CHALMERS |  UNIVERSITY OF GOTHENBURG

DIT045/DAT355

Requirements and User Experience

Lecture 4: Requirements Concepts, Writing Requirements, Requirements Quality

Jennifer Horkoff
jenho@chalmers.se

Agenda

Requirements concepts

Writing Requirements

Requirements Quality

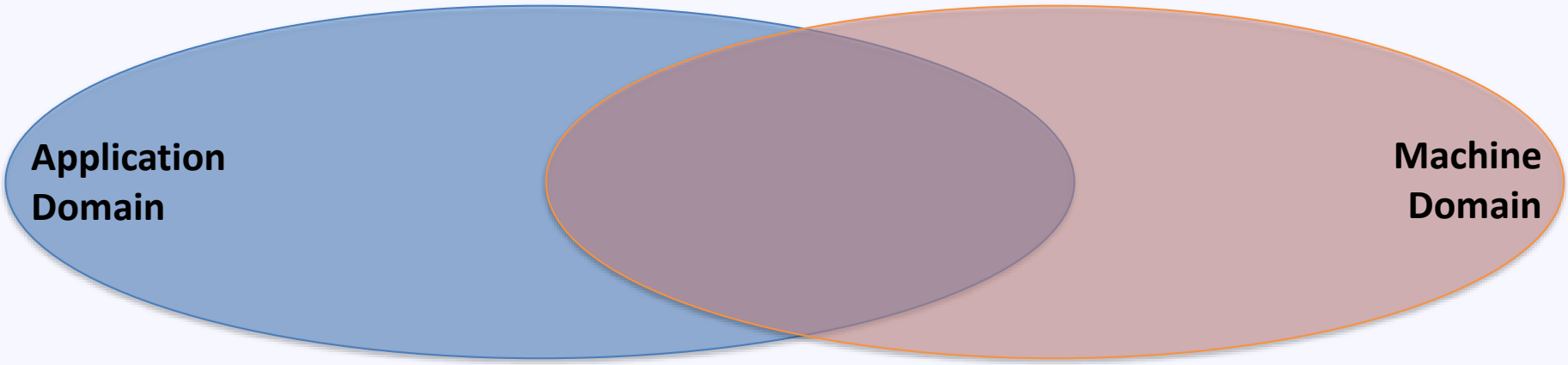
What is Requirements Engineering (RE)?

- “Requirements Engineering (RE) is a set of activities concerned with identifying and communicating the purpose of a software-intensive system, and the contexts in which it will be used. Hence, RE acts as the bridge between the real-world needs of users, customers, and other constituencies affected by a software system, and the capabilities and opportunities afforded by software-intensive technologies.”

  (Easterbrook)

- RE is a bridge between technology and the world

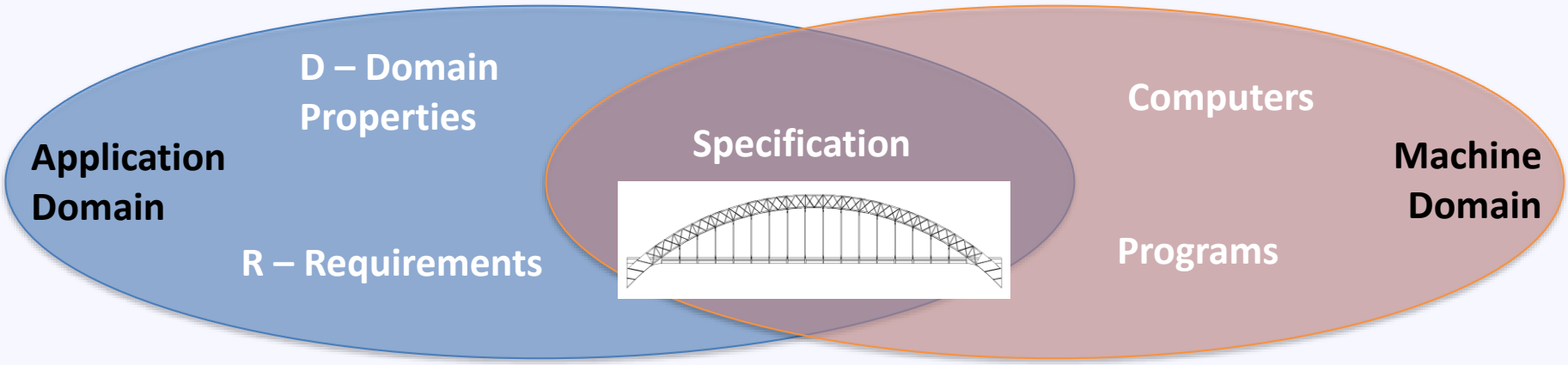
What is Requirements Engineering (RE)?



- Application Domain: the world, where people, organizations and problems live.
  - Think of it as the domain where the (Software) application is applied
- "Machine" (Software, Computer, Program) Domain: the software + hardware that solves some problem, meets some need

Zave & Jackson, Easterbrook

What is Requirements Engineering (RE)?



- Domain Properties: things in the application domain that are true whether or not we build the system
- Requirements: things in the application domain we want to make true by building the system
- Specification: the behavior that a program needs in order to solve the problem
- S, D |- R (the specification along with domain assumptions entails (satisfies) Requirements)

(Zave & Jackson, Easterbrook)

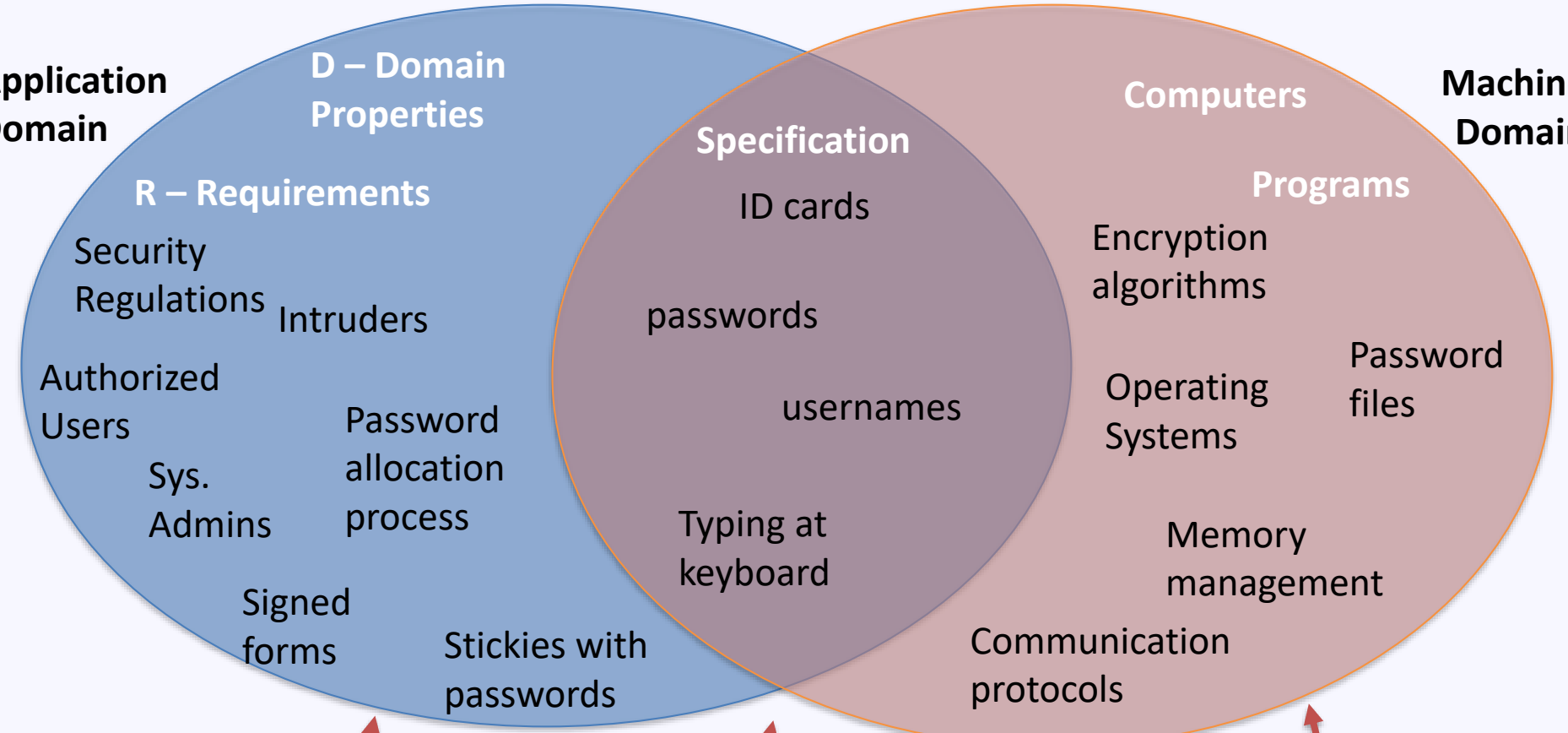
Recall RE Example 1: Network Security

- Example: Network Security
- Requirement R: “The network shall only be accessible by authorized personnel”
- Domain Properties D:
  - Authorized personnel have passwords
  - Passwords are never shared with non-authorized personnel
- Specification S:
  - Access to the network shall only be granted after the user types an authorized password

(Easterbrook & Campbell)

- Is the network secure?

Network Example



(Easterbrook & Campbell)

What is a Requirement?

- “The effects that the client wishes to be brought about in the problem domain” (Bray)
- “A software capability needed by the user to solve a problem to achieve an objective” (Dorfman & Thayer)
- “A requirement is something the product must do or a quality it must have” (Robertson^2)
- “ (1) A condition or capability needed by a user to solve a problem or achieve an objective.” (IEEE Std 610.12-1990)

(Another) Running Example

- Bike share system
- Can rent pedal bikes in urban areas

- For such a system, what are the:
  - Requirements, constraints, assumptions…

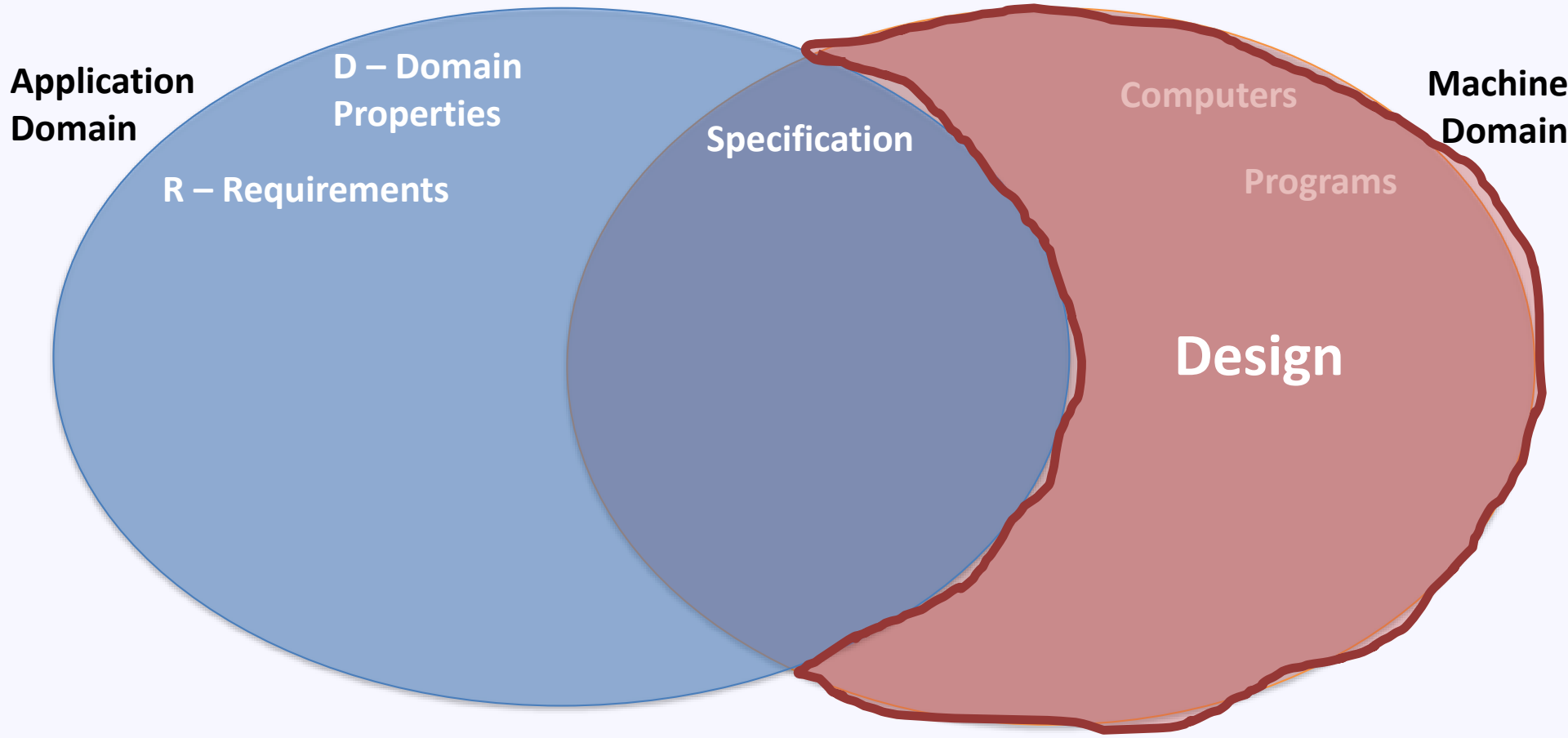
Requirements vs. Design

- Requirements describe ***what*** the system should do
  - The system shall allow users to rent a bicycle
  - The system should support payment by common credit and debit cards
  - The bike lock will not be released until the payment transaction has completed successfully

- Requirements do NOT describe ***how*** the system will do these things (this is Design)
  - The system will implement an event-based architecture to satisfy rental demands
  - The system will use a payment gateway which connects to a merchant account
  - The system will include a lock sensor to detect the presence of bikes

- Design: concerns the internal workings of the solution system. (Bray)

Requirements vs. Design



- Don’t include design information/unnecessary constraints in requirements

Requirements vs. Design

- “Requirements are expressed in a technically neutral way so as to avoid influencing the design of the solution” (Robertson^2)
- Why is the distinction between requirements and design important?
  - Design information in requirements become constraints for developers
    - Did this come from a user need or constraint?
    - Or did the person writing the requirement just think it should be done this way (in which case it can be changed)
  - Allow designers/architects/developers as much freedom as possible
    - To allow them to make the best (or a good) solution
  - While still making sure user needs are met

User vs. Stakeholder

- Users: people who directly use the system (any function of the system)
  - (Not developers or installers)

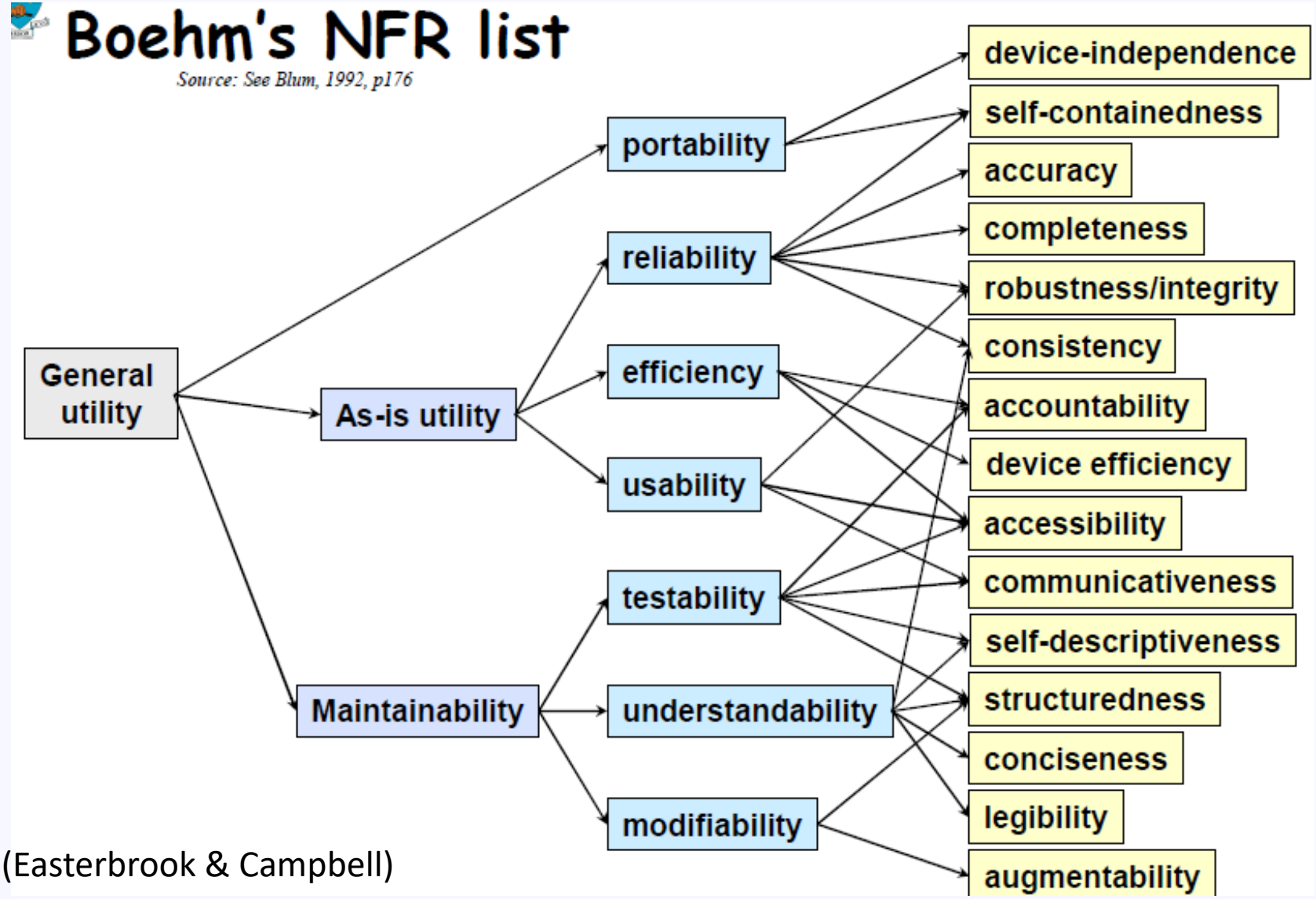
- Stakeholders: anyone who has a “stake” in the system
  - Anyone who can be positively or negatively affected

  - Users are stakeholders
    - Bike renters, maintenance staff (who use the system)
  - Managers (those who pay for the system)
    - City of Gothenburg or Bike Share Company
  - Users/managers of interacting systems
    - Whoever runs the payment system used, transportation planners
  - Anyone else affected
    - Neighbors to bike locations, city cleaners, ….

Functional vs. Non-Functional Requirements

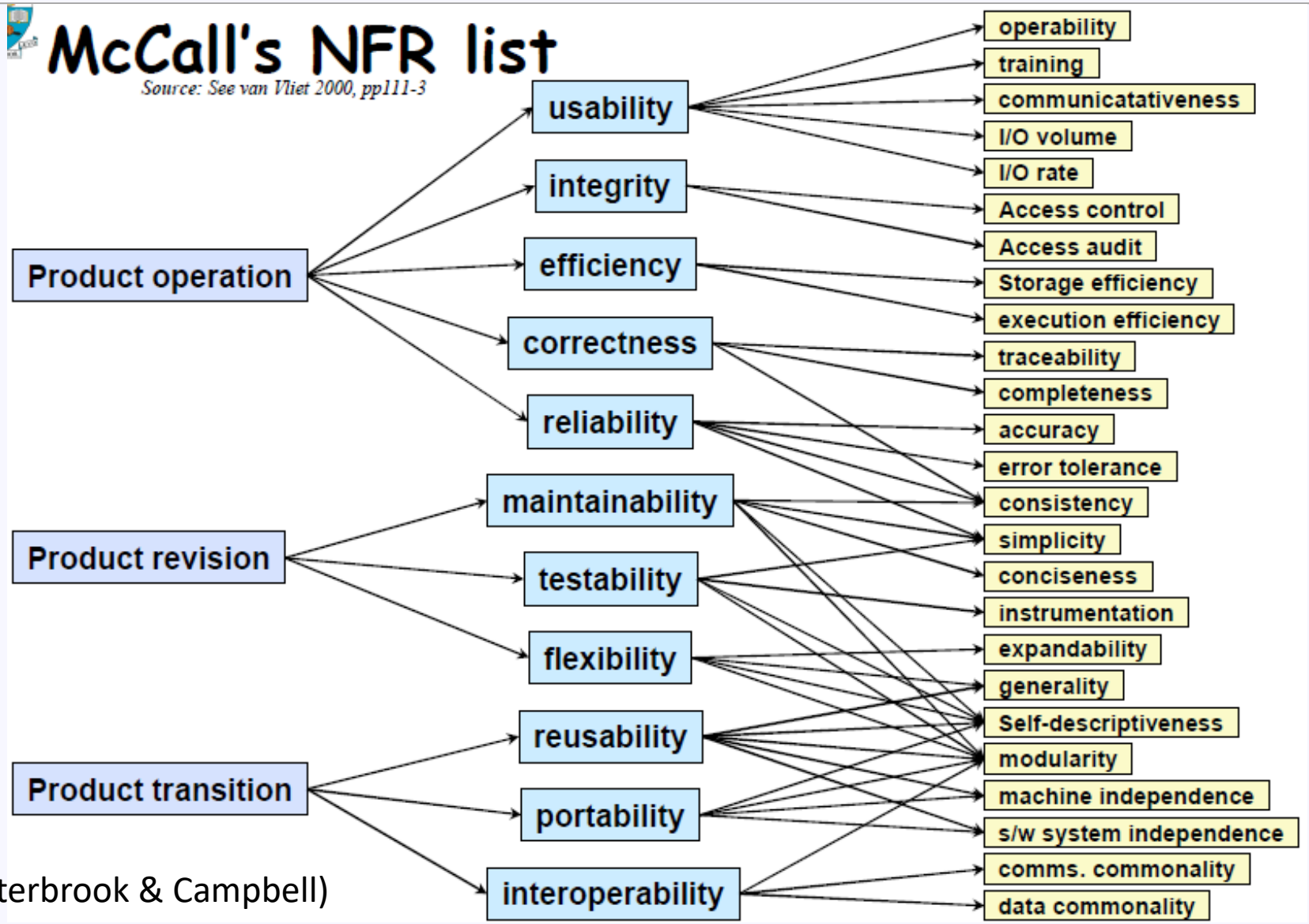
- “Functional requirements are things the system must do” (Robertson^2)
- “Nonfunctional requirements are qualities the product must have” (Robertson^2)
- Functional Requirements (FRs):
  - I want to rent a bike
  - I want to return a bike
  - If the bike is returned before it is due, the system shall display a positive message to the user
  - The system shall allow the user to indicate how long they would like to rent the bike, choosing from 1, 3, or 5 hours.
- Non-Functional Requirements (NFRs):
  - It should be easy to rent a bike
  - The system should allow quick bike rentals
  - The system should be usable, it should be evaluated at 0.8 on a standard usability questionnaire
  - 80% of users should be able to rent a bike within 2 minutes

NFR Hierarchies



(Easterbrook & Campbell)

NFR Hierarchies



(Easterbrook & Campbell)

NFRs as Qualities

- Technically an NFR is any requirement that is not-functional
- This is not a very helpful definition (defining something in terms of what it is not)
- More useful to think about qualities
- When eliciting requirements, we not only need to understand ***What*** the system needs to do, but ***How Well*** it needs to do it

- A system could meet all it's FRs, but still fail due to poor quality
  - Poor performance
  - Poor usability
  - Poor availability

Requirements vs. Domain Properties

- Domain Properties/Assumptions:
  - Characteristics of the problem domain that are assumed to hold
  - Characteristics which are relevant for the functionality expressed by the requirements
  - Also called domain assumptions, as you assume they are true
  - They are not requirements, because they don’t have to be achieved by the system
  - But they should be stated and remembered, as the system may break if they later do not hold

Requirements vs. Domain Properties

- Example domain assumptions:
  - We assume the bike rental system has access to a secure power source
    - But what if it doesn't? What happens if the power goes out?
  - We assume the city will provide sufficient law enforcement to prevent vandalism
  - We assume connection to the external payment system is available 24/7.
  - We assume the system will not be used during weather conditions in which the bikes and mechanical parts can freeze

Requirements vs. Constraints

- “Constraints are global issues that shape the requirements” (Robertson^2)
  - The system shall be ready in fall of 2018
  - Wireless networking must follow the IEEE 802.11 standard

- Where do they come from?
  - The business (money, agreements)
  - Laws and regulations
  - Existing infrastructure
  - ….

- “Constraints are simply another type of requirement” (Robertson^2)
- They are a requirement that doesn’t come from user needs

Constraints

- Examples of constraints include:
  - interfaces to already existing systems (e.g., format, protocol, or content) where the interface cannot be changed,
  - physical size limitations (e.g., a controller shall fit within a limited space in an airplane wing),
  - laws of a particular country,
  - available duration or budget,
  - pre-existing technology platform,
  - user or operator capabilities and limitations

ISO/IEC/IEEE 29148:2011(E)

Virtual Museum/Gallery

- Think of some:
  - Users, Stakeholders, Functional Requirements, Non-Functional Requirements, Domain Properties, Constraints

Writing Requirements

How to Capture/Document Requirements

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

- Using Natural language (this lecture)
  - Traditional SRS (Software Requirements Specifications) form
  - User stories
  - Templates
  - Structured Natural Language

- Using models (last week)

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"
- 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

(Cohn)

User Stories

- Three aspects
  - A written description as a reminder (often on a card)
  - 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 (acceptance criteria)
- Ideally they are written by the customers
- Use common, non-technical language
- User stories are not contractual obligations
  - The tests should serve this purpose

(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 >*
- Examples:
  - As a bike renter, I want to rent a bike, so that I can see more of the city
  - As a bike renter, I want free choice to select a specific bike to rent, so that I can rent a nice bike
  - As a bike renter, I want a clear notification of the duration of my rental, so I can avoid late charges
  - As a manager, I want reports of bike rentals per week, so I can understand how well the business is operating
  - As a maintenance personnel, I want to know which bike rental locations are low on bikes, so I can make sure there are always bikes to rent

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 (Acceptance Criteria)

- "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)

Epics

- They can be decomposed or refined, but in separate stories, not in the typical requirement hierarchy (Cohn)
- 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"
- 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."

Software Requirement Specification (SRS)

- As in a collection of requirements in a document
- ISO/IEC/ IEEE 29148 - **Systems and software engineering — Life cycle processes — Requirements engineering**
- Focuses on System Requirements, but can include user requirements, and other useful information
- Usually long
- Rather formal, acts as a contract between the requirements analyst/business/users and the architect/developers
- (Usually very boring)
- Not as common now (although they are still made and used)
- Agile backlogs of user stories + other agile processes are meant to replace this

SRS Sections (Example)

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)

(Some) SRSes for Bike Rental

1. The system shall allow a user to select an available bike from the bike rack to rent

   1. 1 The system shall allow the user to indicate how long they would like to rent the bike with options: 60 minutes, 120 minutes, 3 hours.

2. The system shall allow the user to return their bike to any available slot in any available bike rack

   2.1 The system should provide a user a confirmation that they have successfully returned the bike

3. The system shall allow frequent users to get a discount. When a user has rented a bike for the fourth time in a month, the price is reduced by 20%. Every rental after this until the end of the month is reduced by 20%.

SRS Requirements

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

- Examples for an online marking tool
  1. There must be a place for summary comments in all types of Rubrics/Marking Schemes
  2. The summary comments given by a Marker should be stored and potentially reused in later instances of the same assignments by the same Marker
  3. If a summary comment is reused, this comment should be editable, and the Marker should be allowed to add additional comments to it
  4. 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
  5. 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):
  1. A lift will only reverse direction when stopped at a floor
  2. The system will cycle the lift doors every time that a lift stops at a floor
  3. The lift must never be moved with the doors open
  4. Each lift should be used an approximately equal amount

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

(Bray)

Some Tips from IEEE

- Requirements are mandatory binding provisions and use 'shall'.
- 'Will' can be used to establish context or limitations of use. However, 'will' can be construed as legally binding, so it is best to avoid using it for requirements.
- Preferences or goals are desired, non-mandatory, non-binding provisions and use 'should'.
- Suggestions or allowances are non-mandatory, non-binding provisions and use 'may'.
- Non-requirements, such as descriptive text, use verbs such as ‘are', ‘is', and ‘was'. It is best to avoid using the term ‘must', due to potential misinterpretation as a requirement.
- Use positive statements and avoid negative requirements such as ‘shall not’.
- Use active voice: avoid using passive voice, such as 'shall be able to select'.

ISO/IEC/IEEE 29148:2011(E)

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 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

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

The Easy Approach to Requirements Syntax (EARS)*

The *Easy Approach to Requirements Syntax* (EARS) consists of a set of patterns for specific types of functional requirements and constraints

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

Requirements Quality

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) requirements to User Stories

User Story Quality (INVEST)

- 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

Characteristics of a Good SRS Requirement

- IEEE Standard 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: Necessary

- 1) Necessary
  - If an implementation is possible within the constraints of the program or product
  - The requirement defines an essential capability, characteristic, constraint, and/or quality factor.
  - If it is removed or deleted, a deficiency will exist, which cannot be fulfilled by other capabilities of the product or process.
  - 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: Avoids Design

- 2) Implementation Free
  - The requirement, while addressing what is necessary and sufficient in the system, avoids placing unnecessary constraints on the architectural design.
  - The objective is to be implementation independent.
  - The requirement states what is required, not how the requirement should be met.
  - Recall past lecture
  - "The system shall use a service-based architecture"
  - "The user interface shall contain a "register user" button on the first page"

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"

More on Unambiguity

- The following are types of unbounded or ambiguous terms:
  - Superlatives (such as 'best', 'most')
  - Subjective language (such as 'user friendly', 'easy to use', 'cost effective')
  - Vague pronouns (such as 'it', 'this', 'that')
  - Ambiguous adverbs and adjectives (such as 'almost always', 'significant', 'minimal')
  - Open-ended, non-verifiable terms (such as 'provide support', 'but not limited to', 'as a minimum')
  - Comparative phrases (such as 'better than', 'higher quality')
  - Loopholes (such as 'if possible', 'as appropriate', 'as applicable')
  - Incomplete references (not specifying the reference with its date and version number; not specifying just the applicable parts of the reference to restrict verification work)
  - Negative statements (such as statements of system capability not to be provided)

Characteristics of a Good Requirement: Consistent

- 4) Consistent
  - The requirement is free of conflicts with other requirements.
  - Examples
    - The system will cycle the lift doors every time that a lift stops at a floor
    - The system will only open the doors when there is a user request
    - The system shall allow the user to return their bike to any available slot in any available bike rack
    - Users can only return their bikes to racks within 5 km of the original bike rack from which the bike was rented

Characteristics of a Good Requirement: Consistent

- 4) Consistent
  - The requirement is free of conflicts with other requirements.
  - Examples
    - The system will cycle the lift doors every time that a lift stops at a floor
    - The system will only open the doors when there is a user request
    - The system shall allow the user to return their bike to any available slot in any available bike rack
    - Users can only return their bikes to racks within 5 km of the original bike rack from which the bike was rented

Characteristics of a Good Requirement: Complete

- 5) Complete
  - The stated requirement needs no further amplification because it is measurable and sufficiently describes the capability and characteristics to meet the stakeholder's need.
  - The requirement is not missing information
  - The user can enter and modify race details
  - The system shall allow frequent users to get a discount

Characteristics of a Good Requirement: Singular

- 6) Singular
  - The requirement statement includes only one requirement with no use of conjunctions.

1. All input to the system is to be entered by the user, the user can enter and modify boat details, and the user can enter and modify race details

Characteristics of a Good Requirement: Feasible

- 7) 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 second
  - 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 Good Requirements: Traceable

- 8) 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 previous lecture 6)

Characteristics of a Good Requirement: Verifiable

- 9) 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."
    - What is acceptable?
    - "The product shall return (file) search results in within 0.7 seconds."

(Li et al.)

Characteristics of Good Requirements (a Set): Complete

- 1) 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
  - 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: Consistent

- 2) Consistent (Berenbach et al.)
  - The set of requirements does not have individual requirements which are contradictory.
  - Requirements are not duplicated.
  - The same term is used for the same item in all requirements.

Characteristics of Good Requirements: Affordable

- 3) Affordable
  - The complete set of requirements can be satisfied by a solution that is obtainable/feasible within life cycle constraints (e.g., cost, schedule, technical, legal, regulatory).

Characteristics of Good Requirements: Bounded

- 4) Bounded
  - The set of requirements maintains the identified scope for the intended solution without increasing beyond what is needed to satisfy user needs.
  - Don’t go outside the defined scope.

Characteristics of Good Requirements: Summary

- For a requirement
  1. Necessary
  2. Implementation Free
  3. Unambiguous
  4. Consistent
  5. Complete
  6. Singular
  7. Feasible
  8. Traceable
  9. Verifiable
- For a set of requirements
  1. Complete
  2. Consistent
  3. Affordable
  4. Bounded

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

Modifiable: Potential problems if R5 is changed or moved

Inconsistent, contradicts R.11

User Story vs. SRS Requirements Quality

User Story Quality

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

SRS Requirements Quality

- Necessary
- Implementation Free
- Unambiguous
- Consistent
- Complete
- Singular
- Feasible
- Traceable
- Verifiable
- Affordable
- Bounded

Virtual Museum/Gallery

- I'll create a short video writing some requirements
  - SRS
  - User Stories

Questions?

Readings

- In Canvas
  - (for extra help)
  - Writing Good Requirements, RE'17 Tutorial
  - 29148-2011.pdf The IEEE standard

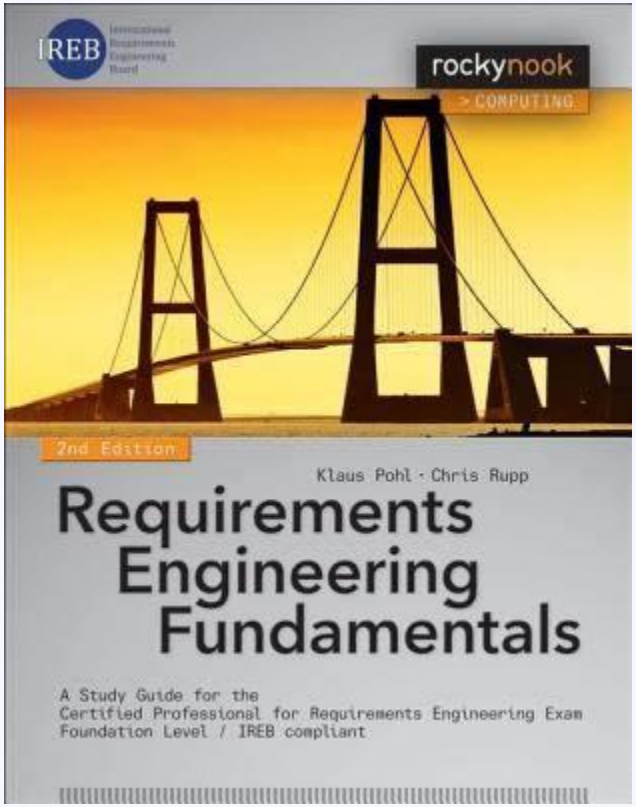
Lecture Sources

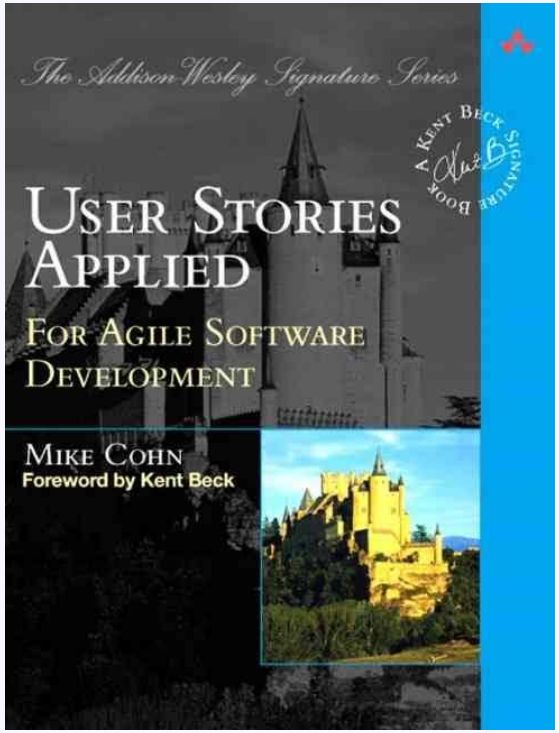
- IREB (International Requirements Engineering Board)
  - https://www.ireb.org/en/downloads/
- Requirements Engineering (CSC340) S. Easterbrook, J. Campbell
  - http://www.cs.toronto.edu/~sme/CSC340F/
- Requirements Engineering for Software and Systems, Second Edition, By Phillip A. Laplante Kilicay-Ergin, Nil, and Phillip A. Laplante. "An online graduate requirements engineering course." *IEEE Transactions on Education* 56.2 (2013): 208-216.
- Dick, Jeremy, Elizabeth Hull, and Ken Jackson. *Requirements engineering*. Springer, 2017.
- "What is Requirements Engineering?" the draft chapter 1 and "What are Requirements?" the draft chapter 2 of Fundamentals of Requirements Engineering (FoRE), S. Easterbrook, 2004.
- Bray, Ian K. *An introduction to requirements engineering*. Pearson Education, 2002.

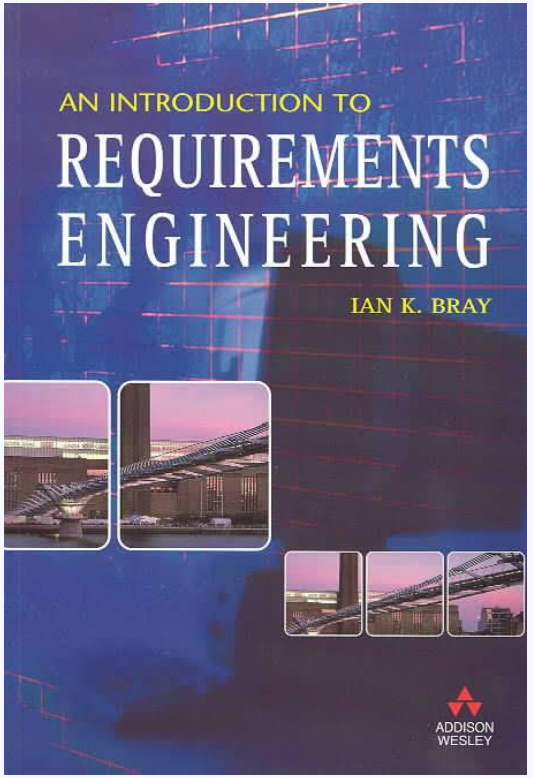
Lecture Sources

- Wilson, Tom. "Software failure: management failure. Amazing stories and cautionary tales: S. Flowers Wiley, Chichester, New York,(1996) 197 pp£ 19.99 ISBN 0 171-95113-7." *International Journal of Information Management* 17.5 (1997): 387.
- Macaulay, Linda A. *Requirements engineering*. Springer Science & Business Media, 2012.
- Pohl K (1993) The three dimensions of requirements engineering. CAiSE'93, Paris, France
- Jackson, Michael, and Pamela Zave. "Deriving specifications from requirements: an example." *Software Engineering, 1995. ICSE 1995. 17th International Conference on*. IEEE, 1995.

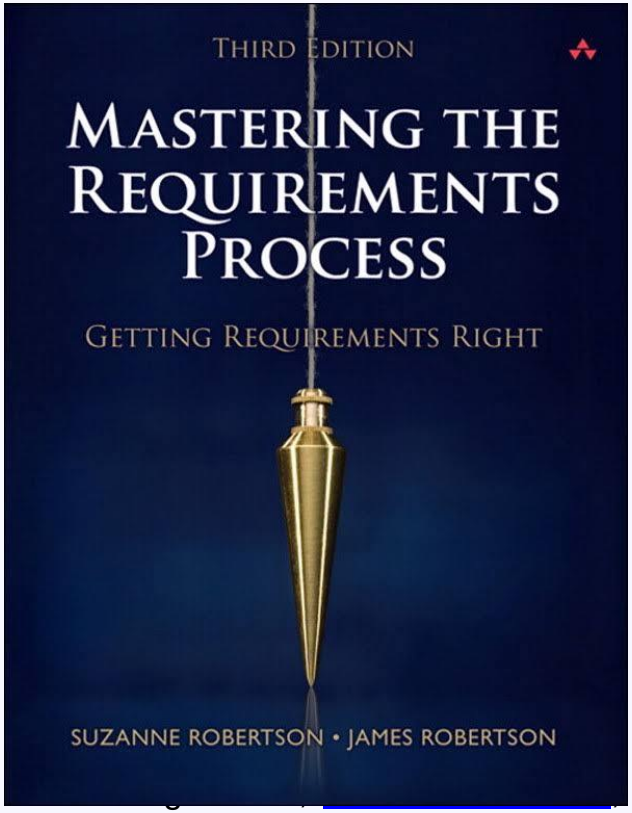
Lecture Sources

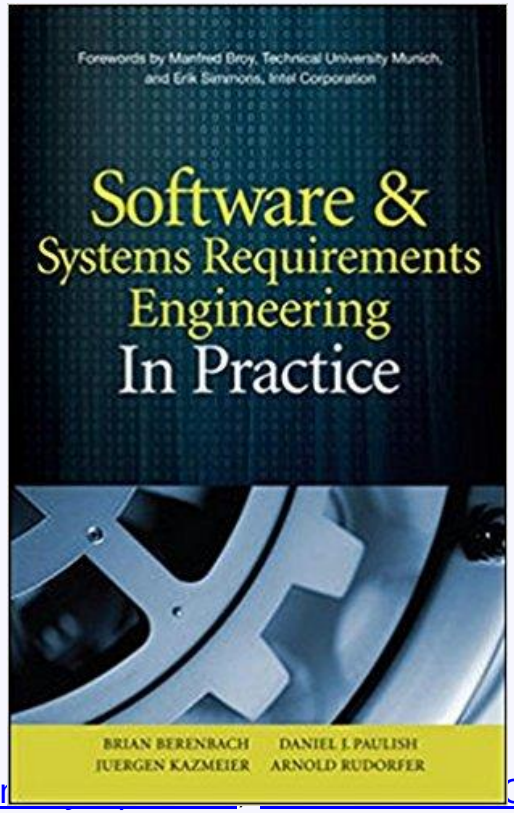






- https://www.gilb.com/





[Joh]Guizzardi, Giancarlo Guizzardi, Alexander Borgida, Lin Liu:

Non-functional requirements as qualities, with a spice of ontology. RE 2014: 293-302