

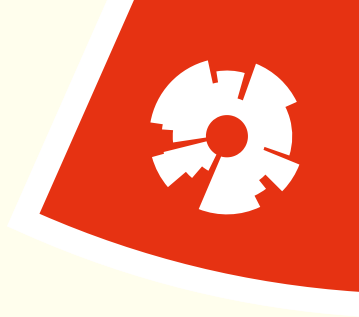
T-303-HUGB: Software
Engineering

M01: Requirements Engineering

Grischa Liebel



This Lecture



- Activities in RE:
Elicitation, specification, validation, management
- Stakeholders vs. Users
- Elicitation: Recap from T-216-GHOH
- Specification: Specification styles, Non-functional requirements
- Validation techniques
- Elements of Requirements Management

Literature

- [Sommerville] Ch. 4



Learning Outcomes



Contrast software engineering techniques required for different types of software systems.

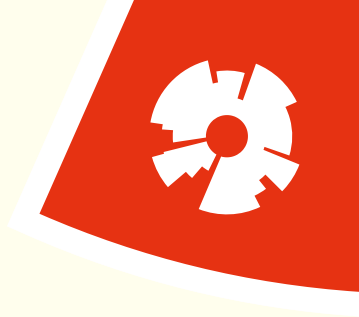
Illustrate the term stakeholder in relation to different types of software systems.

Classify different kinds of requirements needed in software engineering.

Formulate functional and quality requirements using different techniques.

Summarise different techniques for performing requirements validation.

Why the Vasa Sank



2. Changing needs

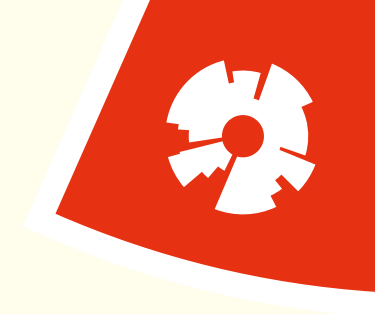
3. Lack of technical specifications

4. Lack of a documented project plan

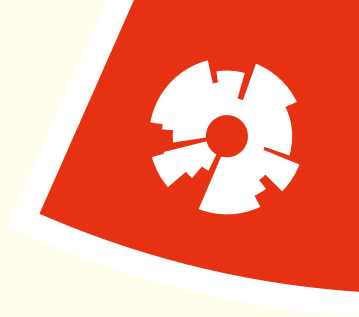
5. & 6. Excessive and secondary innovations

7. Requirements creep

Software Project Failure Factors	Percentage of Projects (%)		
	In-house	Outsourced	Overall
Delivery date impacted the development process	93.9	90.5	92.9
Project under-estimated	83.7	76.2	81.4
Risks were not re-assessed, controlled, or managed through the project	73.4	80.9	75.7
Staff were not rewarded for working long hours	81.6	57.1	74.3
Delivery decision made without adequate requirements information	83.7	47.6	72.9
Staff had an unpleasant experience working on the project	83.7	47.6	72.9
Customers/Users not involved in making schedule estimates	69.4	76.2	71.4
Risk not incorporated into the project plan	65.3	80.9	70.0
Change control not monitored, nor dealt with effectively	63.3	85.7	70.0
Customer/Users had unrealistic expectations	69.4	66.7	68.6
Process did not have reviews at the end of each phase	75.5	47.6	67.1
Development Methodology was inappropriate for the project	71.4	52.4	65.7
Aggressive schedule affected team motivation	69.4	57.1	65.7
Scope changed during the project	67.3	57.1	64.3
Schedule had a negative effect on team member's life	71.4	42.9	62.9
Project had inadequate staff to meet the schedule	63.3	57.1	61.4
Staff added late to meet an aggressive schedule	61.2	61.9	61.4
Customers/Users did not make adequate time available for requirements gathering	61.2	57.1	60.0



Cerpa & Verner, 2009. "Why did your project fail?", CACM 52 (12)



Here are 10 signs of IS project failure:³

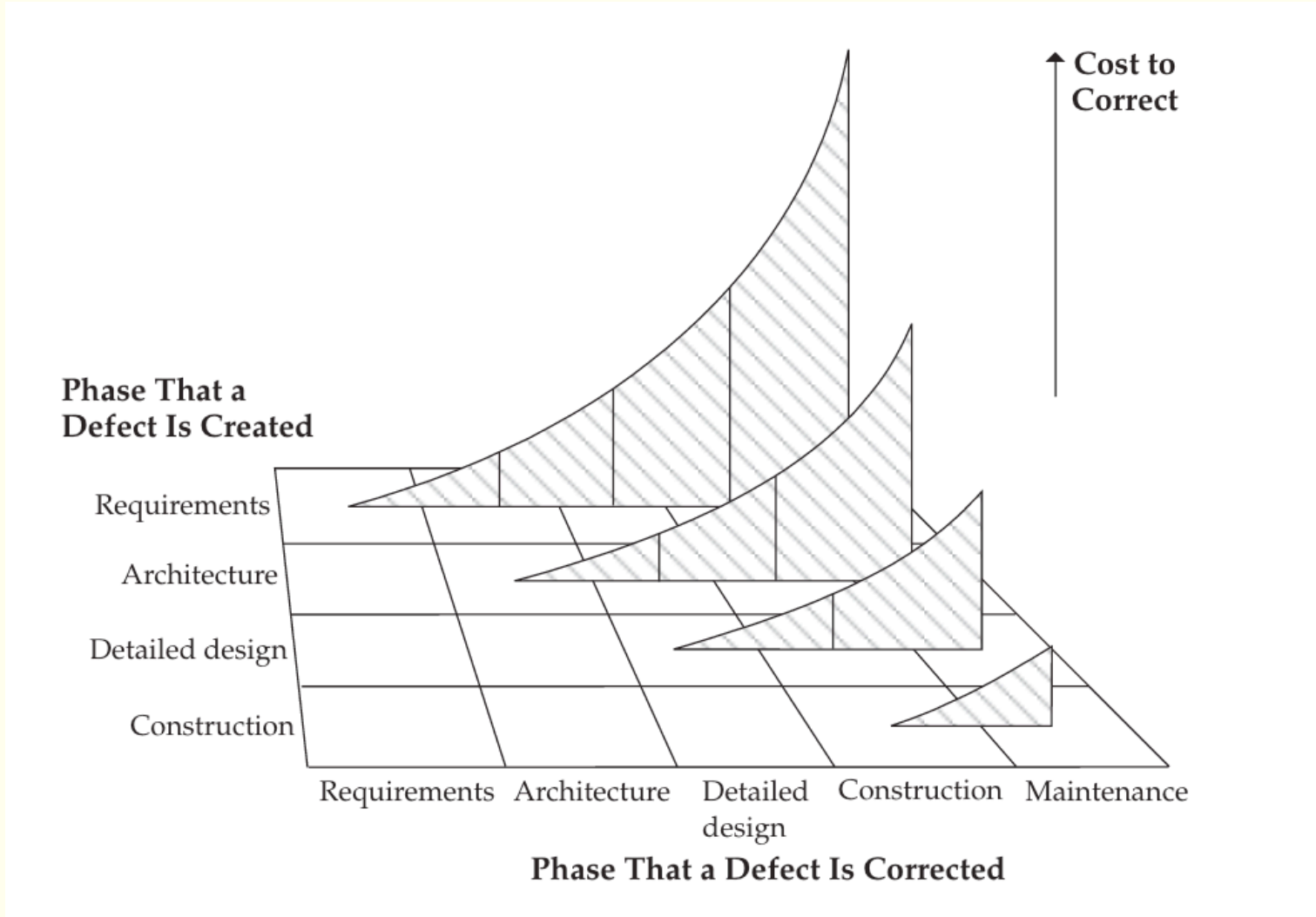
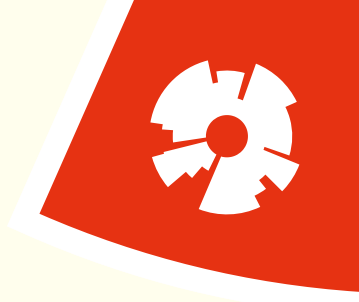
1. Project managers don't understand users' needs.
2. The project's scope is ill-defined.
3. Project changes are managed poorly.
4. The chosen technology changes.
5. Business needs change.
6. Deadlines are unrealistic.
7. Users are resistant.
8. Sponsorship is lost.
9. The project lacks people with appropriate skills.
10. Managers ignore best practices and lessons learned.

Reel, 2002. "Critical success factors in software projects", IEEE SW 16 (3)



"Programs have now got very large and very critical [..]. There have been many problems and failures, but these have nearly always been attributable to inadequate analysis of requirements or inadequate management control."

- Sir Tony Hoare, 1995

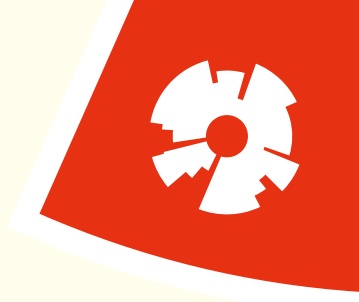


McConnel, Software Project Survival Guide, 1998

Four activities

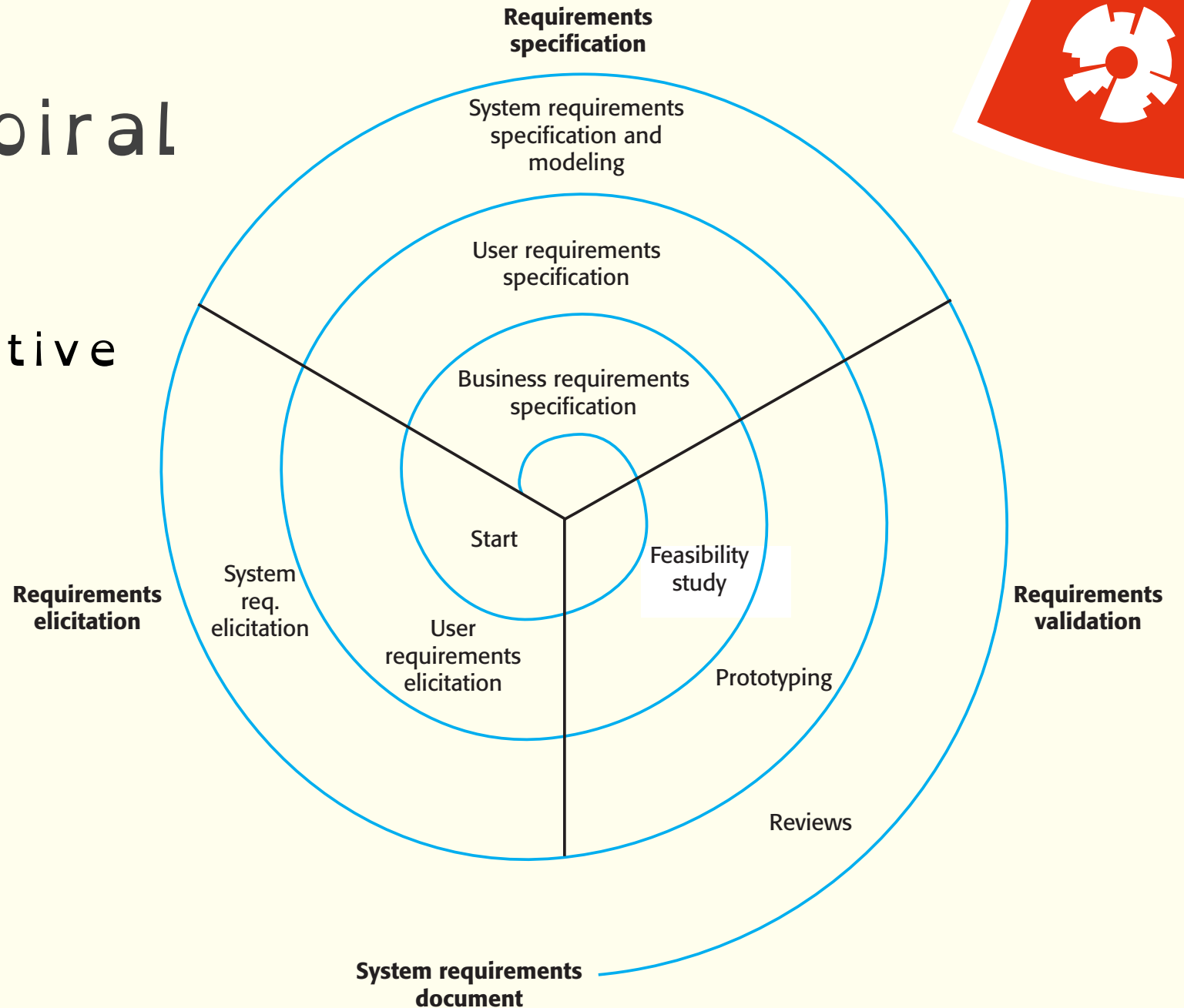
- Elicitation (Discovering Requirements)
- Specification (Writing them down)
- Validation (Checking whether they are appropriate)
- Management (Keeping track/evolving them)





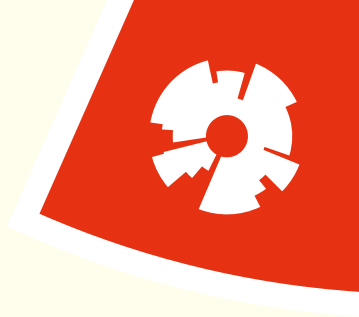
RE Process Spiral

• Main point: It's iterative



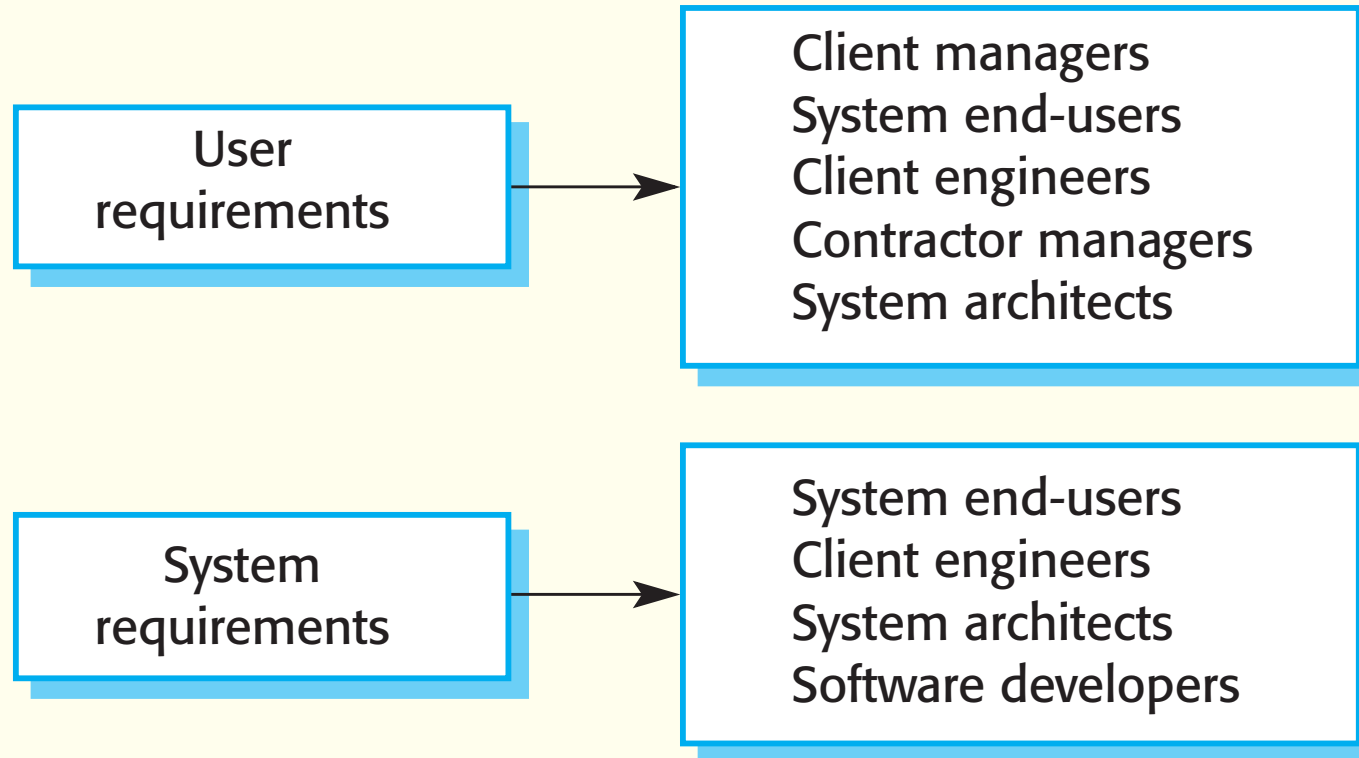
[Sommerville]

System vs. User Requirements



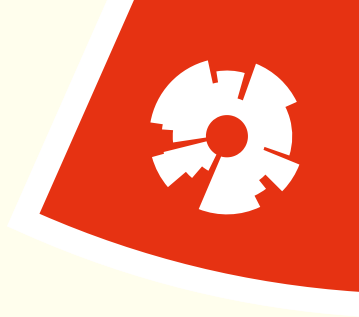
- Requirement: high-level abstract statement to detailed mathematical functional specification
 - Basis for a bid for a contract
 - Basis for an actual contract
- Even if market-driven software
 - High-level requirements (to discuss with stakeholders)
 - Detailed technical requirements (towards implementation)
- In practice: Often accidental ("right" abstraction is hard)

Who needs what?



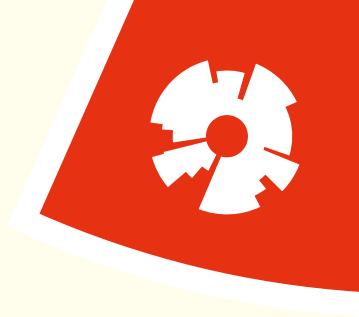
[Sommerville]

Stakeholders



- Any person or organization who is **affected** by the system in some way
- Not just the end user(s), but a large scope
 - End users
 - System managers
 - System owners
 - External stakeholders (regulators, authorities)

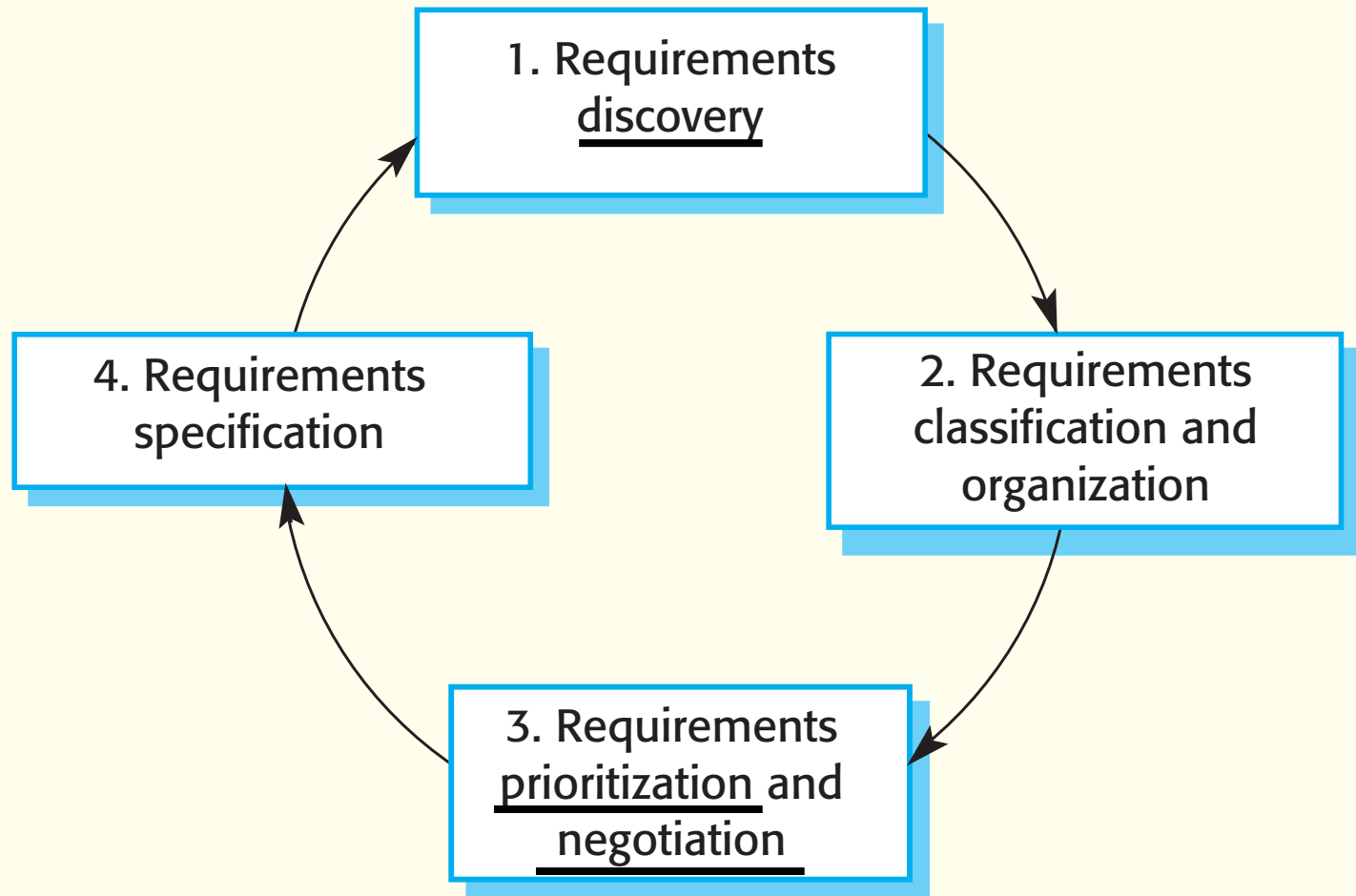
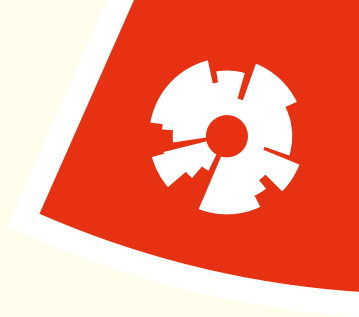
Functional and non-functional requirements



- Functional requirements
 - What should the system do? (Not how!)
- Non-functional requirements ("quality requirements")
 - Constraints/"qualities" of the system (e.g., performance, maintainability)
 - Difficult to break down to sub-systems (they affect the entire system)
 - Difficult to decide limits

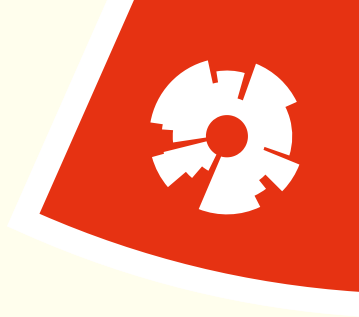


Elicitation



[Sommerville]

Techniques



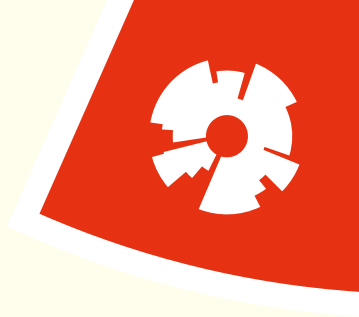
- To a large extent covered in T-216-GHOH
- Interviews - not just planlessly, but different strategies (structured, semi-structured)
- Surveys
- Observation
- Studying existing systems, documentation, processes
- Important: Not just asking stakeholders what they want

Issues

- Stakeholders don't know what they really need
- Domain-specific language
- Conflicting requirements
- Politics and organisational factors
- Requirements and stakeholders change
(continuously and unplanned: Scope creep)



Scenarios



- Real-life examples of the system in use
- Rationales often implicitly included (context)
- Often easier to relate to than single requirements
- But: Hard to tell what is required and what is extra
- Sommerville describes scenarios as a tool for **elicitation**
 - Also a tool for specification: Describing requirements as scenarios.

Scenarios

Scenarios should include

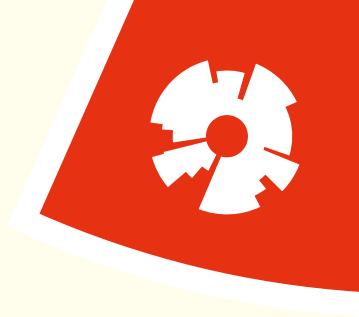
- Starting situation
- Normal event flow
- What can go wrong - exception flows
- Other concurrent activities
- State when the scenario finishes





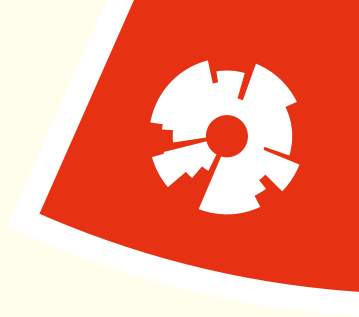
Specification

Functional Requirements Specification



- "Writing down" requirements in a document
 - Or adding them to a system (JIRA, Doors, ...)
- User requirements: understandable by end-users and customers without technical background
- System requirements: more detailed, may include more technical information

Functional Requirements Specification



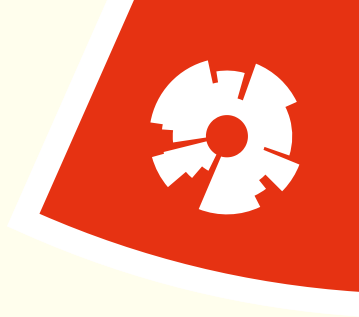
- Overall: What's the purpose of your specification?
 - Driving development
 - Contracting
 - Explaining to stakeholders
 - Documentation



Natural language specification

- One requirement = One natural language sentence
 - Often supplemented (id, rationale, priority, traces, dependencies)
- Expressive, intuitive and universal
- Highly ambiguous

Natural language specification



- Invent a standard format
- Use consistent vocabulary
 - "shall" for mandatory requirements
 - "should" for desirable requirements
- Avoid jargon
- Include a rationale
 - Often helps to avoid misunderstandings!
 - Can help to remember why requirements were added



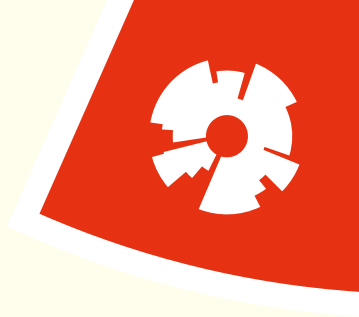
Examples

3.2 The system shall measure the blood sugar and deliver insulin, if required, every 10 minutes. (*Changes in blood sugar are relatively slow so more frequent measurement is unnecessary; less frequent measurement could lead to unnecessarily high sugar levels.*)

3.6 The system shall run a self-test routine every minute with the conditions to be tested and the associated actions defined in Table 1. (*A self-test routine can discover hardware and software problems and alert the user to the fact the normal operation may be impossible.*)

[Sommerville]

Structured specifications



- Pre-defined structure (=templates)
- Freedom of writer is limited
 - Might not work for everything
 - Ensures consistency
 - Allows/facilitates automation (e.g., generating tests)
- Common in critical systems!

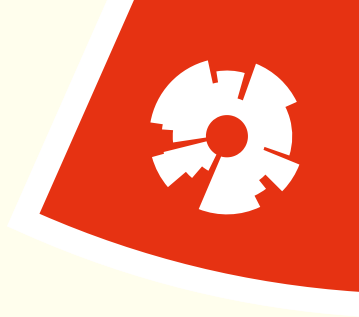
User Stories

As a profile owner on Instagram, I want to be able to add tags to my pictures, so that they can be found easier.

As a driver, I want the car to react to voice commands, so that I am not distracted by searching the menu.

- Very common/popular in Agile processes
- Avoid common problems with requirements
 - Who is the actor?
 - What is the rationale (here: the benefit)?
- Most common format:
"As a [Persona], I want to [action], so that [benefit]"

Form-based specifications



- Natural language + mandatory/optional fields
 - Definition
 - In/Outputs
 - ID
 - Pre/Post conditions
 - Priority
 - ...

Structured specification

[Sommerville]



— Insulin Pump/Control Software/SRS/3.3.2

Function Compute insulin dose: safe sugar level.

Description Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.

Inputs Current sugar reading (r_2); the previous two readings (r_0 and r_1).

Source Current sugar reading from sensor. Other readings from memory.

Outputs CompDose—the dose in insulin to be delivered.

Destination Main control loop.



Additional fields

[Sommerville]

— Insulin Pump/Control Software/SRS/3.3.2

Function Compute insulin dose: safe sugar level.

Description Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.

Inputs Current sugar reading (r2); the previous two readings (r0 and r1).

Source Current sugar reading from sensor. Other readings from memory.

Outputs CompDose—the dose in insulin to be delivered.

Destination Main control loop.

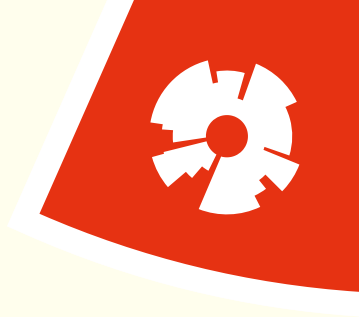
RATIONALE: In order to attract users to new products, program releases, discounts, etc.

DEPENDENCIES: Req A, Req B

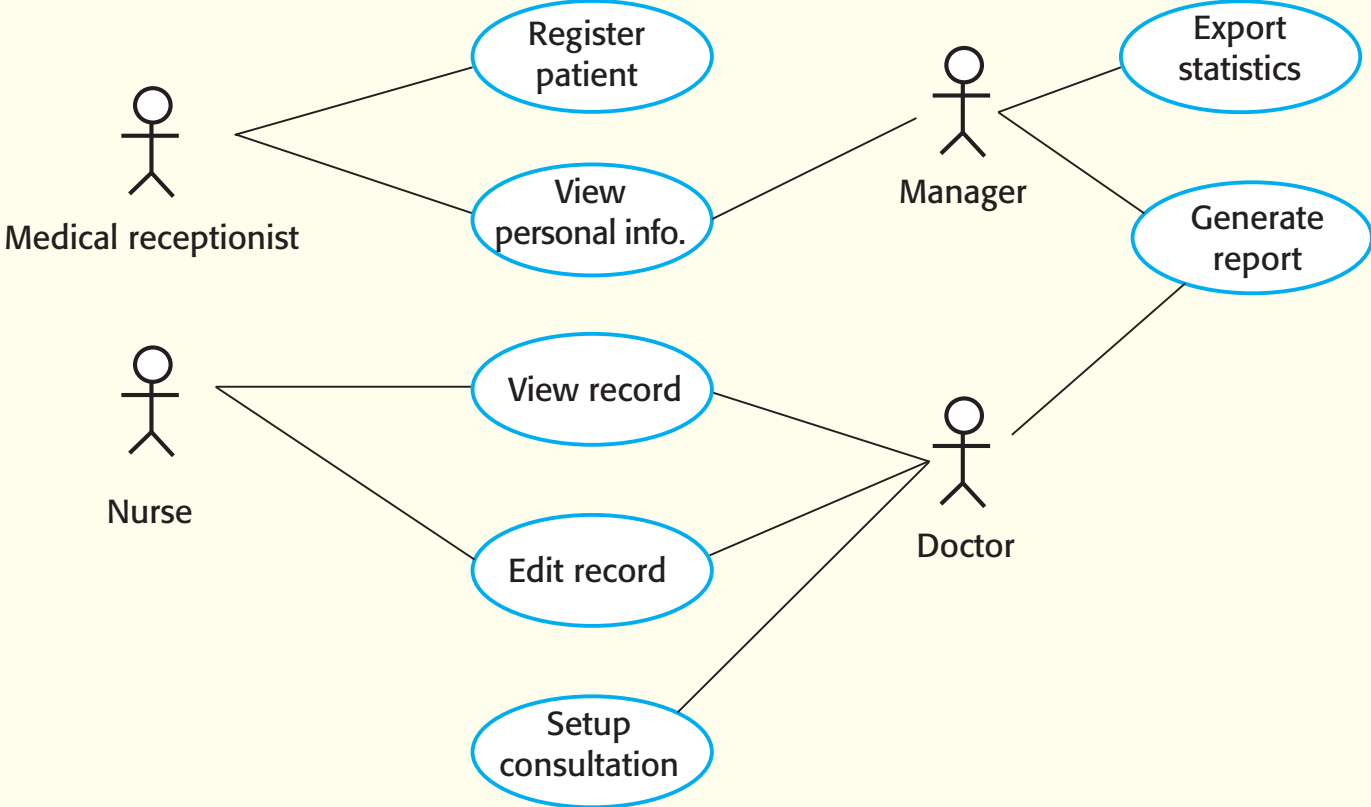
Use cases

- A structured kind of scenario
- Central parts: The actors involved, and the interaction between them
- UML has a high-level model as an overview
- Use cases are useful for automated generation of other artefacts

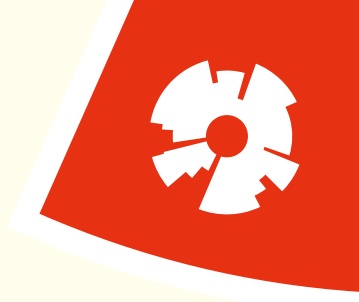




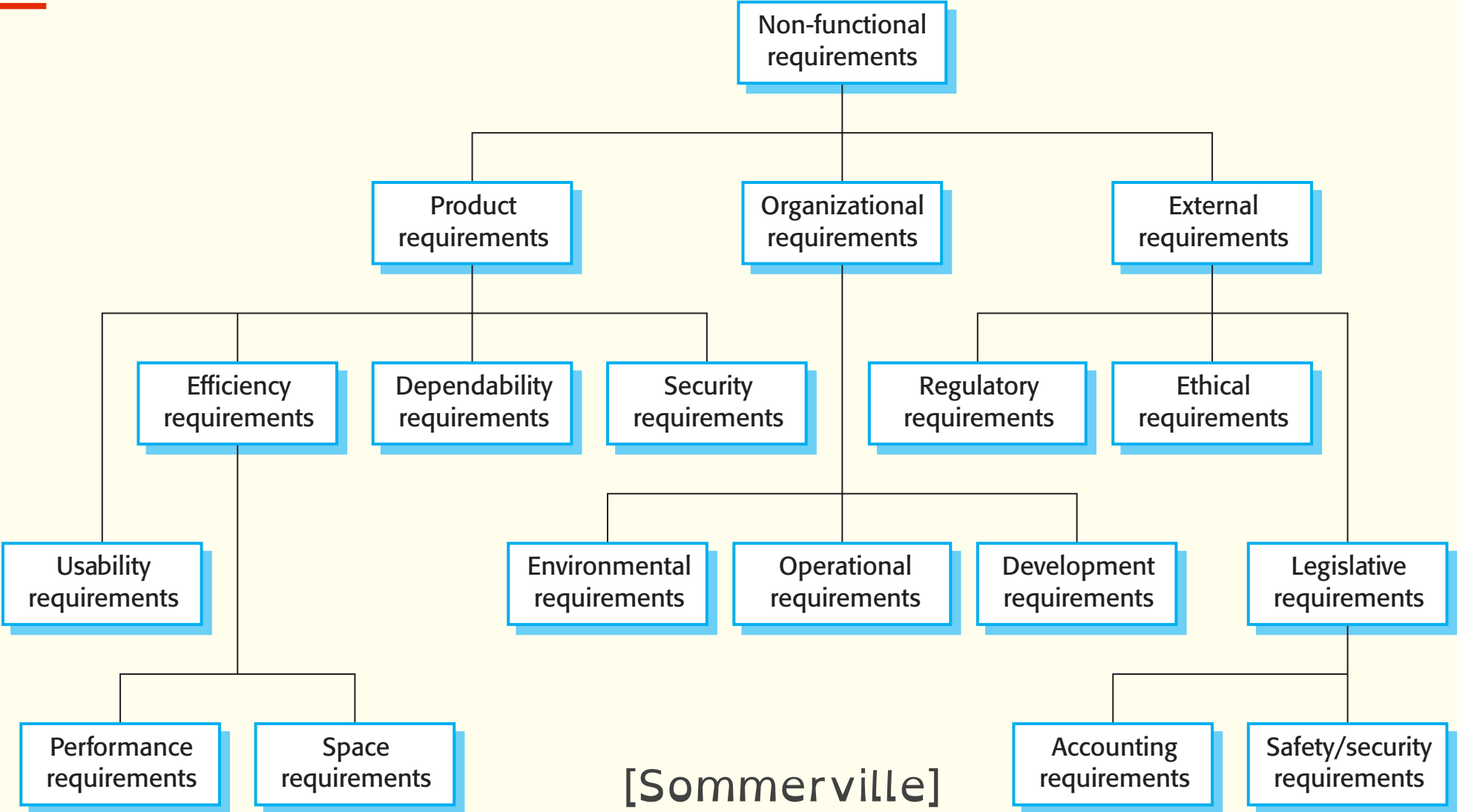
Use cases (Mentcare)



[Sommerville]



Non-functional/quality requirements



[Sommerville]

Metrics

Property	Measure
Speed	Processed transactions/second User/event response time Screen refresh time
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

[Sommerville]

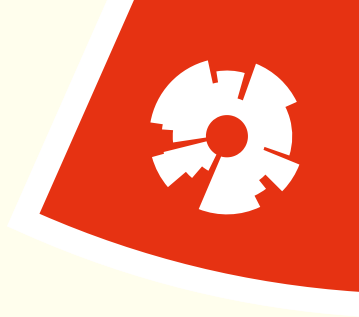
T-216-GHOH

- A bad requirement:
 - "The system should be really fast"
- Better:
 - "The average response time should be less than 500 milliseconds when executing a query"

Why 500ms?

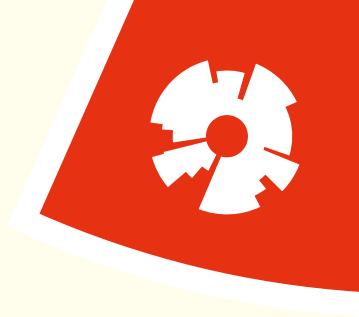


Limits



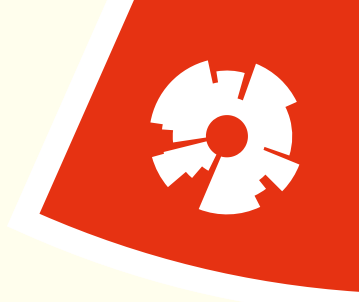
- In practice, deciding limits for quality requirements is really hard
- Beware of arbitrary limits (is 500ms the default value?)
 - It could be that 500ms costs you 1000000\$ more than 600ms

Limits



- If you can, make an informed decision
 - E.g., based on literature
(500ms is the threshold where users start complaining?)
 - Usage statistics
(e.g., 80% of the users leave if the application does not respond in 500ms)
 - Typical response times in old systems
(The old system took 1s. Users found this irritating.)

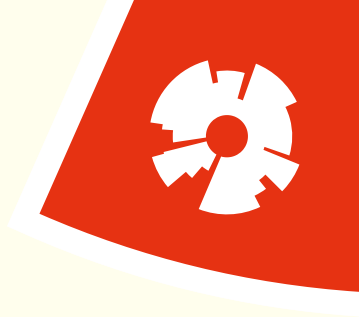
PLanguage



TAG	A unique, persistent identifier
GIST	A short, simple description of the concept contained in the PLanguage statement
STAKEHOLDER	A party materially affected by the requirement
SCALE	The scale of measure used to quantify the statement
METER	The process or device used to establish location on a SCALE
MUST	The minimum level required to avoid failure
PLAN	The level at which good success can be claimed
STRETCH	A stretch goal if everything goes perfectly
WISH	A desirable level of achievement that may not be attainable through available means
PAST	An expression of previous results for comparison
TREND	An historical range or extrapolation of data
RECORD	The best-known achievement
DEFINED	The official definition of a term
AUTHORITY	The person, group, or level of authorization

Simmons, "Quantifying Quality Requirements Using PLanguage", 2001

Requirements Specification Document

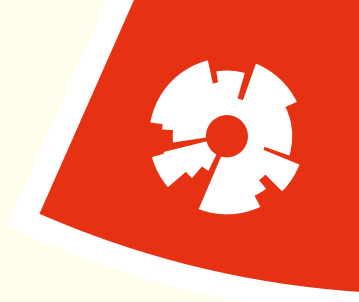


- (Not necessarily standard today)
- But: Guidance!

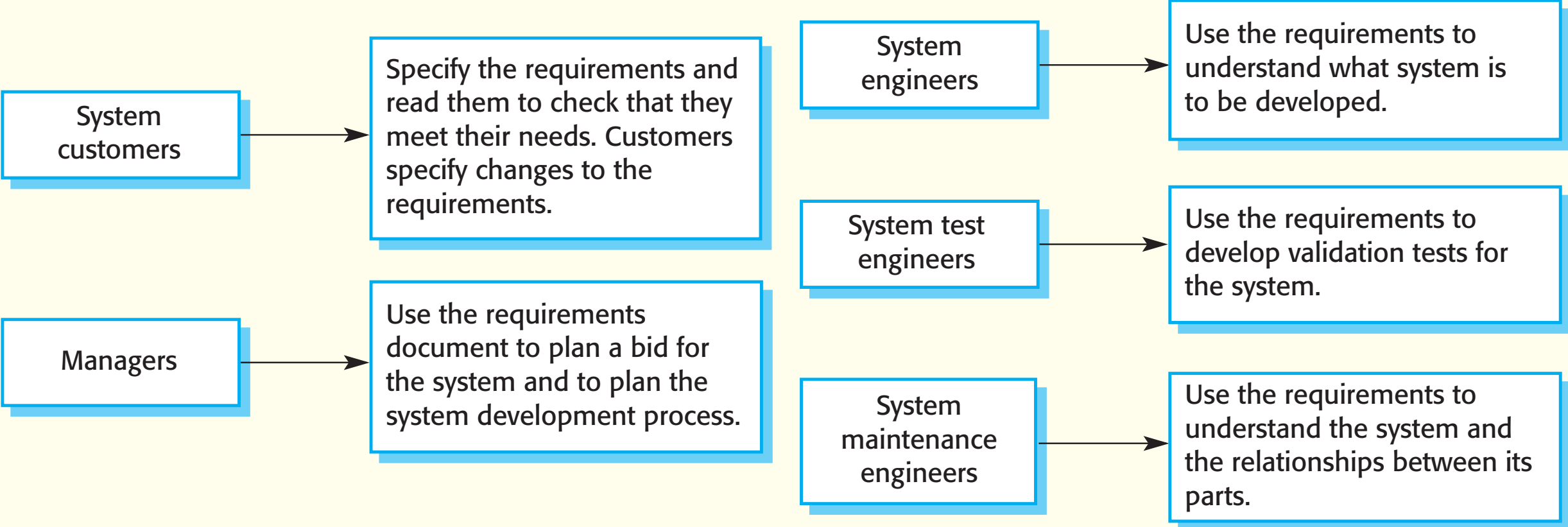
- Example: IEEE Std 830-1998, "Recommended Practice for Software Requirements Specifications"

Table of Contents

1. Introduction
 - 1.1 Purpose
 - 1.2 Scope
 - 1.3 Definitions, acronyms, and abbreviations
 - 1.4 References
 - 1.5 Overview
 2. Overall description
 - 2.1 Product perspective
 - 2.2 Product functions
 - 2.3 User characteristics
 - 2.4 Constraints
 - 2.5 Assumptions and dependencies
 3. Specific requirements (See 5.3.1 through 5.3.8 for explanations of possible specific requirements. See also Annex A for several different ways of organizing this section of the SRS.)
- Appendixes
- Index



Specification Audiences



[Sommerville]

Good Requirements Specifications



- A requirements specification should be
 - Correct
 - Complete
 - Unambiguous
 - Consistent
 - Ranked for importance/stability
 - Modifiable
 - Verifiable
 - Traceable

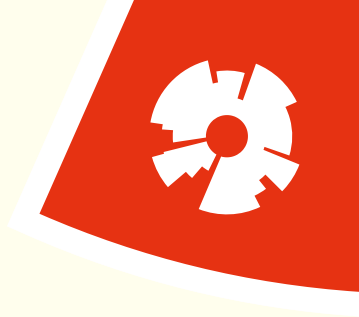
Achieving most of these is practically impossible

From: Soren Lauesen: Software Requirements
© Pearson / Addison-Wesley 2002



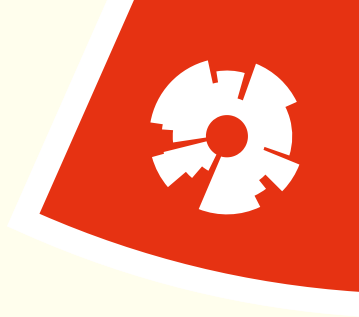
Validation

Requirements validation



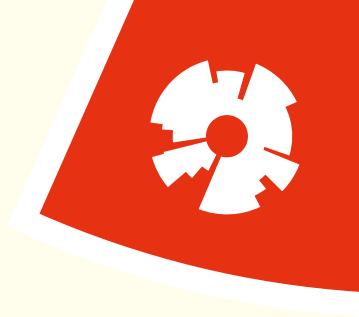
- Do the requirements define the system that the customer needs?
- Finding errors early: large cost savings
- Check for:
 - Correctness, Completeness, Free from ambiguity, Consistency, Ranked for importance/stability, Modifiability, Verifiability, Traceability
- Techniques: Reviews, Prototypes, Checklists...

Requirements Reviews



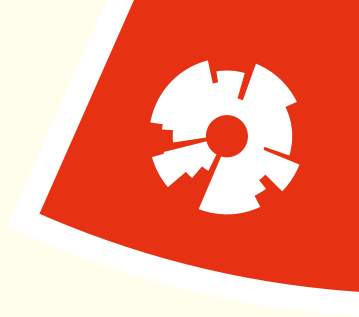
- Do regularly!
- Include stakeholders (not just engineers)
- Different techniques
 - Ad-hoc/informal: No specific guidelines - just "read"
 - Checklists: Lists of questions/items to focus the reading
 - Perspective-based reading: Read the specification from a specific perspective (role)
 - N-fold inspection: n different groups go through the inspection process in parallel

Requirements Checks



- Content check : all required items/sections/parts present
- Structure check: right structure for each requirement (e.g., all requirements have an ID)
- Consistency check: contradictions between requirements
- CRUD check: read/write/update/delete of all entities

Requirements Tests

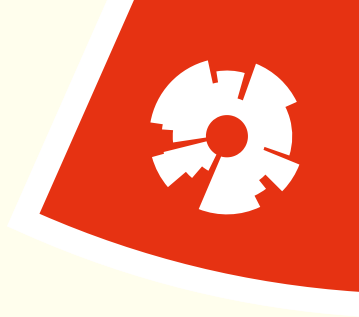


- Paper prototypes
 - Sketch the system (frontend) on paper
 - Have a user "use" the paper system
- Usability testing
 - T-216-GHOH
- Executable prototypes
 - Skeleton: Runs, but might not have functionality/quality
 - Formal models: More in the modelling lectures!



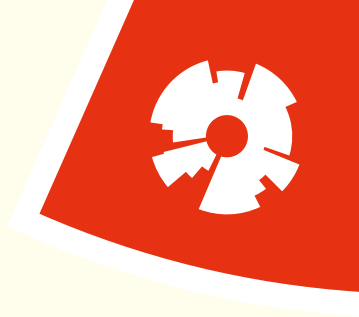
Management

Requirements management



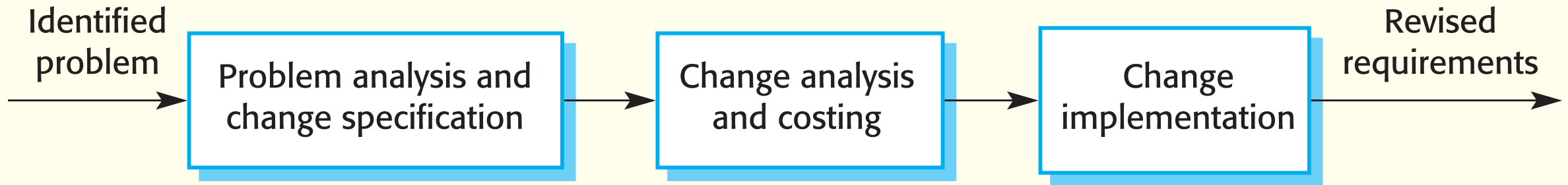
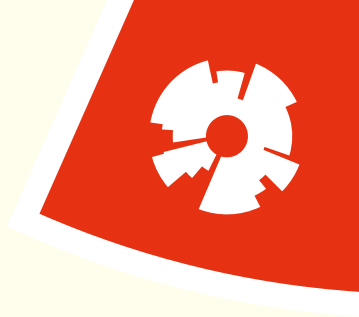
- Managing changing requirements (during the life-cycle)
- New requirements emerge all the time
- Keep track of individual requirements
 - Changes in content, priority, dependencies, ...
- Maintain links between requirements (and other artefacts)
 - "Traceability" - an entire research area
 - Many standards/regulations require traceability from code or tests to requirements!

Elements of Requirements Management



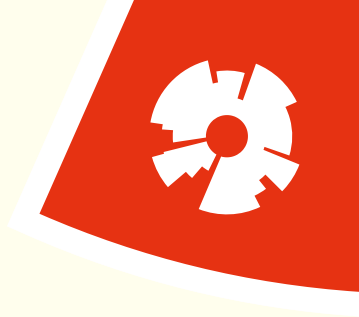
- Requirements identification: Is each requirement uniquely identified?
- Change Process: Where do changes come from? How are they registered and handled?
- Traceability policy: What is traced to what? Who creates/maintains traces?
- Tool support: What tools are used? With what other systems do they interact?

Requirements Change Management



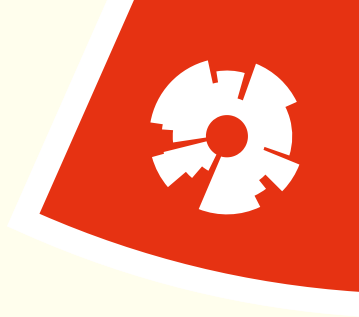
[Sommerville]

Discussion Points



- RE is not only about creating a plan that is then implemented
- For many companies, the value is **knowledge management**
 - Point of reference to understand the system
 - Can help introducing new staff
 - Helps to understand impact of changes
 - Avoids accidental changes
- RE matters also in agile development!

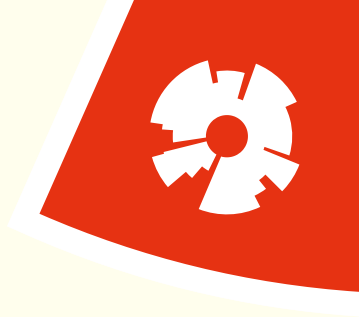
Questions



- What does the term 'stakeholder' describe in RE?
- Describe the purpose of requirements elicitation/specification/management
- What potential purposes can a specification serve?
- What makes requirements elicitation difficult?
- What alternatives exist to natural language specifications?
- What makes quality requirements challenging?
- What is the aim of requirements validation?

Next Module

- Processes
 - Plan-driven processes (waterfall, incremental)
 - Agile development
 - Scrum
- Literature: Ch. 2, 3 [Sommerville]





Sources

Education: Designed by Freepik

Assignment icons: Designed by ibrandify /
Freepik

Todo: Designed by Makyzz / Freepik

Megaphones: Designed by Freepik

