Requirements Engineering

Acknowledgement

Slides on RE are mostly inherited from Dr. Emmanuel Letier, UCL

1

Introduction to RE

2

Objectives

- Understand state-of-the-art for research and practice in requirements engineering
  - Role of RE in software and systems engineering
  - Fundamental concepts and techniques
  - Current notations, methods, processes and tools
- Gain practical experience in selected RE techniques
  - Especially goal-oriented and object-oriented modelling techniques

3

Introduction

- What is Requirements Engineering?
- Fundamental Concepts of Requirements Engineering
- Target Qualities of Requirements Documents

4

What is RE?

Not a phase or stage!

Communication is as important as the analysis

Quality means fitness for purpose. Cannot say anything about quality unless you understand the purpose

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

Designers need to know how and when the system will be used

Requirements are partly about what is needed ...

... and partly about what is possible

Need to identify all the stakeholders, not just users and costumers

5

(slide: Steve Easterbrook)

The Voice of the Ancients

- Poor requirements are ubiquitous ...

  "requirements need to be *engineered* and have continuing review & revision"

  (Bell & Thayer, empirical study, 1976)

- RE is hard & critical ...

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

  (F. Brooks, 1987)

- Prohibitive cost of late correction ...

  "up to 200 x cost of early correction" (Boehm, 1981)

6

Revisiting the RE problem

- Survey of US software projects by Standish Group

	1994	1998
Successful	16%	26%
Challenged	53%	46%
Cancelled	31%	28%

- Perceived causes of successes and failures (Top 3)

	Successful	Challenged	Cancelled
1.	User involvement	Lack of user input	Incomplete reqts
2.	Executive management support	Incomplete reqts	Lack of user input
3.	Clear statement of reqts	Changing reqts	Lack of resources

7

Factors that cause a project to be challenged

1.	**Lack of User Input**	12.8%
2.	**Incomplete Requirements & Specifications**	12.3%
3.	**Changing Requirements & Specifications**	11.8%
4.	Lack of Executive Support	7.5%
5.	Technology Incompetence	7.0%
6.	Lack of Resources	6.4%
7.	**Unrealistic Expectations**	5.9%
8.	**Unclear Objectives**	5.3%
9.	Unrealistic Time Frames	4.3%
10.	New Technology	3.7%
	Other	23.0%

Anecdotal but gives indication of perceived problems of software development

8

Factors that cause a project to be cancelled

1.	**Incomplete Requirements**	13.1%
2.	**Lack of User Involvement**	12.4%
3.	Lack of Resources	10.6%
4.	**Unrealistic Expectations**	9.9%
5.	Lack of Executive Support	9.3%
6.	**Changing Requirements & Specifications**	8.7%
7.	Lack of Planning	8.1%
8.	Didn't Need It Any Longer	7.5%
9.	Lack of IT Management	6.2%
10.	Technology Illiteracy	4.3%
	Other	9.9%

9

Factors that cause a project to be successful

1.	User Involvement	15.9%
2.	Executive Management Support	13.9%
3.	Clear Statement of Requirements	13.0%
4.	Proper Planning	9.6%
5.	Realistic Expectations	8.2%
6.	Smaller Project Milestones	7.7%
7.	Competent Staff	7.2%
8.	Ownership	5.3%
9.	Clear Vision & Objectives	2.9%
10.	Other	13.9%

10

Similar results in other surveys ...

- Survey of 3800 *EU* organizations, 17 countries

main software problems are in...

– requirements specification
  > 50% responses

– requirements management
  50% responses

(European Software Institute, 1996)

11

What makes RE so Complex ? (1)

- Broad scope
  - composite systems: human organizations + physical devices + software components
  - more than one system: system-as-is, alternative proposals for system-to-be, system evolutions, product family
  - multiple abstraction levels: high-level goals, operational details

12

(slide adapted from A. van Lamsweerde)

What makes RE so Complex ? (2)

- Multiple concerns
  - functional, quality, development
  - hard and soft concerns

→ *conflicts*

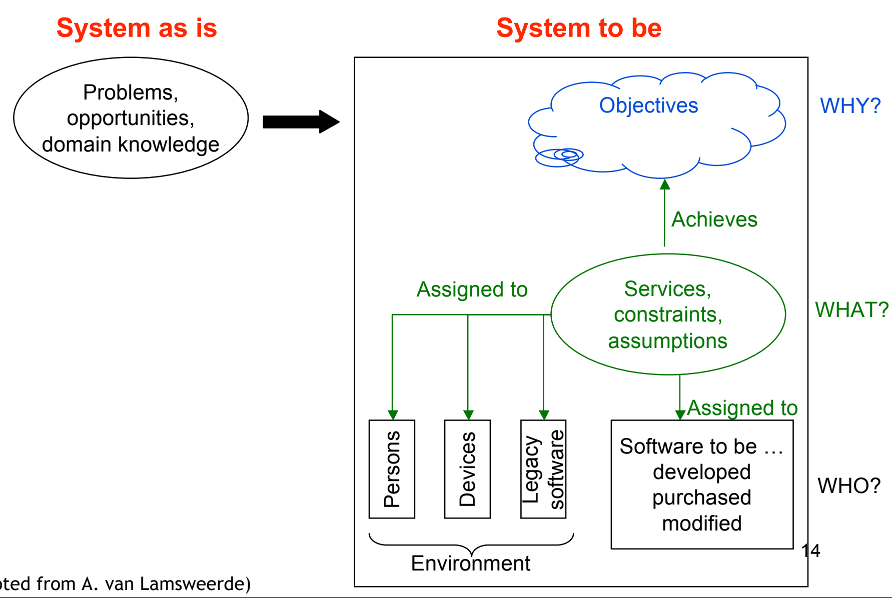
- Multiple stakeholders with different background
  - clients, users, domain experts, developers, ...

→ *conflicts*

13

(slide adapted from A. van Lamsweerde)

The dimensions of RE



14

(adapted from A. van Lamsweerde)

WHY dimension

- Understand SAI and S2B
- Analyze alternative ways satisfying the objectives in S2B
- Gain thorough understanding of application domain and opportunities provided by new technologies
- *Involves multiple parties*

WHAT dimension

- Addresses the functional services the S2B must provide wrt the objectives identified along the WHY dimension
  – services
  – constraints
  – assumptions

WHO dimension

- Assigns responsibilities for achieving the identified objectives
- Notice that the boundary between system and environment is not defined a-priori

What do requirements engineers do? (1)

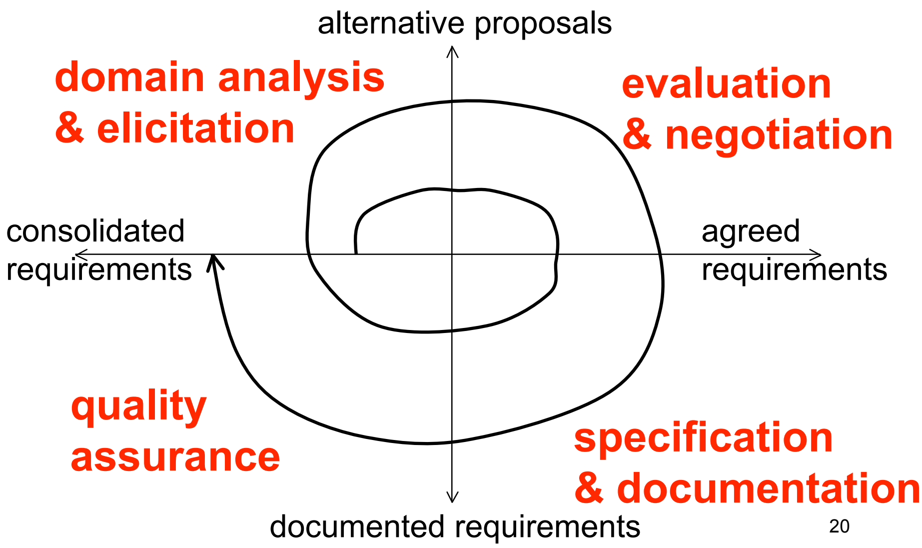
- Eliciting information
  - project objectives, context and scope
  - domain knowledge and requirements
- Modelling and analysis
  - goals, objects, use cases, scenarios, ...
- Communicating requirements
  - analysis feedback, SRS document, system prototypes, ...

What do requirements engineers do? (2)

- Negotiating and agreeing on requirements
  - handling conflicts and risks
  - helping in rqmts selection and prioritization
- Managing and evolving requirements
  - managing rqtms during development
    - backward and forward traceability
  - managing reqts changes and their impacts

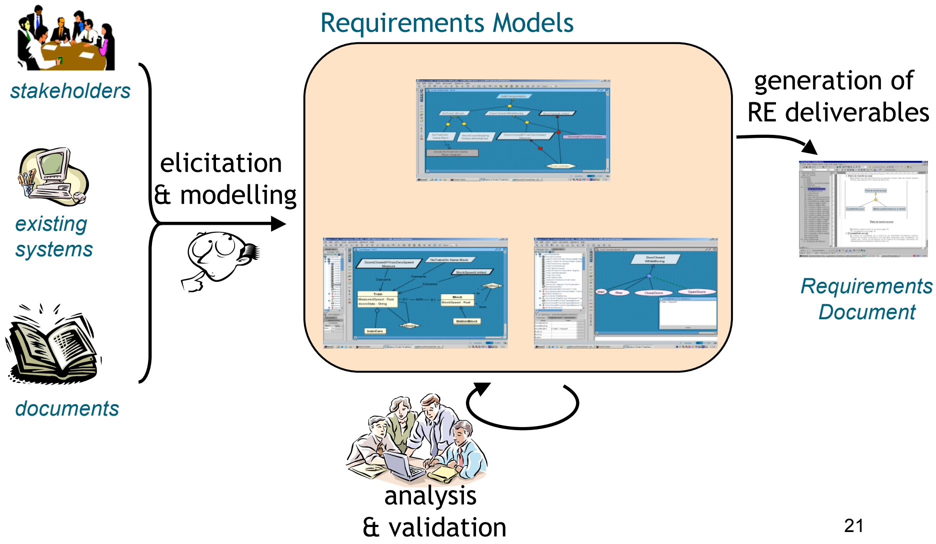
19

A RE lifecycle



20

The central role of requirements models



So far….

- Definition of requirements engineering
- Importance and difficulties of RE in system development
- Requirements engineering activities
- Central role of requirements modelling

Part 2: Fundamental Concepts of RE

The World and the Machine
(M. Jackson & P. Zave, 1995)

- Terminology
  - The machine = the portion of system to be developed
    typically, software-to-be + hardware
  - The world (a.k.a the environment) = the portion of the real-world affected by the machine
- The purpose of the machine is always in the world

  Examples
  - An Ambulance Dispatching System
  - A Banking Application
  - A Word Processor
  - ...

World phenomena

- Requirements engineering is concerned with phenomena occurring in the world

  E.g. for an ambulance dispatching system
  - the occurrences of incidents
  - the report of incidents by public calls
  - the encodings of calls details into the dispatching software
  - the allocation of an ambulance
  - the arrival of an ambulance at the incident location

  as opposed to phenomena occurring inside the machine
  - the creation of a new object of class Incident
  - the update of a database entry

⇒ ***Requirements models are models of the world***

25

Shared phenomena

- Some world phenomena are shared with the machine. Shared phenomena can be
  – *controlled by* the world and *observed by* the machine,

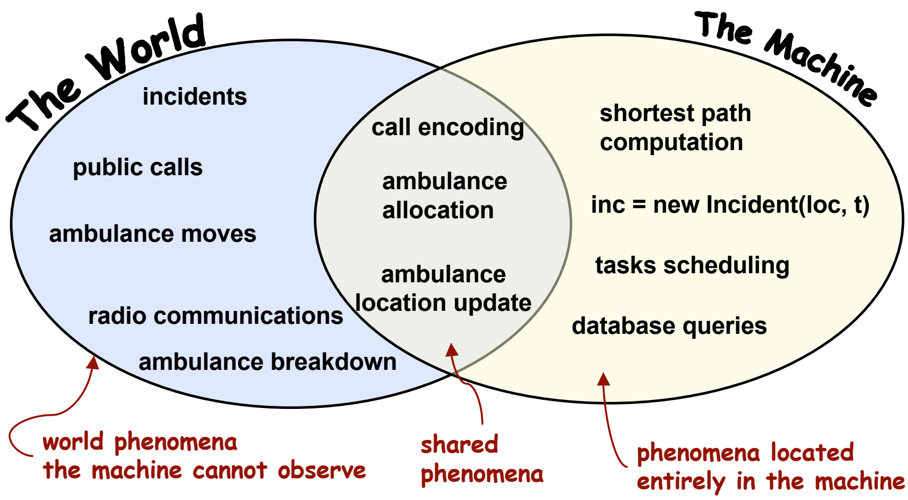
    E.g. the encodings of calls details into the dispatching software

  – or *controlled by* the machine and *observed by* the world

    E.g. the allocation of an ambulance to an incident

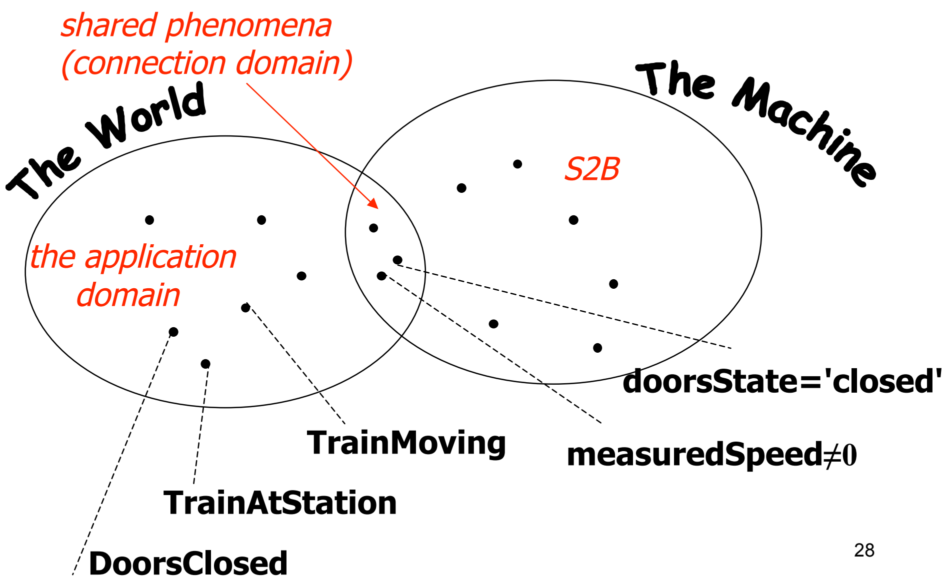
26

Example - An Ambulance Dispatching System



27

Example: Train control



28

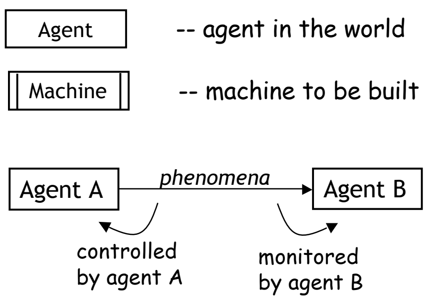
System Context Diagram

- Useful model for showing relevant parts of the world and their interfaces with the machine
- Agent = active entity
  - i.e. capable of controlling some world phenomena
  - can be human, software, or device
- A system context diagrams is composed of
  - a set of agents
  - for each agent,
    - a set of phenomena controlled by the agent
    - a set of phenomena monitored by the agent

29

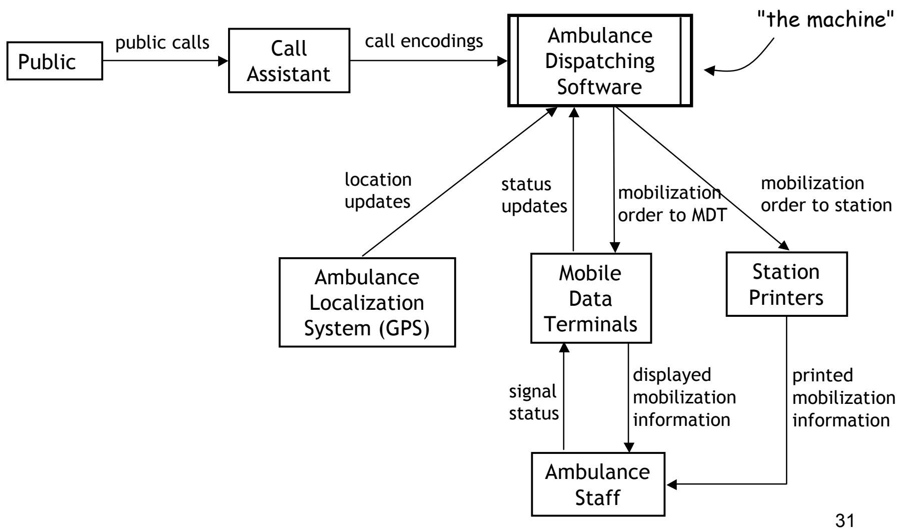
System Context Diagram

- Notation



30

SDC for Ambulance Dispatching System



31

Types of assertions (1)

- An assertion (statement) in a notation can be in *descriptive* (or *indicative*) mood
- They hold because of natural laws or physical constraints
- They state something that further observation may refute (i.e., falsify)
  - $\forall$ x, y, z (parent(x,z) $\wedge$ parent(y,z) $\rightarrow$ ((female(x) $\wedge$ male(y)) $\vee$ (male(x) $\wedge$ female(y))
  - $\forall$ d:door $\neg$(closed(d) $\wedge$ open(d))
  - $\forall$ b:book, x,y: person
    borrowed(b, x) $\wedge$ borrowed(b, y) $\rightarrow$ x=y

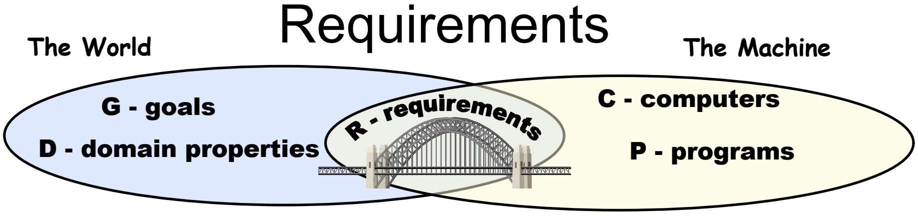
32

Types of assertions (2)

- A statement in a notation can be in *prescriptive* (or *optative*) mood
- It states a desirable property that may hold or not, depending on how the system works
  - must be enforced by some system's component(s)
  - they express our wishes
    - If the train is moving, the doors must be closed
      TrainMoving $\Rightarrow$ DoorsClosed

33

Goals vs. Domain Properties vs. Requirements



- **Goals** are ***prescriptive assertions*** formulated in terms of ***world*** phenomena *(not necessarily shared)*
- **Domain properties/assumptions** are ***descriptive* assertions** **assumed to hold** in the world
- **Requirements** are ***prescriptive assertions*** formulated in terms of ***shared*** phenomena

34

Example - ADS

- Goal:
  - *'For every urgent call reporting an incident, an ambulance should arrive at the incident scene within 14 minutes'*
- Domain assumptions:
  - *For every urgent call, details about the incident are correctly encoded*
  - *When an ambulance is mobilized, it will reach the incident location in the shortest possible time*
  - *Accurate ambulances locations are known by GPS*
  - *Ambulance crews correctly signal ambulance availability through mobile data terminals on board of ambulances'*
- Requirement:
  - *When a call reporting a new incident is encoded, the Automated Dispatching Software should mobilize the nearest available ambulance according to information available from the ambulances GPS and Mobile Data Terminals*

35

Correctness arguments

- Must prove the following properties
  1. R ensure satisfaction of the goals G in the context of the domain properties D

     R, D |= G

     - **Analogy with program correctness**: a Program P running on a particular Computer C must satisfy the Requirements R

       P, C |= R
  2. G adequately capture all of the stakeholders needs
  3. D represents valid properties/assumptions about the world

36

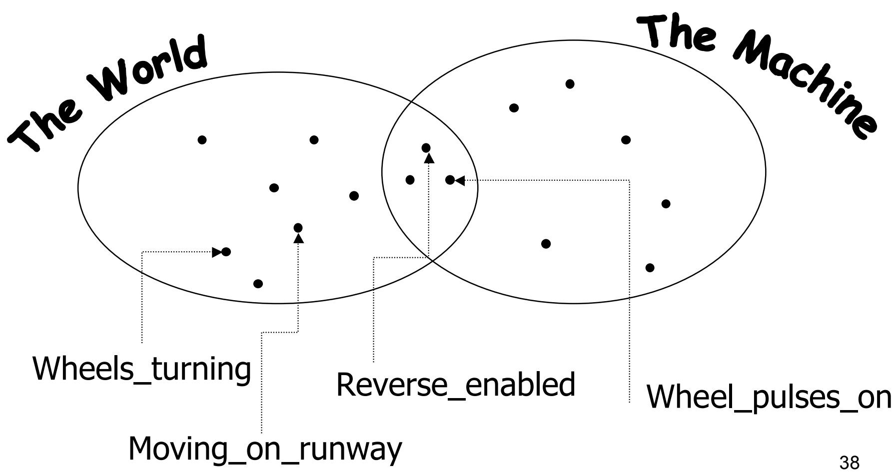
Failures in correctness arguments

- G
  - Wrong goals, the problem may not be the real problem
- D
  - Wrong domain assumptions
- Cannot prove that
  - G follows from D and R
    - the machine is inadequate to enforcing G

37

Example – Airbus A320 Braking Logic

(source: M. Jackson, *Software Requirements and Specifications*, 1995)



38

Example (cont.)

- Goal G:
  - “Reverse thrust shall be enabled if and only if the aircraft is moving on the runway”
- Domain assumptions D:
  - Wheel pulses on if and only if wheels turning
  - Wheels turning if and only if moving on runway
- Requirements R:
  - Reverse thrust enabled if and only if wheel pulses on

39

Correctness arguments

- Goal
  - Reverse_enabled $\Leftrightarrow$ Moving_on_runway
- Domain properties
  - Wheel_pulses_on $\Leftrightarrow$ Wheels_turning
  - Wheels_turning $\Leftrightarrow$ Moving_on_runway
- Requirements
  - Reverse_enabled $\Leftrightarrow$ Wheels_pulses_on

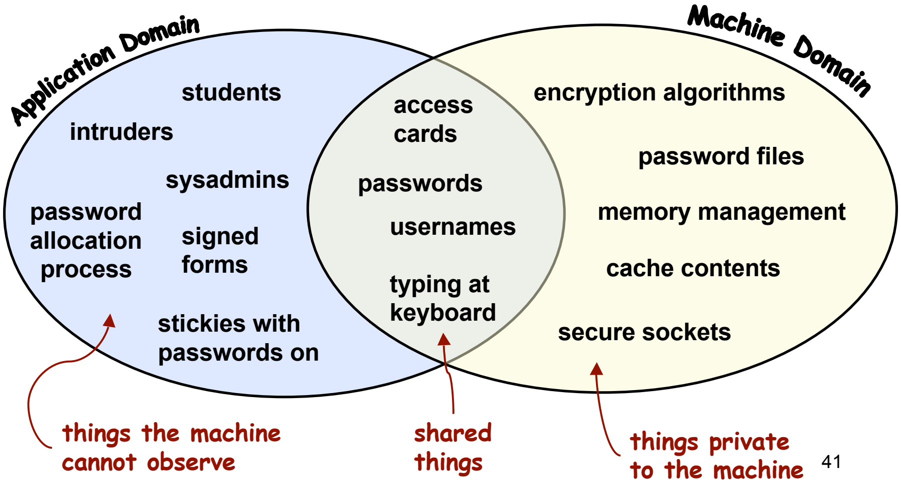
can prove that $R \wedge D \models G$

but D are not valid assumptions!!!

40

Example 3 - Access Control System

Goal: prevent unauthorized access to our machines



41

Example 3 - ACS (cont.)

- Goal G:
  - "The database shall only be accessible by authorized personnel"
- Domain Properties D:
  - Authorized personnel have passwords
  - Passwords are never shared with non-authorized personnel
- Requirement R :
  - Access to the database shall only be granted after the user types an authorized password
- R + D entail G *(Is this correct?)*
  - But what if the domain assumptions are wrong?

42

(slide: Steve Easterbrook)

A warning about terminology

- RE is a relatively young domain, there's no consensus on terminology yet, in particular about what is a requirement
- In Jackson's work
  – what we call a goal is called a requirement
  – what we call a requirement is called a specification
- vanLamsweerde uses the terms 'System Requirements' & 'Software Requirements'

43

Observation

- The boundary between the World & the Machine is generally not given at the start of a development project
- The purpose of a RE activity is
  – to identify the real goals of the project
  – to explore alternative ways to satisfy the goals, through
    - alternative pairs (Req, Dom) st. Req, Dom |= G
    - alternative interfaces between the world and the machine
  – to evaluate the strengths and risks of each alternative, in order to select the most appropriate one

(Techniques for exploring, evaluating and selecting among alternatives will be seen later)

44

So far…

- The World & the Machine
  - RE is concerned with phenomena occurring in the world
  - Shared phenomena
  - Goals vs. Domain Assumptions vs. Requirements
  - Goal satisfaction argument
- System Context Diagrams
  - show system agents with monitoring and control capabilities
- RE is about exploring alternatives

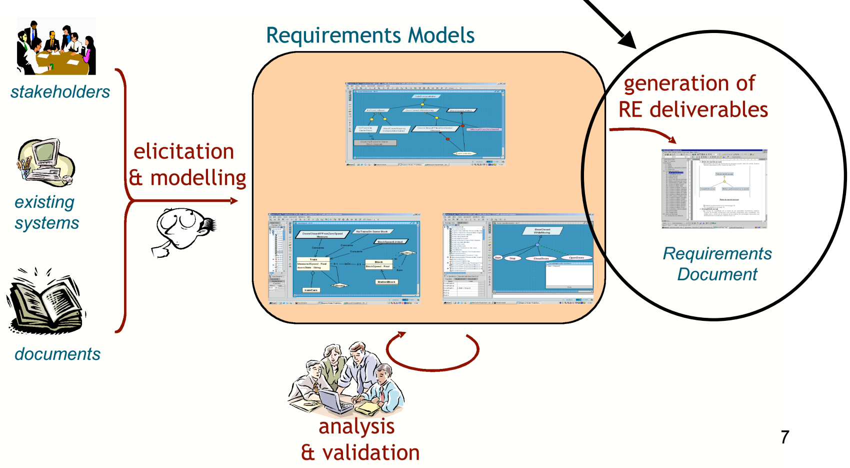
45

Target qualities of requirements documents

46

Software Requirements Specification

- Requirements captured in a document called the Software Requirements Specification (SRS)



7

Purposes of the SRS

- Communicates an understanding of the requirements
  - explains both the application domain and the system to be developed
- Contractual
  - May be legally binding!
- Baseline for project planning and estimation (size, cost, schedule)
- Baseline for software evaluation
  - supports system testing, verification and validation activities
  - should contain enough information to verify whether the delivered system meets requirements
- Baseline for change control
  - requirements change, software evolves

48

(slide: Steve Easterbrook)

Audience of the SRS

- Costumers & Users
  - Most interested in validating system goals and high-level description of functionalities
  - Not generally interested in detailed software requirements
- Systems Analysts, Requirements Analysts
  - Write various specifications of other systems that inter-relate
- Developers, Programmers
  - Have to implement the requirements
- Testers
  - Determine that the requirements have been met
- Project Managers
  - Measure and control the analysis and development processes

49

(slide: Steve Easterbrook)

Appropriate specification

Source: *Adapted from Blum 1992, p154-5*

- Consider two different projects:

  A) Small project, 1 programmer, 6 months work

  programmer talks to customer, then writes up a 5-page memo

  B) Large project, 50 programmers, 2 years work

  team of analysts model the requirements, then document them in a 500-page SRS

	Project A	***Project B***
Purpose of spec?	Crystalizes programmer's understanding; feedback to customer	Build-to document; must contain enough detail for all the programmers
Management view?	Spec is irrelevant; have already allocated resources	Will use the spec to estimate resource needs and plan the development
Readers?	**Primary**: Spec author; **Secondary**: Customer	**Primary**: all programmers + V&V team, managers; **Secondary**: customers

50

(slide: Steve Easterbrook)

A complication: **procurement**

- An 'SRS' may be written by…
  - **…the procurer:**
    - so the SRS is really a call for proposals
    - Must be general enough to yield a good selection of bids…
    - …and specific enough to exclude unreasonable bids
  - **…the bidders:**
    - Represents a proposal to implement a system to meet the CfP
    - must be specific enough to demonstrate feasibility and technical competence
    - …and general enough to avoid over-commitment
  - **…the selected developer:**
    - reflects the developer's understanding of the customers needs
    - forms the basis for evaluation of contractual performance
  - **…or by an independent RE contractor!**

51

(slide: Steve Easterbrook)

A complication: **procurement**

- Choice over what point to compete the contract
  - **Early (conceptual stage)**
    - can only evaluate bids on apparent competence & ability
  - **Late (detailed specification stage)**
    - more work for procurer; appropriate RE expertise may not be available in-house
  - **IEEE Standard recommends SRS jointly developed by procurer & developer**

52

(slide: Steve Easterbrook)

SRS Contents

Source: *adapted from* ***IEEE-STD-830***

- Software Requirements Specification should address:
  - Functionality. What is the software supposed to do?
  - External interfaces. How does the software interact with people, the system's hardware, other hardware, and other software?
  - Performance. What is the speed, availability, response time, recovery time of various software functions, and so on?
  - Attributes. What are the portability, correctness, maintainability, security, and other considerations?
  - Design constraints imposed on an implementation. Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) and so on?

IEEE Standard for SRS

Source: *Adapted from IEEE-STD-830-1993* ***See also****, Blum 1992, p160*

1 Introduction
- Purpose
- Scope
- Definitions, acronyms, abbreviations
- Reference documents
- Overview

2 Overall Description
- Product perspective
- Product functions
- User characteristics
- Constraints
- Assumptions and Dependencies

3 Specific Requirements

Appendices

Index

Identifies the product, & application domain

Describes contents and structure of the remainder of the SRS

Describes all external interfaces: system, user, hardware, software; also operations and site adaptation, and hardware constraints

Summary of major functions

Anything that will limit the developer's options (e.g. regulations, reliability, criticality, hardware limitations, parallelism, etc)

All the requirements go in here (i.e. this is the body of the document). IEEE STD provides 8 different templates for this section

IEEE STD Section 3 (example)

Source: *Adapted from IEEE-STD-830-1993.* ***See also****, Blum 1992, p160*

3.1 External Interface Requirements
- 3.1.1 User Interfaces
- 3.1.2 Hardware Interfaces
- 3.1.3 Software Interfaces
- 3.1.4 Communication Interfaces

3.2 Functional Requirements

this section organized by mode, user class, feature, etc. For example:

3.2.1 User Class 1

3.2.1.1 Functional Requirement 1.1

...

3.2.2 User Class 2

3.2.1.1 Functional Requirement 1.1

...

...

3.3 Performance Requirements

3.4 Design Constraints
- *3.4.1 Standards compliance*
- *3.4.2 Hardware limitations*
- *etc.*

3.5 Software System Attributes
- 3.5.1 Reliability
- 3.5.2 Availability
- 3.5.3 Security
- 3.5.4 Maintainability
- 3.5.5 Portability

3.6 Other Requirements

55

(slide: Steve Easterbrook)

Target qualities for a SRS (1)

- **Completeness**
  - **wrt. goals:** the requirements are sufficient to satisfy the stakeholders goals under given domain assumptions

    Req, Dom |= Goals
    - all goals have been correctly identified, including all relevant quality goals
    - Dom represent valid assumptions; incidental and malicious behaviours have been anticipated
  - **wrt. inputs:** the required software behaviour is specified for all possible inputs
  - **Structural completness:** no TBDs

56

(slide: adapted from A. van Lamsweerde)

Target qualities for a SRS (2)

- **Pertinence**
  - Each requirement or domain assumption is needed for the satisfaction of some goal
  - Each goal is truly needed by the stakeholders
  - The SRS does not contain items that are unrelated to the definition of requirements (e.g. design or implementation decisions)
- **Consistency**
  - No contradiction in formulation of goals, requirements, and assumptions

57

(slide: adapted from A. van Lamsweerde)

Target qualities for a SRS (3)

- **Unambiguity**
  - Unambiguous vocabulary: every term is defined and used consistently
  - Unambiguous assertions: Goals, requirements and assumption must be stated clearly in a way that precludes different interpretations
  - Verifiability: A process exists to test satisfaction of each requirement
  - Unambiguous Responsibilities: the split of responsibilities between the software-to-be and its environment must be clearly indicated

58

(slide: adapted from A. van Lamsweerde)

Ambiguous assertion test

- Natural Language?
  - "The system shall report to the operator all faults that originate in critical functions or that occur during execution of a critical sequence and for which there is no fault recovery response."

    (adapted from the specifications for the international space station)
- Or a decision table?

Originate in critical functions	F	T	F	T	F	T	F	T
Occur during critical seqeunce	F	F	T	T	F	F	T	T
No fault recovery response	F	F	F	F	T	T	T	T
Report to operator?								

59

Slide: S. Easterbrook - Source: *Adapted from Easterbrook & Callahan, 1997.*

Target qualities for a SRS (4)

- **Feasibility**
  - The goals and requirements must be realisable within the assigned budget and schedules
- **Comprehensibility**
  - must be comprehensible by all in the target audience

60

(slide: adapted from A. van Lamsweerde)

Target qualities for a SRS (5)

- **Traceability**
  - must indicate sources of goals, requirements and assumptions
  - must link requirements and assumptions to underlying goals
  - facilitates referencing of requirements in future documentation (design, test cases, etc.)
- **Good structuring**
  - E.g. highlights links between goals, reqts and assumptions
  - Every item must be defined before it is used
- **Modifiability**
  - must be easy to adapt, extend or contract through local modifications
  - impact of modifying an item should be easy to assess

61

(slide: adapted from A. van Lamsweerde)

Errors in requirements documents (1)

- Incompleteness
  - missing goals and requirements (the most critical !)
  - unspecified response to some input
- Inadequate requirements
- Contradictions
- Ambiguities
  - undefined or ambiguous terms
  - ambiguous expression of goals, reqts, assumptions
  - unverifiable reqts (wishfull thinking)
  - ambiguous split of responsibilities
- Unfeasible goals and requirements

62

(slide: adapted from A. van Lamsweerde)

Errors in requirements documents (2)

- Noise
  - Items unrelated to reqts definition
  - Duckspeak: reqts that are only there to conform to standards
  - Uncontrolled redundancy
- Overspecification
  i.e. including elements of design and implementation
- Lack of clarity
  - unnecessary invention of terminology
  - bad writing, hard for readers to decipher the intent
- Poor structuring
  - forward reference: using a concept that is defined only later
  - remorse: defining a reqts item lately or incidentally (use of parenthesis)
  - hard-to-follow cross-referencing
- Lack of traceability

63

(slide: adapted from A. van Lamsweerde)

Requirements errors are ...

- the most numerous
  - 33% of software errors
- the most persistent
  - often found after delivery
- the most expensive
  - detection/fix costs 5x more during design, 10x more during implementation, 20x more during testing, 200x more after delivery
- among the most dangerous
  - Aegis, LAS, Airbus Warsaw accident, BMW, etc...
  - cfr. Peter Neumann's Risks Digest
    http://catless.ncl.ac.uk/risks

64

(slide: A. van Lamsweerde)

Some observations about RE (1)

- The requirements specification will be imperfect
  - RE models are approximations of the world
    - will contain inaccuracies and inconsistencies
    - will omit some information
    - analysis should reduce the risk that these will cause serious problems
- Perfecting a specification may not be cost-effective
  - Requirements analysis has a cost
  - For different projects, the cost-benefit balance will be different
  - This is often a bad excuse for not investing in RE (cfr. causes of software failures)

65

(slide: adapted from S. Easterbrook)

Some observations about RE (2)

- Software development is not a sequential process
  - RE activities continue throughout the development process
  - We don't necessarily have to write *all* the requirements upfront
    - (Re-)writing requirements can be useful at any stage of development
    - However, the late discovery of some requirements may trigger the need for major redesign work (hence, major time and cost overruns, or project cancellation)
- Requirements should never be treated as fixed
  - Change is inevitable, and therefore must be planned for
  - There should be a way of incorporating changes periodically

66

(slide: adapted from S. Easterbrook)

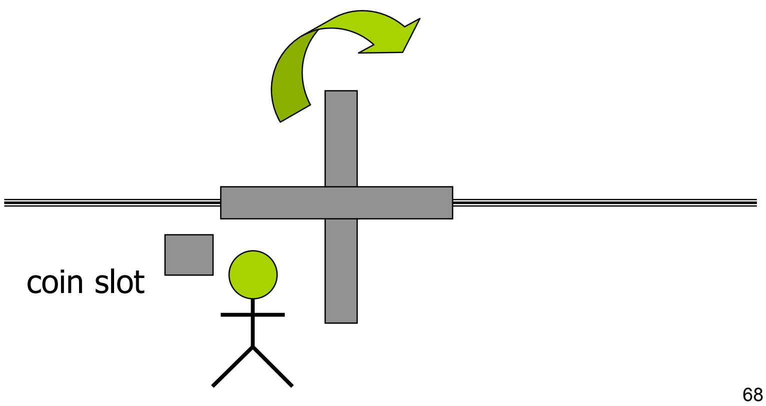
So far…

1. Definition and role of RE in systems development
2. Fundamentals of RE
   - Goals vs. domain properties vs. requirements
   - system context diagrams
3. Software Requirements Documents
   - Purposes & audiences
   - Target qualities, errors, & flaws

67

An example: how to derive rqmts from goals

Turnstile controlling entrance to the zoo



68

Domain analysis

- Identify the relevant environment phenomena (here: *events*)
- "Relevant" with respect to the goals (i.e, control people entrance)

machine controlled

Push *frees turnstile*

Lock
Unlock } *electric signals*

Enter *puts turnstile back in home position*

Coin *coin inserted*

shared phenomena

environment controlled

Domain descriptions (indicative)

They are real world properties

They do not depend on the machine

D1: Push and Enter alternate (starting with Push)

- one cannot enter without first pushing
- one cannot push until the previous visitor entered

D2: Push always leads to Enter

- a hydraulic system imposes it

D3: If Locked, Push cannot occur

Goals

G1: At any time entries should never exceed accumulated payments (for simplicity, assume 1 coin for 1 entrance)

G2: Those who pay are not prevented from entering (by the "machine")

- *They are optative descriptions*
- *Both are said to be **safety** properties (they state that nothing bad will ever occur)*

71

Deriving rqmts from goals (1)

- Must find the constraints on *shared phenomena* to be *enforced by the machine* to achieve the goal
  - G1 can be enforced by controlling entries or coins

G1: At any time entries should never exceed accumulated payments

the machine cannot compel Coin events

it can prevent Enter events

72

How to constrain entry events?

- From D1 we derive (*):

D1: Push and Enter alternate (starting with Push)

(*) At any time t, if e Enter and p Push events were observed, then p-1<=e<=p

- We get (**) by strengthening G1 via (*), by referring to shared phenomena:

(**) At any time t, if p Push and c Coin events were observed, then p<=c

G1: At any time entries should never exceed accumulated payments

if this can be enforced then G1 holds

73

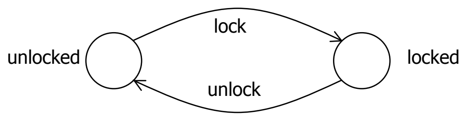
How to ensure p<=c?

- When p=c, must prevent further Push until Coin event occurs
- But how can the machine prevent Push?

R1: Impose that Lock and Unlock alternate (initially locked)

R2:

i. If locked and p=c, the machine must not unlock the turnstile, and
ii. If unlocked and p=c, the machine must perform a Lock in time to prevent further Push



74

Proof that G1 holds

G1: At any time entries should never exceed accumulated payments

- Need to prove that if p=c no further p can occur if c does not change

  Two possible cases

  - locked (i.e., #L=#U+1)
    - one cannot push (according to D3)
  - unlocked (i.e. #L=#U)
    - immediate lock prevents push (again according to D3)

#L (#U) number of Locks (Unlocks)

D3: If Locked, Push cannot occur

75

Deriving rqmts from goals (2)

Goal G2

Those who pay are not prevented from entering (by the "machine")

must be transformed into a requirement R3:

i. If unlocked and p<c, the machine does not lock until there is credit, and
ii. If locked and p<c, the machine must perform Unlock event

76

Proof of G2

- From i, a new p can occur, hence an e (from D1, D2)

R3, i: If unlocked and p<c, the machine does not lock until there is credit

- From ii, we enter case i immediately

R3, ii: If locked and p<c, the machine must perform Unlock event

77

References

- M. Jackson, P. Zave, "Deriving Specifications from Requirements: An Example", Proceedings of ICSE 95, 1995
- M. Jackson, P. Zave, "Four Dark Corners of Requirements Engineering",TOSEM, 1997
- B. Nuseibeh, S. Easterbrook, "Requirements Engineering: A Roadmap", Proceedings ICSE 2000
- A. van Lamsweerde, Requirements Engineering, Wiley and Sons, expected march 2007 (draft on the web)
- M. Jackson, Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices, ACM Press Books, 1995
- S. Robertson and J. Robertson, Mastering the Requirements Process, Addison Wesley, 1999
- Requirements Engineering Specialist Group of the British Computer Society http://www.resg.org.uk/

78