CSCI222 System Development

Requirements

Outline

- What is a requirement?
- Functional vs. non-functional requirements
- Requirements Engineering activities

What is a Requirement?

A requirement is:

An informal (and/or formal) description or statement of a function, feature or condition that a user seeks to have implemented in a system.

There are 2 broad categories of requirements:

- Functional Requirements:
  - are those that relate directly to the functioning of the system.
- Non-functional requirements
  - cover aspects of the system such as user interface, performance, quality issues, interfaces to other systems, security etc.

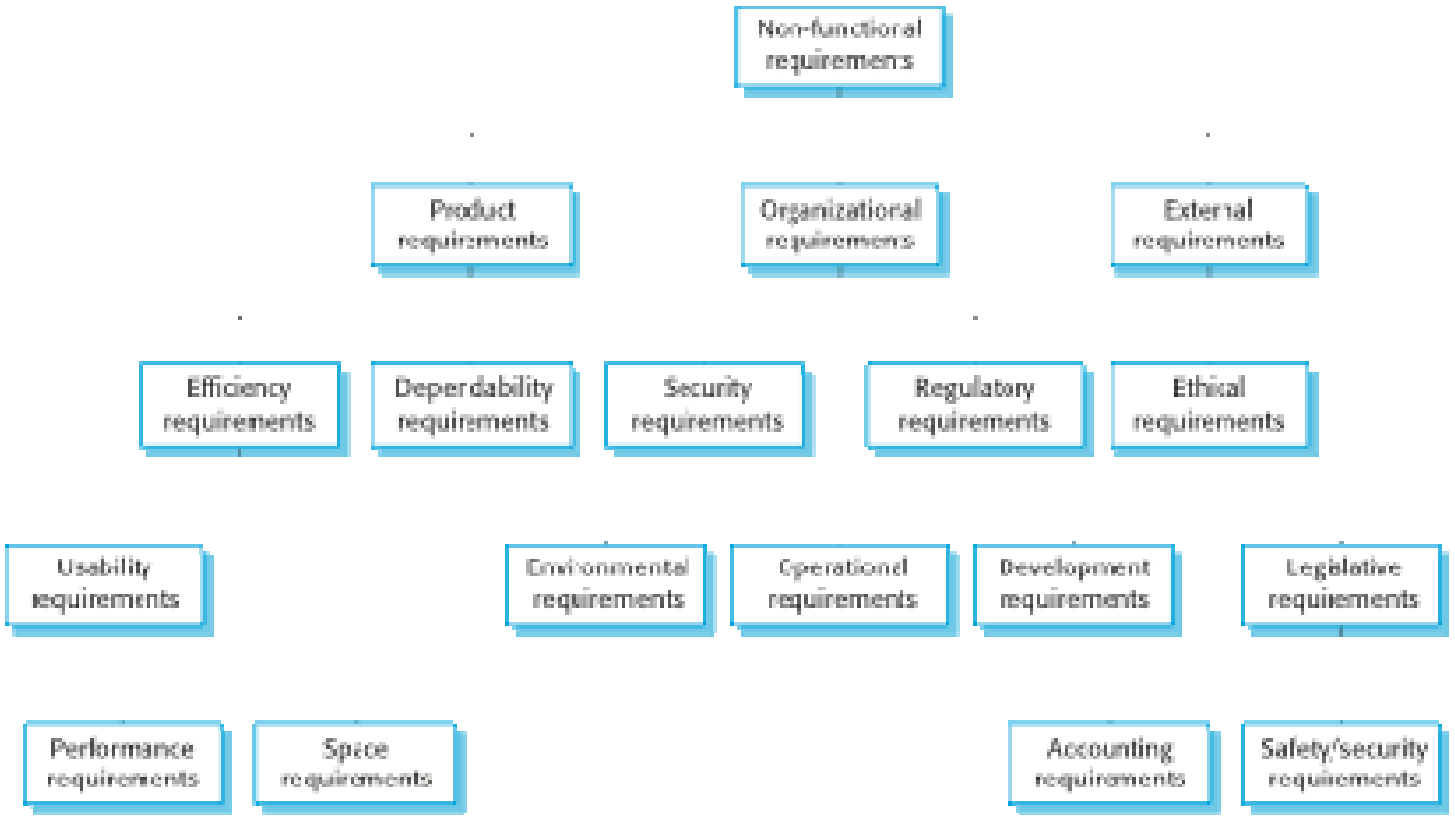
Functional requirements

- These specify what the system will do
  - "The system should authenticate users before allowing them access."
  - "The system should allow a user to check their bank account balance."
  - A user shall be able to search the appointments lists for all clinics.
  - The system shall generate each day, for each clinic, a list of patients who are expected to attend appointments that day.
  - Each staff member using the system shall be uniquely identified by his or her 8-digit employee number.

Non-functional requirements

- These are requirements not directly related to system functions for example performance requirements
- These are often very badly written in an ambiguous and totally untestable fashion e.g. “The system should be user friendly”
- We need to make them MEASURABLE

Types of nonfunctional requirement



Examples of nonfunctional requirements

Product requirement

The MHC-PMS shall be available to all clinics during normal working hours (Mon–Fri, 0830–17.30). Downtime within normal working hours shall not exceed five seconds in any one day.

Organizational requirement

Users of the MHC-PMS system shall authenticate themselves using their health authority identity card.

External requirement

The system shall implement patient privacy provisions as set out in HStan-03-2006-priv.

Defining non-functional requirements

Here is a checklist of measurable properties.

Property	**Metric**
Speed	Processed transactions/sec
	User/Event response time
	Screen refresh time
Size	Kbytes
	Memory
Ease of use	Training Time
	Number of help frames
Reliability	Mean time to failure
	Probability of unavailability
	Rate of failure occurrence
	Availability
	Time to restart after failure
	% of events causing failure
Portability	No. of target systems

Some other examples of non-functional requirements

- "The system should be easy to use by experienced controllers and should be organised in such a way that user errors are minimised”

  = poor specification - not able to be tested

- "Experienced controllers should be able to use all of the system function after a total of 2 hours training. After this training, the average number of errors made by experienced users should not exceed 2 per day”

  = GOOD specification - able to be tested.

Requirements Engineering Activities

- **Requirements Elicitation**
  - Gathering requirements(?)
- **Requirements Analysis**
  - Categorizing and organizing requirements(?)
- **Requirements Specification**
  - Documenting requirements(?)
- **Requirements Validation & Verification**
  - Making sure the requirements are correct and complete
- **Requirements Management**
  - Looking after requirements throughout the project

Requirements Validation

- Are the requirements correct?
  - Do they accurately reflect what the client wants?
- Are they complete?
  - Look for holes, assumptions

Requirements Verification

- Are they consistent?
  - Is there any conflict?
- Are they testable?
  - Is it possible to tell if the requirement has been achieved?
  - There is no point having a requirement if it won't be possible to tell if it has been achieved

Managing requirements

- Dropping requirements
- Adding new requirements
- Adjusting priorities

- Often require negotiation
- May require budget/schedule adjustment
- May be a result of budget/schedule adjustment

Requirements Elicitation - Outline

- What is requirements elicitation?
- Why requirements elicitation is difficult?
- Stakeholder identification
- Requirements elicitation techniques
- Evaluating requirements

Acknowledgement: some materials are from Chapter 4 - Ian Sommerville (2010), *Software Engineering,* 9th Edition, Addison-Wesley.

What is requirements elicitation?

- Requirements elicitation is the practice of obtaining the requirements of a system from users, customers and other stakeholders. (*)

- Requirements elicitation is also sometimes referred to as *requirements gathering*.

(*) Source: Requirements Engineering A good practice guide, Ian Sommerville and Pete Sawyer, John Wiley and Sons, 1997

Why requirements elicitation is difficult?

- Problems of Scope
  - The boundary of the system is usually ill-defined.
- Problems of understanding
  - Customers don’t really know what they need or can’t express it.
  - Poor understanding of the problem domain
- Problems of volatility
    - The requirements keep changing!

Stakeholder identification – the very first step

- Stakeholders (SH) are persons or organizations. They may be affected by it either directly or indirectly.

- Stakeholders may include:
  - anyone who operates the system
  - anyone who benefits from the system
  - anyone involved in purchasing or procuring the system.
  - organizations which regulate aspects of the system
  - people or organizations opposed to the system
  - organizations responsible for systems which interface with the system under design

Example
A patient information system for mental health care

- A MHC-PMS (Mental Health Care-Patient Management System) to support mental health care is a medical information system that maintains information about patients suffering from mental health problems and the treatments that they have received.
- It makes use of a centralized database of patient information but has also been designed to run on a PC, so that it may be accessed and used from sites that do not have secure network connectivity.

MHC-PMS key features

- Individual care management
  - Clinicians can create records for patients, edit the information in the system, view patient history, etc. The system supports data summaries so that doctors can quickly learn about the key problems and treatments that have been prescribed.
- Patient monitoring
  - The system monitors the records of patients that are involved in treatment and issues warnings if possible problems are detected.
- Administrative reporting
  - The system generates monthly management reports showing the number of patients treated at each clinic, the number of patients who have entered and left the care system, number of patients sectioned, the drugs prescribed and their costs, etc.

Stakeholders in the MHC-PMS

- Patients whose information is recorded in the system.
- Doctors who are responsible for assessing and treating patients.
- Nurses who coordinate the consultations with doctors and administer some treatments.
- Medical receptionists who manage patients' appointments.
- IT staff who are responsible for installing and maintaining the system.
- A medical ethics manager who must ensure that the system meets current ethical guidelines for patient care.
- Health care managers who obtain management information from the system.
- Medical records staff who are responsible for ensuring that system information can be maintained and preserved, and that record keeping procedures have been properly implemented.
- …

Elicitation Techniques

- A large part of requirements elicitation is concerned with finding information.
- A number of techniques are suggested:
  - Documents
  - Interviews
  - Scenarios (e.g. Use cases)
  - Meetings
  - Questionnaires
  - Observations
  - Prototypes

Background Reading

(Adapted from Easterbrook, 2006)

- Sources of information:
  - company reports, organization charts, policy manuals, job descriptions, reports, documentation of existing systems, etc.
- Advantages:
  - Helps the analyst to get an understanding of the organization before meeting the people who work there.
  - Helps to prepare for other types of fact finding
    - e.g. by being aware of the business objectives of the organization.
  - may provide detailed requirements for the current system.
- Disadvantages:
  - written documents often do not match up to reality.
  - Can be long-winded with much irrelevant detail
- Appropriate for
  - Whenever you not familiar with the organization being investigated.

"Hard Data" and Sampling

(Adapted from Easterbrook, 2006)

- Hard data includes facts and figures…
  - Forms, Invoices, financial information,…
  - Reports used for decision making,…
  - Survey results, marketing data,…

- Sampling
  - Decide what data should be collected - e.g. banking transactions
  - Determine the population - e.g. all transactions at 5 branches over one week
  - Choose type of sample - e.g. simple random sampling
  - Choose sample size - e.g. every 20th transaction

Example of hard data

(Adapted from Easterbrook, 2006)

- Questions:
  - What does this data tell you?
  - What would you do with this data?

Agate

Campaign Summary

Date 23rd February 1999

Client Yellow Partridge
Park Road Workshops
Park Road
Jewellery Quarter
Birmingham B2 3DT
U.K.

Campaign Spring Collection 1999

Billing Currency GB £

Item	Curr	Amount	Rate	Billing amount
Advert preparation: photography, artwork, layout etc.	GB £	15,000.00	1	15,000.00
Placement French Vogue	FFr.	47 000,00	11.35	4,140.97
Placement UK Vogue	GB £	5,000.00	1	5,000.00
Placement US Vogue	US $	15,000.00	2.47	6.072.87
Total				30,213.84

This is not a VAT Invoice. A detailed VAT Invoice will be provided separately.

210-212 Carstairs Street, Birmingham B1 5TG
Tel.0121-111-1234 Fax.0121-111-1245
Email.agate@agateltd.co.uk

Interview

- Formal or informal interviews with stakeholders are part of most RE processes.
- Types of interview
  - Closed interviews based on pre-determined list of questions
  - Open interviews where various issues are explored with stakeholders.

Interviews in practice

- Normally a mix of closed and open-ended interviewing.
- Interviews are good for getting an overall understanding of what stakeholders do and how they might interact with the system.
- Interviews are not good for understanding domain requirements
  - Requirements engineers cannot understand specific domain terminology;
  - Some domain knowledge is so familiar that people find it hard to articulate or think that it isn't worth articulating.

Scenarios

- Scenarios are real-life examples of how a system can be used.
  - a means for providing context to the elicitation of user requirements.
  - allow the software engineer to provide a framework for questions about user tasks by permitting "what if" and "how is this done" questions to be asked.
  - The most common type of scenario is the use case.

- 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.

Scenario for collecting medical history in MHC-PMS

Initial assumption: The patient has seen a medical receptionist who has created a record in the system and collected the patient's personal information (name, address, age, etc.). A nurse is logged on to the system and is collecting medical history.

Normal: The nurse searches for the patient by family name. If there is more than one patient with the same surname, the given name (first name in English) and date of birth are used to identify the patient.

The nurse chooses the menu option to add medical history.

The nurse then follows a series of prompts from the system to enter information about consultations elsewhere on mental health problems (free text input), existing medical conditions (nurse selects conditions from menu), medication currently taken (selected from menu), allergies (free text), and home life (form).

Scenario for collecting medical history in MHC-PMS

What can go wrong: The patient's record does not exist or cannot be found. The nurse should create a new record and record personal information.

Patient conditions or medication are not entered in the menu. The nurse should choose the 'other' option and enter free text describing the condition/medication.

Patient cannot/will not provide information on medical history. The nurse should enter free text recording the patient's inability/unwillingness to provide information. The system should print the standard exclusion form stating that the lack of information may mean that treatment will be limited or delayed. This should be signed and handed to the patient.

Other activities: Record may be consulted but not edited by other staff while information is being entered.

System state on completion: User is logged on. The patient record including medical history is entered in the database, a record is added to the system log showing the start and end time of the session and the nurse involved.

Evaluating Requirements

All requirements are not equal and may be ranked by different methods.

For example:

- Requirements that absolutely MUST be met (essential)
- Requirements that are highly desirable but not necessary (desirable)
- Requirements that are possible but could be eliminated (wishful thinking)

OR:

- Ensure that the value of each requirement to each group of stakeholders has been considered.
- Each stakeholder might be asked to rank each requirement on a scale of 1 - 5 for
  - (a) the impact on the system if it is included and
  - (b) the impact on the system if it is not included.

Quiz: true or false?

- Which of the following are good requirements?
  - A. The system should be reliable.
  - B. The system should use the Model-View-Controller architecture.
  - C. The system should allow the user to check their bank account balance.
  - D. The system should solve all the management problems for the university.

Requirements Specification

- Software Requirements Specification
- Why do we need to write specification
- Ways of writing a system requirements specification
- Contents of the SRS
  - What should SRS not include?
- Desirable properties of an SRS

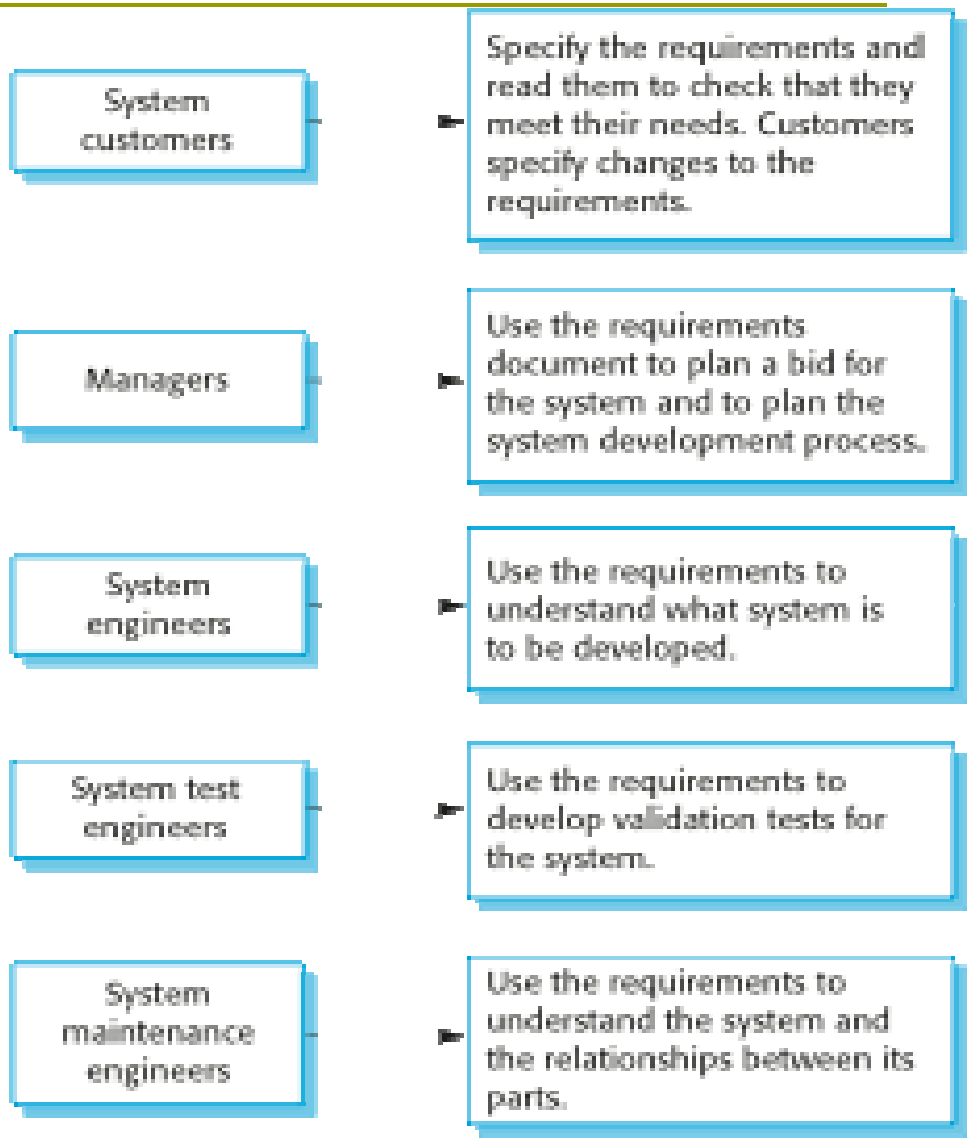
Software Requirements Specification

- SRS both as a document and the process!
  - Specification is a noun and verb

- SRS as a document:
  - Is a specification for a particular software product, program, or set of programs that performs certain functions in a specific environment.
  - is produced at the end of the Requirements Elicitation and Analysis stage (or as a separate stage). This document is a *major milestone* in the software development process.

Software Requirements Specification
Why do we need to write specifications?

- Establish the basis for agreement between the customers and the suppliers on what the software product is to do.
- Users to determine if the software specified meets their needs.
- Provide a basis for estimating costs and schedules.
- Provide a baseline for validation and verification.
- Serve as a basis for enhancement.



Ways of writing a system requirements specification

Notation	Description
Natural language	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract

Natural language specification

- Requirements are written as natural language sentences supplemented by diagrams and tables.
- Used for writing requirements because it is expressive, intuitive and universal. This means that the requirements can be understood by users and customers.

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.

Example requirements for the insulin pump software system

3.2 The system shall measure the blood sugar and deliver insulin, if required, every 10 minutes. *(Changes in blood sugar are relatively slow so more frequent measurement is unnecessary; less frequent measurement could lead to unnecessarily high sugar levels.)*

3.6 The system shall run a self-test routine every minute with the conditions to be tested and the associated actions defined in Table 1. *(A self-test routine can discover hardware and software problems and alert the user to the fact the normal operation may be impossible.)*

Note: A personal insulin pump is an embedded software system in an insulin pump used by diabetics to maintain blood glucose control.

Structured specifications

- An approach to writing requirements where the freedom of the requirements writer is limited and requirements are written in a standard way.
- This works well for some types of requirements e.g. requirements for embedded control system but is sometimes too rigid for writing business system requirements.

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.
- Information about the information needed for the computation and other entities used.
- Description of the action to be taken.
- Pre and post conditions (if appropriate).
- The side effects (if any) of the function.

A structured specification of a requirement for an insulin pump

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.

A structured specification of a requirement for an insulin pump

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.	
Requirements	
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.

Tabular specification

- Used to supplement natural language.
- Particularly useful when you have to define a number of possible alternative courses of action.
- For example, the insulin pump systems bases its computations on the rate of change of blood sugar level and the tabular specification explains how to calculate the insulin requirement for different scenarios.

Tabular specification of computation for an insulin pump

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) If rounded result = 0 then CompDose = MinimumDose

This slide is not examinable

Mathematical/formal specification

Library

$$held : \mathbb{P}\, Books$$
$$borrowers : \mathbb{P}\, People$$
$$onloan : Books \pfun People$$

$$\mathrm{dom}\ onloan \subseteq held$$
$$\mathrm{ran}\ onloan \subseteq borrowers$$
$$\forall\, p : People \bullet \#(onloan \rhd \{p\}) \leqslant max$$

(Example using Z)

PurchaseBook

$$\Delta Library$$
$$b? : Books$$

$$b? \notin held$$
$$held' = held \cup \{b?\}$$
$$borrowers' = borrowers$$
$$onloan' = onloan$$

Flashback quiz

- Which of the following is not true about the waterfall model?
  A. a phase has to be *complete* and absolutely *correct* before moving onto the next phase.
  B. being difficult to respond to requirement changes.
  C. develop a system through repeated cycles and in smaller portions at a time.
  D. designer will have to fully predict problem areas of the system.

Contents of the Software Requirements Specification

- Introduction
- Current System (brief)
- Proposed System (functional and non-functional requirements)
- Systems models (use case, class, sequence, and state diagrams and user interface screens)
- Validation Criteria
- Bibliography
- Appendices

Contents of the Software Requirements Specification (cont.)

- Software Requirements Specification should address:
  - Functionality.
    - What is the software supposed to do?
  - External interfaces.
    - How does the software interact with people, the system's hardware, other hardware, and other software?
    - What assumptions can be made about these external entities?
  - Required Performance.
    - What is the speed, availability, response time, recovery time of various software functions, and so on?
  - Quality Attributes.
    - What are the portability, correctness, maintainability, security, and other considerations?

SRS: Constraints

- The constraints are to be considered along with other requirements
  - **Design constraints**
    - Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) and so on?
    - Limits on the design; e.g. requiring a relational database stipulates the approach that we take in developing the system.
  - **Implementation constraints**
    - Limits on coding or construction; e.g. use C#, use IIS
  - **Interface constraints**
    - A requirement to interact with an external item; often describing communication protocols that must be supported
  - **Physical constraints**
    - Constraints on the physical hardware used for the system (requirements for power consumption, cooling etc)

SRS should not include…

Source: Adapted from Davis, 1990, p183

- Project development plans
  - E.g. cost, staffing, schedules, methods, tools, etc
    - Lifetime of SRS is until the software is made obsolete
    - Lifetime of development plans is much shorter
- Product assurance plans
  - Configuration Management, Verification & Validation, test plans, Quality Assurance, etc
    - Different audiences
    - Different lifetimes
- Designs
  - Requirements and designs have different audiences
  - Analysis and design are different areas of expertise
    - I.e. requirements analysts shouldn't do design!
  - Except where application domain constrains the design
    - e.g. limited communication between different subsystems for security reasons.
- Implementation
  - Should NOT commit the developer to any particular platform or programming language.

Desirable properties of an SRS

(adapted from IEEE Std 830)

- Unambiguous
  - Every statement can be read in exactly one way.
  - The software development team will be unable to produce a product that satisfies users' needs if one or more requirements can be interpreted in multiple ways.
- Complete
  - All the things the system must do …
  - … and all the things it must not do.
  - Conceptual completeness
    - E.g. responses to all classes of input
  - Structural completeness
    - E.g. no TBDs
- Understandable (clear)
  - E.g. by non-technical specialists
- Ranked
  - Indicates relative importance/stability of each requirement

Desirable properties of an SRS (cont.)

(adapted from IEEE Std 830)

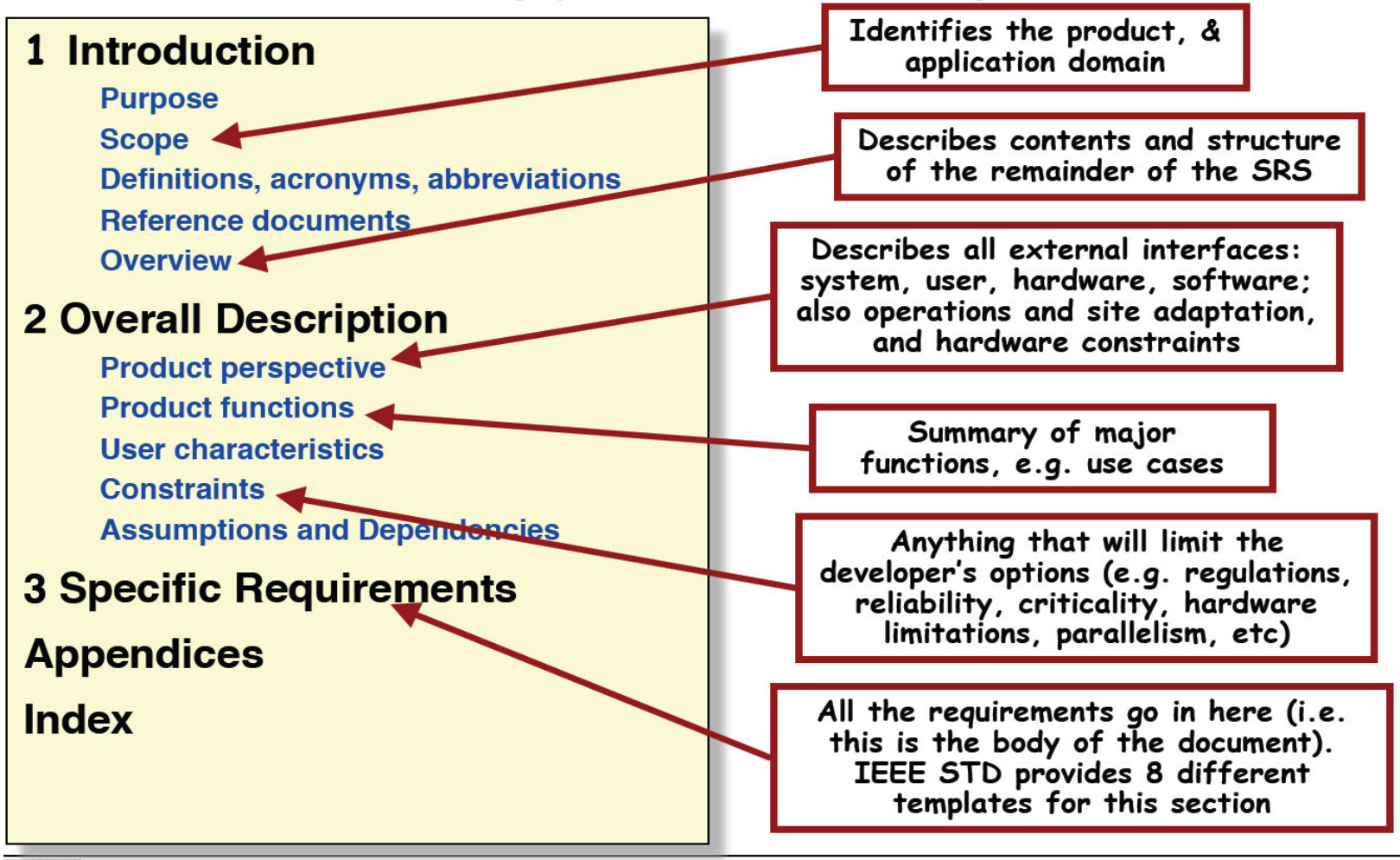
- Verifiable
  - A process exists to test satisfaction of each requirement
- Consistent
  - Doesn't contradict itself
  - Use all terms consistently
  - You cannot build a system that satisfies all requirements if two requirements conflict
- Modifiable
  - Can be changed without difficulty
    - Good structure and cross-referencing
- Traceable
  - Origin of each requirement is clear
  - Labels each requirement for future referencing
  - The team should track the source of each requirement, e.g. whether it evolved from a specific meeting with a target user.

Guidelines for writing requirements

- Invent a standard format and use it for all requirements.
  - Many standard formats, eg ANSI/IEEE Std 830, DoD 2167A, AS3563, Rational Unified Process
- 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.
- Include an explanation (rationale) of why a requirement is necessary.

IEEE Standard for SRS

*Source: Adapted from IEEE-STD-830-1993 **See also**, Blum 1992, p160*



IEEE Standard for SRS

3.1 External Interface Requirements

- 3.1.1 User Interfaces
- 3.1.2 Hardware Interfaces
- 3.1.3 Software Interfaces
- 3.1.4 Communication Interfaces

3.2 Functional Requirements

this section organized by mode, user class, feature, etc. For example:

- 3.2.1 Mode 1
  - *3.2.1.1 Functional Requirement 1.1*
  - …
- 3.2.2 Mode 2
  - *3.2.1.1 Functional Requirement 1.1*
  - …
- ...
- 3.2.2 Mode n
  - ...

3.3 Performance Requirements

Remember to state this in measurable terms!

3.4 Design Constraints

3.4.1 Standards compliance

3.4.2 Hardware limitations

etc.

3.5 Software System Attributes

- 3.5.1 Reliability
- 3.5.2 Availability
- 3.5.3 Security
- 3.5.4 Maintainability
- 3.5.5 Portability

3.6 Other Requirements

Cross-referencing requirements

- In simple systems, requirements can be almost independent
  - E.g. the requirements for a web site that has "find courses", "enroll in course", "view library books" operations are essentially independent;
- In more complex systems, you are likely to find relations between different requirements
- It may be useful to create a cross-reference matrix illustrating interdependencies.

	R1	R2	R3	R4	R5
R1					
R2	x				
R3					
R4		x	x		
R5			x		

Review

- Functional vs. non-functional requirements
- How to write a good requirements
- Techniques for eliciting requirements
- Properties of a good SRS

- Want to learn more about Requirements Engineering
  - CSCI450/928 Software Engineering Requirements and Specification