Chapter 4, Requirements Elicitation
Dealing with Complexity

♦ Three ways to deal with complexity:
  ◆ Abstraction
  ◆ Decomposition (Technique: Divide and conquer)
  ◆ Hierarchy (Technique: Layering)

♦ Two ways to deal with decomposition:
  ◆ Object-orientation and functional decomposition
  ◆ Functional decomposition leads to unmaintainable code
  ◆ Depending on the purpose of the system, different objects can be found

♦ What is the right way?
  ◆ Start with a description of the functionality (Use case model). Then proceed by finding objects (object model).

♦ What activities and models are needed?
  ◆ This leads us to the software lifecycle
Software Lifecycle Definition

♦ Software lifecycle:
  ♦ Set of activities and their relationships to each other to support the development of a software system

♦ Typical Lifecycle questions:
  ♦ Which activities should I select for the software project?
  ♦ What are the dependencies between activities?
  ♦ How should I schedule the activities?
  ♦ What is the result of an activity
Example: Selection of Software Lifecycle Activities for a specific project

The Hacker knows only one activity

![Diagram showing the selection of software lifecycle activities]

Activities used in software engineering

- Requirements Elicitation
- Analysis
- System Design
- Object Design
- Implementation
- Testing

Each activity produces one or more models
Software Lifecycle Activities
Requirements Elicitation

♦ Very challenging activity
♦ 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
• Identifying actors. During this activity, developers identify the different types of users the future system will support.

• Identifying scenarios. During this activity, developers observe users and develop a set of detailed scenarios for typical functionality provided by the future system. Scenarios are concrete examples of the future system in use. Developers use these scenarios to communicate with the user and deepen their understanding of the application domain.

• Identifying use cases. Once developers and users agree on a set of scenarios, developers derive from the scenarios a set of use cases that completely represent the future system. Whereas scenarios are concrete examples illustrating a single case, use cases are abstractions describing all possible cases. When describing use cases, developers determine the scope of the system.

• Refining use cases. During this activity, developers ensure that the requirements specification is complete by detailing each use case and describing the behavior of the system in the presence of errors and exceptional conditions.

• Identifying relationships among use cases. During this activity, developers identify dependencies among use cases. They also consolidate the use case model by factoring out common functionality. This ensures that the requirements specification is consistent.

• Identifying nonfunctional requirements. During this activity, developers, users, and clients agree on aspects that are visible to the user, but not directly related to functionality. These include constraints on the performance of the system, its documentation, the resources it consumes, its security, and its quality.
Figure 4-1, Products of requirements elicitation and analysis.
System Specification vs Analysis Model

♦ Both models focus on the requirements from the user’s view of the system.

♦ System specification uses natural language (derived from the problem statement)

♦ The analysis model uses formal or semi-formal notation (for example, a graphical language like UML)

♦ The starting point is the problem statement
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 can 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.
What is usually not in the requirements?

- System structure, implementation technology
- Development methodology
- Development environment
- Implementation language
- Reusability

- It is desirable that none of these above are constrained by the client. Fight for it!
Requirements Validation

- Requirements validation is a critical step in the development process, usually after requirements engineering or requirements analysis. Also at delivery (client acceptance test).

- Requirements validation criteria:
  - Correctness:
    - The requirements represent the client’s view.
  - Completeness:
    - All possible scenarios, in which the system can be used, are described, including exceptional behavior by the user or the system
  - Consistency:
    - There are functional or nonfunctional requirements that contradict each other
  - Realism:
    - Requirements can be implemented and delivered
  - Traceability:
    - Each system function can be traced to a corresponding set of functional requirements
Example Problems and Solutions

**Complete**—All features of interest are described by requirements.

*Example of incompleteness:* The SatWatch specification does not specify the boundary behavior when the user is standing within GPS accuracy limitations of a state's boundary.

*Solution:* Add a functional requirement stating that the time depicted by SatWatch should not change more often than once every 5 minutes.

**Consistent**—No two requirements of the specification contradict each other.

*Example of inconsistency:* A watch that does not contain any software faults need not provide an upgrade mechanism for downloading new versions of the software.

*Solution:* Revise one of the conflicting requirements from the model (e.g., rephrase the requirement about the watch not containing any faults, as it is not verifiable anyway).

**Unambiguous**—A requirement cannot be interpreted in two mutually exclusive ways.

*Example of ambiguity:* The SatWatch specification refers to time zones and political boundaries. Does the SatWatch deal with daylight saving time or not?

*Solution:* Clarify the ambiguous concept to select one of the mutually exclusive phenomena (e.g., add a requirement that SatWatch should deal with daylight saving time).

**Correct**—The requirements describe the features of the system and environment of interest to the client and the developer, but do not describe other unintended features.

*Example of fault:* There are more than 24 time zones. Several countries and territories (e.g., India) are half an hour ahead of a neighboring time zone.
Types of Requirements Elicitation

♦ Greenfield Engineering
  ♦ Development starts from scratch, no prior system exists, the requirements are extracted from the end users and the client
  ♦ Triggered by user needs
  ♦ Example: Develop a game from scratch: Asteroids

♦ Re-engineering
  ♦ Re-design and/or re-implementation of an existing system using newer technology
  ♦ Triggered by technology enabler
  ♦ Example: Reengineering an existing game

♦ Interface Engineering
  ♦ Provide the services of an existing system in a new environment
  ♦ Triggered by technology enabler or new market needs
  ♦ Example: Interface to an existing game (Bumpers)
Scenarios

♦ “A narrative description of what people do and experience as they try to make use of computer systems and applications” [M. Carrol, Scenario-based Design, Wiley, 1995]

♦ A concrete, focused, informal description of a single feature of the system used by a single actor.

♦ Scenarios can have many different uses during the software lifecycle
  ❖ Requirements Elicitation: As-is scenario, visionary scenario
  ❖ Client Acceptance Test: Evaluation scenario
  ❖ System Deployment: Training scenario.
Types of Scenarios

♦ As-is scenario:
  ♦ Used in describing a current situation. Usually used in re-engineering projects. The user describes the system.
  ♦ Example: Description of Letter-Chess

♦ Visionary scenario:
  ♦ Used to describe a future system. Usually used in greenfield engineering and reengineering projects.
  ♦ Can often not be done by the user or developer alone
  ♦ Example: Description of an interactive internet-based Tic Tac Toe game tournament.

♦ Evaluation scenario:
  ♦ User tasks against which the system is to be evaluated.
  ♦ Example: Four users (two novice, two experts) play in a TicTac Toe tournament in ARENA.

♦ Training scenario:
  ♦ Step by step instructions that guide a novice user through a system
  ♦ Example: How to play Tic Tac Toe in the ARENA Game Framework.
How do we find scenarios?

♦ Don’t expect the client to be verbal if the system does not exist (greenfield engineering)
♦ Don’t wait for information even if the system exists
♦ Engage in a dialectic approach (evolutionary, incremental engineering)
  ♦ You help the client to formulate the requirements
  ♦ The client helps you to understand the requirements
  ♦ The requirements evolve while the scenarios are being developed
Heuristics for finding Scenarios

♦ Ask yourself or the client the following questions:
  ♦ 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?
  ♦ What changes or events will the actor of the system need to be informed about?

♦ However, don’t rely on questionnaires alone.

♦ Insist on task observation if the system already exists (interface engineering or reengineering)
  ♦ Ask to speak to the end user, not just to the software contractor
  ♦ Expect resistance and try to overcome it
**Example: Accident Management System**

- What needs to be done to report a “Cat in a Tree” incident?
- What do you need to do if a person reports “Warehouse on Fire?”
- Who is involved in reporting an incident?
- What does the system do, if no police cars are available? If the police car has an accident on the way to the “cat in a tree” incident?
- What do you need to do if the “Cat in the Tree” turns into a “Grandma has fallen from the Ladder”?
- Can the system cope with a simultaneous incident report “Warehouse on Fire?”
**Scenario Example: Warehouse on Fire**

<table>
<thead>
<tr>
<th>Scenario name</th>
<th>warehouseOnFire</th>
</tr>
</thead>
</table>
| Participating actor instances | bob, alice; FieldOfficer  
|                      | john: Dispatcher  |
| Flow of events  | 1. Bob, driving down main street in his patrol car, notices smoke coming out of a warehouse. His partner, Alice, activates the “Report Emergency” function from her FRIEND laptop. 
2. Alice enters the address of the building, a brief description of its location (i.e., northwest corner), and an emergency level. In addition to a fire unit, she requests several paramedic units on the scene, given that the area appears to be relatively busy. She confirms her input and waits for an acknowledgment. 
3. John, the Dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and two paramedic units to the Incident site and sends their estimated arrival time (ETA) to Alice. 
4. Alice receives the acknowledgment and the ETA. |

**Figure 4-6** warehouseOnFire scenario for the ReportEmergency use case.
Observations about Warehouse on Fire Scenario

♦ Concrete scenario
  - Describes a single instance of reporting a fire incident.
  - Does not describe all possible situations in which a fire can be reported.

♦ Participating actors
  - Bob, Alice and John
Next goal, after the scenarios are formulated:

♦ Find all the use cases in the scenario that specifies all possible instances of how to report a fire
  ♦ Example: “Report Emergency “ in the first paragraph of the scenario is a candidate for a use case

♦ Describe each of these use cases in more detail
  ♦ Participating actors
  ♦ Describe the Entry Condition
  ♦ Describe the Flow of Events
  ♦ Describe the Exit Condition
  ♦ Describe Exceptions
  ♦ Describe Special Requirements (Constraints, Nonfunctional Requirements)
Use Cases

♦ A use case is a flow of events in the system, including interaction with actors
♦ It is initiated by an actor
♦ Each use case has a name
♦ Each use case has a termination condition
♦ Graphical Notation: An oval with the name of the use case

ReportEmergency

Use Case Model: The set of all use cases specifying the complete functionality of the system
Example: Use Case Model for Incident Management

Diagram:

- FieldOfficer
- Dispatcher
- OpenIncident
- ReportEmergency
- AllocateResources
Heuristics: How do I find use cases?

♦ Select a narrow vertical slice of the system (i.e. one scenario)
  ♦ Discuss it in detail with the user to understand the user’s preferred style of interaction
♦ Select a horizontal slice (i.e. many scenarios) to define the scope of the system.
  ♦ Discuss the scope with the user
♦ Use illustrative prototypes (mock-ups) as visual support
♦ Find out what the user does
  ♦ Task observation (Good)
  ♦ Questionnaires (Bad)
Use Case Example: ReportEmergency

♦ Use case name: ReportEmergency
♦ Participating Actors:
  ♦ Field Officer (Bob and Alice in the Scenario)
  ♦ Dispatcher (John in the Scenario)
♦ Exceptions:
  ♦ The FieldOfficer is notified immediately if the connection between her terminal and the central is lost.
  ♦ The Dispatcher is notified immediately if the connection between any logged in FieldOfficer and the central is lost.
♦ Flow of Events: on next slide.
♦ Special Requirements:
  ♦ The FieldOfficer’s report is acknowledged within 30 seconds. The selected response arrives no later than 30 seconds after it is sent by the Dispatcher.
Use Case Example: ReportEmergency
Flow of Events

♦ The **FieldOfficer** activates the “Report Emergency” function of her terminal. FRIEND responds by presenting a form to the officer.

♦ The FieldOfficer fills the form, by selecting the emergency level, type, location, and brief description of the situation. The FieldOfficer also describes possible responses to the emergency situation. Once the form is completed, the FieldOfficer submits the form, at which point, the Dispatcher is notified.

♦ The Dispatcher reviews the submitted information and creates an Incident in the database by invoking the OpenIncident use case. The Dispatcher selects a response and acknowledges the emergency report.

♦ The FieldOfficer receives the acknowledgment and the selected response.
## Refined version of ReportEmergency

<table>
<thead>
<tr>
<th>Use case name</th>
<th>ReportEmergency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participating actors</td>
<td>Initiated by FieldOfficer</td>
</tr>
<tr>
<td></td>
<td>Communicates with Dispatcher</td>
</tr>
</tbody>
</table>

### Flow of events

1. The FieldOfficer activates the “Report Emergency” function of her terminal.

2. FRIEND responds by presenting a form to the officer. The form includes an emergency type menu (general emergency, fire, transportation) and location, incident description, resource request, and hazardous material fields.

3. The FieldOfficer completes the form by specifying minimally the emergency type and description fields. The FieldOfficer may also describe possible responses to the emergency situation and request specific resources. Once the form is completed, the FieldOfficer submits the form.

4. FRIEND receives the form and notifies the Dispatcher by a pop-up dialog.

5. The Dispatcher reviews the submitted information and creates an Incident in the database by invoking the OpenIncident use case. All the information contained in the FieldOfficer's form is automatically included in the Incident. The Dispatcher selects a response by allocating resources to the Incident (with the AllocateResources use case) and acknowledges the emergency report by sending a short message to the FieldOfficer.

6. FRIEND displays the acknowledgment and the selected response to the FieldOfficer.

### Entry condition

- ...
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

![Diagram](image-url)
<<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.

![Diagram showing extend association]

**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.
### Adding the Exception Condition

<table>
<thead>
<tr>
<th>ReportEmergency (include relationship)</th>
<th>ReportEmergency (extend relationship)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ...</td>
<td>1. ...</td>
</tr>
<tr>
<td>2. ...</td>
<td>2. ...</td>
</tr>
<tr>
<td>3. The FieldOfficer completes the form by selecting the emergency level, type, location, and brief description of the situation. The FieldOfficer also describes possible responses to the emergency situation. Once the form is completed, the FieldOfficer submits the form, at which point, the Dispatcher is notified. <em>If the connection with the Dispatcher is broken, the ConnectionDown use case is used.</em></td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. The FieldOfficer completes the form by selecting the emergency level, type, location, and brief description of the situation. The FieldOfficer also describes possible responses to the emergency situation. Once the form is completed, the FieldOfficer submits the form, at which point, the Dispatcher is notified.</td>
</tr>
<tr>
<td></td>
<td>4. The Dispatcher reviews the submitted information and creates an Incident in the database by invoking the OpenIncident use case. The Dispatcher selects a response and acknowledges the emergency report. <em>If the connection is broken, the ConnectionDown use case is used.</em></td>
</tr>
<tr>
<td></td>
<td>5. ...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ConnectionDown (include relationship)</th>
<th>ConnectionDown (extend relationship)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The FieldOfficer and the Dispatcher are notified that the connection is broken. They are advised of the possible reasons why such an event would occur (e.g., &quot;Is the FieldOfficer station in a tunnel?&quot;).</td>
<td></td>
</tr>
<tr>
<td>2. The situation is logged by the system and recovered when the connection is reestablished.</td>
<td></td>
</tr>
<tr>
<td>3. The FieldOfficer and the Dispatcher enter in contact through other means and the Dispatcher initiates ReportEmergency from the Dispatcher station.</td>
<td></td>
</tr>
<tr>
<td><em>The ConnectionDown use case extends any use case in which the communication between the FieldOfficer and the Dispatcher can be lost.</em></td>
<td></td>
</tr>
<tr>
<td>1. The FieldOfficer and the Dispatcher are notified that the connection is broken. They are advised of the possible reasons why such an event would occur (e.g., &quot;Is the FieldOfficer station in a tunnel?&quot;).</td>
<td></td>
</tr>
<tr>
<td>2. The situation is logged by the system and recovered when the connection is reestablished.</td>
<td></td>
</tr>
<tr>
<td>3. The FieldOfficer and the Dispatcher enter in contact through other means and the Dispatcher initiates ReportEmergency from the Dispatcher station.</td>
<td></td>
</tr>
</tbody>
</table>

*Bernd Bruegge & Allen H. Dutoit*
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 case terminates when…”
- Exceptions
  - Describe what happens if things go wrong
- Special Requirements
  - Nonfunctional Requirements, Constraints
Requirements Analysis Document

1. Introduction
   1.1 Purpose of the system
   1.2 Scope of the system
   1.3 Objectives and success criteria of the project
   1.4 Definitions, acronyms, and abbreviations
   1.5 References
   1.6 Overview

2. Current system

3. Proposed system
   3.1 Overview
   3.2 Functional requirements
   3.3 Nonfunctional requirements
      3.3.1 Usability
      3.3.2 Reliability
      3.3.3 Performance
      3.3.4 Supportability
      3.3.5 Implementation
      3.3.6 Interface
      3.3.7 Packaging
      3.3.8 Legal
   3.4 System models
      3.4.1 Scenarios
      3.4.2 Use case model
      3.4.3 Object model
      3.4.4 Dynamic model
      3.4.5 User interface—navigational paths and screen mock-ups

4. Glossary
ARENA: The Problem

♦ The Internet has enabled virtual communities
  ♦ Groups of people sharing common 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>organizeTicTacToeTournament</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>1. Joe, a friend of Alice, is a Tic Tac Toe aficionado and volunteers to organize a tournament.</td>
</tr>
<tr>
<td></td>
<td>2. Alice registers Joe in the arena as a league owner.</td>
</tr>
<tr>
<td></td>
<td>3. 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></td>
<td>4. Joe schedules the first tournament in the league for 16 players starting the next day.</td>
</tr>
<tr>
<td></td>
<td>5. 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></td>
<td>6. Bill and Mary receive the E-mail notification.</td>
</tr>
<tr>
<td></td>
<td>7. Mary is interested in playing the tournament and registers. 19 others apply.</td>
</tr>
<tr>
<td></td>
<td>8. Joe schedules 16 players for the tournament and rejects the 4 that applied last.</td>
</tr>
<tr>
<td></td>
<td>9. The 16 players, including Mary, receive an electronic token for entering the tournament and the time of their first match.</td>
</tr>
<tr>
<td></td>
<td>10. 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></td>
<td>11. 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></td>
<td>12. 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></td>
<td>13. 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></td>
<td>14. 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></td>
<td>15. Also, the winner of the tournament accumulates expert rating points.</td>
</tr>
<tr>
<td></td>
<td>16. 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.

alice: Operator

joe: LeagueOwner

mary: Player

bill: Spectator

zoe: Advertiser

defineKnockOutStyle

installTicTacToeGame

organizeTicTacToeTournament

analyzeTicTacToeTournament

sponsorTicTacToeBeginnersLeague
### Working glossary for ARENA

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Game</strong></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><strong>Match</strong></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><strong>Tournament</strong></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><strong>League</strong></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><strong>TournamentStyle</strong></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.
### Figure 4-21 (continued)

<table>
<thead>
<tr>
<th>Role</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register</td>
<td>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.</td>
</tr>
<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>
<tbody>
<tr>
<td><strong>Participating actors</strong></td>
<td>Initiated by LeagueOwner Communicates with Advertiser, Player, and Spectator</td>
</tr>
</tbody>
</table>
| **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
Group Work

♦ List all the things that might go wrong when someone tries to announce a tournament
Figure 4-25, Exceptions occurring in AnnounceTournament represented as extending use cases.
## Nonfunctional Requirements

<table>
<thead>
<tr>
<th>Category</th>
<th>Example questions</th>
</tr>
</thead>
</table>
| **Usability** | • What is the level of expertise of the user?  
• What user interface standards are familiar to the user?  
• What documentation should be provided to the user? |
| **Reliability** (including robustness, safety, and security) | • How reliable, available, and robust should the system be?  
• Is restarting the system acceptable in the event of a failure?  
• How much data can the system loose?  
• How should the system handle exceptions?  
• Are there safety requirements of the system?  
• Are there security requirements of the system? |
| **Performance** | • How responsive should the system be?  
• Are any user tasks time critical?  
• How many concurrent users should it support?  
• How large is a typical data store for comparable systems?  
• What is the worse latency that is acceptable to users? |
| **Supportability** (including maintainability and portability) | • What are the foreseen extensions to the system?  
• Who maintains the system?  
• Are there plans to port the system to different software or hardware environments? |
| **Implementation** | • Are there constraints on the hardware platform?  
• Are constraints imposed by the maintenance team?  
• Are constraints imposed by the testing team? |
| **Interface** | • Should the system interact with any existing systems?  
• How are data exported/imported into the system?  
• What standards in use by the client should be supported by the system? |
| **Operation** | • Who manages the running system? |
| **Packaging** | • Who installs the system?  
• How many installations are foreseen?  
• Are there time constraints on the installation? |
| **Legal** | • How should the system be licensed?  
• Are any liability issues associated with system failures?  
• Are any royalties or licensing fees incurred by using specific algorithms or components? |
Summary

♦ The requirements process consists of requirements elicitation and analysis.
♦ The requirements elicitation activity is different for:
  ♦ Greenfield Engineering, Reengineering, Interface Engineering
♦ Scenarios:
  ♦ Great way to establish communication with client
  ♦ Different types of scenarios: As-Is, visionary, evaluation and training
  ♦ Use cases: Abstraction of scenarios
♦ Pure functional decomposition is bad:
  ♦ Leads to unmaintainable code
♦ Pure object identification is bad:
  ♦ May lead to wrong objects, wrong attributes, wrong methods
♦ The key to successful analysis:
  ♦ Start with use cases and then find the participating objects
  ♦ If somebody asks “What is this?”, do not answer right away. Return the question or observe the end user: “What is it used for?”