, joe:LeagueOwner, bill:Spectator, mary:Player</td>
</tr>
<tr>
<td>Flow of events</td>
<td></td>
</tr>
<tr>
<td>1.</td>
<td>Joe, a friend of Alice, is a Tic Tac Toe aficionado and volunteers to organize a tournament.</td>
</tr>
<tr>
<td>2.</td>
<td>Alice registers Joe in the arena as a league owner.</td>
</tr>
<tr>
<td>3.</td>
<td>Joe first defines a Tic Tac Toe beginners league, in which any players can be admitted. This league, dedicated to Tic Tac Toe games, stipulates that tournaments played in this league will follow the knockout tournament style and “Winner Takes All” formula.</td>
</tr>
<tr>
<td>4.</td>
<td>Joe schedules the first tournament in the league for 16 players starting the next day.</td>
</tr>
<tr>
<td>5.</td>
<td>Joe announces the tournament in a variety of forums over the Web and sends mail to other Tic Tac Toe community members.</td>
</tr>
<tr>
<td>6.</td>
<td>Bill and Mary receive the E-mail notification.</td>
</tr>
<tr>
<td>7.</td>
<td>Mary is interested in playing the tournament and registers. 19 others apply.</td>
</tr>
<tr>
<td>8.</td>
<td>Joe schedules 16 players for the tournament and rejects the 4 that applied last.</td>
</tr>
<tr>
<td>9.</td>
<td>The 16 players, including Mary, receive an electronic token for entering the tournament and the time of their first match.</td>
</tr>
<tr>
<td>10.</td>
<td>Other subscribers to the Tic Tac Toe mailing list, including Bill, receive a second notice about the Tournament, including the name of the players and the schedule of matches.</td>
</tr>
<tr>
<td>11.</td>
<td>As Joe kicks off the tournament, the players have a limited amount of time to enter the match. If a player fails to show up, he loses the game.</td>
</tr>
<tr>
<td>12.</td>
<td>Mary plays her first match and wins. She advances in the tournament and is scheduled for the next match against another winner of the first round.</td>
</tr>
<tr>
<td>13.</td>
<td>After visiting the Tic Tac Toe Tournament’s home page, Bill notices Mary’s victory and decides to watch her next match. He selects the match, and sees the sequence of moves of each player as they occur. He also sees an advertisement banner at the bottom of his browser, advertising other tournaments and tic tac toe products.</td>
</tr>
<tr>
<td>14.</td>
<td>The tournament continues until the last match, at which point the winner of the tournament is declared and his league record is credited with all the points associated with the tournament.</td>
</tr>
<tr>
<td>15.</td>
<td>Also, the winner of the tournament accumulates expert rating points.</td>
</tr>
<tr>
<td>16.</td>
<td>Joe can choose to schedule more tournaments in the league, in which case, known players are notified about the date and given priority over new players.</td>
</tr>
</tbody>
</table>
Some Questions

Steps 2, 7: Different actors register with the system. In the first case, the operator registers Joe as a league owner; in the second case, a player registers herself with the system.

- Registration of users should follow the same paradigm. Who provides the registration information and how is the information reviewed, validated, and accepted?
- Client: Two processes are confused in steps 2 & 7, the registration process, during which new users (e.g., a player or a league owner) establish their identity, and the application process, during which players indicate they want to take part in a specific tournament. During the registration process, the user provides information about themselves (name, nickname, E-mail) and their interests (types of games and tournaments they want to be informed about). The information is validated by the operator. During the application process, players indicate which tournament they want to participate in. This is used by the league owner during match scheduling.
- Since the player information has already been validated by the operator, should the match scheduling be completely automated?
- Client: Yes, of course.

Step 5: Joe sends mail to the Tic Tac Toe community members:

- Does ARENA provide the opportunity to users to subscribe to individual mailing lists?
- Client: Yes. There should be mailing lists for announcing new games, new leagues, new tournaments, etc.
- Does ARENA store a user profile (e.g., game watched, games played, interests specified by a user survey) for the purpose of advertisement?
- Client: Yes, but users should still be able to register without completing a user survey, if they want to. They should be encouraged to enter the survey, but this should not prevent them from entering. They will be exposed to advertisements anyway.
- Should the profile be used to automatically subscribe to mailing lists?
- Client: No, we think users in our community would prefer having complete control over their mailing list subscriptions. Guessing subscriptions would not give them the impression they are in control.

Step 13: Bill browses match statistics and decides to see the next match in real time.

- How are players identified to the spectators? By real name, by E-mail, by nickname?
- Client: This should be left to the user during the registration.
- Can a spectator replay old matches?
- Client: Games should be able to provide this ability, but some games (e.g., real-time, 3D action games) may choose not to do so because of resource constraints.
- ARENA should support real-time games?
- Client: Yes, these represent the largest share of our market. In general, ARENA should support as broad a range of games as possible.
- ...
Figure 4-20, High-level scenarios identified for ARENA.
## Working glossary for ARENA

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Game</td>
<td>A Game is a competition among a number of Players that is conducted according to a set of rules. In ARENA, the term Game refers to a piece of software that enforces the set of rules, tracks the progress of each Player, and decides the winner. For example, tic tac toe and chess are Games.</td>
</tr>
<tr>
<td>Match</td>
<td>A Match is a contest between two or more Players following the rules of a Game. The outcome of a Match can be a single winner and a set of losers or a tie (in which there are no winners or losers). Some Games may disallow ties.</td>
</tr>
<tr>
<td>Tournament</td>
<td>A Tournament is a series of Matches among a set of Players. Tournaments end with a single winner. The way Players accumulate points and Matches are scheduled is dictated by the League in which the Tournament is organized.</td>
</tr>
<tr>
<td>League</td>
<td>A League represents a community for running Tournaments. A League is associated with a specific Game and TournamentStyle. Players registered with the League accumulate points according to the ExpertRating defined in the League. For example, a novice chess League has a different ExpertRating formula than an expert League.</td>
</tr>
<tr>
<td>TournamentStyle</td>
<td>The TournamentStyle defines the number of Matches and their sequence for a given set of Players. For example, Players face all other Players in the Tournament exactly once in a round robin TournamentStyle.</td>
</tr>
</tbody>
</table>
Figure 4-21, High-level use cases identified for ARENA.
<table>
<thead>
<tr>
<th>Register</th>
<th>Anonymous users register with an Arena for a Player or a League-Owner account. User accounts are required before applying for a tournament or organizing a league. Spectators do not need accounts.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ManageUserAccounts</td>
<td>The Operator accepts registrations from LeagueOwners and for Players, cancels existing accounts, and interacts with users about extending their accounts.</td>
</tr>
<tr>
<td>ManageComponents</td>
<td>The Operator installs new games and defines new tournament styles (generalizes defineKnockOutStyle and installTicTacToeGame).</td>
</tr>
<tr>
<td>DefineLeague</td>
<td>The LeagueOwner defines a new league (generalizes the first steps of the scenario organizeTicTacToeTournament).</td>
</tr>
<tr>
<td>OrganizeTournament</td>
<td>The LeagueOwner creates and announces a new tournament, accepts player applications, schedules matches, and kicks off the tournament. During the tournament, players play matches and spectators follow matches. At the end of the tournament, players are credited with points (generalizes the scenario organizeTicTacToeTournament).</td>
</tr>
<tr>
<td>ManageAdvertisements</td>
<td>The Advertiser uploads banners and sponsors league or tournaments (generalizes sponsorTicTacToeBeginnersLeague).</td>
</tr>
<tr>
<td>ManageOwnProfile</td>
<td>The Players manage their subscriptions to mailing lists and answer a marketing survey.</td>
</tr>
<tr>
<td>BrowseTournamentHistory</td>
<td>Spectators examine tournament statistics and player statistics, and replay matches that have already been concluded (generalizes the scenario analyzeTicTacToeTournament).</td>
</tr>
</tbody>
</table>
## Organize Tournament Use Case

<table>
<thead>
<tr>
<th>Use case name</th>
<th>OrganizeTournament</th>
</tr>
</thead>
</table>
| **Participating actors** | Initiated by LeagueOwner  
Communicates with Advertiser, Player, and Spectator |
| **Flow of events** | 1. The LeagueOwner creates a Tournament, solicits sponsorships from Advertisers, and announces the Tournament (include use case AnnounceTournament).  
2. The Players apply for the Tournament (include use case ApplyForTournament).  
3. The LeagueOwner processes the Player applications and assigns them to matches (include use case ProcessApplications).  
4. The LeagueOwner kicks off the Tournament (include use case KickoffTournament).  
5. The Players compete in the matches as scheduled and Spectators view the matches (include use case PlayMatch).  
6. The LeagueOwner declares the winner and archives the Tournament (include use case ArchiveTournament). |
| **Entry condition** | • The LeagueOwner is logged into ARENA. |
| **Exit conditions** | • The LeagueOwner archived a new tournament in the ARENA archive and the winner has accumulated new points in the league, OR  
• The LeagueOwner cancelled the tournament and the players’ standing in the league is unchanged. |
Figure 4-23, Detailed use cases refining the OrganizeTournament high-level use case
Activity Diagram

Supplements the use case by providing a graphical representation of the flow of interaction within a specific scenario.
Swimlane Diagrams

*Allows the modeler to represent the flow of activities described by the use-case and at the same time indicate which actor (if there are multiple actors involved in a specific use-case) or analysis class has responsibility for the action described by an activity rectangle.*

These slides are designed to accompany *Software Engineering: A Practitioner’s Approach, 7/e* (McGraw-Hill, 2009). Slides copyright 2009 by Roger Pressman.
Data Modeling

- examines data objects independently of processing
- focuses attention on the data domain
- creates a model at the customer’s level of abstraction
- indicates how data objects relate to one another
What is a Data Object?

- A representation of almost any composite information that must be understood by software.
  - *composite information*—something that has a number of different properties or attributes
- Can be an *external entity* (e.g., anything that produces or consumes information), a *thing* (e.g., a report or a display), an *occurrence* (e.g., a telephone call) or *event* (e.g., an alarm), a *role* (e.g., salesperson), an *organizational unit* (e.g., accounting department), a *place* (e.g., a warehouse), or a *structure* (e.g., a file).
- The description of the data object incorporates the data object and all of its attributes.
- A data object encapsulates data only—there is no reference within a data object to operations that act on the data.
Data Objects and Attributes

A data object contains a set of attributes that act as an aspect, quality, characteristic, or descriptor of the object.

<table>
<thead>
<tr>
<th>object: automobile</th>
</tr>
</thead>
<tbody>
<tr>
<td>attributes:</td>
</tr>
<tr>
<td>make</td>
</tr>
<tr>
<td>model</td>
</tr>
<tr>
<td>body type</td>
</tr>
<tr>
<td>price</td>
</tr>
<tr>
<td>options code</td>
</tr>
</tbody>
</table>
What is a Relationship?

- Data objects are connected to one another in different ways.
  - A connection is established between **person** and **car** because the two objects are related.
    - A person *owns* a car
    - A person *is insured to drive* a car
- The relationships *owns* and *insured to drive* define the relevant connections between **person** and **car**.
- Several instances of a relationship can exist
- Objects can be related in many different ways
ERD Notation

**One common form:**

![Diagram showing ERD notation with objects and relationships marked as (0, m) and (1, 1).]

**Another common form:**

![Diagram showing ERD notation with objects and relationships marked as (0, m) and (1, 1).]

These slides are designed to accompany *Software Engineering: A Practitioner’s Approach, 7/e* (McGraw-Hill, 2009). Slides copyright 2009 by Roger Pressman.
Building an ERD

- **Level 1**—model all data objects (entities) and their “connections” to one another
- **Level 2**—model all entities and relationships
- **Level 3**—model all entities, relationships, and the attributes that provide further depth
The ERD: An Example

Customer (1,1) places (1,m) request for service (1,1)

places (1,m) generates (1,n) work order (1,1)

standard task table (1,1) selected from (1,w) work tasks (1,w)

work tasks (1,w) consists of (1,1) materials (1,i)

lists (1,i)
Class-Based Modeling

- Class-based modeling represents:
  - objects that the system will manipulate
  - operations (also called methods or services) that will be applied to the objects to effect the manipulation
  - relationships (some hierarchical) between the objects
  - collaborations that occur between the classes that are defined.

- The elements of a class-based model include classes and objects, attributes, operations, CRC models, collaboration diagrams and packages.
Identifying Analysis Classes

- Examining the usage scenarios developed as part of the requirements model and perform a "grammatical parse" [Abb83]
  - Classes are determined by underlining each noun or noun phrase and entering it into a simple table.
  - Synonyms should be noted.
  - If the class (noun) is required to implement a solution, then it is part of the solution space; otherwise, if a class is necessary only to describe a solution, it is part of the problem space.
- But what should we look for once all of the nouns have been isolated?
## Mapping parts of speech to object model components

[Abbott, 1983]

<table>
<thead>
<tr>
<th>Part of speech</th>
<th>Model component</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proper noun</td>
<td>object</td>
<td>Jim Smith</td>
</tr>
<tr>
<td>Improper noun</td>
<td>class</td>
<td>Toy, doll</td>
</tr>
<tr>
<td>Doing verb</td>
<td>method</td>
<td>Buy, recommend</td>
</tr>
<tr>
<td>being verb</td>
<td>inheritance</td>
<td>is-a (kind-of)</td>
</tr>
<tr>
<td>having verb</td>
<td>aggregation</td>
<td>has an</td>
</tr>
<tr>
<td>modal verb</td>
<td>constraint</td>
<td>must be</td>
</tr>
<tr>
<td>adjective</td>
<td>attribute</td>
<td>3 years old</td>
</tr>
<tr>
<td>transitive verb</td>
<td>method</td>
<td>enter</td>
</tr>
<tr>
<td>intransitive verb</td>
<td>method (event)</td>
<td>depends on</td>
</tr>
</tbody>
</table>
Another Example

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.
♦ He tries a videogame, which uses a data glove and a head-mounted display. He likes it.

Is this a good use Case?

An assistant helps him.
The suitability of the game depends on the age of the child.
His daughter is only 3 years old.
The assistant recommends another type of toy, namely the boardgame “Monopoly”.

“Monopoly” is probably a left over from the scenario

The use case should terminate with the customer leaving the store
### Textual Analysis using Abbot‘s technique

<table>
<thead>
<tr>
<th>Example</th>
<th>Grammatical construct</th>
<th>UML Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>“Monopoly”</td>
<td>Concrete Person, Thing</td>
<td>Object</td>
</tr>
<tr>
<td>“toy”</td>
<td>noun</td>
<td>class</td>
</tr>
<tr>
<td>&quot;3 years old&quot;</td>
<td>Adjective</td>
<td>Attribute</td>
</tr>
<tr>
<td>“enters”</td>
<td>verb</td>
<td>Operation</td>
</tr>
<tr>
<td>“depends on….”</td>
<td>Intransitive verb</td>
<td>Operation (Event)</td>
</tr>
<tr>
<td>“is a&quot; ,“either..or&quot;,</td>
<td>Classifying verb</td>
<td>Inheritance</td>
</tr>
<tr>
<td>“kind of…”</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;Has a &quot;, “consists of&quot;</td>
<td>Possessive Verb</td>
<td>Aggregation</td>
</tr>
<tr>
<td>“must be&quot;, “less than…”</td>
<td>modal Verb</td>
<td>Constraint</td>
</tr>
</tbody>
</table>
Identifying boundary objects

### Heuristics for identifying boundary objects

- Identify user interface controls that the user needs to initiate the use case (e.g., ReportEmergencyButton).
- Identify forms the users needs to enter data into the system (e.g., EmergencyReportForm).
- Identify notices and messages the system uses to respond to the user (e.g., AcknowledgmentNotice).
- When multiple actors are involved in a use case, identify actor terminals (e.g., DispatcherStation) to refer to the user interface under consideration.
- Do not model the visual aspects of the interface with boundary objects (user mock-ups are better suited for that).
- *Always* use the end user’s terms for describing interfaces; do not use terms from the solution or implementation domains.
Identifying control and entity objects

Heuristics for identifying control objects

- Identify one control object per use case.
- Identify one control object per actor in the use case.
- The life span of a control object should cover the extent of the use case or the extent of a user session. If it is difficult to identify the beginning and the end of a control object activation, the corresponding use case probably does not have well-defined entry and exit conditions.

Heuristics for identifying entity objects

- Terms that developers or users need to clarify in order to understand the use case
- Recurring nouns in the use cases (e.g., Incident)
- Real-world entities that the system needs to track (e.g., FieldOfficer, Dispatcher, Resource)
- Real-world activities that the system needs to track (e.g., EmergencyOperationsPlan)
- Data sources or sinks (e.g., Printer).
Identifying attributes and methods of a class

Heuristics for identifying attributes

- Examine possessive phrases.
- Represent stored state as an attribute of the entity object.
- Describe each attribute.
- Do not represent an attribute as an object; use an association instead (see Section 5.4.6).
- Do not waste time describing fine details before the object structure is stable.

- Identifying methods
  - Look at verbs in the problem statement
  - Look at interactions between objects in the use case scenarios
  - Look at interactions in the sequence diagrams
  - Use prior knowledge of the problem domain
Identifying associations between classes

- Inheritance is the strongest form of association; it is based on a “kind of” relationship that is easy to identify
- Aggregation is the next strongest form of association; it is based on a “part of” relationship
- A strong form of aggregation is composition where the “part” uniquely belongs to the “whole”
- Other associations are more difficult to find

Heuristics for identifying associations

- Examine verb phrases.
- Name associations and roles precisely.
- Use qualifiers as often as possible to identify namespaces and key attributes.
- Eliminate any association that can be derived from other associations.
- Do not worry about multiplicity until the set of associations is stable.
- Too many associations make a model unreadable.
Manifestations of Analysis Classes

- *Analysis classes* manifest themselves in one of the following ways:
  - *External entities* (e.g., other systems, devices, people) that produce or consume information
  - *Things* (e.g., reports, displays, letters, signals) that are part of the information domain for the problem
  - *Occurrences or events* (e.g., a property transfer or the completion of a series of robot movements) that occur within the context of system operation
  - *Roles* (e.g., manager, engineer, salesperson) played by people who interact with the system
  - *Organizational units* (e.g., division, group, team) that are relevant to an application
  - *Places* (e.g., manufacturing floor or loading dock) that establish the context of the problem and the overall function
  - *Structures* (e.g., sensors, four-wheeled vehicles, or computers) that define a class of objects or related classes of objects
Potential Classes

- **Retained information.** The potential class will be useful during analysis only if information about it must be remembered so that the system can function.

- **Needed services.** The potential class must have a set of identifiable operations that can change the value of its attributes in some way.

- **Multiple attributes.** During requirement analysis, the focus should be on "major" information; a class with a single attribute may, in fact, be useful during design, but is probably better represented as an attribute of another class during the analysis activity.

- **Common attributes.** A set of attributes can be defined for the potential class and these attributes apply to all instances of the class.

- **Common operations.** A set of operations can be defined for the potential class and these operations apply to all instances of the class.

- **Essential requirements.** External entities that appear in the problem space and produce or consume information essential to the operation of any solution for the system will almost always be defined as classes in the requirements model.
Defining Attributes

- *Attributes* describe a class that has been selected for inclusion in the analysis model.
- build two different classes for professional baseball players
  - **For Playing Statistics software:** name, position, batting average, fielding percentage, years played, and games played might be relevant
  - **For Pension Fund software:** average salary, credit toward full vesting, pension plan options chosen, mailing address, and the like.
Defining Operations

- Do a grammatical parse of a processing narrative and look at the verbs
- Operations can be divided into four broad categories:
  - (1) operations that manipulate data in some way (e.g., adding, deleting, reformatting, selecting)
  - (2) operations that perform a computation
  - (3) operations that inquire about the state of an object, and
  - (4) operations that monitor an object for the occurrence of a controlling event.
CRC Models

- **Class-responsibility-collaborator (CRC) modeling** [Wir90] provides a simple means for identifying and organizing the classes that are relevant to system or product requirements. Ambler [Amb95] describes CRC modeling in the following way:

  - A CRC model is really a collection of standard index cards that represent classes. The cards are divided into three sections. Along the top of the card you write the name of the class. In the body of the card you list the class responsibilities on the left and the collaborators on the right.
CRC Modeling

Class: FloorPlan

Description:

Responsibility: Collaborator:
- defines floor plan name/type
- manages floor plan positioning
- scales floor plan for display
- incorporates walls, doors and windows
- shows position of video cameras
  Wall
  Camera
Class Types

- **Entity classes**, also called *model* or *business classes*, are extracted directly from the statement of the problem (e.g., FloorPlan and Sensor).

- **Boundary classes** are used to create the interface (e.g., interactive screen or printed reports) that the user sees and interacts with as the software is used.

- **Controller classes** manage a “unit of work” [UML03] from start to finish. That is, controller classes can be designed to manage:
  - the creation or update of entity objects;
  - the instantiation of boundary objects as they obtain information from entity objects;
  - complex communication between sets of objects;
  - validation of data communicated between objects or between the user and the application.
Responsibilities

- System intelligence should be distributed across classes to best address the needs of the problem.
- Each responsibility should be stated as generally as possible.
- Information and the behavior related to it should reside within the same class.
- Information about one thing should be localized with a single class, not distributed across multiple classes.
- Responsibilities should be shared among related classes, when appropriate.
Collaborations

- Classes fulfill their responsibilities in one of two ways:
  - A class can use its own operations to manipulate its own attributes, thereby fulfilling a particular responsibility, or
  - a class can collaborate with other classes.
- Collaborations identify relationships between classes
- Collaborations are identified by determining whether a class can fulfill each responsibility itself
- three different generic relationships between classes [WIR90]:
  - the *is-part-of* relationship
  - the *has-knowledge-of* relationship
  - the *depends-upon* relationship
Composite Aggregate Class

These slides are designed to accompany Software Engineering: A Practitioner’s Approach, 7/e (McGraw-Hill, 2009). Slides copyright 2009 by Roger Pressman.
Associations and Dependencies

- Two analysis classes are often related to one another in some fashion
  - In UML these relationships are called associations
  - Associations can be refined by indicating multiplicity (the term cardinality is used in data modeling)
- In many instances, a client-server relationship exists between two analysis classes.
  - In such cases, a client-class depends on the server-class in some way and a dependency relationship is established
Multiplicity

These slides are designed to accompany Software Engineering: A Practitioner’s Approach, 7/e (McGraw-Hill, 2009). Slides copyright 2009 by Roger Pressman.
Dependencies

These slides are designed to accompany Software Engineering: A Practitioner’s Approach, 7/e (McGraw-Hill, 2009). Slides copyright 2009 by Roger Pressman.
Analysis Packages

- Various elements of the analysis model (e.g., use-cases, analysis classes) are categorized in a manner that packages them as a grouping.
- The plus sign preceding the analysis class name in each package indicates that the classes have public visibility and are therefore accessible from other packages.
- Other symbols can precede an element within a package. A minus sign indicates that an element is hidden from all other packages and a # symbol indicates that an element is accessible only to packages contained within a given package.
Analysis Packages
Reviewing the CRC Model

- All participants in the review (of the CRC model) are given a subset of the CRC model index cards.
  - Cards that collaborate should be separated (i.e., no reviewer should have two cards that collaborate).
- All use-case scenarios (and corresponding use-case diagrams) should be organized into categories.
- The review leader reads the use-case deliberately.
  - As the review leader comes to a named object, she passes a token to the person holding the corresponding class index card.
  - When the token is passed, the holder of the class card is asked to describe the responsibilities noted on the card.
    - The group determines whether one (or more) of the responsibilities satisfies the use-case requirement.
- If the responsibilities and collaborations noted on the index cards cannot accommodate the use-case, modifications are made to the cards.
  - This may include the definition of new classes (and corresponding CRC index cards) or the specification of new or revised responsibilities or collaborations on existing cards.