Lecture 3

Software Requirements

Hoang Huu Hanh
hoanghuuhanh@ptit·edu·vn

– Software Requirements –
Descriptions and specifications of a system

Objectives:

- To introduce the concepts of **user and system requirements**

- To describe **functional** / **non-functional requirements**

- To explain **two techniques** for describing system requirements

- To explain **how software requirements may be organised** in a requirements document
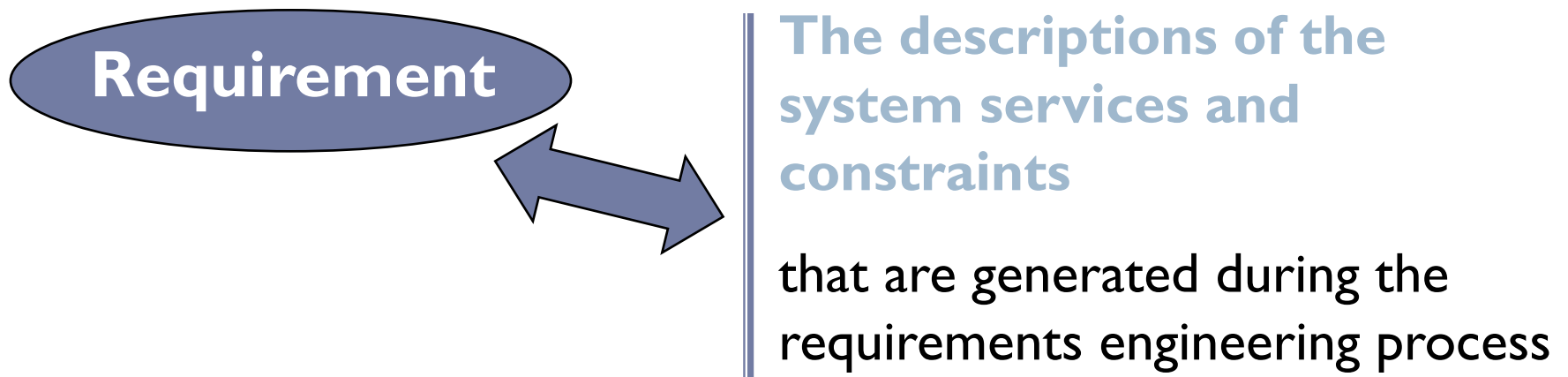
Topics covered

– **Functional and non-functional requirements**

  – **User requirements**

    – **System requirements**

      – **The software requirements document**

Software Requirements

Requirements engineering

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

Requirement

⟷

The descriptions of the system services and constraints

that are generated during the requirements 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

Software Requirements

Types of requirement

- **User requirements**
  - Statements in natural language plus diagrams of the services the system provides and its operational constraints. Written for customers
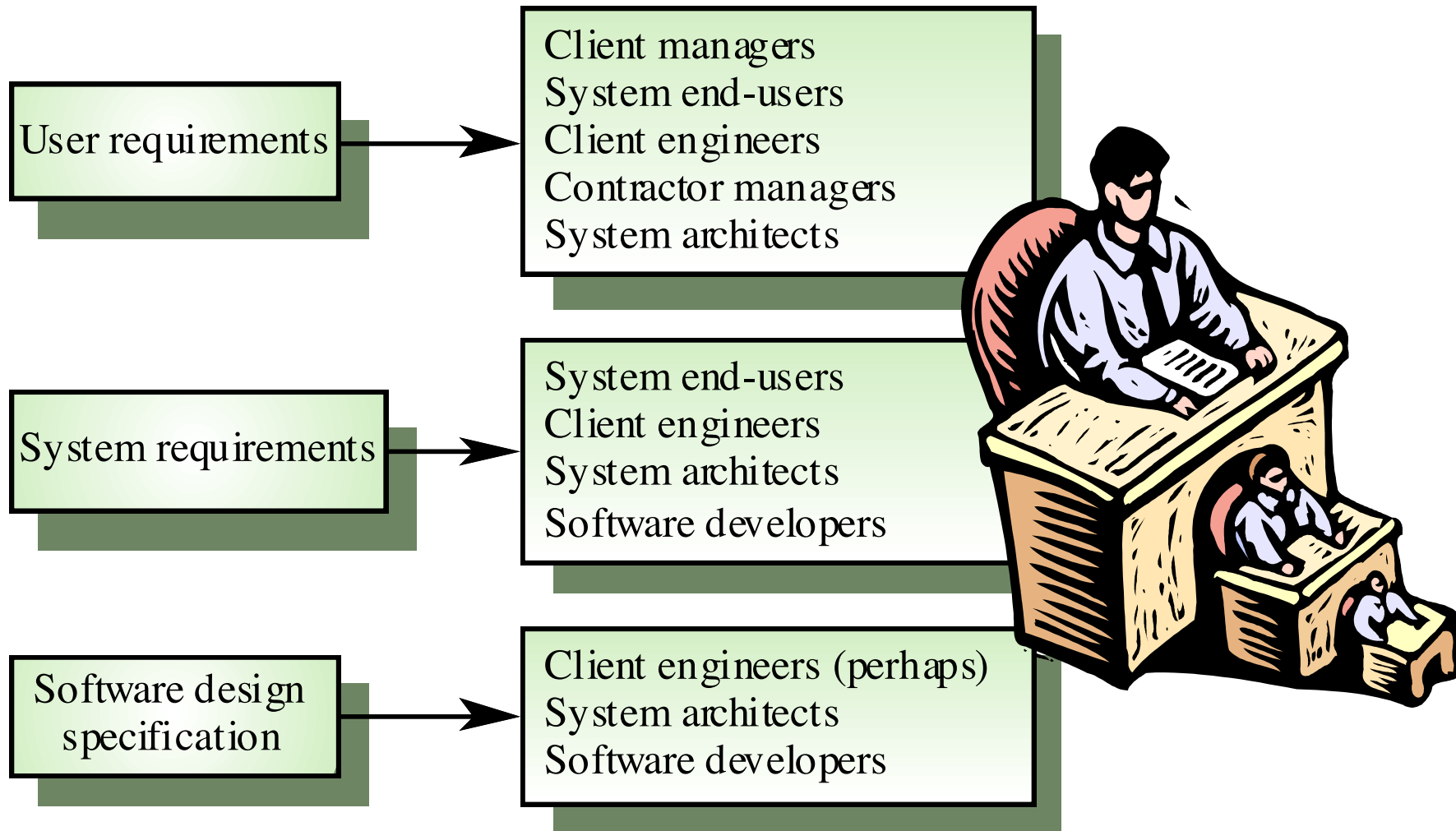
- **System requirements**
  - A structured document setting out detailed descriptions of the system services. Written as a contract between client and contractor

- **Software specification (requirements)**
  - A detailed software description which can serve as a basis for a design or implementation. Written for developers

Requirements readers

User requirements → Client managers
System end-users
Client engineers
Contractor managers
System architects

System requirements → System end-users
Client engineers
System architects
Software developers

Software design specification → Client engineers (perhaps)
System architects
Software developers

Functional and non-functional requirements

- **Functional requirements**
  - Statements of services the system should provide, how the system should react to particular inputs and 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

Functional Requirements

Describe functionality or system services

▸ **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 <u>system services in detail</u>

Examples of functional 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 imprecision

▸ 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

Requirements completeness and consistency

▸ **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

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.

▸ **Process requirements** may also be specified mandating a particular CASE system, programming language or development method

▸ **Non-functional requirements** may be more critical than functional requirements. If these are not met, the system is useless

Non-functional classifications

▶ **Product requirements**
  ▶ Requirements which specify that the **delivered product must behave in a particular way** e.g. execution speed, reliability, etc.
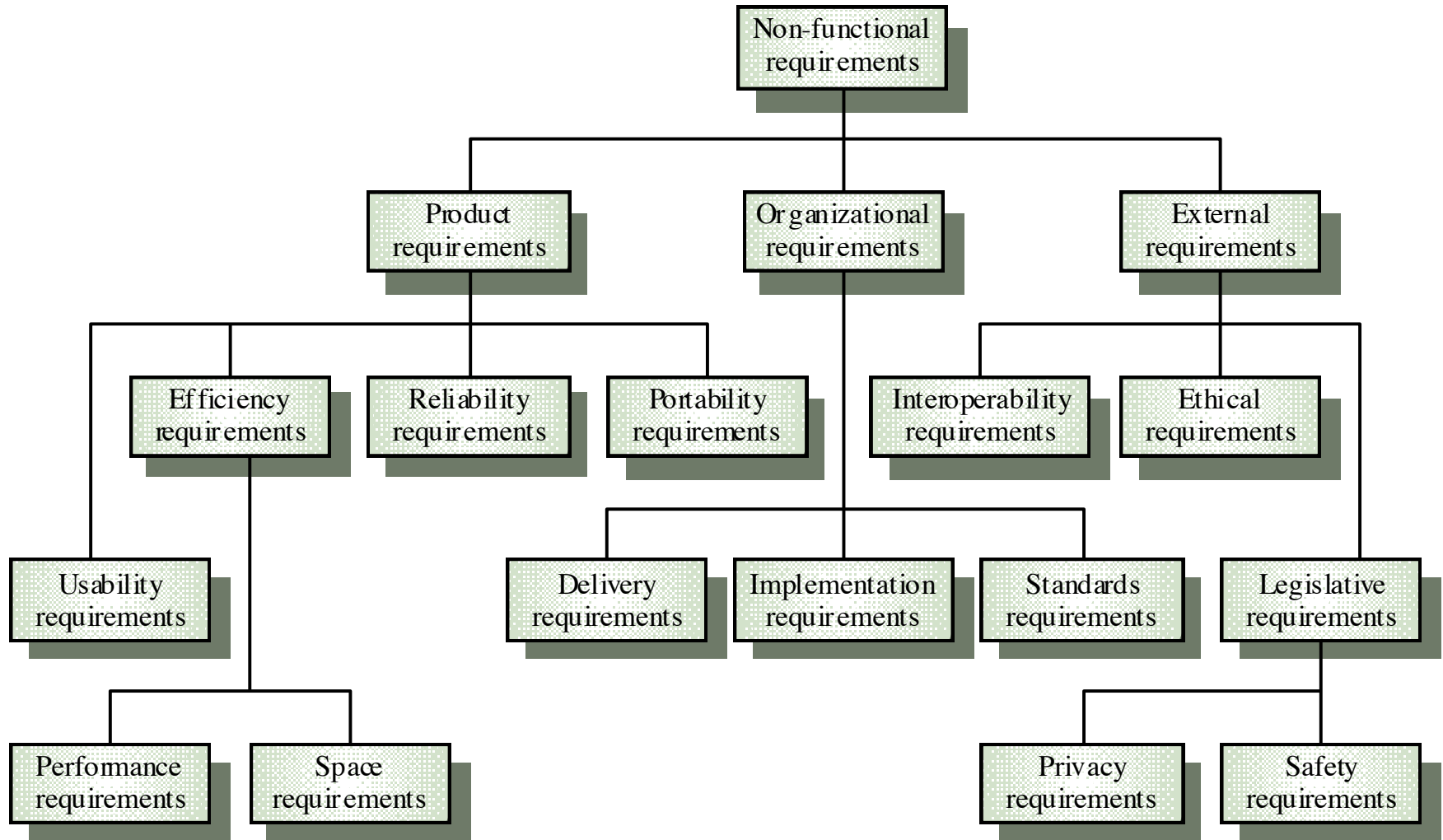
▶ **Organisational requirements**
  ▶ Requirements which are a **consequence of organisational policies and procedures** e.g. process standards used, implementation requirements, etc.

▶ **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 requirement types

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

Examples

▸ **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.

Requirements 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

Software Requirements

Requirements interaction

▸ **Conflicts between different non-functional requirements are common in complex systems**

▸ Spacecraft system

  ▸ To minimise weight, the number of separate chips in the system should be minimised

  ▸ To minimise power consumption, lower power chips should be used

  ▸ However, using low power chips may mean that more chips have to be used.

Which is the most critical requirement?

Software Requirements

Domain requirements

▶ **Derived from the application domain** and **describe system characteristics 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 problems

‣ **Understandability**

   ‣ Requirements are expressed in the language of the application domain

   ‣ This is often not understood by software engineers developing the system

‣ **Implicitness**

   ‣ Domain specialists understand the area so well that they do not think of making the domain requirements explicit

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**

Software Requirements

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

Guidelines for writing requirements

▶ Invent a standard format and use it for all requirements

▶ Use language in a consistent way. Use

shall for mandatory requirements,

should for desirable requirements

▶ Use text highlighting to identify key parts of the requirement

Avoid the use of computer jargon !!!

System requirements

– More detailed specifications of user requirements

▸ Serve as a basis for designing the system

▸ May be used as part of the system contract

▸ System requirements may be expressed using system models (will be discussed in later)

Problems with NL specification

‣ **Ambiguity**

  ‣ The 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

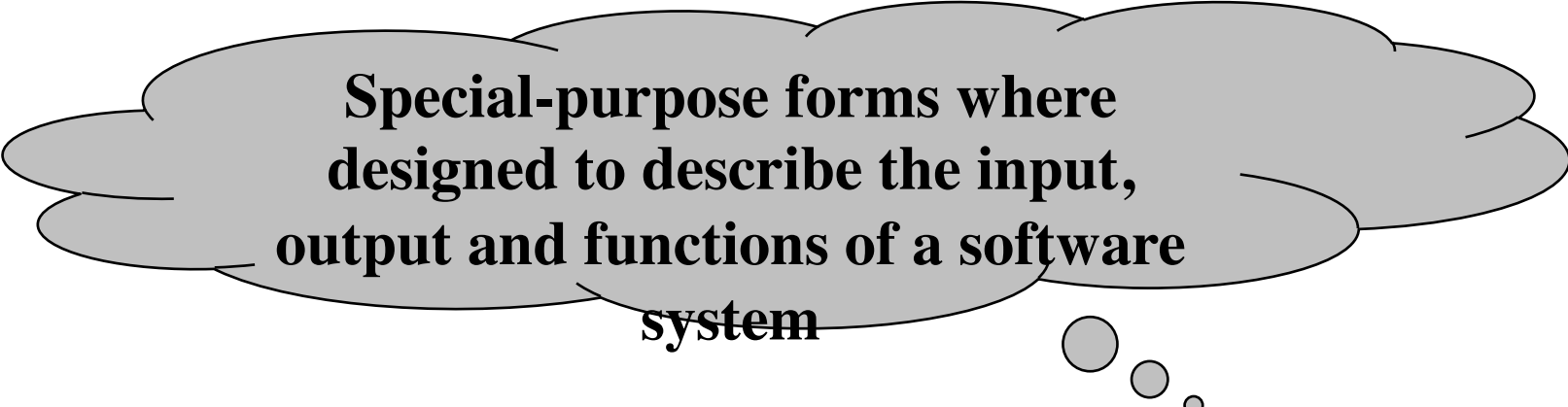Software Requirements

Requirements and design

▶ **In principle**, <u>requirements</u> should state **what the system should** do and the <u>design</u> 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 inter-operate with other systems that generate design requirements

   ▶ The use of a specific design may be a domain requirement

Alternatives to NL specification

Notation	Description
Structured natural language	This approach depends on defining standard forms or templates to express the requirements specification.
Design description languages	This approach uses a language like a programming language but with more abstract features to specify the requirements by defining an operational model of the system.
Graphical notations	A graphical language, supplemented by text annotations is used to define the functional requirements for the system. An early example of such a graphical language was SADT (Ross, 1977; Schoman and Ross, 1977). More recently, use-case descriptions (Jacobsen, Christerson et al., 1993) have been used.
Mathematical specifications	These are notations based on mathematical concepts such as finite-state machines or sets. These unambiguous specifications reduce the arguments between customer and contractor about system functionality. However, most customers don't understand formal specifications and are reluctant to accept it as a system contract.

Structured language specifications

▸ **A limited form of natural language** may be used to express requirements

▸ **This removes some of the problems** resulting from ambiguity and flexibility and imposes a degree of uniformity on a specification

Special-purpose forms where designed to describe the input, output and functions of a software system

▸ Often best supported using a forms-based approach

Form-based specifications

▸ Definition of the function or entity

▸ Description of inputs and where they come from

▸ Description of outputs and where they go to

▸ Indication of other entities required

▸ Pre and post conditions (if appropriate)

▸ The side effects (if any)

Form-based specifications

ECLIPSE/Workstation/Tools/DE/FS/3.5.1

Function: Add node

Description: Adds a node to an existing design.

Inputs: Node type, Node Position

Outputs: Design identifier

Pre/Post conditions:

Other attributes:

Definition: ECLIPSE/Workstation/Tools/DE/RD/3.5.1

Software Requirements

PDL-based requirements definition

Requirements may be defined operationally using a language like a programming language but with more flexibility of expression

▸ **Most appropriate in two situations**

- ▸ Where an operation is specified as a sequence of actions and the order is important
- ▸ When hardware and software interfaces have to be specified

▸ **Disadvantages are**

- ▸ The PDL may not be sufficiently expressive to define domain concepts
- ▸ The specification will be taken as a design rather than a specification

Part of an ATM specification

```
class ATM {
    // declarations here
    public static void main (String args[]) throws InvalidCard {
        try {
            thisCard.read () ;  // may throw InvalidCard exception
            pin = KeyPad.readPin () ; attempts = 1 ;
            while ( !thisCard.pin.equals (pin) & attempts < 4 )
                {    pin = KeyPad.readPin () ;  attempts = attempts + 1 ;
                }
                if (!thisCard.pin.equals (pin))
                        throw new InvalidCard ("Bad PIN");
            thisBalance = thisCard.getBalance () ;
            do {  Screen.prompt (" Please select a service ") ;
                service = Screen.touchKey () ;
                switch (service) {
                        case Services.withdrawalWithReceipt:
                                receiptRequired = true ;
```

PDL disadvantages

‣ **PDL may not be sufficiently expressive** to express the system functionality in an understandable way

‣ **Notation is only understandable to people with programming language knowledge**

‣ **The requirement may be taken as a design specification** rather than a model to help understand the system
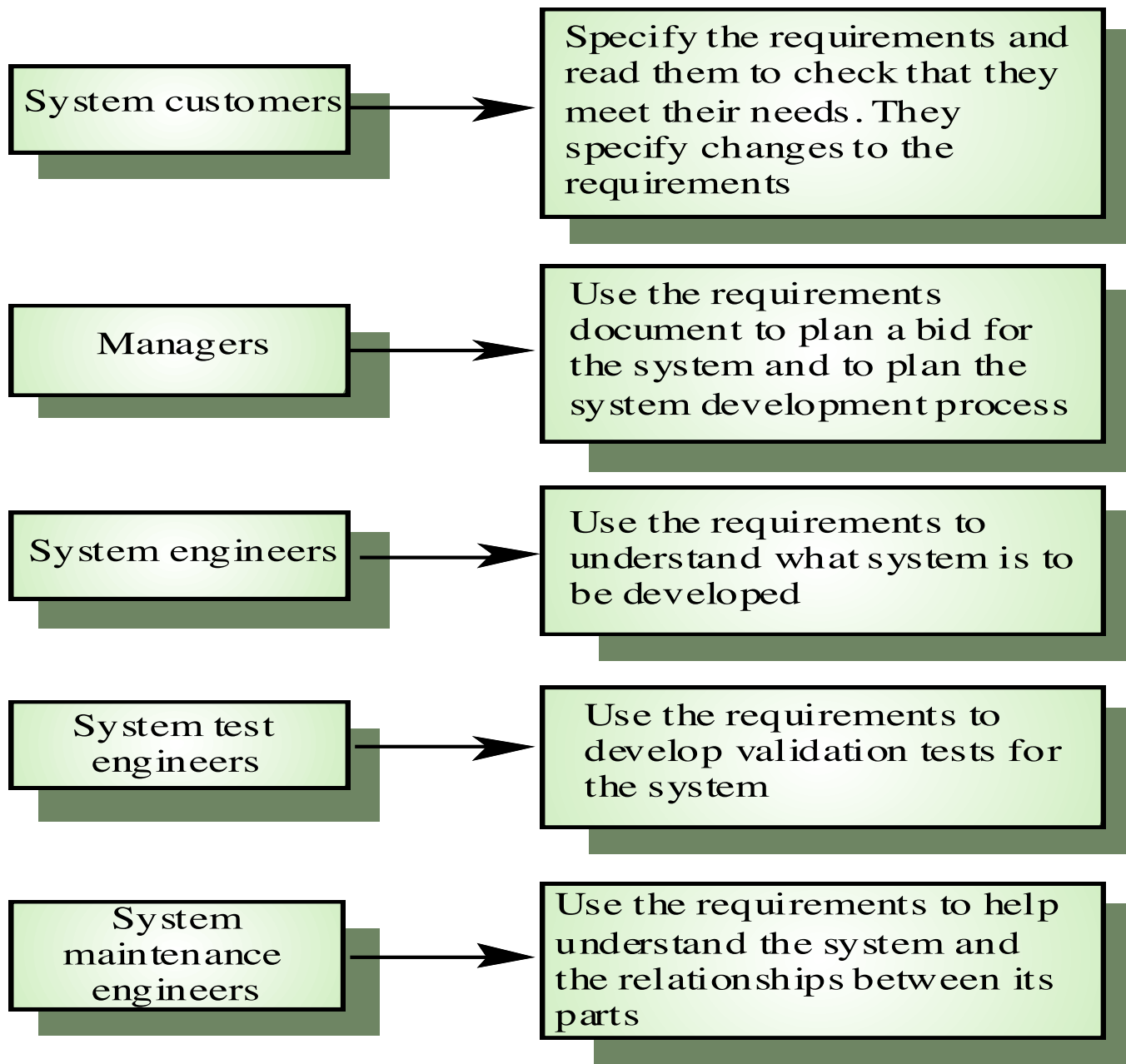
Interface specification

▸ **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

PDL interface description

```
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
```

Software Requirements

The requirements document

▸ The requirements document is the official statement of what is required of the system developers

▸ Should include both a definition and a specification of requirements

▸ It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it

System customers →	Specify the requirements and read them to check that they meet their needs. They specify changes to the requirements
Managers →	Use the requirements document to plan a bid for the system and to plan the system development process
System engineers →	Use the requirements to understand what system is to be developed
System test engineers →	Use the requirements to develop validation tests for the system
System maintenance engineers →	Use the requirements to help understand the system and the relationships between its parts

Users of a requirements document

Requirements document requirements

- Specify external system behaviour

- Specify implementation constraints

- Easy to change

- Serve as reference tool for maintenance

- Record forethought about the life cycle of the system i.e. predict changes

- Characterise responses to unexpected events

Software Requirements

IEEE requirements standard

- Introduction
- General description
- Specific requirements
- Appendices
- Index
- This is a generic structure that must be instantiated for specific systems

Software Requirements

Requirements document structure

▸ Introduction

▸ Glossary

▸ User requirements definition

▸ System architecture

▸ System requirements specification

▸ System models

▸ System evolution

▸ Appendices

▸ Index

Key points

▸ 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

Key points

▸ User requirements should be written in natural language, tables and diagrams

▸ System requirements are intended to communicate the functions that the system should provide

▸ System requirements may be written in structured natural language, a PDL or in a formal language

▸ A software requirements document is an agreed statement of the system requirements

Thank you!

*Next lecture … **System Models***

Software Requirements