CSC 342

Software Engineering

Chapter 5:

Software Requirements

Instructor:

Dr. Ghazy Assassa

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 Process

- Requirements engineering is 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:
  - System services and constraints that are generated during the requirements engineering process

What is a requirement? (Services)

- It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification

- Requirements may serve a dual function
  - May be the basis for a Request For Proposal (RFP) / Request For Quotation (RFQ) i.e. 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)

"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."

Types of Requirement Documents -1

- User requirements
- System requirements
- Software design specification

Types of Requirement Documents -2

1. User requirements

- Written for client managers with little technical knowledge
- Statements in natural language plus diagrams & tables
- Abstract of services/functions the system provides and its operational constraints.

Types of Requirement Documents -3

2. System requirements

- Written for technical staff
- A structured document setting out detailed descriptions of the system services and requirements :
  - functional,
  - non-functional, and
  - domain
- Written as a CONTRACT between client and contractor (system provider)

Types of Requirement Documents - 4

3. Software design specification

- Written for developers
- A detailed software description which can serve as a basis for a design or implementation.
- Basis for design : Abstract description for s/w design to bridge the requirements engineering & design activities

Definitions and specifications

Requirements definition

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

Requirements specification

Basis for a design

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.

Types of Requirements

1. Business requirements
   - WHY requirements - goals of the organization requesting the system
2. Functional requirements
   - What requirements – What the product must do
3. Non-Functional requirements
   - How well requirements
4. Domain requirements
   - Application domain technical requirements
5. Inverse requirements
   - Shall not list – constraint on allowable behavior – State certain behaviour that must never occur (Easier than stating all requirement guaranteeing acceptable behavior under all circumstances) e.g. s/w security and safety requirements

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.
  - Technology specific & -ility e.g. reliability, compatibility, availability, security, recoverability, portability, scalability, maintainability, robustness, etc
- Domain requirements
  - Requirements that come from the application domain of the system and that reflect characteristics of that domain

Functional requirements (Shall list)

- Describe functionality or system services:
  - What the system should do – as shall list
  - What the system should not do – as shall not list
  - System reactions to particular input
- 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 abstract statements of what the system should do
- 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.

The Ten Characteristics of Excellent Requirements

1. Complete
2. Consistent
3. Correct
4. Feasible
5. Modifiable
6. Necessary
7. Prioritized
8. Traceable
9. Unambiguous
10. Verifiable

- In Search of Excellent Requirements.htm

Characteristics of Excellent Requirements (cont.) – 1. Complete

- No requirements or necessary information should be missing.
- While missing requirements are difficult to identify, sometimes we know we are lacking certain information.
- Use TBD ("to be determined") as a flag to highlight this gap until the necessary information is obtained.
- Requirements Verification will help maximise the completeness and consistency of the requirements document

Characteristics of Excellent Requirements (cont.) – 2. Consistent

- Internal conflicts between requirements must be resolved before development can proceed.
- You don't know which one (if any) is correct until some further research is done.
- Requirements Verification will help maximise the completeness and consistency of the requirements document

Requirements Conflict - Example

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

- Example: 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 of the above two?

Characteristics of Excellent Requirements (cont.) – 3. Correct

- Each requirement statement must accurately state a customer need.

- Only customer representatives can determine this, which is why it is essential to include customers or their surrogates in reviews of requirements documents.

Characteristics of Excellent Requirements (cont.) – 4. Feasible

- It must be possible to implement each stated requirement within the known constraints and limitations of the system and its environment.

Characteristics of Excellent Requirements (cont.) – 5. Modifiable

- To properly manage requirements, we must be able to maintain a history of changes made in each requirement.

- This requires that each requirement be uniquely labeled so that it can be referred to unambiguously.

Characteristics of Excellent Requirements (cont.) – 6. Necessary

- Each requirement should document something the customers really need.
- Avoid the temptation to gold-plate the requirements by adding features the developers are "just sure the users will love.“
- Avoid nice-to-have
- Focus on must-to-have

Characteristics of Excellent Requirements (cont.) – 7. Prioritized

- Assign an implementation priority to each requirement.
- If the requirements are all equally important, then we lose a degree of negotiation freedom if we have to respond to new requirements added during development, budget cuts, schedule overruns, or loss of project personnel.
- Consider a three-level priority scheme:
  - High: must be present in release 1.0
  - Medium: can be deferred to a subsequent release
  - Low: would be nice-to-have, but we can live without it

Characteristics of Excellent Requirements (cont.) – 8. Traceable

- Try to link each software requirement to its source (a higher-level system requirement, a use case, or a voice of the customer statement).

- We also want to be able to link each software requirement to the design elements, source code elements, and test cases that are constructed to implement and verify the requirement.

Characteristics of Excellent Requirements (cont.) – 9. Unambiguous

- Ambiguity in requirements is a serious problem.
- Team inspect of requirements: To discover ambiguity have a group of people representing different perspectives inspect the requirements as a team.
- Simply passing around the requirements document for comments is unlikely to reveal ambiguity.
  - If a requirements statement is interpreted in different ways by different reviewers, but it makes sense to each of those reviewers, the ambiguity will never surface. (Actually, it will eventually surface, but late in the project, when it can cost a great deal to correct.)

Requirements Ambiguity - Example

- 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’

  The system shall provide appropriate viewers for the user to read documents in the document store.

  - User intention - special purpose viewer for each different document type
  - Developer interpretation - Provide a text viewer (single viewer) that shows the contents of the document

Characteristics of Excellent Requirements (cont.) – 10. Verifiable/Measurable/Testable

- Examine each requirement from the perspective of whether you can devise a small number of tests, or use other verification approaches such as inspection or demonstration, to determine whether the requirement has been properly implemented.
- A good guideline for the appropriate level of detail and granularity to write requirements is to make them individually testable.

Requirements Traceability Matrix - RTM

- Displays
  - all of the functional requirements from the SRS (S/W Requirement Specification),
  - along with the design components that address each requirement,
  - the source files and procedures within them that implement the requirement,
  - and the test cases that verify the proper implementation of the requirement.

Requirements Traceability Matrix – RTM (cont.)

Go and Look for it !!!

Non-functional requirements: Properties and Constraints

Define system properties and constraints

- Properties/ attributes
  - Reliability (failure rate MTBF: mean time before failure)
  - Response time
  - Storage requirements
- Constraints
  - I/O device capability
  - System representations, etc.
- Constraint on development process:
  - Use of particular CASE system,
  - Use of particular programming language or development method

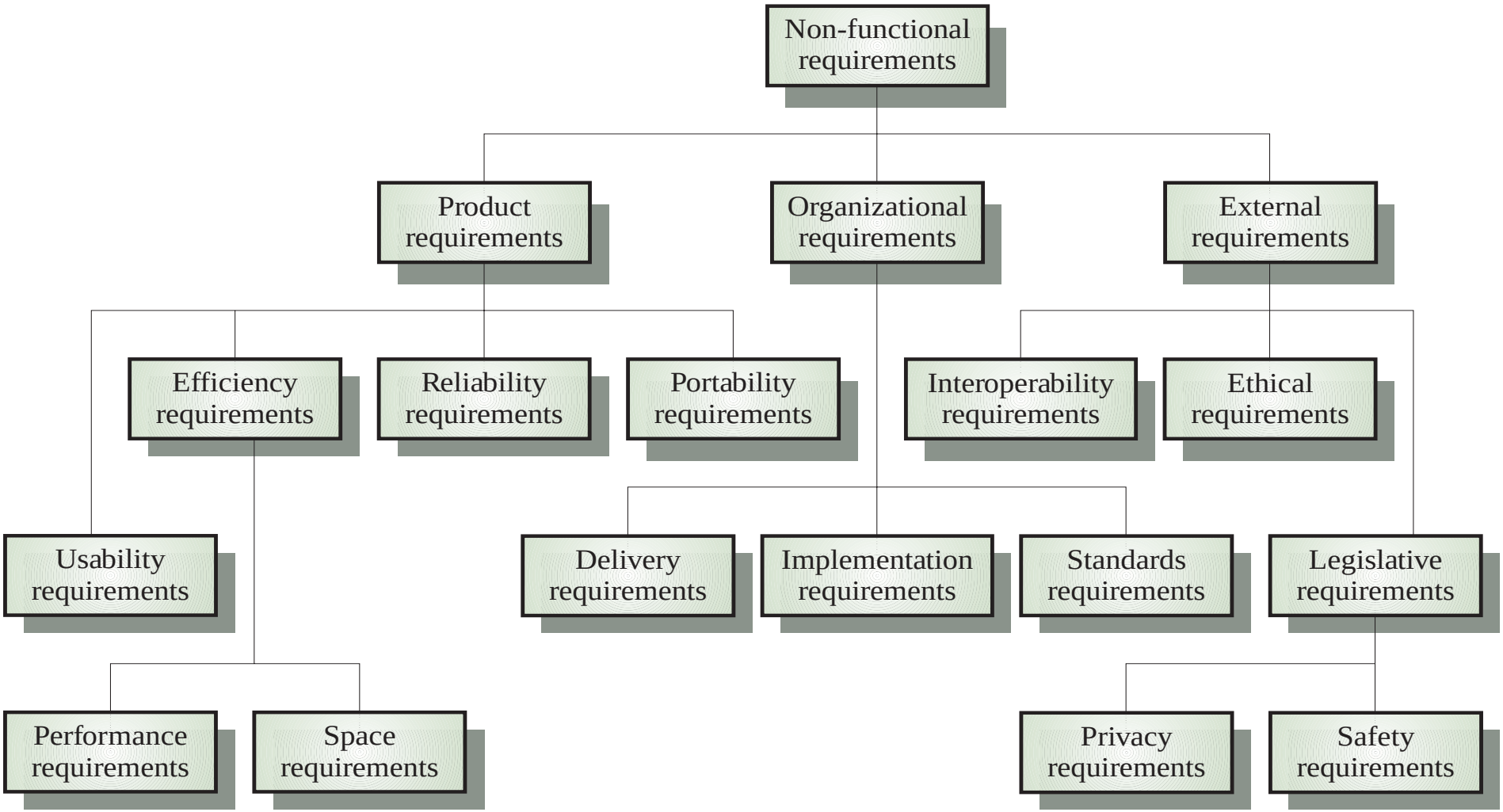
Failure to Meet Functional & Non-functional Requirements

- Failure to meet individual functional requirements
  - Degrade the system
- Failure to meet non-functional system requirements
  - Unusable system
  - Non-functional requirements may be more critical than functional requirements. If these are not met, the system is useless
  - Example 1: Performance speed requirements for a real-time control system. Failure leads to unusable system
  - Example 2: Reliability requirements for Air Traffic Control System. Failure leads to unusable system (no certification as safe system)

Non-functional classifications

- Product non-functional requirements
  - Requirements which specify that the delivered product must behave in a particular way e.g. execution speed, reliability, etc.
- Organisational non-functional requirements
  - Requirements which are a consequence of customer's and developer's organisational policies and procedures e.g. process standards used, implementation requirements (e.g. programming language), etc.
- External non-functional requirements
  - Requirements which arise from factors which are external to the system and its development process e.g.
    - Interoperability requirements (with systems in other organizations)
    - Legislative requirements: Safety and Privacy

Non-functional requirement types



Non-functional Requirements - Examples

- Product non-functional requirement
  - Requirement 4.C.8:

    It shall be possible for all necessary communication between the APSE (Ada Programming Support Environment) and the user to be expressed in the standard Ada character set

- Organisational non-functional requirement
  - 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 non-functional requirement
  - 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 Measurable Requirements

- Non-functional requirements
  - may be very difficult to state precisely
  - imprecise requirements may be difficult to verify.
- System Goal (non-verifiable/ non-measurable)
  - A general intention of the user such as ease of use
- Measurable/Verifiable non-functional requirement
  - A statement using some measure that can be objectively measured and tested against the objective
- Goals are helpful to developers as they convey the intentions of the system users
- Non-functional requirements should be verifiable/measurable by testing expressed quantitatively using metrics that can be tested

Examples: Measurable Requirements

- A system goal (non-verifiable/ non-measurable)
  - The system should be easy to use by experienced controllers
  - The system 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 Documents

- Requirements Document is a mix of:
  - Goals (Wishes):
    - difficult to measure
    - Inform client that they are not verifiable
  - Requirements

Metrics for Non-functional Requirements

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

Software Metrics & Measurements

- Software Metrics
  - Broad range of measures for computer software

- Measurements
  - Measurements of non-functional requirements are done during testing

Domain requirements

- Derived from the application domain and describe system characteristics and features that reflect the domain
- Highly technical
- May be new functional requirements, constraints on existing requirements or define specific computations
- If domain requirements are not satisfied, the system may be unworkable

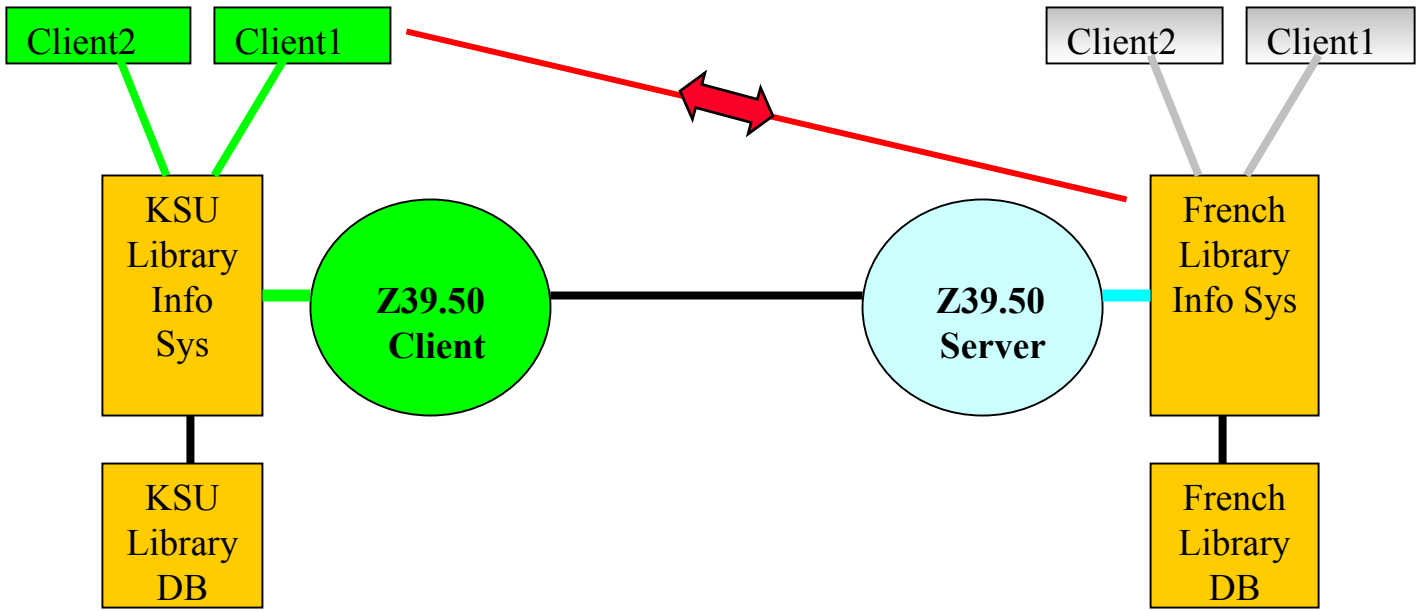
Example: Library System Domain Requirements

- There shall be a standard user interface to all databases which shall be based on the Z39.50 standard – (Constraint on sys functional requirements: User interface to DB should respect library domain standards)
- 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- (Domain copyright translates to: delete-on-print requirement).

Z39.50 Protocol for Library Information Systems

- Server Z39.50
  - Allows Server side to be accessed by other Library Information Systems (LIS) clients using their own local (different) interfaces (different from the server LIS)
- Client Z39.50
  - Allows Clients to access other LIS using clients' own local (different) interfaces (different from the server LIS)

Z39.50 Protocol (LIS)



Domain Requirement for Train protection system

- Stop the train if it goes through a red signal
- Train deceleration is computed by the system

- Domain specification states: The deceleration of the train shall be computed as:
  - $D_{train} = D_{control} + D_{gradient}$

  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.

- Domain specific terminology

Domain Requirements Problems

- Understandability
  - Requirements are expressed in the language of the application domain
  - Application domain language may not be understood by developers
- Non-explicitness
  - Domain specialists understand their area and do not make domain requirements explicit for others
- Consequences of misunderstanding
  - S/W engineers & developers may implement Domain requirements wrongly

User requirements

- Describe functional and non-functional requirements
- Written for client managers/users with little technical knowledge
- Abstract of services/functions the system provides and its operational constraints.

- User requirements are defined using
  - natural language,
  - tables
  - diagrams

Problems With Natural Language

- Lack of clarity
  - Precision is difficult without making the document wordy and difficult to read
- Requirements confusion
  - Functional, non-functional requirements and system goals tend to be mixed-up
- Requirements amalgamation
  - Several different requirements may be expressed together in a single requirement

User Requirements Example:
Database Requirement - Natural Language

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.

Red: Conceptual info (OK user requirements)
Black: Detailed info (not OK, it is System requirement)

Guidelines for Writing Requirements

- Invent your own standard format and use it for all requirements
- Use language in a consistent way:
  - Shall: mandatory requirements,
  - Should: desirable requirements
- Highlight key parts of the requirement (bold, underline, italic, …)
- Avoid the use of computer jargon (Not avoidable for domain requirements which are highly technical)

Requirements Format Structure - Example

- Requirement # and name
  - Sub-requirement # and name
    - Body text
    - Rational
    - Reference # to related system requirement

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 (Chapter 7)

Requirements and design

- Requirements should state what the system should do
- Design should describe how it does this
- In practice, some design info might be needed for detailed requirements
  - The use of a specific design may be a domain requirement
  - A system architecture may be designed to structure the requirements
  - The system may inter-operate with other systems that generate design requirements

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

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 (PDL: Program Description Language).
Graphical notations	A graphical language, supplemented by text annotations is used to define the functional requirements for the system.
Mathematical specifications	Unambiguous specifications. These are notations based on mathematical concepts. However, most customers don't understand formal specifications and are reluctant to accept it as a system contract.

Structured NL Specifications

- Imposes a degree of uniformity on specifications
- Removes some of the problems resulting from ambiguity and flexibility
- Supported using a form-based approach

Structured NL
Form-based specifications

Information to be included in forms:

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 other entities required
5. Pre and post conditions (if appropriate)
6. The side effects (if any) of the operation

Structured NL: Form-based node specification Example

ECLIPSE/Workstation/Tools/DE/FS/3.5.1

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

- PDL: Program Description 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, e.g. nested conditional loops
  - When hardware and software interfaces have to be specified
- Advantage:
  - May be checked (syntax & semantics) by s/w tools

PDL-based requirements definition (cont.)

- Disadvantages of PDL:
  - Difficult for non specialist
  - The PDL may not be sufficiently expressive to define domain concepts
  - Yields very detailed specs .. taken as design/implementation rather than requirements specification describing the system

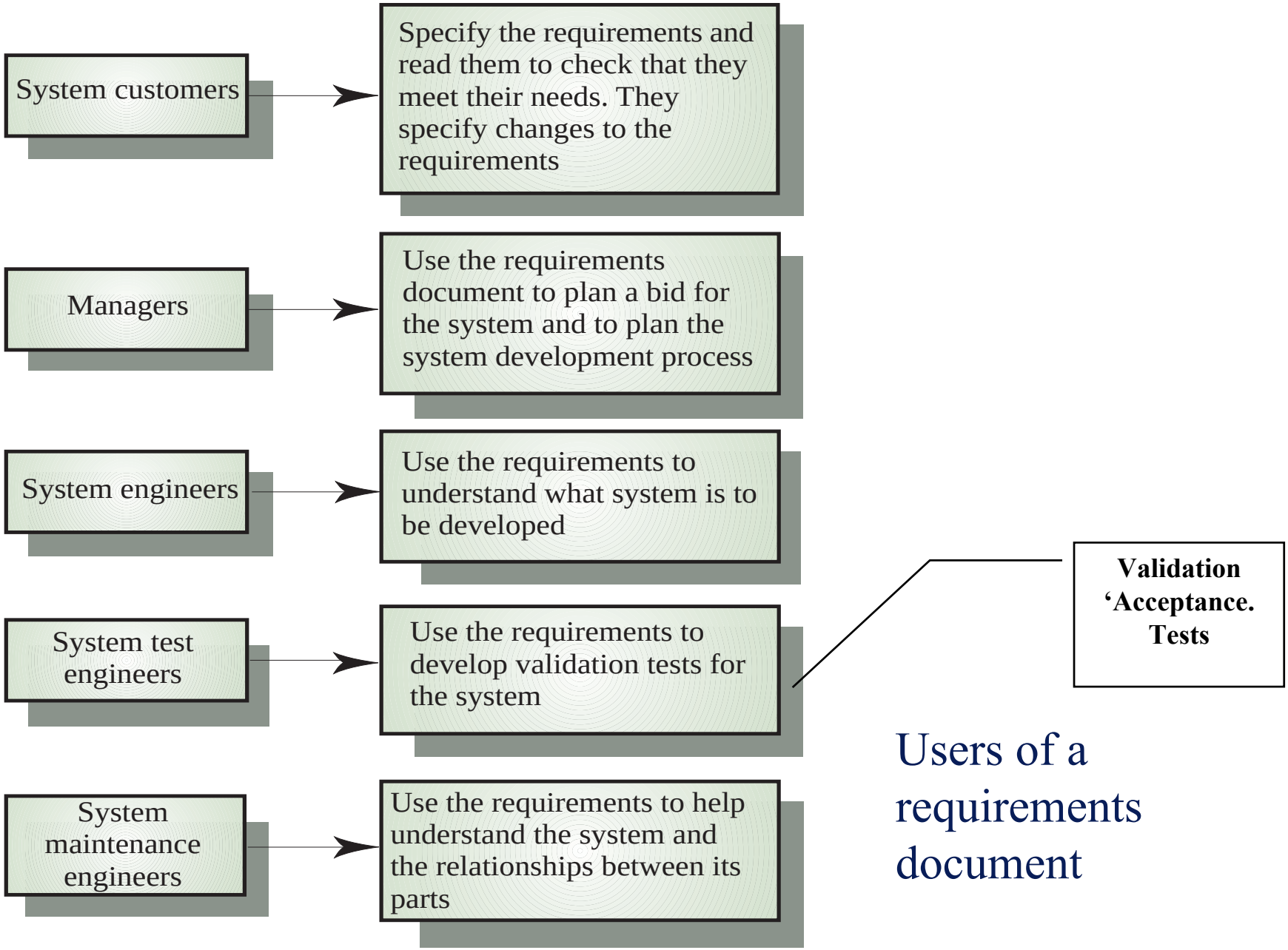
- Best solution: Hybrid
  - Structured NL form-based for the overall system
  - + PDL (Program Description Language) to define control sequences

Interface Specification (with other systems)

- Most systems must operate with other systems
  - Operating interfaces must be specified as part of the requirements
- Three types of interface between the system and other systems:
  - Procedural interfaces
    - Other existing sub-systems offer a range of services accessible by calling interface procedures
  - Data structures that are exchanged between sub-systems
    - (specify using PDL or ER diagrams)
  - Data representations (e.g. ordering of bits)
- Formal notations are an effective technique for interface specification

The Software Requirements Document

- Official statement of WHAT is required of the system developers
- Should include both
  - definition of requirements
  - and a specification of requirements
- It is NOT a design document.
- As far as possible, it should set of WHAT the system should do rather than HOW it should do it


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

Requirements Document Requirements

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

IEEE Standard for Requirements Document - Generic Structure

1. Introduction
2. General description
3. Specific requirements
4. Appendices
5. Index

- IEEE standard is not rigid 'ideal' - can be adapted for specific systems

Requirements Document - Adapted Structure

- Introduction
  - need for the system, abstract functions, how the sys will interact with other existing systems,..
- Glossary (technical terms)
- User requirements definition
  - Abstract: Business, functional, non-functional, domain, inverse req
- System architecture
  - assignment of functions to modules, reused components
- System requirements specification
  - Details: Business, functional, non-functional (Interface to other systems), domain, inverse req..
- System models (process, info/data, object,..)
- System evolution
- Appendices
- Index (for big document)

IEEE Standard for Requirements Document – Detailed Structure

1. Introduction
   - 1.1 Purpose (of the document)
   - 1.2 Scope (of the product)
   - 1.3 Definitions, Acronyms, and Abbreviations
   - 1.4 References
   - 1.5 Overview

IEEE Standard for Requirements Document – Detailed Structure (cont.)

2. General Description
   - 2.1 Product Perspective
   - 2.2 Product Functions
   - 2.3 User Characteristics
   - 2.4 General Constraints
   - 2.5 Assumptions and Dependencies

IEEE Standard for Requirements Document – Detailed Structure (cont.)

3. Specific Requirements (functional, non-functional, domain)

3.1 External Interface Requirements (interoperability)

Provide a detailed description of all inputs into and outputs from the software

3.1.1 User Interfaces

3.1.2 Hardware Interfaces

3.1.3 Software Interfaces

3.1.4 Communications Interfaces

3.2 System Features (Functions) ***"Bulk of requirements document"***

3.2.1 System Feature 1

3.2.1.1 Introduction/Purpose of Feature

3.2.1.2 Stimulus/Response Sequence

3.2.1.3 Associated Functional Requirements (if any)

3.2.1.3.1 Functional Requirement 1, etc.

3.2.2 System Feature 2

3.2.3 System Feature 3, etc.

IEEE Standard for Requirements Document – Detailed Structure (cont.)

3.3 **Performance Requirements (Non-functional requirements)**

3.4 Design/Implementation Constraints, if any (use of CASE or particular programming language,..)

3.5 Quality Attributes

3.6 Other Requirements (DB, other emergent system properties)

3.1 External Interface Requirements (interoperability)

- Provide a detailed description of all inputs into and outputs from the software.
- Include both content and format as follows
  - *name of item*
  - *description of purpose*
  - *source of input or destination of output*
  - *valid range, accuracy, and/or tolerance*
  - *units of measure*
  - *timing*
  - *relationships to other inputs/outputs*
  - *screen formats/organization*
  - *window formats/organization*
  - *data formats*
  - *command formats*
  - end message

Software Requirements Specification Template

[Project]

Software Requirements Specification

CxTemp_Software Requirements Specification.doc

Draft X

June 27, 2001

[Organization ABC]

[Paste Your Organization's Logo Here]

- C:\Software_Eng\SW Resources\CxTemp_SoftwareRequirementsSpecification.doc

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