Lecture 4

System Requirements


Software Requirements

- Descriptions and specifications 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 organised in a 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
- The requirements themselves are 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.

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

Davis, 1993

Types of requirement

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

Definitions and specifications

Requirements definition

1. The software must provide a means of representing and accessing external files created by other tools.
2. Each external file type may have an associated tool which may be applied to the file.
3. Each external file type may be represented as a specific icon on the user’s display.
4. Facilities should be provided for the user to examine an external file type and define it to the user.
5. When a user selects an icon representing an external file, the effect of that action is to invoke the tool associated with the type of the external file to the file represented by the selected icon.

Requirements specification

- 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 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 (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 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

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

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?

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
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 \( D_{\text{gradient}} \) is \( 9.81 \text{ms}^2 \) * compensated gradient/alpha and where the values of \( 9.81 \text{ms}^2/\alpha \) 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

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

Database requirement

4A.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.
Editor grid requirement

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.

Structured presentation

Facilities

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 passive 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

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 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 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.

Alternatives to NL specification

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured natural language description</td>
<td>This approach depends on defining standard forms or templates to express the requirements specification. 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.</td>
</tr>
<tr>
<td>Graphical notations</td>
<td>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; Schuman and Ross, 1977). More recently, use-case descriptions (Jacobson, Christerson et al., 1993) have been used. I discuss these in the following chapter.</td>
</tr>
<tr>
<td>Mathematical specifications</td>
<td>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.</td>
</tr>
</tbody>
</table>

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).

Form-based node specification

<table>
<thead>
<tr>
<th>ECLIPSE/Workstation/Tools/DE/RD/3.5.1</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>Add node.</td>
</tr>
<tr>
<td>Description</td>
<td>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 the current selection. The user chooses the node position by moving the cursor to the area where the node is added.</td>
</tr>
<tr>
<td>Inputs</td>
<td>Node type, Node position, Design identifier.</td>
</tr>
<tr>
<td>Source</td>
<td>Node type and Node position are input by the user, Design identifier from the database.</td>
</tr>
<tr>
<td>Outputs</td>
<td>Design identifier.</td>
</tr>
<tr>
<td>Destination</td>
<td>The design database. Design is committed to the database on completion of the operation.</td>
</tr>
<tr>
<td>Requires</td>
<td>Design graph rooted at input design identifier.</td>
</tr>
<tr>
<td>Pre-condition</td>
<td>The design is open and displayed on the user's screen.</td>
</tr>
<tr>
<td>Post-condition</td>
<td>The design is exchanged apart from the addition of a node of the specified type at the given position.</td>
</tr>
<tr>
<td>Side-effects</td>
<td>None.</td>
</tr>
<tr>
<td>Definition</td>
<td>ECLIPSE/Workstation/Tools/DE/RD/3.5.1</td>
</tr>
</tbody>
</table>
**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 ;
            var (thisCard.pins.equals (pin) & attempts < 4 )
                { pin = KeyPad.readPin () ; attempts = attempts + 1 ;
                if (!thisCard.pins.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 ;
                        break;
                }
            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) ;
                // anything here
            }
        }
    
**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

**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
Users of a requirements document

- 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.

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

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

- 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
Requirements Engineering Processes

- Processes used to discover, analyse and validate system requirements

Objectives

- To describe the principal requirements engineering activities
- To introduce techniques for requirements elicitation and analysis
- To describe requirements validation
- To discuss the role of requirements management in support of other requirements engineering processes

Topics covered

- Feasibility studies
- Requirements elicitation and analysis
- Requirements validation
- Requirements management

Requirements engineering processes

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

Feasibility studies

- A feasibility study decides whether or not the proposed system is worthwhile
- A short focused study that checks
  - If the system contributes to organisational objectives
  - If the system can be engineered using current technology and within budget
  - If the system can be integrated with other systems that are used
Feasibility study implementation

- Based on information assessment (what is required), information collection and report writing
- Questions for people in the organisation
  - What if the system wasn’t implemented?
  - What are current process problems?
  - How will the proposed system help?
  - What will be the integration problems?
  - Is new technology needed? What skills?
  - What facilities must be supported by the proposed system?

Elicitation and analysis

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

Problems of requirements analysis

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

The requirements analysis

- Domain understanding
- Requirements collection
- Classification
- Conflict resolution
- Prioritisation
- Requirements checking

System models

- Different models may be produced during the requirements analysis activity
- Requirements analysis may involve three structuring activities which result in these different models
  - Partitioning. Identifies the structural (part-of) relationships between entities
  - Abstraction. Identifies generalities among entities
  - Projection. Identifies different ways of looking at a problem
- System models covered in Chapter 7
Viewpoint-oriented elicitation

- Stakeholders represent different ways of looking at a problem or problem viewpoints.
- This multi-perspective analysis is important as there is no single correct way to analyse system requirements.

Banking ATM system

- The example used here is an auto-teller system which provides some automated banking services.
- I use a very simplified system which offers some services to customers of the bank who own the system and a narrower range of services to other customers.
- Services include cash withdrawal, message passing (send a message to request a service), ordering a statement and transferring funds.

Autoteller viewpoints

- Bank customers
- Representatives of other banks
- Hardware and software maintenance engineers
- Marketing department
- Bank managers and counter staff
- Database administrators and security staff
- Communications engineers
- Personnel department

Types of viewpoint

- Data sources or sinks
  - Viewpoints are responsible for producing or consuming data. Analysis involves checking that data is produced and consumed and that assumptions about the source and sink of data are valid.
- Representation frameworks
  - Viewpoints represent particular types of system model. These may be compared to discover requirements that would be missed using a single representation. Particularly suitable for real-time systems.
- Receivers of services
  - Viewpoints are external to the system and receive services from it. Most suited to interactive systems.

External viewpoints

- Natural to think of end-users as receivers of system services.
- Viewpoints are a natural way to structure requirements elicitation.
- It is relatively easy to decide if a viewpoint is valid.
- Viewpoints and services may be sued to structure non-functional requirements.

Method-based analysis

- Widely used approach to requirements analysis. Depends on the application of a structured method to understand the system.
- Methods have different emphases. Some are designed for requirements elicitation, others are close to design methods.
- A viewpoint-oriented method (VORD) is used as an example here. It also illustrates the use of viewpoints.
The VORD method

VORD process model
- Viewpoint identification
  - Discover viewpoints which receive system services and identify the services provided to each viewpoint
- Viewpoint structuring
  - Group related viewpoints into a hierarchy. Common services are provided at higher levels in the hierarchy
- Viewpoint documentation
  - Refine the description of the identified viewpoints and services
- Viewpoint-system mapping
  - Transform the analysis to an object-oriented design

VORD standard forms

<table>
<thead>
<tr>
<th>Viewpoint template</th>
<th>Service template</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reference:</td>
<td>Reference:</td>
</tr>
<tr>
<td>The viewpoint name.</td>
<td>The service name.</td>
</tr>
<tr>
<td>Attributes:</td>
<td>Rationale:</td>
</tr>
<tr>
<td>Attributes:</td>
<td>Reason why the service is provided.</td>
</tr>
<tr>
<td>Events:</td>
<td>Specification:</td>
</tr>
<tr>
<td>A reference to a set of event scenarios describing how the system reacts to viewpoint events.</td>
<td>Reference to a list of service specifications. These may be expressed in different notations.</td>
</tr>
<tr>
<td>Services:</td>
<td>Viewpoints:</td>
</tr>
<tr>
<td>A reference to a set of service descriptions.</td>
<td>List of viewpoint names receiving the service.</td>
</tr>
<tr>
<td>Sub-VPs:</td>
<td>Non-functional requirements:</td>
</tr>
<tr>
<td>The names of sub-</td>
<td>Reference to a list of non-functional requirements which constrain the service.</td>
</tr>
<tr>
<td>viewpoints.</td>
<td>Reference to a list of system objects which provide the service.</td>
</tr>
<tr>
<td>Provider:</td>
<td>Provider:</td>
</tr>
</tbody>
</table>

VORD standard forms

Viewpoint service information

- **ACCOUNT HOLDER**
  - Service list:
    - Withdraw cash
    - Query balance
    - Order cheques
    - Send message
    - Transaction list
    - Order statement
    - Transfer funds

- **FOREIGN CUSTOMER**
  - Service list:
    - Withdraw cash
    - Query balance

- **BANK TELLER**
  - Service list:
    - Run diagnostics
    - Add cash
    - Add paper
    - Send message

Viewpoint data/control

- **ACCOUNT HOLDER**
  - Control input:
    - Start transaction
    - Cancel transaction
    - End transaction
    - Select service
  - Data input:
    - Card details
    - PIN
    - Amount required
    - Message
Scenarios

- Scenarios are descriptions of how a system is used in practice
- They are helpful in requirements elicitation as people can relate to these more readily than an abstract statement of what they require from a system
- Scenarios are particularly useful for adding detail to an outline requirements description

Event scenarios

- Event scenarios may be used to describe how a system responds to the occurrence of some particular event such as ‘start transaction’
- VORD includes a diagrammatic convention for event scenarios:
  - Data provided and delivered
  - Control information
  - Exception processing
  - The next expected event

Scenario descriptions

- System state at the beginning of the scenario
- Normal flow of events in the scenario
- What can go wrong and how this is handled
- Other concurrent activities
- System state on completion of the scenario
Notation for data and control analysis

- Ellipses: data provided from or delivered to a viewpoint
- Control information enters and leaves at the top of each box
- Data leaves from the right of each box
- Exceptions are shown at the bottom of each box
- Name of next event is in box with thick edges

Exception description

- Most methods do not include facilities for describing exceptions
- In this example, exceptions are
  - Timeout. Customer fails to enter a PIN within the allowed time limit
  - Invalid card. The card is not recognised and is returned
  - Stolen card. The card has been registered as stolen and is retained by the machine

Use cases

- Use-cases are a scenario based technique in the UML which identify the actors in an interaction and which describe the interaction itself
- A set of use cases should describe all possible interactions with the system
- Sequence diagrams may be used to add detail to use-cases by showing the sequence of event processing in the system

Lending use-case

Library use-cases

Catalogue management
Social and organisational factors

- Software systems are used in a social and organisational context. This can influence or even dominate the system requirements
- Social and organisational factors are not a single viewpoint but are influences on all viewpoints
- Good analysts must be sensitive to these factors but currently no systematic way to tackle their analysis

Example

- Consider a system which allows senior management to access information without going through middle managers
  - Managerial status. Senior managers may feel that they are too important to use a keyboard. This may limit the type of system interface used
  - Managerial responsibilities. Managers may be made redundant may deliberately provide misleading or incomplete information so that the system will fail

Ethnography

- A social scientist spends a considerable time observing and analysing how people actually work
- People do not have to explain or articulate their work
- Social and organisational factors of importance may be observed
- Ethnographic studies have shown that work is usually richer and more complex than suggested by simple system models

Focused ethnography

- Developed in a project studying the air traffic control process
- Combines ethnography with prototyping
- Prototype development results in unanswered questions which focus the ethnographic analysis
- Problem with ethnography is that it studies existing practices which may have some historical basis which is no longer relevant

Ethnography and prototyping

- Requirements that are derived from the way that people actually work rather than the way which process definitions suggest that they ought to work
- Requirements that are derived from cooperation and awareness of other people’s activities
Requirements validation
• Concerned with demonstrating that the requirements define 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
• Prototyping
  – Using an executable model of the system to check requirements. Covered in Chapter 8
• Test-case generation
  – Developing tests for requirements to check testability
• Automated consistency analysis
  – Checking the consistency of a structured requirements description

Requirements reviews
• Regular reviews should be held while the requirements definition is being formulated
• Both client and contractor staff should be involved in reviews
• Reviews may be formal (with completed documents) or informal. Good communications between developers, customers and users can resolve problems at an early stage

Review checks
• Verifiability. Is the requirement realistically testable?
• Comprehensibility. Is the requirement properly understood?
• Traceability. Is the origin of the requirement clearly stated?
• Adaptability. Can the requirement be changed without a large impact on other requirements?

Automated consistency checking
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 change

- The priority of requirements from different viewpoints changes during the development process.
- System customers may specify requirements from a business perspective that conflict with end-user requirements.
- The business and technical environment of the system changes during its development.

Requirements evolution

- Enduring requirements. Stable requirements derived from the core activity of the customer organisation. E.g. a hospital will always have doctors, nurses, etc. May be derived from domain models.
- Volatile requirements. Requirements which change during development or when the system is in use. In a hospital, requirements derived from health-care policy.

Classification of requirements

- Mutable requirements:
  - Requirements that change due to the system’s environment.
- Emergent requirements:
  - Requirements that emerge as understanding of the system develops.
- Consequential requirements:
  - Requirements that result from the introduction of the computer system.
- Compatibility requirements:
  - Requirements that depend on other systems or organisational processes.

Requirements management planning

- During the requirements engineering process, you have to plan:
  - Requirements identification:
    - How requirements are individually identified.
  - A 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.
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

A traceability matrix

<table>
<thead>
<tr>
<th>Req.</th>
<th>1.1</th>
<th>1.2</th>
<th>1.3</th>
<th>2.1</th>
<th>2.2</th>
<th>3.1</th>
<th>3.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>R</td>
<td>U</td>
<td>R</td>
<td>U</td>
<td>R</td>
<td>U</td>
<td>U</td>
</tr>
<tr>
<td>1.1</td>
<td></td>
<td></td>
<td>R</td>
<td>U</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R</td>
<td></td>
</tr>
<tr>
<td>3.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>R</td>
</tr>
</tbody>
</table>

CASE tool support

• Requirements storage
  – Requirements should be managed in a secure, managed data store
• Change management
  – The process of change management is a workflow process whose stages can be defined and information flow between these stages partially automated
• Traceability management
  – Automated retrieval of the links between requirements

Requirements change management

• Should apply to all proposed changes to the requirements
• Principal stages
  – Problem analysis. Discuss requirements problem and propose change
  – Change analysis and costing. Assess effects of change on other requirements
  – Change implementation. Modify requirements document and other documents to reflect change

Key points

• The requirements engineering process includes a feasibility study, requirements elicitation and analysis, requirements specification and requirements management
• Requirements analysis is iterative involving domain understanding, requirements collection, classification, structuring, prioritisation and validation
• Systems have multiple stakeholders with different requirements
Key points

- 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

Tutorial Question

- Sommerville (6th Edition) Chapter 5:
  - Question 5.2
  - Question 5.6
  - Question 5.9
- Sommerville (6th Edition) Chapter 6:
  - Question 6.2
  - Question 6.6
  - Question 6.9