So Many "Requirements"… (1)

- A goal is an objective or concern that guides the RE process. It can be used to discover and evaluate functional and non-functional requirements
  - A goal is not yet a requirement…
- Note: All requirements must be verifiable (by some test, inspection, audit etc.)
- A functional requirement is a requirement defining functions of the system under development
  - Describes what the system should do
- A non-functional requirement is a requirement that is not functional. This includes many different kinds of requirements. – Therefore one often considers the following sub-categories:

16

Different types of non-functional requirements

- Performance requirements, characterizing system properties such as expected performance, capacity, reliability, robustness, usability, etc.
- Design constraints (also called process requirements), providing constraints on how the system should be designed and built – related to development process, documentation, programming language, maintainability, etc.
- Commercial constraints, such as development time frame and costs.

17

So Many "Requirements"… (2)

- A user requirement is a desired goal or function that a user and other stakeholders expect the system to achieve
  - Does not necessarily become a system requirement
- Application domain requirement (sometimes called business rules) are requirements derived from business practices within a given industrial sector, or in a given company, or defined by government regulations or standards.
  - May lead to system requirements. Can be functional or non-functional
- Problem domain requirements should be satisfied within the problem domain in order to satisfy some of the goals
- System requirements are the requirements for the system to be built, as a whole
  - A system is a collection of interrelated components working together towards some common objective (may be software, mechanical, electrical and electronic hardware and be operated by people)
  - Systems Engineering is a multidisciplinary approach to systems development – software is only a part (but often the problematic part)

18

So Many "Requirements"… (3)

- Important note: Software Requirements Engineering is a special case of Requirements Engineering. Many topics discussed in this course are quite general and apply to requirements engineering, in general.
- In a system containing software, software requirements are derived from the system requirements. The system then consists of hardware and software, and the hardware (and often the operating system and other existing software modules) are part of the environment in which the software is used.

19

Functional Requirements

- What inputs the system should accept
- What outputs the system should produce
- What data the system should store other systems might use
- What computations the system should perform
- The timing and synchronization of the above

- 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

20

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.

Note: not all requirements on this and following slides are high quality requirements but are typical requirements found too often in documents

21

Next Lecture

Non-Functional Requirements (NFR) (1)

- Non-functional requirements are important
  - If they are not met, the system is useless
  - Non-functional requirements may be very difficult to state precisely (especially at the beginning) and imprecise requirements may be difficult to verify
- They are sometimes called quality requirements, quality of service, or extra-functional requirements.
- *Three main categories [1]:*
  - **Performance requirements** reflecting: usability, efficiency, reliability, maintainability and reusability (note: also security requirements)
    - Response time, throughput
    - Resource usage
    - Reliability, availability
    - Recovery from failure
    - Allowances for maintainability and enhancement
    - Allowances for reusability

[1] Lethbridge and Laganière, Object Oriented Software Engineering: Practical Software Development using UML and Java, 2005

Non-Functional Requirements (NFR) (2)

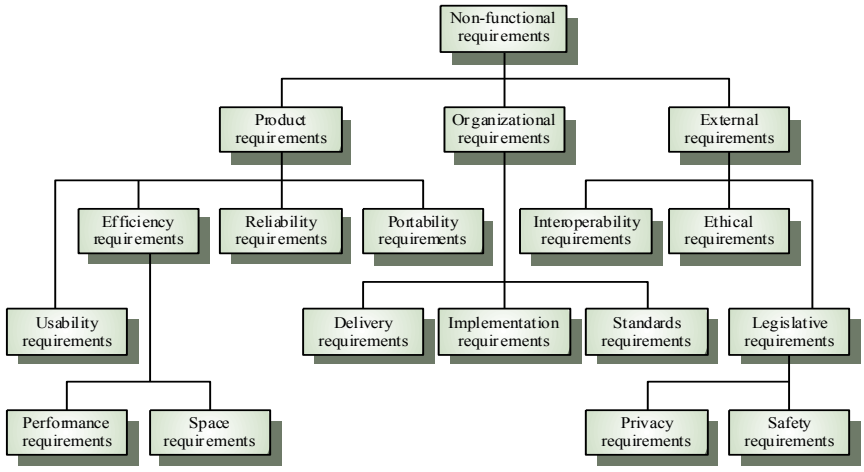
- **Design constraints:** Categories constraining the environment and technology of the system.
  - Platform (minimal requirements, OS, devices…)
  - Technology to be used (language, DB, …)

- **Commercial constaints:** Categories constraining the project plan and development methods
  - Development process (methodology) to be used
  - Cost and delivery date
    - Often put in contract or project plan instead

24

Various NFR Types

- *Other ontologies also exist*



Source: Gerald Kotonya and Ian Sommerville, Requirements Engineering – Processes and Techniques, Wiley, 1998

25

Measurable 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

Source: Gerald Kotonya and Ian Sommerville, Requirements Engineering – Processes and Techniques, Wiley, 1998

26

Goals

- A Goal
  - Conveys the intention or the objective of one or many stakeholders
  - Can guide the discovery of verifiable non-functional requirements that can be tested objectively

27

Example of Goal and NFR

- A system goal
  - The system should be easy to use by experienced controllers and should be organized in such a way that user errors are minimized.
- A verifiable usability requirement derived from this goal
  - Experienced controllers shall be able to use all the system functions after a total of three hours of training.
  - The average number of errors made by experienced controllers shall not exceed two per day.
  - Assumption: An experienced controller has at least 2 years experience with the old system (as stated by the stakeholder)

28

Application-Domain Requirements

- Derived from the application domain
- 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

29

Examples of Application-Domain Requirements

- **Library system**
  - The system interface to the database must comply with standard Z39.50.
  - Because of copyright restrictions, some documents must be deleted immediately on arrival. Depending on the user's requirements, these documents will first be printed either locally or printed to a network printer and retrieved by the user.

- **Train protection system**
  - 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.

30

Problems Concerning Application-Domain Requirements

- **Understandability**
  - Requirements are expressed in the language of the application domain
  - This is often not understood by software engineers developing the system

- **Implicitness / Tacit knowledge**
  - Domain specialists understand the area so well that they do not think of making the domain requirements explicit
  - People are often unaware of the tacit knowledge they possess and therefore cannot express it to others

31

Emergent Properties (when the system consists of several sub-systems)

- **Properties of the system as a whole**
  - Requirements which cannot be addressed by a single component, but which depend for their satisfaction on how all the software components interoperate
  - Only emerge once all individual subsystems have been integrated
  - Dependent on the system architecture

- **Examples of emergent properties**
  - Reliability
  - Maintainability
  - Performance
  - Usability
  - Security
  - Safety

32

The Requirements Engineering Process