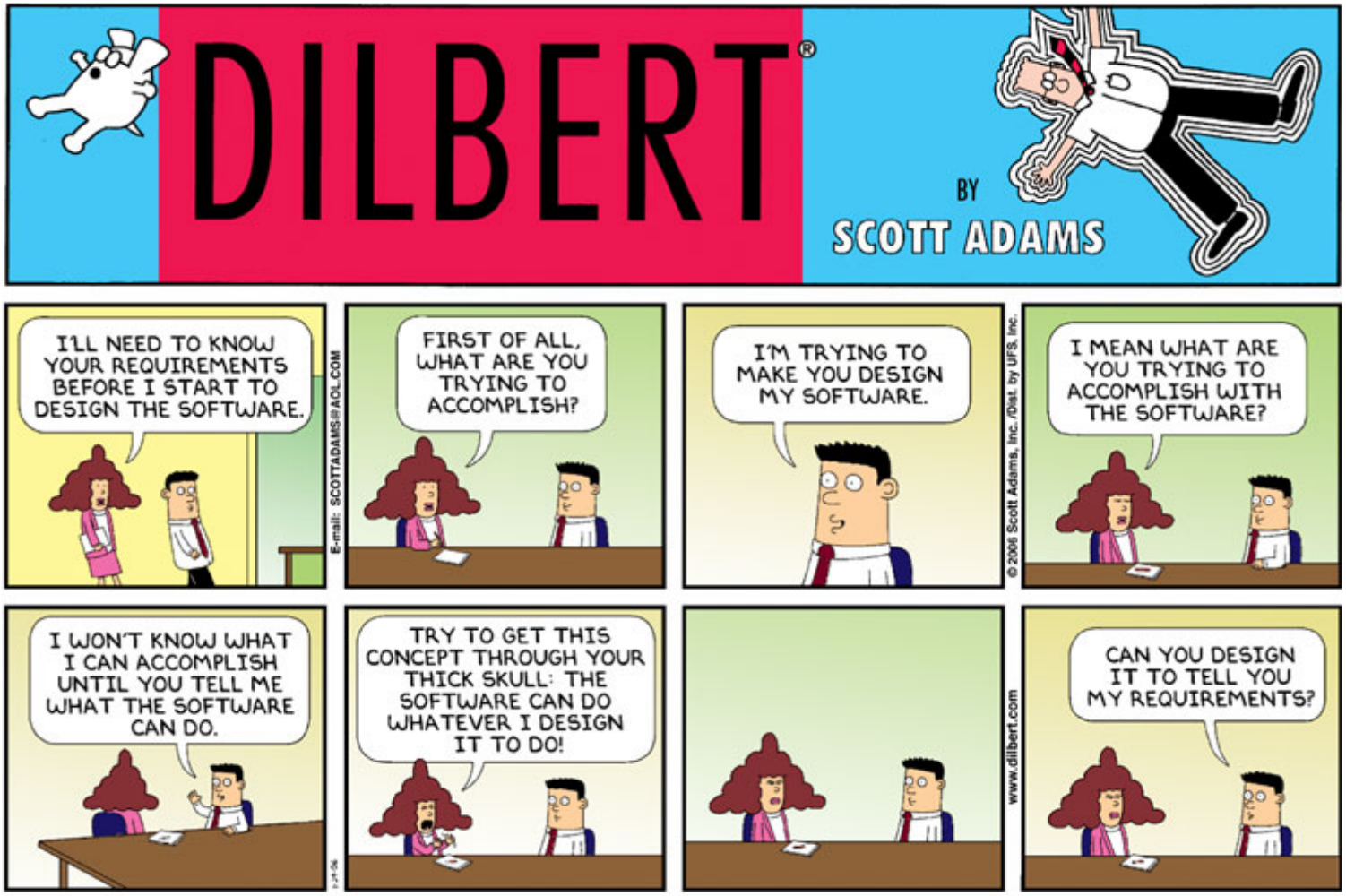
RE Overview

Based on presentations by G. Mussbacher, G.V Bochmann, N. Niu



© Scott Adams, Inc./Dist. by UFS, Inc.

Importance of RE (1)

Mars Climate Orbiter

- In 1999, the Mars Climate Orbiter disappears around Mars
- Cost: about $125M US
- Problem caused by a misunderstanding between a team in Colorado and one in California
- One team used the metric system while the other used the English system for a key function…

Importance of RE (2)

- Problems
  - Increased reliance on software
    - e.g. cars, dishwashers, cell phones, web services, …
  - Software now the biggest cost element for mission critical systems
    - e.g. Boeing 777
  - Wastage on failed projects
    - e.g. 1997 GAO report: $145 billion over 6 years on software that was never delivered
  - High consequences of failure
    - e.g. Ariane 5: $500 million payload
    - e.g. Intel Pentium bug: $475 million
- Key factors:
  - Certification costs
    - e.g. Boeing 777≥40% of software budget spent on testing
  - Re-work from defect removal
    - e.g. Motorola: 60-80% of software budget (was) spent on re-work
  - Changing Requirements
    - e.g. California DMV system

(I) Introduction

- **What are Requirements?**
  - Scope (for this course): “Software-intensive Systems”
  - Separating the Problem from the Solution
  - What Requirements Engineers do
- **What is Engineering?**
  - Engineering as a profession
  - Engineering projects
  - Engineering lifecycles
  - Engineering design
- **What is a System?**
  - General systems theory
  - Formal foundations of software systems
  - Conceptual foundations of information systems
  - Empirical foundations of human activity systems
  - Observability of systems

(II) Eliciting and Planning

* **Elicitation Targets**
  - Stakeholders & User Classes
  - System boundaries
  - Goals
  - Scenarios
* **Elicitation Techniques**
  - Interviews, questionnaires, surveys, meetings
  - Prototyping
  - Ethnographic techniques
  - Knowledge elicitation techniques
  - Conversation Analysis
  - Text Analysis
* **The Feasibility Study**
  - Types of Feasibility
  - Cost/benefit analysis
* **Risk Analysis**
  - Identifying and managing risk

(III) Modeling & Analyzing

* **Basics of Modeling**
  - Notations and their uses
  - Formality and Expressiveness
  - Abstraction and Decomposition
  - Model management and viewpoints
  - Types of Analysis
* **Enterprises**
  - Business rules and organizational structures
  - Goals, tasks and responsibilities
  - Soft Systems analysis
* **Information Structures**
  - Entities and Relationships
  - Classes and Objects
  - Domain Ontologies
* **Behavior**
  - Activities and Interactions
  - States and Transitions
  - Concurrency
* **Quality Requirements**
  - Taxonomies of NFRs
  - Performance
  - Usability
  - Safety
  - Security
  - Reliability
  - Maintainability

(IV) Communicating & Agreeing

* Validation
  - Refutable descriptions
  - Role of contracts and procurement
  - Role of organizational politics
* Documenting Requirements
  - Properties of a good specification
  - Documentation standards
  - Specification languages
  - Making requirements testable
* Prototyping and Walkthroughs
  - Throwaway prototyping
  - Operational prototyping
  - Walkthroughs of operational models
* Reviews and Inspections
  - Effectiveness of Inspection
  - Conducting an Inspection
  - Collaborative Requirements Workshops
* Negotiation and Prioritization
  - Representing argumentation and rationale
  - Computer-supported negotiation
  - Trade-off analysis
  - Release planning

(V) Realizing and Evolving

- Software Evolution
  - Laws of evolution
  - Release planning
  - Product families
  - Requirement Reuse
- Requirements and Architectures
  - Architectural Patterns and Description Languages
  - Mapping requirements to architectures
  - Architectural Robustness
- Managing Change
  - Baselines and change requests
  - Configuration management and version control
  - Impact Analysis
- Traceability and Rationale
  - Pre- and Post- traceability
  - Capturing Design Rationale
  - Traceability techniques
- Managing Inconsistency
  - On the inevitable intertwining of inconsistency and change
  - Learning from inconsistency
  - Feature interaction
  - Living with inconsistency
- IR and NLP in RE
- Security Requirements

Definition and Importance of Requirements

What are “Requirements”?

- A requirement is:
  - Capturing the purpose of a system

- An expression of the ideas to be embodied in the system or application under development

  - A statement about the proposed system that all stakeholders agree must be made true in order for the customer’s problem to be adequately solved
    - Short and concise piece of information
    - Says something about the system
    - All the stakeholders have agreed that it is valid
    - It helps solve the customer’s problem

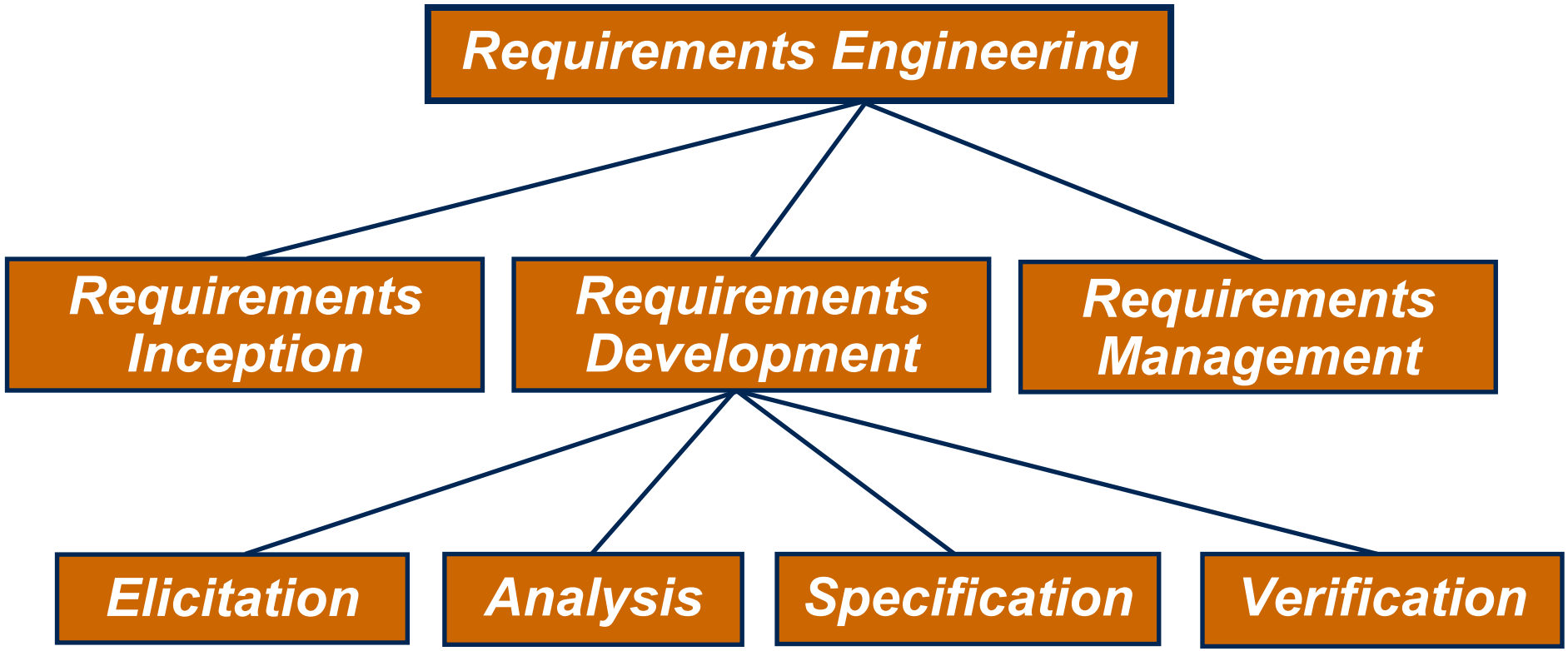
According to IEEE 830-1993

- A **requirement** is defined as:
  - A condition or capability needed by a user to solve a problem or achieve an objective
  - A condition or a capability that must be met or possessed by a system … to satisfy a contract, standard, specification, or other formally imposed document …

What is "Requirements Engineering"?

- **Requirements Engineering (RE)** is:
  - The activity of development, elicitation, specification, analysis, and management of the stakeholder requirements, which are to be met by a new or evolving system
  - RE is concerned with identifying the purpose of a software system… and the contexts in which it will be used
    - How/where the system will be used
    - Big picture is important
  - Captures real world needs of stakeholders affected by a software system and expresses them as artifacts that can be implemented by a computing system
    - Bridge to design and construction
    - How to communicate and negotiate?
    - Is anything lost in the translation between different worlds?

Requirements Engineering Activities



Source: Larry Boldt, Trends in Requirements Engineering People-Process-Technology, Technology Builders, Inc., 2001

About these RE Activities…

- **Inception**
  - Start the process (business need, market opportunity, great idea, ...), business case, feasibility study, system scope, risks, etc.
- **Requirements elicitation**
  - Requirements discovered through consultation with stakeholders
- **Requirements analysis and negotiation**
  - Requirements are analyzed and conflicts resolved through negotiation
- **Requirements specification**
  - A precise requirements document is produced
- **Requirements validation**
  - The requirements document is checked for consistency and completeness
- **Requirements management**
  - Needs and contexts evolve, and so do requirements!

General Problems with the Requirements Process

- Lack of the right expertise (software engineers, domain experts, etc.)
- Initial ideas are often incomplete, wildly optimistic, and firmly entrenched in the minds of the people leading the acquisition process
- Difficulty of using complex tools and diverse methods associated with requirements gathering may negate the anticipated benefits of a complete and detailed approach

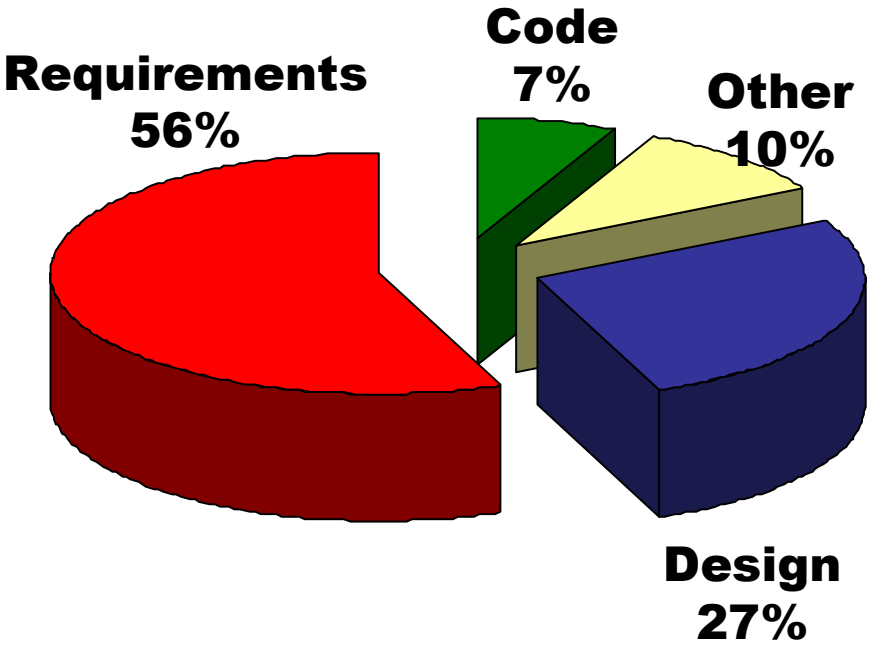
Statistics from NIST Report

- NIST (National Institute of Standards and Technology) has published a comprehensive (309 pages) and very interesting report on project statistics and experiences based on data from a large number of software projects[1]
  - 70% of the defects are introduced in the specification phase
  - 30% are introduced later in the technical solution process
  - Only 5% of the specification inadequacies are corrected in the specification phase
  - 95% are detected later in the project or after delivery where the cost for correction on average is 22 times higher compared to a correction directly during the specification effort
  - The NIST report concludes that extensive testing is essential, however testing detects the dominating specification errors late in the process

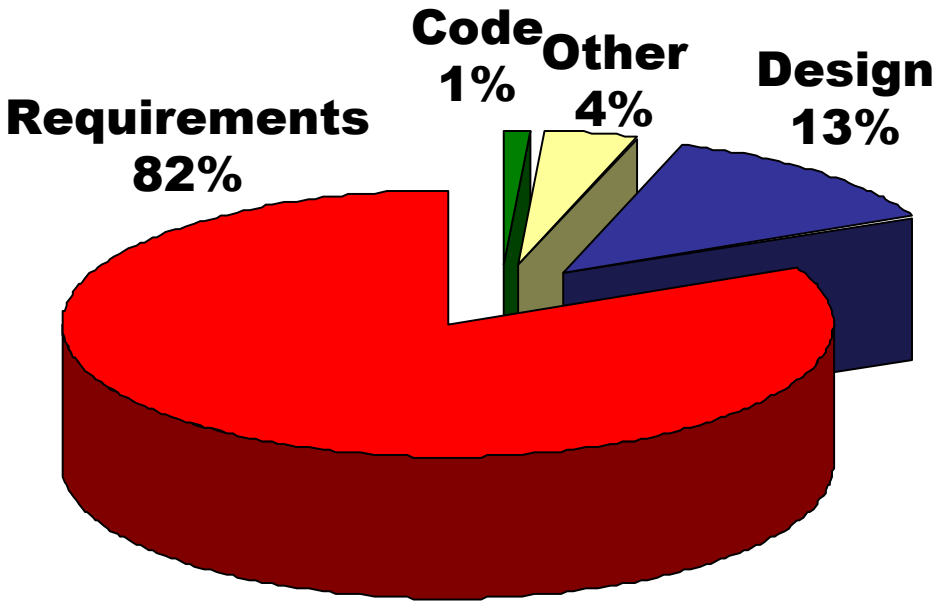
[1] http://www.nist.gov/public_affairs/releases/n02-10.htm (May 2002)

Why Focus on Requirements ?

- Distribution of Defects



- Distribution of Effort to Fix Defects



Source: Martin & Leffinwell

View of the Software Engineering Institute (SEI)

- Improve software development with the CMM/CMMI model for software development
  - Capability Maturity Model (CMM)
  - For software development, superseded by Capability Maturity Model Integration (CMMI)

- SEI's vision is:
  - The right software, delivered defect free, on time & on cost, every time
  - "Right software" implies software that satisfies requirements for functionality and qualities (e.g., performance, cost…) throughout its lifetime
  - "Defect free" software is achieved either through exhaustive testing after coding or by developing the code right the first time

CHAOS Report (2004)[1]

RESOLUTION OF PROJECTS

This year's results show that 29% of all projects succeeded (delivered on time, on budget, with required features and functions); 53% are challenged (late, over budget and/or with less than the required features and functions); and 18% have failed (cancelled prior to completion or delivered and never used), as shown in Figure 2.0.

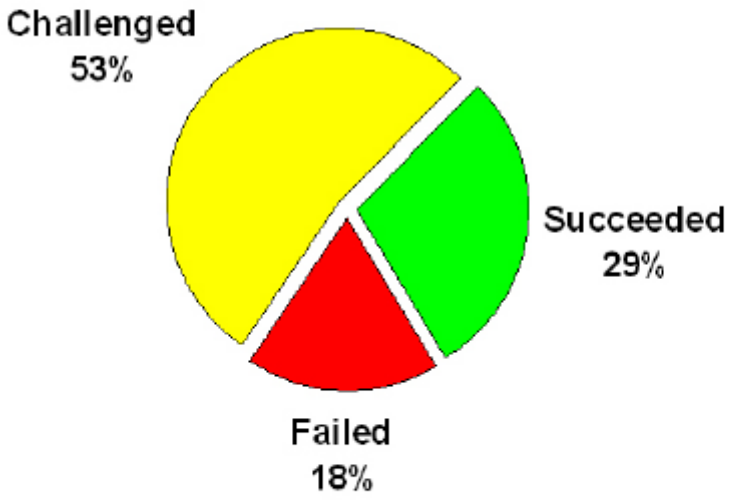
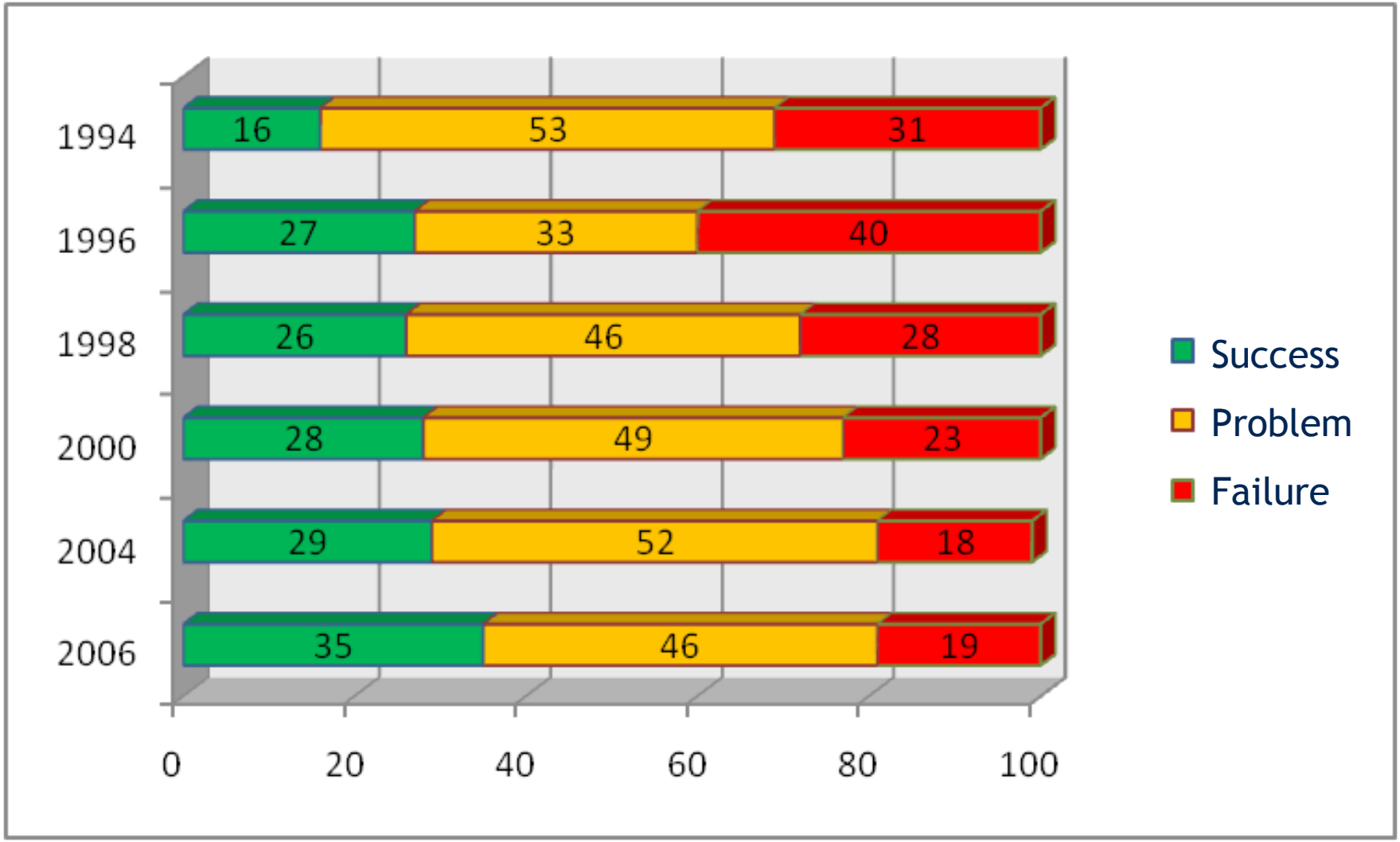


Figure 2.0

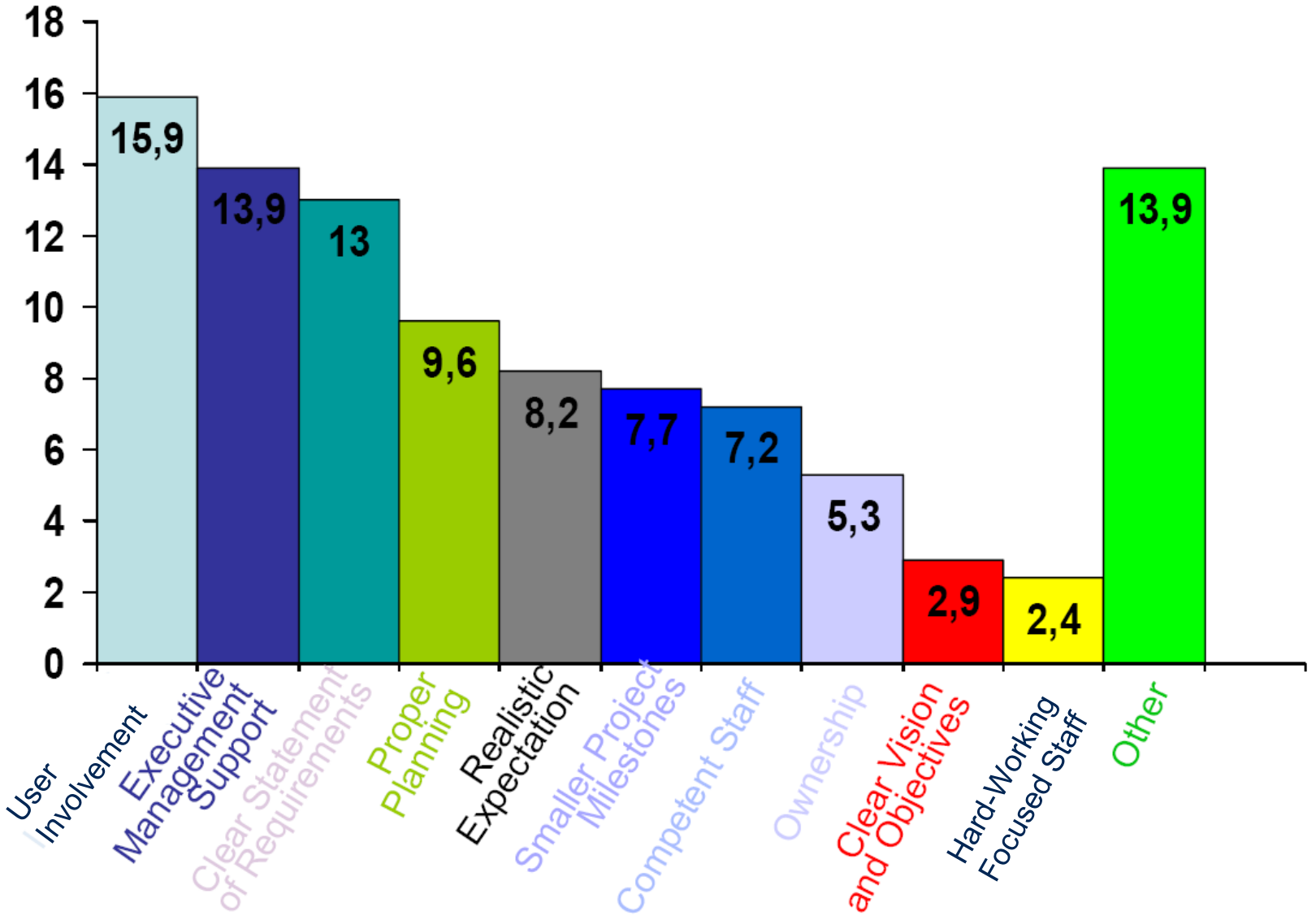
[1] Standish Group Inc., 2004

Progression since 1994



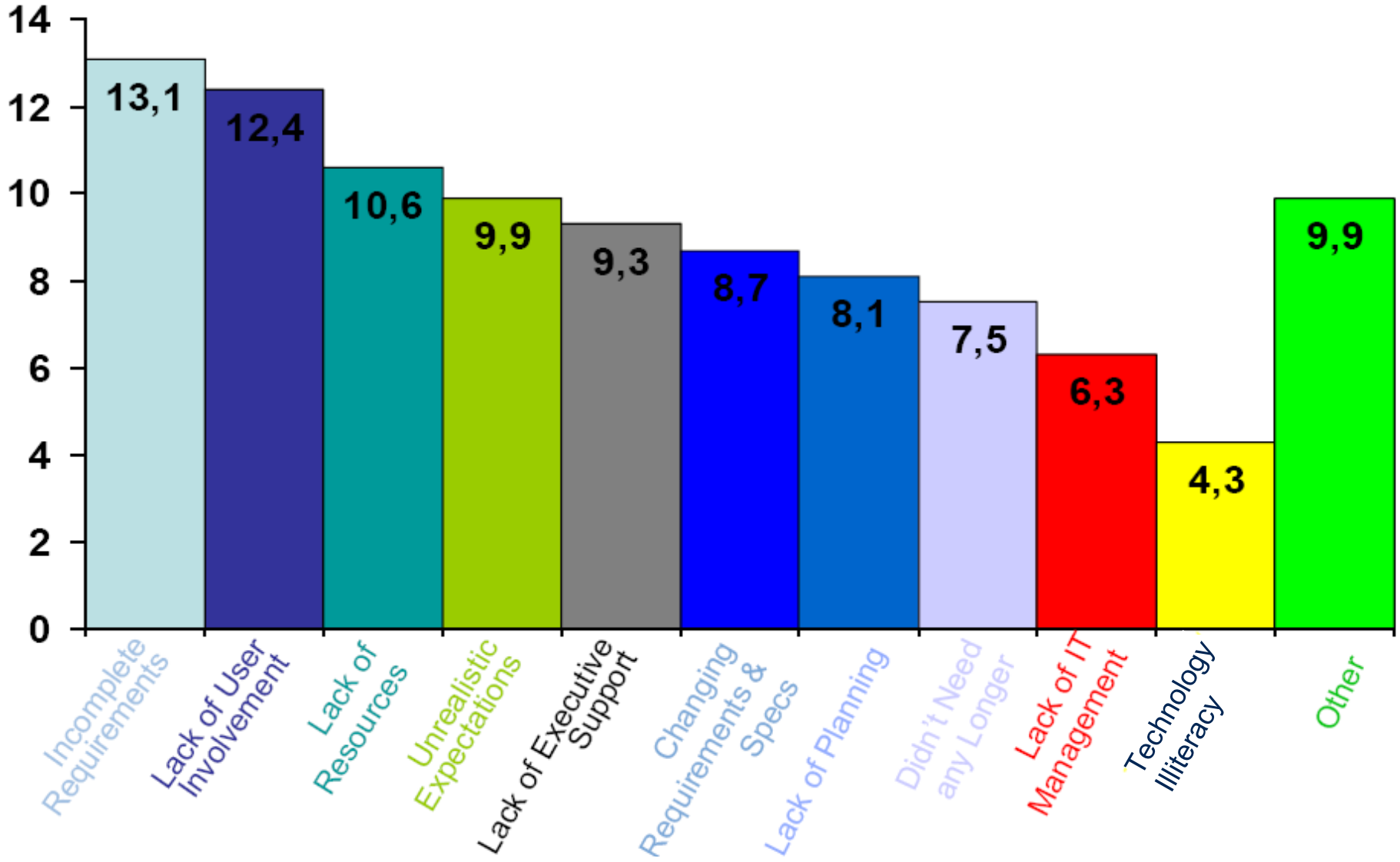
Source: Standish Group Inc., 1994-2006

Success Factors



Source: Standish Group Inc., 1995

Problem Causes



Source: Standish Group Inc., 1995

Evolution of Success Factors

CHAOS			
1994	1996	1999	2000
User involvement	User involvement	User involvement	Executive management support
Executive management support	Executive management support	Executive management support	User involvement
Clear statement of requirements	Clear statement of requirements	Clear statement of requirements	Experienced project manager
Proper planning	Firm basic requirements	Experienced project manager	Clear business objectives
Realistic expectations	Competent staff	Small project milestones	Minimized scope
Small project milestones	Small project milestones	Firm basic requirements	Standard software infrastructure
Competent staff	Experienced project manager	Competent staff	Firm basic requirements
Ownership	Proper planning	Proper planning	Formal methodology
Clear vision and objectives	Ownership	Ownership	Reliable estimates
Hard-working, focused staff	Other	Other	Other

Involvement/ Ownership | Requirements | Management Support | Planning | Staff

Source: Standish Group Inc., 2000

Managing Evolving Requirements

"Changing requirements is as certain as death and taxes"

Requirement tools: These seem to have the biggest impact on the success of a project. This may seem strange since "Firm Basic Requirements" is number six on the top ten list. However these tools, if used as a platform for communications between all the stakeholders, such as executive sponsors and users, can provide enormous benefits. This tool needs to be at the top of the shopping list for any firm involved in developing software applications.

Source: http://standishgroup.com/sample_research/PDFpages/extreme_chaos.pdf, 1999

Types of Requirements

So Many “Requirements”… (1)

- A goal is an objective or concern that guides the RE process. It can be used to discover and evaluate functional and non-functional requirements
  - A goal is not yet a requirement…
- Note: All requirements must be verifiable (by some test, inspection, audit etc.)
- A functional requirement is a requirement defining functions of the system under development
  - Describes what the system should do
- A non-functional requirement is a requirement that is not functional. This includes many different kinds of requirements. – Therefore one often considers the following sub-categories:

Different types of non-functional requirements

- Performance requirements, characterizing system properties such as expected performance, capacity, reliability, robustness, usability, etc.
- Design constraints (also called process requirements), providing constraints on how the system should be designed and built – related to development process, documentation, programming language, maintainability, etc.
- Commercial constraints, such as development time frame and costs.

So Many “Requirements”… (2)

- A user requirement is a desired goal or function that a user and other stakeholders expect the system to achieve
  - Does not necessarily become a system requirement
- Application domain requirement (sometimes called business rules) are requirements derived from business practices within a given industrial sector, or in a given company, or defined by government regulations or standards.
  - May lead to system requirements. Can be functional or non-functional
- Problem domain requirements should be satisfied within the problem domain in order to satisfy some of the goals
- System requirements are the requirements for the system to be built, as a whole
  - A system is a collection of interrelated components working together towards some common objective (may be software, mechanical, electrical and electronic hardware and be operated by people)
  - Systems Engineering is a multidisciplinary approach to systems development – software is only a part (but often the problematic part)

So Many “Requirements”… (3)

- Important note: Software Requirements Engineering is a special case of Requirements Engineering. Many topics discussed in this course are quite general and apply to requirements engineering, in general.
- In a system containing software, software requirements are derived from the system requirements. The system then consists of hardware and software, and the hardware (and often the operating system and other existing software modules) are part of the environment in which the software is used.

Functional Requirements

- What inputs the system should accept
- What outputs the system should produce
- What data the system should store other systems might use
- What computations the system should perform
- The timing and synchronization of the above

- Depend on the type of software, expected users, and the type of system where the software is used
- Functional user requirements may be high-level statements of what the system should do, but functional system requirements should describe the system services in detail

Examples of Functional Requirements

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

Note: not all requirements on this and following slides are high quality requirements but are typical requirements found too often in documents

Non-Functional Requirements (NFR) (1)

- Non-functional requirements are important
  - If they are not met, the system is useless
  - Non-functional requirements may be very difficult to state precisely (especially at the beginning) and imprecise requirements may be difficult to verify
- They are sometimes called quality requirements, quality of service, or extra-functional requirements.
- *Three main categories [1]:*
  - **Performance requirements** reflecting: usability, efficiency, reliability, maintainability and reusability (note: also security requirements)
    - Response time, throughput
    - Resource usage
    - Reliability, availability
    - Recovery from failure
    - Allowances for maintainability and enhancement
    - Allowances for reusability

[1] Lethbridge and Laganière, Object Oriented Software Engineering: Practical Software Development using UML and Java, 2005

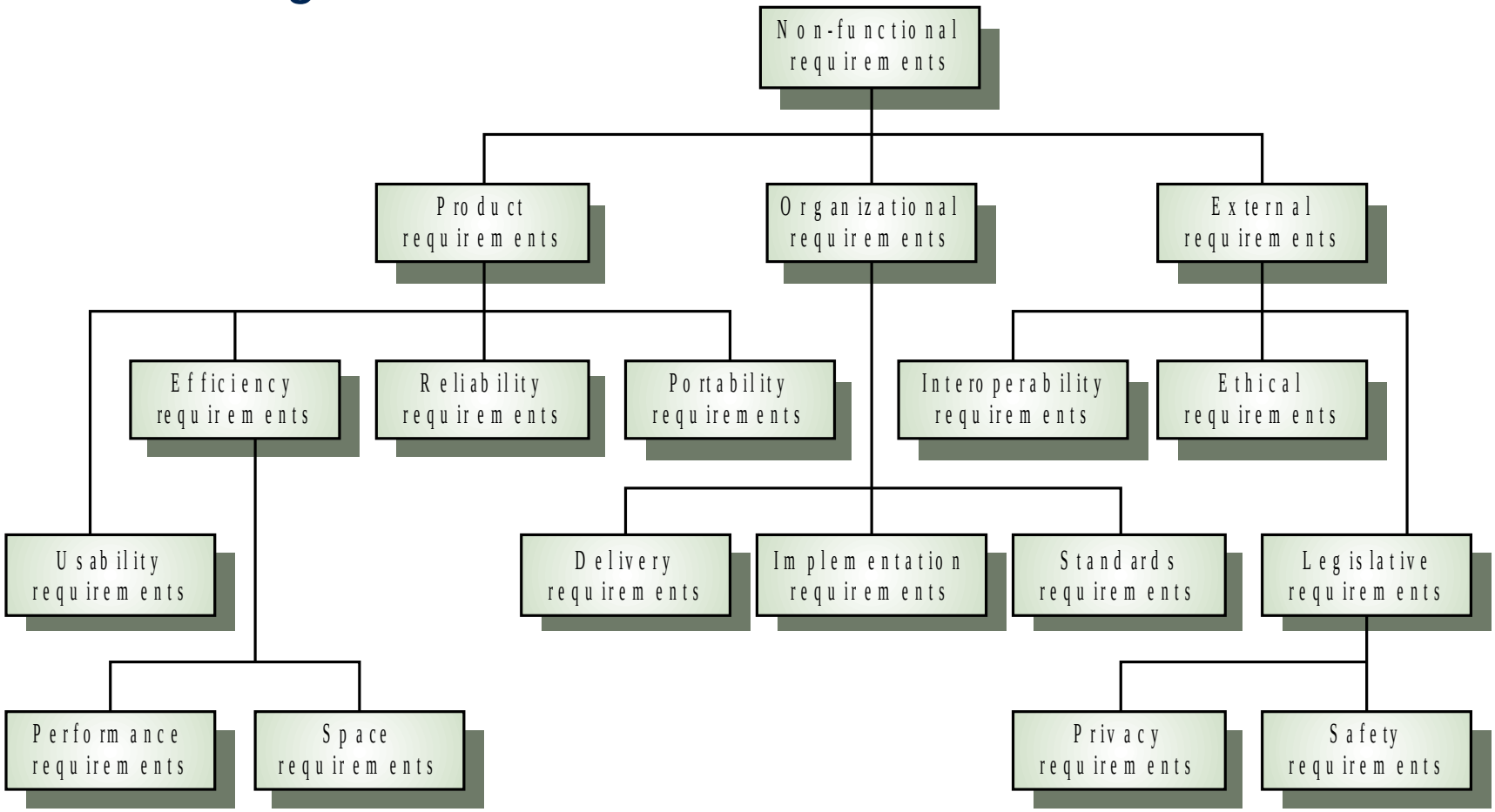
Non-Functional Requirements (NFR) (2)

- **Design constraints:** Categories constraining the environment and technology of the system.
  - Platform (minimal requirements, OS, devices…)
  - Technology to be used (language, DB, …)

- **Commercial constaints:** Categories constraining the project plan and development methods
  - Development process (methodology) to be used
  - Cost and delivery date
    - Often put in contract or project plan instead

Various NFR Types

- *Other ontologies also exist*



Source: Gerald Kotonya and Ian Sommerville, Requirements Engineering – Processes and Techniques, Wiley, 1998

Examples of Non-Functional Requirements

- **Product requirement**
  - It shall be possible for all necessary communication between the APSE and the user to be expressed in the standard Ada character set.
- **Process requirement**
  - The system development process and deliverable documents shall conform to the process and deliverables defined in XYZCoSPSTAN95.
- **Security requirement**
  - The system shall not disclose any personal information about customers apart from their name and reference number to the operators of the system.

Measurable Non-Functional Requirements

Property	Measure
Speed	Processed transactions/second User/Event response time Screen refresh time
Size	K Bytes Number of RAM chips
Ease of use	Training time Number of help frames
Reliability	Mean time to failure Probability of unavailability Rate of failure occurrence Availability
Robustness	Time to restart after failure Percentage of events causing failure Probability of data corruption on failure
Portability	Percentage of target dependent statements Number of target systems

Source: Gerald Kotonya and Ian Sommerville, Requirements Engineering – Processes and Techniques, Wiley, 1998

Goals

- A Goal
  - Conveys the intention or the objective of one or many stakeholders
  - Can guide the discovery of verifiable non-functional requirements that can be tested objectively

Example of Goal and NFR

- A system goal
  - The system should be easy to use by experienced controllers and should be organized in such a way that user errors are minimized.

- A verifiable usability requirement derived from this goal
  - Experienced controllers shall be able to use all the system functions after a total of three hours of training.
  - The average number of errors made by experienced controllers shall not exceed two per day.

  - Assumption: An experienced controller has at least 2 years experience with the old system (as stated by the stakeholder)

Application-Domain Requirements

- Derived from the application domain
- Describe system characteristics and features that reflect the domain
- May be new functional requirements, constraints on existing requirements, or define specific computations
- If domain requirements are not satisfied, the system may be unworkable

Examples of Application-Domain Requirements

- **Library system**
  - The system interface to the database must comply with standard Z39.50.
  - Because of copyright restrictions, some documents must be deleted immediately on arrival. Depending on the user's requirements, these documents will first be printed either locally or printed to a network printer and retrieved by the user.

- **Train protection system**
  - The deceleration of the train shall be computed as:
    $D_{train} = D_{control} + D_{gradient}$

    where $D_{gradient}$ is $9.81 ms^2$ * compensated gradient / alpha and where the values of $9.81 ms^2$ / alpha are known for different types of train.

Problems Concerning Application-Domain Requirements

- **Understandability**
  - Requirements are expressed in the language of the application domain
  - This is often not understood by software engineers developing the system

- **Implicitness / Tacit knowledge**
  - Domain specialists understand the area so well that they do not think of making the domain requirements explicit
  - People are often unaware of the tacit knowledge they possess and therefore cannot express it to others

Emergent Properties (when the system consists of several sub-systems)

- **Properties of the system as a whole**
  - Requirements which cannot be addressed by a single component, but which depend for their satisfaction on how all the software components interoperate
  - Only emerge once all individual subsystems have been integrated
  - Dependent on the system architecture

- **Examples of emergent properties**
  - Reliability
  - Maintainability
  - Performance
  - Usability
  - Security
  - Safety

The Requirements Engineering Process

Requirements within the software development process

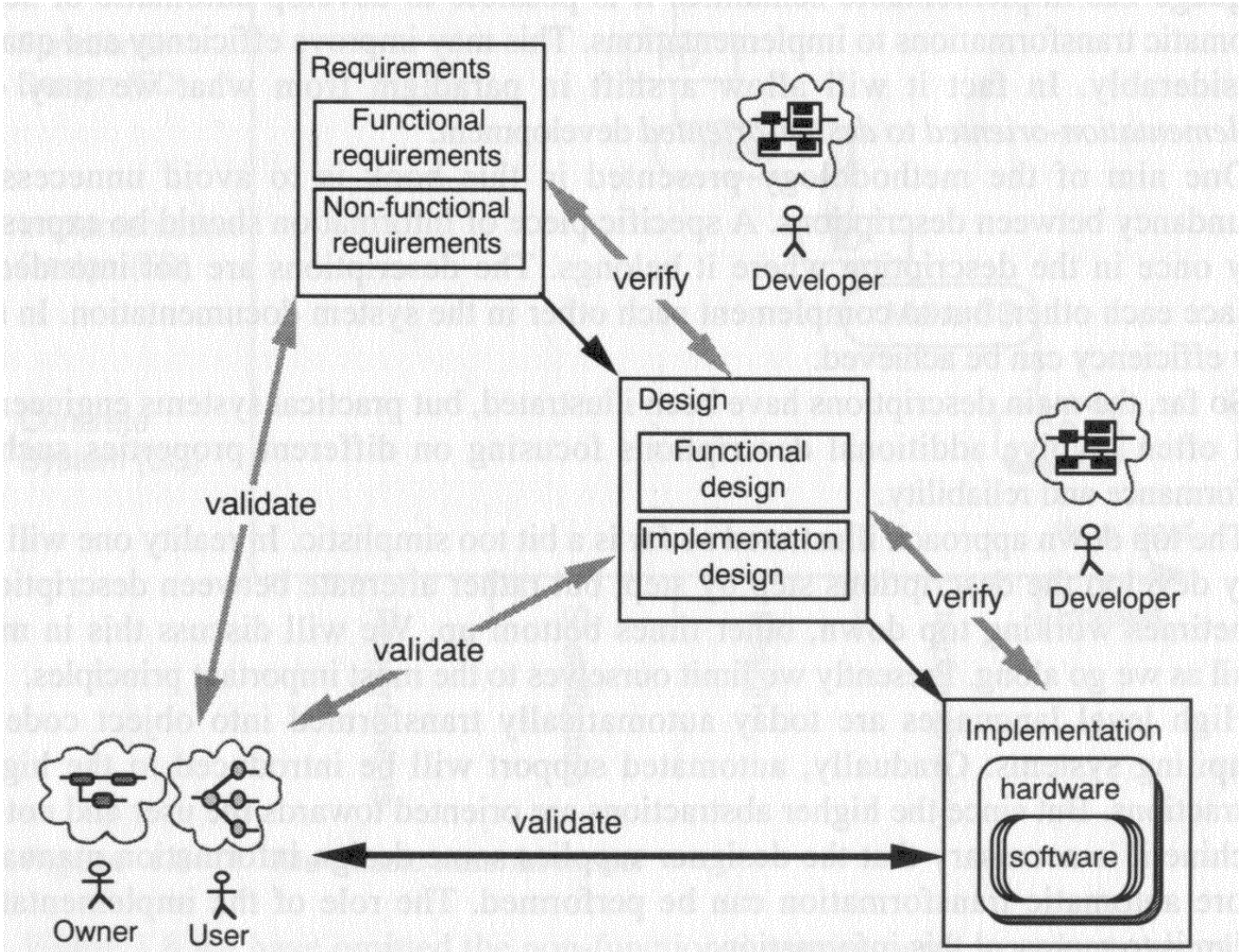
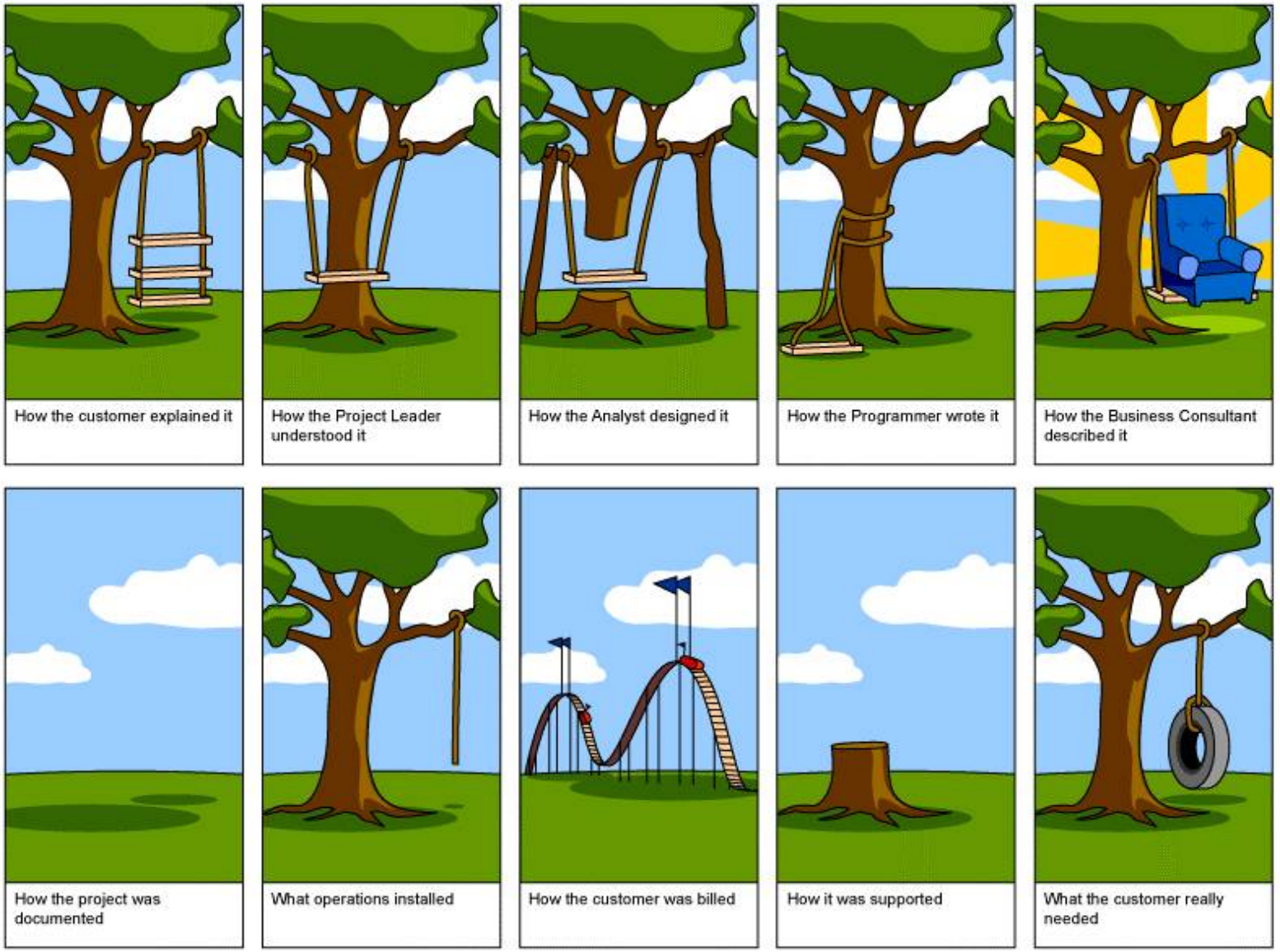


Figure 1.7 *Step-wise quality assurance*

What is the right system to build ?



RE activities and documents (Wiegers)

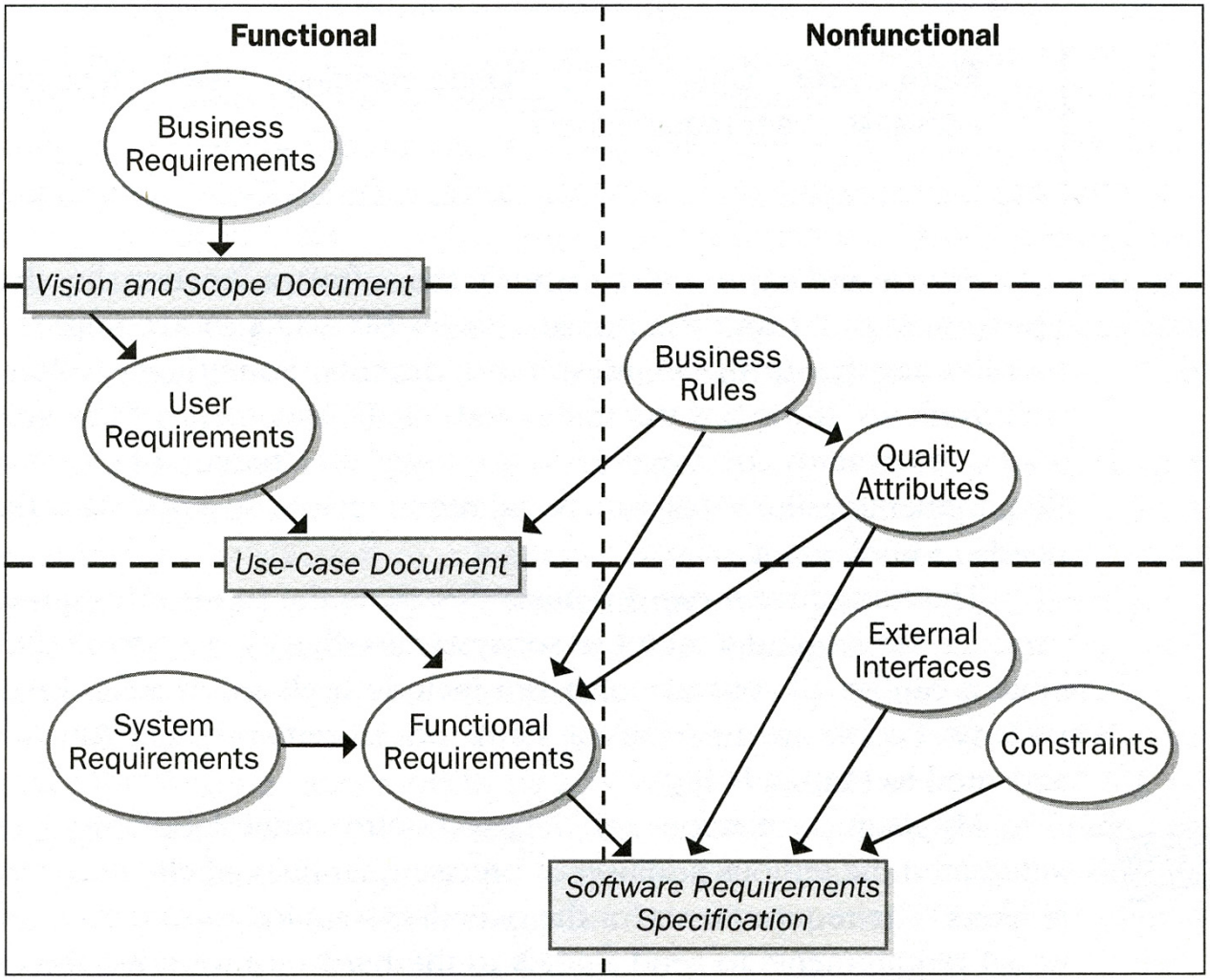


Figure 1-1 Relationship of several types of requirements information.

Notes on previous slide

- There needs to be an arrow from User requirements to System requirements. (The system has to be able to perform certain use cases. The same use cases must be supported by the software, therefore become Software requirements.)
- Business rules (including standards and regulations) are not only non-functional, they also include functional aspects (as shown by the arrows in the diagram).

RE process model

(suggested by Bray)

Again, this diagram shows

- RE activities (elicitation, analysis, specification, HMI design)
- subsequent design activity (internal design)
- RE documents (elicitation notes, requirements, specification, HMI specification)

Important point:

Distinction between

- Problem domain (described by requ. doc.)
- System (to be built) (described by spec. doc.)

Note: One has to distinguish between current (problematic) version of the problem domain, and the projected future version which includes the system to be built.

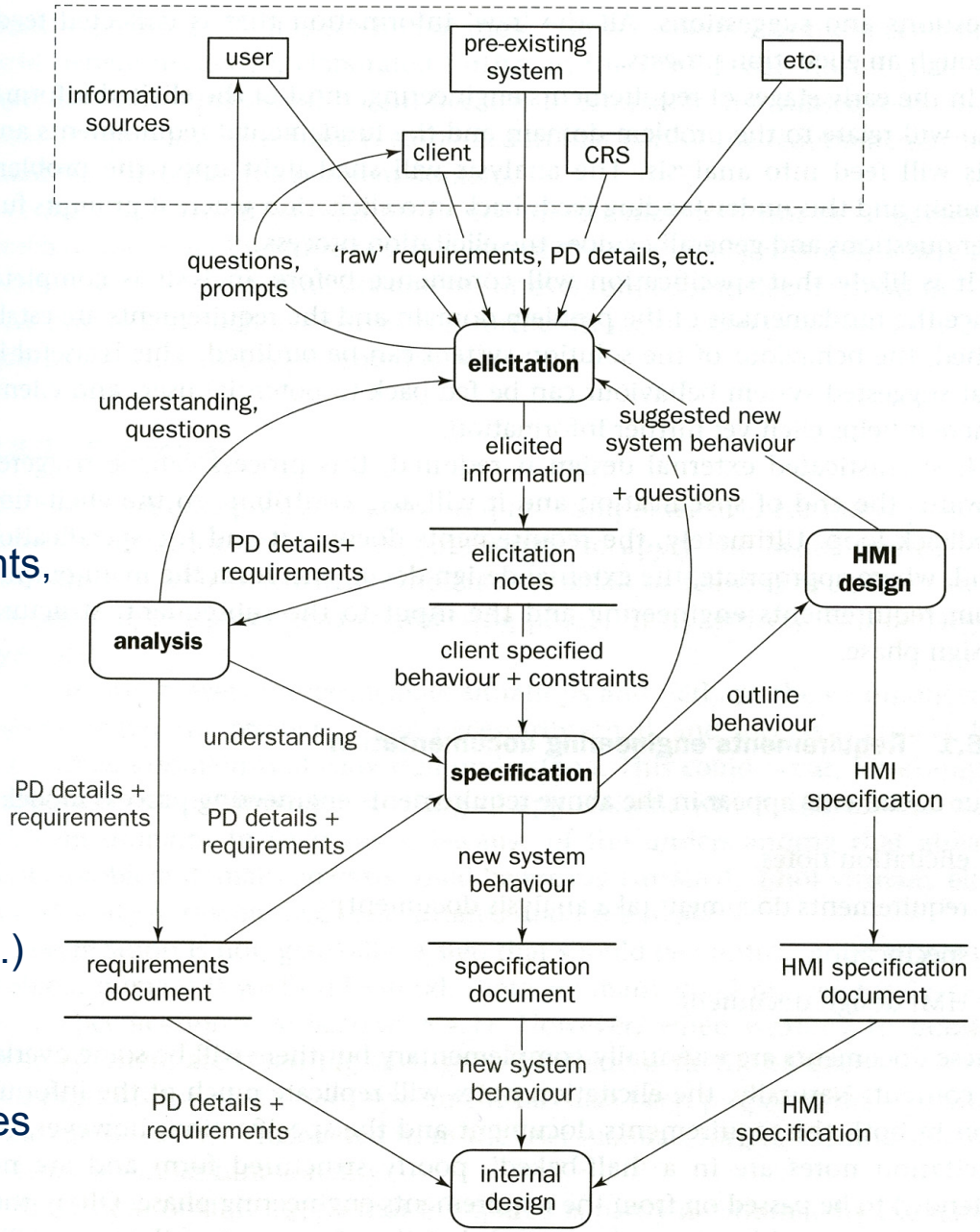
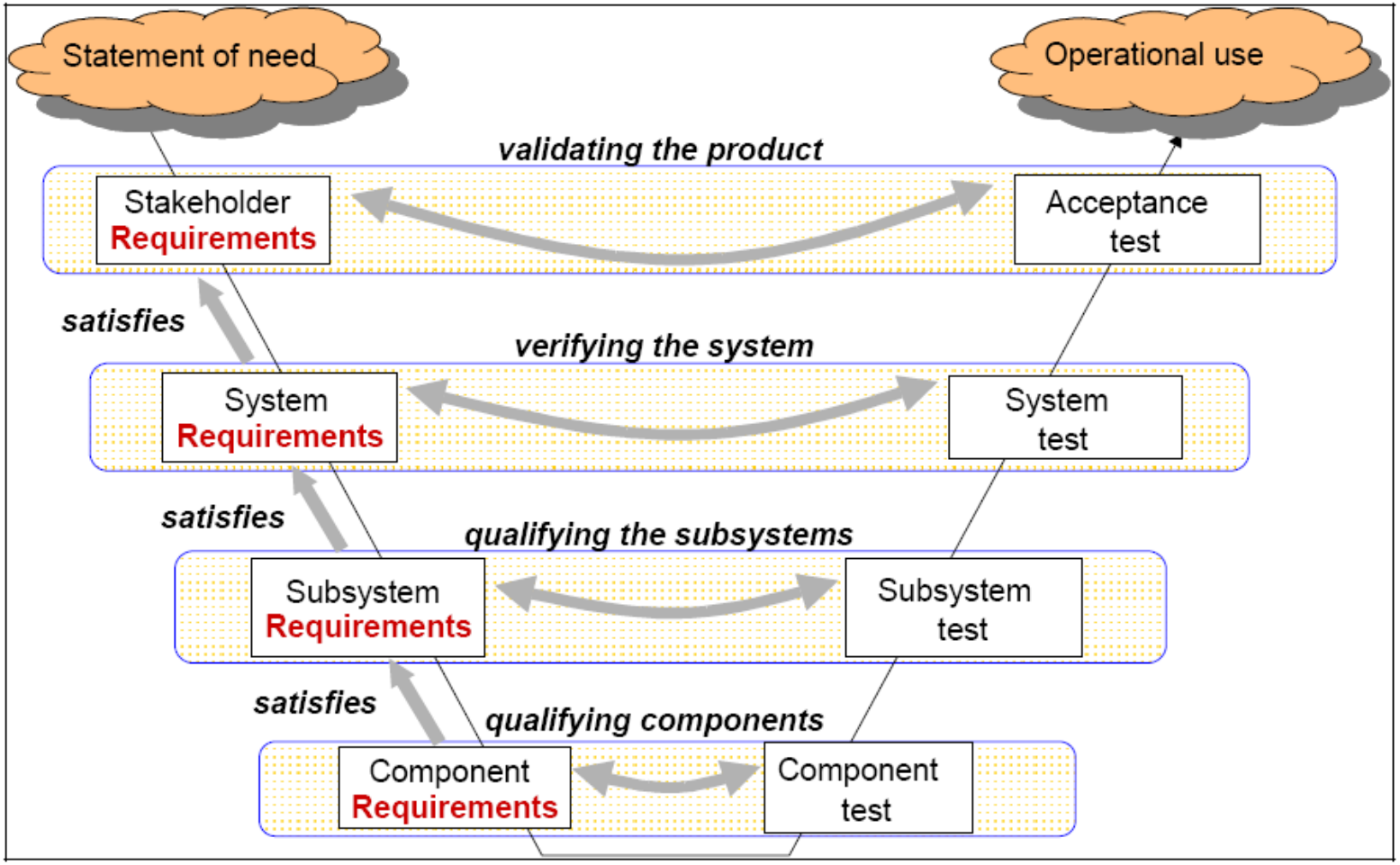


Figure 2.1
Note: CRS – client requirements specification

Typical Layered Approach (V-shaped)



Source: Hull, Jackson, Dick: Requirements Engineering, 2004

Notes on previous slide

- **This looks like the waterfall process model, but this diagram describes a quite different situation.**
- The layers correspond to step-wise refinement in terms of component decomposition.
- For instance, the transition from the first to the second layer is the typical RE process: one starts with the information from elicitation (shown in the first layer), that is, the problematic domain model, and one ends up with a proposal for a new system to be built (which is a component within the projected new domain model).
- **Important note:** The process of identification of the system to be built and defining its relationship with the new domain model (note that the environment of the system to be built may also be re-organized within the new domain model) is a kind of “design process” that requires creativity.
- The transitions to the lower layers in the diagram are similar processes (you may call them RE at a more detailed level or design processes)

Difference between RE and design ?

- Much research towards automated SE
  - Compilers automatically generate machine code (correct in respect to program source code)
  - CASE tools automatically generate implementations of UML State Machine models (correct in respect to the given model)
  - CASE tools automatically generate state machine models from a set of use case scenarios
    - E.g. PhD work of Dr. Somé
    - Tool for Live Sequence Charts by Dave - described in the book ”Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine”

Harel's "scenario-based programming" (1)

- Scenarios (use cases) are played into the tool, and may be played out for testing the recorded behavior model.

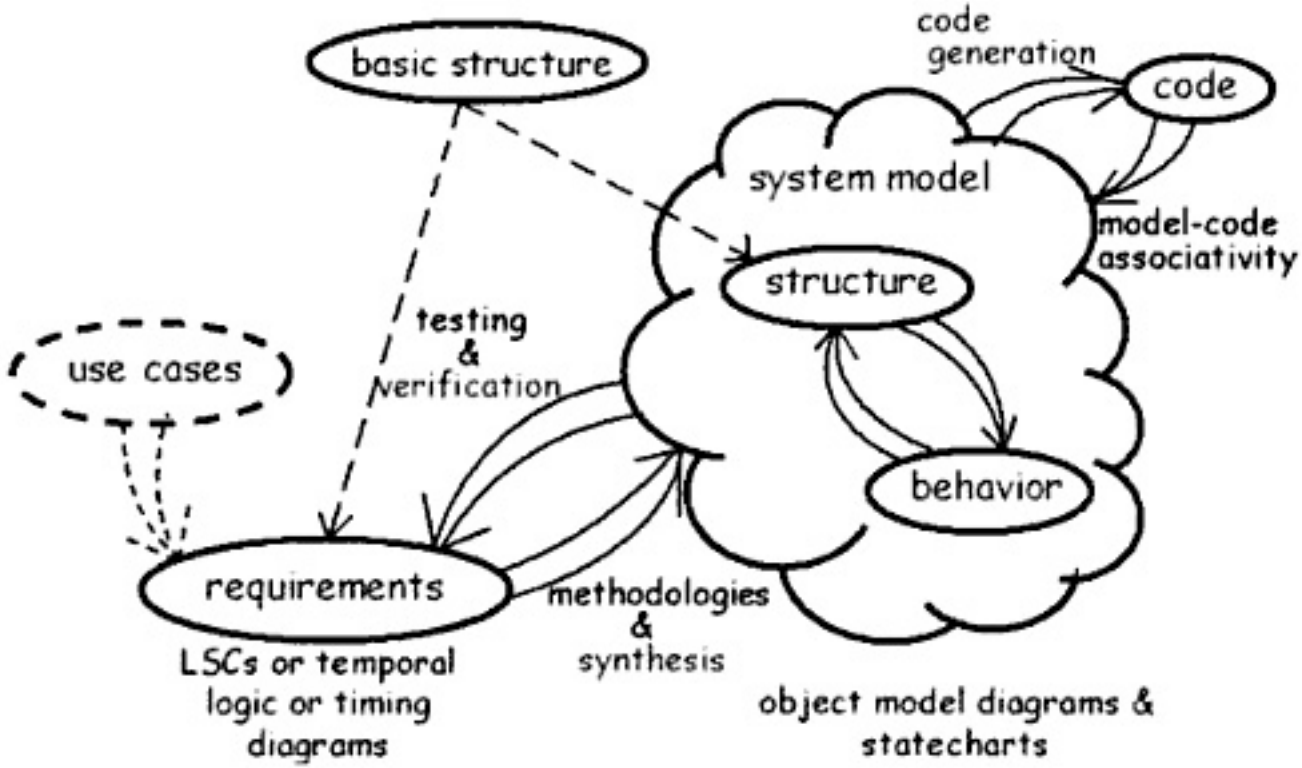


Fig. 2.4. Conventional system development

Harel's "scenario-based programming" (2)

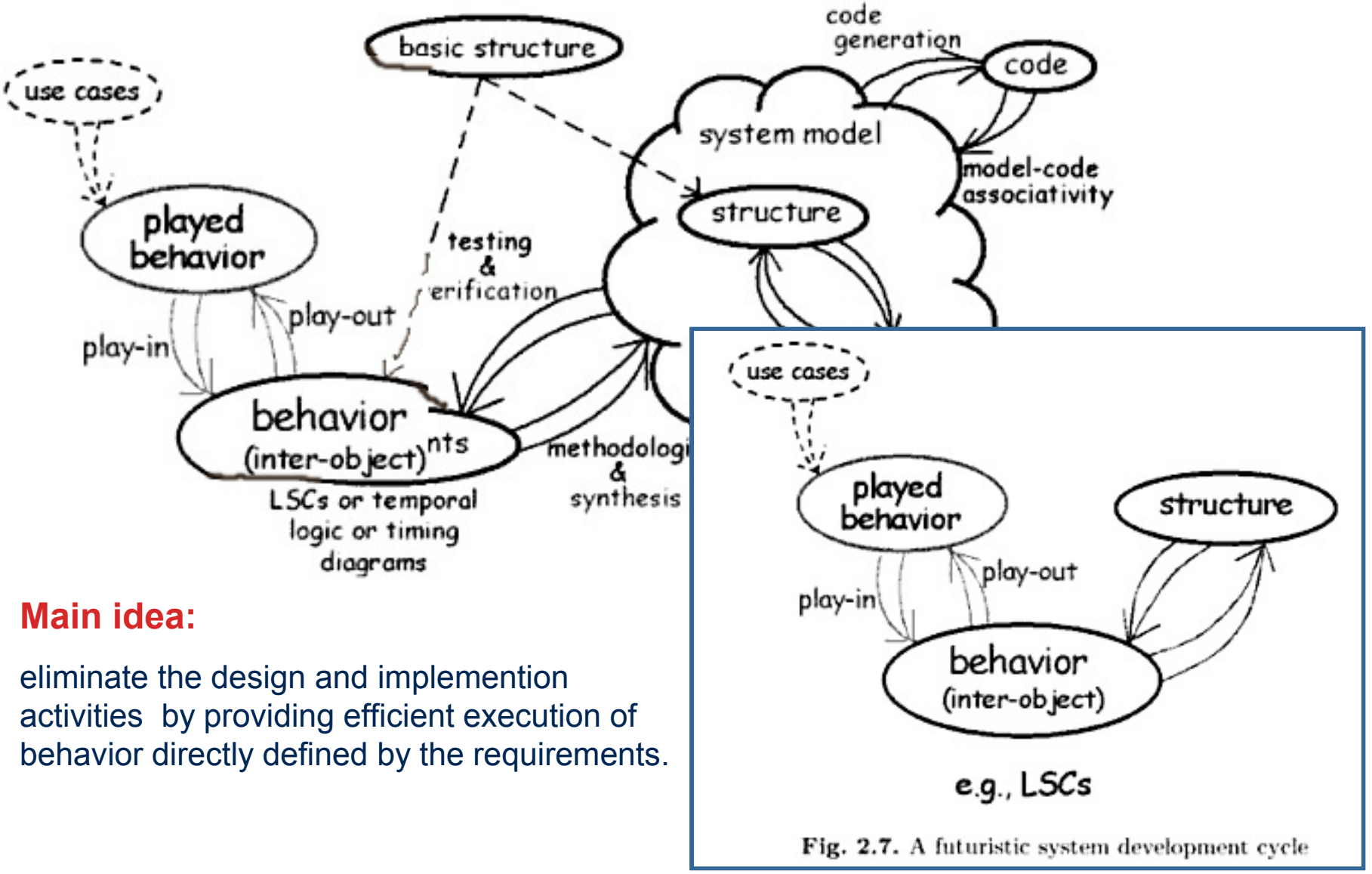


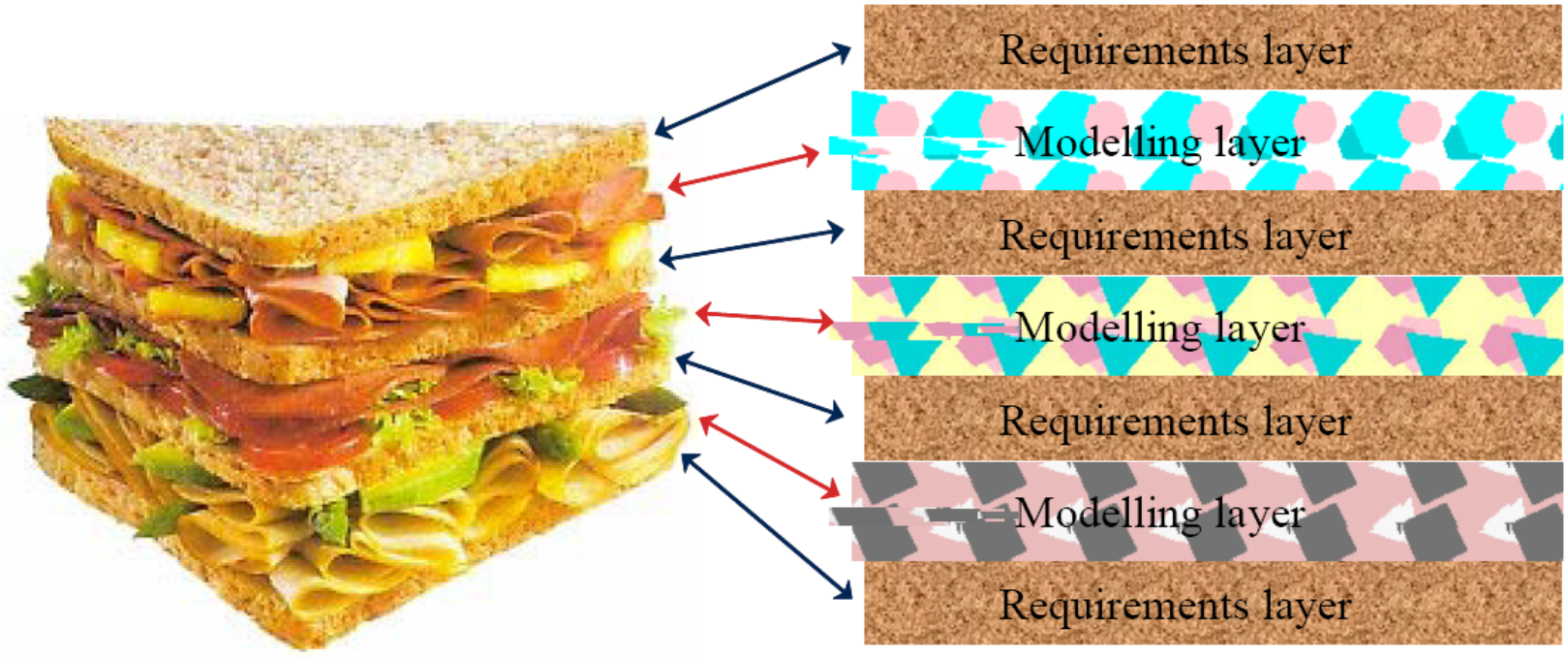
Fig. 2.7. A futuristic system development cycle

Main idea:

eliminate the design and implemention activities by providing efficient execution of behavior directly defined by the requirements.

Requirements and Modeling go together

- The systems engineering sandwich!



Source: http://www.telelogic.com/download/paper/SystemsEngineeringSandwich.pdf

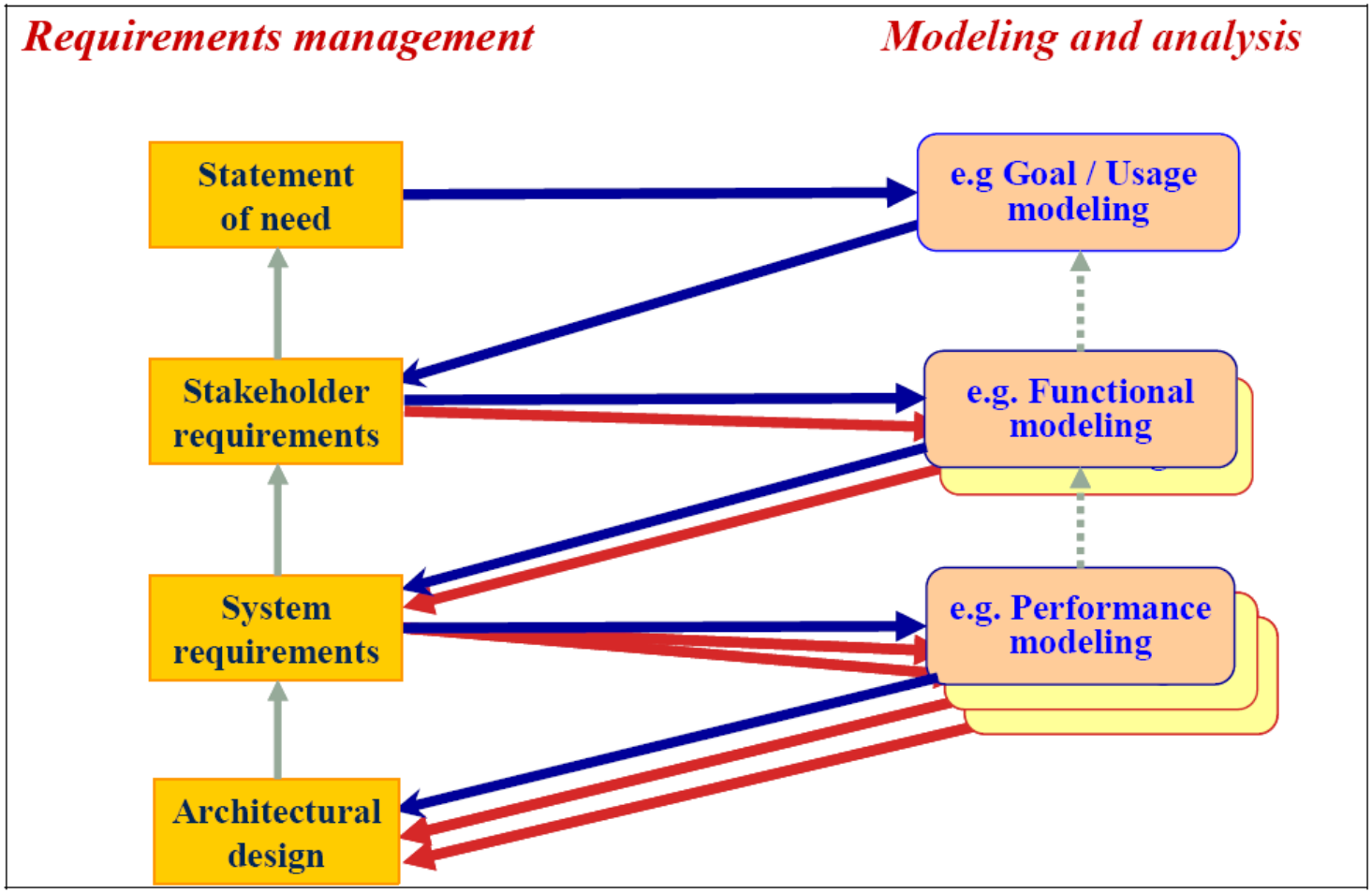
Comments on previous slide

Why combine RE with modeling ?

- **For analysis** – models help to understand the problem domain

- **For documentation** – models can be used for describing requirements (instead of solely using natural language)

Back to the Sandwich – consider different levels of details



Source: Hull, Jackson, Dick: Requirements Engineering, 2004

Benefits of Requirement Levels (Sandwich)

Principle of step-wise refinement:

- Focus the attention on the big picture before addressing the details
- Reduce the number of changes by specifying at a lower level the requirements that will not affect the requirements at a higher level
- Promote the division of work

This diagram [Lamsweerde] is another way to present this kind of (spiral) process

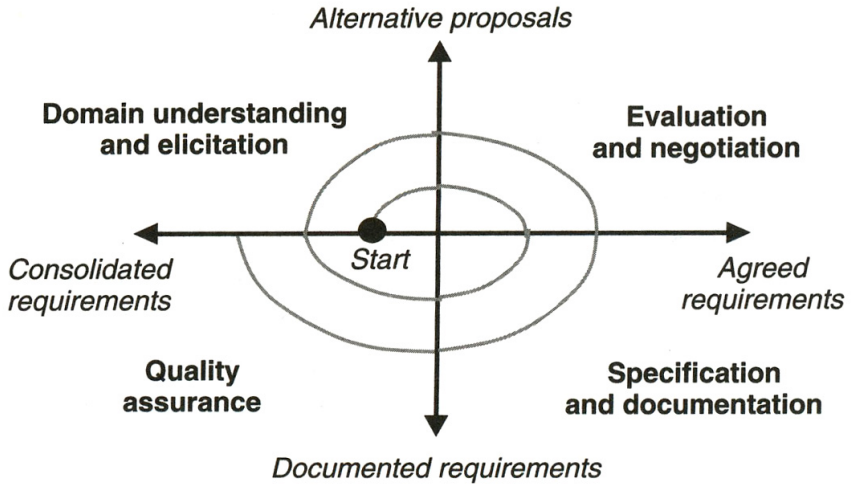
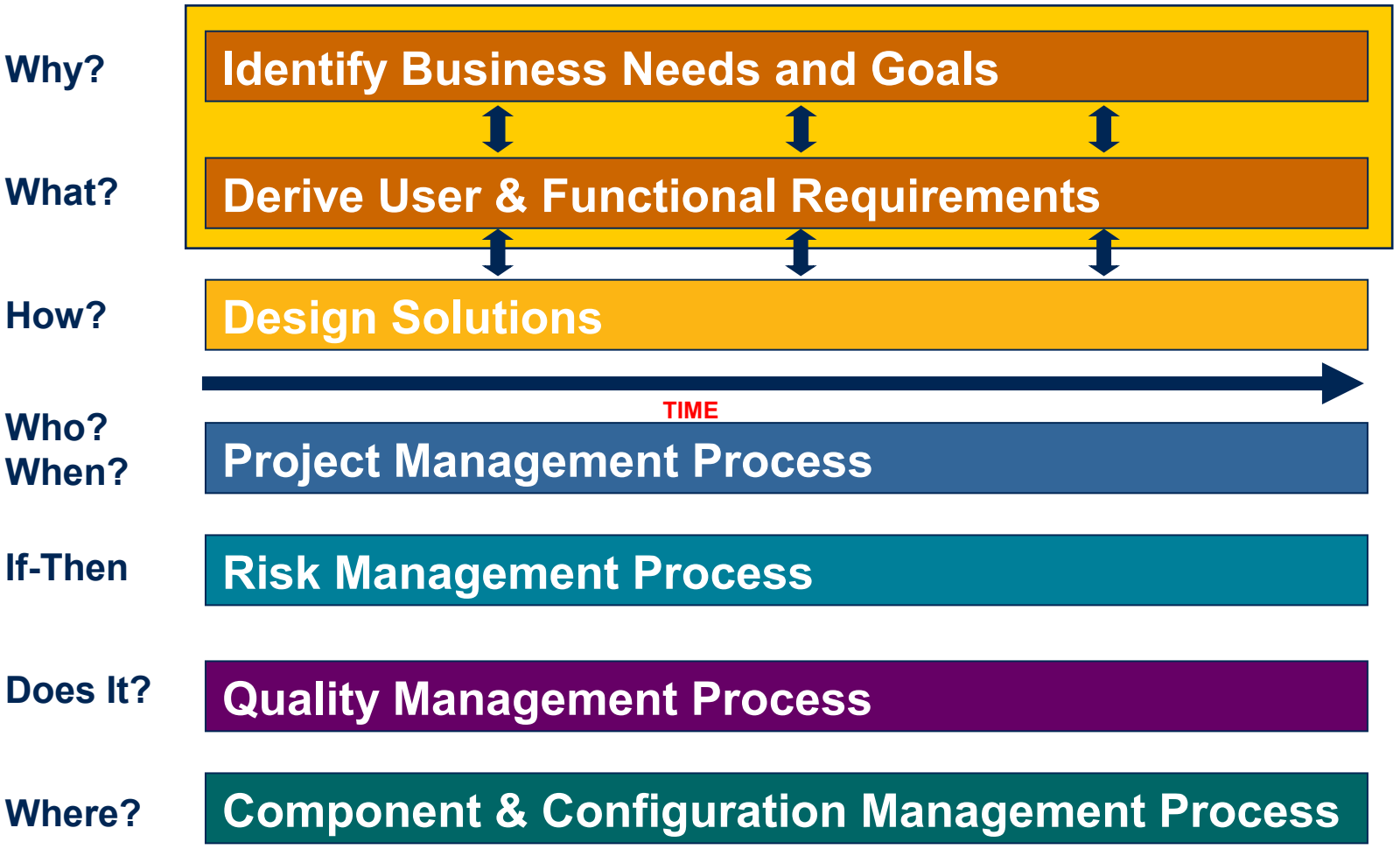


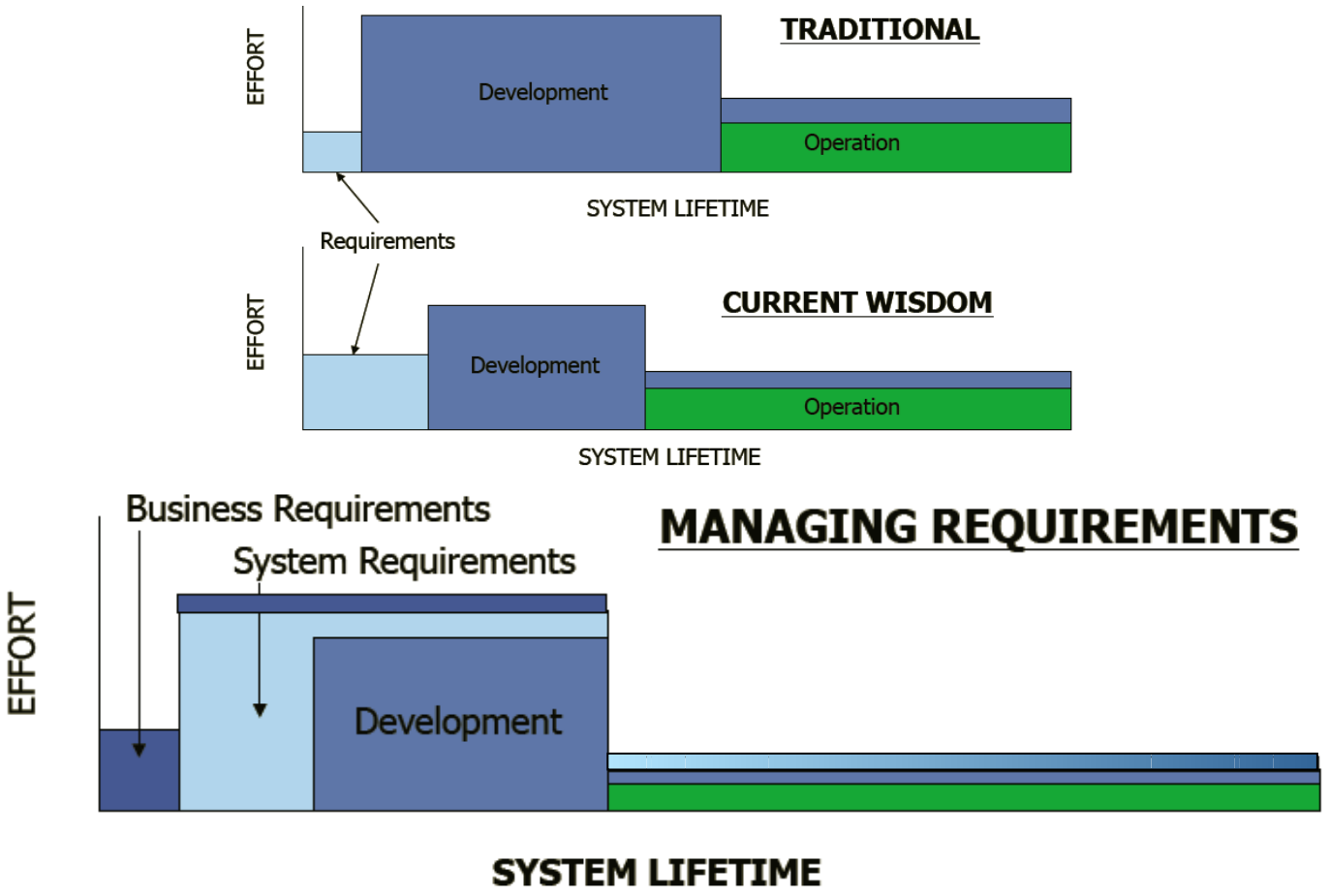
Figure 1.6 *The requirements engineering process*

RE Process and Related Activities



Requirements Engineering

- Requirements engineering is a set of activities but not necessarily a separate phase



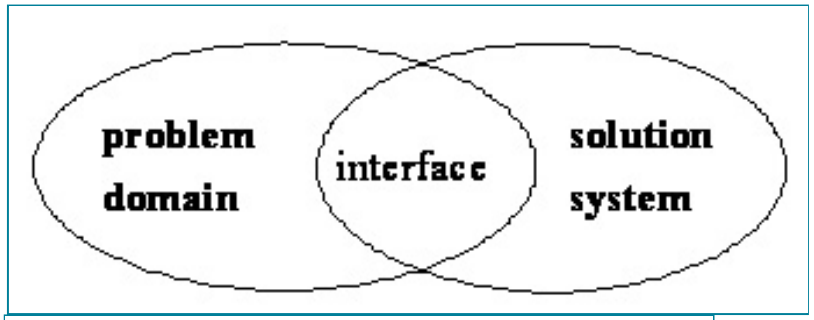
Source: Donald C. Gause, Risk Focused Requirements Management, Tutorial at RE'09, September 2009

The Problem Domain and the System/Software-to-be

Problem Domain

- The problem domain is the context for requirements
  - Part of the world within which the problem exists
  - Needs to be understood for effective requirements engineering
- Domain model
  - Set of properties assumed / believed to be true about the environment
  - Properties relevant to the problem
  - Problem domain requirements should hold in the proposed new version of the domain.
- Define the system requirements such that:
  - If the system that is built satisfies the system requirements and the environment satisfies the properties assumed for the environment, then the problem domain requirements will be satisfied.
  - In simple words: The system will behave as required if the assumptions hold.

Problem Domain and System-to-be



A domain model should be reusable
(Michael Jackson, 1995)

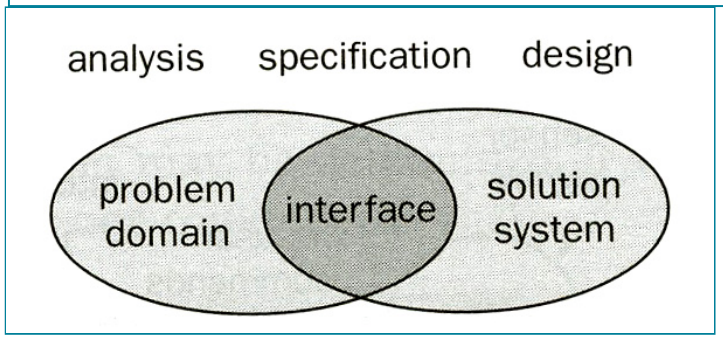


Diagram also showing activities [Bray]

better: *problem domain (as-is and to-be)*

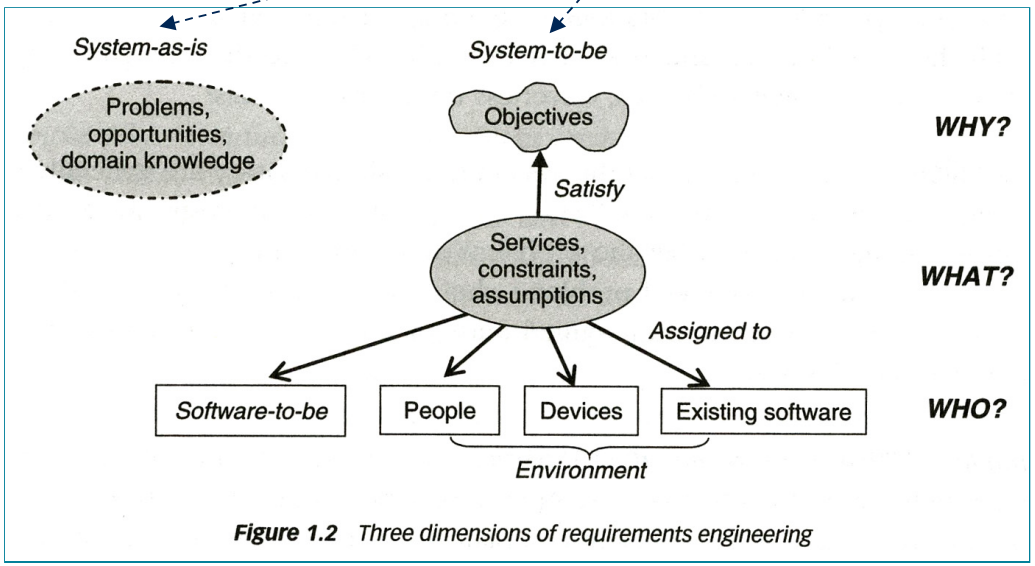
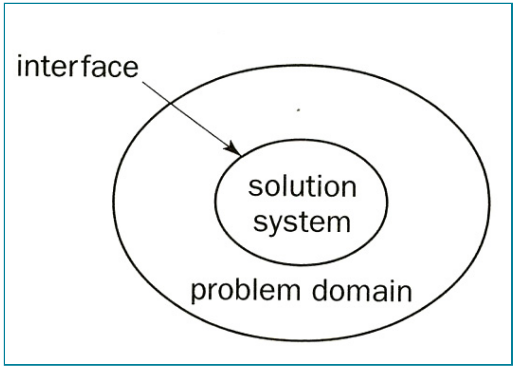


Diagram showing existing and future situation [Lamsweerde]



Problem domain with system-to-be [Bray]

System interface and software interface

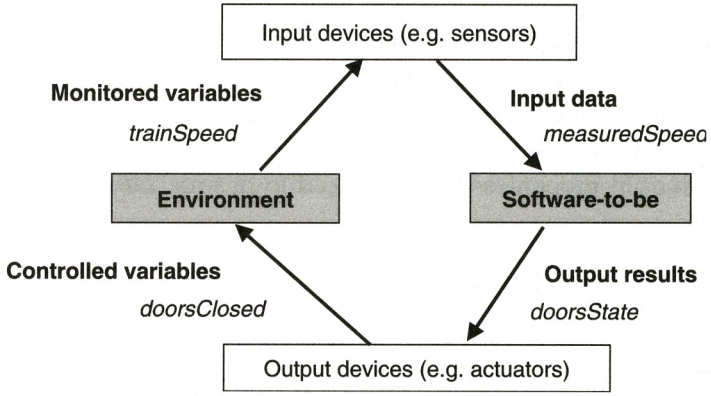


Figure 1.4 *Four-variable model*

Generic architecture of a control system including embedded software [Lamsweerde]

- System and software interface for a control system with embedded software:
  - Software interface: through input and output variables, for instance *measuredSpeed* (is read by program) and *doorState* (is set by program)
  - The system includes the software and I/O devices. Therefore the interface of the system with the environment are the monitored and controlled variables of the real world, for instance *trainSpeed* and *doorsClosed*.

Software objects representing real objects

- The software (model) normally contains objects that represent objects in the system environment (e.g. the doorState variable represents the state of the doors in the train)
- Whether they represent the situation in the environment correctly, is another question (for the doorState variable, this may depend on the correct functioning of the door state sensing device).

better: Problem domain requirements (if one considers the train to be the environment of the control system)

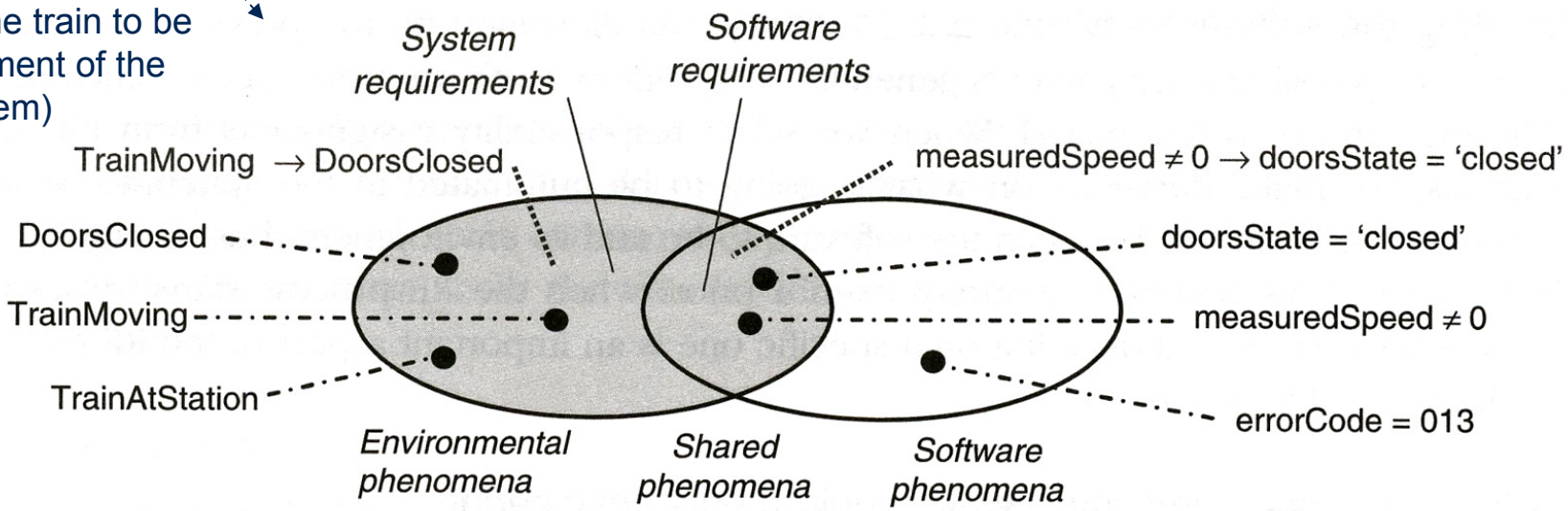


Figure 1.3 *Phenomena and statements about the environment and the software-to-be*

[Lamsweerde]

Main Requirements Activities

Requirements Inception

- **Start the process**
  - Identification of business need
  - New market opportunity
  - Great idea
- **Involves**
  - Building a business case
  - Preliminary feasibility assessment
  - Preliminary definition of project scope
- **Stakeholders**
  - Business managers, marketing people, product managers...
- **Examples of techniques**
  - Brainstorming, Joint Application Development (JAD) meeting…

Requirements Elicitation (1)

- **Gathering of information**
  - About problem domain
  - About problems requiring a solution
  - About constraints related to the problem or solution
- **Questions that need to be answered**
  - What is the system?
  - What are the goals of the system?
  - How is the work done now?
  - What are the problems?
  - How will the system solve these problems?
  - How will the system be used on a day-to-day basis?
  - Will performance issues or other constraints affect the way the solution is approached?

Requirements Elicitation (2)

- **Overview of different sources**
  - Customers and other stakeholders
  - Existing systems
  - Documentation
  - Domain experts
  - More ...
- **Overview of different techniques**
  - Brainstorming
  - Interviews
  - Task observations
  - Use cases / scenarios
  - Prototyping
  - More ...

Requirements Analysis

- The process of studying and analyzing the needs of stakeholders (e.g., customer, user) in view of coming up with a “solution”. Such a solution may involve:
  - A new organization of the workflow in the company.
  - A new system (system-to-be, also called solution domain) which will be used in the existing or modified workflow.
  - A new software to be developed which is to run within the existing computer system or involving modified and/or additional hardware.
- Objectives
  - Detect and resolve conflicts between requirements (e.g., through negotiation)
  - Discover the boundaries of the system / software and how it must interact with its environment
  - Elaborate system requirements to derive software requirements

Requirements Specification

- The invention and definition of the behavior of a new system (solution domain) such that it will produce the required effects in the problem domain
- Requirements Analysis has defined the problem domain and the required effects

- Specification Document
  - A document that clearly and precisely describes, each of the essential requirements (functions, performance, design constraints, and quality attributes) of the software and the external interfaces
  - Each requirement being defined in such a way that its achievement is capable of being objectively verified by a prescribed method (e.g., inspection, demonstration, analysis, or test)
  - Different guidelines and templates exist for requirements specification

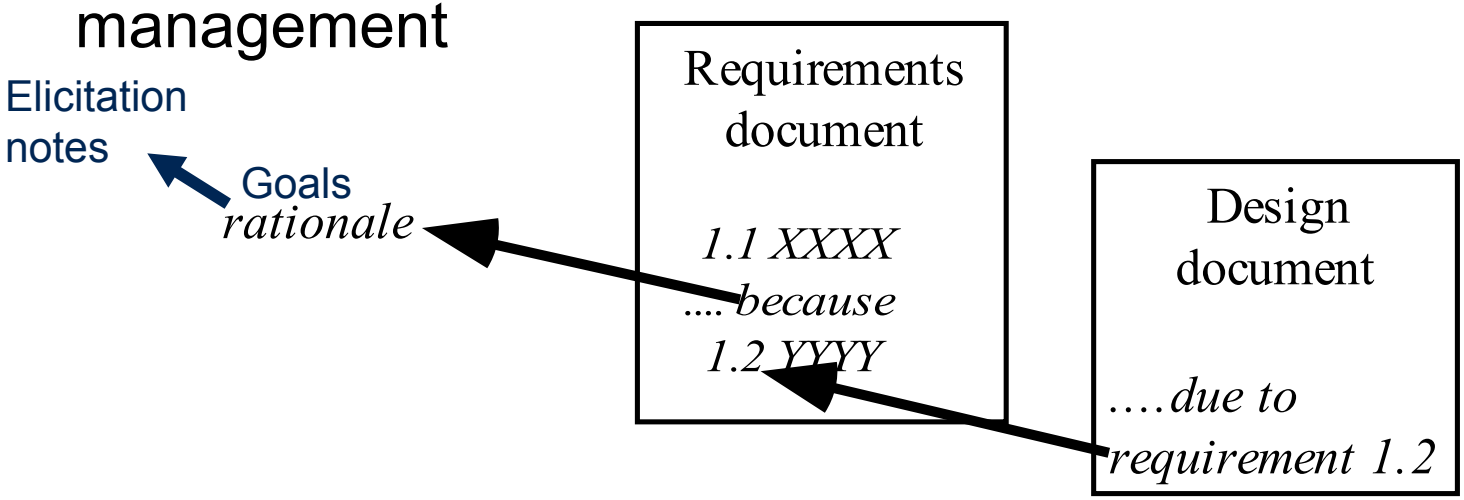
Requirements Verification and Validation

- Validation and verification
  - Both help ensure delivery of what the client wants
  - Need to be performed at every stage during the process
- Validation: checks that the right product is being built (refers back to stakeholders – main concern during RE)
- Verification: checks that the product is being built right
  - During design phase: refers back to the specification of the system or software requirements
  - During RE: mainly checking consistency between different requirements, detecting conflicts
- Techniques used during RE
  - Simple checks
  - Formal Review
  - Logical analysis
  - Prototypes and enactments
  - Design of functional tests
  - Development of user manual

Requirements Management

- Necessary to cope with changes to requirements
- Requirements change is caused by:
  - Business process changes
  - Technology changes
  - Better understanding of the problem

- Traceability is very important for effective requirements management



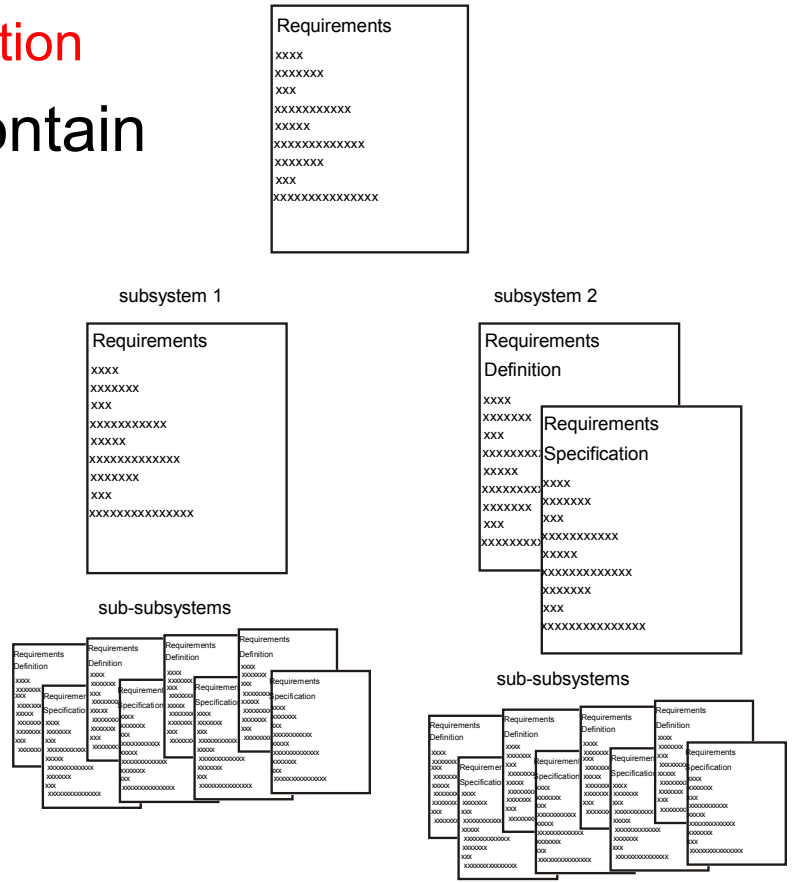
Requirements Documents

- Vision and Scope Document
- Elicitation notes: (Raw) requirements obtained through elicitation; often unstructured, incomplete, and inconsistent
- (Problem domain) Requirements document
- System requirements document
- Software requirements document
  - The software is normally part of a system that includes hardware and software. Therefore the software requirements are normally part of the system requirements.
- Note: System and Software requirements may exist in several versions with different levels of details, such as
  - User (customer) requirements: Statements in natural language plus diagrams of the services the system provides and its operational constraints; written for customers
  - Detailed requirements: A structured document setting out detailed descriptions of the system services; often used as a contract between client and contractor. This description can serve as a basis for a design or implementation; used by developers.

Types of Requirements Documents

Two extremes:

- An informal outline of the requirements using a few paragraphs or simple diagrams
  - This is called the requirements definition
- A long list of specifications that contain thousands of pages of intricate requirements describing the system in detail
  - This is called the requirements specification
- Requirements documents for large systems are normally arranged in a hierarchy



The Requirements Analyst[1]

- Plays an essential communication role
  - Talks to users: application domain
  - Talks to developers: technical domain
  - Translates user requirements into functional requirements and quality goals
- Needs many capabilities
  - Interviewing and listening skills
  - Facilitation and interpersonal skills
  - Writing and modeling skills
  - Organizational ability
- RE is more than just modeling…
  This is a social activity!

[1] Karl Wiegers, In Search of Excellent Requirements

For More Information

a. B. A. Nuseibeh and S. M. Easterbrook, Requirements Engineering: A Roadmap. In A. C. W. Finkelstein (ed) The Future of Software Engineering, ACM Press, 2000
http://www.cs.toronto.edu/~sme/papers/2000/ICSE2000.pdf

b. Simson Garfinkel, History's Worst Software Bugs, Wired News, 2005
http://www.wired.com/news/technology/bugs/0,2924,69355,00.html

c. INCOSE Requirements Working Group
http://www.incose.org/practice/techactivities/wg/rqmts/

d. Tools Survey: Requirements Management (RM) Tools
http://www.incose.org/productspubs/products/rmsurvey.aspx
http://www.volere.co.uk/tools.htm

e. IEEE (1993) Recommended Practice for Software Requirements Specifications. IEEE Std 830-1993, NY, USA.

f. IEEE (1995) Guide for Developing System Requirements Specifications. IEEE Std P1233/D3, NY, USA.

g. Requirements Engineering Conference
http://www.requirements-engineering.org/

Main References

a. Jeremy Dick, Elizabeth Hull, Ken Jackson: Requirements Engineering, Springer-Verlag, 2004

b. Soren Lauesen: Software Requirements - Styles and Techniques, Addison Wesley, 2002

c. Ian K. Bray: An Introduction to Requirements Engineering, Addison Wesley, 2002

d. Karl E. Wiegers: Software Requirements, Microsoft Press, 2003

e. Gerald Kotonya, Ian Sommerville: Requirements Engineering – Processes and Techniques, Wiley, 1998

f. Roger S. Pressman: Software Engineering – A Practitioner's Approach, McGraw-Hill, 2005

g. Tim Lethbridge, Robert Laganière: Object Oriented Software Engineering: Practical Software Developement using UML and Java, 2nd edition, McGraw-Hill, 2005

h. Ivy F. Hooks, Kristin A. Farry: Customer-Centered Products – Creating Successful Products Through Smart Requirements Management, Amacom, 2001

i. CHAOS Report, Standish Group