Requirement and Analysis Using UML

List of Material

- Requirement Engineering Process
- Requirement Elicitation
- Identifying Use Case
- CRC Card
- Class and Its Relation
- Statechart Diagram

Requirement Engineering Process

What is a requirement?

- 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.

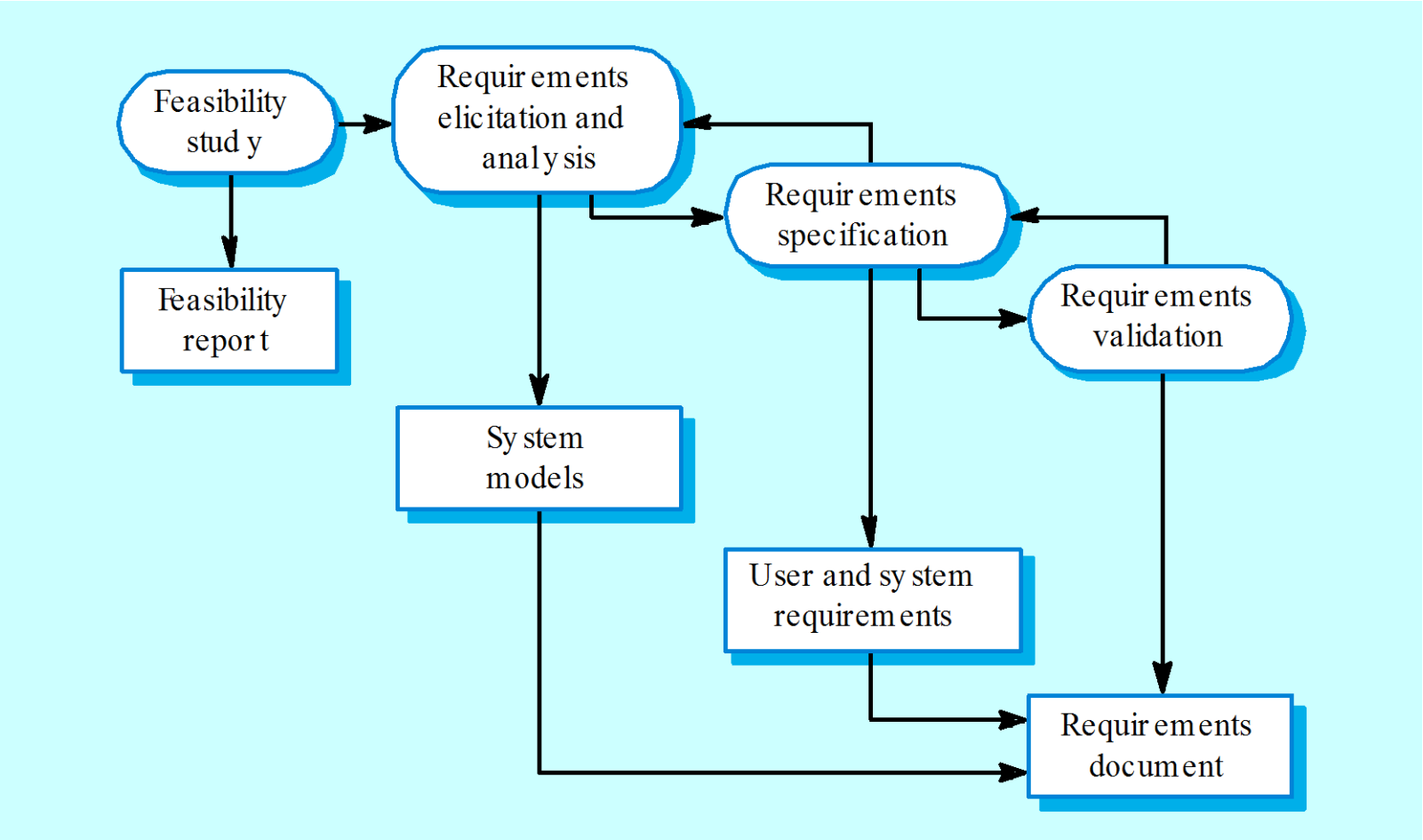
Requirements engineering

- The process of establishing the services that the customer requires from a system and the constraints under which it operates and is developed.
- The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.

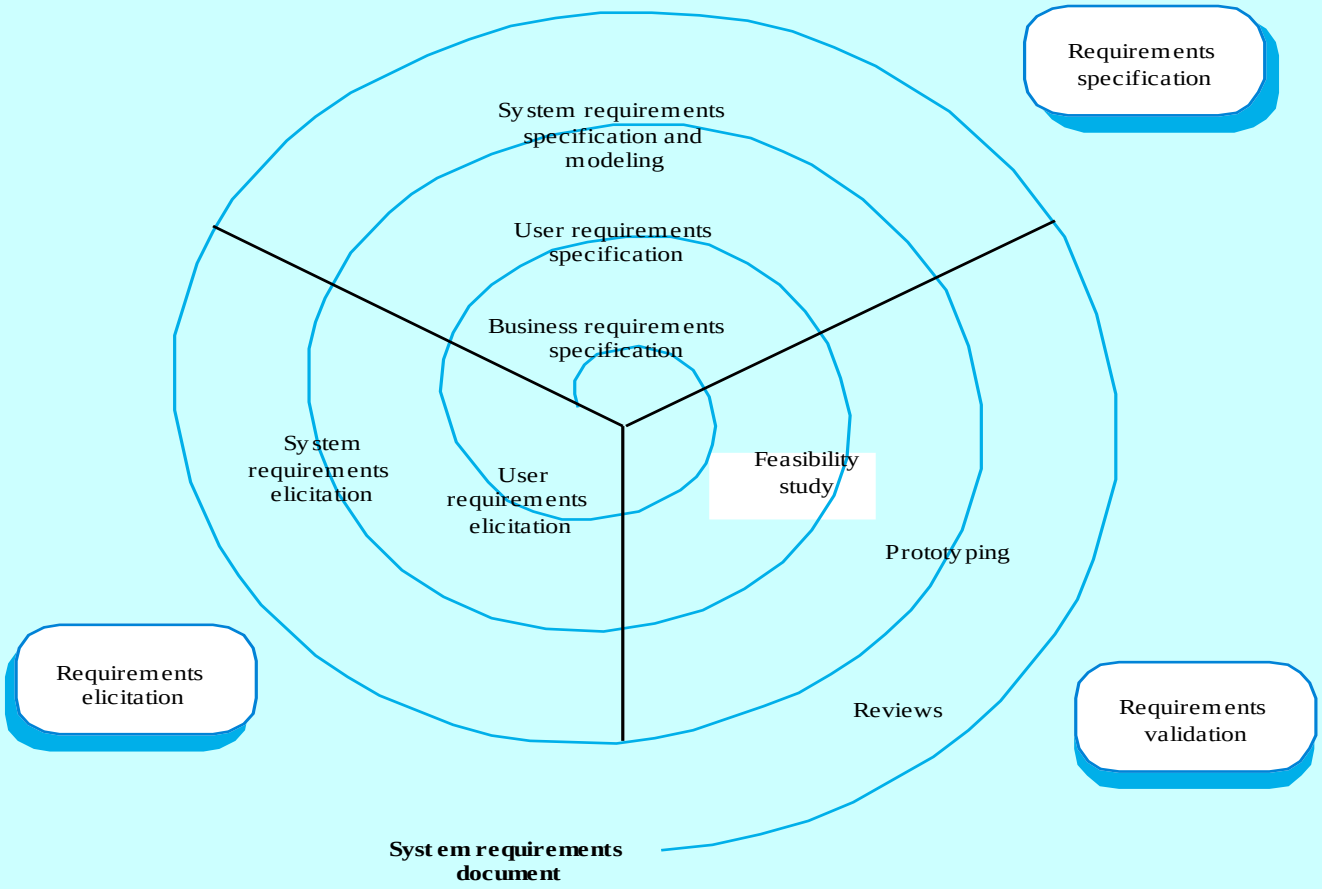
Requirements engineering processes

- The processes used for RE vary widely depending on the application domain, the people involved and the organisation developing the requirements.
- However, there are a number of generic activities common to all processes
  - Requirements elicitation;
  - Requirements analysis;
  - Requirements validation;
  - Requirements management.

The requirements engineering process



Requirements engineering



Feasibility studies

- 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.

Feasibility study implementation

- 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?

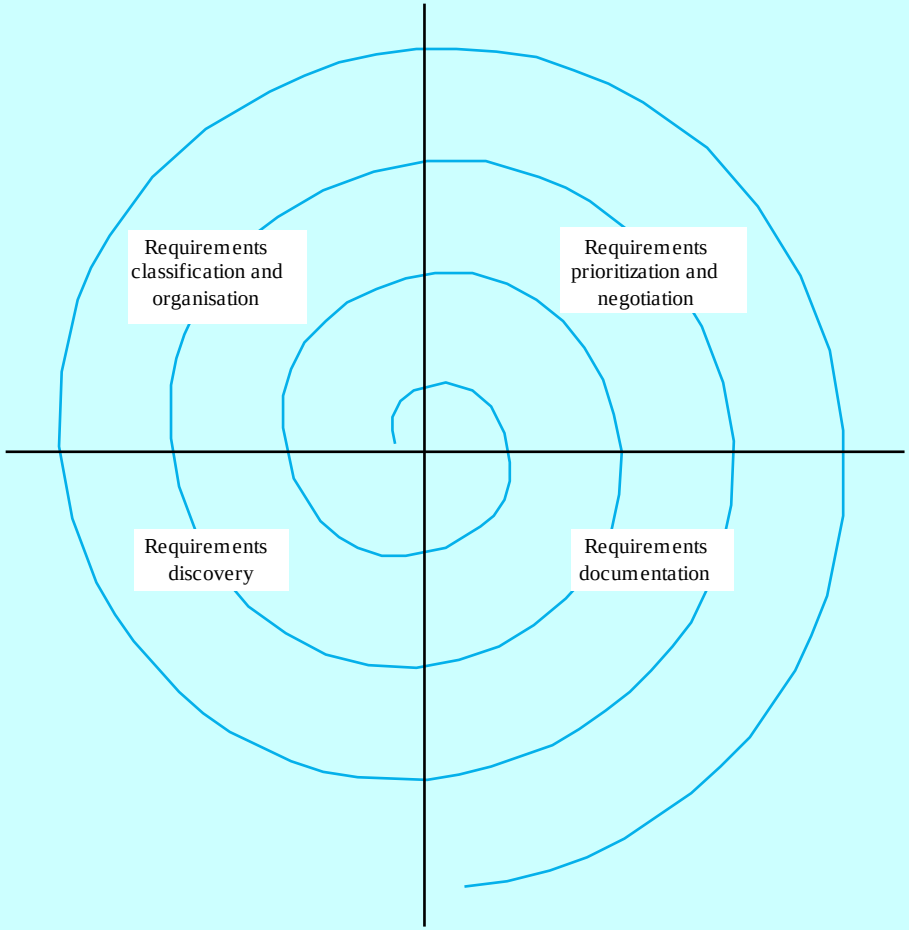
Elicitation and analysis

- Sometimes called requirements elicitation or requirements discovery.
- Involves technical staff working with customers to find out about the application domain, the services that the system should provide and the system's operational constraints.
- May involve end-users, managers, engineers involved in maintenance, domain experts, trade unions, etc. These are called *stakeholders.*

Problems of requirements analysis

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

The requirements spiral



Process activities

- Requirements discovery
  - Interacting with stakeholders to discover their requirements. Domain requirements are also discovered at this stage.
- Requirements classification and organisation
  - Groups related requirements and organises them into coherent clusters.
- Prioritisation and negotiation
  - Prioritising requirements and resolving requirements conflicts.
- Requirements documentation
  - Requirements are documented and input into the next round of the spiral.

Requirements discovery

- The process of gathering information about the proposed and existing systems and distilling the user and system requirements from this information.
- Sources of information include documentation, system stakeholders and the specifications of similar systems.

Requirements abstraction (Davis)

"If a company wishes to let a contract for a large software development project, it must define its needs in a sufficiently abstract way that a solution is not pre-defined. The requirements must be written so that several contractors can bid for the contract, offering, perhaps, different ways of meeting the client organisation's needs. Once a contract has been awarded, the contractor must write a system definition for the client in more detail so that the client understands and can validate what the software will do. Both of these documents may be called the *requirements document* for the system."

Types of requirement

- User requirements
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written **for customers**
- System requirements
  - A structured document setting out detailed descriptions of the system services. Written as a **contract** between client and contractor
- Software specification
  - A detailed software description which can serve as a basis for a design or implementation. Written **for developers**

Definitions and specifications

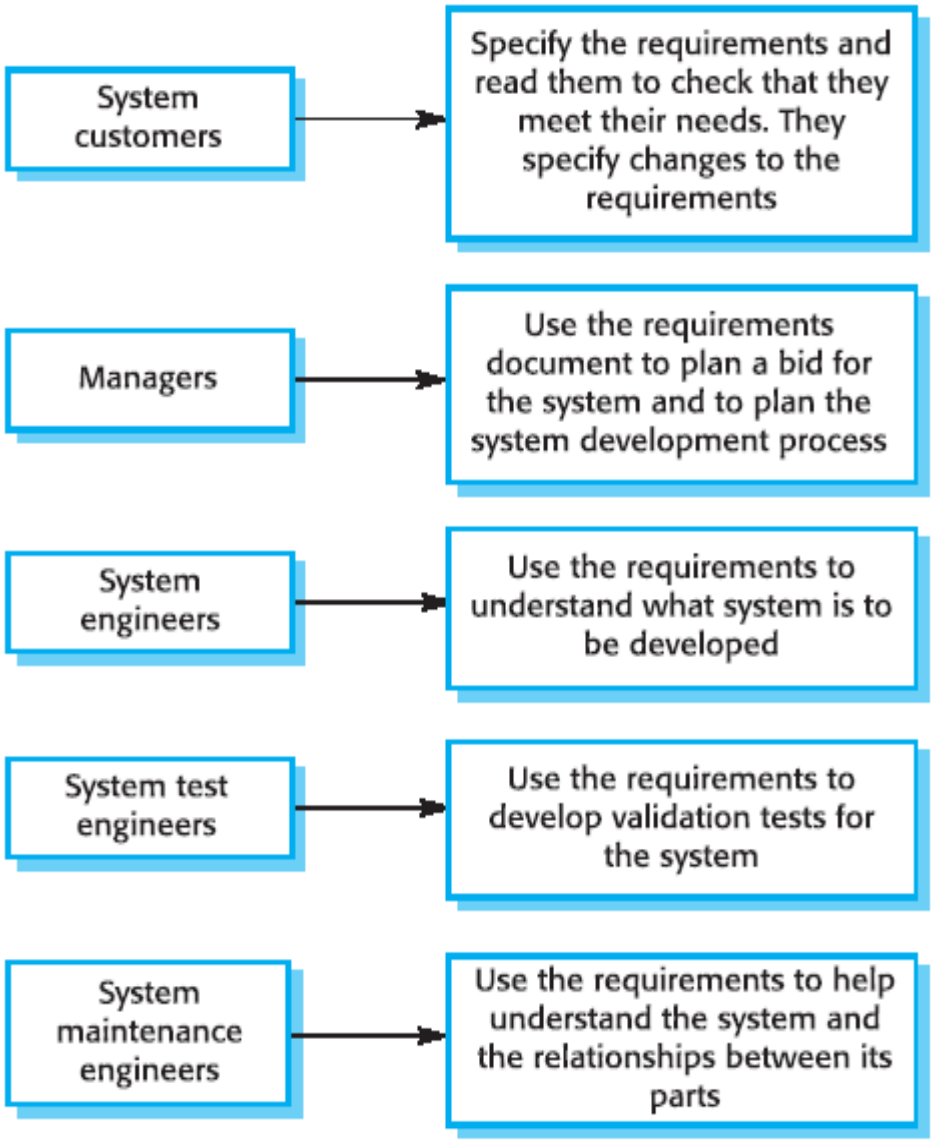
Requirements definition

1. The software must provide a means of representing and accessing external files created by other tools.

Requirements specification

1.1 The user should be provided with facilities to define the type of external files.
1.2 Each external file type may have an associated tool which may be applied to the file.
1.3 Each external file type may be represented as a specific icon on the user's display.
1.4 Facilities should be provided for the icon representing an external file type to be defined by the user.
1.5 When a user selects an icon representing an external file, the effect of that selection is to apply the tool associated with the type of the external file to the file represented by the selected icon.

Requirements readers



Another classification of requirements

- Functional requirements
- Non-functional requirements
- Domain requirements

Functional requirements (1)

- Describe functionality or system services, how the system should react to particular inputs and how the system should behave in particular situations.
- Depend on the type of software, expected users and the type of system where the software is used
- Functional user requirements may be high-level statements of what the system should do but functional system requirements should describe the system services in detail

Functional requirements (2)

- Examples:
  - The user shall be able to search either all of the initial set of databases or select a subset from it.
  - The system shall provide appropriate viewers for the user to read documents in the document store.
  - Every order shall be allocated a unique identifier (ORDER_ID) which the user shall be able to copy to the account's permanent storage area.

Functional requirements (3)

- 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’ in the previous example
  - User intention - special purpose viewer for each different document type
  - Developer interpretation - Provide a text viewer that shows the contents of the document

Functional requirements (4)

- Requirements should be 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 impossible to produce a complete and consistent requirements document

Non-functional requirements

- Define system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system representations, etc.
- Can be constraints on the process too
  - Use a particular CASE system, programming language or development method
- System maybe unusable if non-functional requirements are not satisfied (Critical)

Non-functional classifications

- Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- Organisational requirements
  - Requirements which are a consequence of organisational 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 examples

- Product requirement
  - *4.C.8 It shall be possible for all necessary communication between the APSE and the user to be expressed in the standard Ada character set*
- Organisational requirement
  - *9.3.2 The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95*
- External requirement
  - *7.6.5 The system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system*

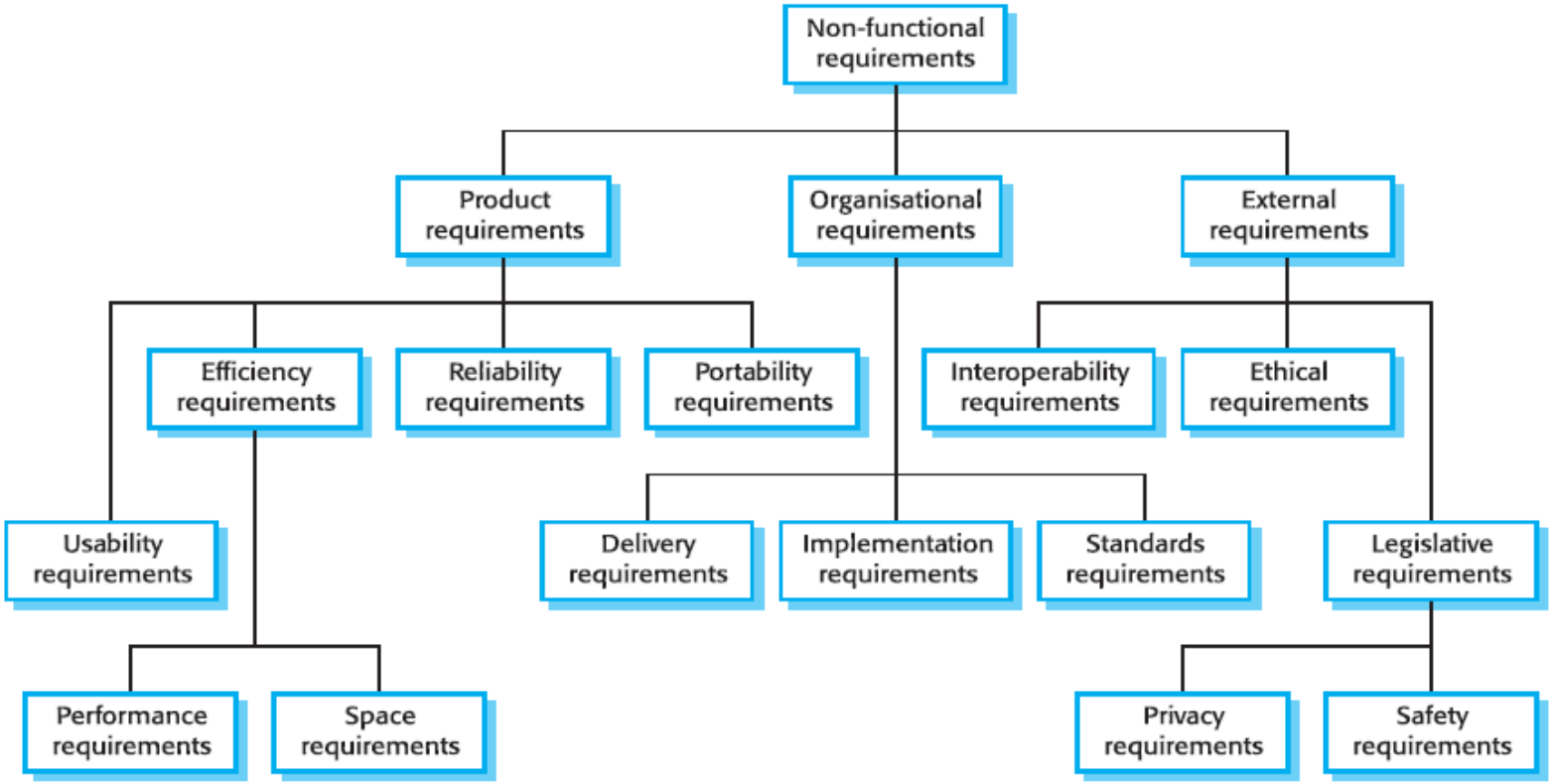
Goals and requirements

- 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

System Goal VS Non Functional

- **A system goal**
  - *The system should be easy to use by experienced controllers and should be organised in such a way that user errors are minimised*.
- **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.*

List of Non Functional Req.



Gambar 2: Cakupan dari Non-Functional requirement

Requirement Measures

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	K Bytes Number of RAM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Domain requirements

- Derived from the application domain and describe system characterisics and features that reflect the domain
- May be new functional requirements, constraints on existing requirements or define specific computations
- If domain requirements are not satisfied, the system may be unworkable

Domain requirements (examples)

- Library system
  - *There shall be a standard user interface to all databases which shall be based on the Z39.50 standard.*
  - *Because of copyright restrictions, some documents must be deleted immediately on arrival. Depending on the user's requirements, these documents will either be printed locally on the system server for manually forwarding to the user or routed to a network printer*.
- Train Protection system
  - *The deceleration of the train shall be computed as:*

    $D_{train} = D_{control} + D_{gradient}$

    where $D_{gradient}$ *is* $9.81ms^2$ * *compensated gradient/alpha and where the values of* $9.81ms^2$ */alpha are known for different types of train*

User requirements

- Should describe functional and non-functional requirements so that they *are understandable by system users* who don’t have detailed technical knowledge
- User requirements are defined using natural language, tables and diagrams

Problems with natural language

- 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

Identifying Use Case Diagrams

Developing Use Case Diagrams

- Review the business specifications and identify the actors involved
- Identify the high-level events and develop the primary use cases that describe those events and how the actors initiate them
- Review each primary use case to determine the possible variations of flow through the use case
- The context-level data flow diagram could act as a starting point for creating a use case

Define use cases

Use cases - narrative descriptions of domain processes in a structured prose format

- **Use case : Play a game**
- **Actors : Player**

- **Description: This use case begins when the player picks up and rolls the die….**

Define domain model

- OO Analysis concerns
  - specification of the problem domain
  - identification of concepts (objects)
- Decomposition of the problem domain includes
  - identification of objects, attributes, associations
- Outcome of analysis expressed as a domain model.

Developing the Use Case Scenarios

- The description of the use case
- Three main areas
  - Use case identifiers and initiators
  - Steps performed
  - Conditions, assumptions, and questions

High level vs. Low Level Use cases (1)

- Consider the following use cases:
  - Log out
  - Handle payment
  - Negotiate contract with a supplier
- These use cases are at different levels. Are they all valid?

  To check, use the EBP definition.

High level vs. Low Level Use cases (2)

- Log out: a secondary goal; it is necessary to do something but not useful in itself.
- Handle payment: A necessary EBP. Hence a primary goal.
- Negotiate contract: Most likely this is too high a level. It is composed of several EBPs and hence must be broken down further.

More On Use Case

- Narrate use cases independent of implementation
- State success scenarios (how do you determine the success of a use case).
- A use case corresponds to one or more scenarios.
- Agree on a format for use case description
- Name a use case starting with a verb in order to emphasize that it is a process (Buy Items, Enter an order, Reduce inventory)

Why Use Case Diagrams Are Helpful

- Identify all the actors in the problem domain
- Actions that need to be completed are also clearly shown on the use case diagram
- The use case scenario is also worthwhile
- Simplicity and lack of technical detail

The Main Reason for Writing Use Case

- Use cases effectively communicate systems requirements because the diagrams are kept simple.
- Use cases allow people to tell stories.
- Use case stories make sense to nontechnical people.
- Use cases do not depend on a special language.
- Use cases can describe most functional requirements (such as interactions between actors and applications).
- Use cases can describe nonfunctional requirements (such as performance and maintainability) through the use of stereotypes.
- Use cases help analysts define boundaries.
- Use cases can be traceable, allowing analysts to identify links between use cases and other design and documentation tools.

CRC Card

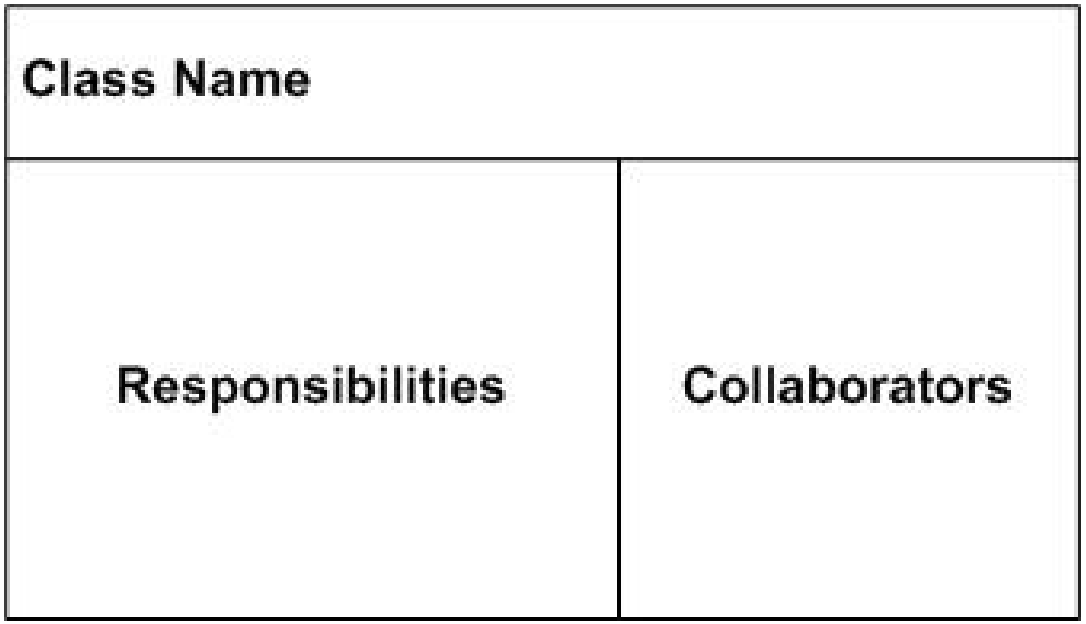
CRC Cards and Object Think

- CRC
  - Class
  - Responsibilities
  - Collaborators
- CRC cards are used to represent the responsibilities of classes and the interaction between the classes

Interacting during a CRC Session

- Identify all the classes you can
- Creating scenarios
- Identify and refine responsibilities

CRC Layout



Student CRC Card

Student	
Student number Name Address Phone number Enroll in a seminar Drop a seminar Request transcripts	Seminar

CRC Model

Professor	
Name Address Phone number Email address Salary Provide information Seminars instructing	Seminar

Transcript	
See the prototype Determine average mark	Student Seminar Professor Enrollment

Seminar	
Name Seminar number Fees Waiting list Enrolled students Instructor Add student Drop student	Student Professor

Enrollment	
Mark(s) received Average to date Final grade Student Seminar	Seminar

Student Schedule	
See the prototype	Seminar Professor Student Enrollment Room

Student	
Name Address Phone number Email address Student number Average mark received Validate identifying info Provide list of seminars taken	Enrollment

Room	
Building Room number Type (Lab, class, ...) Number of Seats Get building name Provide available time slots	Building

Building	
Building Name Rooms Provide name Provide list of available rooms for a given time period	Room

How to Create CRC Diagram

- **Find classes**.
- **Find responsibilities**.
- **Define collaborators**.
- **Move the cards around**. To improve everyone's understanding of the system, the cards should be placed on the table in an intelligent manner.

Class and Its Relation

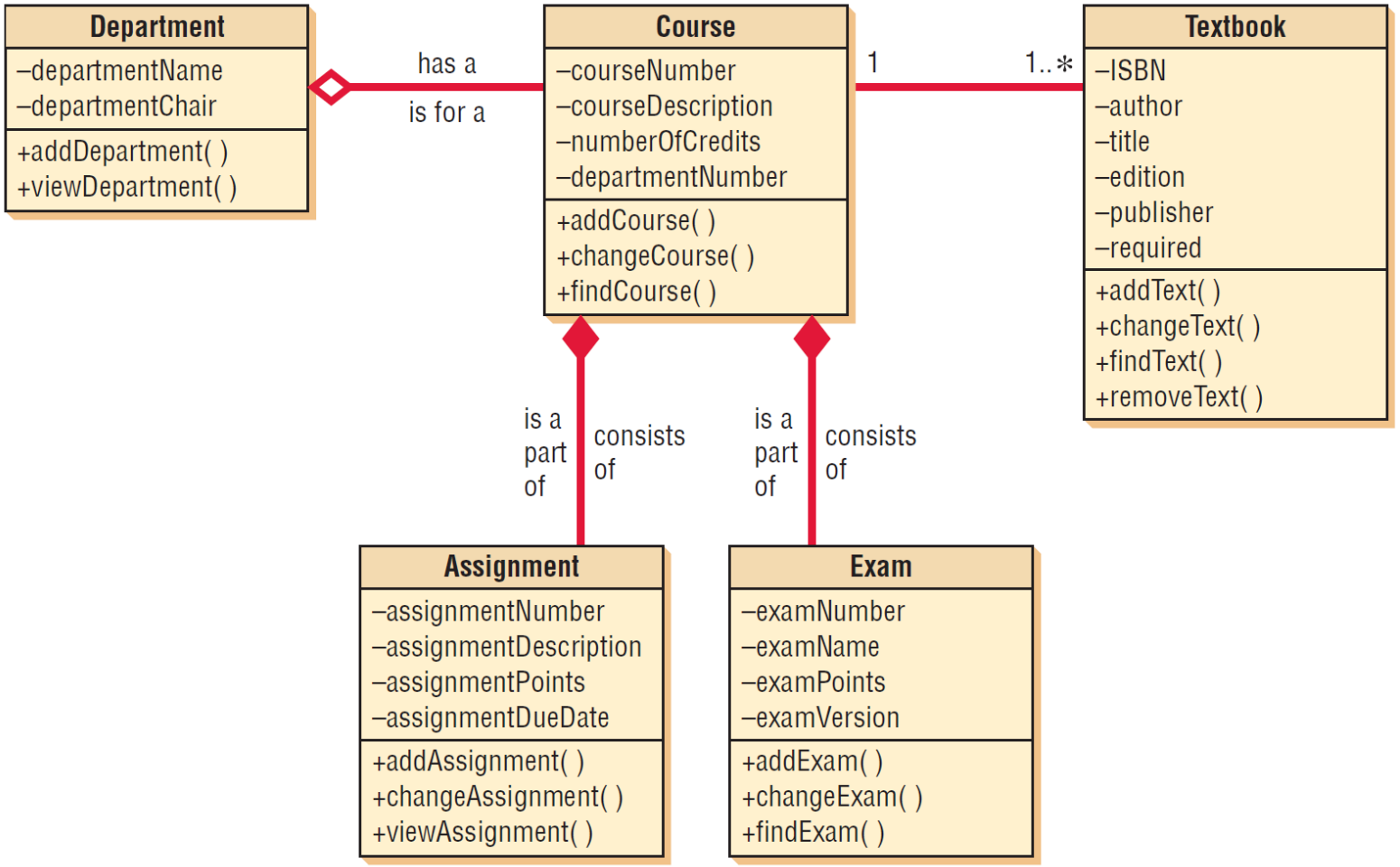
Class Diagrams (1)

- Show the static features of the system and do not represent any particular processing
- Shows the nature of the relationships between classes
- Shows data storage requirements as well as processing requirements

Class Diagrams (2)

- Classes
- Attributes
  - Private
  - Public
  - Protected
- Methods
  - Standard
  - Custom

Example of Class Diagrams



Method Overloading

- Including the same method (or operation) several times in a class
- The same method may be defined more than once in a given class, as long as the parameters sent as part of the message are different

Types of Classes

- Entity classes
- Interface classes
- Abstract classes
- Control classes

Entity Classes

- Represent real-world items
- The entities represented on an entity-relationship diagram

Interface or Boundary Classes

- Provide a means for users to work with the system
- Human interfaces may be a display, window, Web form, dialogue box, touch-tone telephone, or other way for users to interact with the system
- System interfaces involve sending data to or receiving data from other

Abstract Classes

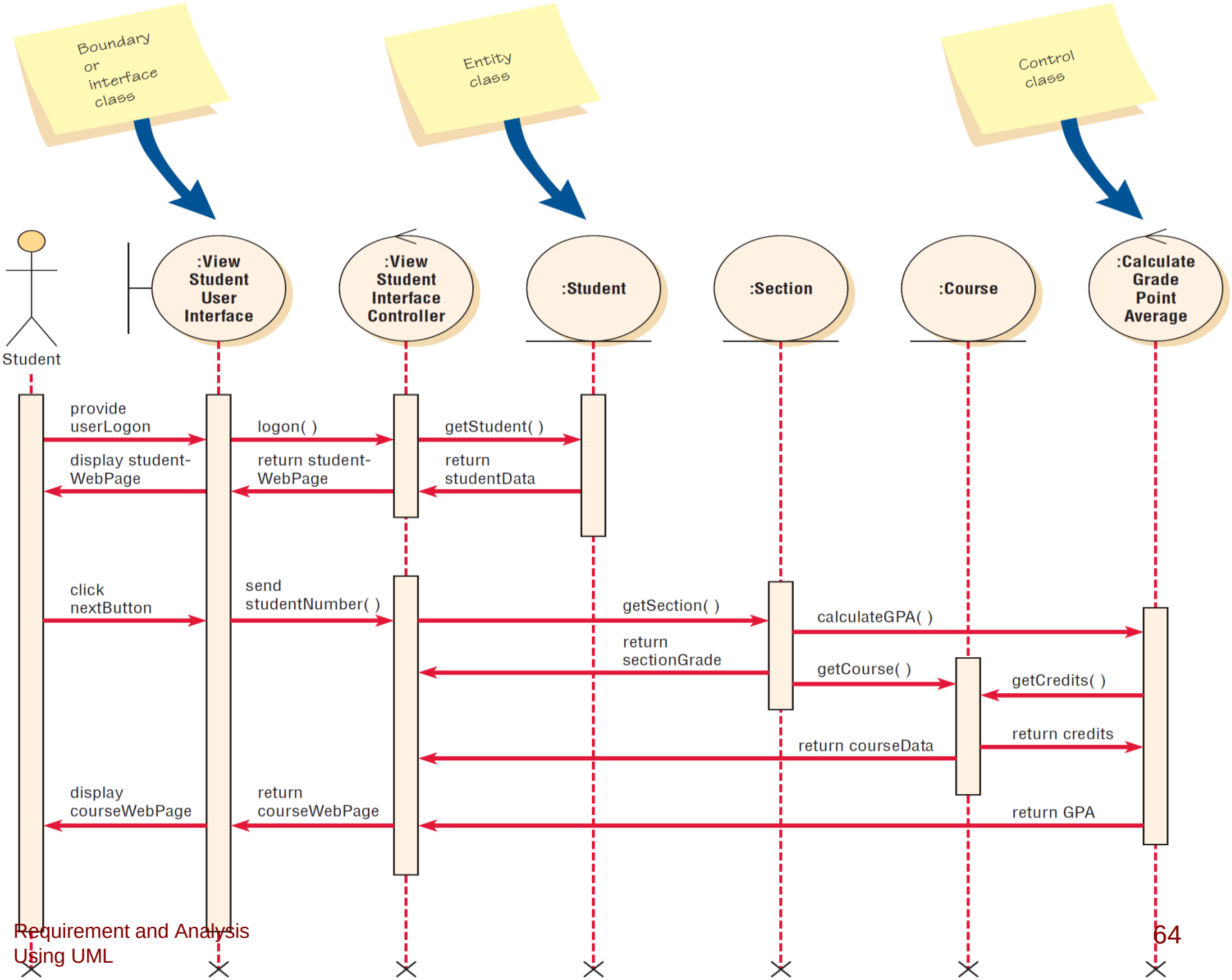
- Linked to concrete classes in a generalization/specialization relationship
- Cannot be directly instantiated

Control Classes

- Used to control the flow of activities
- Many small control classes can be used to achieve classes that are reusable

Defining Messages and Methods

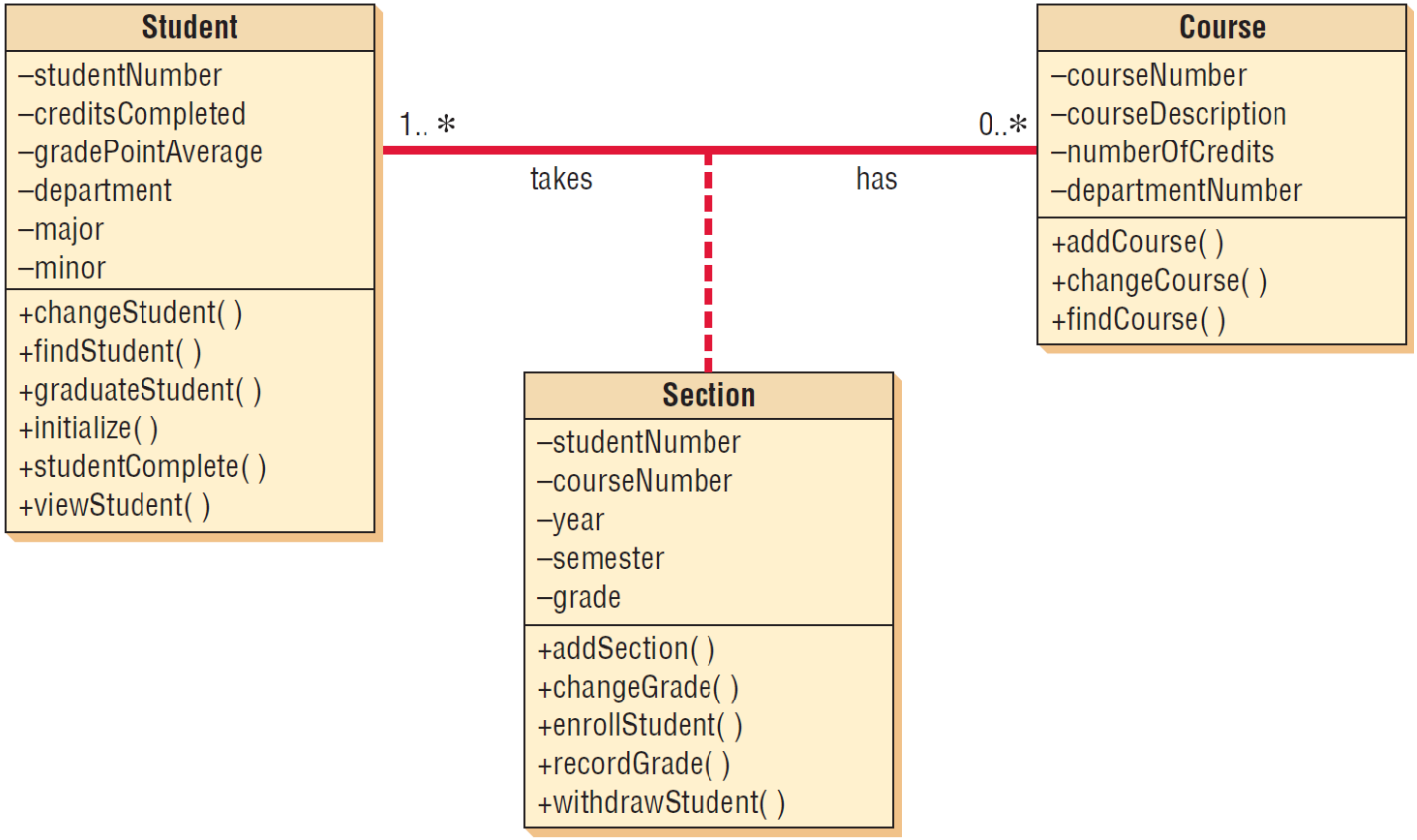
- Each message may be defined using a notation similar to that described for the data dictionary
- The methods may have logic defined using structured English, a decision table, or a decision tree


Boundary or interface class
Entity class
Control class
Student
:View Student User Interface
:View Student Interface Controller
:Student
:Section
:Course
:Calculate Grade Point Average
provide userLogon
logon( )
getStudent( )
display student-WebPage
return student-WebPage
return studentData
click nextButton
send studentNumber( )
getSection( )
calculateGPA( )
return sectionGrade
getCourse( )
getCredits( )
return credits
return courseData
display courseWebPage
return courseWebPage
return GPA

Relationships

- The connections between classes
  - Associations
  - Whole/part

Associative Class Between Student and Course



Associations

- The simplest type of relationship
- Association classes are those that are used to break up a many-to-many association between classes
- An object in a class may have a relationship to other objects in the same class, called a reflexive association

Whole/Part Relationships

- When one class represents the whole object, and other classes represent parts
- Categories
  - Aggregation
  - Collection
  - Composition

Aggregation

- A “has a” relationship
- Provides a means of showing that the whole object is composed of the sum of its parts

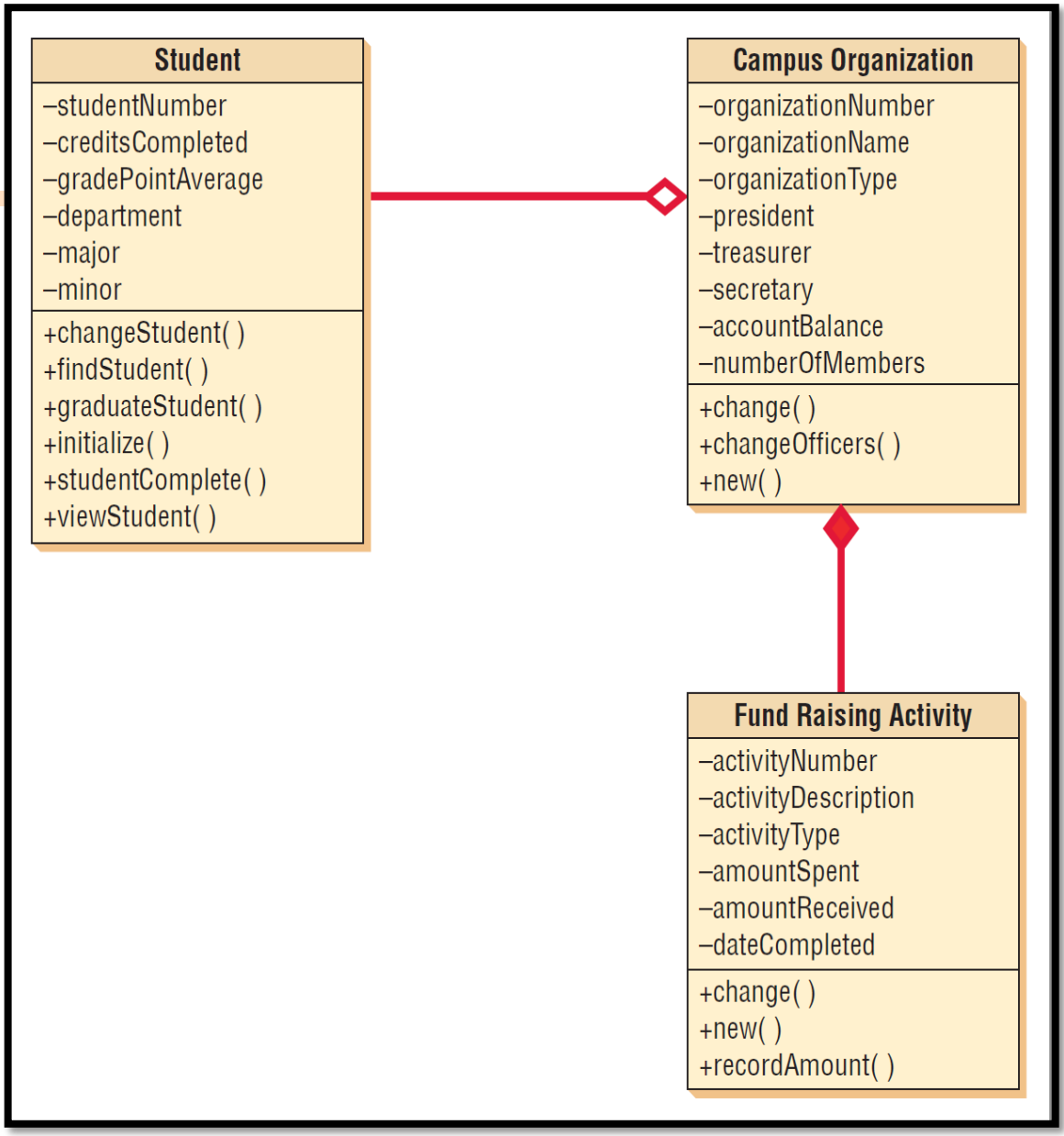
Collection

- Consists of a whole and its members
- Members may change, but the whole retains its identity
- A weak association

Composition

- The whole has a responsibility for the parts, and is a stronger relationship
- If the whole is deleted, all parts are deleted

Composition VS Aggregation



Generalization/Specialization Diagrams

- Generalization
- Inheritance
- Polymorphism
- Abstract classes
- Messages

Generalization

- Describes a relationship between a general kind of thing and a more specific kind of thing
- Described as an “is a” relationship
- Used for modeling class inheritance and specialization
- General class is a parent, base, or superclass
- Specialized class is a child, derived, or subclass

Inheritance

- Helps to foster reuse
- Helps to maintain existing program code

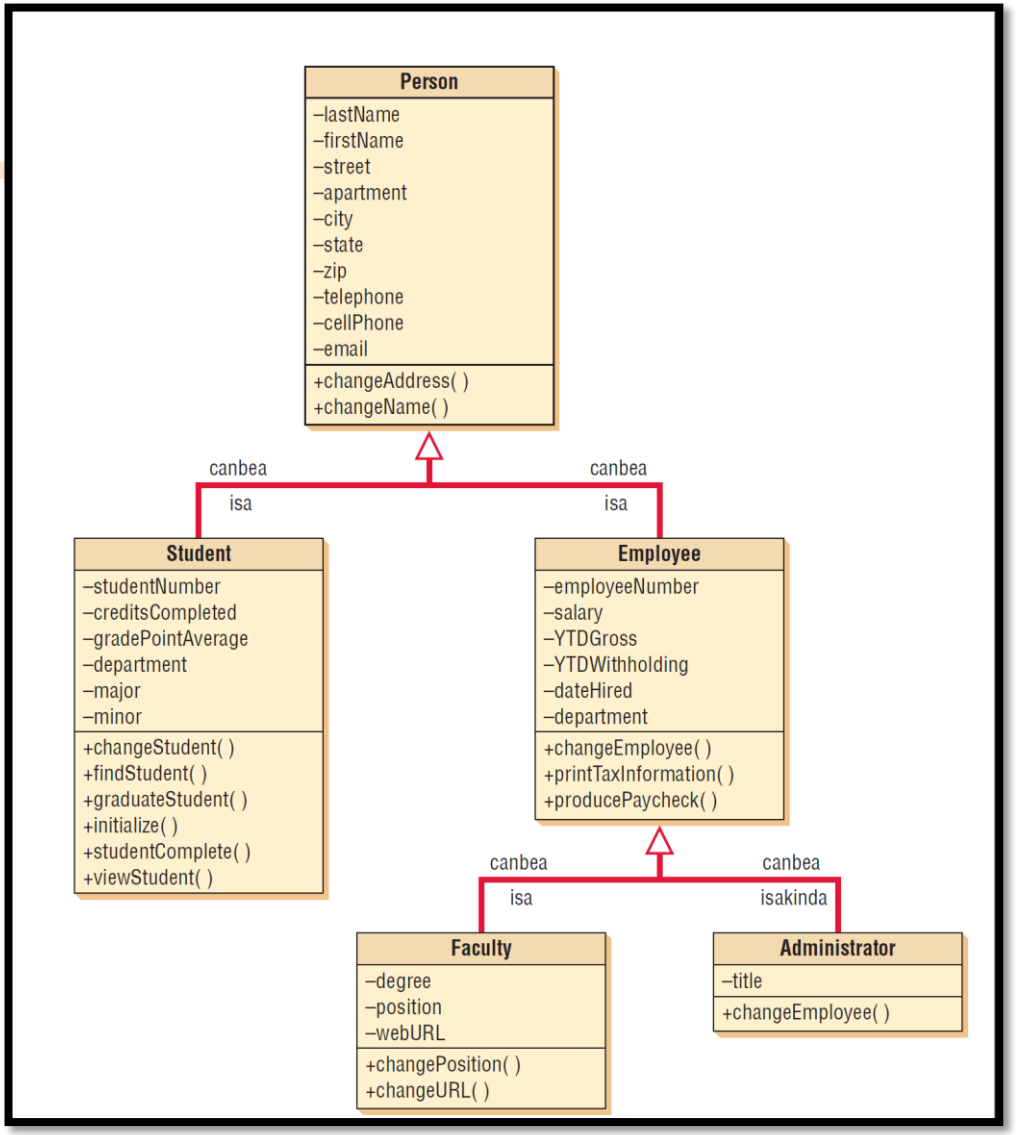
Polymorphism

- The capability of an object-oriented program to have several versions of the same method with the same name within a superclass/subclass relationship
- The subclass method overrides the superclass method
- When attributes or methods are defined more than once, the most specific one is used

Abstract Classes

- Abstract classes are general classes
- No direct objects or class instances, and is only used in conjunction with specialized classes
- Usually have attributes and may have a few methods

Gen/Spec Diagram



Finding Classes

- During interviewing or JAD sessions
- During facilitated team sessions
- During brainstorming sessions
- Analyzing documents and memos
- Examining use cases, looking for nouns

Determining Class Methods

- Standard methods
- Examine a CRUD matrix

Messages

- Used to send information by an object in one class to an object in another class
- Acts as a command, telling the receiving class to do something
- Consists of the name of the method in the receiving class, as well as the attributes that are passed with the method name
- May be thought of as an output or an input

Statechart Diagram

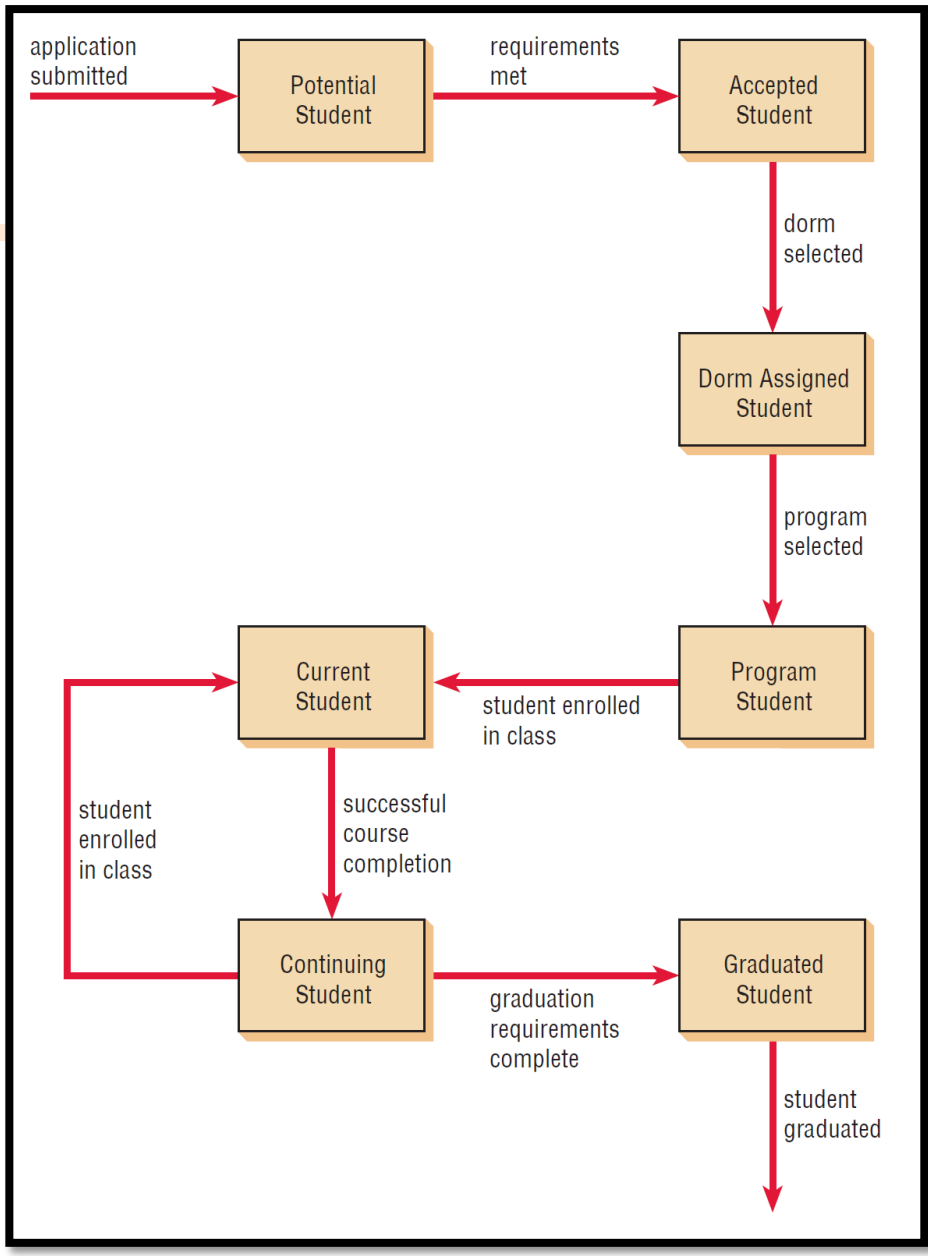
Statechart Diagrams (1)

- Used to examine the different states that an object may have
- Created for a single class
  - Objects are created, go through changes, and are deleted or removed
- Objects
- States
- Events
  - Signals or asynchronous messages
  - Synchronous
  - Temporal events

Statechart Diagrams (2)

- Created when
  - A class has a complex life cycle
  - An instance of a class may update its attributes in a number of ways through the life cycle
  - A class has an operational life cycle
  - Two classes depend on each other
  - The object's current behavior depends on what happened previously

Example of Statechart Diagrams



Putting UML to Work

The steps used in UML are:

- Define the use case model
- Continue UML diagramming to model the system during the systems analysis phase
- Develop the class diagrams
- Draw statechart diagrams
- Begin systems design by refining the UML diagrams
- Document your system design in detail

References

1. Roger S. Presmann, Software Engineering, 6th edition.
2. Kendall, System Analysis and Design, 7th edition.
3. Ian Sommerville, Software Engineering, 8th Edition
4. PPT of Roger S. Pressman (chung and zheng)
5. PPT of Kendall
6. Saiful Akbar, Handouts PPL – ITB, 2011
7. Scott W. Embler, Elements of UML Style 2.0
8. Martin Fowler, UML Distilled 3, Third Edition