
Chapter 5

Software Requirements: Description and specification of a system

Objectives

- To introduce the concepts of user and system requirements
- To describe functional and non-functional requirements
- To explain two techniques for describing system requirements
- To explain how software requirements may be organized in a requirements document

What is a requirement?

It may range from a high-level abstract statement of a service or of a system constraint to a detailed specification of mathematical function. It is so because

- It may be used as the basis for a bid for a contract - therefore must be open to interpretation, or
- It may be used as the basis for the contract itself - therefore must be defined in detail.

Requirements definition vs. specification

- Requirements definition: requirements described in the language of problem domain
- Requirements specification: requirements described in the language of software engineering

Requirements definition: example

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

Requirements specification: example (continued)

- It should allow the user to define the type of files.
- It should be able to process every type of file defined.
- Each file is to be displayed by an icon uniquely associated with its type.
- It should allow the user to choose the icon associated with each file type.

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**
Also known as functional specification, it is 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

Various requirements and their intended readers

	Client managers	Client engineers	End users	Contractor managers	System architect	Software developers
User requirements	X	X	X	X	X	
System requirements		X			X	X
Software design requirements					X	X

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, depending 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 whereas functional system requirements should describe the system services in detail.

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, 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 impossible to produce a complete and consistent requirements document.

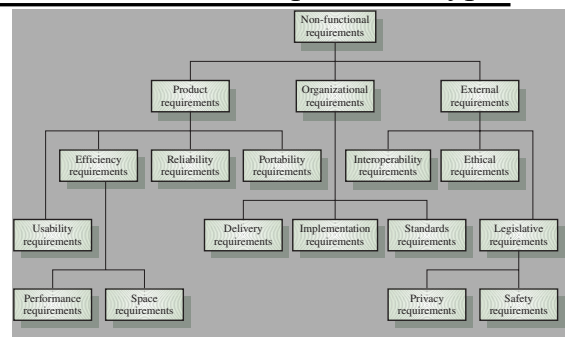
Non-functional requirements

- 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

Examples of non-functional requirements

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

Non-functional 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
- **Organizational 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 organized in such a way that user errors are minimized.

- A verifiable non-functional requirement

Experienced controllers shall be able to use all the system functions after a total of two hours training. After this training, the average number of errors made by experienced users shall not exceed two per day.

Metrics for non-functional req.

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

Requirements interaction

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

Problem-domain requirements

- Derived from the application domain and describe system characteristics and features that reflect the problem 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

Library system domain requirements

- 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:

$$D_{\text{train}} = D_{\text{control}} + D_{\text{gradient}}$$
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.

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

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, and 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 discussed in Chapter 7

Requirements and design

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

Problems with natural-language 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 modularization
 - NL structures are inadequate to structure system requirements

Alternatives to natural language

- Structured natural languages
- Design description languages
- Graphical notations
- Mathematical specifications

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
- 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)

ECLIPSE/Workstation/Tools/DE/FS/3.5.1 Form-based node specification

Function Add node

Description Add a node to an existing design. The user selects the type of node, and its position. When added to the design, the node becomes a permanent 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 design database.

Outputs Design identifier.

Destination The design database. The design is committed to the database on completion of 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

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

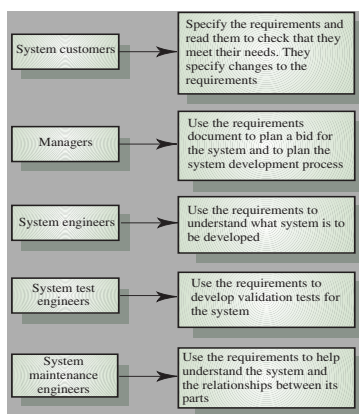
- Most systems must operate with other systems and the operating interfaces must be specified as part of the requirements
- Three types of interface may have to be defined
 - Procedural interfaces
 - Data structures that are exchanged
 - Data representations
- Formal notations are an effective technique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

PDL interface description

```
interface PrintServer {  
  // defines an abstract print 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
```

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 state WHAT the system should do rather than HOW it should be done



Users of a requirements document

Desired properties of a requirements spec.

- 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
- Characterize responses to unexpected events

IEEE requirements standard

- Introduction
- General description
- Specific requirements
- Appendices
- Index

This is a generic structure that must be instantiated for specific systems

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 (continued)

- User requirements should be written in a natural language supplemented by tables and diagrams.
- System requirements are intended to communicate the functions that the system should provide, and 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.