Lecture 1

Introduction

SOEN 6481, Winter 2016/17

René Witte
Department of Computer Science
g
ty

Slides Credit

Axel van Lamsweerde

- Copyright © 2009 Wi[redacted]

Outline

Outline

What is requirements engineering ?

- Set of activities producing the requirements on a software-intensive system
  - elicitation, evaluation, specification, analysis, evolution management
  - system objectives, functionalities, target qualities, constraints, assumptions
- **Requirements quality assurance** is a key concern for software quality assurance

René Witte
Concordia University

Requirements engineering (RE), roughly ...

- Identify & analyze problems with an existing system (system-**as-is**),
- Identify & evaluate objectives, opportunities, options for new system (system-**to-be**),
- Identify & define functionalities of, constraints on, responsibilities in system-to-be,
- Specify & organize all of these in a **requirements document** to be maintained throughout system evolution

System = software + environment
(people, devices, other software)

Example: transportation between airport terminals

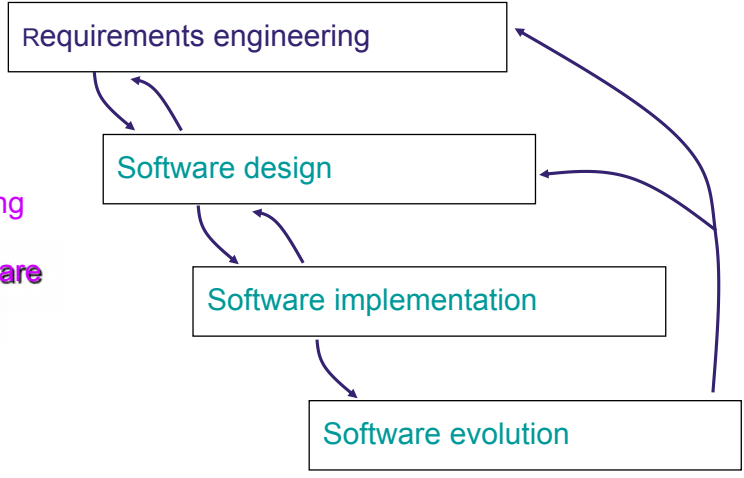
- Problem (system-**as-is**):
  - passengers frequently missing flight connections among different terminals; slow & inconvenient transportation
  - number of passengers regularly increasing
- Objectives, options (system-**to-be**):
  - support high-frequency trains between terminals
  - with **or** without train drivers ?
- Functionalities, constraints:
  - software-based control of train accelerations, doors opening etc. to achieve *prompt* and *safe* transportation
- RE del

René Witte
Concordia University

Requirements in the software lifecycle

Getting the *right* system

Getting the software *right*



Why requirements engineering ?

- RE is critical
  - Major cause of software failure
    Requirements-related errors are the most numerous, persistent, expensive, dangerous
  - Severe consequences: cost overruns, delivery delays, dissatisfaction, degradations, accidents, ...
  - RE has multiple impact: legal, social, economical, technical
  - Certification issues
- RE is hard

What makes RE hard ?

- Broad scope
  - multiple system versions: *as-is*, *to-be*, *to-be-next*
  - hybrid environment:
    human organizations, policies, regulations
    devices, physical laws
- Multiple concerns
  - functional, quality, development concerns
- Multiple abstraction levels
  - strategic objectives, operational details

René Witte

What makes RE hard ? (2)

- Multiple stakeholders
  - with different background
  - with different interests and conflicting viewpoints
- Multiple intertwined tasks during iterative elicitation-evaluation-specification-consolidation
  - conflict management
  - risk management
  - evaluation of alternatives, prioritization
  - quality assurance
  - change anticipation

Outline

Setting the scene: outline (1)

- **Why engineer requirements?**
  - The requirements problem: facts, data, citations
  - Role and stakes of Requirements Engineering
- Obstacles to good RE practice
- Agile development and RE

The requirements problem: facts, data, citations

Bell&Thayer '76

- Poor requirements are ubiquitous ...

  "Requirements need to be engineered and have continuing review and revision"

Boehm '81

- Prohibitive cost of late correction ...

  "Up to 200 x cost of early correction"

Brooks '87

- RE is hard & critical ...

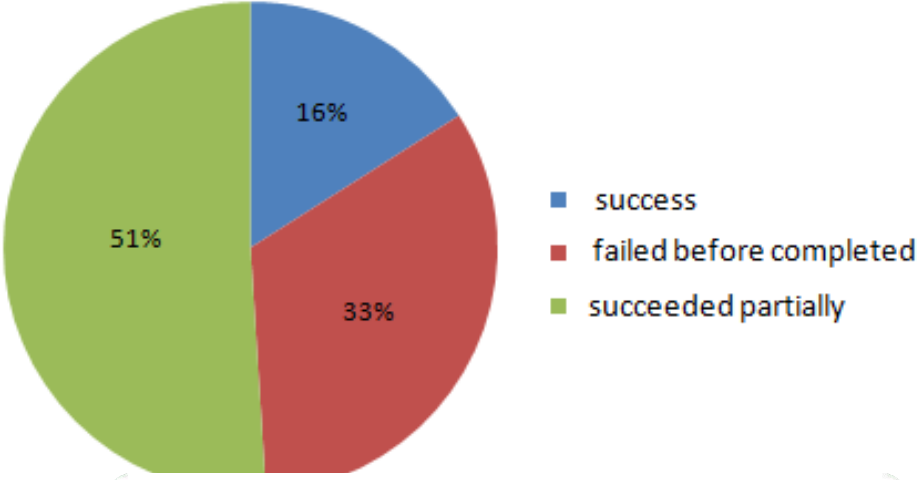
  "Hardest, most important function of SE is the iterative *extraction* & *refinement* of requirements"

René Witte

René Witte
Concordia University

The requirements problem: Standish report, 1995

Survey of 350 US companies, 8000 projects



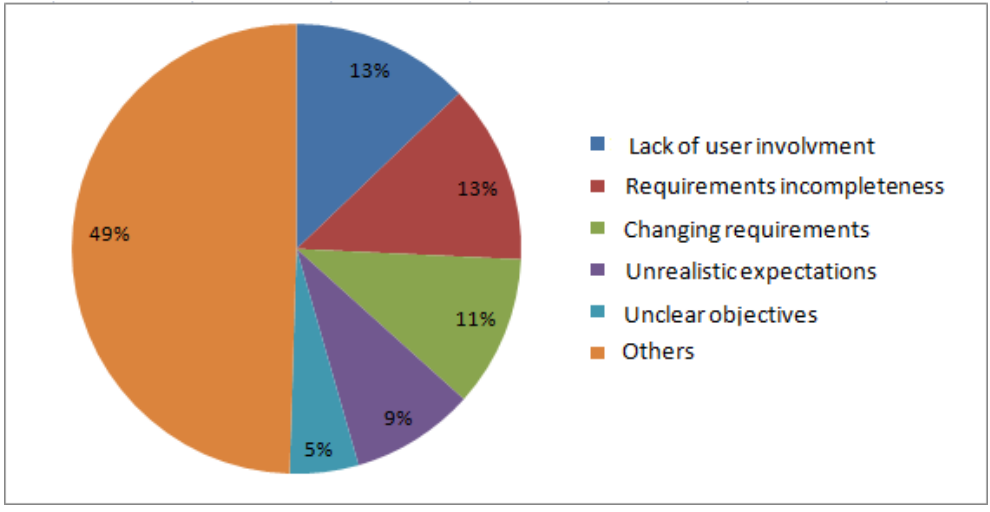
(partial succes

Reasons for project failure?. . .

René Witte
Concordia University

Major Cause: Requirements Engineering

Major source of failure: poor requirements engineering (ca. 50% responses)



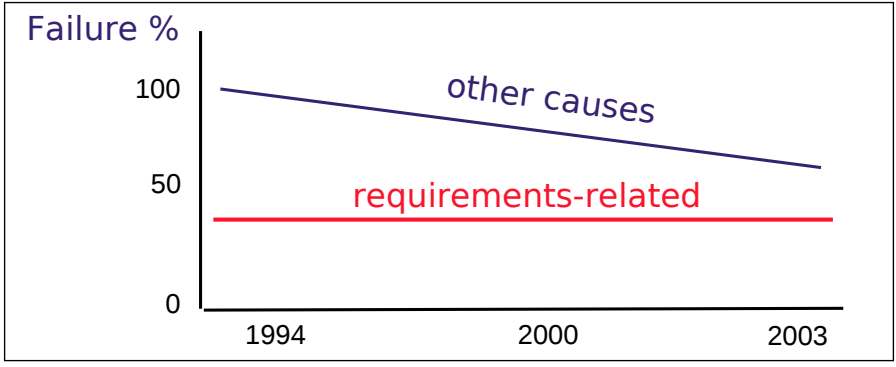
The requirements problem: European survey, 1996

- Coverage: 3800 EUR organizations, 17 countries
- Main software problems perceived to be in...
  - requirements specification
    - > 50% responses
  - requirements evolution management
    - 50% responses

[European Software Institute, 1996]

Improvements?

What are the problems?



[J. Maresco, IBM developersWork, 2007]

Requirements-related errors are ...

- the most **numerous**
  - ± 40% of software errors
- the most **persistent**
  - found very late, often after software delivery
- the most **expensive**
  - cost ... 5x more if fixed during design
    10x more if fixed during implementation
    20x more if fixed during integration testing
    200x more if fixed after delivery
  - account for 66% of software error costs

[Boehm, Jones, Lutz, Hooks & Farry, ...]

René Witte
Concordia University

Requirements-related errors can be dangerous

- US Aegis/Vincennes (1988): shooting of IranAir airbus
  - Missing timing between 2 threat events in requirements on alarm software
- Patriot anti-missile system (1st Gulf war)
  - Hidden assumption on maximum usage time
- London Ambulance System (1993): fatal delays
  - Wrong assumptions on crew behavior, ambulance localization system, radio communication, ...
- Boeing 757 crash, Cali (1995)
  - Autopilot 's wrong timing/localization assumption on flap extension point
- Cf. ACM RISKS Digest Forum website

René Witte
Concordia University

Example: inadequate domain property in A320 braking logic

SofReq: reverse = 'on' iff WheelPulses = 'on'

ASM: reverse = 'on' iff ReverseThrustEnabled

WheelPulses = 'on' iff WheelsTurning

Dom: ~~MovingOnRunway iff WheelsTurning~~

SysReq: ReverseThrustEnabled iff MovingOnRunway

Warsaw crash: plane moving on waterlogged runway with no wheels turning (aquaplaning)

René Witte
Concordia University

Role and stakes of RE

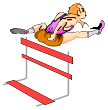
- **Technical** impact
  - on many software-related artefacts (as seen before)
- **Managerial** impact
  - basis for communication among parties and for project management
- **Legal** impact
  - contractual commitment client-provider-subcontractors
- Impact on **certification**
  - Mastered RE process required by many quality standards & certification authorities

René Witte
Concordia University

Role and stakes of RE (2)

- Impact on **economy**, security, and safety
  - **Cost** and **consequences** of errors in requirements on the software-to-be, assumptions about its environment
- **Social** impact
  - *from* user satisfaction
    to degradation of working conditions
    to system rejection

René Witte
Concordia University



Obstacles to good RE practice

- RE efforts often spent without guarantee of project contract being concluded
- Pressure on tight schedules, short-term costs, catching up on technology
- Too little work available on RE economics
  - Lack of quantitative data on RE benefits & cost savings
  - Progress in RE process is harder to measure than in design, implementation
- RDs are sometimes felt ...
  - big, complex, to be quickly outdated
  - too far away from the executable product customers are paying for

René Witte
Concordia University

Outline

René Witte
Concordia University

The problem world and the machine solution

- To make sure a software solution "correctly" solves some real-world problem, we must first fully **understand** and **define** ...
  - **what problem** needs to be solved in the real world
  - the **context** in which the problem arises
- Example: car control
  - **Problem**: manual handbrake release can be inconvenient in certain situations
  - **Context**: car driving, braking, driver 's intent, safety rules, ...

René Witte
Concordia University

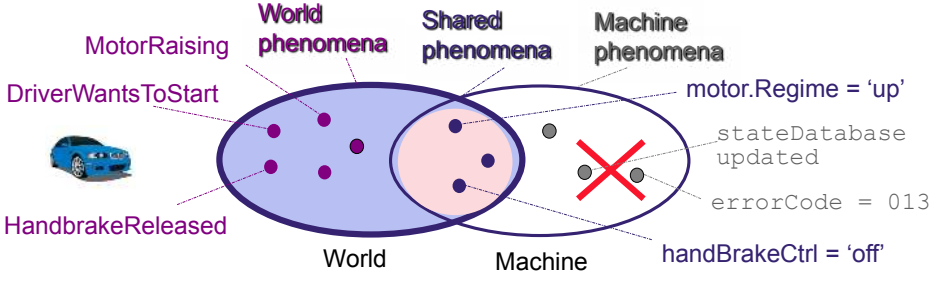
The problem world and the machine solution (2)

- **World**: problematic part of the real-world, made of
  - human components: organization units, staff, operators, ...
  - physical components: devices, legacy software, mother Nature, ...
- **Machine**: what needs to be installed to solve the problem
  - software to be developed and/or purchased
  - hardware/software implementation platform, associated input/output devices (e.g. sensors & actuators)
- Requirements engineering (RE) is concerned with ...
  - the desired machine's **effect on the problem world**
  - the **assumptions** and **relevant properties** about this world

René Witte
Concordia University

The problem world and the machine solution (3)

- The world and the machine have their own phenomena while sharing others
- RE is solely concerned with **world** phenomena, including shared ones [Jackson95]
  - unlike software design, concerned with machine phenomena

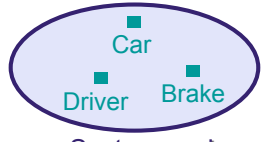


René Witte
Concordia University

The problem world involves two system versions

- **System**: set of interacting components structuring the problem world
- System-**as-is**: system as it exists before the machine is built into it
- System-**to-be**: system as it should be when the machine will operate into it

Concepts, phenomena, rules about car handbraking



System-**as-is**

Concepts, phenomena, rules about automated handbraking

René Witte

Concordia University

RE: a preliminary definition

Coordinated set of **activities** ...

- for **exploring**, **evaluating**, **documenting**, **consolidating**, **revising** and **adapting** the **objectives**, **capabilities**, **qualities**, **constraints** & **assumptions** on a software-intensive **system**
- based on **problems** raised by the system-**as-is** and **opportunities** provided by new technologies

What others said ...

Ross'77

- Requirements definition must say ...
  - **why** a new system is needed, based on current or foreseen conditions,
  - **what** system features will satisfy this context,
  - **how** the system is to be constructed

Zave'97

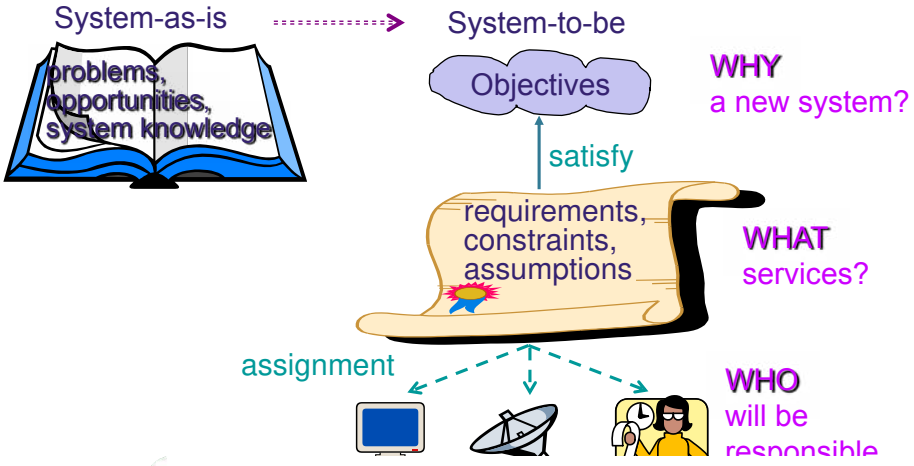
- RE is concerned with the real-world **goals** for, **functions** of, **constraints** on software systems; and with their
  - **link** to precise **specifications of software behavior**,
  - **evolution** over time and families

System requirements vs. software requirements

- **Software-to-be**: software to be developed - part of the machine, component of the system-to-be
- **Environment**: all other components of the system-to-be, including people, devices, pre-existing software, etc.
- **System requirements**: what the *system*-to-be should meet; formulated in terms of phenomena in the environment
  "The handbrake shall be released when the driver wants to start."
- **Software requirements**: what the *software*-to-be should meet on its own; formulated in terms of phenomena **shared** by the software and the environment
  "The software output variable *handBrakeCtrl* shall have the value *off* when the software input variable *motorRegime* gets the value *up*."

René Witte
Concordia University

The scope of RE: the *WHY, WHAT, WHO* dimensions



René Witte

The WHY dimension

- Identify, analyze, refine the system-to-be's **objectives**
  - to address analyzed deficiencies of the system-as-is
  - in alignment with business objectives
  - taking advantage of technology opportunities
- Example: airport train control
  "Serve more passengers"
  "Reduce transfer time among terminals"
- Difficulties
  - Acquire domain knowledge
  - Evaluate alternative options (e.g. alternative ways of satisfying the same objective)
  - Match problems-opportunities, and evaluate these: implications, associated risks
  - Handle conflicting objectives

René Witte
Concordia University

The WHAT dimension

- Identify & define the system-to-be's **functional services** (software services, associated manual procedures)
  - to satisfy the identified objectives
  - according to quality constraints: security, performance, ...
  - based on realistic assumptions about the environment
- Example: airport train control

  "Computation of safe train accelerations"

  "Display of useful information for passengers inside trains"
- Difficulties
  - Identify the right set of features
  - Specify these precisely for understanding by all parties
  - Ensure backward traceability to system objectives

René Witte
Concordia University

The WHO dimension

René Witte
Concordia University

- Assign responsibilities for the objectives, services, constraints among system-to-be components
  - based on their capabilities and on the system's objectives
  - yielding the software-environment boundary
- Example: airport train control
  - "Safe train acceleration" ... under direct responsibility of software-to-be (driverless option) *or* of driver following software indications ?
  - "Accurate estimation of train speed/position" ... under responsibility of tracking system *or* of preceding train ?
- Difficulties
  - Evaluate alternative options to decide on the right degree of automation

Setting the scene: outline

- What is Requirements Engineering?
  - The problem world & the machine solution
  - The scope of RE: the WHY, WHAT and WHO dimensions
  - Types of statements involved: descriptive vs. prescriptive
  - Categories of requirements: functional vs. non-functional
  - The requirements lifecycle: actors, processes, products
  - Target qualities and defects to avoid
  - Types of software projects
  - Requirements in the software lifecycle
  - Rel[illegible]

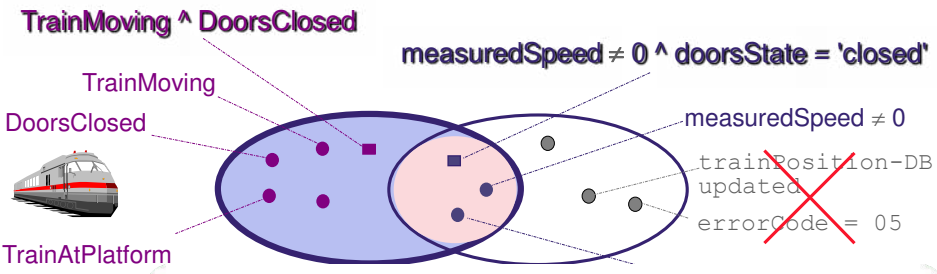
Statements may differ in mood

- **Descriptive** statements state system properties holding regardless of how the system should behave (indicative mood)
  - natural law, physical constraint, etc
  - e.g. "If train doors are closed, they are not open"
    "If the train's acceleration is positive, its speed is non-null"
- **Prescriptive** statements state desirable properties holding or not depending on how the system behaves (optative mood)
  e.g. "Doors shall always remain closed when the train is moving"
- Important distinction for RE:
  - prescriptive statements can be negotiated, weakened, replaced by alternatives
  - descriptive statements cannot

René Witte
Concordia University

Statements may differ in scope

- A RE statement may refer to phenomena ...
  - owned by the environment
  - or shared between the environment & the software-to-be: one **controls** phenomena **monitored** by the other, and resp.



René Witte
Concordia University

Types of statements: system requirements, software requirements

- **System requirement**: *prescriptive* statement refering to *environment* phenomena (not necessarily shared)
  - to be enforced by the software-to-be possibly together with other system components
  - formulated in a vocabulary understandable by all parties

    $\text{TrainMoving} \Rightarrow \text{DoorsClosed}$
- **Software requirement**: *prescriptive* statement refering to *shared* phenomena
  - to be enforced by the software-to-be solely
  - formulated in the vocabulary of software developers

    $\text{measuredSpeed} \neq 0 \Rightarrow \text{doorsState} = \text{'closed'}$

(A software req is a system req; the converse is not true)

René Witte

Types of statements: domain properties, assumptions, definitions

- **Domain property**: *descriptive* statement about problem world phenomena (holds regardless of any software-to-be)

  $\text{trainAcceleration} > 0 \Rightarrow \text{trainSpeed} \neq 0$

- **Assumption**: statement to be satisfied by the environment of the software-to-be
  - formulated in terms of environment phenomena
  - generally prescriptive (e.g. on sensors or actuators)

  $\text{measuredSpeed} \neq 0$ **iff** $\text{trainSpeed} \neq 0$

- **Definition**: statement providing a precise meaning to system concepts or auxiliary terms
  - no truth value

"me

René Witte

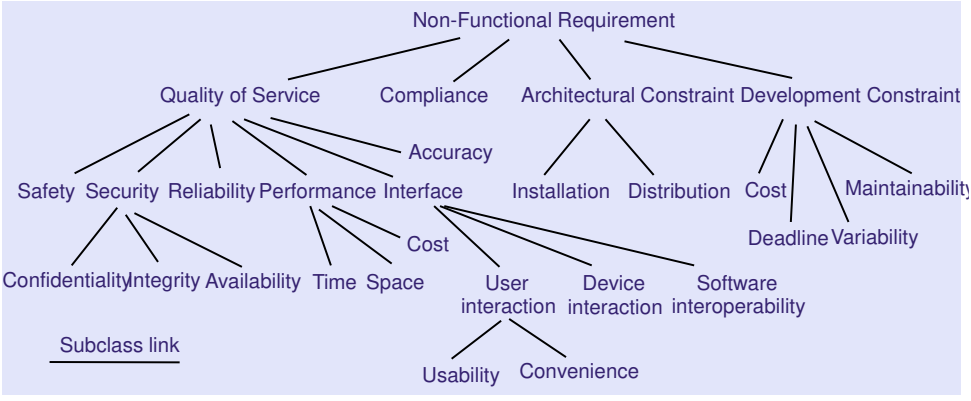
Categories of requirements

- **Functional requirements:** prescribe what services the software-to-be should provide
  - capture intended software effects on environment, applicability conditions
  - units of functionality resulting from software operations

  "The software shall control the acceleration of all trains"
- **Non-functional requirements:** constrain how such services should be provided
  - **Quality** requirements: safety, security, accuracy, time/space performance, usability, ...
  - Others: compliance, architectural, development reqs
  - To be made precise in system-specific terms

  "Acceleration commands shall be issued every 3 seconds to every train"

A taxonomy of non-functional requirements



- See definitions and examples in the book
- No clear-cut boundaries, possible overlaps
  - Functional/non-functional: e.g. functional reqs for firewall management are security-related
  - Nor rela

René Witte
Concordia University

Requirements taxonomies are helpful ...

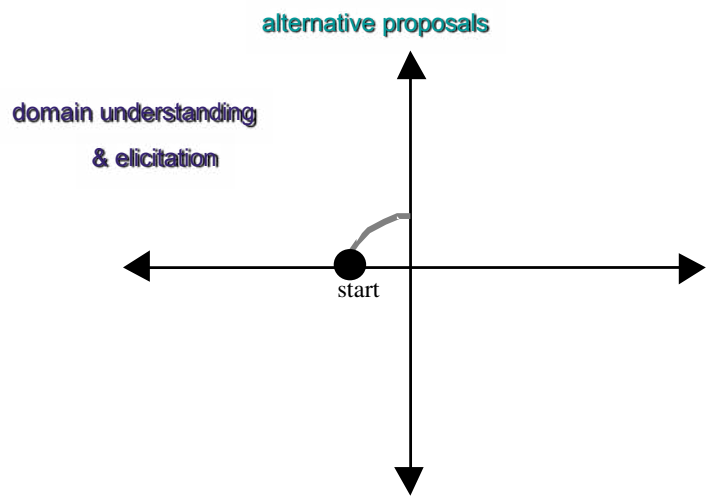
- More specific definition of what requirements are in specific classes
- More semantic characterization of requirements ...
  - prescribing **desired behaviors** e.g. many functional reqs
  - ruling out **inadmissible behaviors** e.g. many safety, security, accuracy reqs
  - indicating **preferred behaviors** e.g. soft, "ility" reqs
- Elicitation/analysis can be guided by taxonomy browsing
  - Is there any confidentiality req on information *X*?
  - Is there any accuracy req on information *Y*?
  - Is there any conflict between confidentiality and accountability reqs in my system?

René Witte
Concordia University

Setting the scene: outline

- What is Requirements Engineering?
  - The problem world & the machine solution
  - The scope of RE: the WHY, WHAT and WHO dimensions
  - Types of statements involved: descriptive *vs.* prescriptive
  - Categories of requirements: functional *vs.* non-functional
  - **The requirements lifecycle: actors, processes, products**
  - Target qualities and defects to avoid
  - Types of software projects
  - Requirements in the software lifecycle
  - Rel[illegible]

The RE process (1)



Domain understanding

- Studying the system-as-is
  - Business organization: structure, dependencies, strategic objectives, policies, workflows, operational procedures, ...
  - Application domain: concepts, objectives, tasks, constraints, regulations, ...
  - Strengths & weaknesses of the system-as-is
- Identifying the system **stakeholders**:
  - Groups or individuals affected by the system-to-be, who may influence its elaboration and its acceptance
  - Decision makers, managers, domain experts, users, clients, subcontractors, analysts, developers, ...

Products: Initial sections for preliminary draft proposal

René Witte
Concordia University

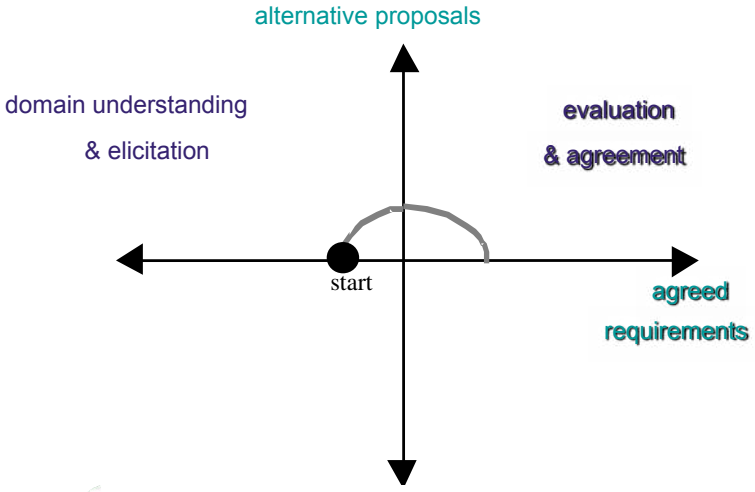
Requirements elicitation

Exploring the problem world ...

- Further analysis of problems with system-as-is: symptoms, causes, consequences
- Analysis of technology opportunities, new market conditions
- Identification of ...
  - improvement objectives
  - organizational/technical constraints on system-to-be
  - *alternative* options for satisfying objectives, for assigning responsibilities
  - scenarios of hypothetical software-environment interaction
  - requirements on software, assumptions on environment

Product: Additional sections for preliminary draft proposal

The RE process (2)



René Witte
Concordia University

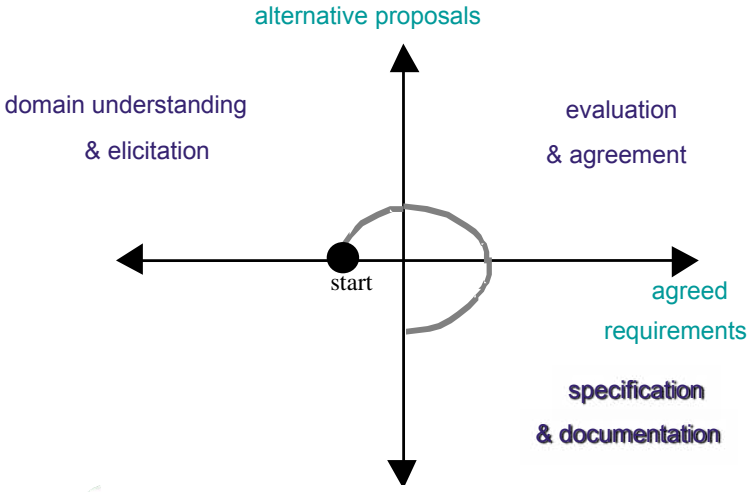
Evaluation & agreement

- Negotiation-based decision making ...
  - Identification & resolution of **conflicting** concerns
  - Identification & resolution of **risks** with proposed system
  - Comparison of **alternative options** against objectives & risks, and selection of preferred ones
  - Requirements **prioritization**: to resolve conflicts, address cost/schedule constraints, support incremental development

Product: Final sections of draft proposal documenting the selected/agreed objectives, requirements, asssumptions (incl. rationale for selected options)

René Witte
Concordia University

The RE process (3)



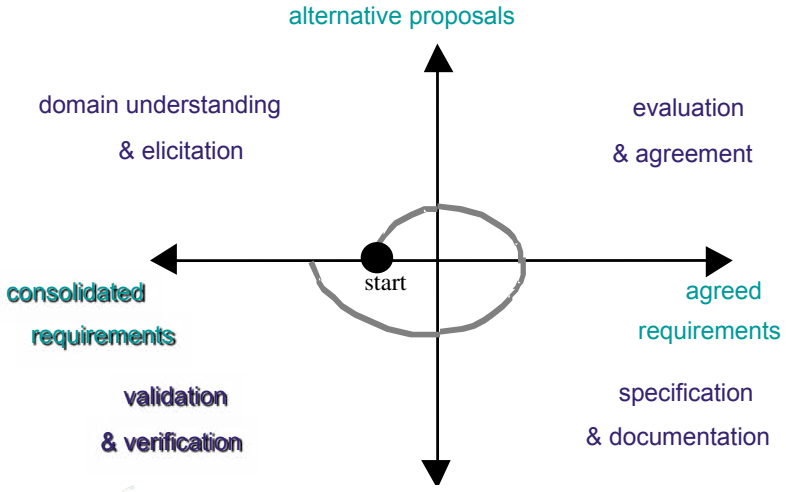
Specification & documentation

- Precise definition of all features of the agreed system
  - Objectives, concepts, relevant domain properties, system/software requirements, assumptions, responsibilities
  - Satisfaction arguments, rationale for options taken
  - Likely system variants & evolutions
  - Estimated costs
- Organization of these in a coherent structure
- Documentation in a form understandable by all parties

Resulting product: **Requirements Document** (RD)

René Witte
Concordia University

The RE process (4)



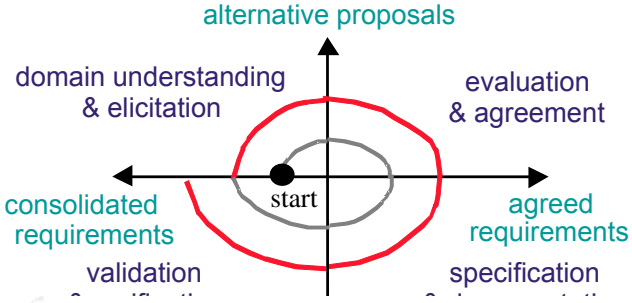
Requirements consolidation

- Quality assurance activity on RD ...
  - Validation: adequacy of RD items wrt real needs ?
  - Verification: omissions, inconsistencies ?
  - Checks for other target qualities (discussed next)
  - Fixing of errors & flaws
- **Products**: Consolidated RD
  Acceptance test data, prototype
  Development plan
  Project contract

René Witte
Concordia University

RE: an iterative process

- RE phases are ordered by data dependencies
- No strict sequencing: intertwining, overlap, backtracking
- Iterated cycles due to error corrections & **evolving needs**
  - during RE, during software development, after deployment



René Witte
Concordia University

Setting the scene: outline

- What is Requirements Engineering?
  - The problem world & the machine solution
  - The scope of RE: the WHY, WHAT and WHO dimensions
  - Types of statements involved: descriptive *vs.* prescriptive
  - Categories of requirements: functional *vs.* non-functional
  - The requirements lifecycle: actors, processes, products
  - **Target qualities and defects to avoid**
  - Types of software projects
  - Requirements in the software lifecycle
  - Relationship to other disciplines

Target qualities for RE process

- **Completeness** of objectives, requirements, assumptions
- **Consistency** of RD items
- **Adequacy** of requirements, assumptions, domain props
- **Unambiguity** of RD items
- **Measurability** of requirements, assumptions
- **Pertinence** of requirements, assumptions
- **Feasibility** of requirements
- **Comprehensibility** of RD items
- **Good structuring** of the RD
- **Modifiability** of RD items
- **Traceal**

René Witte

Types of RE errors & flaws: a wide palette

- Omission (critical error!)
- Contradiction (critical error!)
- Inadequacy (critical error!)
- Ambiguity (critical error!)
- Unmeasurability
- Noise, overspecification
- Unfeasibility (wishful thinking)
- Unintelligibility
- Poor structuring, forward reference, remorse
- Opacity

René Witte

The RE process may vary according to project type

- **Greenfield** *vs.* **brownfield** projects
- **Customer-driven** *vs.* **market-driven** projects
- **In-house** *vs.* **outsourced** projects
- **Single-product** *vs.* **product-line** projects

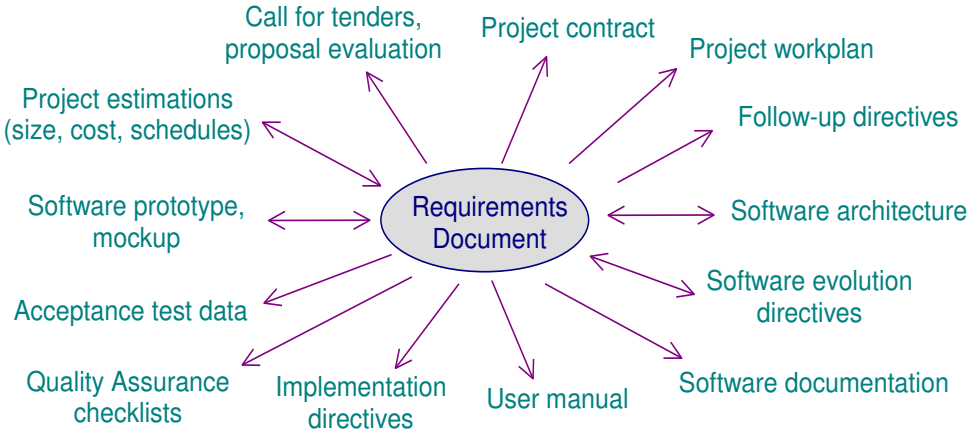
Variation factors ...

- Respective weights of elicitation, evaluation, documentation, consolidation, evolution
- Intertwining RE/design
- Respective weights of functional vs. non-functional reqs
- Types of stakeholder & developer involved
- Specific uses of the RD
- Use of specific techniques

René Witte
Concordia University

Requirements in the software lifecycle

- The RD impacts on many software artefacts



- RE, system design & software architecture design are inevitably intertwined

René Witte
Concordia University

RE has multiple connections with other disciplines

- Primarily with Software Engineering (SE)
- Other connections:
  - **Domain understanding & requirements elicitation**: system engineering, control theory, management science, organization theory, behavioral psychology, anthropology, AI knowledge acquisition
  - **Requirements evaluation & agreement**: multicriteria analysis, risk management, conflict management, negotiation theory
  - **Requirements specification, documentation & consolidation**: software specification, formal methods in SE
  - **Requirements evolution**: change management, configuration management in SE
  - **System modeling**: conceptual models in DB & MIS; task mo

Outline

Reading Material

Required

- [vL09, Chapter 1] (Setting the Scene)

Supplemental

- [LW03, Foreword, Chapters 1, 2] (Introduction to RE)
- [Lar05, Chapter 1] (Applying UML and Patterns Introduction)

Further Reading

- [Poh10, Chapters 1, 2] (Motivation, Requirements)

References

[Lar05] Craig Larman.
Applying UML and Patterns.
Prentice Hall, third edition, 2005.

[LW03] Dean Leffingwell and Don Widrig.
Managing Software Requirements: A Use Case Approach.
Addison-Wesley, 2003.
Available online at http://clues.concordia.ca/record=b2529323.

[Poh10] Klaus Pohl.
Requirements Engineering: Fundamentals, Principles, and Techniques.
Springer-Verlag Berlin Heidelberg, 2010.

[vL09] Axel van Lamsweerde.
Requirements Engineering: From System Goals to UML Models to Software Specifications.
John Wiley & Sons, 2009.