CHALMERS |  UNIVERSITY OF GOTHENBURG

DIT045 H17 Requirements and User Experience

Requirements Documentation & Quality

Jennifer Horkoff

Requirements Documentation

- Last lecture we talked about:
  - Requirements
  - System Requirements
  - User Requirements
  - Goals
  - Constraints
  - Domain Properties

- How do we document this important information?
  - In an SRS?
  - In a backlog?
  - Could be, but in what form? How do we write them?

- What makes a requirement good or bad? What are the qualities of good requirements?

How to Capture/Document Requirements

- A few ways, we'll cover some of the main options

- Using Natural language (Part 1 of this lecture)
  - Traditional SRS form
  - User stories
  - Templates
  - Structured Natural Language

- Using models (Lecture 5)

Requirements in Natural Language

- Natural language is just written language, e.g., English, Swedish
- Good in that everyone (who understands the language) can read it
  - Technical background not needed
  - Unlike mathematical formulas or models, not much training is needed to read
  - Very flexible
- Challenging in that natural language can be ambiguous (more on this later)
- Everyone reading the requirement has a different background and experiences, this effects how they are read and understood (Pohl & Rupp)

SRS Requirements

- One sentence statements
- Often starts with:
  - "The system shall..."

- Examples for an online marking tool
  - There must be a place for summary comments in all types of Rubrics/Marking Schemes
  - The summary comments given by a Marker should be stored and potentially reused in later instances of the same assignments by the same Marker
  - If a summary comment is reused, this comment should be editable, and the Marker should be allowed to add additional comments to it
  - The marking scheme/rubric should allow extra deductions/additions to be added, this section will require both a text explanation, a number for the addition/deduction and an indication of whether it is a deduction or addition
  - The marks given or taken away by extra deductions/additions should be automatically calculated in with the mark total

SRS Requirements

- Examples for a lift (elevator):
  - A lift will only reverse direction when stopped at a floor
  - The system will cycle the lift doors every time that a lift stops at a floor
  - The lift must never be moved with the doors open
  - Each lift should be used an approximately equal amount

- Examples for a boat racing results program:
  - All input to the system is to be entered by the user
  - The user can enter and modify boat details
  - The user can enter and modify race details
  - When the user selects the option to modify the boat details, they are prompted to enter the boat's name
  - Subject to the constrains detailed below, the user can enter amend and delete details of: boat-class, boat, series, race, series-entry, race-entry
    - <details>

(Bray)

User Stories

- Associated with Agile methods
- Typically represent higher-level user requirements
  - Can used at a lower level of abstraction to capture system requirements (but this is strange unless you can identify clearly what the user is doing)
- "Describes functionality that will be valuable either to a user or purchaser of a system or software" (Cohn)
- User story cards "represent customer requirements, rather than document them"
- They are a placeholder for a conversation, like a to-do list, what most people would use sticky notes for
- All about communication
- Burden of limited resources is shared
  - It's not my problem, it's our problem

User Stories

- Three aspects
  - A written description as a reminder (often on a card) (Cohn)
  - Conversations about the story to flesh out details
  - Tests that convey and document details and that can be used to determine when a story is complete
- Ideally they are written by the customers
- Use common, non-technical language
- User stories are not contractual obligations
  - The tests should serve this purpose

Big User Stories

- They can be decomposed or refined, but in separate stories, not in the typical requirement hierarchy
- Big user stories are called epics
- Epic: "A user can search for a job", split into:
  - "A user can search for jobs by attributes like location, salary range, job title, company name and the date the job was posted."
  - "A user can view information about each job that is matched by a search."
  - "A user can view detailed information about a company that has posted a job"
- Not: "4.6) A user can view information about each job that is matched by a search.
  - 4.6.1) A user can view job description
  - 4.6.2) A user can view a job's salary range
  - 4.6.3) A user can view the location of the job"
- These details can be discussed later

(Cohn)

User Story Format

- The written description is often written in structured language:
- *As a < type of user >, I want < some goal > so that < some reason >*

- Example Epic: "As a user, I can backup my entire hard drive."
- Split into
  - "As a power user, I can specify files or folders to backup based on file size, date created and date modified.
  - As a user, I can indicate folders not to backup so that my backup drive isn't filled up with things I don't need saved."

(Cohn, https://www.mountaingoatsoftware.com/agile/user-stories)

Example User Stories

As a/an	I want to...	so that...
moderator	create a new game by entering a name and an optional description	I can start inviting estimators
moderator	invite estimators by giving them a url where they can access the game	we can start the game
estimator	join a game by entering my name on the page I received the url for	I can participate
moderator	start a round by entering an item in a single multi-line text field	we can estimate it
estimator	see the item we're estimating	I know what I'm giving an estimate for
estimator	see all items we will try to estimate this session	I have a feel for the sizes of the various items
moderator	see all items we try to estimate this session	I can answer questions about the current story such as "does this include ___"
moderator	select an item to be estimated or re-estimated	the team sees that item and can estimate it

(Cohn, https://www.mountaingoatsoftware.com/blog/advantages-of-the-as-a-user-i-want-user-story-template)

Conditions of Satisfaction

- "a high-level acceptance test that will be true after the agile user story is complete."
- "As a vice president of marketing, I want to select a holiday season to be used when reviewing the performance of past advertising campaigns so that I can identify profitable ones."
- Detail could be added to that user story example by adding the following conditions of satisfaction:
  - Make sure it works with major retail holidays: Christmas, Easter, President's Day, Mother's Day, Father's Day, Labor Day, New Year's Day.
  - Support holidays that span two calendar years (none span three).
  - Holiday seasons can be set from one holiday to the next (such as Thanksgiving to Christmas).
  - Holiday seasons can be set to be a number of days prior to the holiday.
- Usually written on the back of the card

(Cohn, https://www.mountaingoatsoftware.com/agile/user-stories)

Requirements Templates

- Structured text formats for capturing requirements
- Help you (the analyst/requirements engineer) remember all of the information you should collect for a requirement
- Encourages structure and organization
- More than one type/design

- Example: Volere (Atlantic Systems Guild, Robertson^2)

Volere Template

Requirement #: Requirement Type: Event/use case #:

Description:

Rationale:

Source:
Fit Criteria:

Customer Satisfaction: Customer Disatisfaction:
Dependencies: Conflicts:
Supporting Materials:
History:

Volere

Volere Template

http://www.itib.net/Conform/Models/Volere/Release%2011/Volere%20Requirements%20Resources.htm

Volere Example

Requirement #: 75 Requirement Type: 9 Event/BUC/PUC #: 7, 9

Description: The product shall record all the roads that have been treated

Rationale: To be able to schedule untreated roads and highlight potential danger

Originator: Arnold Snow - Chief Engineer

Fit Criterion: The recorded treated roads shall agree with the drivers' road treatment logs and shall be up to date within 30 minutes of the completion of the road's treatment

Customer Satisfaction: 3 Customer Dissatisfaction: 5

Dependencies: All requirements using road and scheduling data Conflicts: 105

Supporting Materials: Work context diagram, terms definitions in section 5

History: Created February 29,2010

Volere

Requirement Types

- They have 27 types of requirements...
  - Allows to sort by type, check for gaps or duplicates
  - Easier to write appropriate fit criterion when the type is known
  - Can show them to stakeholders by categories
  - All types are listed here: http://www.volere.co.uk/template.htm

PROJECT DRIVERS

1. The Purpose of the Project
2. Client, Customer, other Stakeholders
3. Users of the Product

PROJECT CONSTRAINTS

4. Mandated Constraints
5. Naming Conventions and Definitions
6. Relevant Facts and Assumptions

FUNCTIONAL REQUIREMENTS

7. The Scope of the Work
8. The Scope of the Product
9. Functional and Data Requirements

NON-FUNCTIONAL REQUIREMENTS

10. Look and Feel Requirements
11. Usability Requirements
12. Performance Requirements
13. Operational Requirements
14. Maintainability Requirements
15. Security Requirements
16. Cultural and Political Requirements
17. Legal Requirements

PROJECT ISSUES

18. Open Issues
19. Off-the-Shelf Solutions
20. New Problems
21. Tasks
22. Cutover
23. Risks
24. Costs
25. User Documentation and Training
26. Waiting Room
27. Ideas for Solutions

Structured Natural Language

- Can add structure to natural language
  - Makes it easier to read (probably)
  - Enforces completeness and uniformity
  - Similar to a template, but each individual line of text is a template, instead of whole page of information

- As with templates, there's more than one to structure natural language requirements
  - User stories
  - EARS
  - PLanguage

The Easy Approach to Requirements Syntax (EARS)*

The *Easy Approach to Requirements Syntax* (EARS) consists of a set of patterns for specific

Pattern	Pattern
Ubiquito	The <system|actor> shall <action> <object>.
Event-	**When** <trigger> <optional precondition>, the <object>
State-	**While** <system state|actor state>, the <system|actor>
Unwante Behavior	**If** <unwanted state|unwanted event>, THEN the <object>
Optional Feature	**Where** <feature is included>, the <system|actor> shall
Compou	(combinations of the above patterns)

Examples of EARS Syntax

When a user commands installation of an _application that accesses _communicationsFunctions, the system shall prompt the user to acknowledge the access and agree before continuing installation.

When <trigger> <optional precondition>, the <system|actor> shall <action> <object>

While the _phoneFunction is active and _speakerMode is off, **when** the system detects the user's face in proximity to the display, the system shall turn off the display and deactivate the display's touch sensitivity.

(combinations of the above patterns)

While <system state|actor state>,

When <trigger> <optional precondition>, the <system|actor> shall <action> <object>

Examples of EARS Syntax

While in _standby, **if** the battery capacity falls below _lowBatteryThreshold remaining, the system shall change the _systemLED to flashing red.

While <system state|actor state>,

If <unwanted state|unwanted event>, THEN the <system|actor> shall <action> <object>

Where the system contains two SIM cards, the system shall allow the user to assign a default network to each contact in the address book, and shall use each contact's assigned network for outgoing voice and SMS communications.

Where <feature is included>, the <system|actor> shall <action> <object>

PLanguage

- Invented by consultant Tom Gilb, structure requirements using keywords
- Typical requirement: "At least 95 percent of the time, the system shall take no more than 8 seconds to display any of the predefined accounting reports."

- **TAG:** Performance.Report.ResponseTime
- **AMBITION:** Fast response time to generate accounting reports on the base user platform.
- **SCALE:** Seconds of elapsed time between pressing the Enter key or clicking OK to request a report and the beginning of the display of the report.
- **METER:** Stopwatch testing performed on 30 test reports that represent a defined usage operational profile for a field office accountant.
- **GOAL:** No more than 8 seconds for 95 percent of reports. ß Field Office Manager
- **STRETCH:** No more than 2 seconds for predefined reports, 5 seconds for all reports.
- **WISH:** No more than 1.5 seconds for all reports.
- **base user platform DEFINED:** Quad-core processor, 8GB RAM, Windows 8, QueryGen 3.3 running, single user, at least 50 percent of system RAM and 70 percent of system CPU capacity free, network connection speed of at least 30 Mbps.

(Example, J. Beatty, http://www.seilevel.com/requirements/specifying-quality-requirements-with-planguage)

Always Dilbert

Requirements Quality

- Note: we can have requirements expressing system quality (i.e., quality requirements, NFRs)
  - See last lecture, this is not what we are talking about here
- This lecture is about the quality of requirements
  - What makes requirements good or bad?
  - What are the desirable characteristics of requirements?
- Note: the quality of good requirement differs somewhat from SRS-style (IEEE 83o) requirements to User Stories
  - We start with qualities for SRS Requirements
  - Then describe desired qualities for user stories

Characteristics of a Good Requirement

- IEEE 830 Standard, 1998 provides basic characteristics
- Others have added further useful characteristics

- Some characteristics apply to a single requirement
- Others apply to a set of requirements (e.g., an entire specification, a backlog, a list)

Characteristics of a Good Requirement: Feasible

- 1) Feasible (Berenbach et al.)
  - If an implementation is possible within the constraints of the program or product
  - The system must process 1 billion transactions per system
  - The system must allow users to reach Mars
  - The system will be available during power outages
  - (whether these requirements are feasible or not depends on context)

Characteristics of a Good Requirement: Valid

- 2) Valid
  - If and only if the requirement is one that the system must meet (Berenbach et al.)
    - The users asked for it
    - It is a constraint
    - The business asked for it
    - Else… ? Why is it there?
  - The inclusion of requirements that are nice but not valid is called "gold plating"
  - The system shall allow users to export their data to Excel
    - But what if no one asked for this?
  - The system should be compatible with both Android and Apple
    - But the customer only asked for an Android app…

Characteristics of a Good Requirement: Unambiguous

- 3) Unambiguous
  - if it has only one interpretation (Berenbach et al.)
  - "The data complex shall withstand a catastrophe (fire, flood)"
    - Only fires and floods? "The data complex shall withstand a catastrophe of type fire or flood"
    - All catastrophes, including fires and floods? "The data complex shall withstand any catastrophe, two examples being fires and floods"
  - "The interface shall have standard menu buttons for navigation"
    - Four possible interpretations: (Li et al.)
      - (1) there should be buttons all of which should be standard;
      - (2) there should be buttons some of which should be standard;
      - (3) if there are buttons then all of them should be standard;
      - (4) if there are buttons then at least some should be standard.
      - Fix in the case of (1), "the user interface shall have menu buttons, and all of them shall be standard"

Characteristics of a Good Requirement: Unambiguous

- 3) Unambiguous
  - Avoid nouns without reference indices
  - "The data should be displayed to the user on the terminal"
    - What data?
    - What user?
    - "The system shall display billing data to the registered user on the terminal she is logged in to"

(Pohl & Rupp)

  - Avoid passive voice
  - "To log a user in, the login data is entered"
    - By whom? How?
    - "The system must allow the user to enter his user name and password using the keyboard of the terminal"

Characteristics of a Good Requirement: Verifiable

- 4) Verifiable (Berenbach et al.)
  - if the finished product or system can be tested to ensure that it meets the requirement
  - "The car shall have power breaks"
    - Does not have sufficient detail to be testable
    - "The car shall come to a full stop from 60 miles per hours within 5 seconds"
  - "The system should have good performance"
    - What is good? How to test good?
    - "The system should be able to process interactions within 0.05 seconds"
  - "The product shall return (file) search results in an acceptable time." (Li et al.)
    - What is acceptable?
    - "The product shall return (file) search results in within 0.7 seconds."

Characteristics of a Good Requirement: Avoids Design

- 5) Avoids Unnecessary Design
  - Recall last lecture
  - "The system shall use a service-based architecture"
  - "The user interface shall contain a "register user" button on the first page"
    - "The system shall allow users to register"

Characteristics of Good Requirements: Modifiable

- 6) Modifiable (Berenbach et al.)
  - Characteristic of a set of requirements or a specification
  - "if its structure and style are such that any changes to a requirement can be made easily, compete and consistently while retaining the structure and style"
  - The specification has a coherent, easy-to-follow organization and has not redundancy,
    - Keeps requirements distinct rather than intermixed"
  - General rule: information in a set of requirements should be in one and only one place so that a change to a requirements does not require cascading changes to other requirements.
  - Similar to coupling and cohesion: requirements should have low coupling and sections or grouping should have high cohesion

Characteristics of Good Requirements: Modifiable

- 6) Modifiable (Berenbach et al.)
  - Characteristic of a set of requirements or a specification
  - **All requirements should be labelled**
    - R.1 The user can enter and modify boat details
    - R.2 The user can enter and modify race details
  - Allows for cross-reference
    - R.15 There must be a place for summary comments in all types of Rubrics/Marking Schemes, see R.23 for a list of Rubric/Marking Scheme types
  - Can use sections and subsections as labels, or user and system requirements division, levels of detail
    - R.1 The user can enter and modify race details
    - R.2 The user can enter and modify boat details
      - R.2.1 When the user selects the option to modify the boat details, they are prompted to enter the boat's name

Characteristics of Good Requirements: Consistent

- 7) Consistent (Berenbach et al.)
  - Characteristic of a set of requirements or a specification
  - if it does not contradict or is not in conflict with other requirements or other stated knowledge
  - If a summary comment is reused, this comment should be editable, and the Marker should be allowed to add additional comments to it
  - …
  - Once summary comments are submitted, they cannot be edited
  - Each lift should be used an approximately equal amount
  - …
  - When all lifts are available, the master lift shall respond first to lift calls

Characteristics of Good Requirements: Complete

- 8) Complete
  - Characteristic of a set of requirements or a specification
  - "A specification is complete if it includes all relevant correct requirements, and sufficient information is available for the product to be built" (Berenbach et al.)
  - Complete upfront design is less emphasized with the rise of Agile methods (more in Lecture 4)
  - Instead take a "chunk" of work, do some (just enough) requirements analysis, and build something that works
    - Repeat, this is a "sprint"
  - Even for sprints the notion of complete is good to keep in mind
  - Are the requirements (user stories) complete enough for the sprint?
    - Enough detail? Too little? Or too much?

Characteristics of Good Requirements: Traceable

- 9) Traceable
  - Characteristic of a set of requirements or a specification
  - "ability to describe and follow the life of a requirement, in both a forward and backward direction, i.e., from its origins, through its development and specification, to its subsequent deployment and use..."

    (Gotel et al.)
  - i.e, for any requirement, where did it come from? Why is it there?
  - Users -> goals -> user requirements -> system requirements ->
  - How?
    - Structure of a document, keep refinement/decomposition clear
    - Through models
    - Through explicit notes (e.g., "derived from ...")
    - Keeping a record of elicitation information (more in Lecture 6)

Characteristics of Good Requirements: Concise

- 10) Concise
  - Characteristic of a set of requirements or a specification
  - No unnecessary or repetitive requirements (Berenbach et al.)
  - If you removed this requirement, would it change the resulting system?
    - If no, probably remove
  - Concise wording of requirements (relates to technical writing)

  - In order to save time and effort, rubrics/marking schemes should be able to be reused. There should be some way to store rubrics/marking schemes so that they can be used by other markers in the future while marking. Only administrators should be able to store rubric/marking schemes.

  - The Administrator should be able to chose whether to store a rubric/marking scheme in the system for future use

Characteristics of Good Requirements: Prioritized

- 11) Prioritized
  - Characteristic of a set of requirements or a specification
  - Three should be some indication how important a requirement is relative to the other requirements
  - Lecture 7 covers prioritization
    - You'll have to worry about this in A2

Characteristics of Good Requirements: Summary

- For a requirement
  - Feasible
  - Valid
  - Unambiguous
  - Verifiable
  - Avoids Unnecessary Design Information
- For a set of requirements
  - Modifiable
  - Consistent
  - Complete
  - Traceable
  - Concise
  - Prioritized

Exercise: Requirements Quality

- R.9 The summary comments given by a Maker should be stored and potentially reused in later assignments by other Markers.
- R.10 There should be an underlying formula which calculates the final mark based on the marks in the rubric/marking scheme
- R.11 The system shall allow importing of marking schemes, this is the only way to add a scheme
- R.12 The above format should be both interpretable by the system and easily understandable by the Administrators.
- R.13 The marking rubric should be stored in an SQL database
- R.14 The system should facilitate the creation of a marking scheme online by Professors.
- R.15 Can chose to reuse and edit previously entered summary comments, see R.5.

Exercise: Requirements Quality

- R.9 The summary comments given by a Maker should be stored and potentially reused in later assignments by other Markers.
- R.10 There should be an underlying formula which calculates the final mark based on the marks in the rubric/marking scheme
- R.11 The system shall allow importing of marking schemes, this is the only way to add a scheme
- R.12 The above format should be both interpretable by the system and easily understandable by the Administrators.
- R.13 The marking rubric should be stored in an SQL database
- R.14 The system should facilitate the creation of a marking scheme online by Professors.
- R.15 Can chose to reuse and edit previously entered summary comments, see R.5.

Ambiguous: Later instances of this assignments or later assignments?

Incomplete: Where does this formulas come from? Who enters it?

Modifiable: the above format

Unverifiable: How do you know the format created will be easy for the Administrators to understand?

Avoid Design: part of a potential solution, not a problem

Inconsistent, contradicts R.11

Modifiable: Potential problems if R5 is changed or moved

User Story Quality

- Independent – user stories don't depend on each other (as much as possible) (Cohn)
- Negotiable – not written contracts or SRS requirements, details to be negotiated in a conversation between the customer and development team. Avoid too much detail
- Valuable to purchasers or Users
- Estimatable – important for developer to be able to estimate the size of a story or the amount of time to turn it into working code
- Small – If stories are too large or too small you can't use them in planning
- Testable – stories must be testable so the developers know when they have finished coding

User Story vs. SRS Requirements Quality

User Story Quality

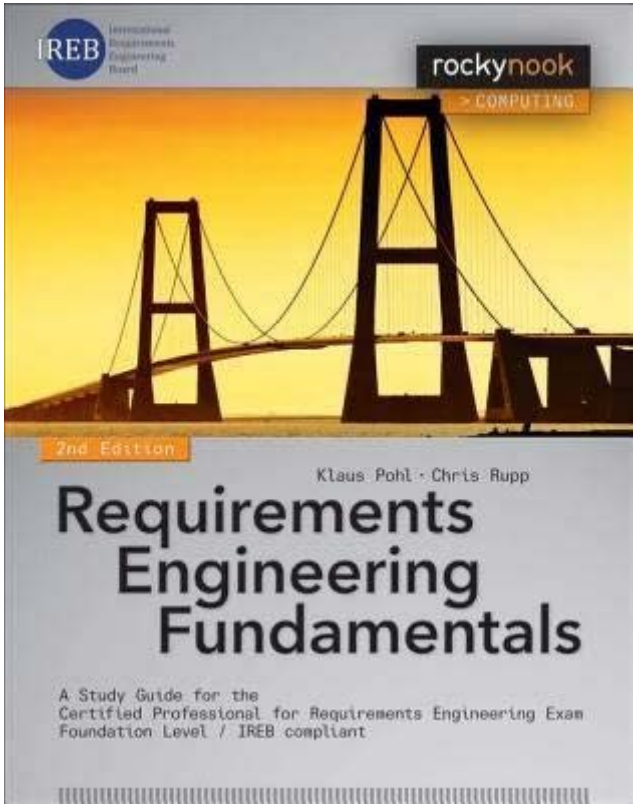
- Estimatable
- Testable
- Independent
- Valuable
- Small
- Negotiable

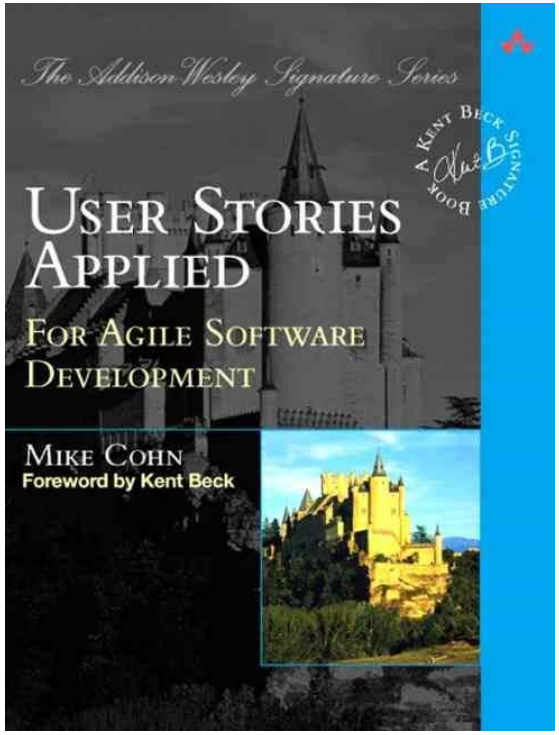
SRS Requirements Quality

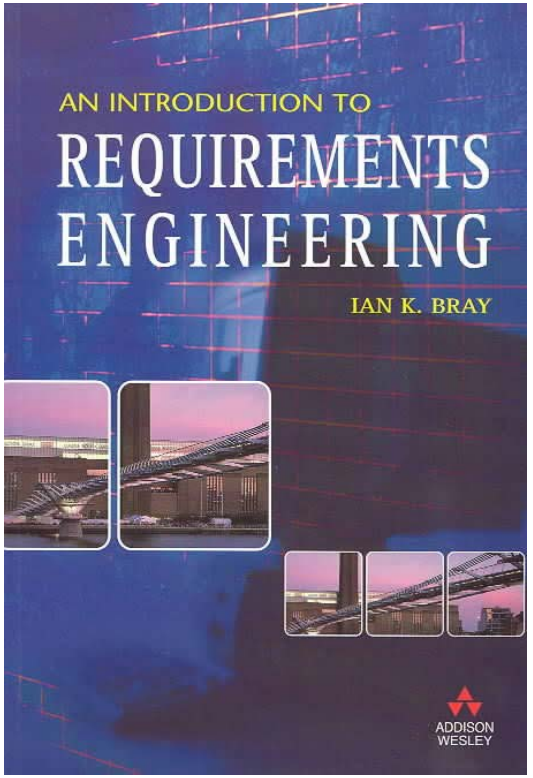
- Feasible
- Prioritized
- Verifiable
- Modifiable
- Valid
- Traceable
- Concise
- Unambiguous
- Consistent
- Avoids Unnecessary Design Information
- Complete

Questions?

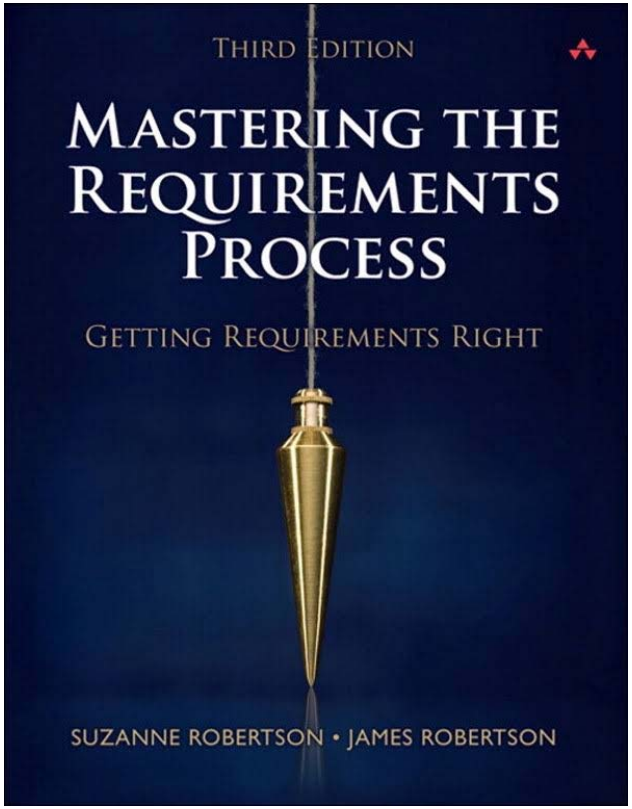
Lecture Sources

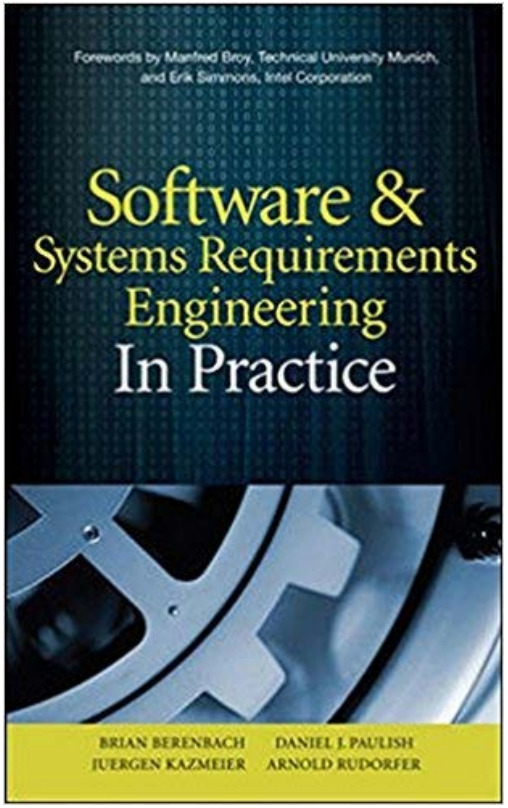






- https://www.gilb.com/





- Feng-Lin Li, Jennifer Horkoff, John Mylopoulos, Renata S. S. Guizzardi, Giancarlo Guizzardi, Alexander Borgida, Lin Liu: **Non-functional requirements as qualities, with a spice of ontology.** RE 2014: 293-302