# Chapter 5. Software Requirements

- Descriptions and specifications of a system



# Objectives

- To introduce the concepts of user and system requirements
- Requirements may be expressed in different notations
- To describe functional and non-functional requirements
- To explain two techniques for describing system requirements(structured natural language and PDL)
- To explain how software requirements may be organised in a SW requirements document

# Topics covered

- Functional and non-functional requirements
- User requirements
- System requirements
- The software 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(abstract statement of a service that a system provides or a constraint on the system)
- The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process(especially for a large SW developing project)
- Requirements must be written for several contractors to bid for the contract and understand how to validate the developing SW

# What is a requirement?

- It may range from a high-level abstract statement(user requirement) of a service or of a system constraint to a detailed mathematical functional specification(system requirement)
- This is inevitable that 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 abstraction (Davis '93)

"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 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(functional spec.)
  - A structured document setting out detailed descriptions of the system services and constraints. Written as a contract between buyer and system developer
- Software design specification
  - A detailed software description which can serve as a basis for a design or implementation. Written for developers(implemented-oriented document)



# Definitions and specifications

**user** **Requirements definition**

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

**System** **Requirements specification**

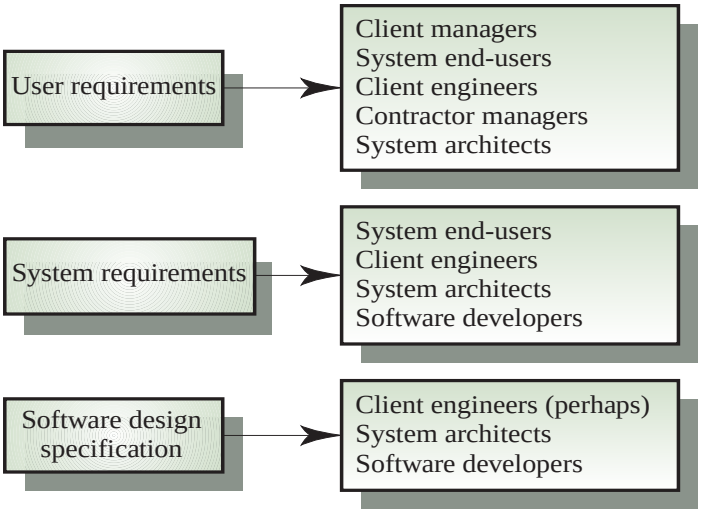
1.1 The user should be provided with facilities to define the type of external files.
1.2 Each external file type may have an associated tool which may be applied to the file.
1.3 Each external file type may be represented as a specific icon on the user's display.
1.4 Facilities should be provided for 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.

**File icon**

**Tool activation**

**User selection**
**File icon**

# Requirements readers






# 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.(security …)
- Domain requirements
  - Requirements that come from the application domain of the system and that reflect characteristics of that domain may be functional or non-functional
- The distinction among these types of requirement is not clear
- User requirement concerns with security(non-functional)

# 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 and usually describe in a fairly general way
- **Functional system requirements** should describe the system services in detail(I/O, exception)



# Examples of functional requirements

Functional requirements for an university library system:

- 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
- New requirements have to be made and changed →delay and cost increase
- Consider the term 'appropriate viewers'
  - User intention – general purpose viewer for all of the different document type
  - Developer interpretation - Provide a text viewer that shows the contents of the document and claim the requirement had been met under schedule pressure



# Requirements completeness and consistency

- In principle, requirements should be both complete and consistent
- Complete
  - All services required by the user should be defined. They should include descriptions of all facilities required
- Consistent
  - The requirement 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 especially for a large system→trade-off

# Non-functional requirements

- Define emergent system properties and constraints e.g. reliability, response time and storage requirements. Constraints are I/O device capability, system data representations in system interface, etc.
- Many non-functional requirement influence a whole system not partial
- Fail to meet a functional requirement may degrade the system but fail to meet a non-functional requirement may make the system unusable
- Ex. An aircraft system does not meet its reliability requirement, a real-time control system fail to meet its performance requirement
- Process requirements may also be specified assigned by 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

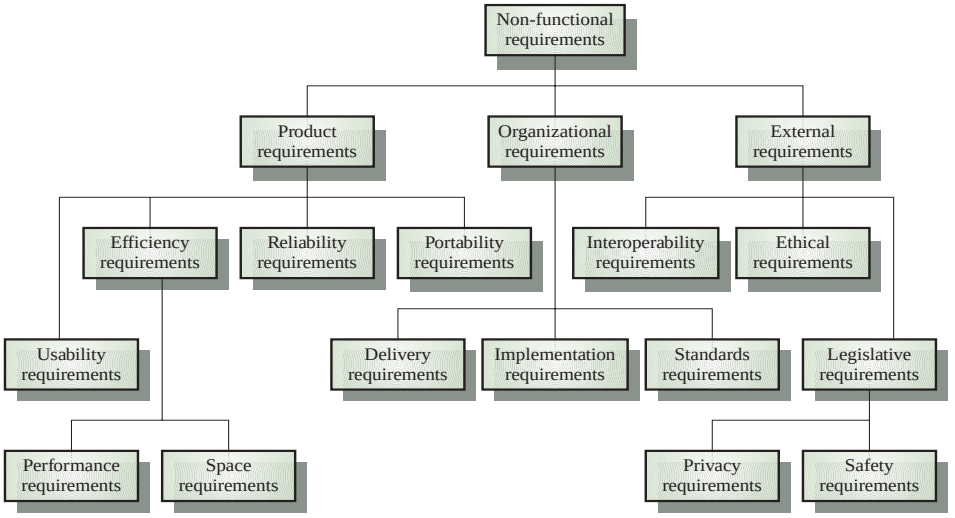


# Non-functional classifications

Classify different types of non-functional requirement:

- Product requirements
  - Requirements which specify the product behaviour e.g. performance requirement: execution speed, reliability requirement: acceptable failure rate
- Organisational requirements
  - Requirements which are a consequence of organisational policies and procedures e.g. process standards used, implementation requirements: PL or design method, delivery requirement: the product and its document delivered, etc.
- External requirements
  - Requirements which are derived from factors external to the system and its development process e.g. interoperability requirements: define how the system interacts with other system in organisation, legislative requirements: system operates within the law, etc.

# Non-functional requirement types






# Non-functional requirements examples

- Product requirement(program support environment)
  - 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 → system constraint
- Organisational requirement(follow a company standard)
  - 9.3.2 The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95 v.s. DOD-2167A
- External requirement(security)
  - 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, recover from failure
- Verifiable non-functional requirement
  - A statement using some measure that can be objectively tested
- Goals are helpful to developers as they transfer 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.
- Ideally, non-functional requirement should be expressed quantitatively using **metrics** that can be objectively tested

# Requirements measures(metrics)

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



# Requirements interaction

- Quantitative requirement spec is often difficult. The cost to verify the non-functional requirement is high
- 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?➔trade-off
- Memory allocation management(4MB for developing Ada) v.s. JVM in a Java phone
- It is difficult to separate functional and non-functional consideration to identify 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 functional requirements or define specific computations
- If domain requirements are not satisfied, the system may be unworkable➔domain knowledge



# Domain requirements problems

- Understandability
  - Requirements are expressed in the language of the application domain
  - Need domain-specific knowledge terminology or technology
  - Usually these **domain-specific terminology** is difficult for SW Engineer
  - This is often not understood by software engineers developing the system
- Implicitness
  - Need **domain expert** to leave the information from a requirement
  - Domain specialists(experts) understand the area so well that they do not think of making the domain requirements explicit
  - But the developer may implement the system unsatisfactory because they are not know about the domain-specific knowledge

## Library system domain requirements

- There shall be a standard user interface to all databases which shall be based on the **Z39.50 standard**.(constraint on System requirement)
- 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.(system provide an automatic delete-on-print facility)



## Automatic Train protection system

- System automatically stop a train if it goes through a red light
- The deceleration of the train shall be computed as:
  - $\mathbf{D_{train} = D_{control} + D_{gradient}}$

  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.➔need domain-specific terminology

# 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 intuitive diagrams



# Problems with natural language

Problem arises when requirements are written in NL:

- Lack of clarity
  - Precision and unambiguous are difficult without making the document difficult to read
- Requirements confusion
  - Functional and non-functional requirements, system goals and design information tend to be mixed-up
- Requirements amalgamation
  - Several different requirements may be expressed together as a single document

# Database and Editor grid requirement

**4.A.5** The database shall support the generation and control of configuration objects; that is, objects which are themselves groupings of other objects in the database. The configuration control facilities shall allow access to the objects in a version group by the use of an incomplete name.

**2.6 Grid facilities** To assist in the positioning of entities on a diagram, the user may turn on a grid in either centimetres or inches, via an option on the control panel. Initially, the grid is off. The grid may be turned on and off at any time during an editing session and can be toggled between inches and centimetres at any time. A grid option will be provided on the reduce-to-fit view but the number of grid lines shown will be reduced to avoid filling the smaller diagram with grid lines.



# Requirement problems

- Database requirements includes both conceptual and detailed information(separate into system requirement)
  - Describes the **concept** of **configuration control** facilities
  - Includes the **detail** that objects may be accessed using an **incomplete name**
- Grid requirement mixes three different kinds of requirement
  - **Conceptual functional** requirement (the **need for a grid**)
  - **Non-functional** requirement (**grid units → cm or inch?**)
  - **Non-functional** UI requirement (**grid switching on/off by user**)

# Structured presentation

**2.6 Grid facilities**

**2.6.1** **The editor shall provide a grid facility where a matrix of horizontal and vertical lines provide a background to the editor window.** This grid shall be a p assive grid where the alignment of entities is the user's responsibility.

*Rationale*: A grid helps the user to create a tidy diagram with well-spaced entities. Although an active grid, where entities 'snap-to' grid lines can be useful, the positioning is imprecise. The user is the best person to decide where entities should be positioned.

*Specification*: ECLIPSE/WS/Tools/DE/FS Section 5.6



# Detailed user requirement(too detailed)

**3.5.1 Adding nodes to a design**

3.5.1.1 **The editor shall provide a facility for users to add nodes of a specified type to their design.**

3.5.1.2 The sequence of actions to add a node should be as follows:

1. The user should select the type of node to be added.

2. The user should move the cursor to the approximate node position in the diagram and indicate that the node symbol should be added at that point.

3. The user should then drag the node symbol to its final position.

*Rationale*: The user is the best person to decide where to position a node on the diagram. This approach gives the user direct control over node type selection and positioning.

*Specification*: ECLIPSE/WS/Tools/DE/FS. Section 3.5.1

back

## Guidelines for writing requirements

- Invent a standard format and use it for all requirements as your standard
- 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 and implementing*** the system
- May be used as part of the system contract
- System requirements may be expressed using system models discussed in Chapter 7(context, behavioural, data, object model)

# Requirements and design

- In principle, requirements should state what the system should do and not how it should be implemented
- In practice, requirements and design are inseparable because
  - A initial system architecture may be designed to structure the requirements spec.
  - The system may inter-operate with other existing systems that generate the new system design requirements
  - The use of a specific design (for reliability) may be a domain requirement(an external system requirement)



# Problems with NL specification

NL is often used to write system requirement spec. Further problems arise when it is used for detailed spec.

- 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 many different ways in the specification
- Lack of modularisation
  - NL structures are inadequate to structure system requirements

➔ Requirement spec. written in NL is prone to misunderstanding

# 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. I discuss these in the following chapter. |
| 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. I discuss formal specification in Chapter 9. |



# 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 uses a degree of uniformity on a specification➔limit the technology use
- Often supported using a ***forms-based*** approach(***templates***)
- Incorporate ***control structure*** derived from PL and ***graphical highlight*** to partition the spec.
- Define one or more standard forms or templates to express the requirements

# Form-based specifications

For specifying the requirement, a standard form should include the following information:

1. Definition of the ***function*** or ***entity***
2. Description of ***inputs*** and where they come from
3. Description of ***outputs*** and where they go to
4. ***Indication of what other entities are used(required part)***
5. ***Pre*** (set out what must be true **before the function is called**) and ***post*** conditions (specify what is true **after the function is called**)
6. The side effects (if any)

➔some ambiguity remains in the spec.



# Form-based node specification

ECLIPSE/Workstation/Tools/DE/FS/3.5.1

P.108, Fig. 5.11

| | |
|---|---|
| **Function** | Add node |
| **Description** | Adds a node to an existing design. The user selects the type of node, and its position. When added to the design, the node becomes the current selection. The user chooses the node position by moving the cursor to the area where the node is added. |
| **Inputs** | Node type, Node position, Design identifier. |
| **Source** | Node type and Node position are input by the user, Design identifier from the database. |
| **Outputs** | Design identifier. |
| **Destination** | The design database. The design is committed to the database on completion of the operation. |
| **Requires** | Design graph rooted at input design identifier. |
| **Pre-condition** | The design is open and displayed on the user's screen. |
| **Post-condition** | The design is unchanged apart from the addition of a node of the specified type at the given position. |
| **Side-effects** | None |

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

# PDL-based requirements definition

- A PDL is a language derived from a PL like Java or Ada
- The advantage of using a PDL is that it may be checked syntactically and semantically by SW tools
- Requirements may be defined operationally using a language like a programming language but with more flexibility of expression
- PDL result in very detailed spec. It's too close to the implementation for inclusion in a requirement document
- Use PDL appropriate in two situations
  - Where an operation is specified as a sequence of actions and the order of execution is important
  - When hardware and software interfaces have to be specified



# 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 () ;
```

# Part of an ATM specification

```
switch (service) {
case Services.withdrawalWithReceipt:
          receiptRequired = true ;
case Services.withdrawalNoReceipt:
        …
   }
  }
 while (service != Service.quit) ….
}
Catch(InvalidCard e) { Screen.printmsg("Invalid card or PIN"); }
} // main()
}// ATM
```



# 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

➔ PDL may be used to define control sequences or interfaces in more detail

# Interface specification

- The new 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 are accessed by calling interface procedures
  - Data structures that are exchanged from one subsystem to another(class definition in Java)
  - Data representations are established for an existing subsystem

    ➔ not suitable to use Java to do this
- Formal notations are an effective technique for interface specification(chapter 9)



# PDL Interface specification

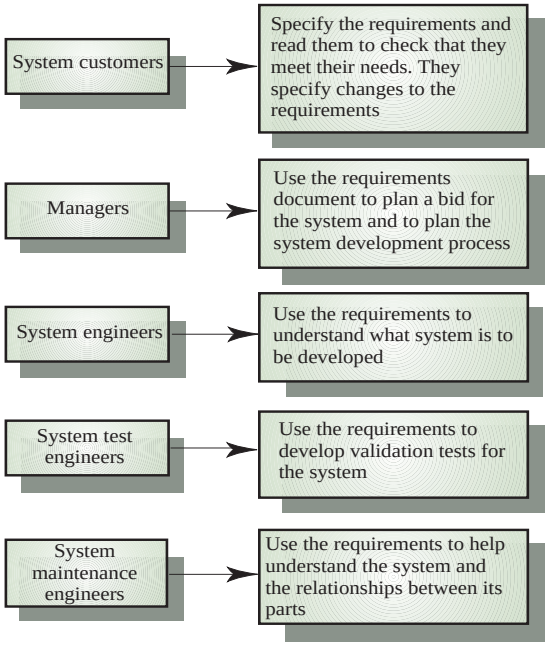
```
Interface PrintServer {
  // defines an abstract print server
  // requires: interface Printer, PrintDoc
  // Provides: initialize, print, displayPrintQueue,
  //              cancelPrintJob, switchPrinter
  void initialize(Printer p) ;
  void print(Printer p, PrintDoc d) ;
  void displayPrintQueue(Printer p);
  void cancelPrintJob(Printer p, PrintDoc d);
  void switchPrinter(Printer p1, Printer p2, PrintDoc d);
}
```

➔ **Application Program Interface(API)**

# The requirements document

- SW requirement document(SRS) is the official statement of what is required of the system developers
- Include user requirement for a system and detailed spec. of system requirement (user+system requirements)
- If there are a large number of requirements, the detailed system requirements may be presented as separate documents
- 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






Users of a requirements document

# Requirements document requirements

Heninger suggests 6 requirements which a SRS should satisfy:

- Specify external system behaviour
- Specify implementation constraints
- Easy to change
- Serve as reference tool for system maintainer
- Record forethought about the life cycle of the system i.e. predict changes
- Characterise acceptable responses to unexpected events



# IEEE requirements standard

IEEE standards suggests the requirement document structure:

- Introduction(簡介)
- General description(一般性描述)
- Specific requirements(functional, Non-functional and interface requirements)
- Appendices
- Index

→This is a generic structure that must be used for specific systems

# Requirements document structure

The author suggests the SRS structures:

- Preface(序)
- Introduction
- Glossary(專有名詞解釋)
- 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



## HomeWork

- 5.2
- 5.3
- 5.6 以你/妳的計劃為例
- 5.7
- Propose your project's user requirement and system requirements(partial PDL, see example as followed)
- Prepare your project content to fit the IEEE standards content or author suggest contents?

# Part of an ATM specification

class ATM {
// **input : 金融卡,密碼, 服務**
// **output :訊息,金融卡**
**// pre-condition:**
**// post-condition:**

1. 讀取金融卡
2. 輸入密碼:
   判斷密碼錯誤不可超過3次
3. 選取服務直到選出退出

}