



**Agent and Object Technology Lab**  
Dipartimento di Ingegneria dell'Informazione  
Università degli Studi di Parma



# Advanced Software Engineering

## Requirements Engineering

**Prof. Agostino Poggi**

- ♦ It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification
- ♦ This is inevitable as requirements may serve a dual function
  - May be the basis for a bid for a contract: therefore must be open to interpretation
  - May be the basis for the contract itself: therefore must be defined in detail
- ♦ Both these statements may be called requirements

- ◆ Requirements engineering is the process of establishing
  - The services that the customer requires from a system
  - The constraints under which it operates and is developed
- ◆ Goal of the requirements engineering is to gather and analyze the descriptions of the system services and constraints

- ◆ Many people have difficulty understanding the difference between scope, requirements and design
  - Scope demonstrates the needs of the organization, and is documented in a vision and scope document
  - Requirements document the behavior of the software that will satisfy those needs
  - Design shows how those requirements will be implemented technically

- ◆ In principle, requirements should state **what** the system should do and the design should describe **how** it does this
- ◆ In practice, requirements and design are inseparable
  - A system architecture may be designed to structure the requirements
  - The system may interoperate with other systems that generate design requirements
  - The use of a specific design may be a domain requirement

- ◆ Use case analysis
  - Mostly focused on writing text and the simple use of overview context diagrams is often enough
  - Use cases are just a part of functional requirements describing the interactions inside the system and between the system and its users and providers
- ◆ Structural analysis – domain modeling
  - Finding the “real-world” objects involved in the use cases and creating class diagrams to represent them

- ◆ Behavioral analysis
  - Creating activity diagrams and sequence diagrams to capture use case details
  - Activity diagrams for business workflow
  - Sequence diagrams for reactive behavior (also with timing)
  - Possibly creating state charts to capture external reactive behavior of the system and other domain objects

- ◆ System customers
  - Help in their specification and read them to check whether they satisfy their needs
  - Help in specifying of changes in the requirements
- ◆ Managers
  - Use them to plan a bid for the realization of the system
  - Use them to plan the system development process



- ◆ System engineers
  - Use them to understand what system must be developed
- ◆ System test engineers
  - Use them to develop the validation tests for the system
- ◆ System maintenance engineers
  - Use them to understand the system and the relationships between its parts

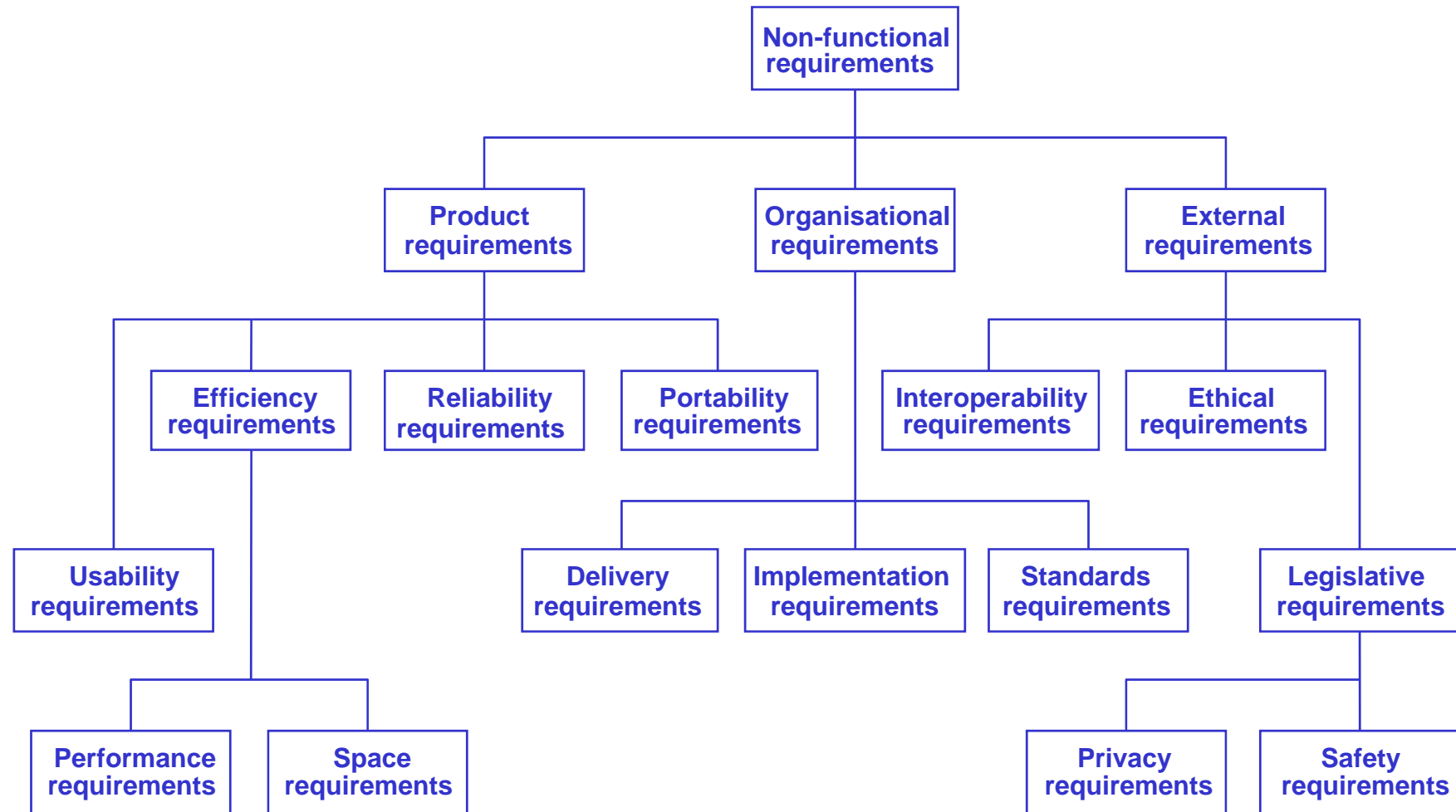
- ◆ On the basis of the type of the system feature
  - Functional requirements
  - Non-functional requirements
  - Domain requirements
- ◆ On the basis of their static / dynamic nature
  - Enduring requirements
  - Volatile requirements
- ◆ On the basis of their audience
  - User requirements
  - System requirements

- ◆ Capture the intended behavior of the system
  - This behavior may be expressed as services, tasks or functions the system is required to perform
  
- ◆ Describe how the system should react to particular inputs and how the system should behave in particular situations
  - Describe functionality or system services

- ◆ Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.
- ◆ Specify criteria that can be used to judge the operation of a system, rather than specific behaviors
- ◆ Constrains the design and the implementation of the system, but does not describe a service that the system should provide

- ◆ Product requirements
  - Requirements which specify that the delivered product must behave in a particular way
    - E.g. execution speed, reliability, etc.
- ◆ Organizational requirements
  - Requirements which are a consequence of organizational policies and procedures
    - E.g. process standards used, implementation requirements, etc.
- ◆ External requirements
  - Requirements which arise from factors which are external to the system and its development process
    - E.g. interoperability requirements, legislative requirements, etc.

# Non-Functional Requirements Classification



- ◆ Non-functional requirements may be very difficult to state precisely and imprecise requirements may be difficult to verify
- ◆ Goal
  - A general intention of the user such as “ease of use”
- ◆ Verifiable non-functional requirement
  - A statement using some measure that can be objectively tested
- ◆ Goals are helpful to developers as they convey the intentions of the system users

- ◆ A goal
  - The system should be easy to use by experienced controllers and should be organized in such a way that user errors are minimized
  
- ◆ A verifiable non-functional requirement
  - Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day



- ◆ Speed: processed transaction per second, user event response time, screen refresh time
- ◆ Size: Kbytes, number of RAM chips
- ◆ Ease to use: training time, number of help frames
- ◆ Reliability: mean time to failure, probability of unavailability
- ◆ Robustness: time to restart after failure, percentage of events causing failure, probability of data corruption on failure
- ◆ Portability: percentage of target dependent statement, number of target systems

- ◆ Requirements that come from the application domain of the system and that reflect characteristics of that domain
- ◆ Can be either functional or non-functional that constraints existing requirements or define specific computations
  - If domain requirements are not satisfied, the system may be unworkable

- ◆ Stable requirements derived from the core activity of the customer organization
- ◆ For example, a hospital will always have doctors, nurses, etc.
- ◆ They may be derived from domain models

- ◆ Requirements which change during development or when the system is in use
- ◆ For example, in a hospital, requirements derived from health-care policy
- ◆ They can be divided in mutable, emergent, consequential and compatibility requirements

- ◆ Mutable requirements

- Requirements that change because of changes to the environment in which the organization is operating
- For example, in hospital systems, the funding of patient care may change and thus require different treatment information to be collected

- ◆ Emergent requirements

- Requirements that emerge as the customer's understanding of the system develops during the system development
- The design process may reveal new emergent requirements

- ◆ Consequential requirements
  - Requirements that result from the introduction of the computer system
  - Introducing the computer system may change the organizations processes and open up new ways of working which generate new system requirements
- ◆ Compatibility requirements
  - Requirements that depend on the particular systems or business processes within an organization
  - As these change, the compatibility requirements on the commissioned or delivered system may also have to evolve

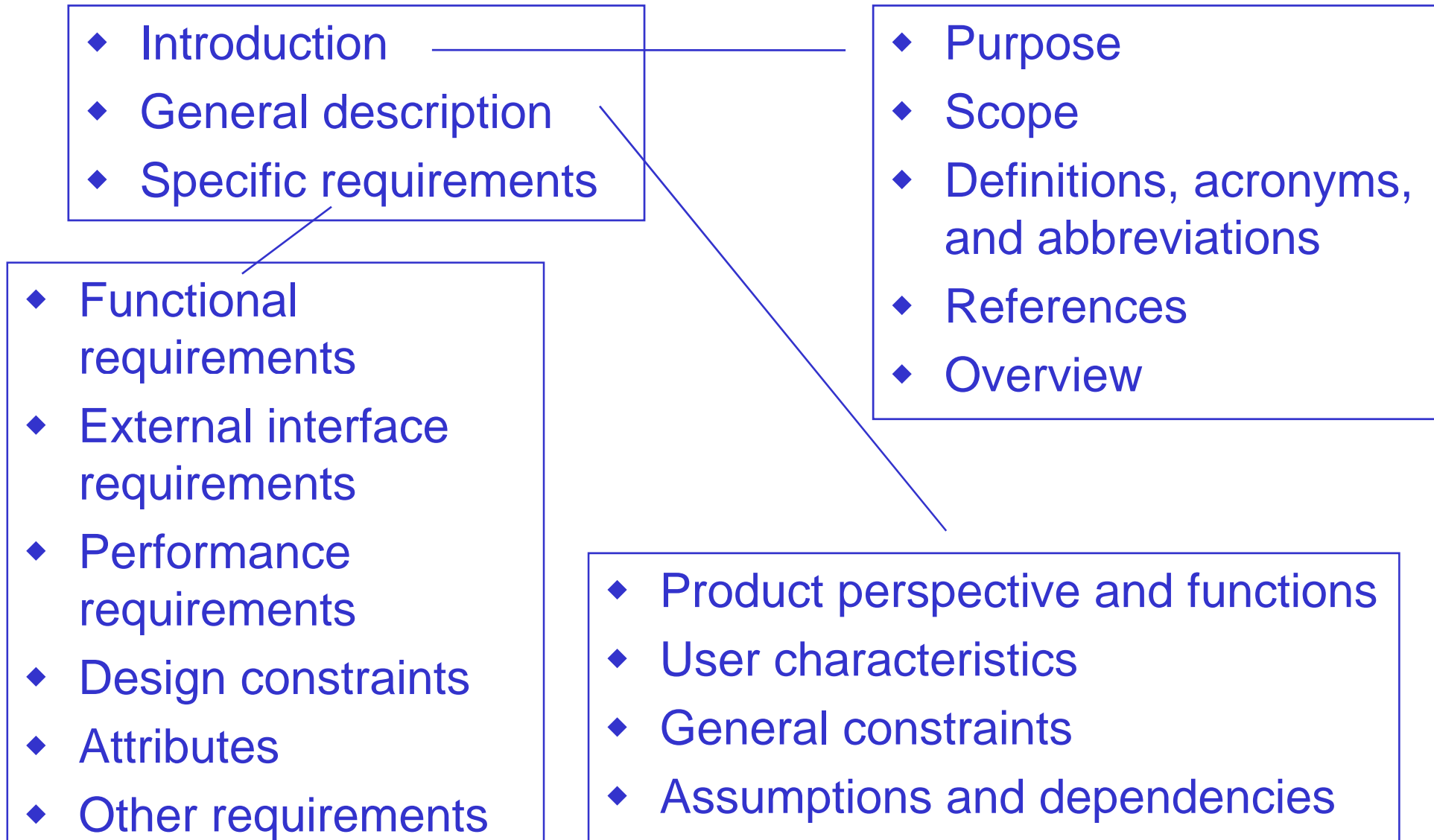
- ◆ Description of the system services and its operational constraints that is understandable by system users who don't have technical knowledge
- ◆ Written mainly for customers
  - Client managers
  - System end-users
  - Client engineering
  - Contractor manager
  - System architects
- ◆ User requirements are defined using natural language, tables and diagrams

- ◆ More detailed specifications of user requirements
- ◆ Written for both client and contractor
  - System end-users
  - Client engineering
  - System architects
  - Software developers
- ◆ Can act as a contract between client and contractor
- ◆ System requirements may be expressed using system models



- ◆ Is a document that clearly specifies the system requirements as identified during the requirements process
- ◆ Should include both a definition and a specification of requirements
- ◆ Should describe what the system should do rather than how it should do it (It is not a design document)
- ◆ This document is called:
  - System Specification if it deals with hardware and software
  - Software Requirements Specification (SRS) if it deals only with software

- ◆ There are standards such as IEEE 830 which dictate the requirements for an SRS conforming to that standard
- ◆ A standard template for software requirements specifications may be used within organizations to aid in the production of consistent and complete requirements documents
  - SRS standards or templates may help to provide structure and consistency
  - SRS standards or templates may also impede the effective specification of requirements



- ◆ Use a layered format that provides increasing detail as the layers deepen
- ◆ Use consistent graphical notation and apply textual terms consistently
- ◆ Be sure to define all acronyms
- ◆ Be sure to include a table of contents and, if possible, an index and/or glossary
- ◆ Write in a simple, unambiguous style
- ◆ Always put yourself in the reader's position

- ◆ Problems arise when requirements are not precisely stated
- ◆ Ambiguous requirements may be interpreted in different ways by developers and users
- ◆ Consider the term appropriate viewers:
  - User intention: special purpose viewer for each different document type
  - Developer interpretation: provide a text viewer that shows the contents of the document

- ◆ In principle requirements should be both complete and consistent
  - Complete: they should include descriptions of all facilities required
  - Consistent: there should be no conflicts or contradictions in the descriptions of the system facilities
- ◆ In practice, it is very difficult or impossible to produce a complete and consistent requirements document

## ◆ Understandability

- Some requirements may be expressed in the language of the application domain
  - This is often not understood by software engineers developing the system
- Some requirements may be expressed in the language of developers
  - This is often not understood by customers of the system

## ◆ Implicitness

- Domain specialists understand the area so well that they do not think of making the domain requirements explicit
- Development specialists may consider that some technical solutions implicitly follows the requirements

- ◆ Conflicts between different non-functional requirements are common in complex systems
- ◆ Consider some requirements of a spacecraft system:
  - To minimize weight, the number of separate chips in the system should be minimized
  - To minimize power consumption, lower power chips should be used
  - However, using low power chips may mean that more chips have to be used



- ◆ Lack of clarity
  - Precision is difficult without making the document difficult to read
- ◆ Requirements confusion
  - Functional and non-functional requirements tend to be mixed-up
- ◆ Requirements amalgamation
  - Several different requirements may be expressed together

- ◆ Ambiguity

- The readers and writers of the requirement must interpret the same words in the same way, but natural language is naturally ambiguous so this is very difficult

- ◆ Over-flexibility

- The same thing may be said in a number of different ways in the specification

- ◆ Lack of modularization

- Natural language structures are inadequate to structure system requirements

- ◆ Invent a standard format and use it for all requirements
- ◆ Use language in a consistent way:
  - Same syntax for the description of the requirements
  - Same terms for the same type of requirements
- ◆ Use text highlighting to identify key parts of the requirement
- ◆ Avoid the use of computer jargon

### ◆ Behaviors/Constraints:

- **Shall:** system has to do it 100% of the time unless specifically excepted
- **Should:** desirable that system does it whenever reasonable to do so
- **Can:** system can do something, but no particular incentive to implement

### ◆ User Actions:

- **Must:** user has to do this (same as “shall”, except for user, not computer)
- **May:** user can exhibit this behavior, but does not have to

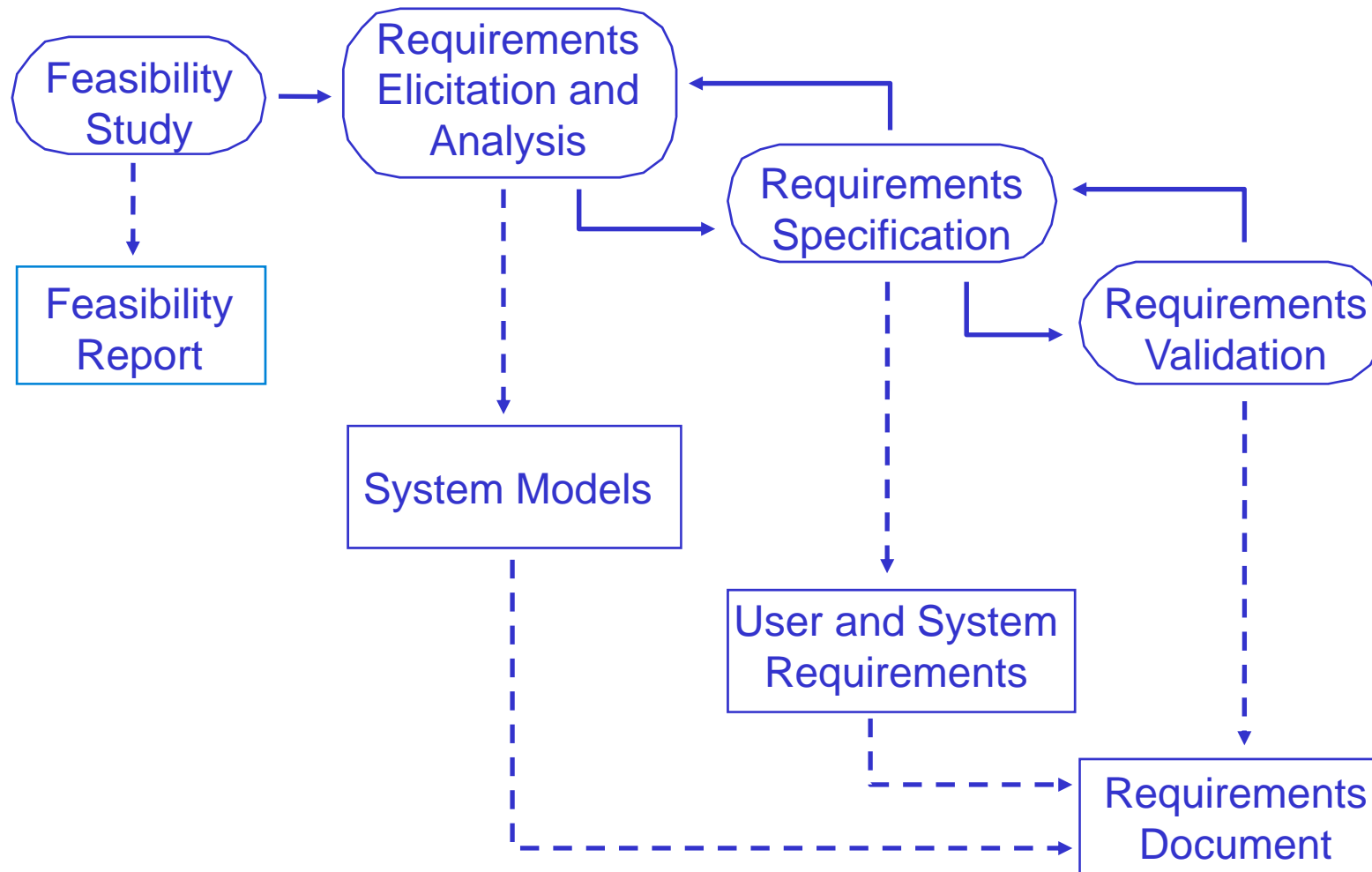
- ◆ Environment:

- **Will:** designer can count on environment being this way
- **Might:** designer has to accommodate situation, but can't count on it

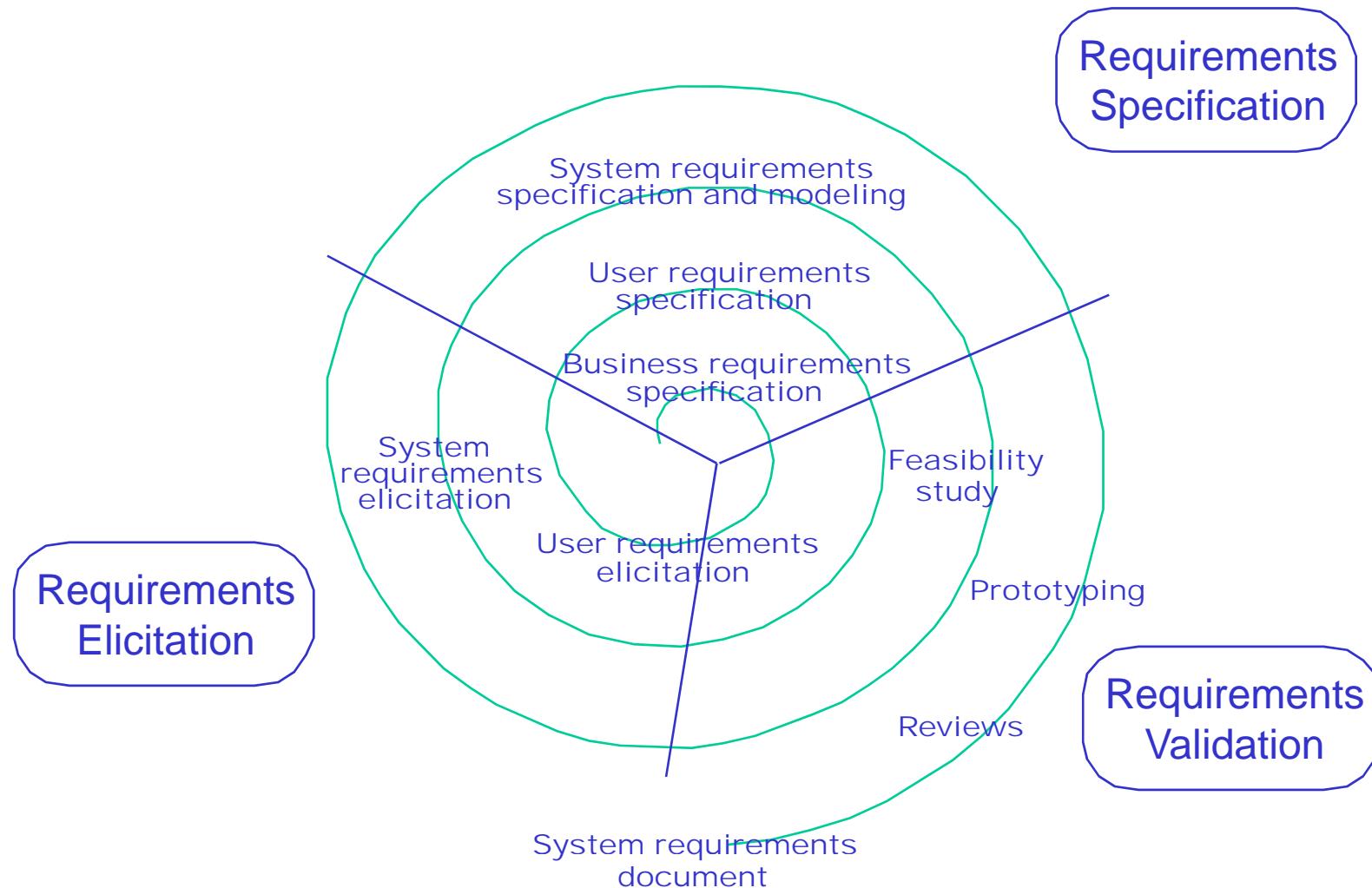
- ◆ Change risk:

- **Expected to:** this area is likely to changed
- **Could:** this area is something that could change, but might not

- ♦ The processes used for requirements engineering vary widely depending on:
  - The application domain
  - The people involved
  - The organization developing the requirements
- ♦ However, there are a number of generic activities common to all processes:
  - Requirements elicitation
  - Requirements analysis
  - Requirements validation
  - Requirements management



# Requirements Engineering Process





- ◆ User involvement
- ◆ Clear statement of requirements
- ◆ Proper planning
- ◆ Realistic expectations
- ◆ Smaller project milestones
- ◆ Clear vision and objectives
- ◆ Hard working and focused staff

- ◆ A feasibility study decides whether or not the proposed system is worthwhile
- ◆ A short focused study that checks
  - If the system contributes to organisational objectives
  - If the system can be engineered using current technology and within budget
  - If the system can be integrated with other systems that are used

- ◆ Based on information assessment (what is required), information collection and report writing
- ◆ Questions for people in the organisation
  - What if the system wasn't implemented?
  - What are current process problems?
  - How will the proposed system help?
  - What will be the integration problems?
  - Is new technology needed? What skills?
  - What facilities must be supported by the proposed system?

- ◆ Identify relevant sources of requirements (usually customer)
- ◆ Determine what information is needed
- ◆ Analyze the gathered information, looking for implications, inconsistencies, or unresolved issues
- ◆ Confirm your understanding of the requirements with the source
- ◆ Synthesize appropriate statements of the requirements

- ◆ The customer fully explore and fully understand their requirements
- ◆ The customers are able to separate their wants from their needs
- ◆ The customers are able to understand the capabilities and limitations of computer technology
- ◆ The customers understand the alternative solutions and impact of each alternative
- ◆ The customers understand the impact of the requirements on the developer and themselves

- ♦ The developers are solving the right problem
- ♦ The developers have confidence that the system to be delivered is feasible to build
- ♦ The developers have the trust and confidence of the customer
- ♦ The developers gain domain knowledge of the system

- ♦ The customer probably will be dissatisfied
- ♦ The customer and developer have to cope with constantly changing requirements
- ♦ The developer is solving the wrong problem
- ♦ The developer has a difficult time building the system

- ◆ Sampling
  - Process of collecting a representative sample of documents, forms, and records
- ◆ Observation of the work environment
  - A fact-finding technique where the systems analyst either participates in or watches a person in to learn about the system
- ◆ Questionnaire
  - A special-purpose document that allows the analyst to collect information and opinions from respondents
- ◆ Interviews
  - A fact-finding technique whereby the systems analysts collect information from individuals through face-to-face interaction



- ♦ Organization chart
- ♦ Memos and other documents that describe the problem
- ♦ Documentation and standard operating procedures for current system
- ♦ Manual and computerized screens and reports
- ♦ Samples of databases
- ♦ Etc.

- ◆ Randomization

- A sampling technique characterized by having no predetermined pattern or plan for selecting sample data

- ◆ Stratification

- A systematic sampling technique that attempts to reduce the variance of the estimates by spreading out the sampling
- For example, choosing documents or records by formula and by avoiding very high or low estimates

- ◆ Determine the who, what, where, when, why, and how of the observation
- ◆ Obtain permission from appropriate supervisors or managers
- ◆ Inform those who will be observed of the purpose of the observation
- ◆ Keep a low profile

- ♦ Take notes during or immediately following the observation
- ♦ Review observation notes with appropriate individuals
- ♦ Don't interrupt the individuals at work
- ♦ Don't focus heavily on trivial activities
- ♦ Don't make assumptions

- ♦ Data gathered can be very reliable
- ♦ Can see exactly what is being done in complex tasks
- ♦ Relatively inexpensive compared with other techniques
- ♦ Can do work measurements

- ◆ People may perform differently when being observed
- ◆ Work observed may not be representative of normal conditions
- ◆ Timing can be inconvenient
- ◆ Interruptions
- ◆ Some tasks not always performed in the same way
- ◆ May observe wrong way of doing things

- ◆ Free-format questionnaire
  - A questionnaire designed to offer the respondent greater latitude in the answer
  - A question is asked, and the respondent records the answer in the space provided after the question
- ◆ Fixed-format questionnaire
  - A questionnaire containing questions that require selecting an answer from predefined available responses

- ◆ Determine what facts and opinions must be collected and from whom you should get them
- ◆ Don't get carried away
- ◆ Based on the facts and opinions sought, determine whether free- or fixed-format questions will produce the best answers
- ◆ Write the questions
- ◆ Test the questions on a small sample of respondents
- ◆ Duplicate and distribute the questionnaire



- ◆ Unstructured interview
  - An interview that is conducted with only a general goal or subject in mind and with few, if any, specific questions
  - The interviewer counts on the interviewee to provide a framework and direct the conversation
- ◆ Structured interview
  - An interview in which the interviewer has a specific set of questions to ask of the interviewee

- ◆ Open-ended question
  - A question that allows the interviewee to respond in any way that seems appropriate
- ◆ Closed-ended question
  - A question that restricts answers to either specific choices or short, direct responses
- ◆ Often the open-ended and closed-ended are mixed

- ◆ Select interviewees
  - Learn about individual prior to the interview
- ◆ Prepare for the interview
  - Interview guide is a checklist of questions
- ◆ Conduct the interview
  - Establish rapport
  - Summarize the problem
  - Offer an incentive for participation / ask the interviewee for assistance
- ◆ Follow up on the interview
  - Memo that summarizes the interview
  - Thank you!

- ◆ Types of questions to avoid
  - Loaded questions
  - Leading questions
  - Biased questions
- ◆ Interview question guidelines
  - Use clear and concise language
  - Don't include your opinion as part of the question
  - Avoid long or complex questions
  - Avoid threatening questions
  - Don't use "you" when you mean a group of people

### **Do's**

- ◆ Be courteous
- ◆ Listen carefully
- ◆ Maintain control
- ◆ Probe
- ◆ Observe mannerisms and nonverbal communication
- ◆ Be patient
- ◆ Keep interviewee at ease
- ◆ Maintain self-control

### **Don'ts**

- ◆ Continue an interview unnecessarily
- ◆ Assume an answer is finished or leading nowhere
- ◆ Reveal verbal and non verbal clues
- ◆ Using jargon
- ◆ Reveal your personal biases
- ◆ Talk instead of listen
- ◆ Assume anything about the topic and the interviewee

- ◆ Find, verify, clarify facts
  - About what stakeholders do
  - About how stakeholders might interact with the system
- ◆ Generate enthusiasm
- ◆ Get the end-user involved
- ◆ Solicit ideas and opinions

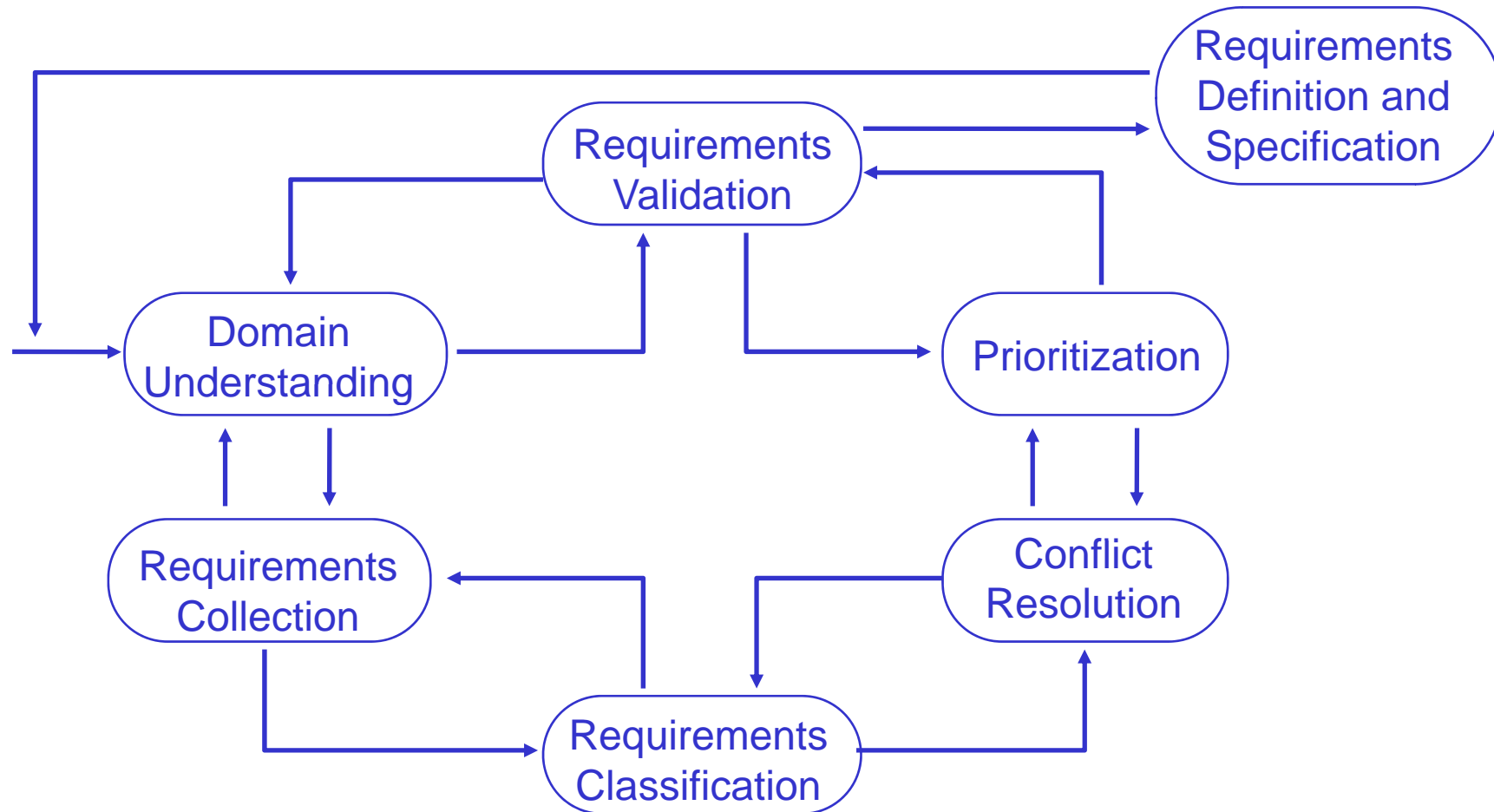
- ♦ Interviews are not good for understanding domain requirements
  - Requirements engineers cannot understand specific domain terminology
  - Some domain knowledge is so familiar that people find it hard to articulate or think that it isn't worth articulating

- ◆ Refines and structures the requirements to facilitate understanding (by developers), reuse, maintenance, etc.
- ◆ Yields a more precise and formal specification of requirements and defines the analysis model
  - Refine the informal description of the requirements
  - Convert the informal descriptions to flow diagrams



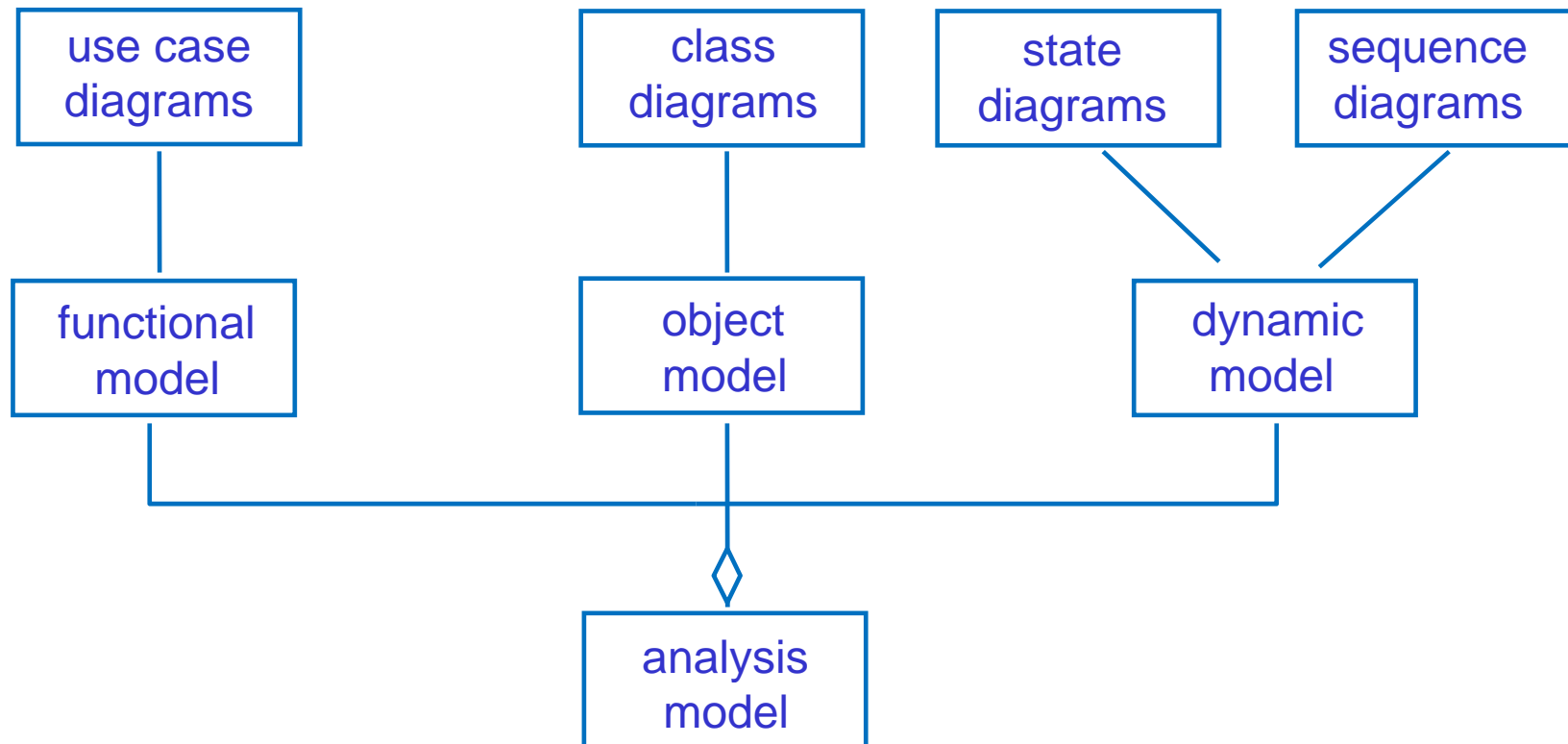
- ◆ Stakeholders don't know what they really want
- ◆ Stakeholders express requirements in their own terms
- ◆ Different stakeholders may have conflicting requirements
- ◆ Organizational and political factors may influence the system requirements
- ◆ The requirements may change during the analysis
  - New stakeholders may emerge and the business environment change

## Requirements Analysis Process



- ♦ The analysis model makes abstractions
  - It is not a design model
  - It is a platform (language, OS, ...) independent model
- ♦ A structured analysis model is based on the functions to be realized by the system
- ♦ An object-oriented analysis model is based on the use cases and on the representation of the real objects and concepts of the system domain

# Object-Oriented Analysis Model



- ◆ An analysis model is represented by an analysis system that is the top-level package of the model
- ◆ An analysis system may contain subsystems or analysis packages
- ◆ Within the analysis model, use cases are realized by analysis classes and interaction diagrams
- ◆ Analysis classes represent an abstraction of classes (and possibly entire subsystems) of the system to be implemented
- ◆ interaction diagrams describe dynamics of the system

- ◆ Use cases, stated simply, allow description of sequences of events that, taken together, lead to a system doing something useful
- ◆ A use case is simply a description (a story) of users and systems interacting in a typical fashion to go through the various paths or parts of a business process or automated process
- ◆ Use cases describe the interaction between a primary actor, the initiator of the interaction, and the system itself, through a sequence of simple steps

- ◆ Use cases should describe all possible interactions within and with the system
- ◆ Use cases are not mapped one-to-one to requirements
  - Each requirement must be covered by at least one use case
  - However, use cases may contain many requirements
- ◆ Use cases are described by combining:
  - Use case diagrams
  - Textual descriptions
  - Interaction diagrams (optional)
- ◆ Use cases are realized by identifying the actors and the interaction between the actors and the system

- ◆ Define system boundary to identify actors correctly
- ◆ Identify users and systems that depend on the system's primary and secondary functionalities
- ◆ Identify hardware and software platforms with which the system interacts
- ◆ Select entities that play distinctly different roles in the system
- ◆ Identify as actors external entities with common goals and direct interaction with the system
- ◆ Denote actors as nouns



- ◆ Business / Domain Use Cases
  - Interactions between users and the business (or domain)
- ◆ System Use Cases
  - Interactions between users and the system
  - One business use cases contains a set of system use cases
- ◆ To name the use cases, give it a verb name to show the action that must be performed
  - Describe a transaction completely
  - No description of user interface whatsoever

- ◆ Tasks to be performed by the user and the system
- ◆ Starting situation and states where it finishes
- ◆ Flow of information to the user and to the system
- ◆ Events that are conveyed to the user and to the system
- ◆ Normal flow of events
- ◆ What can go wrong
- ◆ Other concurrent activities

- ♦ Is described by few steps (up to 5), but most important provides a meaningful result to the end users
- ♦ Describes interactions and mechanisms not policies
- ♦ Excludes user or system interfaces implementation choices
- ♦ Is described by few (5 - 10) pages
- ♦ Has a single initiator actor
- ♦ Includes the major business exceptions and their handling

- ◆ Use cases can be ordered on the basis of their importance for the realization of the system
- ◆ Ordering depends on:
  - Significant impact on the architectural design
  - Include risk, time-critical or complex function
  - Involve significant research and/or new and risky technology
  - Represent primary line-of-business processes
  - Etc.
- ◆ Ranking is usually expressed with qualitative values:
  - E.g., high, medium, low
  - E.g., must have, essential, nice to have

- ◆ Define the start state and the preconditions
- ◆ Define when the use case starts
- ◆ Define the order of activity in the main flow of events
- ◆ Define any alternative flows of events
- ◆ Define any exceptional flows of events
- ◆ Define any post conditions and the end state
- ◆ Mention the actors involved with this use case, and any use cases used or extended by this use case
- ◆ Mention the related interaction diagrams
- ◆ Mention any design issues as an appendix

|                       |  |                    |  |
|-----------------------|--|--------------------|--|
| Use Case ID:          |  |                    |  |
| Use Case Name:        |  |                    |  |
| Created By:           |  | Last Updated By:   |  |
| Date Created:         |  | Date Last Updated: |  |
|                       |  |                    |  |
| Actors:               |  |                    |  |
| Description:          |  |                    |  |
| Trigger:              |  |                    |  |
| Preconditions:        |  |                    |  |
| Postconditions:       |  |                    |  |
| Normal Flow:          |  |                    |  |
| Alternative Flows:    |  |                    |  |
| Exceptions:           |  |                    |  |
| Includes:             |  |                    |  |
| Priority:             |  |                    |  |
| Frequency of Use:     |  |                    |  |
| Business Rules:       |  |                    |  |
| Special Requirements: |  |                    |  |
| Assumptions:          |  |                    |  |
| Notes and Issues:     |  |                    |  |

|                                  |  |
|----------------------------------|--|
| <b>Use Case</b>                  | <i>Use case identifier and reference number and modification history</i> |
| <b>Description</b>               | <i>Goal to be achieved by use case and sources for requirement</i>       |
| <b>Actors</b>                    | <i>List of actors involved in use case</i>                               |
| <b>Assumptions</b>               | <i>Conditions that must be true to terminate successfully</i>            |
| <b>Steps</b>                     | <i>Interactions between actor and system necessary to achieve goal</i>   |
| <b>Variations (optional)</b>     | <i>Any variations in the steps</i>                                       |
| <b>Non-Functional (optional)</b> | <i>List of non-functional requirements that the use case must meet.</i>  |
| <b>Issues</b>                    | <i>List of issues that remain to be resolved</i>                         |

|                       |   |
|-----------------------|---|
| <b>Use Case</b>       | 2. Repairing_Cellular_Network<br>history created 1/5/98 Derek Coleman, modified 5/5/98.   |
| <b>Description</b>    | Operator rectifies a report by changing parameters of a cell.<br>sources [Operating Manual 1993], [Jones 1998].   |
| <b>Assumptions</b>    | Changes to network are always successful when applied to a network.   |
| <b>Actors</b>         | Operator (primary)<br>Cellular network<br>Field maintenance engineer  |
| <b>Steps</b>          | <ol style="list-style-type: none"> <li>1. Operator notified of network problem.</li> <li>2. Operator starts repair session.</li> <li>3. REPEAT <ol style="list-style-type: none"> <li>3.1. Operator runs network diagnosis application.</li> <li>3.2 Operator identifies cells to be changed and their new parameter values.</li> <li>3.3 IN PARALLEL <ol style="list-style-type: none"> <li>3.3.1 Maintenance engineer tests network cells   </li> <li>3.3.2 Maintenance engineer sends fault reports.</li> </ol> </li> </ol> </li> </ol> <p>UNTIL no more reports of problems</p> <li>4. Operator closes repair session.</li> |
| <b>Variations</b>     | #1. System may detect fault and notify operator or<br>Field maintenance engineer may report fault to Operator.  |
| <b>Non-Functional</b> | <b>Performance Mean:</b> time to repair network fault must be less than 3 hours.<br><b>Fault Tolerance:</b> A repair session must be able to tolerate failure of Operator's console.  |
| <b>Issues</b>         | What are the modes of communication between field maintenance engineer and operator?  |

- ◆ Try to describe use cases without thinking about in what way they will be implemented
- ◆ Be as narrative as possible
- ◆ State success scenarios
- ◆ Introduce all the possible scenarios of a use cases
- ◆ Agree on a “format style” for use case description



- ◆ Name a use case starting with a verb in order to emphasize that it is a process
  - E.g., Buy Items, Enter an order, Reduce inventory, etc.
- ◆ Document exception handling or branching
  - E.g., when a “buy item” fails, then ...
  - E.g., when a “credit card approval ” fails, then ...
- ◆ Do not represent individual steps as use cases
  - E.g., define a use case for the operation: printing a receipt

- ♦ Tutors in the organization are assigned courses to teach according to the area that they specialize in and their availability
- ♦ The organization publishes and maintains a calendar of the different courses and the assigns tutors every year
- ♦ There is a group of course administrators in the organization who manage the courses including course content, assigning courses to tutors, and defining the course schedule

- ◆ Courses shall be assigned to the tutors only by course administrators
- ◆ Courses information topics and contents shall be managed only by course administrators
- ◆ Tutors information shall be managed only by course administrators and by corresponding tutor
- ◆ Course calendar shall be visualized by course administrators, tutors, and students

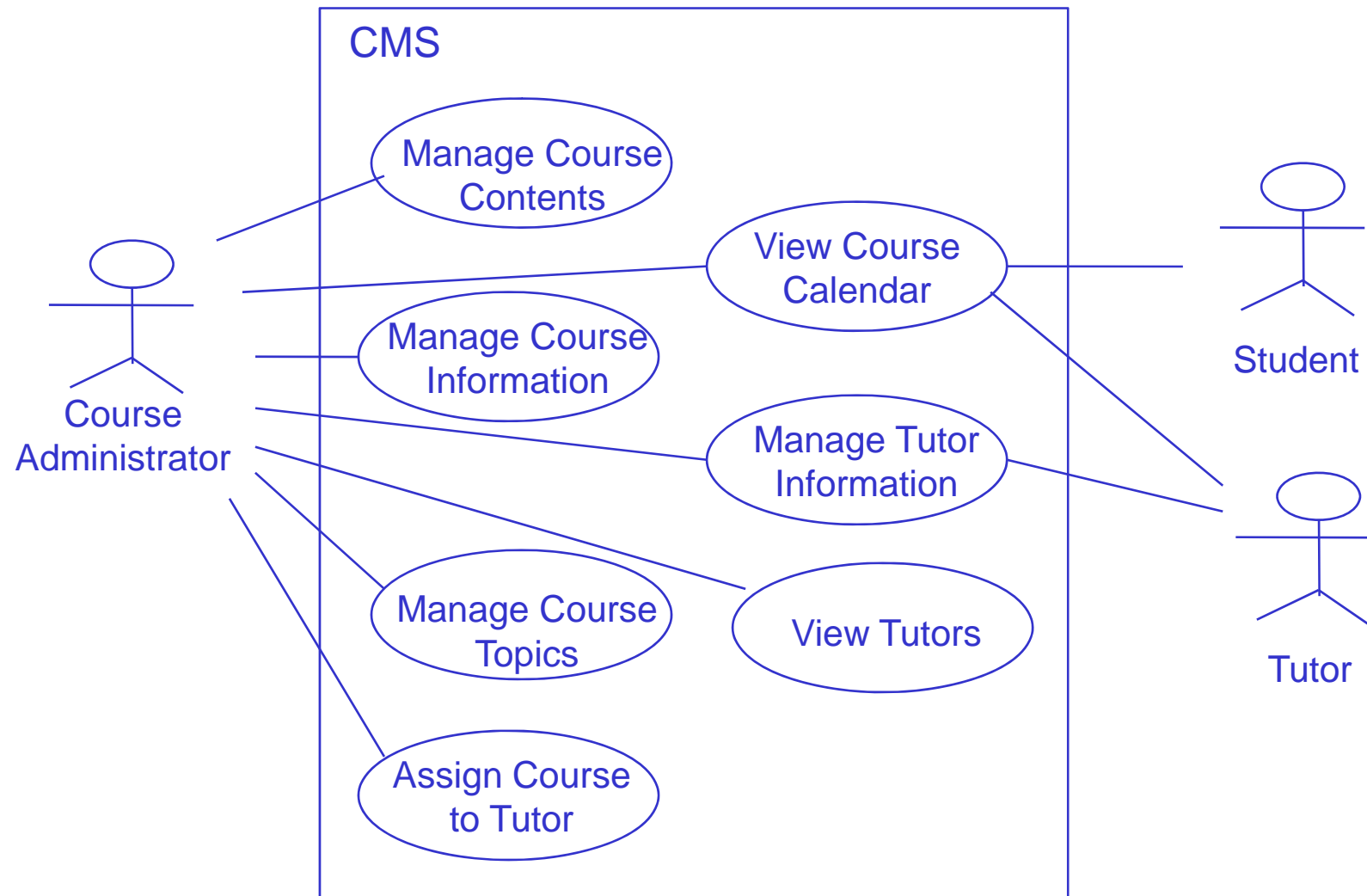
- ♦ **Courses** and **topics** that make up courses
- ♦ **Tutors** that teach courses
- ♦ **Course administrators** who manage the assignment of courses to tutors
- ♦ **Calendars** (course schedules) that are generated as a result of the work performed by the course administrators
- ♦ **Students** who refer to calendars (course schedules) to decide which courses they wish to take up for study

- ◆ Actors

- Tutor, student, course administrator

- ◆ Use Cases

- Manage courses: view courses, manage course topics and manage course information
- Manage tutors: view course calendar, view tutors, manage tutor information, and assign courses to tutors



- ◆ Playlist Support

- Download prebuilt playlist, create playlist, add songs to playlist, reorder playlist, delete song from playlist, calculate cost, check song availability

- ◆ Payment Support

- Display payment options, payment by cash, payment by cell phone, payment by credit card

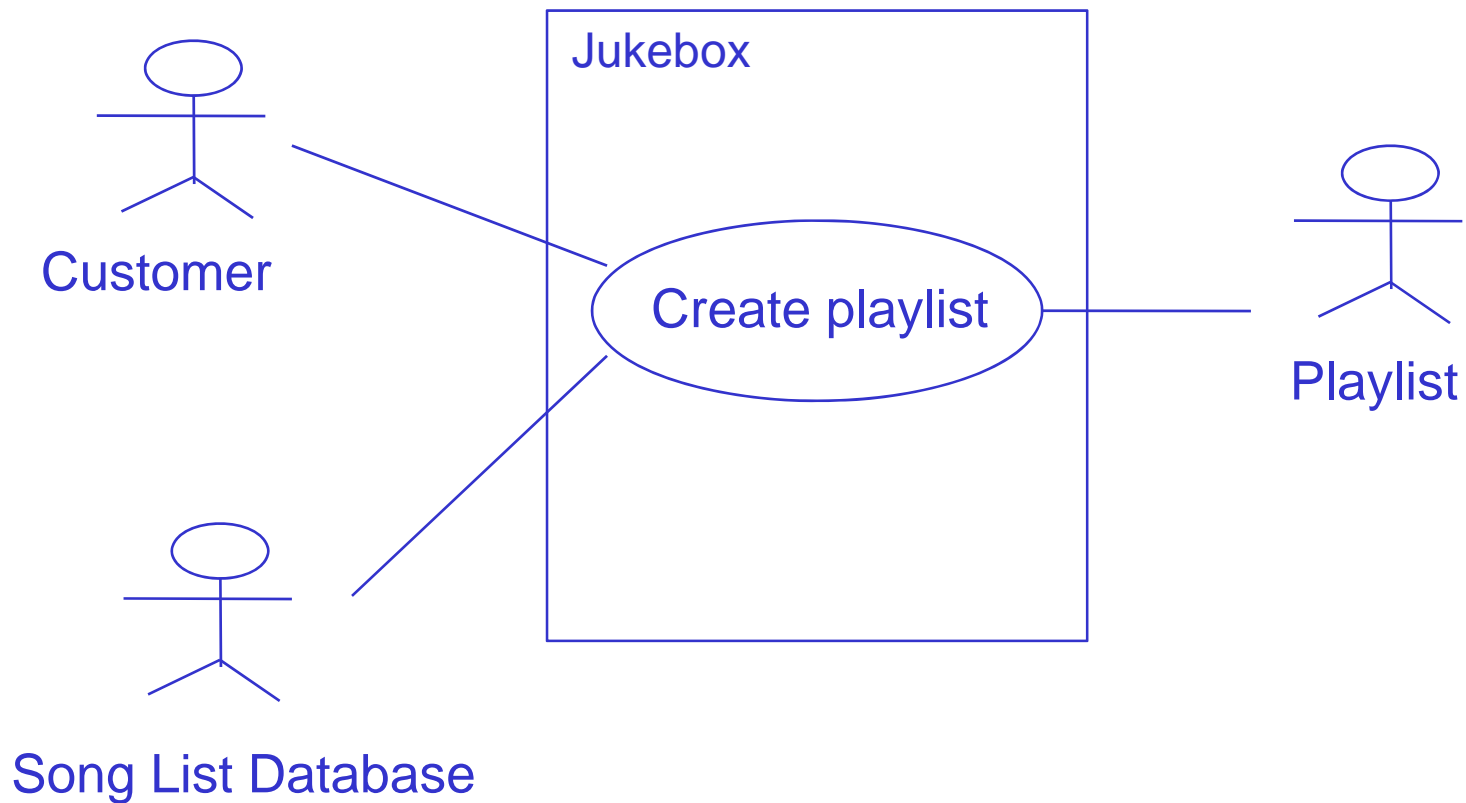
- ◆ Playback Mechanism

- Load song, play song, display song information

- ◆ Payment by cash
  - Monitor cash input
  - Return proper change
  - Refund money
- ◆ Payment by cell phone
  - Display phone number to dial
- ◆ Load Song
  - Fetch song from local database
  - Fetch song from network



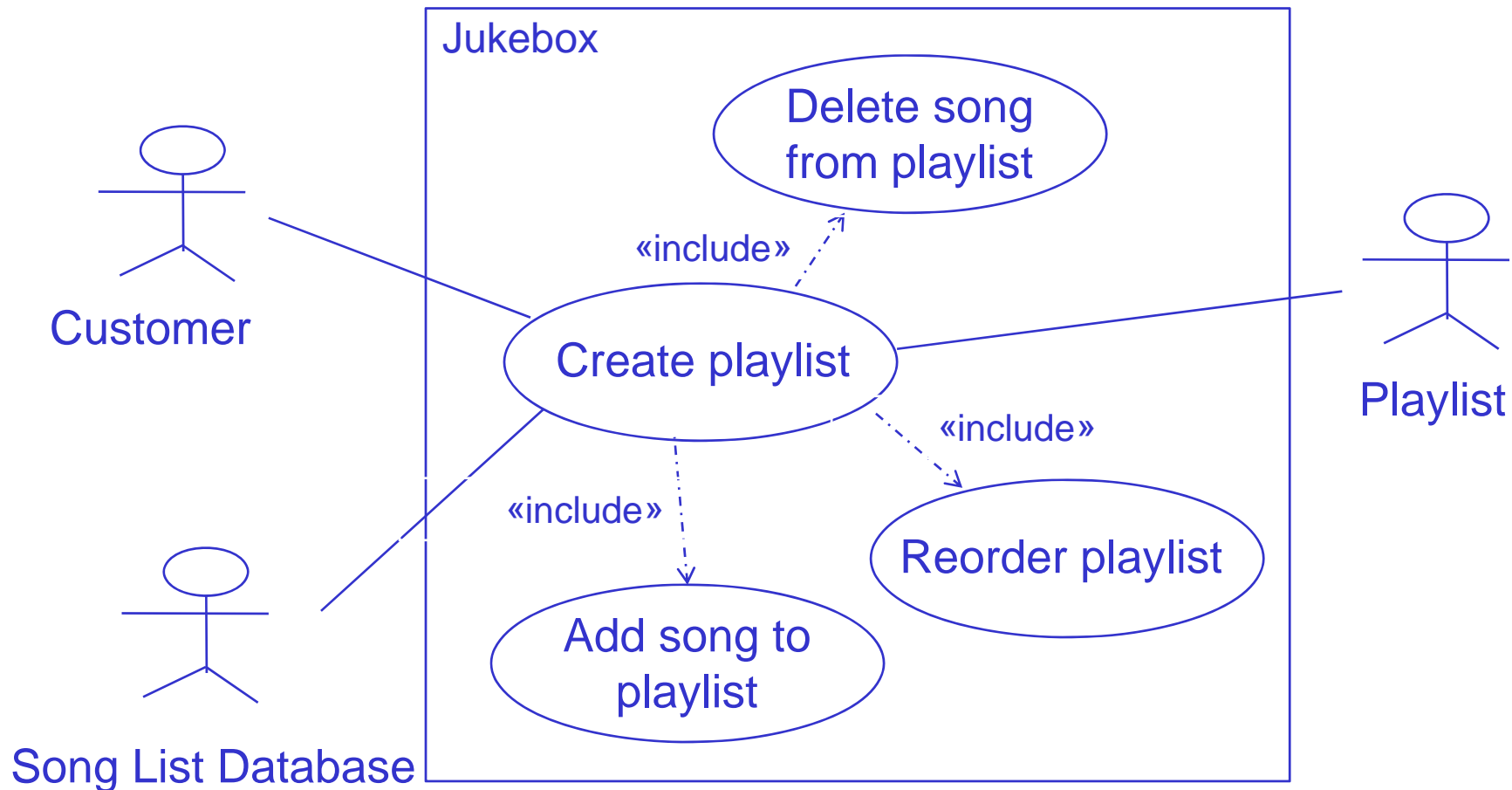
# Create Playlist Use Case Diagram



## Create Playlist Use Case Description

| Property         | Description                                      | Use Case                  |
|------------------|--|---------------------------|
| Title            | Create Playlist                                  |                           |
| Goal             | Allow the customer to create a playlist          |                           |
| Scope            | Playlist Support                                 |                           |
| Level            | Primary Task                                     |                           |
| Precondition(s)  | Access to a song list database must be available |                           |
| Postcondition(s) | Playlist has been created                        |                           |
| Primary Actor 1  | Song List Database                               |                           |
| Primary Actor 2  | Playlist   |                           |
| Primary Actor 3  | Customer   |                           |
| Secondary Actor  | None   |                           |
| Trigger Event    | Customer Chooses to Create a Playlist            |                           |
| Step 1           | Customer browses through song list database      |                           |
| Exception 1.1    | Timeout (Customer waited to long)                |                           |
| Step 2           | Customer selects songs to add to play list       | Add Songs to Playlist     |
| Step 3           | List of songs in playlist is displayed           |                           |
| Variation 3.1    | Customer chooses to reorder playlist             | Reorder Playlist          |
| Variation 3.2    | Customer chooses to delete songs from playlist   | Delete Song from Playlist |

## Create Playlist Use Case Diagram



- ◆ Analysis classes are an abstraction of one or more real classes or subsystems of the to system to be realized
- ◆ Analysis classes handle functional requirements
  - Non-functional requirements are handled in architectural design
- ◆ Analysis classes are usually represented with class diagrams through the Boundary, Control and Entity stereotypes
  - The semantics of these stereotypes provides a consistent method of finding and analyzing such classes

- ◆ Provides a crisp abstraction of something from the problem domain (or solution) domain
- ◆ Embodies a small well defined set of responsibilities and carry them out well
- ◆ Provides clear separation of abstraction, specification, and implementation
- ◆ Is understandable and simple yet extendable and adaptable

- ♦ Noun verb analysis
- ♦ Use case driven
- ♦ Common class patterns
- ♦ CRC cards
- ♦ Mixed approach

- ◆ Set of heuristics for identifying classes, attributes and associations from a requirements specification
  - Nouns may identify: classes, attributes and instances
  - Verb phrases may identify: operations, relationships and constraints
- ◆ Relies on the completeness and correctness of the requirements document
- ◆ Quality depends on style of writing, often there are too many nouns

| <b>Text Component</b> | <b>Model Component</b> | <b>Example</b> |
|-----------------------|------------------------|----------------|
| proper noun           | instance               | Jim Smith      |
| common noun           | class                  | toy, doll      |
| doing verb            | method                 | buy, recommend |
| classifying verb      | inheritance            | is a           |
| possessive verb       | aggregation            | has a          |
| modal verb            | constraint             | must be        |
| adjective             | attribute              | 3 years old    |
| transitive verb       | method                 | enter          |
| intransitive verb     | method (event)         | depends on     |



- ◆ Similar to the noun verb analysis, but centered on use cases
- ◆ Identifies objects, responsibilities of each object, and how these objects collaborate with other objects analyzing the different scenarios described in the use cases
- ◆ Relies on the completeness of use case models

- ♦ Derives classes from the generic classification theory of objects
  - Part of science that concerned with partitioning the world of objects into useful groups
- ♦ Provide useful guidance, but not offering a systematic process to find complete and reliable classes
- ♦ Too loosely bound to specific user requirements and possible misinterpretation of class name

- ◆ Concept class
  - E.g., reservation
- ◆ Event class
  - E.g., arrival
- ◆ Organization class
  - E.g., travelAgency
- ◆ People class
  - E.g., passenger
- ◆ Places class
  - E.g., TravelOffice

Bahrami(1999)

- ◆ Physical class
  - E.g., Airplane
- ◆ Business class
  - E.g., Reservation
- ◆ Logical class
  - E.g., FlightTimetable
- ◆ Application class
  - E.g., ReservationTrans
- ◆ Computer class
  - E.g., Index
- ◆ Behavioral class
  - E.g., ReservationCancel

Rumbaugh et. Al. (1999)

- ♦ An attractive way of interpreting, understanding and teaching about objects based on:
  - Three compartments cards:
    - Class name
    - Responsibilities of the class
    - Collaborators of the class
  - Animated brainstorming sessions
- ♦ CRC cards do not provide a systematic process to find complete and reliable classes
- ♦ CRC cards are a means for the validation of requirements identifying errors and omissions

- ◆ Initial classes come from domain knowledge
- ◆ Common class patterns approach to guide
- ◆ Noun phrase approach to add more classes
- ◆ Use case approach to verify
- ◆ CRC to brainstorm

- ♦ The library contains books and journals. It may have several copies of a given book. Some of the books are reserved for short-term loans only. All others may be borrowed by any library member for three weeks. Members of the library can normally borrow up to six items at a time, but members of staff may borrow up to 12 items at one time. Only members of staff may borrow journals
- ♦ The system must keep track of when books and journals are borrowed and returned and enforce the rules

- ♦ The **library** contains **books** and **journals**. It may have several **copies** of a given book. Some of the books are reserved for **short-term loans only**. All others may be borrowed by any library **member** for three **weeks**. Members of the library can normally borrow up to six **items** at a time, but **members of staff** may borrow up to 12 items at one time. Only members of staff may borrow journals.
- ♦ The **system** must keep track of when books and journals are borrowed and returned and enforce the **rules**.

- ♦ Library
- ♦ Book
- ♦ Journal
- ♦ Copy
- ♦ ShortTermLoan
- ♦ LibraryMember
- ♦ Week
- ♦ Item
- ♦ StaffMember
- ♦ System
- ♦ Rule

## From Library System description to Classes, Relationships and Operations

---

### Classes

- ◆ Library
- ◆ Book
- ◆ Journal
- ◆ Copy
- ◆ ShortTermLoan
- ◆ LibraryMember
- ◆ Week
- ◆ Item
- ◆ StaffMember
- ◆ System
- ◆ Rule

### Relationships

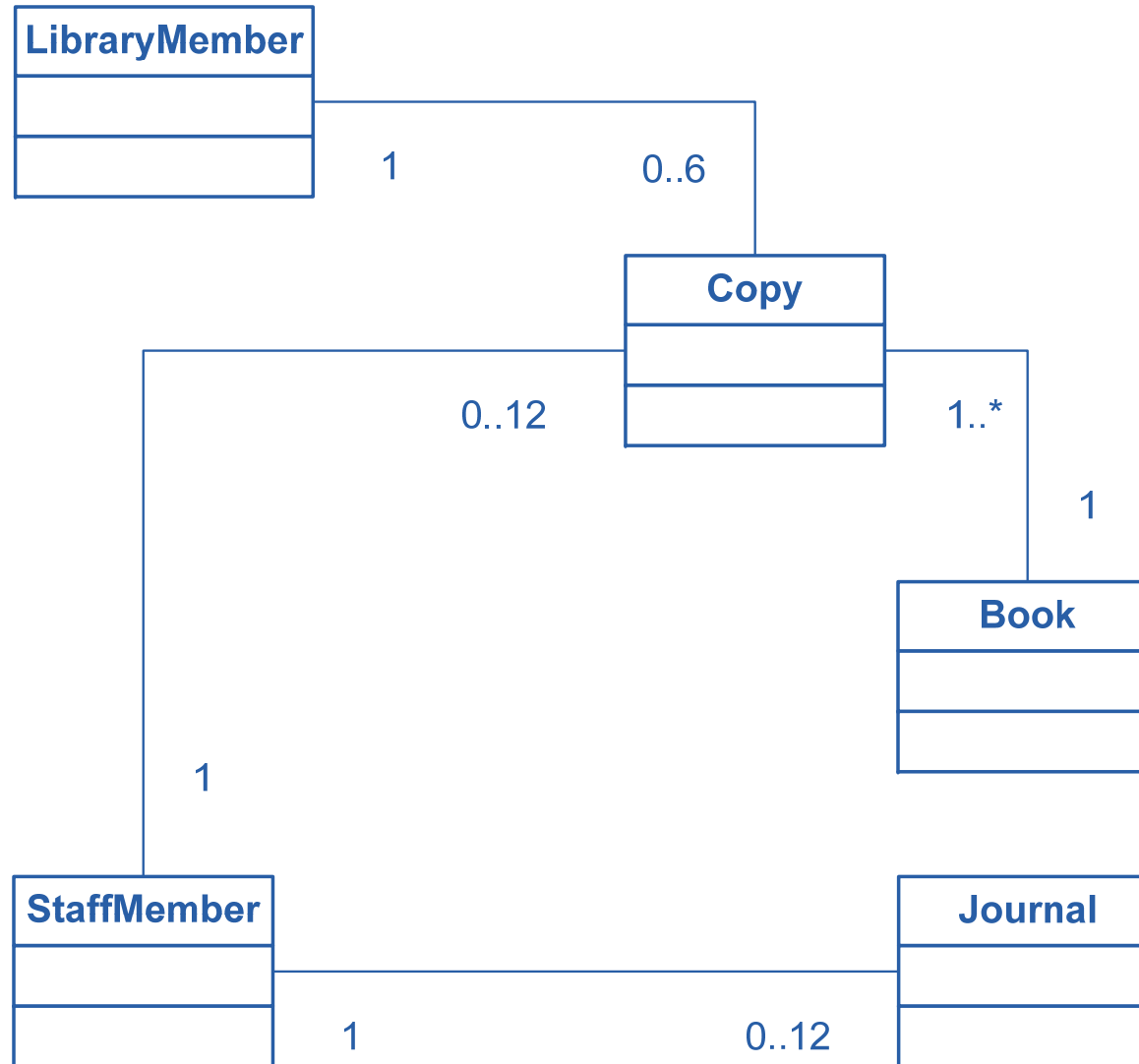
- ◆ Book is an Item
- ◆ Journal is an Item
- ◆ Copy is a copy of a Book
- ◆ StaffMember is a LibraryMember

### Operations

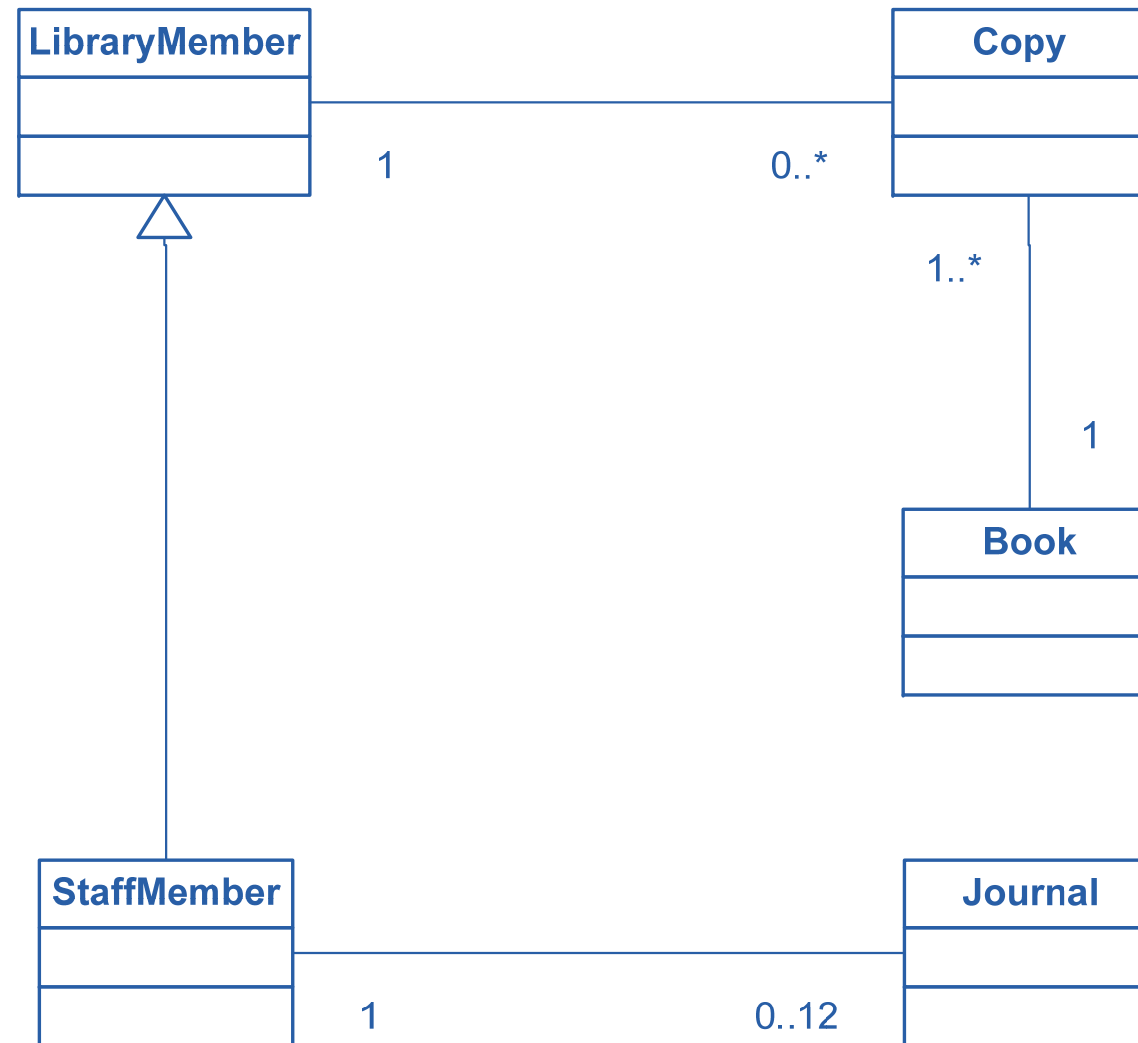
- ◆ LibraryMember borrows Copy
- ◆ LibraryMember returns Copy
- ◆ StaffMember borrows Journal
- ◆ StaffMember returns Journal



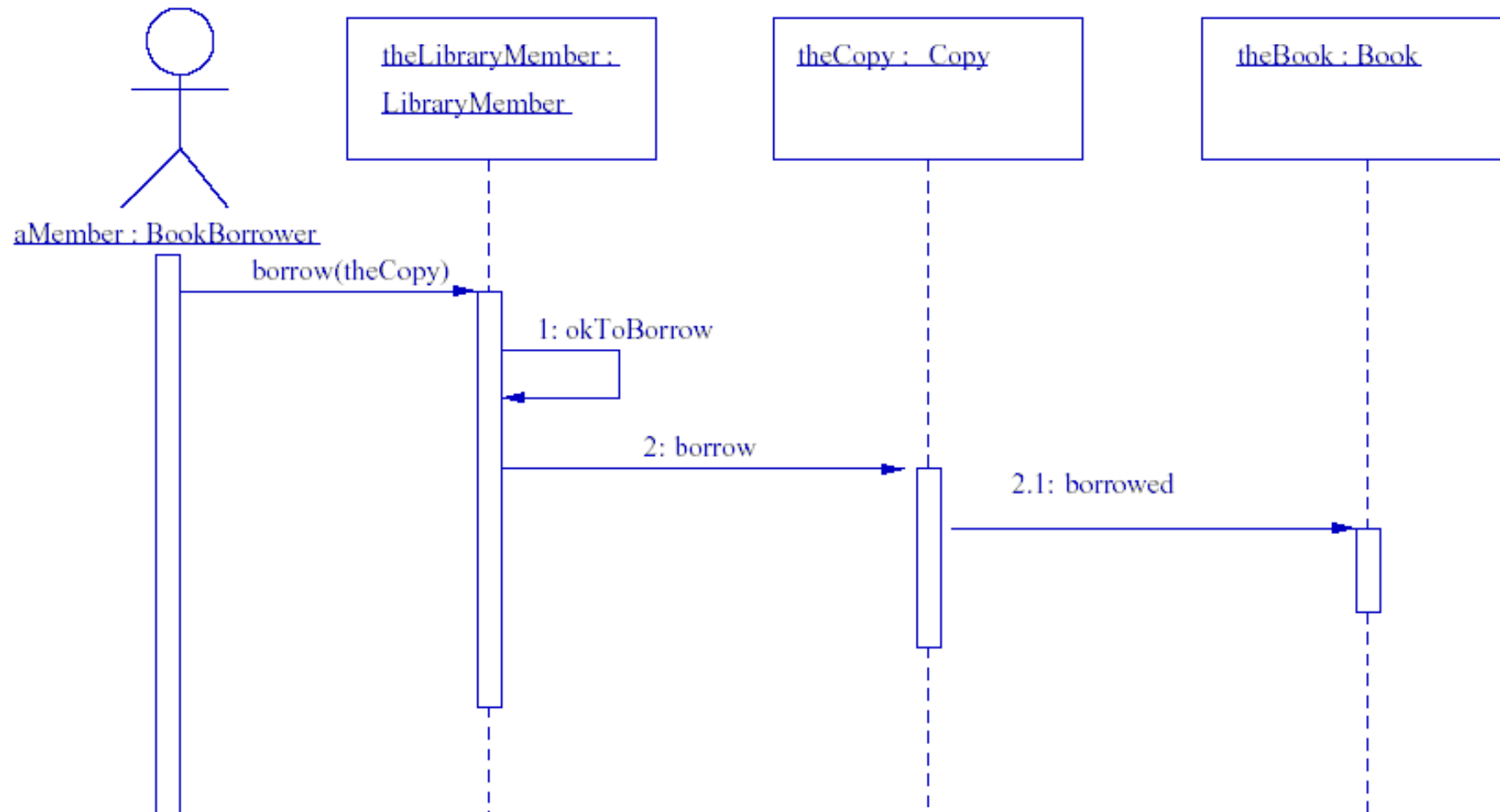
# Library System Class Diagram



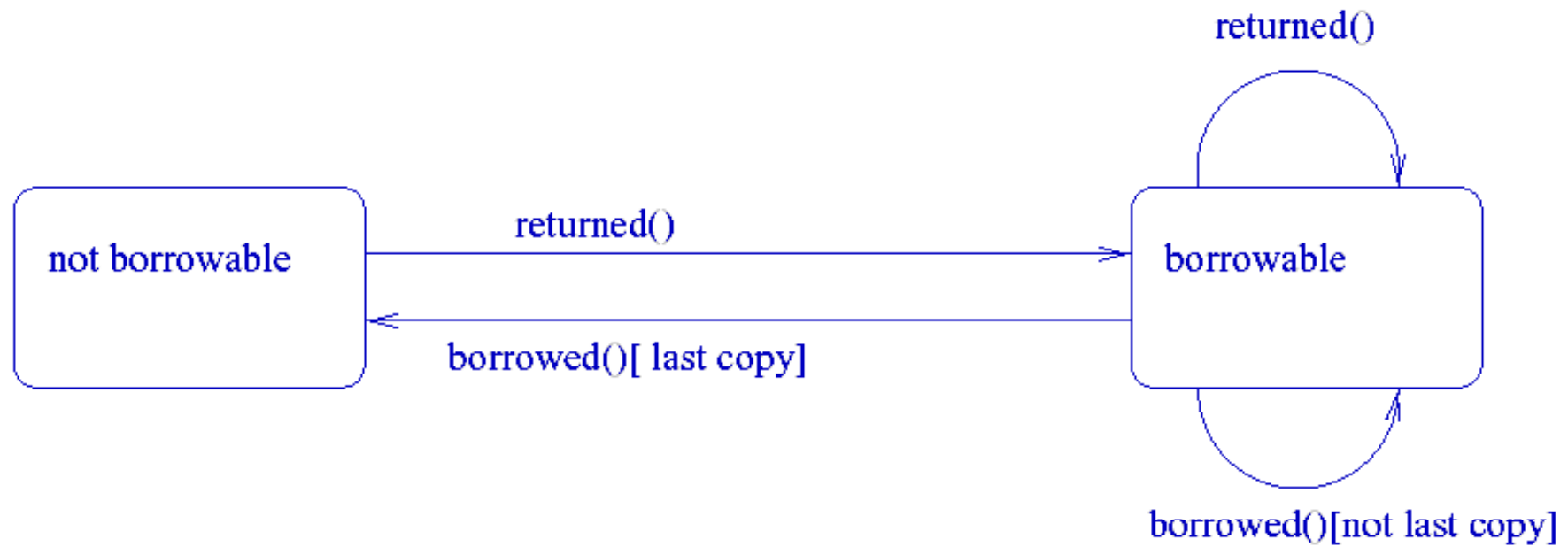
# Library System Class Diagram



## Library Member Borrows a Book Copy Sequence Diagram



## Book Class State Diagram



## From Create Playlist Use Case to Classes and Operations

### Functional Requirements

- ♦ Customer browses through song list database
- ♦ Customer selects songs to add to play list
- ♦ List of songs in playlist is displayed
- ♦ Customer chooses to reorder playlist
- ♦ Customer chooses to delete songs from playlist

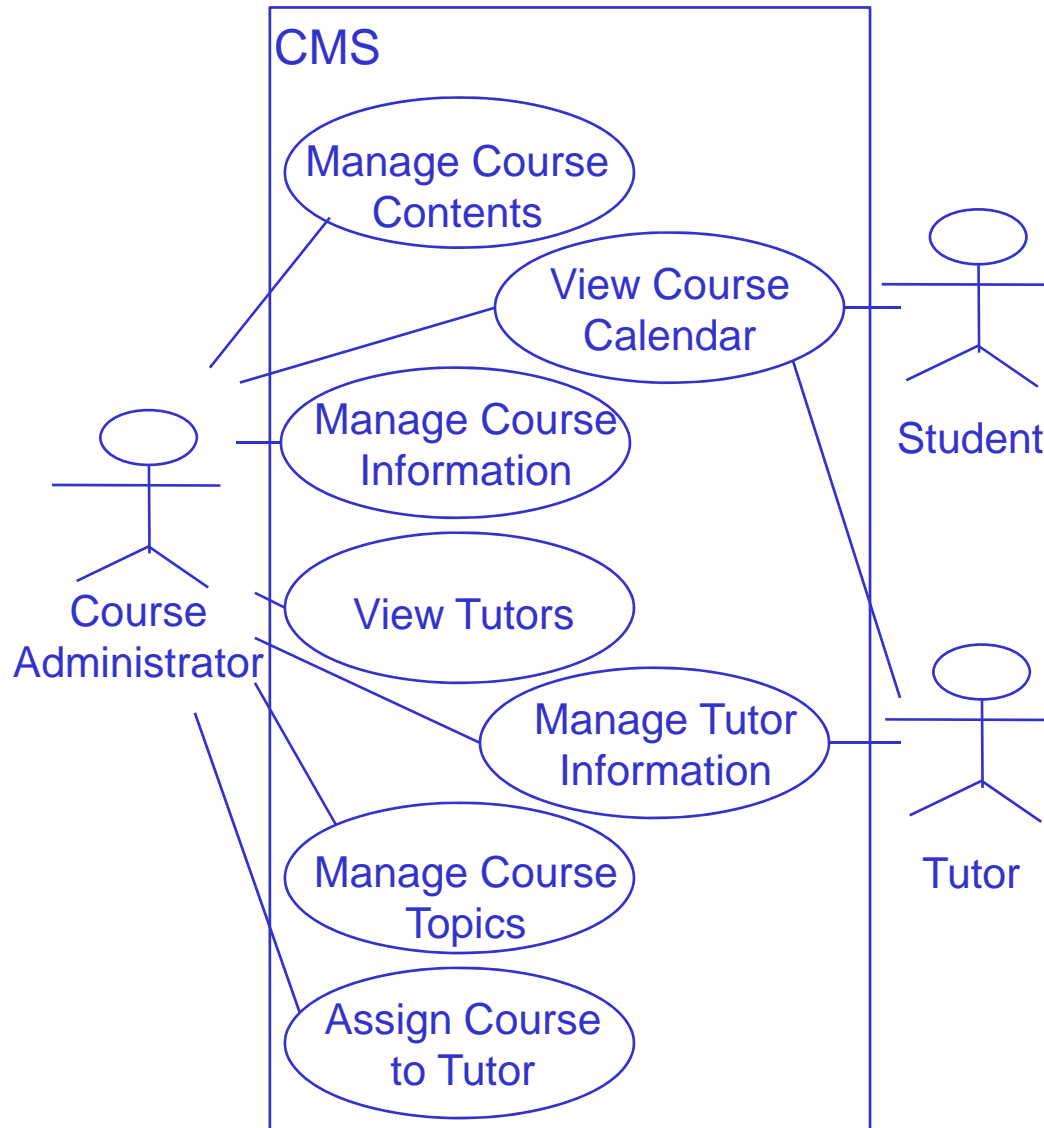
### Classes

- ♦ SongListDatabase
- ♦ Song
- ♦ Playlist

### Operations

- ♦ Playlist.add
- ♦ Playlist.list
- ♦ Playlist.order
- ♦ Playlist.remove

## From CMS Top Use Case Diagram to Classes and Methods



- ◆ CourseAdministrator
- ◆ Tutor
- ◆ Student
- ◆ Course
- ◆ CourseCalendar
- ◆ Topic

Classes

- ◆ viewCourses
- ◆ manageCourseInformation
- ◆ viewCourseCalendar
- ◆ viewTopics
- ◆ manageTopic
- ◆ viewTutors
- ◆ manageTutorInformation
- ◆ assignTutorToCourse

Operations

### Course Operations

- ♦ viewAllCourses
- ♦ viewCourse
- ♦ createCourse
- ♦ modifyCourse
- ♦ removeCourse

### Tutor Operations

- ♦ viewTutor
- ♦ createTutor
- ♦ modifyTutor
- ♦ removeTutor

### Topic Operations

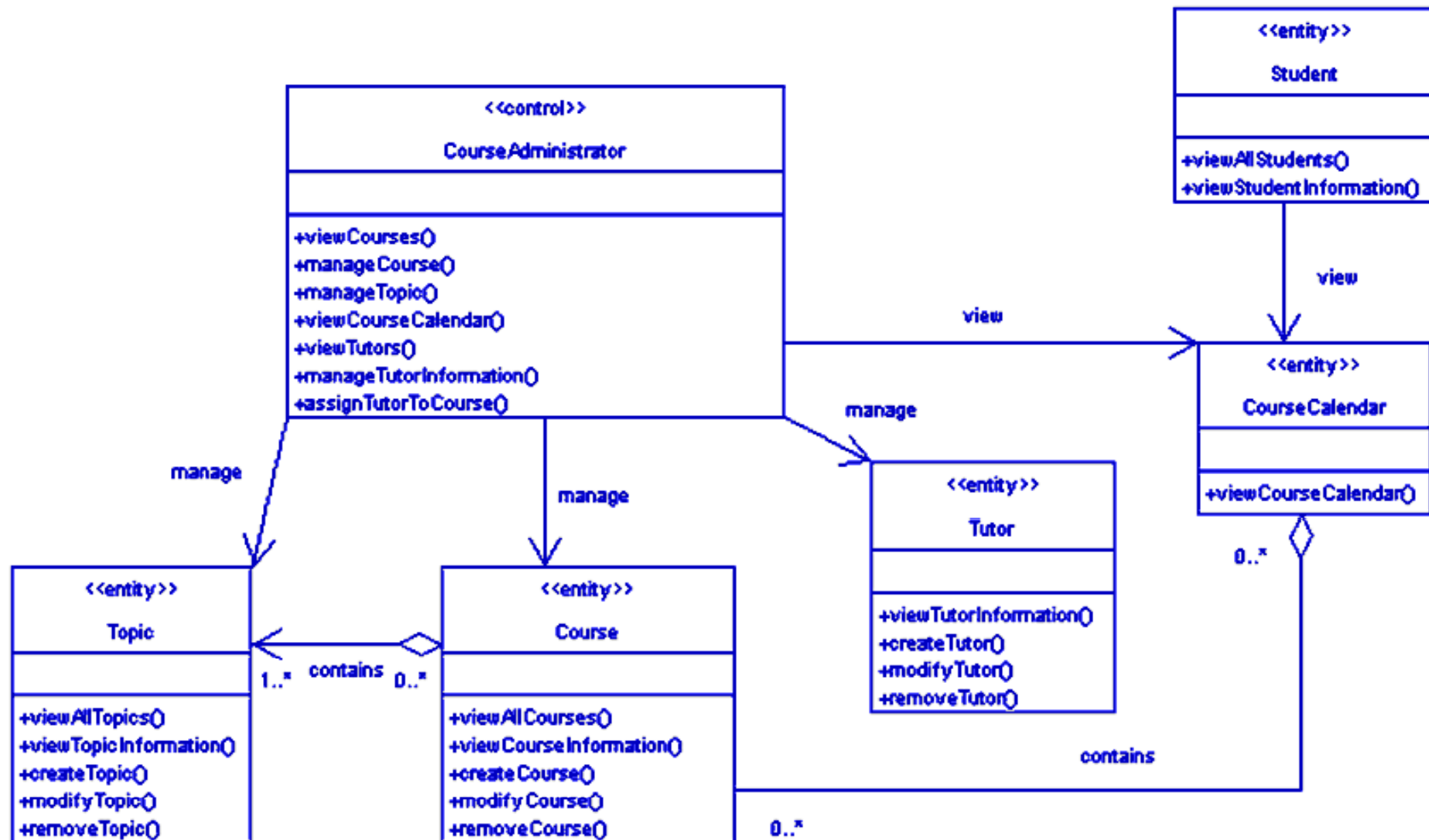
- ♦ viewAllTopics
- ♦ viewTopic
- ♦ createTopic
- ♦ modifyTopic
- ♦ removeTopic

### Course Calendar Operations

- ♦ viewCourseCalendar

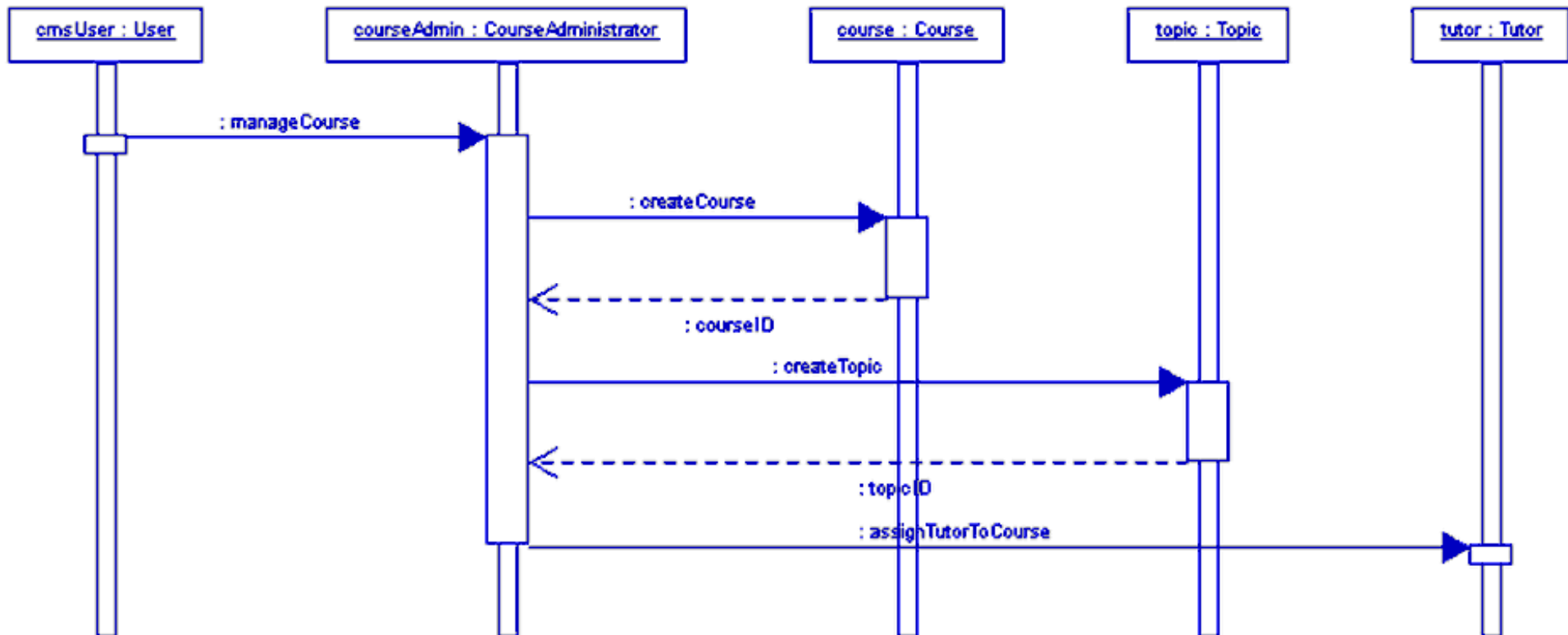
### Student Operations

- ♦ viewAllStudents
- ♦ viewStudentInformation

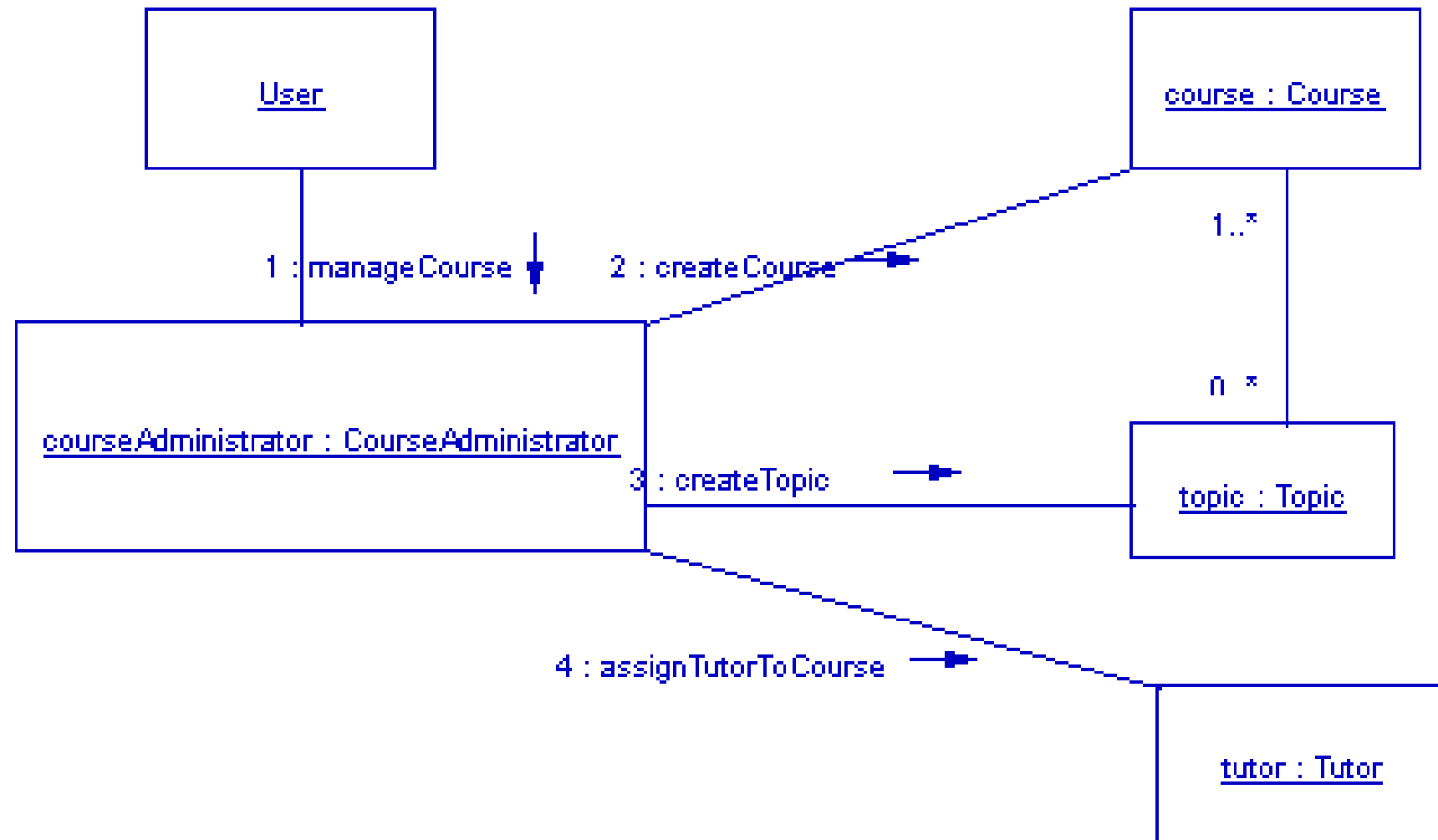




## Manage Course Sequence Diagram



# Manage Course Communication Diagram



- ◆ The process of establishing and justifying the belief that the requirements as documented correspond to what the client organization, the users and the other stakeholders really want
- ◆ Two important issues:
  - Completeness
  - Cost
- ◆ Ideally, validation process should continue as long as necessary, but budget and deadlines constrain it

- ◆ Validity
  - Does the system provide the functions which best support customer's needs?
- ◆ Consistency
  - Are there any requirements conflicts?
- ◆ Completeness
  - Are all functions required by the customer included?
- ◆ Realism
  - Can the requirements be implemented given available budget and technology?
- ◆ Verifiability
  - Can the requirements be checked?

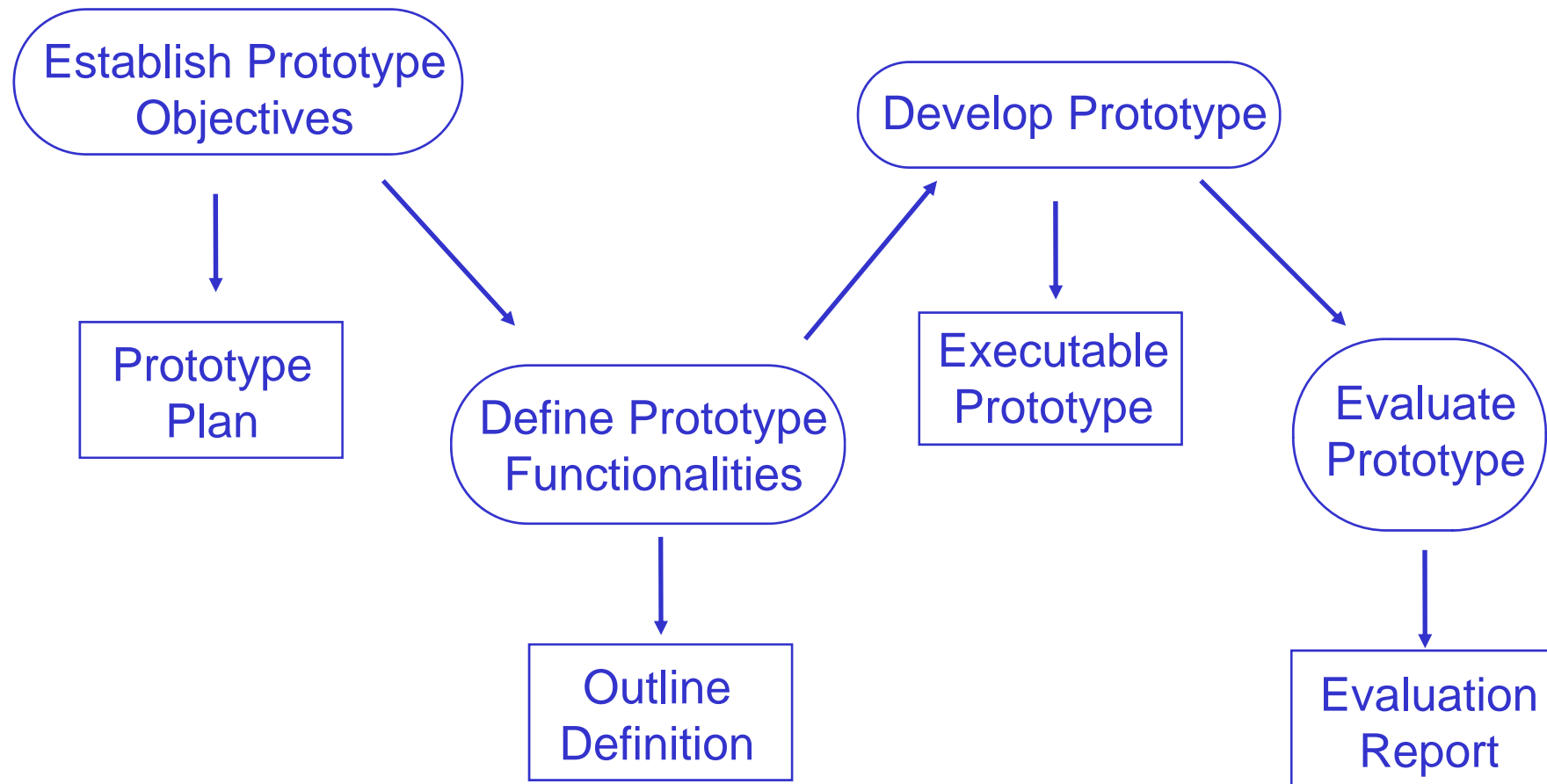
- ◆ Requirements review
  - Systematic manual analysis of the requirements
- ◆ Prototyping
  - Using an executable model of the system to check requirements
- ◆ Test-case generation
  - Developing tests for requirements to check testability

- ◆ Regular reviews should be held while the requirements definition is being formulated
- ◆ Both client and contractor staff should be involved in reviews
- ◆ Reviews may be formal (with completed documents) or informal
  - Good communications between developers, customers and users can resolve problems at an early stage

- ◆ Verifiability
  - Is the requirement realistically testable?
- ◆ Comprehensibility
  - Is the requirement properly understood?
- ◆ Traceability
  - Is the origin of the requirement clearly stated?
- ◆ Adaptability
  - Can the requirement be changed without a large impact on other requirements?

- ◆ A prototype is an initial version of a system used to demonstrate concepts and try out design options
- ◆ A prototype can be used in:
  - The requirements engineering process to help with requirements elicitation and validation
  - In design processes to explore options and develop a UI design
  - In the testing process to run back-to-back tests (i.e., prototype versus real system)
- ◆ A prototype should be discarded after development as it is not a good basis for a production system





- ◆ Inputs to activity: use-case model, detailed use-case descriptions, supplementary requirements, glossary (or business/domain model)
  - Actor interacts by viewing and manipulating elements that represent attributes of use cases
  - Assure each use case is accessible to the actors through the user interface
  - Assure well-integrated, easy-to-use, consistent, navigable user interfaces
    - Analyze usability; don't be fooled by wording of use case
- ◆ Build physical prototype to validate UI and the use cases

- ◆ Improved system usability
- ◆ A closer match to users' real needs
- ◆ Improved design quality
- ◆ Improved maintainability
- ◆ Reduced development effort

- ◆ Developing tests for requirements to check testability
  - Test cases are defined for each requirement, or shall statement
  - Tests are executed manually or automatically through specific test automation tool
- ◆ Tests difficult to implement reveal potential difficulty of implementing requirements

- ◆ Requirements management is the process of managing changing requirements during the requirements engineering process and system development
- ◆ Requirements are inevitably incomplete and inconsistent:
  - The priority of requirements from different viewpoints changes during the development process
  - Customers may specify requirements from a business perspective that conflict with end-user requirements
  - The business and technical environment of the system changes during its development

- ◆ Keep project within costs, within budget, and to meet customers needs
- ◆ Estimate cost of system based on requirements
- ◆ Control the volatility of the requirements
- ◆ Manage the requirements configuration of the system
- ◆ Negotiate requirement changes
- ◆ Re-estimate cost of the system when requirements change

- ◆ Requirements identification
  - How requirements are individually identified
- ◆ A change management process
  - The process followed when analyzing a requirements change
- ◆ Traceability policies
  - The amount of information about requirements relationships that is maintained
- ◆ CASE tool support
  - The tool support required to help manage requirements change

- ◆ Should apply to all proposed changes to the requirements
- ◆ Based on three main steps:
  - Problem analysis
    - Discuss requirements problem and propose change
  - Change analysis and costing
    - Assess effects of change on other requirements
  - Change implementation
    - Modify requirements document and other documents to reflect change



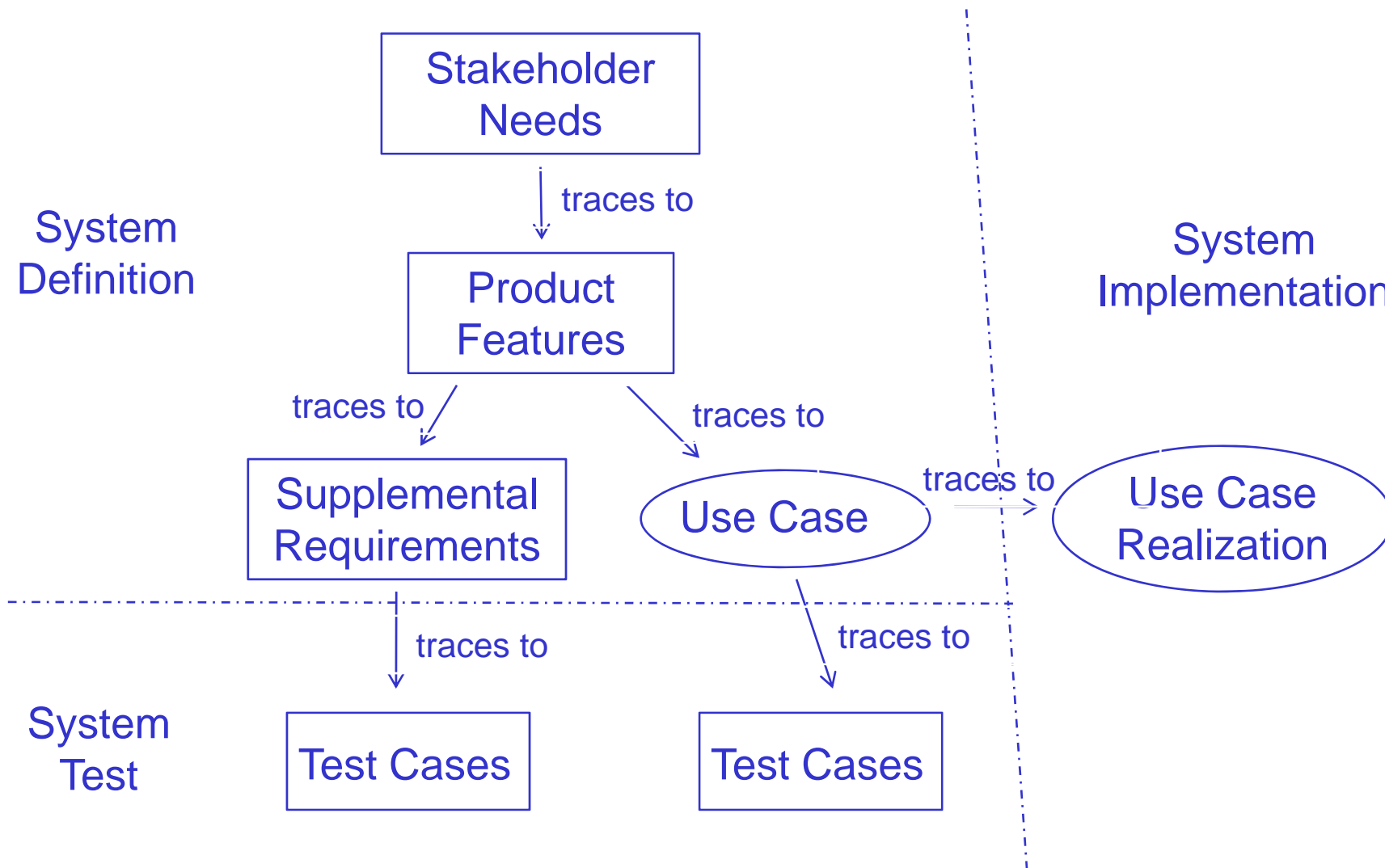
- ◆ Requirements storage
  - Requirements should be managed in a secure, managed data store
- ◆ Change management
  - The process of change management is a workflow process whose stages can be defined and information flow between these stages partially automated
- ◆ Traceability management
  - Automated retrieval of the links between requirements

- ◆ The degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another
  - For example, the degree to which the requirements and design of a given software component match
- ◆ The ability to trace requirements artifacts through the stages of specification, architecture, design, implementation, and testing is a significant factor in assuring a quality software implementation

- ♦ Tracing user needs to product features
- ♦ Tracing features to use cases
- ♦ Tracing features to supplementary requirements
- ♦ Tracing requirements to implementation
- ♦ Tracing use cases to use-case realizations

- ◆ Tracing from the use-case realization into implementation
- ◆ Tracing supplementary requirements into implementation
- ◆ Tracing from requirements to test
- ◆ Tracing from use case to test case

# Generalized Traceability Hierarchy



|                 | Feature 1 | Feature 2 | ... | Feature n |
|-----------------|-----------|-----------|-----|-----------|
| <b>Need #1</b>  | X         |           |     |           |
| <b>Need #2</b>  |           | X         |     | X         |
| <b>Need ...</b> |           | X         | X   |           |
| <b>Need #m</b>  |           |           |     | X         |

|                    | Use case 1 | Use case 2 | ... | Use case n |
|--------------------|------------|------------|-----|------------|
| <b>Feature #1</b>  | X          |            |     | X          |
| <b>Feature #2</b>  |            | X          |     | X          |
| <b>Feature ...</b> |            |            | X   |            |
| <b>Feature #m</b>  |            | X          |     | X          |

| Use Case           | Scenario Number | ... Test Case Id |
|--------------------|-----------------|------------------|
| <b>Use Case #1</b> | 1               | 1.1              |
|                    | 2               | 2.1              |
|                    | 3               | 3.1              |
|                    | 4               | 4.1              |
|                    | 4               | 4.2              |
|                    | 4               | 4.3              |
|                    | 5               | 5.1              |
|                    | 6               | 6.1              |
|                    | 7               | 7.1              |
|                    | 7               | 7.2              |
|                    | 8               | 8.1              |
| <b>Use Case #2</b> | 1               | 1.1              |

- ◆ Lack of understanding of the domain or the real problem
  - Do domain analysis and prototyping
- ◆ Requirements change rapidly
  - Perform incremental development, build flexibility into the design, do regular reviews
- ◆ Attempting to do too much
  - Document the problem boundaries at an early stage, carefully estimate the time



- ◆ It may be hard to reconcile conflicting sets of requirements
  - Brainstorming, competing prototypes
  
- ◆ It is hard to state requirements precisely
  - Break requirements down into simple sentences and review them carefully, look for potential ambiguity, make early prototypes