CS 5704: Software Engineering

Requirements

Dr. Pardha S. Pyla

What are requirements?

- Sommerville: "descriptions of the services provided by the system and its operational constraints"
- Translation of users' needs into the development domain
  - effective description of envisioned system
  - if realized, solves users' particular problems

Amount of detail

users

developers

high-level abstract statement of a service or need

detailed formal statement of a function or constraint

User vs. system requirements*

- User requirement
  - natural language description of needed services and constraints
- System requirements
  - formal, precise, complete description of what is to be implemented

*Sommerville

Example* and discussion

User

1. *LIBSYS shall keep track of all data required by copyright licensing agencies in the UK and elsewhere*

System

1.1. *On making a request for a document from LIBSYS, the requester shall be presented with a form that records details of the user and the request made*

1.2. *LIBSYS request forms shall be stored on the system for five years from the date of the request*

1.3. *All LIBSYS request forms must be indexed by user, by the name of the material requested and by the supplier of the request*

1.4. *LIBSYS shall maintain a log of all requests that have been made to the system*

1.5. *For materials where authors' lending rights apply, loan details shall be sent monthly to copyright licensing agencies that have registered with LIBSYS*

Flexibility

- Mandatory
  - *The compute time for T.23 l transaction shall not exceed 5 seconds*
- Desired
  - *The menus in system should present no more than 6 options at any time*
- Optional
  - *The list of products may be sorted using bubble sort algorithm*

Types of requirements

- Functional requirements
  - *"statements of services the system should provide"*
  - *"how system should react to particular inputs"*
  - *"how system should (not) behave in particular situations"*
- Non-functional requirements
  - *"constraints (timing, standards, reliability) on the services or functions offered by the system"*

Functional requirements

- System functions, inputs, outputs, exceptions, relationships, etc.
- Example requirements for LIBSYS*

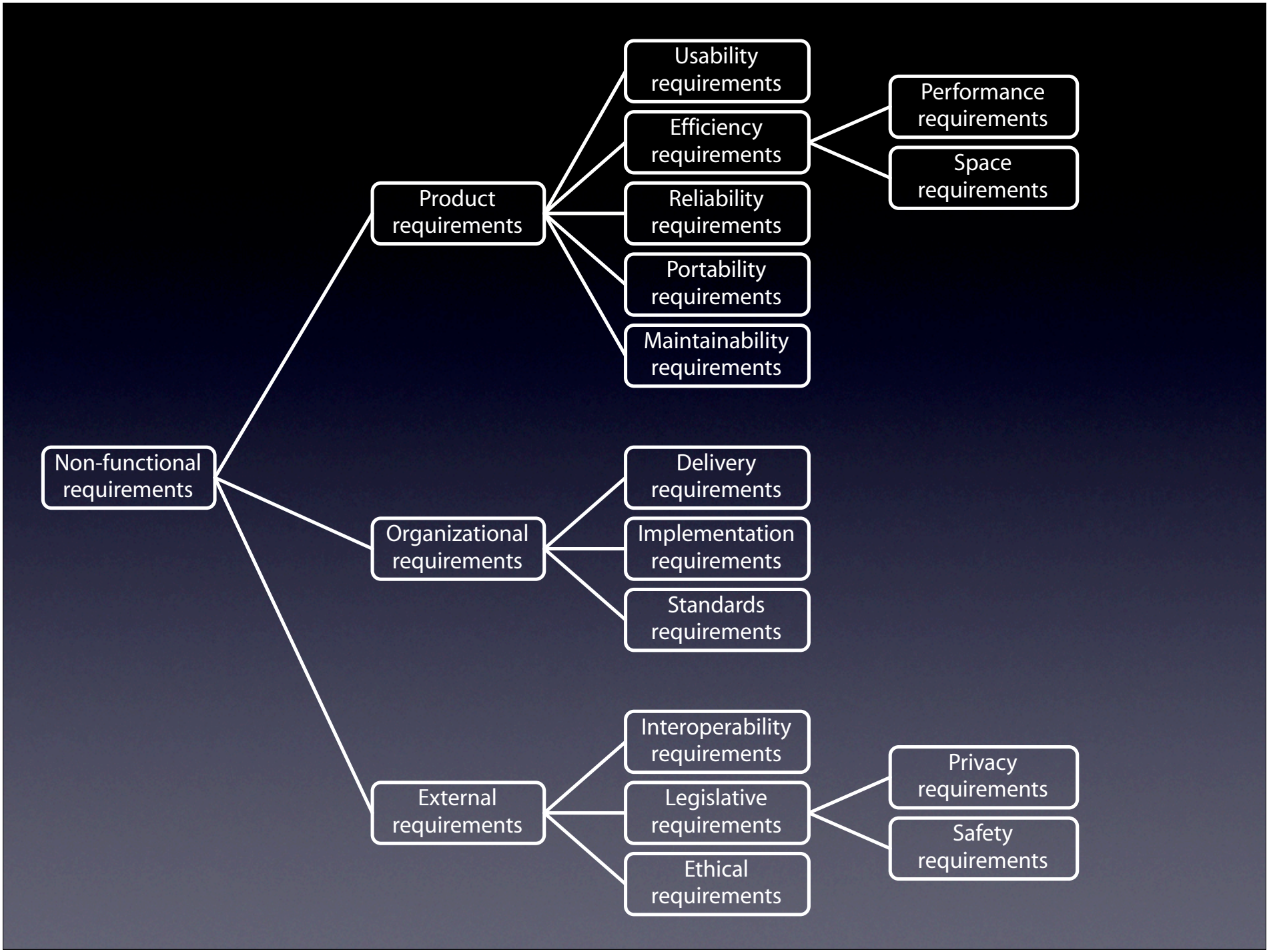
1. The user shall be able to search either all of the initial set of databases or select a subset from it
2. The system shall provide appropriate viewers for the user to read documents in the document store
3. Every order shall be allocated a unique identifier (ORDER_ID), which the user shall be able to copy to the account's permanent storage area

Non-functional requirements

- Usually represent system-wide concerns
- Often more critical than functional requirements
  - Example: reliability in aircraft system
- May constrain the development process
  - Example: standards, interoperability needs

Types of non-functional requirements

- Product requirements: *specify product behavior*
  - Example: The user interface for LIBSYS shall be implemented using simple HTML without frames or Java applets
- Organizational requirements: *derived from policies and procedures*
  - Example: The system development process and deliverables shall conform to the IEEE 0000.0 standard
- External requirements: *cover everything external*
  - Example: The system shall not disclose any personal information about system users apart from their name and library reference number to the library staff


Non-functional requirements
Product requirements
Usability requirements
Efficiency requirements
Performance requirements
Space requirements
Reliability requirements
Portability requirements
Maintainability requirements
Organizational requirements
Delivery requirements
Implementation requirements
Standards requirements
External requirements
Interoperability requirements
Legislative requirements
Privacy requirements
Safety requirements
Ethical requirements

Specifying non-functional requirements

Property	Measure
Speed	Processed transactions/sec Screen refresh time
Usability	Time on task Error counts
Reliability	Mean time to failure Rate of failure occurrence
Robustness	Time to restart after failure Percentage of events causing failure

Domain requirements

- Related to domain characteristics rather than user needs
- May add to or constrain functional and non-functional requirements
- Example: deceleration $D_{train} = D_{control} + D_{gradient}$ where ....

Typical problems with requirements

- Lack of clarity (may solve a non-problem)
- Requirements amalgamation
  - Example: LIBSYS shall provide a financial accounting system that maintains records of all payments made by users of the system. System managers may configure this system so that regular users may receive discounted rates.
- Incompleteness
- Not representing priorities
- Inappropriate level of specification (right amount of constraints)

A good requirements specification is*

- clear (satisfies real user need)
- complete (lists all known requirements)
- consistent (has no conflicting requirements)
- traceable (to the origin of the requirement)
- design-agnostic (prevents premature constraints)
- testable (to ascertain the realization of requirements)

* Davis, Leffengwell

Requirements evaluation: Ambiguity*

- An SRS is unambiguous if an only if every requirement stated in it has only one possible interpretation
  - glossary of terms used can mitigate this problem

* James D. Arthur

Checking for ambiguity*

- Are the functional requirements modular (i.e. input/process/output); is each function explicitly identified?
- Is formal/semiformal language used? Is it unambiguous?
- Are reqts. clear enough to be basis for design/testing?
- Are requirements differentiated from other contextual information?

Examples of ambiguity*

- "The control total is taken from the last record"
  - the control total is taken from record at the end of file
  - the control total is taken from the latest record
  - the control total is taken from the previous record
- "All customers have the same control field"
  - all customers have the same value in their control field
  - all customer control field have the same format
  - one control field is issued for all customers

Requirements evaluation: Completeness*

- An SRS is complete if and only if
  - no requirements or needs of users are overlooked
  - the system response has been specified for every possible set of inputs
  - the specification conforms to any SRS standard that applies to it

Checking for completeness*

- Do functional requirements cover all abnormal situations?
- Have temporal aspects of all functions been considered?
- Are requirements for which future changes are anticipated identified?
- Are environmental conditions specified for all operating modes?
- Are all relevant non-functional requirements specified?

Examples of incompleteness*

- An SRS with "to be determined" (TBD) placeholders
- Missing information about dependent requirements
  - Example: Login requirements not specifying acceptable format of password field
- Non-existent references

Requirements evaluation: Consistency*

- An SRS is consistent if an only if no set of individual requirements described in it conflict
  - designing an object using different terms
  - characterizing an object differently in different places
  - having logical or temporal conflicts among actions

Checking for consistency*

- Is there internal consistency among software requirements?
- Are specified models, algorithms, and numerical techniques compatible?
- Are specified requirements compatible with operational environment of hardware and software?

Examples of inconsistency*

- Request for user input called *prompt* in one requirement and *cue* in another
- Report format is *tabular* in one requirement and *textual* in another
- "A must always follow B" vs. "A and B occur simultaneously"

Requirements evaluation: Testable*

- A requirement is testable if and only if it is stated in terms that permit establishment of test criteria and performance of tests to verify those criteria are met
  - need to have a cost-effective process with which it is possible to check if system meets requirement

Checking for testability*

- Can software be checked to see if requirements have been fulfilled?
- Is there a test plan in the SRS?

Examples of non-testability*

- *Program shall never enter an infinite loop*
  - One cannot test this condition!
- *Product shall work well*
  - Cannot test this without defining "well"
- *Output of program shall usually be given within 20 seconds*
- *Output of program shall be given within 20 seconds of event X, 60% of the time, and within 30 seconds of event X, 100% of the time*

Requirements evaluation: Traceability*

- An SRS has traceability if and only if the origin of each requirement is clear and if it facilitates referencing each requirement in next iteration
  - backward traceability (to previous stages of development)
  - forward traceability (to all documents spawned by SRS)

Checking for traceability*

- Does the SRS have a traceability matrix?
- Are there references in requirements to concept documents, user interviews, etc.?

Examples of traceability

	AI2	AI3	AI4	AI5	AI6
R03		✔	✔		
R04		✔	✔	✔	
R05	✔	✔			

pause

Requirements engineering

- A bridge to design and construction
  - originates from system concept
  - originates from stakeholders needs
- A structured process for
  - understanding what customer wants
  - includes analysis, feasibility, negotiation, specification, validation, management, ...

Key activities

- Inception
- Elicitation
- Elaboration
- Negotiation
- Specification
- Validation
- Management

Inception

- How the project gets started
  - casual conversation
  - business needs identified
  - potential new market or service discovered
- Conduct market analysis
- Initial feasibility analysis
- Specify potential project scope
- Start with a set of context-free questions

Elicitation

- Objectives of customers/users
- High-level needs, scenarios
- Look out for
  - scope of project
  - expectations management
  - domain expertise
  - validation

Elaboration

- Needs to requirements
- All about modeling and refinement
  - Functional requirements
  - Non-functional requirements
- Still not complete
- More insights into needed system capabilities

Negotiation

- Time to tackle conflicts in requirements
- Prioritization of requirements
- Risk assessment
- Preliminary assessment of project costs and time
- Iteratively eliminate, consolidate, prioritize
  - Goal: make all parties feel satisfied

Specification

- “Final” requirements description
- Usually acts as a contract between customer and developers
- Has all features discussed before

Aside: Requirement attributes

- Customer benefit (priority)
- Cost to implement
- Development priority - fn(priority, cost)
- Status (proposed, approved, implemented, validated)
- Context (author, date created/modified, version, etc.)
- Rationale
- Relationship with other requirements (dependencies, etc.)

Validation

- Examining specifications to ensure completeness, consistency, and other quality factors
- Involves formal technical review
  - engineers
  - users/customers

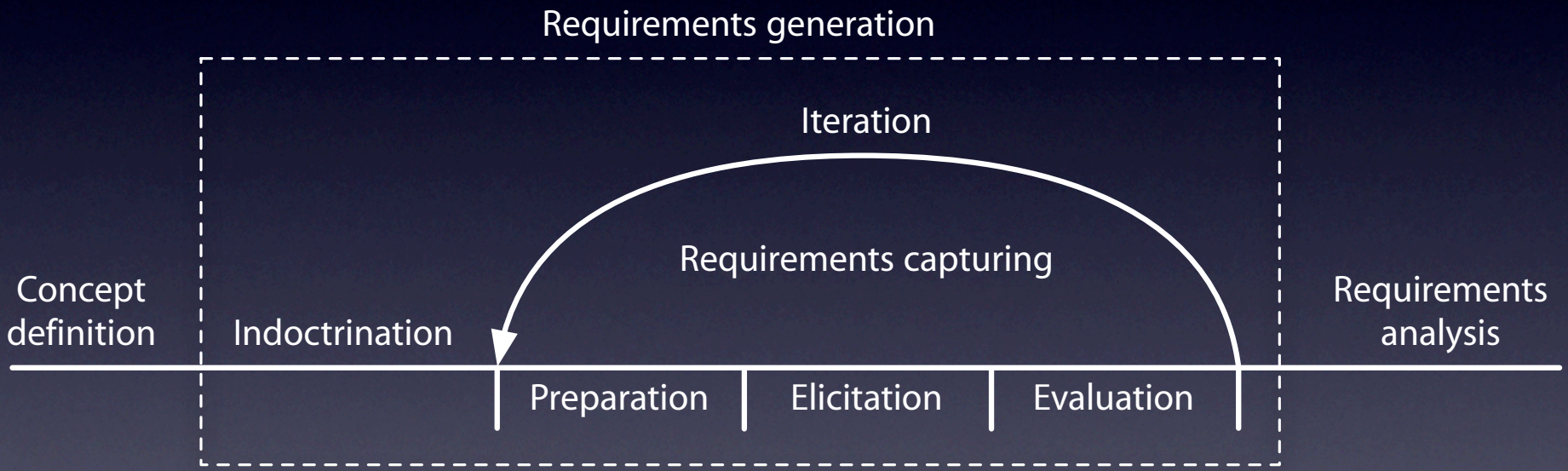
Management

- Activities that help control and track creation and modification of requirements
  - traceability matrices
  - requirements management tools, etc.

Typical RE process activities

- Identifying the stakeholders
- Recognizing multiple viewpoints
- Using participatory/collaborative sessions
- Asking the right questions upfront
- Creating user scenarios, use-cases
- Building analysis models (e.g. activity diagrams, flow diagrams)

The Requirements Generation Model (RGM)



Indoctrination in RGM

- "Focus is on defining the working relationship between customer and requirements engineer"
  - introduce customer to requirements process
  - introduce reqts. engineer to problem domain
  - define participants' tasks and duties
    - agenda for meetings, setting appropriate objectives
    - expectations for each side

Requirements capturing

- "A structured approach to the interaction format and information exchange"
- Preparation
  - indoctrination activities done?
  - new iteration deemed necessary?
  - review/resolve open issues
  - arrange for elicitation/refinement meeting
    - declarative, explorative, elicitive, review

Capturing - elicitation

- objective is to "identify and record accurately system requirements as expressed by customer"
- customers may *imply* requirements
- protocols:
  - set time limits for elicitation meetings
  - identify session objectives and subject matter
  - define participant roles
  - set responsibility expectations
  - control interruptions

Capturing - evaluation

- “focus on assessing the adequacy of evolved requirements”
- basis for judging if additional iteration is necessary
- summary of requirements presented to customer for examination and re-interpretation
- conduct verification of gathered requirements

Other processes

- Contextual inquiry and analysis
- Requirements Triage
- Win-win Spiral Model
- Ethnography

end