Software Requirements

L1. Introduction

Prof. W. Maalej and Dr. Farnaz Fotrousi

Overview

1. **Why Software Requirements?**
2. Types of Requirements
3. Requirements Engineering
4. Requirements Management

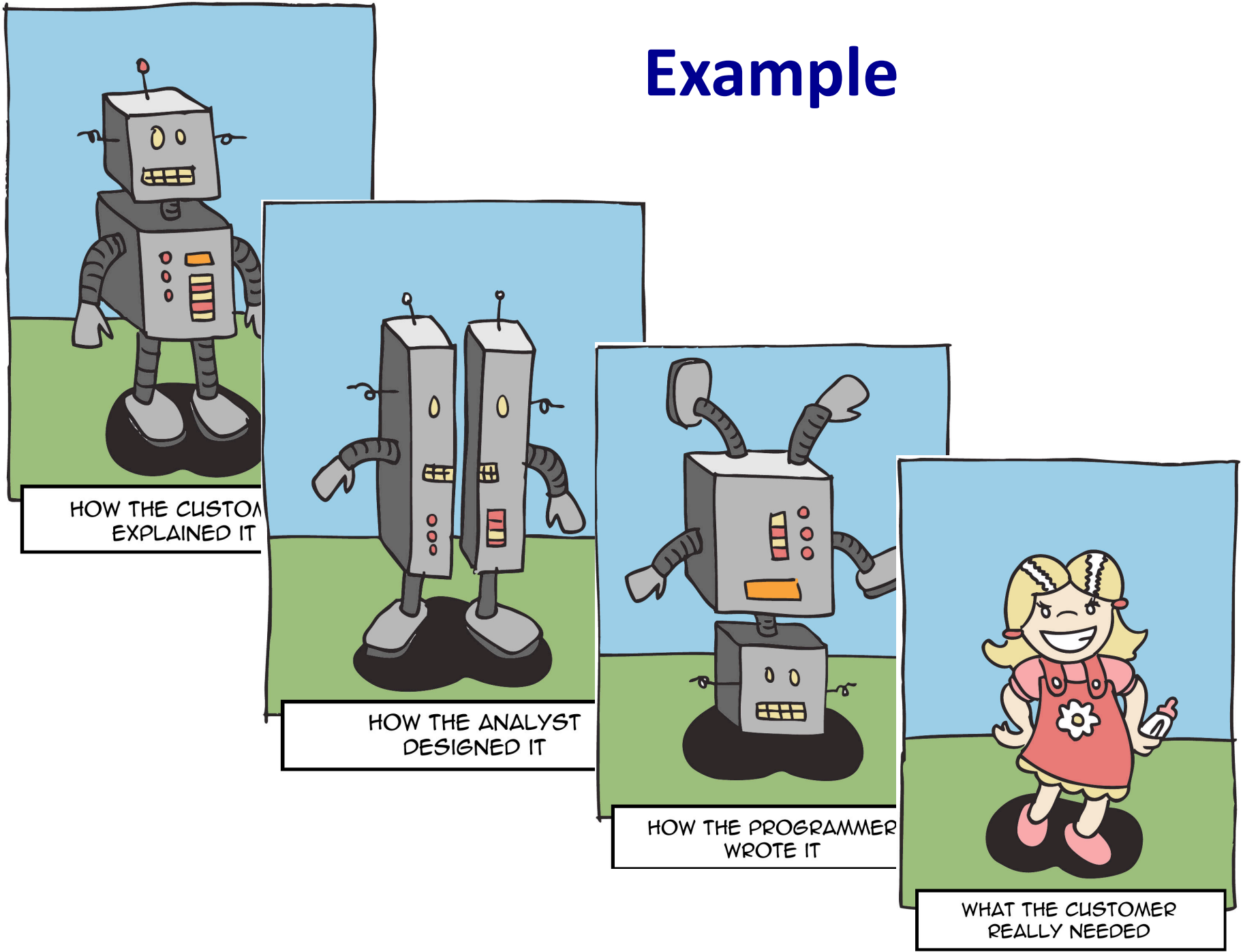
Requirements are hard

„The **hardest** single part of building a software system is **deciding** precisely **what** to build.”

[Brooks, 1987 & 2018]

Fred Brooks, Turing Award Winner

Example


HOW THE CUSTOM
EXPLAINED IT
HOW THE ANALYST
DESIGNED IT
HOW THE PROGRAMMER
WROTE IT
WHAT THE CUSTOMER
REALLY NEEDED

Expensive examples

Example 1: Ariane 5 explosion

- A large project of the European Space Agency
- Lasted for 10 years and cost about $7 billion
- Launched on June 4, 1996 (w. 4 satellites), exploded after 36 seconds

http://personal.victoria.ac.nz/stephen_marshall/SE/Failures/SE_Ariane.html

What happened?

- A piece of code reused from Ariane 4 **did not meet the requirements** of Ariane 5
- Ariane 5 accelerates faster than Ariane 4
- Variable overflow and crash of the flight control system
- Navigation system misinterpreted the crash as real flight data
- Extreme change in the trajectory damaged the rocket and activated the self-destruction
- The reused software **was not needed** for Ariane 5!

Example 2: Mars Climate Orbiter

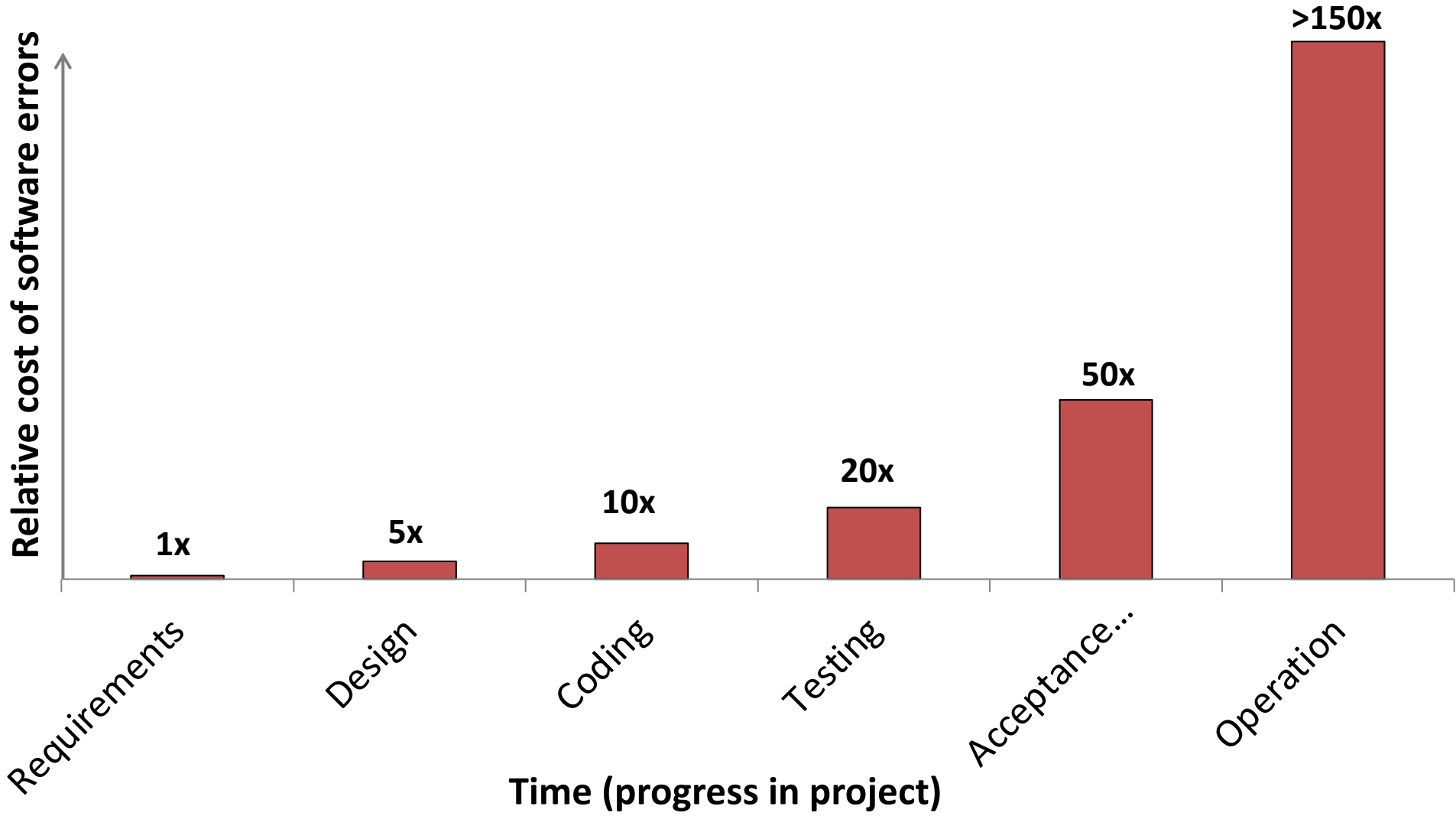
- Series of missions conducted by NASA to explore Mars
- US$327.6 million
- Project scheduled for a last 6 years
- Mission started in Dec. 1998 and arrived to Mars at Sep 23rd, 1999
- Orbiter was then lost
- Cause: Expected information in Newtons, received in Pounds

What happened?

- The orbiter needed to **modify its trajectory and slow down**
- This required a precise set of events which were carefully modeled in advance
- After the first burn, the orbiter lost contact with Earth
- Orbiter got **too fast too close to Mars** and was probably destructed by its atmosphere
- Principle cause of the failure was **confusion in units:** Expected information in **Newtons**, received in **Pounds**
- Error could have been discovered and solved during the flight. This did not happen due to **bad communication**

http://personal.victoria.ac.nz/stephen_marshall/SE/Failures/SE_MCO.html

Cost of software errors



[Software Engineering Economics by Boehm]

Requirements and user involvement are critical for software projects

Project managers surveyed for…

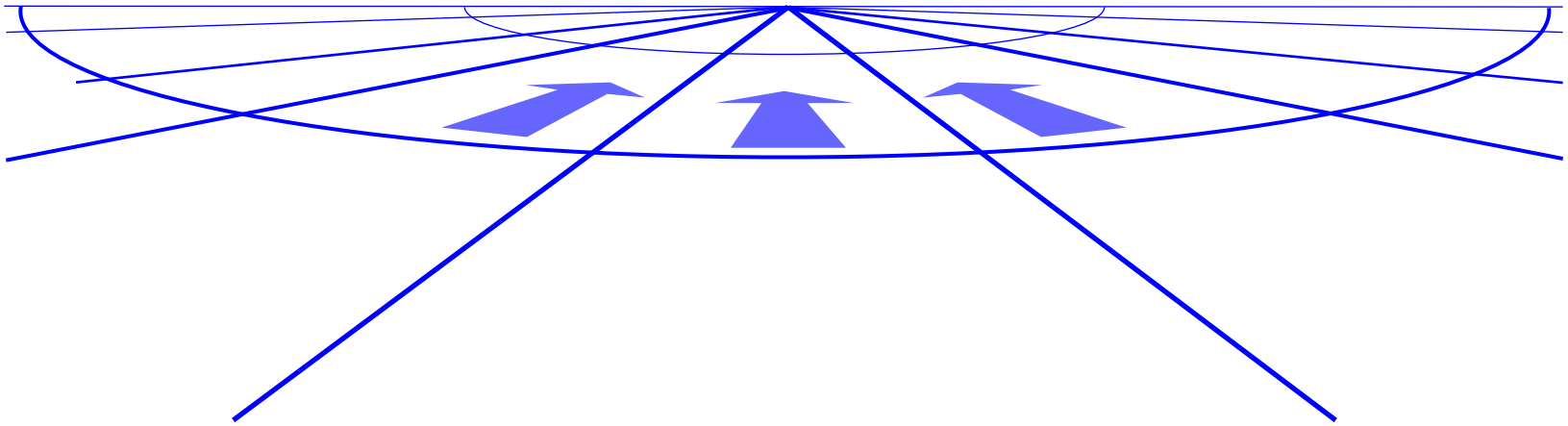
Project failure factors	
Incomplete Requirements	13%
No customer requirements	12%
Lack of resources	11%
Unrealistic expectations	10%
Uncontrolled changes of requirements	9%

Project success factors	
Customer/User involvement	16%
Management support	14%
Clear Statement of requirements	13%
Proper planning	10%
Realistic expectations	8%

[Standish Group 03] [Gartner Report 11]

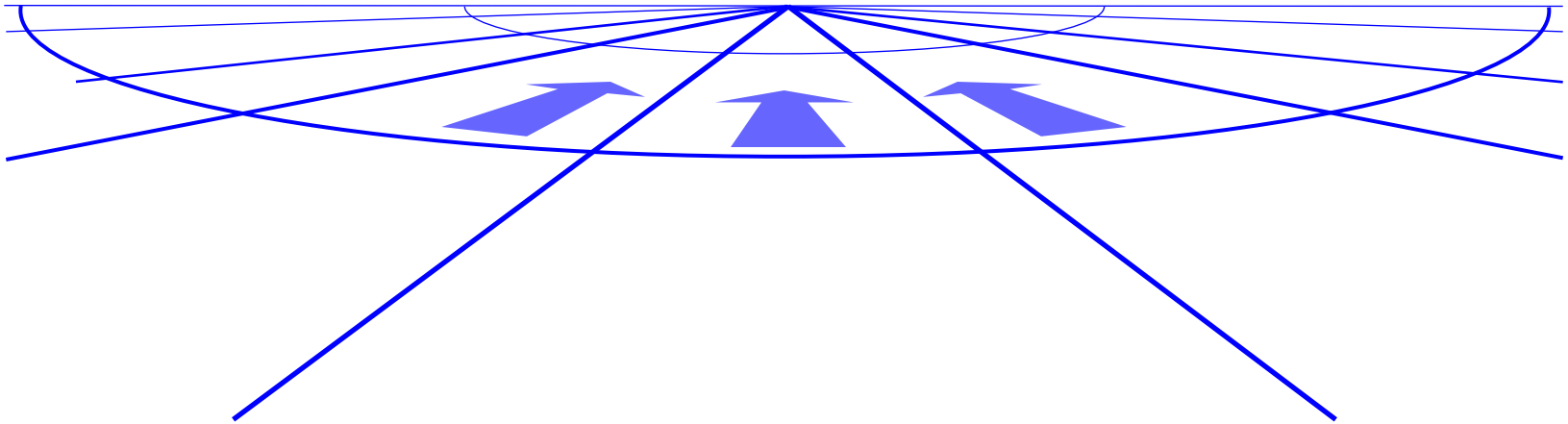
Objectives of the course "Software Requirements"

1. Understand the **importance of requirements** in software projects
2. Learn how to use **empirical & engineering techniques** to develop, analyze and manage requirements
3. Get insights about **current trends** in Software and Requirements Engineering research



Objectives of the module "Empirical Software Engineering"

1. Basic knowledge in **empirical methods** and how they can be **applied** in the field of software engineering
2. Applying empirical research to manage **complex systems** (data driven decisions)
3. Insights about the state of the art and **advance research topics** in software engineering and application



What is empirical research?

+

Observation **Data**

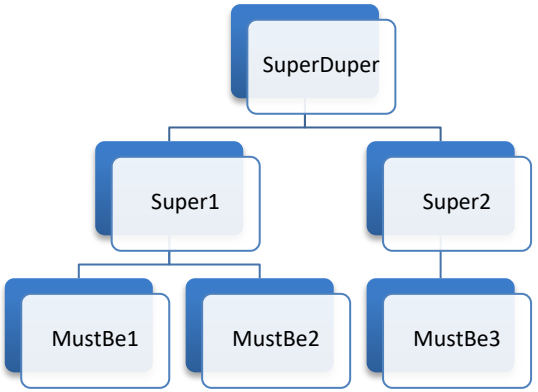
Systematic

The “new standard” in the SE community!

Other research approaches

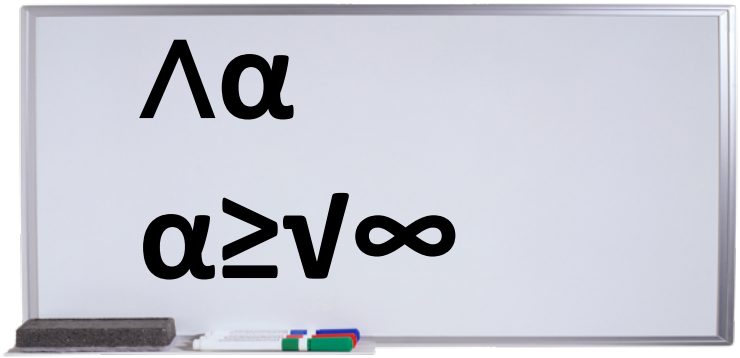
Engineering-driven

Analytical

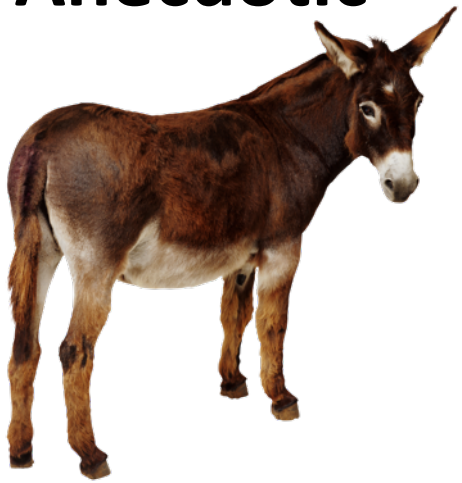


...

Mathematical, formal



Anecdotic



Goals of empirical studies

1. Explore

Understand phenomena and identify problems

2. Evaluate

Check and improve hypotheses, measure impact

Research strategies

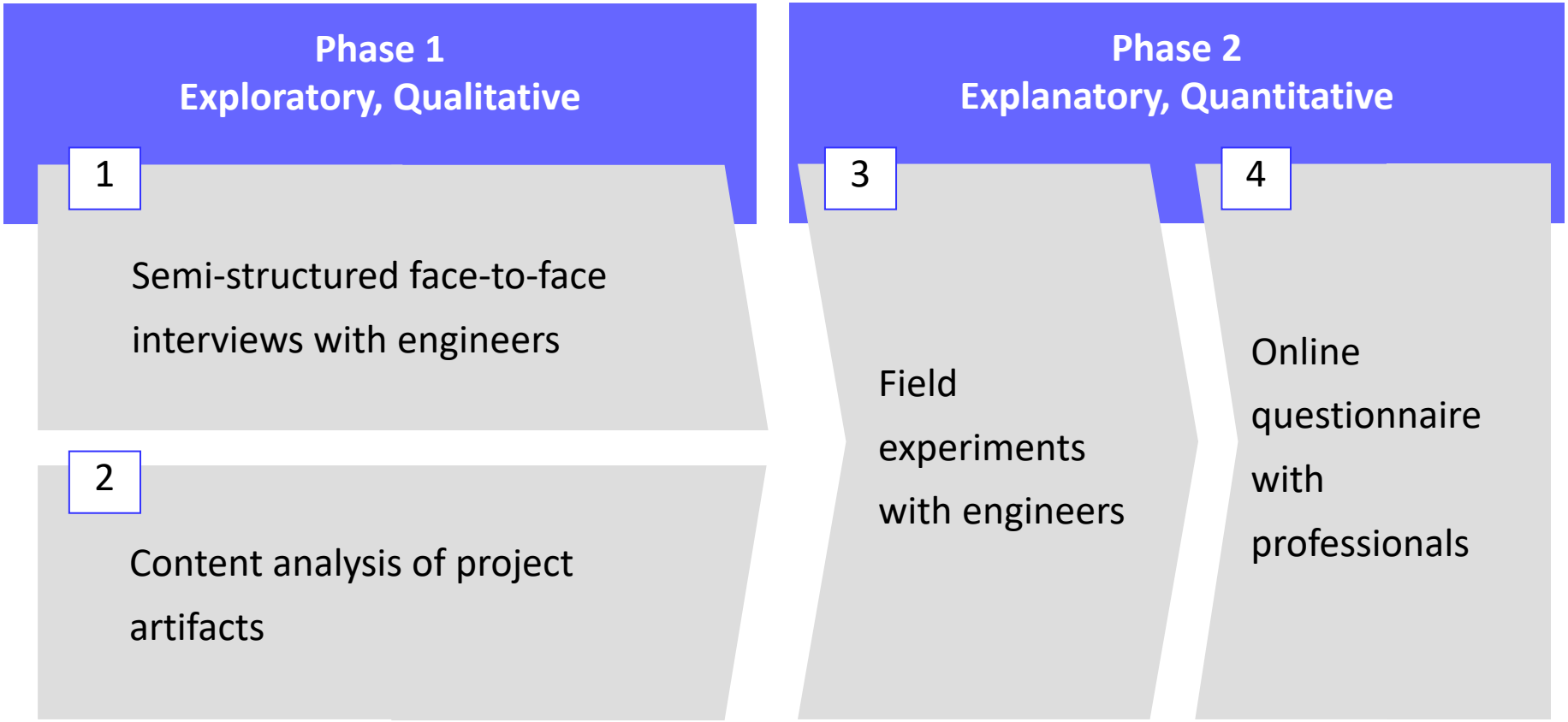
1. Qualitative

2. Quantitative

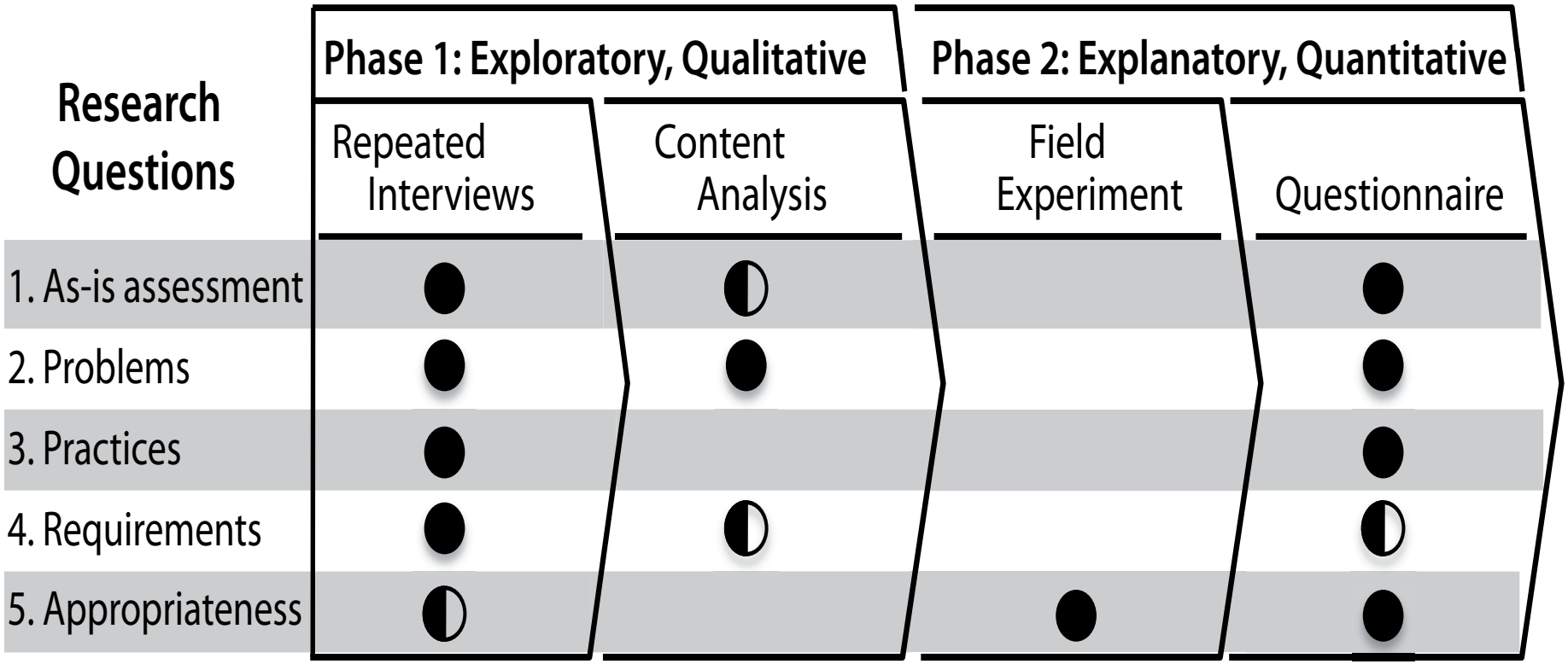
Both are important!

Combine: methodological pluralism, triangulation

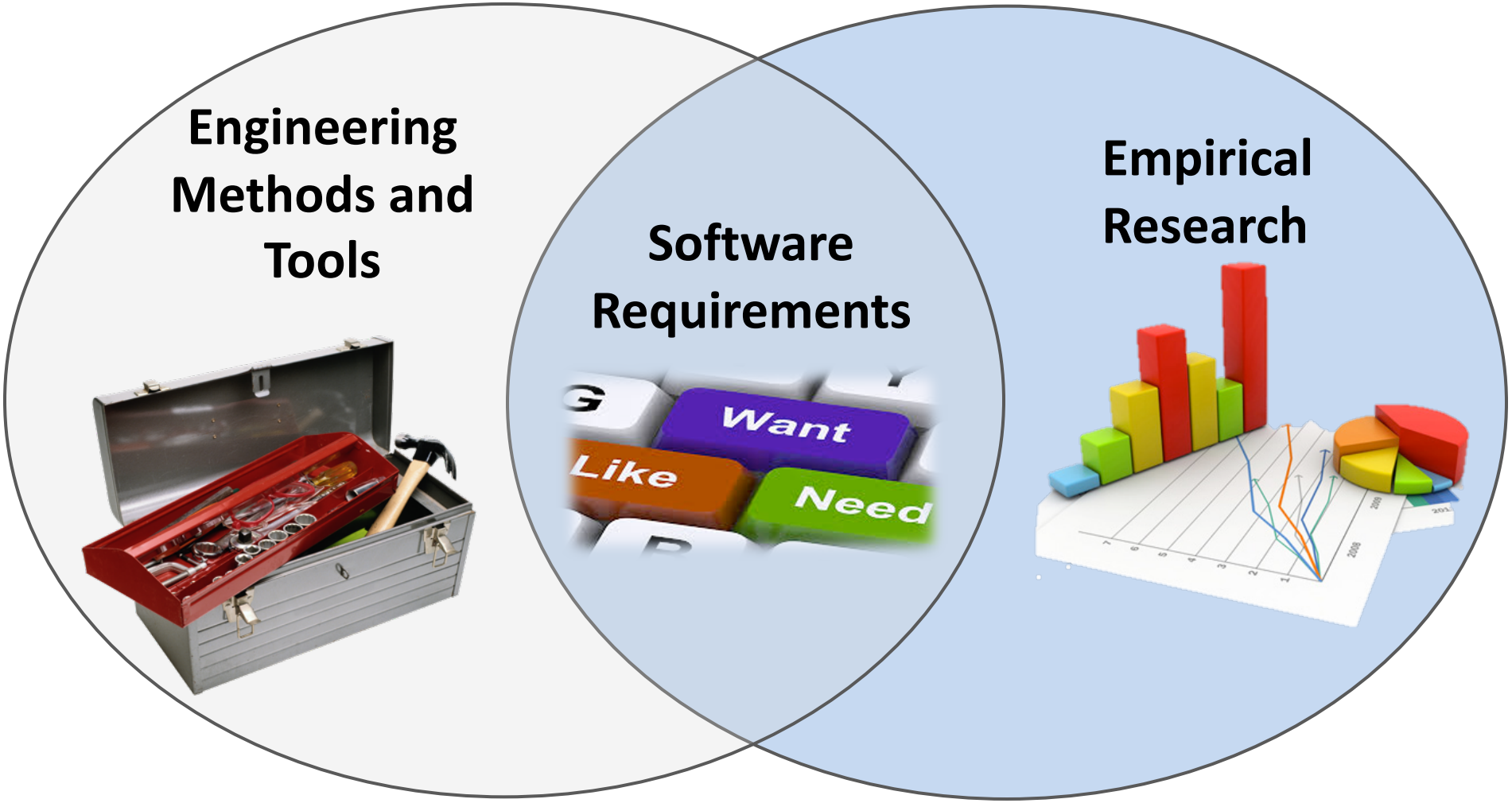
Example: Tool integration revisited [Maalej, 2009]



How it was presented in the paper



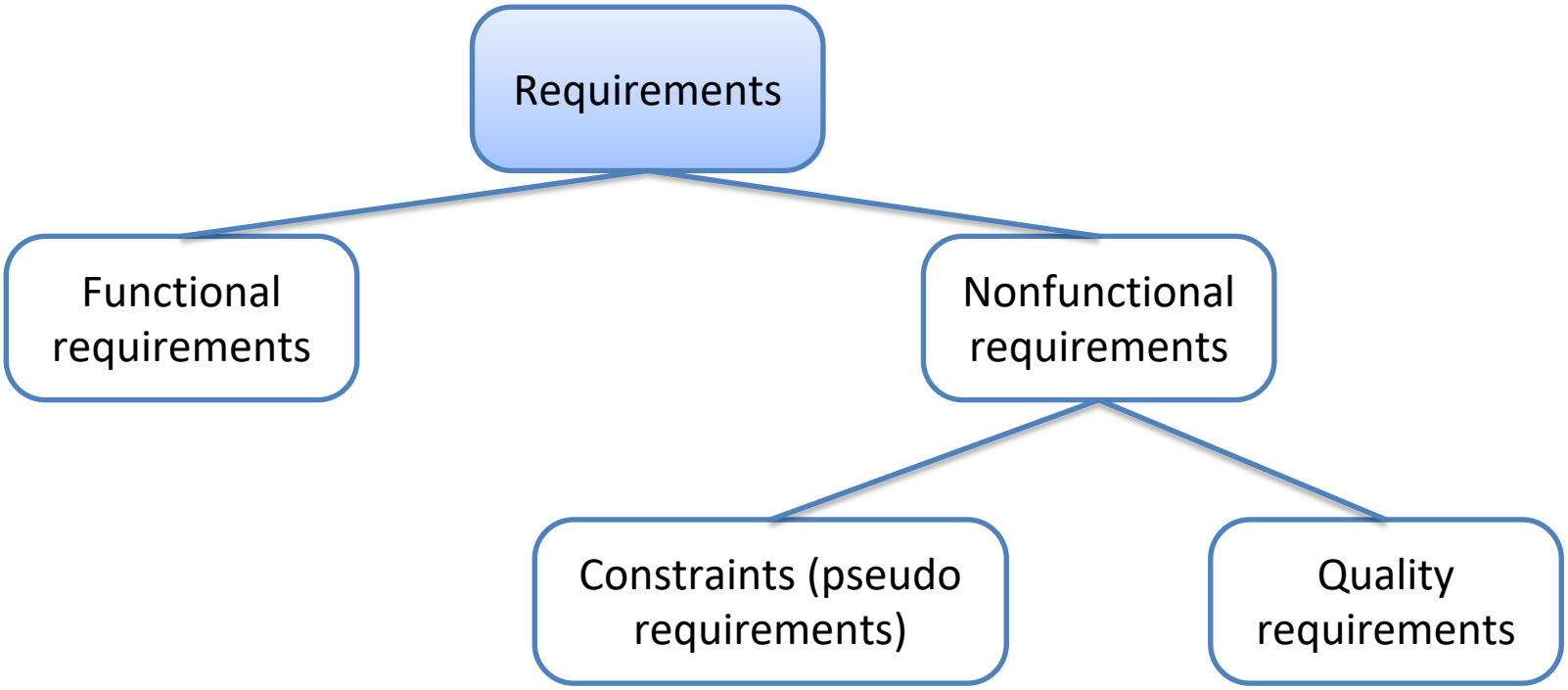
Requirements & empirical research?



Overview

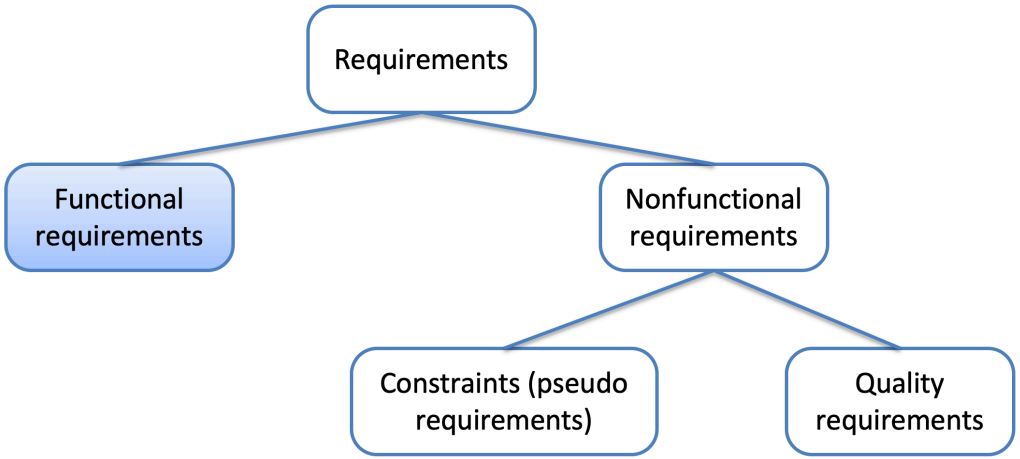
1. Why Software Requirements?
2. **Types of Requirements**
3. Requirements Engineering
4. Requirements Management

Types of requirements



Functional requirements

- Describe the **interactions** between the system and its environment independent from the implementation
- Describe user tasks which the system needs to support
- Describe system behaviour under specific condition

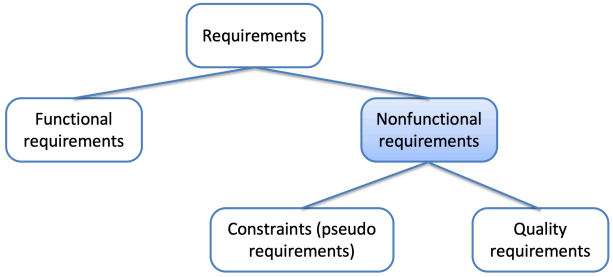


Functional requirements: example

- Example: Smart-parking app that recommends a rout in a map to a free parking slot
  - The user shall be able to choose her destination on the map
  - The system shall recommend routs to a free parking slot.
  - The user shall choose one of the recommended routs
  - The system shall “brighten” display in darkness
  - The system shall give a warning message when the battery is less than 10%

Non-functional requirements

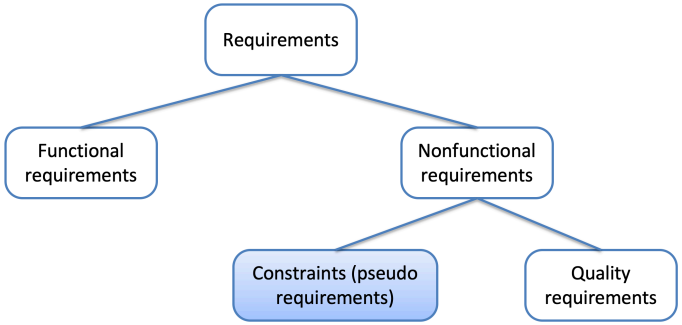
- Describe aspects not directly related to functional behavior
- Example: “The Feature X of the App must be available without a mobile internet connection”
- Describe properties of the system or the domain
- Phrased as constraints or negative assertions
  - “All data **must** remain after battery is empty”
  - “Apps **should not** induce energy consumption when they are in idle mode”
  - “All user inputs **should** be acknowledged within 1 second”



Non-functional requirements: Constraints

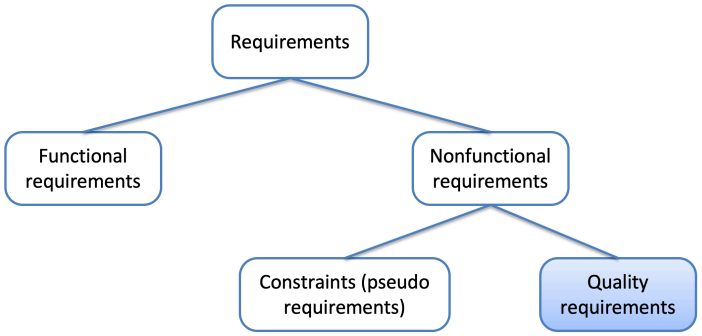
- Imposed by the customer or the environment
- Also called **“Pseudo requirements”**
- Example: “The app must support blinded users using speech“
- Affect different areas
  - Implementation
  - Interface requirements
  - Operation requirements
  - Packaging requirements
  - Legal requirements

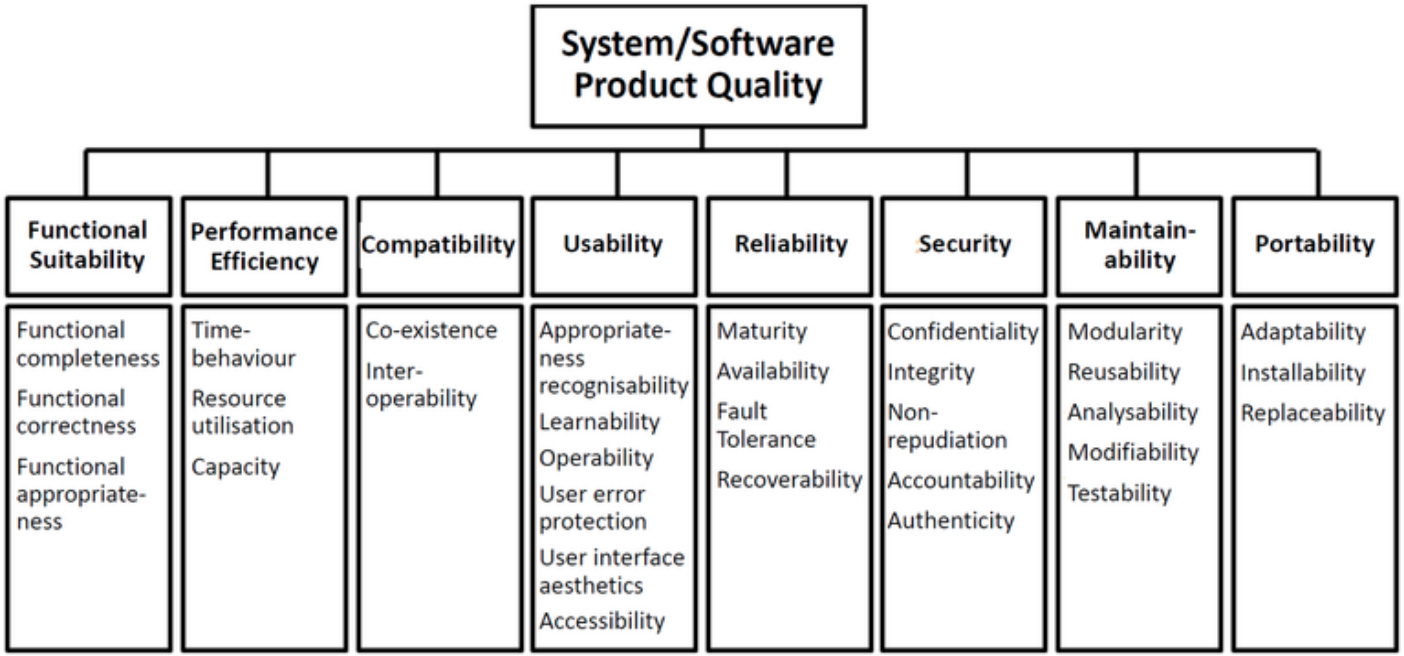
  (e.g., privacy, accessibility, licensing, certification, regulation)



Non-functional requirements: Quality Requirements

- Describing the quality of the software
- ISO-IEC *ISO*/IEC *25010* proposes a relevant category for System/software quality
- ISO-IEC *ISO*/IEC *25010* cancels and replaces *ISO*/IEC 9126-1:2001





Usability (Learnability)

Example: A user should be able to purchase a ticket in 3 step max. The purchase should last for less than 1 minute.

Reliability (Availability)

Example: The system shall have a minimal availability rate of 99%. The system should not be offline for more than X seconds.

Performance Efficiency (Throughput)

Example: The system shall be able to receive 1000 messages within 5 seconds and forward them to all registered users.

Portability (Adaptability)

Example: The game operator must be able to add new games without modifications to the existing system.

Requirements is about asking the right questions

Questions about users

- What type of user will be using the system?
- Will more than one type of user be using the system?
- What training will be required for each type of user?
- Is it important that the system is easy to learn?
- Should users be protected from making errors?
- What input/output devices are available?
- What kind of documentation is required?
- What audience is to be addressed by each document?

Questions about hardware and performance

Hardware considerations

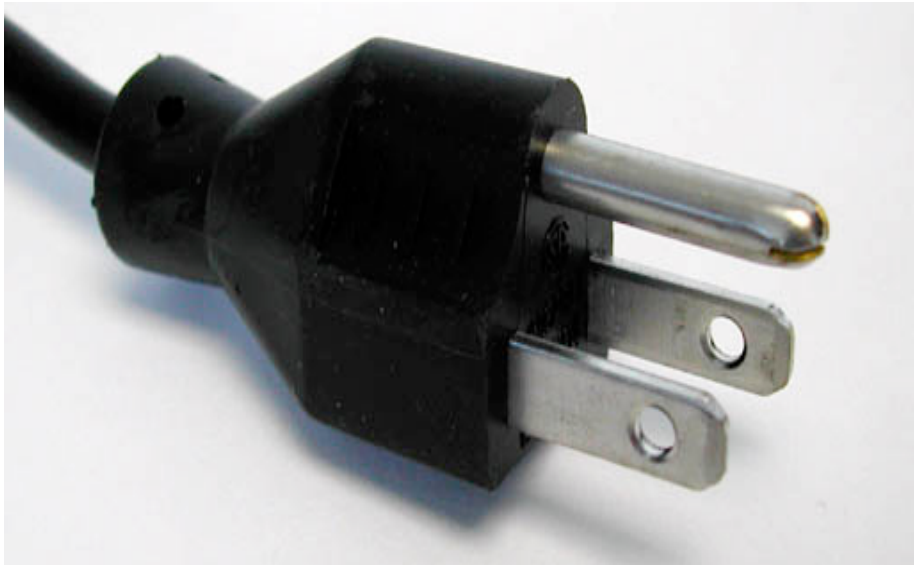
- What hardware is the proposed system to be used on?
- What are the characteristics of the target hardware, including memory size and auxiliary storage space?

Performance characteristics

- Are there speed, throughput, response time constraints on the system?
- Are there size or capacity constraints on the data to be processed by the system?

Questions about the system interfaces

- Is input coming from systems outside to the proposed system?
- Is output going to systems outside the proposed system?
- Are there restrictions on the format or medium that must be used for input or output?



Questions about quality

Quality issues

- What are the requirements for reliability?
- Must the system trap faults?
- What is the time for restarting the system after a failure?
- Is there an acceptable downtime per 24-hour period?

Error handling and extreme conditions

- How should the system respond to input errors?
- How should the system respond to extreme conditions?

Questions about the environment

Physical Environment

- Where will the target equipment operate?
- Is the target equipment in one or several locations?
- Will the environmental conditions be ordinary?

System Modifications

- What parts of the system are likely to be modified?
- What sorts of modifications are expected?

Security Issues

- Must access to data or the system be controlled?
- Is physical security an issue?

Questions about resource management

- How often will the system be backed up?
- Who will be responsible for the back up?
- Who is responsible for system installation?
- Who will be responsible for system maintenance?

Overview

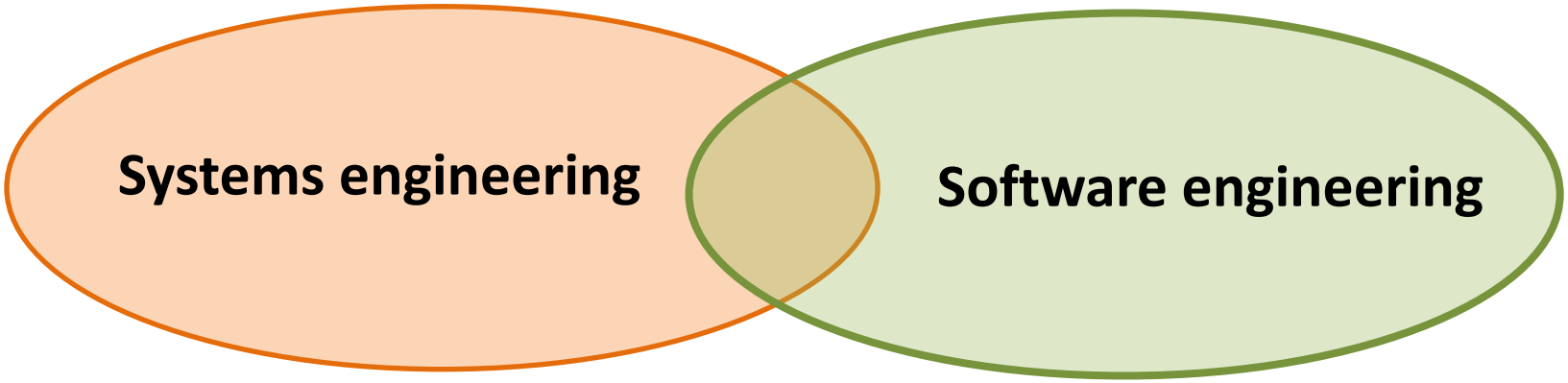
1 Why Software Requirements?

2 Types of Requirements

3 **Requirements Engineering**

4 Requirements Management

System / software engineering



- Interdisciplinary approach to transform a set of customer needs, expectations, and constraints into a solution and to support that solution throughout its life.
- Concerned by designing, developing, deploying, and maintaining systems
- **A system**: an organized set of communicating parts designed for a specific purpose
  - E.g., a mobile phone, a car, an airplane…

- Engineering discipline that concerns all aspects of **software** production from from the early stages of specification, developing, using and maintaining the software.
- A modeling and problem-solving activity, which is knowledge intensive, and rationale driven

[Bruegge, Bernd, and Allen H. Dutoit. "Object--Oriented Software Engineering. Using UML, Patterns, and Java." *Learning* 5, no. 6 (2009): 7.]

Requirements

- "a statement of **what** the system must do, how it must **behave**, the **properties** it must exhibit, the **qualities** it must possess, and the **constraints** that the system and its development must satisfy"

[Institute of Electrical and Electronic Engineers , "IEEE standard glossary of software engineering terminology (IEEE Std 610.12-1990)". Institute of Electrical and Electronics Engineers, New York]]

- "Verbalize **decision** alternatives regarding the **functionality** and **quality** of a system"

[A Aurum and C. Wohlin, "The fundamental nature of requirements engineering activities as a decision making process," Information and Software Technology, vol. 45, no. 14, pp. 945–954, Nov. 2003]

The terms **requirement and feature** are sometimes interchangeably used. However, a **feature** is a set of related **requirements** that allows the user to satisfy a business objective or need.

Requirements engineering

Requirements engineering (RE) is a process of formulating, documenting and maintaining software requirements

[*Kotonya, Gerald; Sommerville, Ian. "Requirements Engineering: Processes and Techniques". John Wiley & Sons. ISBN 978-0471-97208-2.1998*]

It is a common role in systems engineering and software engineering.

Requirements engineering

Requirements engineering (RE) is the branch of systems engineering concerned with:

- The desired **properties** and **constraints** of software-intensive systems
- The **goals** to be achieved in the software's environment
- And **assumptions** about the environment

Views on requirements engineering (RE)

Engineering view

The utilization of **systematic and repeatable techniques** that ensure the completeness, consistency, and

relevance of requirements

[Sommerville and Sawyer 1997]

Lifecycle view

Discovering the purpose of the system being developed, by identifying stakeholders and their needs and documenting these in a form that is amenable to **analysis, communication**, and **subsequent implementation**

[Nuseibeh and Easterbrook 2000]

Knowledge view

The complex task of dealing with, making, and documenting decisions about the functionality and qualities of the system

[Aurum and Wohlin 2003]

First step in identifying the requirements: system identification

- Two questions need to be answered:
  1. How can we identify the **purpose** of a system?
     - What are the requirements, what are the constraints?
  2. What is **inside**, what is **outside** the system?
- These questions are answered during requirements **elicitation** and **analysis**.

Activities of requirements engineering

- **Requirements elicitation:** the process of discovering, reviewing, documenting, and understanding the user's needs and constraints for a system

- **Requirements analysis**: The process of refining the user's needs and constraints

- **Requirements specification:** The process of documenting the user's needs and constraints clearly and precisely

- **Requirements validation:** Ensuring that the system requirements are complete, correct, consistent, and clear

- **Requirements management**: Scheduling, negotiating, coordinating, and documenting the RE activities

Requirements elicitation vs. analysis

- **Requirements elicitation**
  - Definition of the system in terms understood by a **customer or user**
  - Result: "Requirements specification"
- **Analysis**
  - Definition of the system in terms understood by a **developer**
  - Result: "Technical specification", "Analysis model"

Requirements specification

- Requirements are documented in a formal artifact called a Requirements Specification (RS), which will become official only after validation
- IEEE 830-1993 recommends practices for Software Requirements Specification
- Outline of an example template (in IEEE 830-1993):

1. Introduction
    - 1.1 Purpose
    - 1.2 Scope
    - 1.3 Definitions
    - 1.4 Overview
2. Overall descritption
    - 2.1 Product prespective
    - 2.2.Product function
    - 2.3. Users charactristics
    - 2.4. Constraints
    - 2.5. Assumptions and dependencies
3. Specific Requirements
    - 3.1 External interface requirements
        - 3.1.1 User interfaces
        - 3.1.2. Hardware interfaces
        - ...
    - 3.2 Functional requirements
        - 3.2.1. Functional requirements 1.1.
        - …
    - 3.3. Quality requirements
    - 3.4. Design constraints
    - 3.5. Software system attribute
    - 3.6. Other requirements

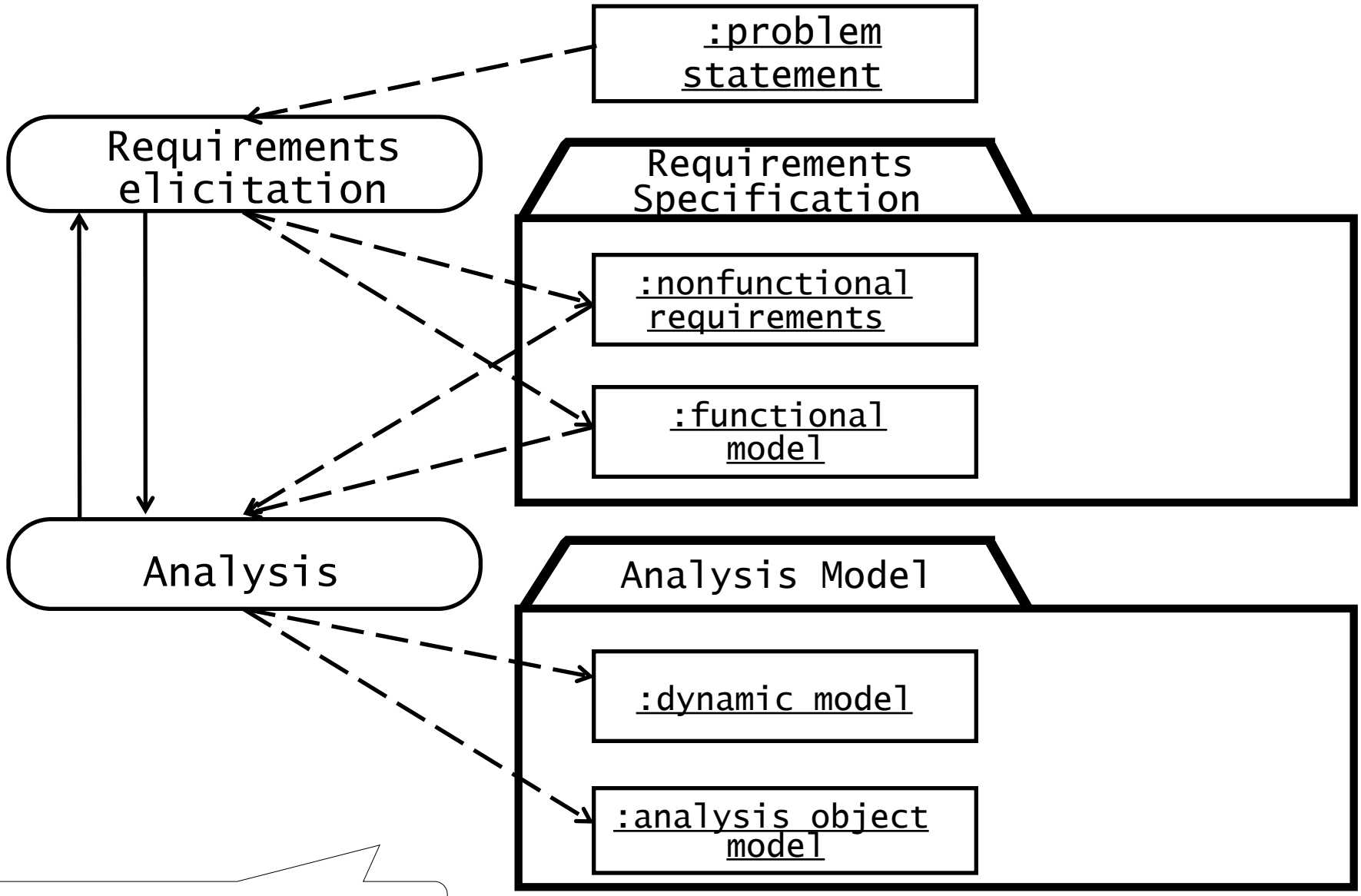
Appendixes

Requirements specification vs analysis model

Both are models focusing on the requirements from the user's view of the system

- The requirements specification uses natural language (derived from the problem statement)
- The analysis model uses a formal or semi-formal notation
- Requirements Modeling Languages
  - Natural Language
  - Graphical Languages: UML, SysML, i*
  - Mathematical Specification Languages: VDM (Vienna Definition Method), Z (based on Zermelo–Fraenkel set theory), Formal methods ….

Key RE activities and artifacts



UML Activity Diagram

Different types of requirements elicitation

- **Greenfield Engineering**
  - Development starts from scratch, no prior system exists
  - Requirements come from end users and clients
  - Triggered by user needs
- **Re-engineering**
  - Re-design and/or re-implementation of an existing system using newer technology
  - Triggered by technology enabler
- **Interface Engineering**
  - Provision of existing services in a new environment
  - Triggered by technology enabler or new market needs

Main techniques to elicit requirements

- **Interview/questionnaire:** A list of pre-selected questions to be answered by the customer/stakeholders
- **Observation:** Observing end users in their operational environment
- **Workshop/brainstorming***:* Gathering ideas from participants (i.e., the customer/stakeholders) spontaneously.
- **Scenario/use case :** Describes the use of the system as a series of interactions between a specific end user and the system (Use case describes a set of scenarios)
- **Mockup: "**cheap" prototype that shows what the software will look like without having to build it

Main requirements elicitation challenges

- Accurate **communication** about the domain and the system is difficult
  - There is a gap between end users and developers
    - End users have **application domain knowledge**
    - Developers have **solution domain knowledge**
- Understanding large and complex system requirements
- Define the system boundary
- Conflicting requirements are there
- Changing customers' needs
- …

A requirements-engineer and his wife…

She: “Honey, we’re out of bread, could you please go to the store and get one. And if they have eggs, bring six.”

He: “Sure, I will.”

He goes and comes back with six breads.

She: “Why on earth did you buy six breads?!?”

He: “They had eggs…”

Validation of requirements

Requirements validation is a quality assurance step, usually performed after requirements elicitation or after analysis to measure quality of requirements.

- **Problems with requirements**
  - Requirements change quickly
  - Inconsistencies are easily added with each change
  - Tool support is needed

Quality measurements

- **Correctness:** The requirements represent the customer's view
- **Completeness:** All possible scenarios, in which the system can be used, are described
- **Consistency:** There are no requirements that contradict each other
- **Clarity:** Requirements can only be interpreted in one way
- **Realism:** Requirements can be implemented and delivered
- **Traceability:** Each system component and behavior can be traced to a set of functional requirements

Principles of requirements validation

- **Principle 1:** Involvement of the correct stakeholders
- **Principle 2:** Separating the identification and the correction of errors
- **Principle 3:** Validation from different views
- **Principle 4:** Adequate change of documentation type
- **Principle 5:** Construction of development artifacts
- **Principle 6:** Repeated validation

Overview

1. Why Software Requirements?
2. Types of Requirements
3. Requirements Engineering
4. **Requirements Management**

Requirements management…

Is about…

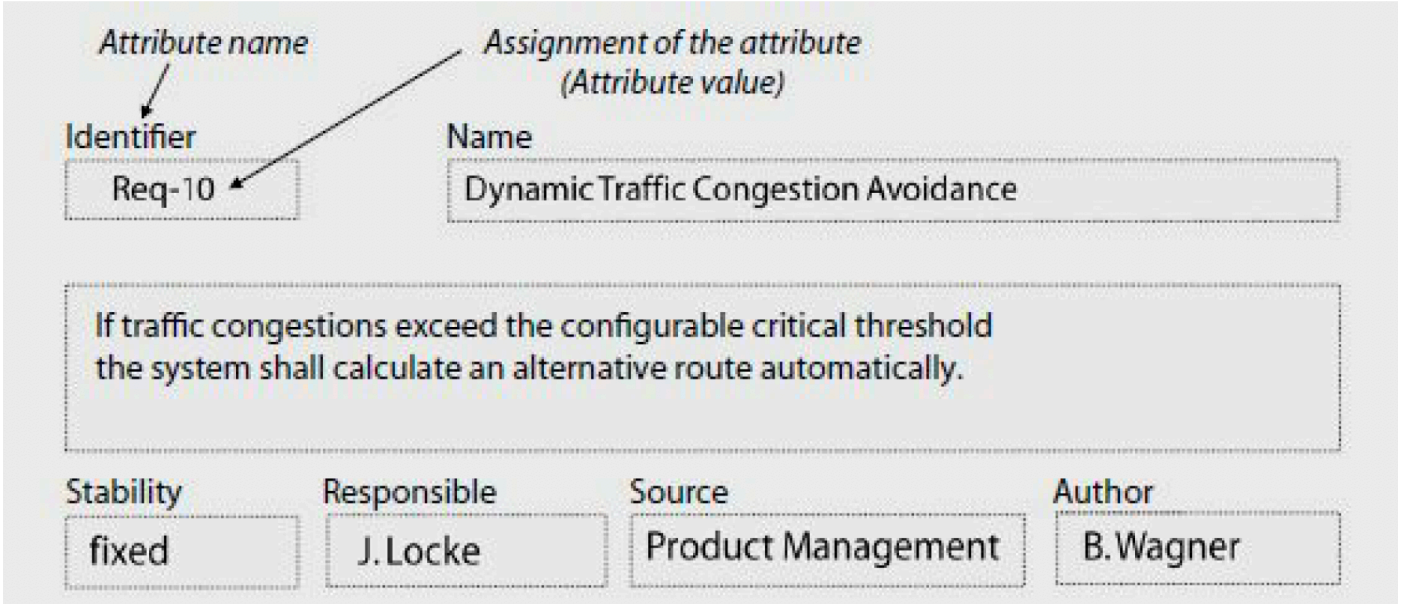
Scheduling, negotiating, coordinating, and documenting the requirements engineering activities.

Requirements for a requirements management system

- Functional requirements:
  - Store the requirements in a shared repository
  - Provide multi-user access to the requirements
  - Organize requirements and create a specification document from the requirements
  - Create dependencies between requirements
  - Manage requirements change and versioning
  - Provide traceability of the requirements throughout the artifacts of the system
  - Set the requirements priorities
  - Assign requirements to releases
  - …

Oragnizing the requirements

- Assignment of requirements identifier
- Identify attributes of requirements, e.g., Requirements name, description, version, author, stability, risk, priority



Requirements Prioritization

Requirements prioritization is the process of managing the relative importance and urgency of different **requirements** to cope with the limited resources of projects.

- **High priority**
  - A high-priority requirement must be demonstrated at the delivery of the system
  - Addressed during analysis, design, and implementation
- **Medium priority**
  - Medium priority requirements are usually demonstrated in future iterations
- **Low priority**
  - Illustrates how the system is going to be used in the future with not yet available technology
  - Low priority requirements have impact only on the analysis model

Aspects of prioritization

Importance

Cost

Risk

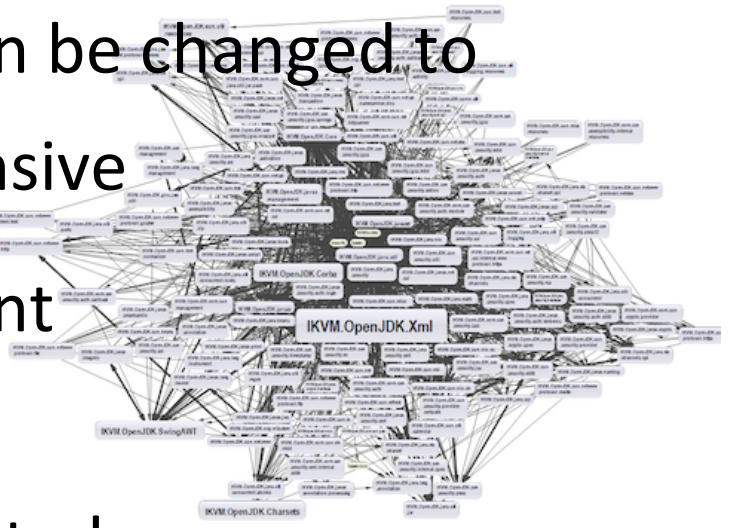
Time

Financial benefit

Penalty

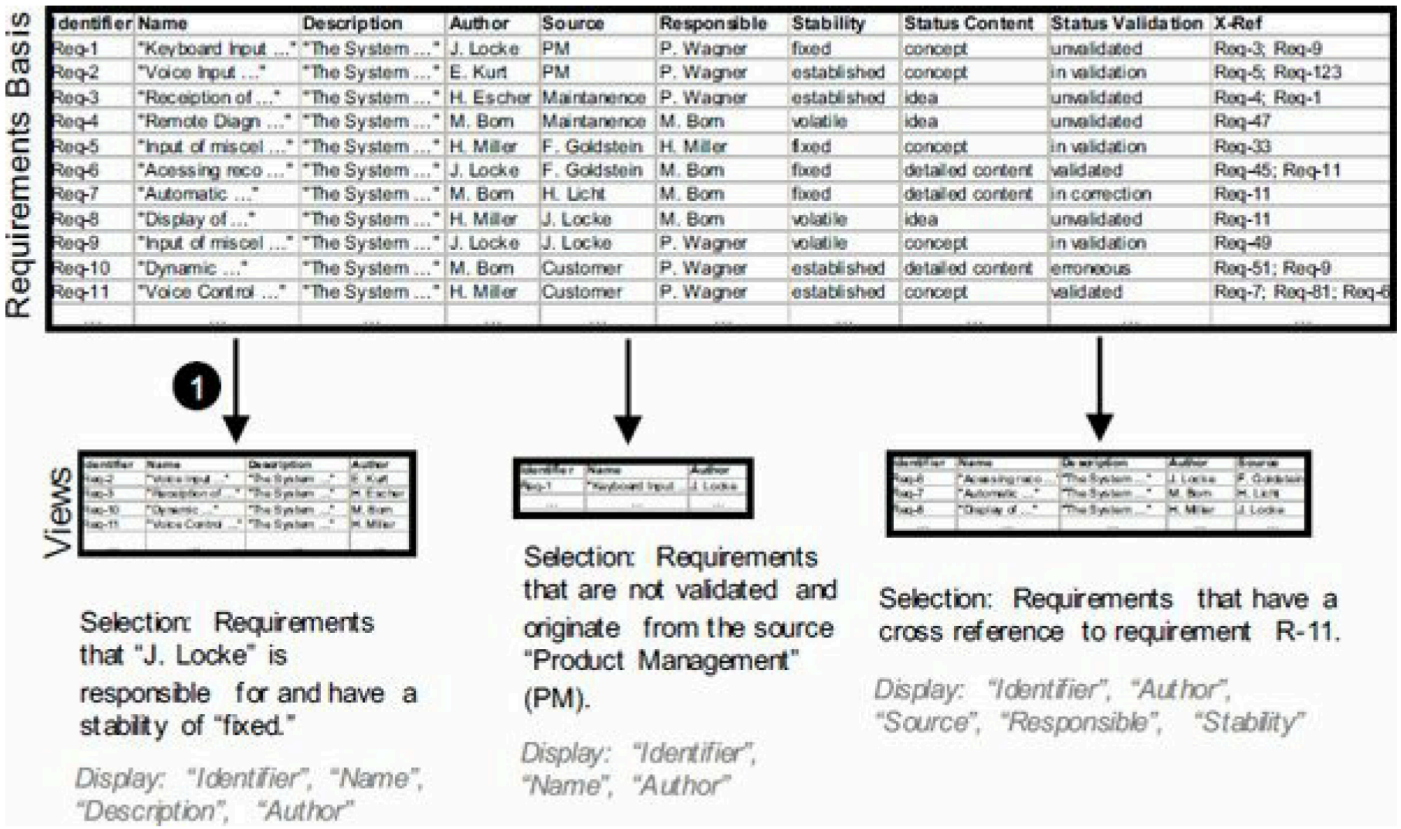
Dependencies of Prioritization aspects

- Change in one aspect could result in another aspect
  - A high priority requirement can be changed to a low priority one, if it is expensive
- It is important to consider different aspects when deciding if a requirement should be implemented
  - Tradeoff Analysis



Views on requirements

- Views on requirements are often defined for different roles in the development process.
- E.g., Views on ”responsible”, “Cross reference”, “Validated”

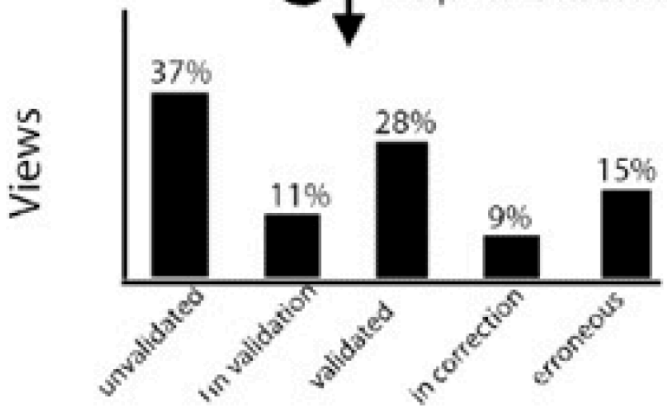


Assign requirements to releases

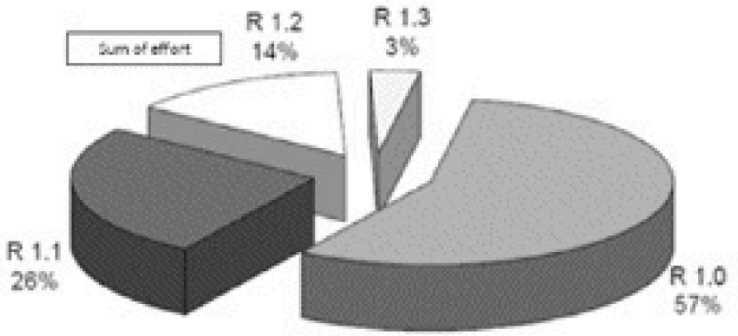
Requirements Basis

Identifier	Name	Description	Author	Source	Responsible	Stability	Status Content	Status Validation	X-Ref
Req-1	Keyboard Input ...	The System ...	J. Locke	PM	P. Wagner	fixed	concept	unvalidated	Req-3; Req-9
Req-2	Voice Input ...	The System ...	E. Kurt	PM	P. Wagner	established	concept	in validation	Req-5; Req-123
Req-3	Receiption of ...	The System ...	H. Escher	Maintanence	P. Wagner	established	idea	unvalidated	Req-4; Req-1
Req-4	Remote Diagn ...	The System ...	M. Born	Maintanence	M. Born	volatile	idea	unvalidated	Req-47
Req-5	Input of miscel ...	The System ...	H. Miller	F. Goldstein	H. Miller	fixed	concept	in validation	Req-33
Req-6	Acessing reco ...	The System ...	J. Locke	F. Goldstein	M. Born	fixed	detailed content	validated	Req-45; Req-11
Req-7	Automatic ...	The System ...	M. Born	H. Licht	M. Born	fixed	detailed content	in correction	Req-11
Req-8	Display of ...	The System ...	H. Miller	J. Locke	M. Born	volatile	idea	unvalidated	Req-11
Req-9	Input of miscel ...	The System ...	J. Locke	J. Locke	P. Wagner	volatile	concept	in validation	Req-49
Req-10	Dynamic ...	The System ...	M. Born	Customer	P. Wagner	established	detailed content	erroneous	Req-51; Req-9
Req-11	Voice Control ...	The System ...	H. Miller	Customer	P. Wagner	established	concept	validated	Req-7; Req-81; Req-6
...	...	...	...	...	...	...	...	...	...

1 Validation status of the Requirements Basis



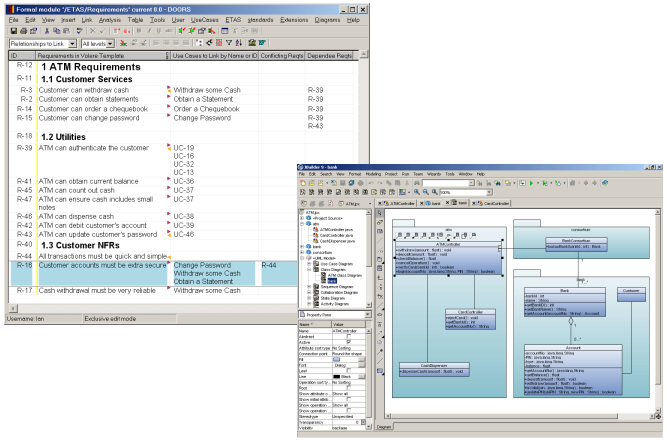
2 Implementation effort by Release (R)



Rationale management

- **Rationale** is justification behind decisions
- It is captured and used in many different forms in Software Engineering
- Benefits
  - Increases the stakeholders understanding of the system, making it easier to adapt or maintain
  - Ability to explain past decisions

Tool supporting



IBM RequisitePro

IBM Rational DOORS Next Generation

MODERN Requirements

ReQtest

Jama software

osseno

Accompa

MICRO FOCUS

pearls The Solution is Simple

HelixALM

visure

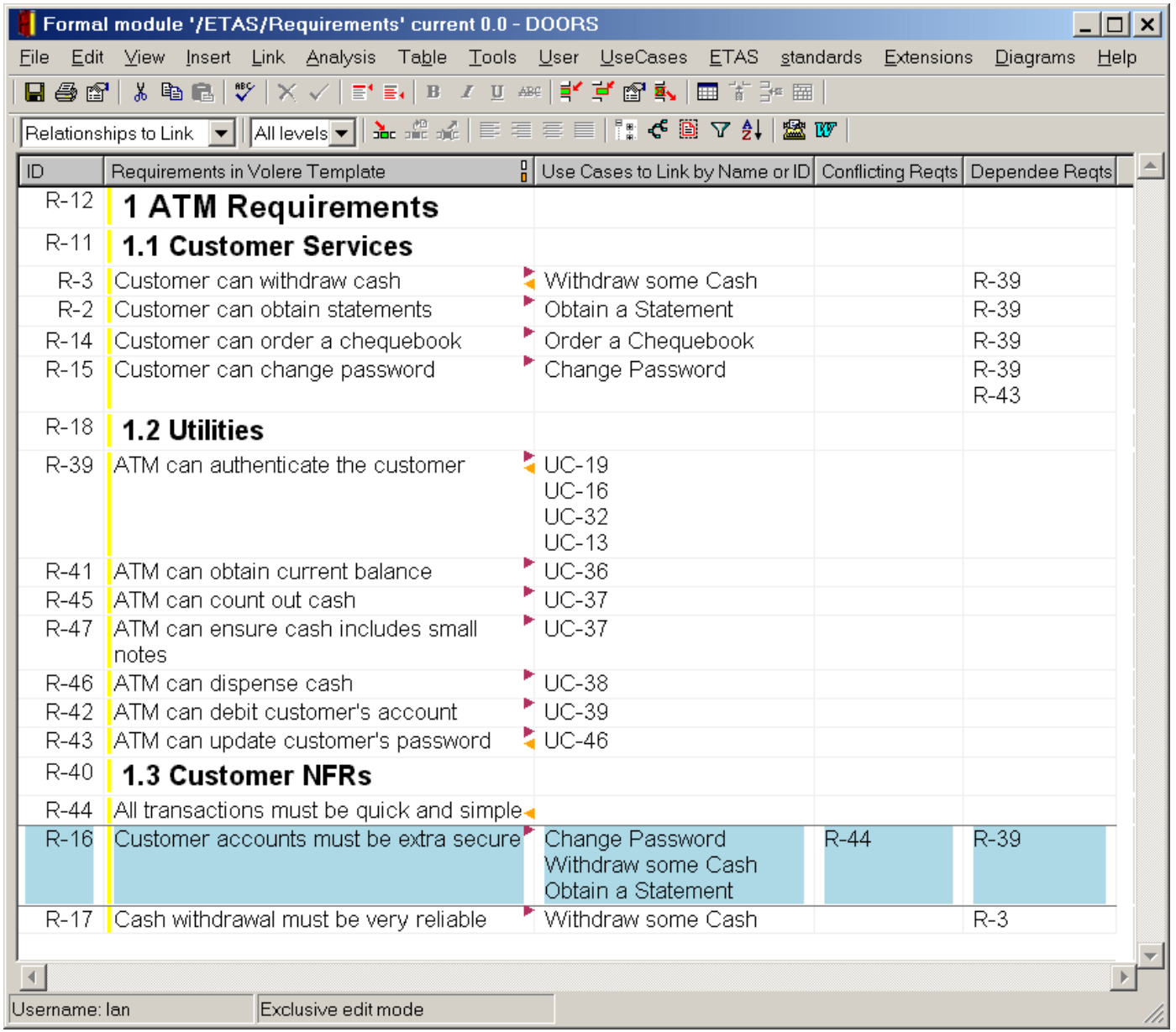
Bests in 2021: https://www.guru99.com/requirement-management-tools.html

More requirements engineering tools

- Doors
- Rational Rose
- Github
- Jira
- IBM Doors
- Polarion Requirements
- Wiki
- Gitlab

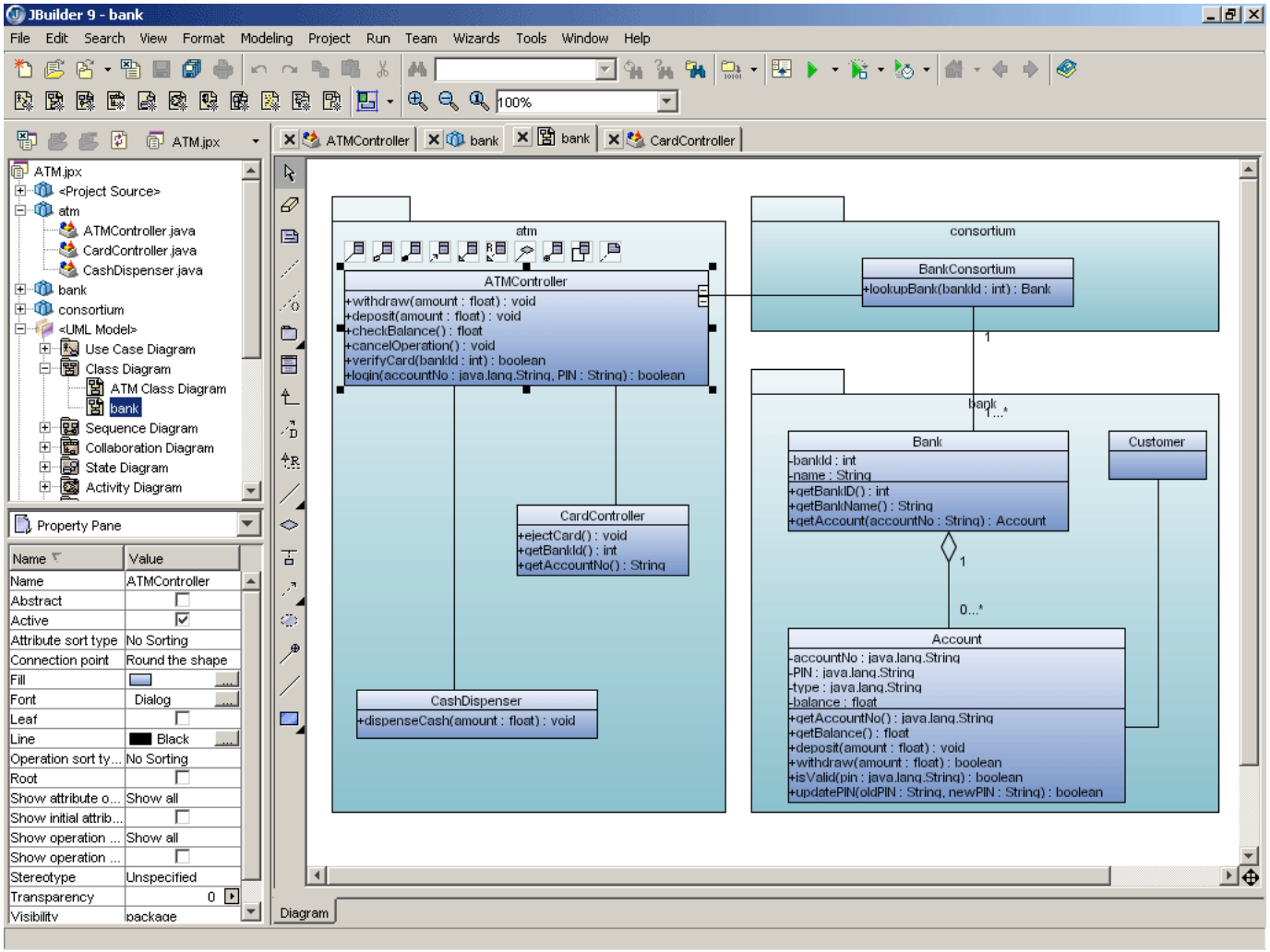
More RE tools:
https://en.wikipedia.org/wiki/List_of_requirements_engineering_tools

Example: Doors



ID	Requirements in Volere Template	Use Cases to Link by Name or ID	Conflicting Reqts	Dependee Reqts
R-12	**1 ATM Requirements**			
R-11	**1.1 Customer Services**			
R-3	Customer can withdraw cash	Withdraw some Cash		R-39
R-2	Customer can obtain statements	Obtain a Statement		R-39
R-14	Customer can order a chequebook	Order a Chequebook		R-39
R-15	Customer can change password	Change Password		R-39 R-43
R-18	**1.2 Utilities**			
R-39	ATM can authenticate the customer	UC-19 UC-16 UC-32 UC-13		
R-41	ATM can obtain current balance	UC-36		
R-45	ATM can count out cash	UC-37		
R-47	ATM can ensure cash includes small notes	UC-37		
R-46	ATM can dispense cash	UC-38		
R-42	ATM can debit customer's account	UC-39		
R-43	ATM can update customer's password	UC-46		
R-40	**1.3 Customer NFRs**			
R-44	All transactions must be quick and simple			
R-16	Customer accounts must be extra secure	Change Password Withdraw some Cash Obtain a Statement	R-44	R-39
R-17	Cash withdrawal must be very reliable	Withdraw some Cash		R-3

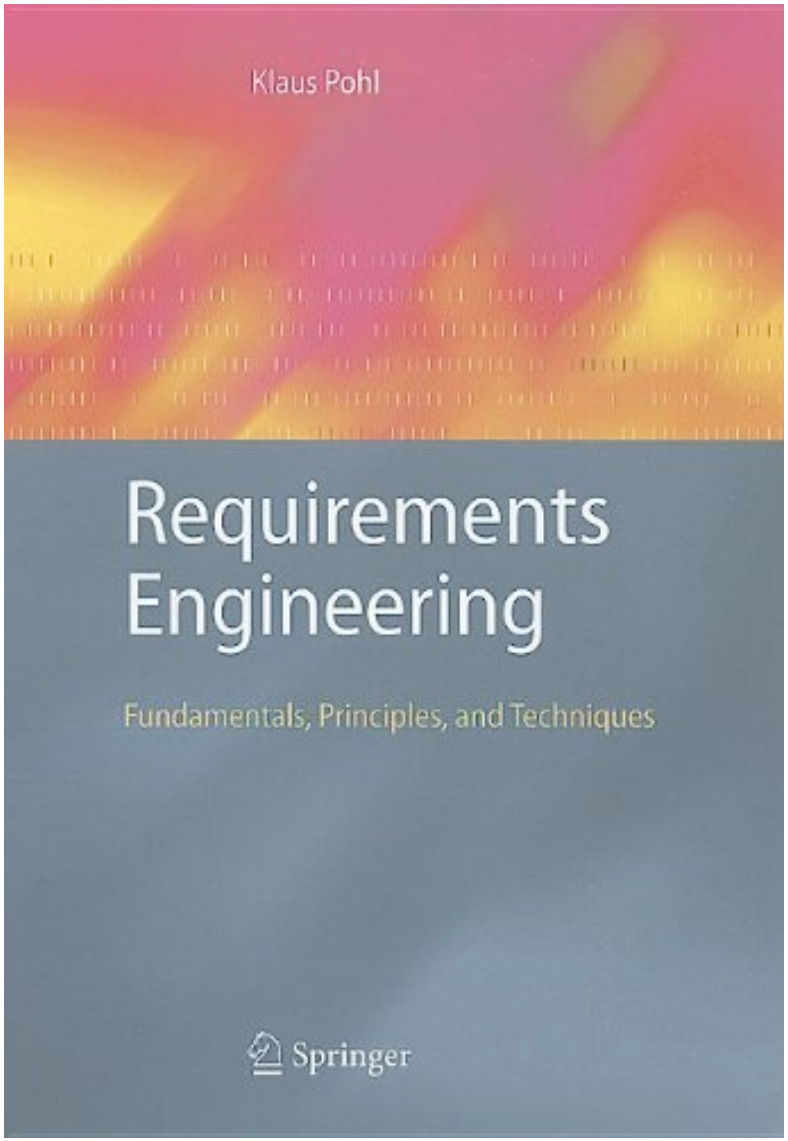
Example: Rational Rose

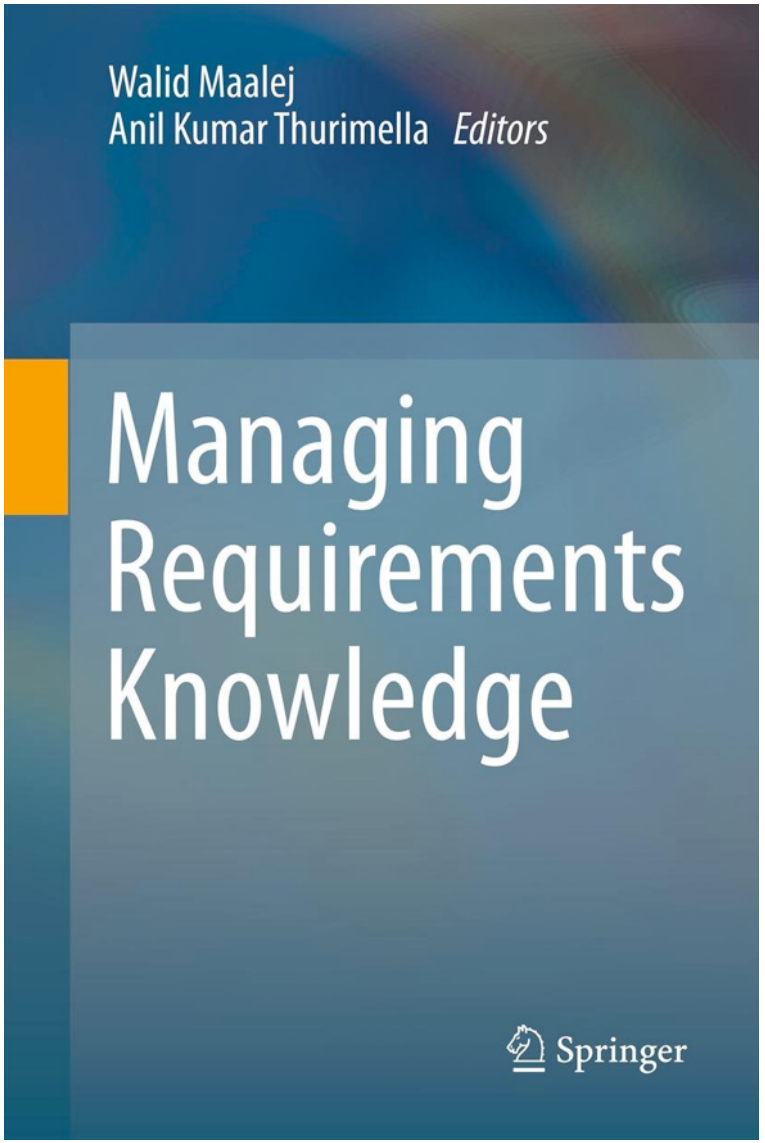


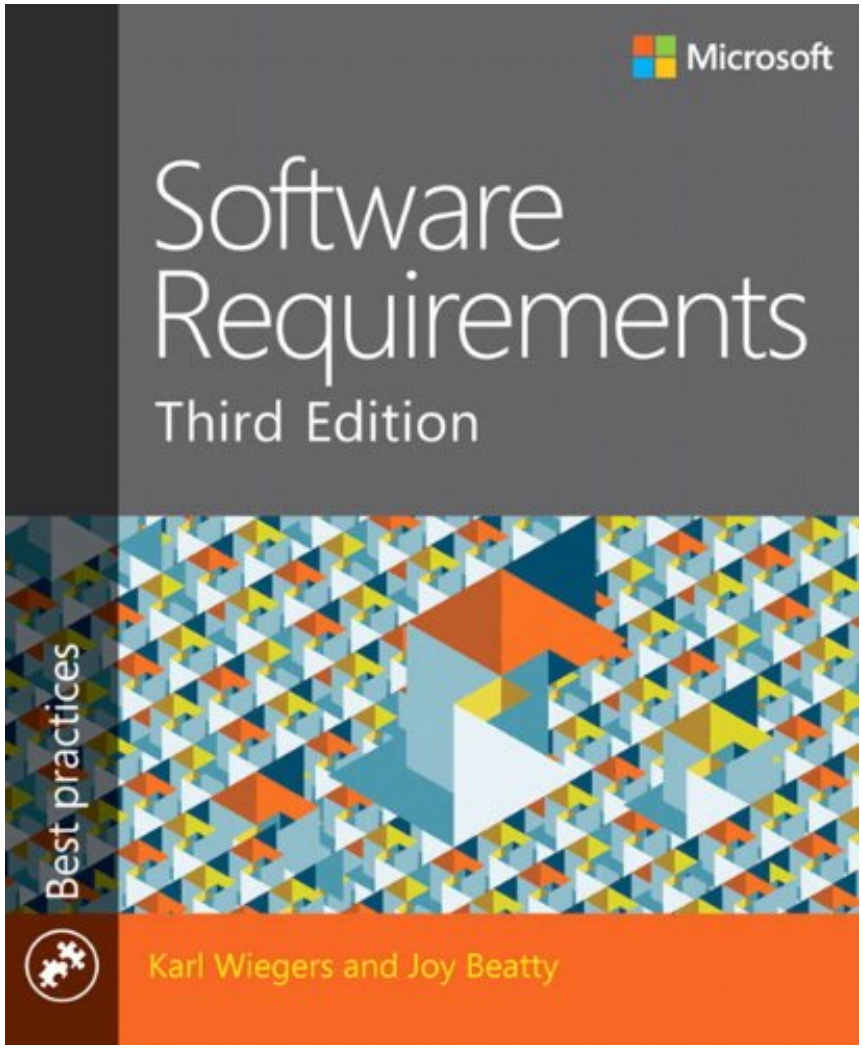
Examples for Web-based requirements managements

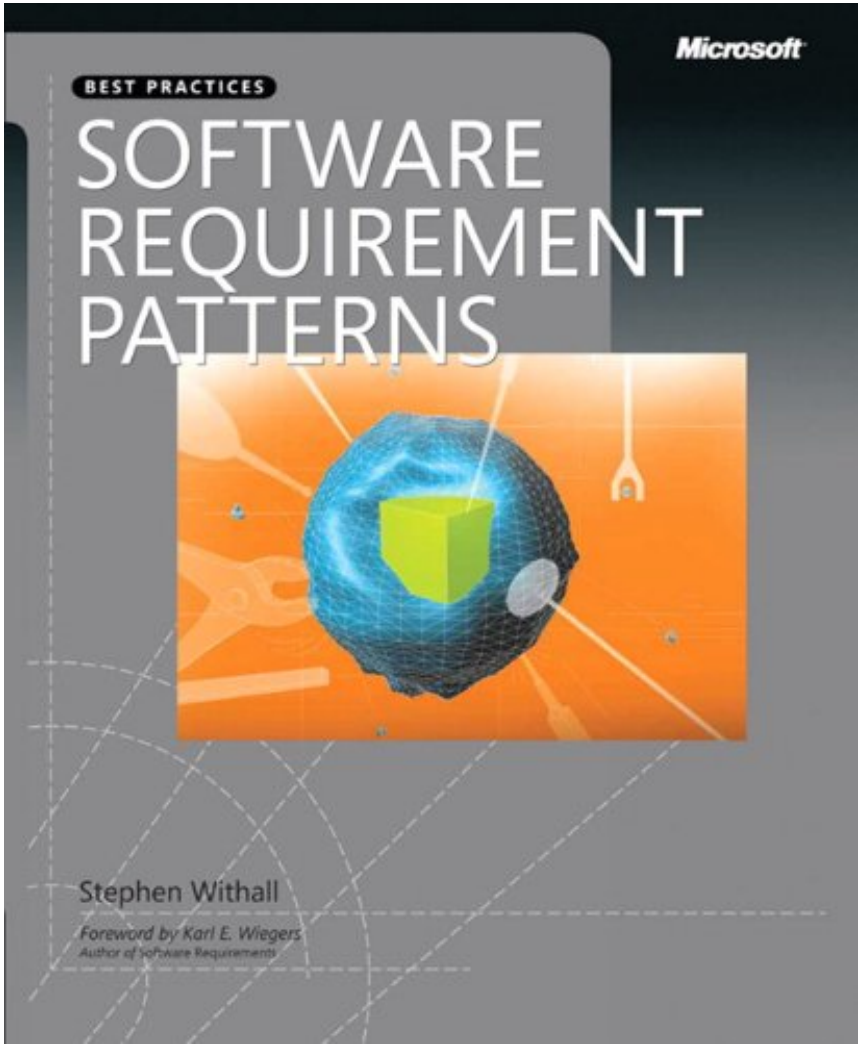
More examples:
https://www.capterra.com/requirements-management-software/s/web-based/

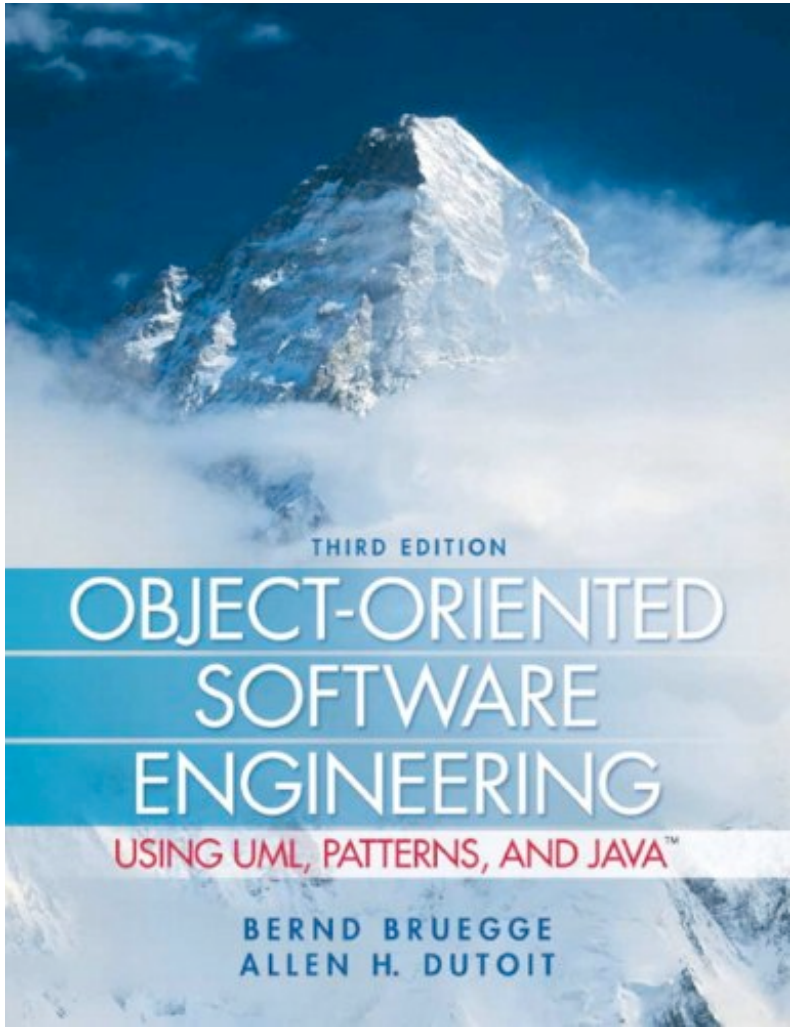
Course literature

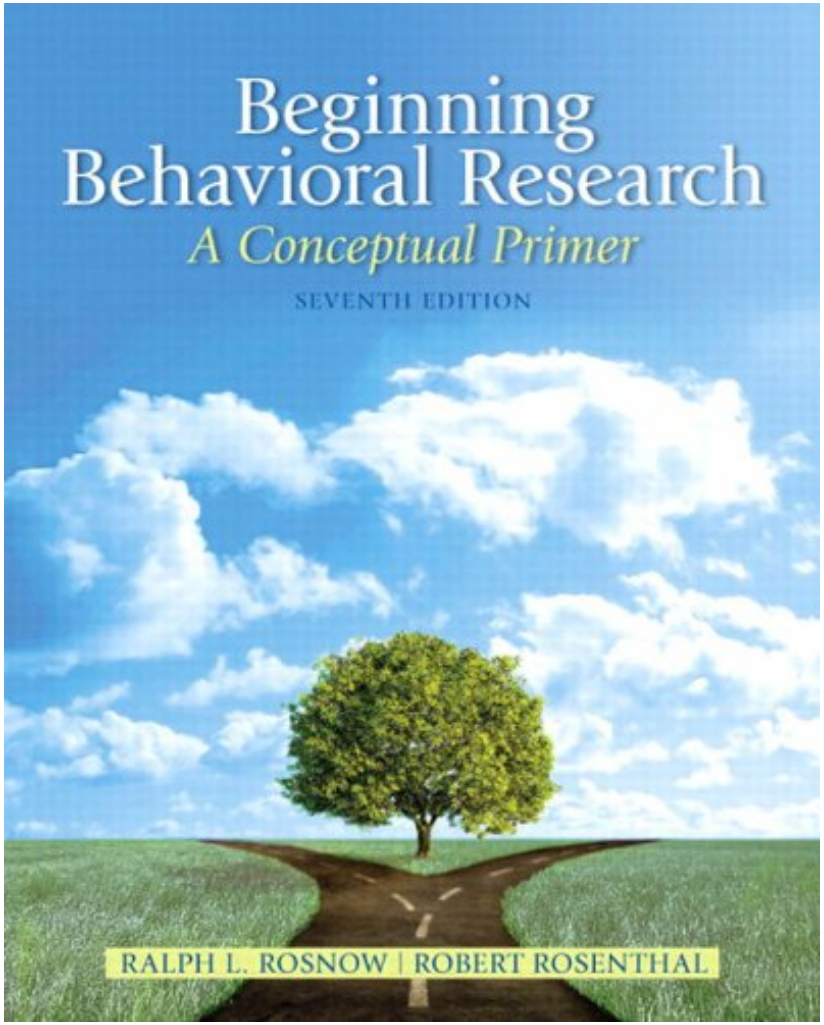

Klaus Pohl
Requirements Engineering
Fundamentals, Principles, and Techniques
Springer


Walid Maalej
Anil Kumar Thurimella Editors
Managing Requirements Knowledge
Springer


Microsoft
Software Requirements
Third Edition
Best practices
Karl Wiegers and Joy Beatty


Microsoft
BEST PRACTICES
SOFTWARE REQUIREMENT PATTERNS
Stephen Withall
Foreword by Karl E. Wiegers
Author of Software Requirements


THIRD EDITION
OBJECT-ORIENTED SOFTWARE ENGINEERING
USING UML, PATTERNS, AND JAVA™
BERND BRUEGGE
ALLEN H. DUTOIT


Beginning Behavioral Research
A Conceptual Primer
SEVENTH EDITION
RALPH L. ROSNOW | ROBERT ROSENTHAL

Summary

1. Requirements engineering is difficult because customers often don’t know what they want and their needs change

2. Requirements can be functional, nonfunctional, legal, or even technical (constraints)

3. Requirements engineering is the process of formulating, documenting, analyzing, and maintaining requirements

4. This course focus on combining engineering techniques with empirical methods to handle software requirements