# Software Requirements

## Objectives

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

## Requirements engineering

- Requirements engineering is a systematic way of developing requirements through an iterative process of analyzing a problem, documenting the resulting observations, and checking the accuracy of the understanding gained.
- Requirements engineering is comprised of two major tasks: analysis and modeling
- The requirements themselves are the descriptions of the system services and constraints that are generated during the requirements engineering process.

## What is a requirement?

- A software requirement is a condition or capability needed by a user to solve a problem or achieve an objective and that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.
- Two classes of requirements:
  - Functional requirements – What the program does
  - Non-functional requirements – Attributes about the program

## 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.

## 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's functions, services and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor.

## Capturing the requirements

- Requirement: a feature of the system or a description of something the system is capable of doing in order to fulfill the system's purpose
- Three kinds of requirements:
  - those that absolutely must be met
  - those that are highly desirable but not necessary
  - those that are possible but could be eliminated

## Characteristics of requirements

- Are they correct?
- Are they consistent?
- Are they complete?
- Are they realistic?
- Does each describe something the customer needs?
- Are they verifiable?
- Are they traceable?

## 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, security requirements, performance, 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.

  “shall” usually means a mandatory requirement

- 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.
- Are these good requirements?

## 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.
- Natural language (English) poor for expressing precise statements

## Requirements completeness and consistency

- In principle, requirements should be both complete and consistent.
- Complete
  - ❑ They include descriptions of all facilities required.
    - What is assumed as domain knowledge as a requirement?
    - For example, does a requirement to list names in alphabetical order require a definition of "alphabetical order"?
- Consistent
  - ❑ There is 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

- These define system properties and constraints e.g. reliability, safety, security, performance, 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 may be useless.

## Functional vs. non-functional requirements

- Functional: describes an interaction between the system and its environment
- Examples:
  - System shall communicate with external system X.
  - What conditions must be met for a message to be sent

- Non-functional: describes a restriction or constraint that limits our choices for constructing a solution
- Examples:
  - Paychecks distributed no more than 4 hours after initial data are read.
  - System limits access to senior managers.

## Non-functional classifications

- Product requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, safety, security, 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 requirements examples

What is "simple?"

- Product requirement

  8.1 The user interface for LIBSYS shall be implemented as simple HTML without frames or Java applets.

- 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 (security) 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.

This means implement no function the operators may invoke directly or indirectly. How do you check that?

## 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. ("May" or "should" is often used instead of "shall.")
- 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 shall be responsive to any user input.
- **A verifiable non-functional requirement**
  - The system shall respond to any user input within 0.01 seconds.

This is now a measurable and testable requirement

## Requirements measures

| Property | Measure |
|---|---|
| Speed | Processed transactions/second<br>User/Event response time<br>Screen refresh time |
| Size | M Bytes<br>Number of ROM 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

- Constraints are a type of non-functional requirement that is imposed by the client that restricts the implementation of the system or the development process.
- Conflicts between different non-functional requirements are common in complex systems.
  - E.g.,Spacecraft system
    - To minimize weight, the number of separate chips in the system should be minimized.
    - To minimize power consumption, lower power (i.e., slower) 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.
- Domain requirements 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.
  - ❑ On the other hand, users rarely understand the requirements jargon of the software engineer.
- 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 in such a way that they are understandable by system users who don’t have detailed technical knowledge.
- “User requirements are defined using natural language, tables and diagrams as these can be understood by all users.” -- Yuk!

## 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.
- 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.

## Major problem with requirements process

| *How developers see users* | *How users see developers* |
|---|---|
| Users don't know what they want. | Developers don't understand operational needs. |
| Users can't articulate what they want. | Developers place too much emphasis on technicalities. |
| Users have too many needs that are politically motivated. | Developers try to tell us how to do our jobs. |
| Users want everything right now. | Developers can't translate clearly-stated needs into a successful system. |
| Users can't prioritize needs. | Developers say no all the time. |
| Users refuse to take responsibility for the system. | Developers are always over budget. |
| Users are unable to provide a usable statement of needs. | Developers are always late. |
| Users are not committed to system development projects. | Developers ask users for time and effort, even to the detriment of the users' important primary duties. |
| Users are unwilling to compromise. | Developers set unrealistic standards for requirements definition. |
| Users can't remain on schedule. | Developers are unable to respond quickly to legitimately changing needs. |

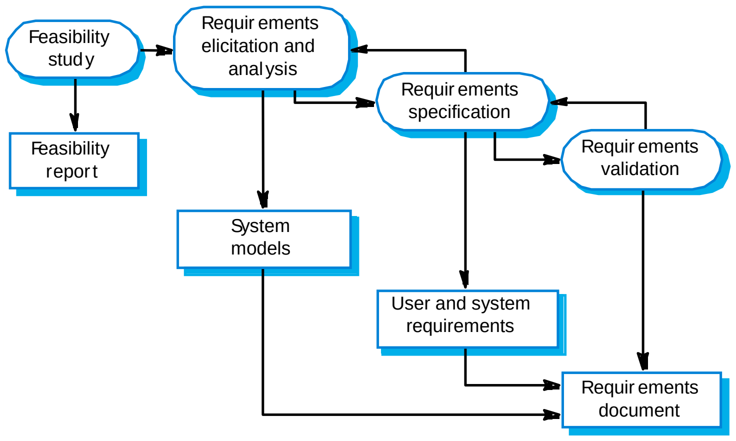
## Guidelines for writing requirements

- Use a standard format and use it for all requirements. (Better, choose an existing format)
- 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 system functions, services and constraints than user requirements.
- They are intended to be a basis for designing the system.
- They may be incorporated into the system contract.
- System requirements may be defined or illustrated using system models

## The requirements engineering process



## 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 organizational 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 organization
  - 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?
- All involved in the outcome of the system are called Stakeholders (e.g., users, developers, management, contracting organization, funding source).

## Methods for requirements elicitation

1. Interviews
2. Observation
3. Examine artifacts (e.g., prior system documents)
4. Joint Application Design sessions
5. Software tools (e.g., groupware)
6. Questionnaires
7. Prototypes (On paper, automated mockups)
8. Customer focus groups
9. Customer part of development team

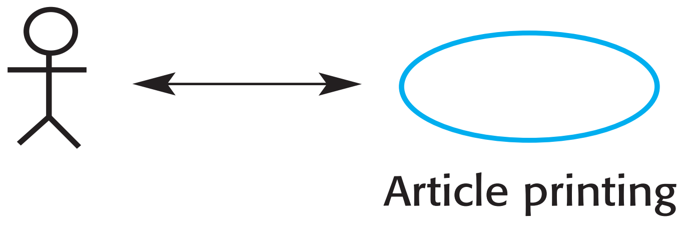
## Interviewing

- Processes:
  - Interviewing: In formal or informal interviewing, the RE team puts questions to stakeholders about the system that they use and the system to be developed.
  - Scenarios are real-life examples of how a system can be used. They should include:
    - A description of the starting situation;
    - A description of the normal flow of events;
    - A description of what can go wrong;
    - Information about other concurrent activities;
    - A description of the state when the scenario finishes.

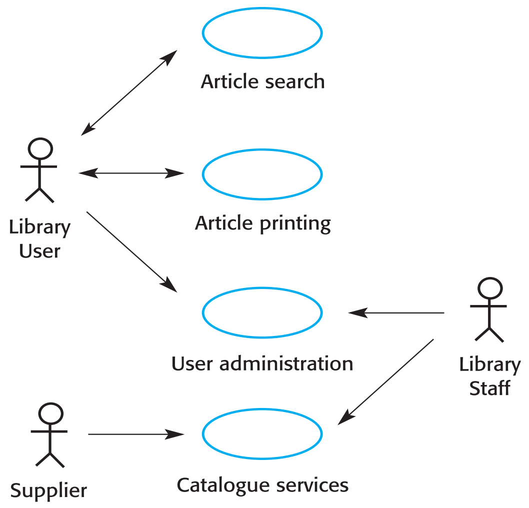
## Use cases

- 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.

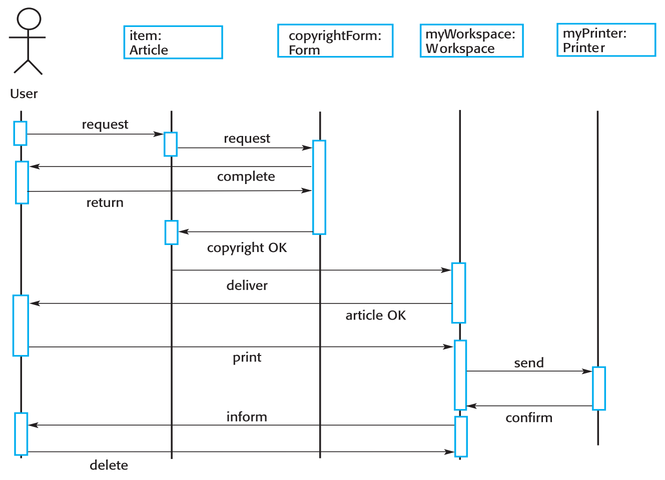
## Article printing use-case



## LIBSYS use cases



## Print article sequence



Use cases covered more fully later

## Ethnography (Observation)

- A social scientists spends a considerable time observing and analyzing how people actually work.
- People do not have to explain or articulate their work.
- Social and organizational 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

- Combine ethnography with prototyping
- Prototype development results in unanswered questions which focus the ethnographic analysis.
- The problem with ethnography is that it studies existing practices which may have some historical basis which is no longer relevant.
- Requirements that are derived from the way that people actually work rather than the way in which process definitions suggest that they ought to work.
- Requirements that are derived from cooperation and awareness of other people's activities.

## Social and organizational factors

- Software systems are used in a social and organizational context. This can influence or even dominate the system requirements.
- Social and organizational 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.

## Ten Problems of Requirements Elicitation

1. The boundary of the system is ill-defined.
2. Unnecessary design information may be given.
3. Stakeholders have incomplete understanding of their needs.
4. Stakeholders have poor understanding of computer capabilities and limitations.
5. Software engineers have poor knowledge of the problem domain.
6. Stakeholder and software engineers speak different languages.
7. “Obvious” information is omitted.
8. Different stakeholders have conflicting views.
9. Requirements are vague and untestable, such as “user friendly” and “robust”.
10. Requirements are volatile and change over time.

## Requirements and design

- In principle, requirements should state what the system should do and the design should describe how it does this.
- But, 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. This approach is not now widely used although it can be useful for interface specifications. |
| 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. Now, use-case descriptions and sequence diagrams are commonly 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

- The freedom of the requirements writer is limited by a predefined template for requirements.
- All requirements are written in a standard way.
- The terminology used in the description may be limited.
- The advantage is that the most of the expressiveness of natural language is maintained but a degree of uniformity is imposed on the specification.

## 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) of the function.

## Form-based node specification

*Insulin Pump/Control Software/SRS/3.3.2*

**Function** Compute insulin dose: Safe sugar level

**Description** Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.

**Inputs** Current sugar reading (r2), the previous two readings (r0 and r1)

**Source** Current sugar reading from sensor. Other readings from memory.

**Outputs** CompDose – the dose in insulin to be delivered

**Destination** Main control loop

**Action:** CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.

**Requires** Two previous readings so that the rate of change of sugar level can be computed.

**Pre-condition** The insulin reservoir contains at least the maximum allowed single dose of insulin..

**Post-condition** r0 is replaced by r1 then r1 is replaced by r2

**Side-effects** None

## Formal descriptions of requirements

- Indirect reference
  - Example: k equations in n unknowns
- Recurrence relations
  - Example: F(0)=1; F(1)=1; F(n+1)=F(n)+F(n-1)
- Axiomatic definition
- Expression as a language
  - Example: Backus-Naur form

## BNF requirements for a language

| | | |
|---|---|---|
| <condition> | ::= | <bool-term> \| <bool-term> or <condition> |
| <bool-term> | ::= | <bool-factor> \| <bool-factor> and <bool- term> |
| <bool-factor> | ::= | <expr> <relop> <expr> \| (<condition>) |
| <relop> | ::= | < \| ≤ \| = \| ≥ \| > \| < > |
| <expr> | ::= | <term> \| <expr> <addop> <term> \| <addop> <expr> |
| <term> | ::= | <factor> \| <term> <mpyop> <factor> |
| <factor> | ::= | <scaled-expr> \| <primary> |
| <scaled-expr> | ::= | (<expr>) <scale> \| <number> <scale> |
| <primary> | ::= | (<expr>) <regname> \| <number> \| <func> (<expr>) |
| <number> | ::= | <integer> \| <integer>. \| .<integer> \| <integer>.<integer> |
| <regname> | ::= | $ <regchar> \| <regname> <regchar> |
| <integer> | ::= | <digit> \| <digit> <integer> |
| <regchar> | ::= | <digit> \| <letter> \| <underscore> |
| <addop> | ::= | + \| - |
| <digit> | ::= | 0 \| 1 \| 2 \| 3 \| 4 \| 5 \| 6 \| 7 \| 8 \| 9 |
| <func> | ::= | abs \| trunc |
| <letter> | ::= | A \| a \| B \| b \| C \| c \| D \| d \| E \| e \| . . . \| Y \| y \| Z \| z |
| <myop> | ::= | * \| / \| mod |
| <scale> | ::= | c \| d \| h \| i \| l \| P \| p \| q \| t \| v |
| <underscore> | ::= | _ (ASCII character 95) |

## Tabular specification

- Used to supplement natural language.
- Particularly useful when you have to define a number of possible alternative courses of action.

| **Condition** | **Action** |
|---|---|
| Sugar level falling (r2 < r1) | CompDose = 0 |
| Sugar level stable (r2 = r1) | CompDose = 0 |
| Sugar level increasing and rate of increase decreasing ((r2-r1)<(r1-r0)) | CompDose = 0 |
| Sugar level increasing and rate of increase stable or increasing. ((r2-r1) □ (r1-r0)) | CompDose = round ((r2-r1)/4)<br>If rounded result = 0 then<br>CompDose = MinimumDose |

## Decision tables

Table consists of True, False, and Don't Care conditions.

| | *Rule 1* | *Rule 2* | *Rule 3* | *Rule 4* | *Rule 5* |
|---|---|---|---|---|---|
| High standardized exam scores | T | F | F | F | F |
| High grades | - | T | F | F | F |
| Outside activities | - | - | T | F | F |
| Good recommendations | - | - | - | T | F |
| Send rejection letter | | | X | X | X |
| Send admission forms | X | X | | | |

Triggers

Action

Find the first rule whose trigger is true and apply that action.

## State machines

- Functional descriptions and transition diagrams

$$f(S_i, C_j) = S_k$$

| *Current state* | *Input* | *Next state* |
|---|---|---|
| $S_1$ | 0 | $S_2$ |
| $S_1$ | 1 | $S_1$ |
| $S_2$ | 0 | $S_2$ |
| $S_2$ | 1 | $S_1$ |
| $S_3$ | 0 | $S_1$ |
| $S_3$ | 1 | $S_3$ |

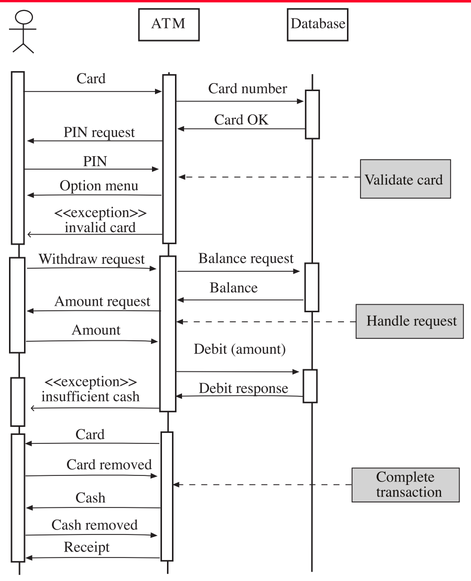
## Event tables

| *Mode* | *Event 1* | *Event 2* | *Event 3* | *Event 4* |
|---|---|---|---|---|
| Graphics | Action 1 | Action 8 | 0 | X |
| Architecture | X | Action 2 followed by Action 3 | Actions 5 and 6 in parallel | 0 |
| Native | 0 | Action 4 | Actions 1, 2 and 3 | Action 7 |

## Graphical models

- Graphical models are most useful when you need to show how state changes or where you need to describe a sequence of actions.
- Example: Sequence diagrams:
  - These show the sequence of events that take place during some user interaction with a system.
  - You read them from top to bottom to see the order of the actions that take place.
  - Cash withdrawal from an ATM
    - Validate card;
    - Handle request;
    - Complete transaction.

## Sequence diagram of ATM withdrawal



## 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.

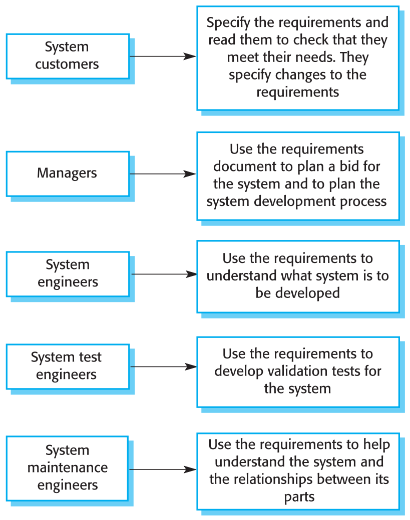
## PDL interface description

```
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
```

## The requirements document

- The requirements document is the official statement of what is required of the system developers.
- Should include both a definition of user requirements and a specification of the system requirements.
- It is NOT a design document. As far as possible, it should 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 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.

Note: This factor of 100 was once stated by Barry Boehm and has been assumed to be absolutely true, but never really validated.

## Requirements checking

- Validity. Does the system provide the functions which best support the customer's needs?
  - ❑ Consistency. Are there any requirements conflicts?
  - ❑ Traceability. Is the origin of the requirement clearly stated?
  - ❑ Verifiability. Is the requirement realistically testable?
- Completeness. Are all functions required by the customer included?
  - ❑ Comprehensibility. Is the requirement properly understood?
- Realism. Can the requirements be implemented given available budget and technology
- Adaptability. Can the requirement be changed without a large impact on other requirements?

## Requirements validation techniques

- Requirements reviews
  - Systematic manual analysis of the requirements.
- Prototyping
  - Using an executable model of the system to check requirements. Test-case generation
  - Developing tests for requirements to check testability.

## Requirements should be:

- Understandable
- Non-prescriptive
- Correct
- Complete
- Concise
- Consistent language
- Unambiguous and testable
- Traceable
- Feasible
- Ranked in importance and stability (or volatility)

## Requirements review

- 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.

## Requirements review -2

- Review goals and objections of system.
- Compare requirements with goals and objectives.
- Describe operational environment.
- Examine
  - interfaces
  - information flow
  - functions
- Check for omissions, incompleteness, inconsistency.
- Document risk.
- Discuss how system will be tested.

## 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.

## 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

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

## 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

- Requirements set out what the system should do and define constraints on its operation and implementation.
- Functional requirements set out services the system should provide.
- Non-functional requirements constrain the system being developed or the development process.
- User requirements are high-level statements of what the system should do. User requirements should be written using natural language, tables and diagrams.
- System requirements are intended to communicate the functions that the system should provide.
- Formal notation better than English to express requirements; pick notation understandable by developers and users.
- A software requirements document is an agreed statement of the system requirements.

## Key points

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