2008 Fall

Software Modeling & Analysis

Part 2. Requirements

- Software Requirements
- Requirements Engineering Processes
- System Models

Lecturer: JUNBEOM YOO
jbyoo@konkuk.ac.kr

※ This lecture note is based on materials from Ian Sommerville 2006.

Chapter 6.

Software Requirements

Objectives

- To introduce concepts of user and system requirements
- To describe functional and non-functional requirements
- To explain how software requirements may be organised in a requirements document

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

- It ranges from a high-level abstract statement of service or of system constraint to detailed mathematical functional specification.

- Types of requirements
  - User requirements
    - Statements in natural language plus diagrams of the services the system provides and its operational constraints.
    - Written for customers.
  - System requirements
    - Structured document setting out detailed descriptions of the system's functions, services and operational constraints.
    - Defines what should be implemented
    - May be part of a contract between client and contractor.

Requirements Definitions and Specifications

User requirement definition

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

System 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.

Functional vs. Non-Functional Requirements

- Functional requirements
  - Statements of services the system should provide.
  - How the system should react to particular inputs.
  - How the system should behave in particular situations.

- Non-functional requirements
  - Constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards, etc.

- Domain requirements
  - Requirements that come from the application domain of the system and that reflect characteristics of that domain.

Example: LIBSYS System

- A library system that provides a single interface to a number of database s of articles in different libraries.
- Users can search for, download, and print these articles for personal study.

- Function requirements:
  - 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.

Requirements Completeness and Consistency

- Problems arise when requirements are not precisely stated.
- Ambiguous requirements may be interpreted in different ways by developers and users.

- 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.

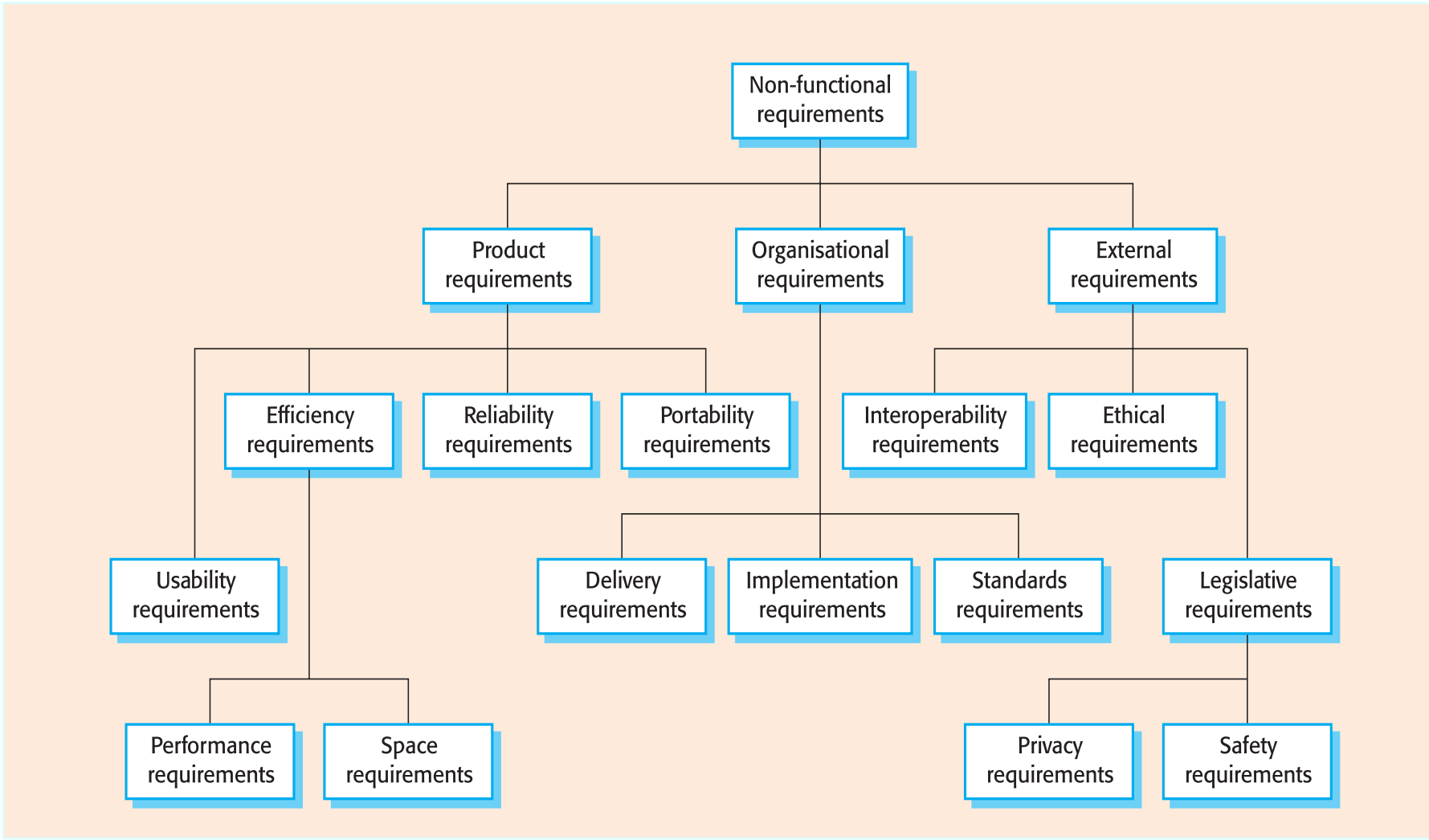
- In practice, it is impossible to produce a complete and consistent requirements document.

Non-Functional Requirements

- Define system properties and constraints e.g. reliability, response time and storage requirements, constraints on I/O device capability, system representations, etc.
- Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless.

- Classification of non-functional requirements
  - Product requirements
    - 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 Requirement Types



Non-Functional Requirements Examples

- Product requirement
  - 8.1 The user interface for LIBSYS shall be implemented as simple HTML without frames or Java applets.
- 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.
  - "The system should be easy to use by experienced controllers and should be organised in such a way that user errors are minimised."

- Verifiable non-functional requirement
  - A statement using some measure that can be tested objectively.
  - "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."

Domain Requirements

- Derived from the application domain
- Describe system characteristics and features that reflect the domain.
- Domain requirements may be
  - new functional requirements
  - constraints on existing requirements
  - define specific computations

- If domain requirements are not satisfied, the system may be unworkable.

Domain Requirements Example : LIBSYS

- 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.

Problems with Natural Language Specification

- Ambiguity
  - Readers and writers of the requirement must interpret the same words in the same way. NL 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 modularisation
  - NL structures are inadequate to structure system requirements.

- Alternatives to NL specifications
  - Structural language specification
  - Design description language
  - Graphical notations
  - Mathematical specifications

Structured Language Specifications

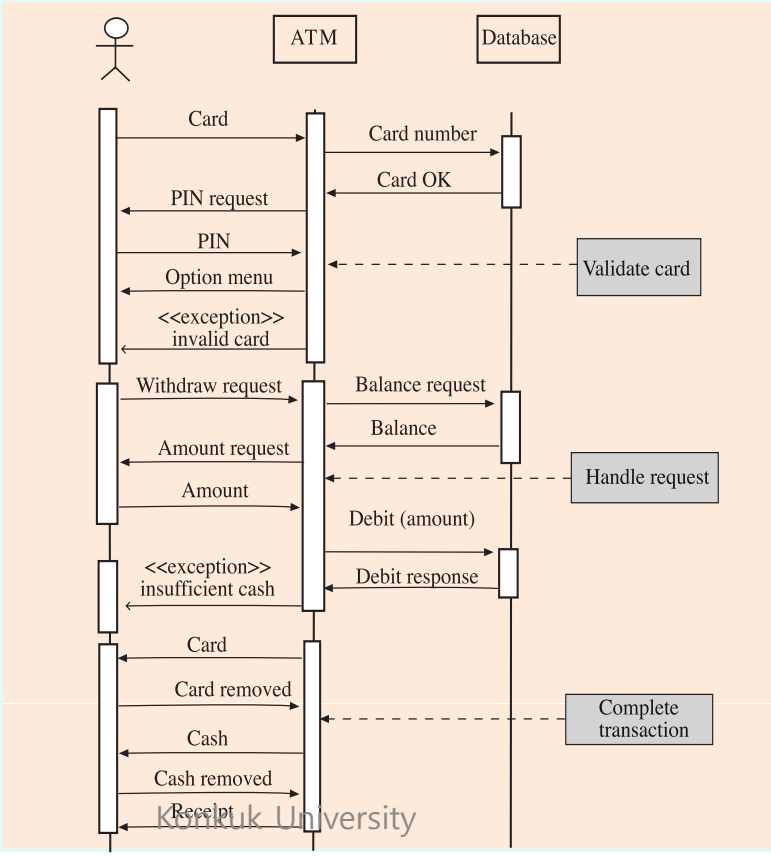
- The freedom of the requirements writer is limited by a predefined templa te for requirements.
- Form-based specifications

Insulin Pump/Control Software/SRS/3.3.2	
Function	Compute insulin dose: Safe sugar level
Description	Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.
Inputs	Current sugar reading (r2), the previous two readings (r0 and r1)
Source	Current sugar reading from sensor. Other readings from memory.
Outputs	CompDose Š the dose in insulin to be delivered
Destination	Main control loop
Action:	CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.
Requires	Two previous readings so that the rate of change of sugar level can be computed.
Pre-condition	The insulin reservoir contains at least the maximum allowed single dose of insulin..
Post-condition	r0 is replaced by r1 then r1 is replaced by r2
Side-effects	None

Graphical Notations

- Graphical models are most useful when you need to show how state changes or where you need to describe a sequence of actions.
- Different graphical models are explained in Chapter 8.

- Sequence diagram : (ATM example)



Interface Specification

- Most systems must operate with other systems and the operating interfaces must be specified as part of the requirements.
- Three types of interface may have to be defined
  - Procedural interfaces
  - Data structures that are exchanged
  - Data representations
- Formal notations are an effective technique for interface specification.

```
interface PrintServer {

// defines an abstract printer server
// requires:          interface Printer, interface PrintDoc
// provides: initialize, print, displayPrintQueue, cancelPrintJob, switchPrinter

        void initialize ( Printer p ) ;
        void print ( Printer p, PrintDoc d ) ;
        void displayPrintQueue ( Printer p ) ;
        void cancelPrintJob (Printer p, PrintDoc d) ;
        void switchPrinter (Printer p1, Printer p2, PrintDoc d) ;
} //PrintServer
```

Requirements Document

- Requirements document is an official statement of what is required of the system developers. Should include both a definition of user requirements and a specification of the system requirements.
- It is NOT a design document. As far as possible, it should be a set of WHAT the system should do rather than HOW it should do it.

- IEEE standard on requirements document
  - Introduction
  - General description
  - Specific requirements
  - Appendices
  - Index

- Preface
- Introduction
- Glossary
- User requirements definition
- System architecture
- System requirements specification
- System models
- System evolution
- Appendices
- Index

Summary

- Requirements set out what the system should do and define constraints on its operation and implementation.
- Functional requirements set out services the system should provide.
- Non-functional requirements constrain the system being developed or the development process.
- User requirements are high-level statements of what the system should do. User requirements should be written using natural language, tables and diagrams.
- System requirements are intended to communicate the functions that the system should provide.
- A software requirements document is an agreed statement of the system requirements.
- The IEEE standard is a useful starting point for defining more detailed specific requirements standards.

Chapter 7.

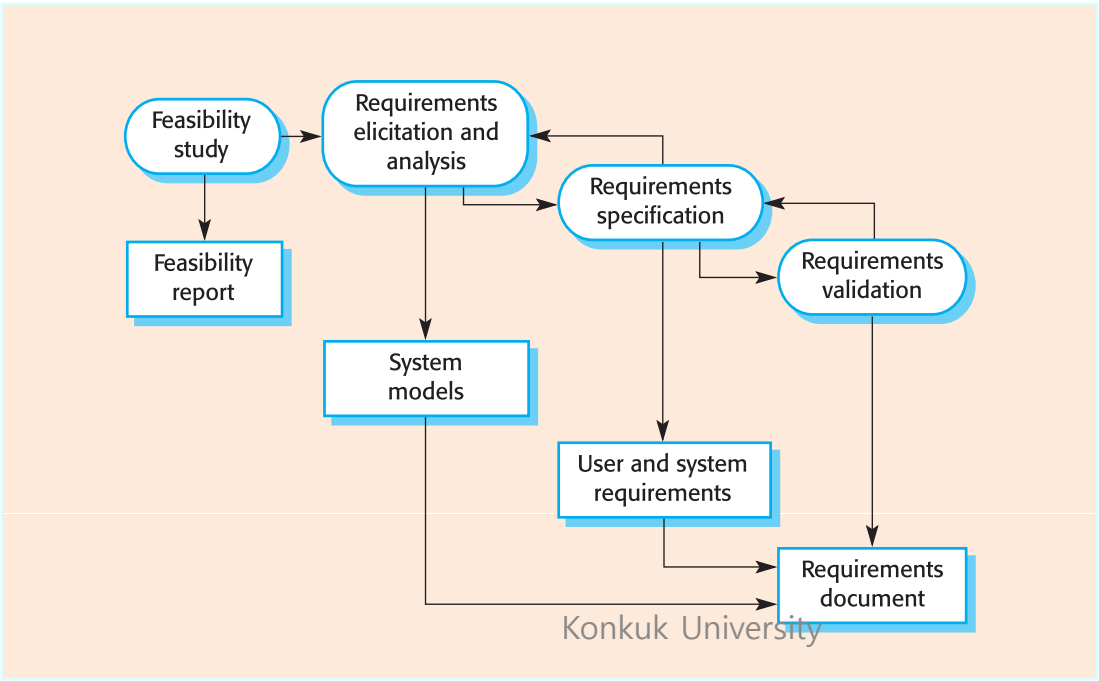
Requirements Engineering Processes

Objectives

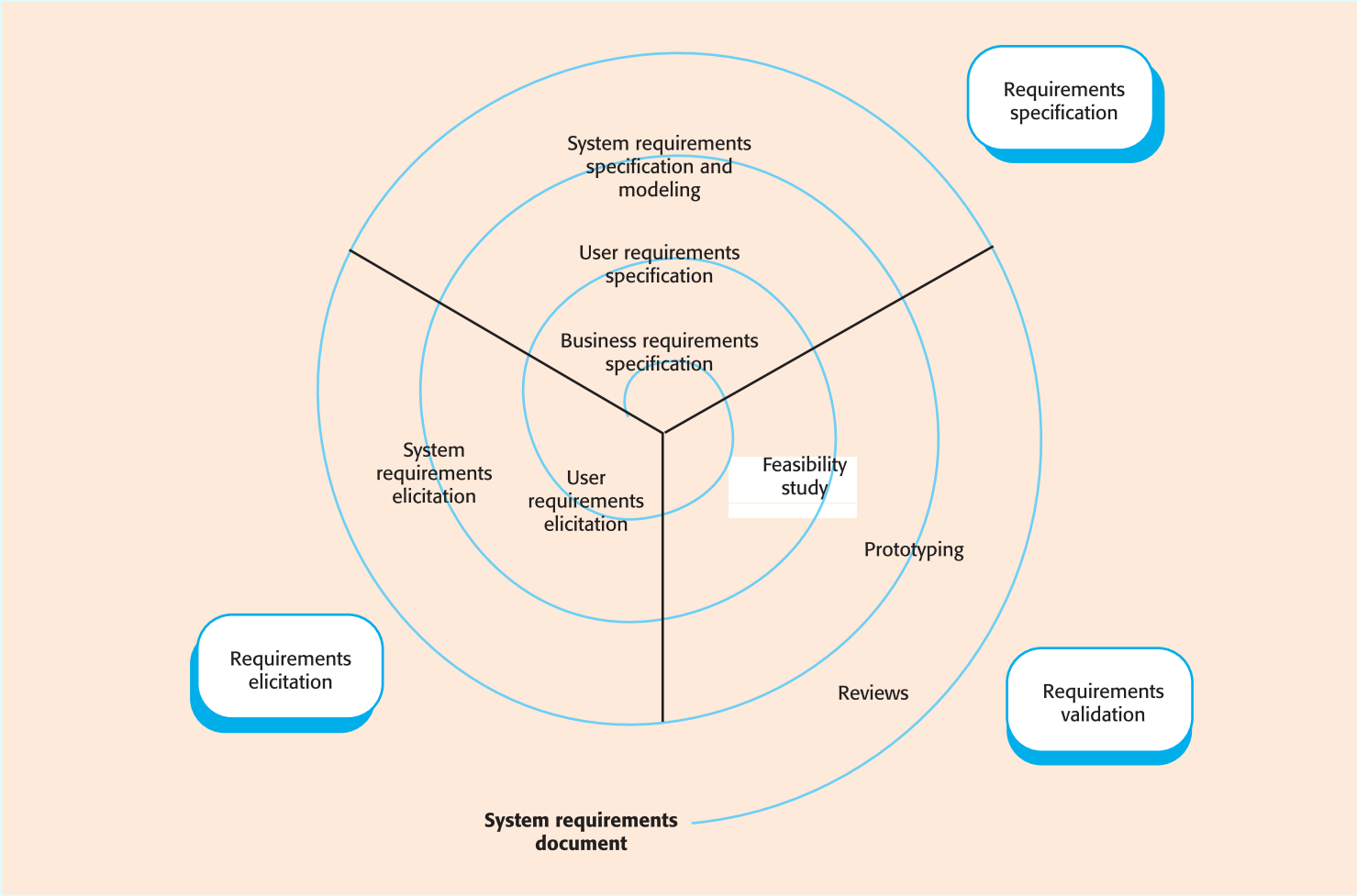
- To describe principal requirements engineering activities and their relationships
- To introduce techniques for requirements elicitation and analysis
- To describe requirements validation and the role of requirements reviews
- To discuss the role of requirements management

Requirements Engineering Processes

- The processes used for RE vary widely depending on application domain, people involved and organisation developing the requirements.
- However, there are a number of generic activities common to all processes
  - Feasibility Study - Requirements Elicitation and Analysis
  - Requirements Validation - Requirements Management



Requirements Engineering Processes



1. Feasibility Study

- 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

- Questions for feasibility:
  - 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?

2. Requirements Elicitation and Analysis

- 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.

Activities in Requirements Elicitation and Analysis

- 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 sy stems, and distilling the user and system requirements from this informat ion.
- Sources of information include
  - documentation
  - system stakeholders
  - specifications of similar systems

Interviewing

- In formal or informal interviewing, the RE team puts questions to stakeho lders about the system that they use and the system to be developed.
- There are two types of interview
  - Closed interviews : pre-defined set of questions are answered.
  - Open interviews : no pre-defined agenda and a range of issues are explored with stakeholders.

- Normally a mix of closed and open-ended interviewing

Scenarios

- Scenarios are real-life examples of how a system can be used.
- They should include
  - A description of the starting situation
  - A description of the normal flow of events
  - A description of what can go wrong
  - Information about other concurrent activities
  - A description of the state when the scenario finishes

- Example: LIBSYS Scenario

Initial assumption: The user has logged on to the LIBSYS system and has located the journal containing the copy of the article.

Normal: The user selects the article to be copied. He or she is then prompted by the system to either provide subscriber information for the journal or to indicate how they will pay for the article. Alternative payment methods are by credit card or by quoting an organisational account number.

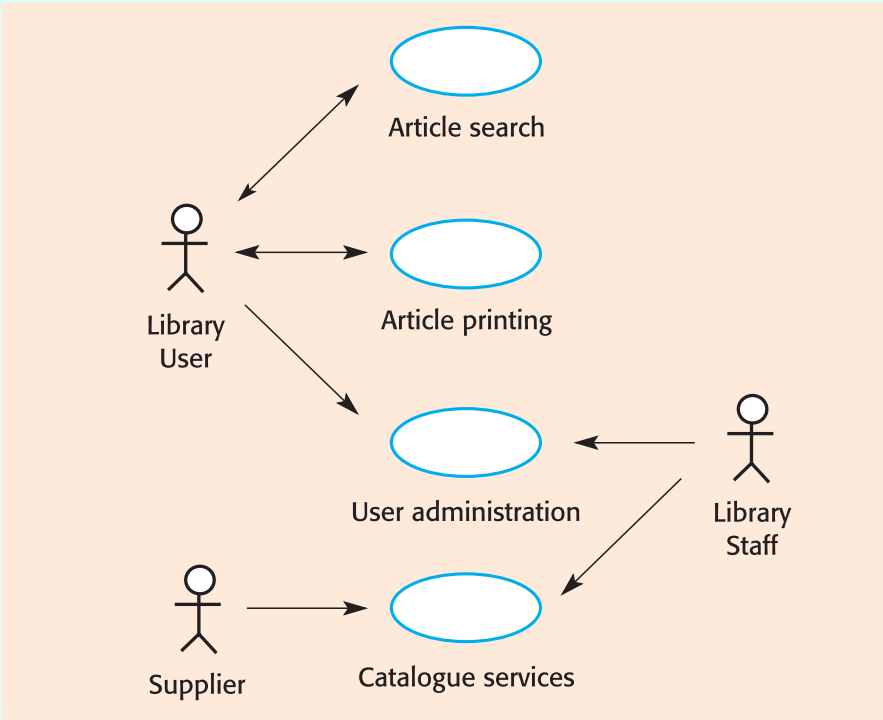
The user is then asked to fill in a copyright form that maintains details of the transaction and they then submit this to the LIBSYS system.

The copyright form is checked and, if OK, the PDF version of the article is downloaded to the LIBSYS working area on the userÕs computer and the user is informed that it is available. The user is asked to select a printer and a copy of the article is printed. If the article has been flagged as Õprint-onlyÕ it is deleted from the userÕs system once the user has confirmed that printing is complete.

Use Cases

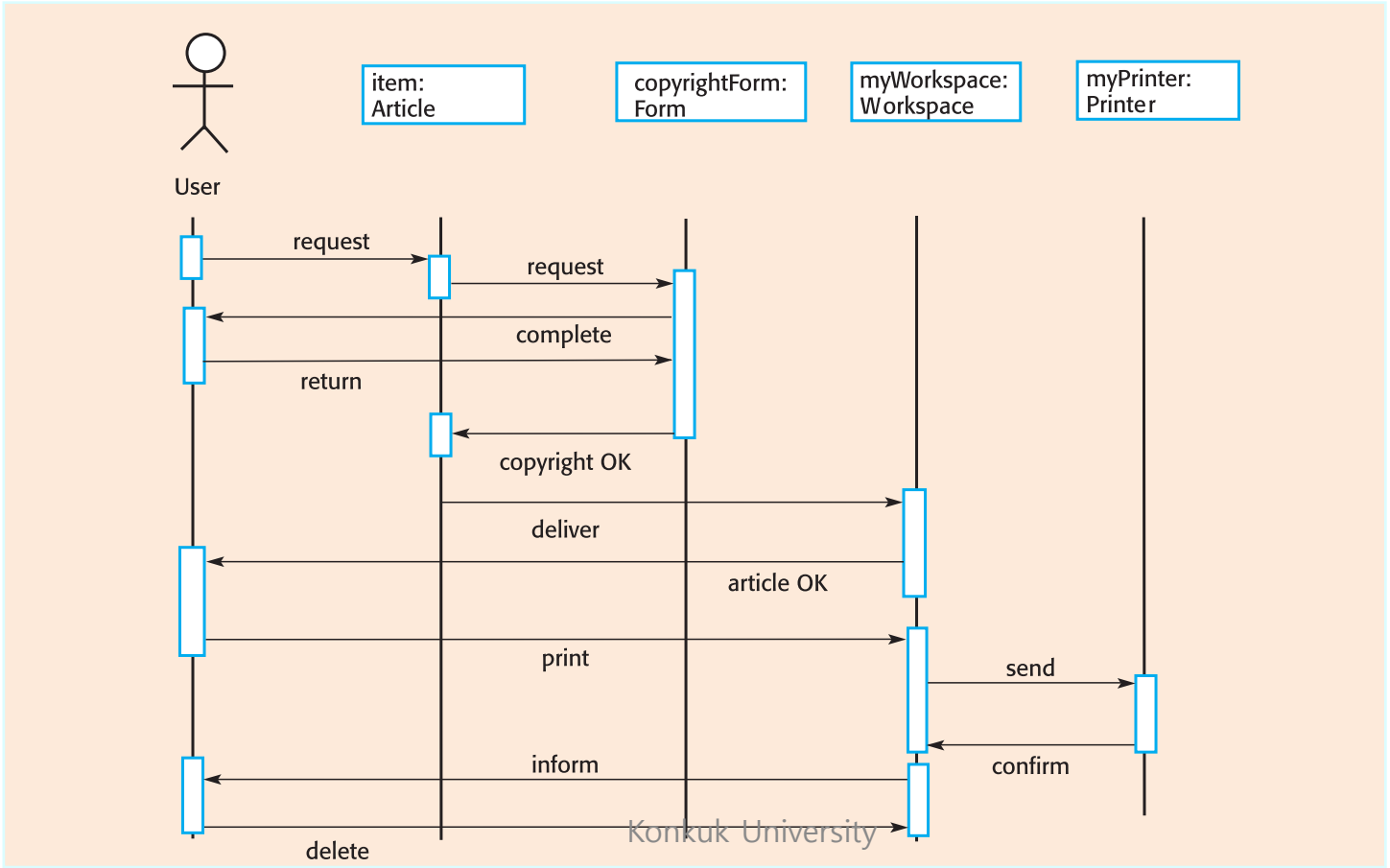
- Use-cases are a scenario based technique in the UML
- Identify actors in an interaction and describe the interaction itself.
- Use cases should describe all possible interactions with the system.

LIBSYS use cases



Use Cases with Sequence Diagram

- Sequence diagrams may be used to add detail to use-cases by showing the sequence of event processing in the system.



3. Requirements Validation

- Concerned with demonstrating that the requirements defined the system that the customer really wants.
- Requirements error costs are high, so validation is very important
  - Fixing a requirements error after delivery may cost up to 100 times the cost of fixing an implementation error.

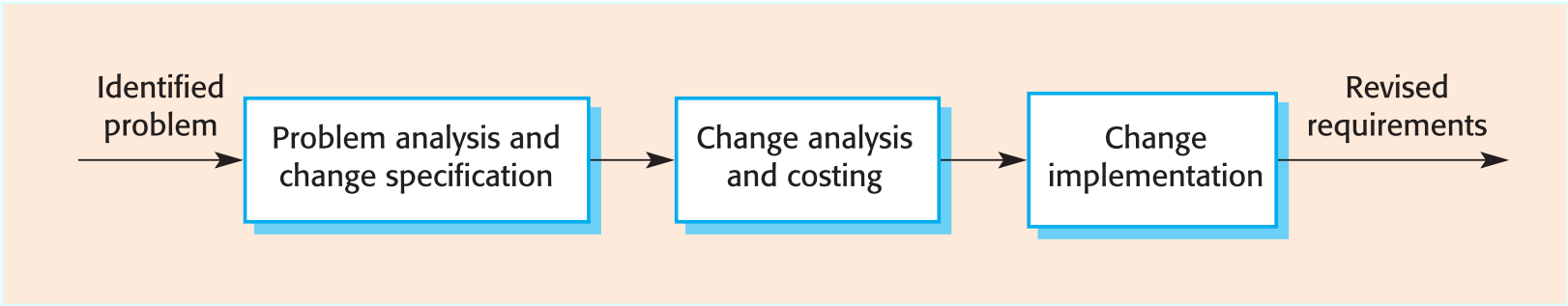
- Requirements Checking:
  - Validity : Does the system provide the functions which best support the customer's needs?
  - Consistency : Are there any requirements conflicts?
  - Completeness : Are all functions required by the customer included?
  - Realism : Can the requirements be implemented given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

Requirements Validation Techniques

- Requirements reviews
  - Systematic manual analysis of the requirements
  - Review focus:
    - Verifiability (Testability)
    - Comprehensibility
    - Traceability
    - Adaptability
- Prototyping
  - Using an executable model of the system to check requirements.
- Test-case generation
  - Developing tests for requirements to check testability

4. Requirements Management

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
- Requirements are inevitably incomplete and inconsistent
  - New requirements emerge during the process as business needs change and a better understanding of the system is developed
  - Different viewpoints have different requirements and these are often contradictory.



Requirements Management Planning

- During the requirements engineering process, we have to plan:
  - Requirements identification
    - How requirements are individually identified
  - Change management process
    - The process followed when analysing 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
    - Requirements storage
    - Change management
    - Traceability management

Traceability

- Traceability is concerned with the relationships between requirements, their sources and the system design
- Source traceability
  - Links from requirements to stakeholders who proposed these requirements
- Requirements traceability
  - Links between dependent requirements
- Design traceability
  - Links from the requirements to the design

Traceability Matrix

Req. id	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2
1.1		D	R					
1.2			D			D		D
1.3	R			R				
2.1			R		D			D
2.2								D
2.3		R		D				
3.1								R
3.2							R	

Summary

- The requirements engineering process includes a feasibility study, requirements elicitation and analysis, requirements specification and requirements management.

- Requirements elicitation and analysis is iterative involving domain understanding, requirements collection, classification, structuring, prioritisation and validation.

- Systems have multiple stakeholders with different requirements.
- Social and organisation factors influence system requirements.
- Requirements validation is concerned with checks for validity, consistency, completeness, realism and verifiability.

- Business changes inevitably lead to changing requirements.
- Requirements management includes planning and change management.

Chapter 8.

System Models

Objectives

- To explain why the context of a system should be modelled as part of the RE process
- To describe behavioural modelling, data modelling and object modelling
- To show how CASE workbenches support system modelling

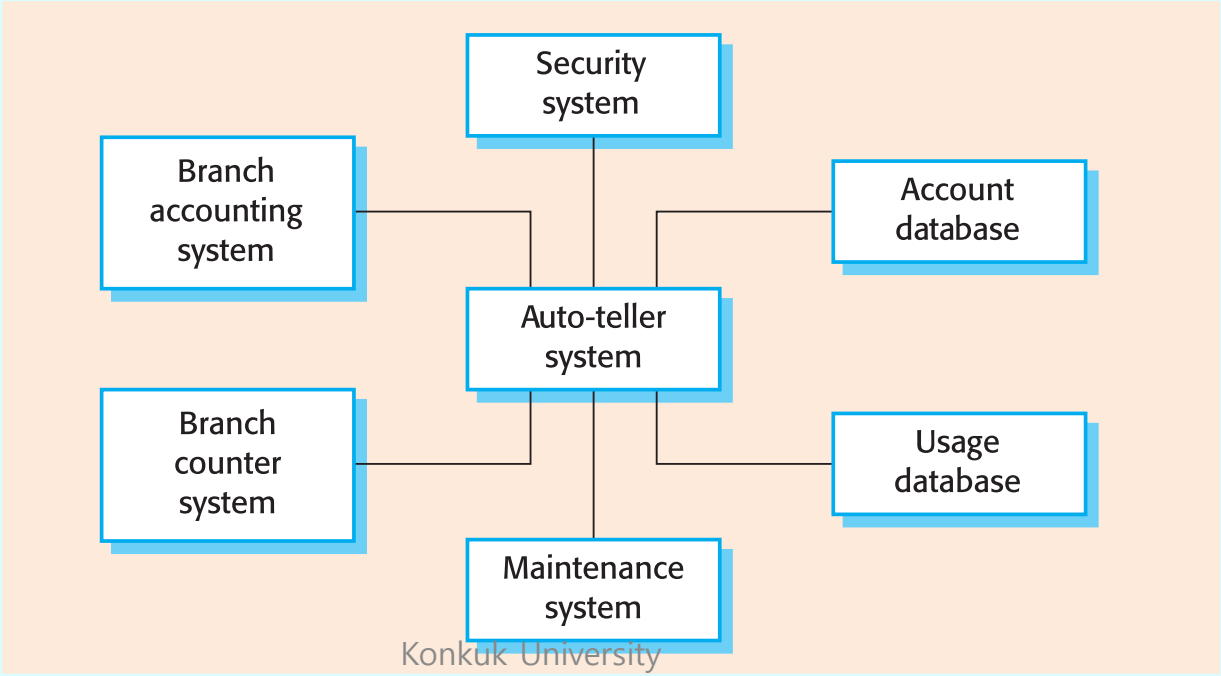
System Modelling

- System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.
- Different models present the system from different perspectives
  - External perspective : showing the system's context or environment
  - Behavioural perspective : showing the behaviour of the system
  - Structural perspective : showing the system or data architecture

- Model types
  - Data processing model: showing how the data is processed at different stages
  - Composition model: showing how entities are composed of other entities
  - Architectural model: showing principal sub-systems
  - Classification model: showing how entities have common characteristics
  - Stimulus/response model: showing the system's reaction to events
  - Many ones

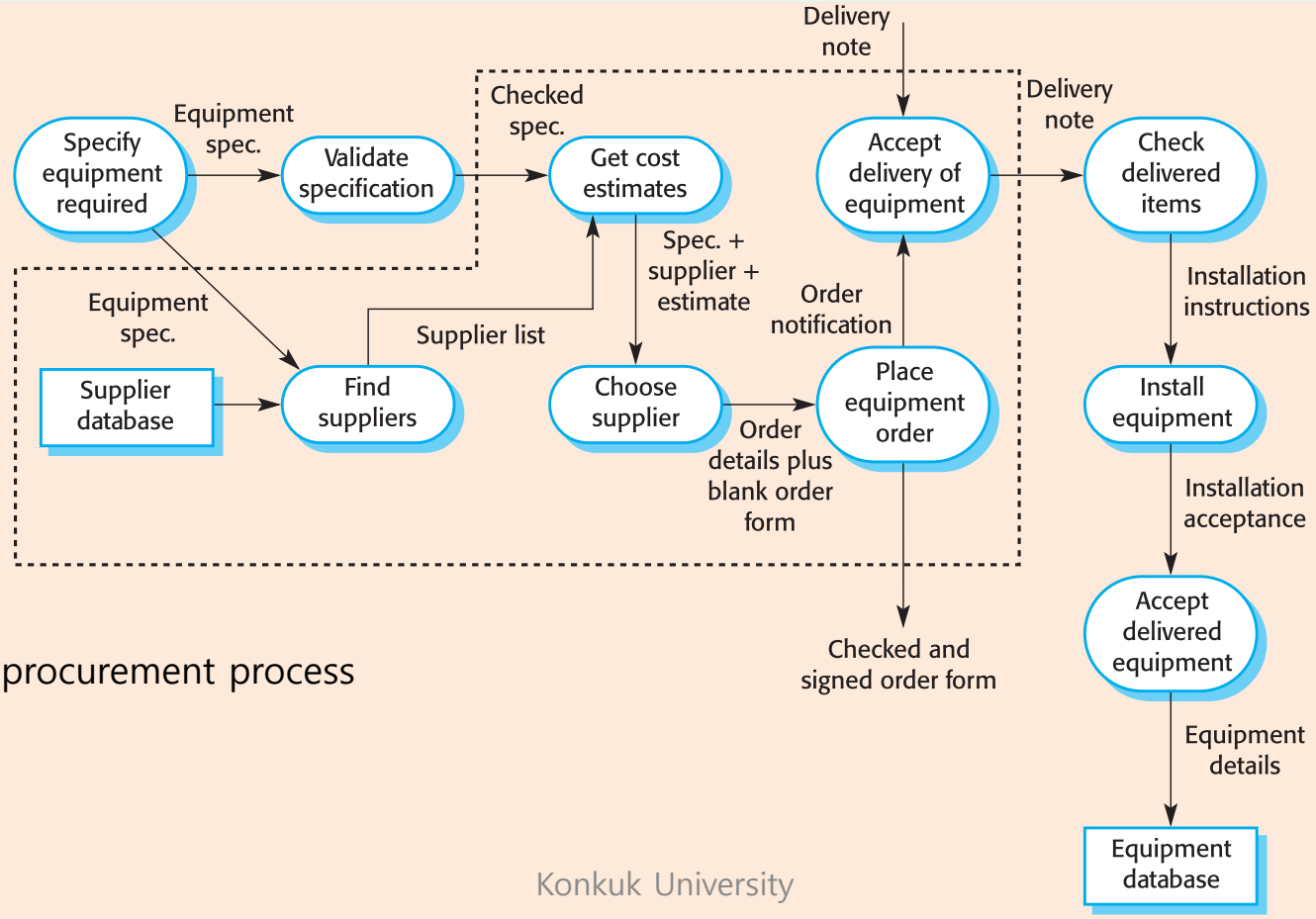
System Context Model

- System Context (models) are used to illustrate the operational context of a system - they show what lies outside the system boundaries.
- Social and organisational concerns may affect the decision on where to position system boundaries.
- Architectural models show the system and its relationship with other systems.



Process Model

- Process models show the overall process and the processes that are supported by the system.



Equipment procurement process

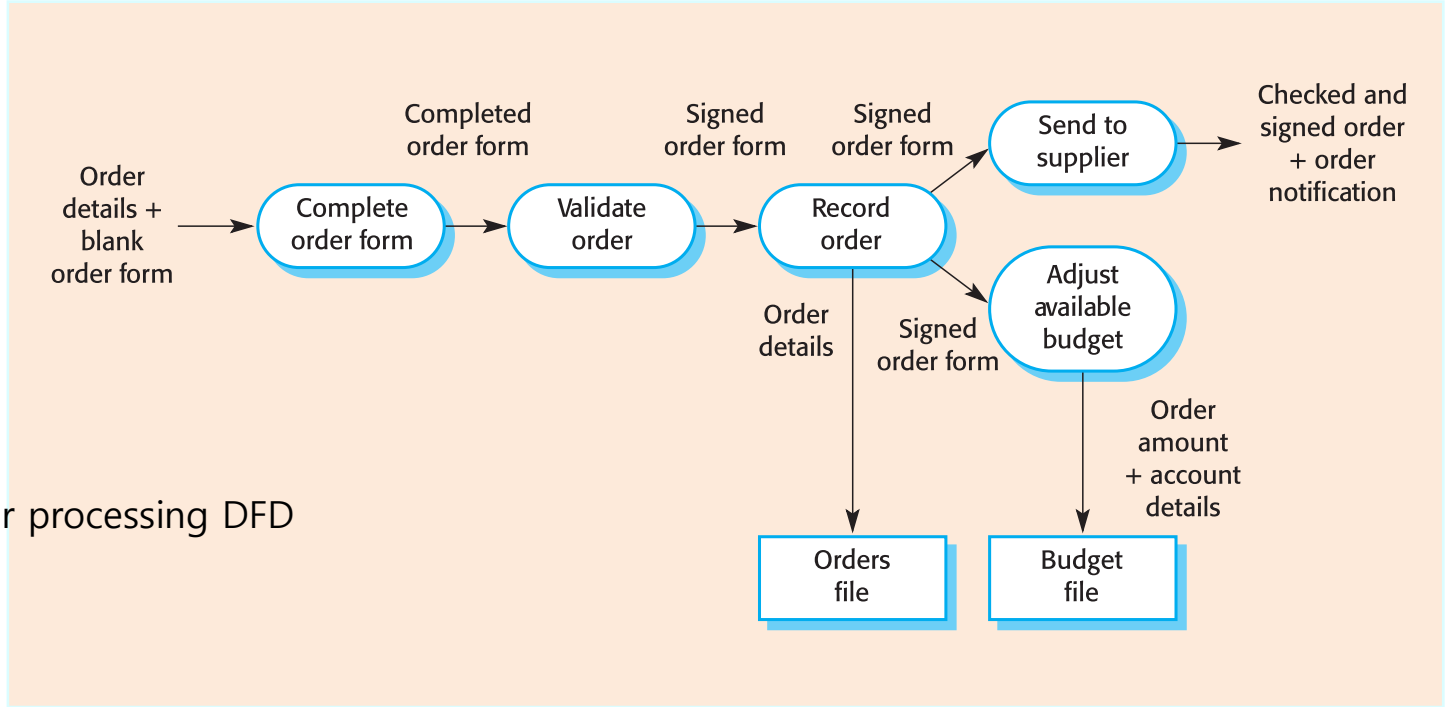
Behavioural Model

- Behavioural models are used to describe the overall behaviour of a system.
- Two types:
  - Data processing models : show how data is processed as it moves through the system
  - State machine models : show the systems response to events

- These models show different perspectives so both of them are required to describe the system's behaviour.

Data-Processing Model

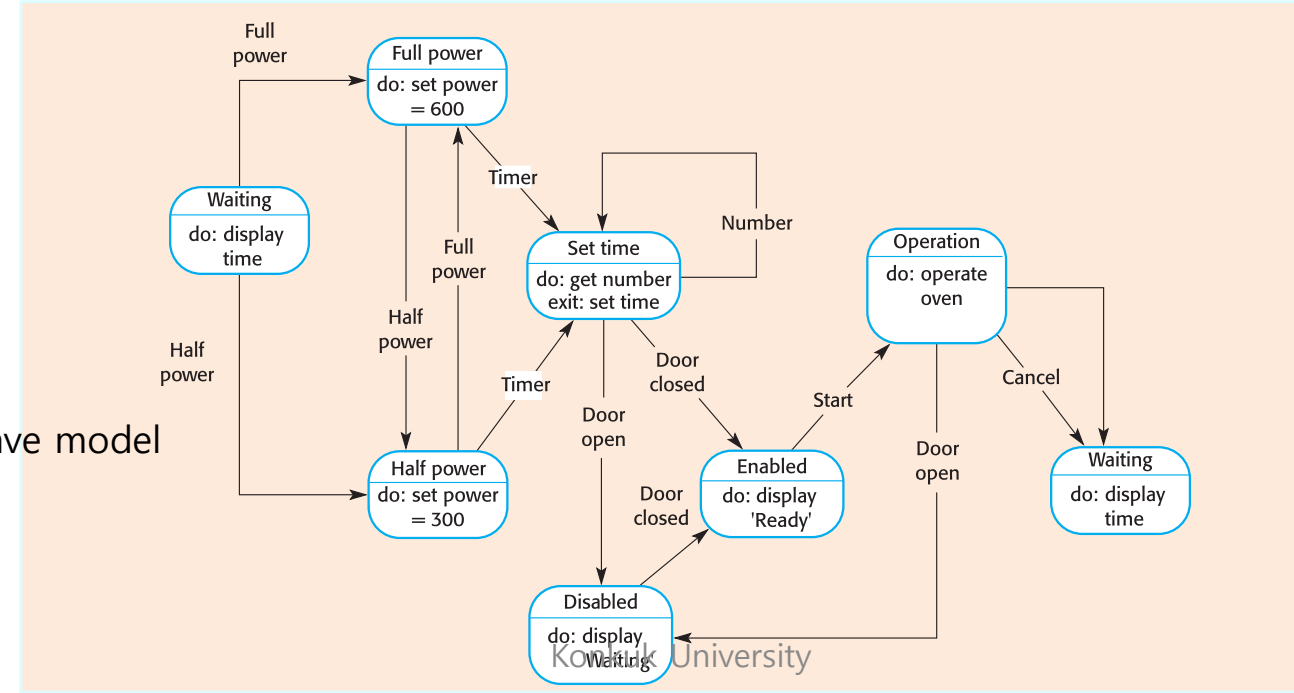
- Data flow diagrams (DFDs) is used to model the system's data processing.
- These show the processing steps as data flows through a system.
- Simple and intuitive notation that customers can understand.
- Show end-to-end processing of data.



Order processing DFD

State Machine Model

- Models the behaviour of the system in response to external and internal events. It show the system's responses to stimuli.
- Often used for modelling real-time systems.
- State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves from one state to another.

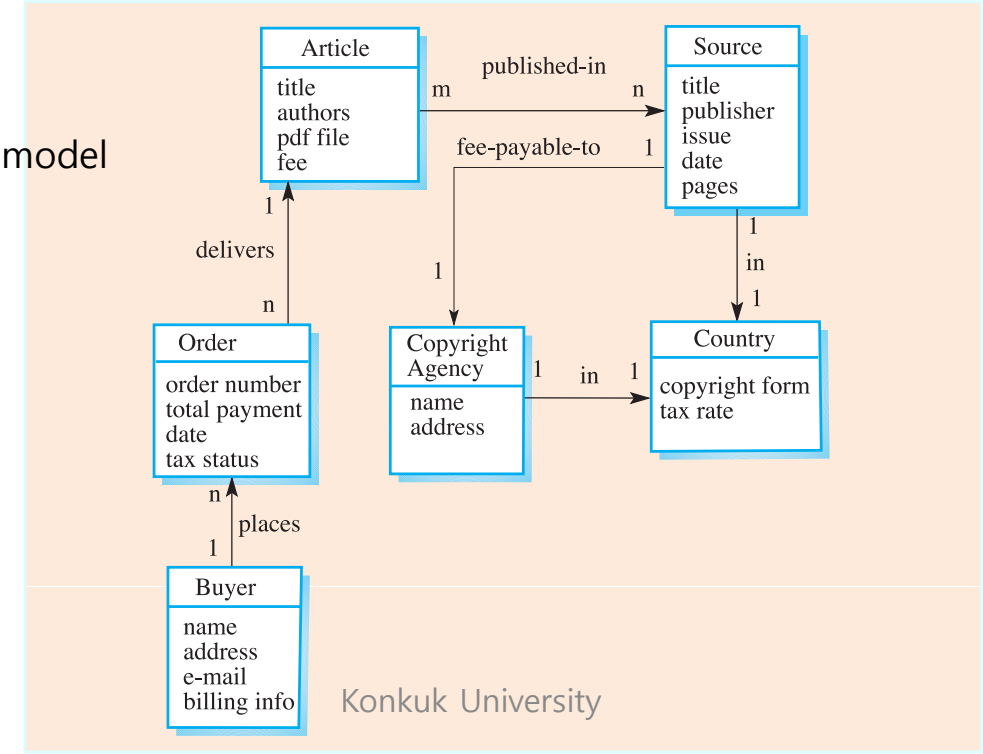


Microwave model

Semantic Data Model

- Used to describe the logical structure of data processed by the system.
- An entity-relation-attribute model sets out the entities in the system, relationships between these entities, and the entity attributes
- Widely used in database design. Can readily be implemented using relational databases.

Library semantic model



Object Model

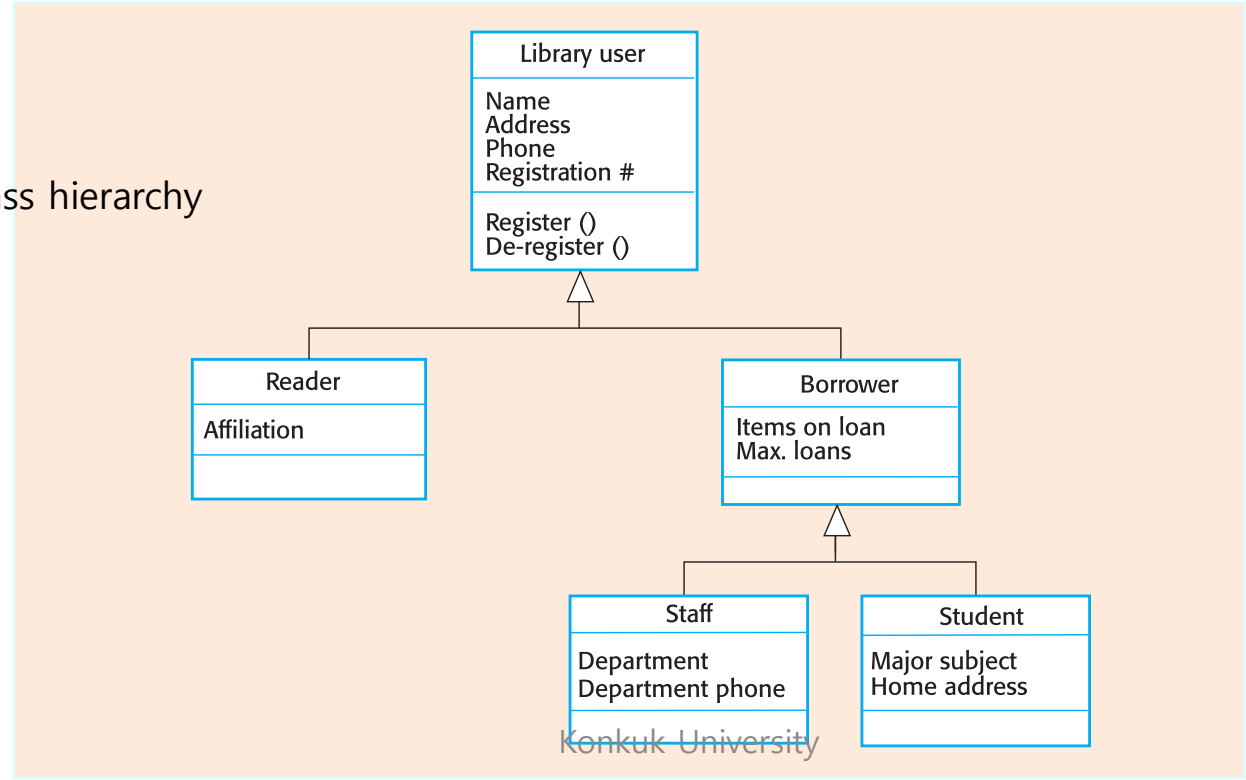
- Object models describe the system in terms of object classes and their associations.
- An object class is an abstraction over a set of objects with common attributes and the services (operations) provided by each object.
- Object classes reflecting domain entities are reusable across systems

- Various object models may be produced
  - Inheritance models
  - Aggregation models
  - Interaction models

Inheritance Model

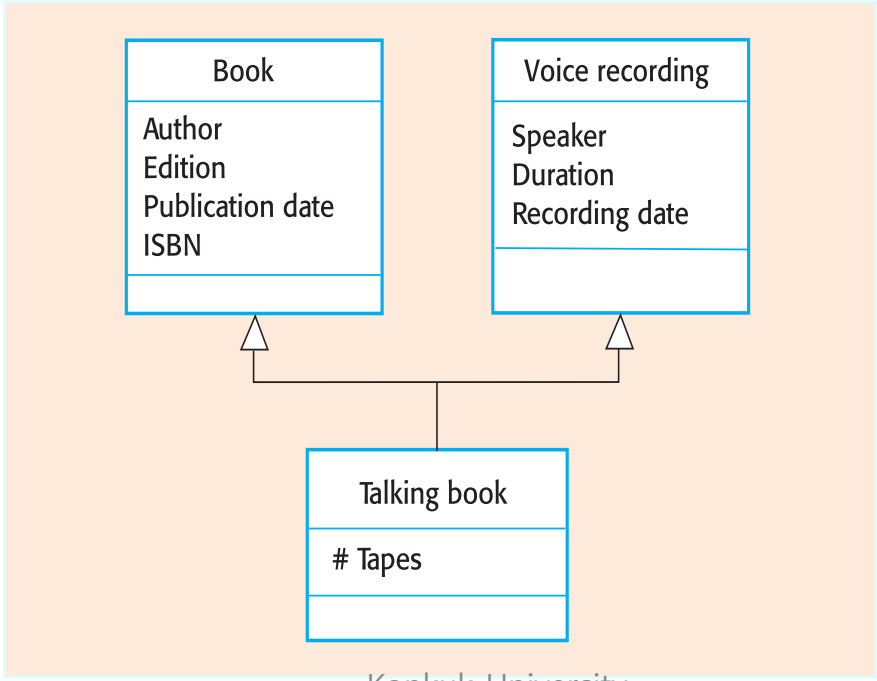
- Organizes domain object classes into a hierarchy.
- Classes at the top of the hierarchy reflect common features of all classes.
- Object classes inherit their attributes and services from one or more super-classes. These may then be specialised as necessary.

User class hierarchy



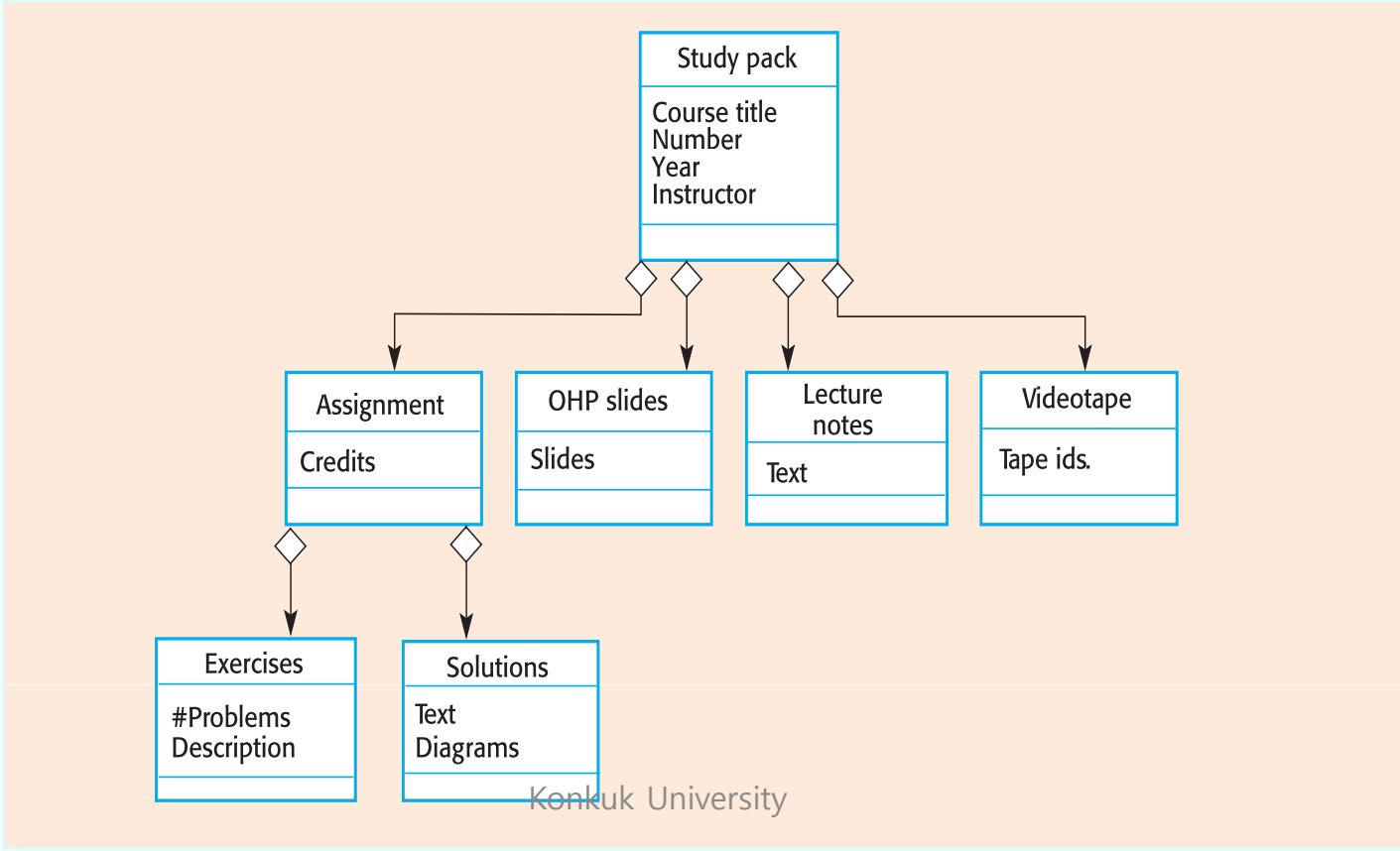
Multiple Inheritance

- Multiple inheritance allows object classes to inherit from several super-classes.
- This can lead to semantic conflicts where attributes/services with the same name in different super-classes have different semantics.
- Multiple inheritance makes class hierarchy reorganisation more complex.



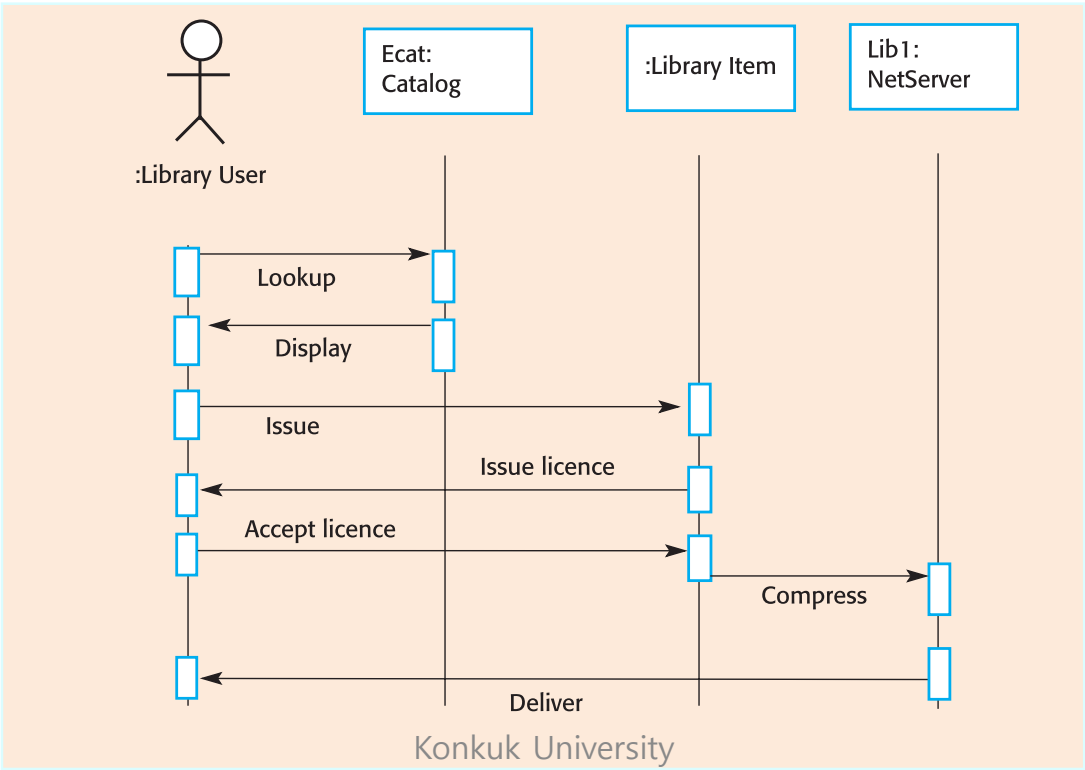
Object Aggregation Model

- An aggregation model shows how classes are composed of other classes.
- Aggregation models are similar to the part-of relationship in semantic data models.



Object Behaviour Model

- A behavioural model shows the interactions between objects to produce some particular system behaviour specified as a use-case.
- Sequence diagrams (or collaboration diagrams) in the UML

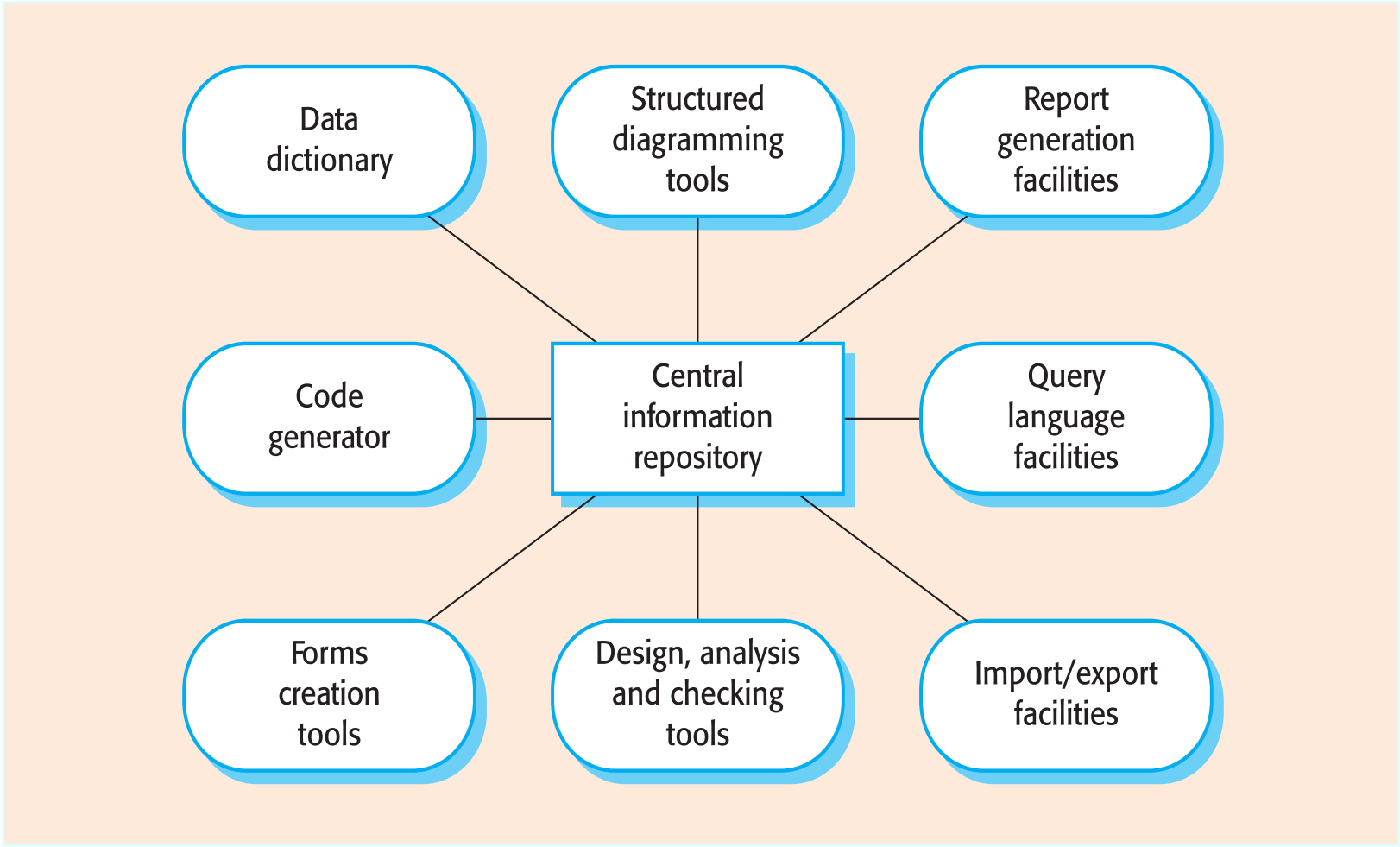


Structured Method

- Structured methods incorporate system modelling as an inherent part of the method.
- Methods define
  - a set of models,
  - a process for deriving these models, and
  - rules and guidelines that should apply to the models.

- CASE tools support system modelling as part of a structured method.
- CASE Workbench:
  - A coherent set of tools that is designed to support related software process activities such as analysis, design or testing.
  - Analysis and design workbenches support system modelling during both requirements engineering and system design.
  - May support a specific design method.
  - May support to create several different types of system model.

Analysis and Design Workbench: Example



Summary

- A model is an abstract system view. Complementary types of model provide different system information.
- Context models show the position of a system in its environment with other systems and processes.
- Data flow models is used to model the data processing in a system.
- State machine models model the system's behaviour in response to internal or external events.
- Semantic data models describe the logical structure of data which is imported to or exported by the systems.
- Object models describe logical system entities, their classification and aggregation.
- Sequence models show the interactions between actors and the system objects that they use.
- Structured methods provide a framework for developing system models.