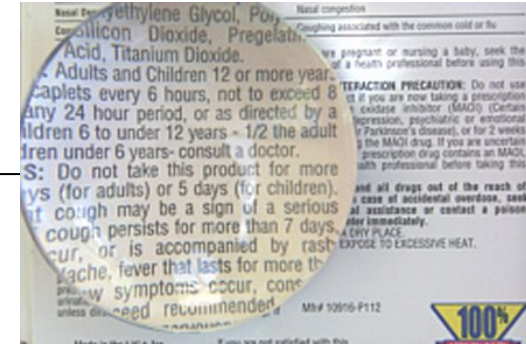


# 11 Validating requirements

- Every requirement needs to be validated (see Principle 6 in Chapter 2)
- Validate content, form of documentation and agreement
- Establish short feedback cycles
- Use appropriate techniques
- Exemplify and disambiguate with acceptance test cases



# Validation of content

---

Identify requirements that are

- Inadequate
- Incomplete or missing
- Inconsistent
- Wrong

Also look for requirements with these quality defects:

- Not verifiable
- Unnecessary
- Not traceable
- Premature design decisions

# Validation of requirements artifacts

---

Scope: checking the requirements **artifacts** (e.g., a systems requirements specification or a collection of user stories) for formal problems

Identify requirements that are

- **Ambiguous**
- **Incomprehensible**
- **Non-conforming** to documentation **rules**, **structure** or **format**

# Validation of agreement

---

- Requirements elicitation involves achieving **consensus** among stakeholders having divergent needs
- When validating requirements, we have to check whether **agreement** has actually been **achieved**
  - All known **conflicts resolved**?
  - For all requirements: have all relevant stakeholders for a requirement **agreed** to this requirement in its **documented form**?
  - For every **changed** requirement, have all relevant stakeholders **agreed to this change**?

# Some validation principles

---

## General principles

- Work with the **right people** (i.e., **stakeholders** for requirements)
- **Separate** the processes of **problem finding** and **correction**
- Validate from different **views** and **perspectives**
- Validate **repeatedly / continuously**

## Additional principles for requirements [Pohl and Rupp 2011]

- Validate by change of **documentation type**  
e.g., identify problems in a natural language specification by constructing a model
- Validate by construction of **artifacts**  
e.g., identify problems in requirements by writing the user manual, test cases or other development artifacts

# Requirements validation techniques

---

## Review

- Main means for requirements validation
- Walkthrough: author guides experts through the specification
- Inspection: Experts check the specification
- Author-reviewer-cycle: Requirements engineer continuously feeds back requirements to stakeholder(s) for review and receives feedback

## Requirements Engineering tools

- Help find gaps and contradictions

## Acceptance test cases

- Help disambiguate / clarify requirements

# Requirements validation techniques – 2

---

## Simulation/Animation

- Means for investigating dynamic system behavior
- Simulator executes specification and may visualize it by animated models

## Prototyping

- Lets stakeholders judge the practical usefulness of the specified system in its real application context
- Prototype constitutes a sample model for the system-to-be
- Most powerful, but also most expensive means of requirements validation

## Formal Verification / Model Checking

- Formal proof of critical properties

# Reviewing practices

---

- Paraphrasing
  - Explaining the requirements in the reviewer's own words
- Perspective-based reading
  - Analyzing requirements from different perspectives, e.g., end-user, tester, architect, maintainer,...
- Playing and executing
  - Playing scenarios
  - Mentally executing acceptance test cases
- Checklists
  - Using checklists for guiding and structuring the review process



# Requirements negotiation

---

- Requirements negotiation implies
  - Identification of conflicts
  - Conflict analysis
  - Conflict resolution
  - Documentation of resolution
- Requirements negotiation can happen
  - While eliciting requirements
  - When validating requirements

# Conflict analysis

---

Identifying the underlying reasons of a conflict helps select appropriate resolution techniques

Typical underlying reasons are

- **Subject matter** conflict (divergent factual needs)
- Conflict of **interest** (divergent interests, e.g. cost vs. function)
- Conflict of **value** (divergent values and preferences)
- **Relationship** conflict (emotional problems in personal relationships between stakeholders)
- **Organizational** conflict (between stakeholders on different hierarchy and decision power levels in an organization)

# Conflict resolution

---

- Various strategies / techniques
- Conflicting stakeholders must be involved in resolution
- Win-win techniques
  - Agreement
  - Compromise
  - Build variants
- Win-lose techniques
  - Overruling
  - Voting
  - Prioritizing stakeholders (important stakeholders override less important ones)

# Conflict resolution – 2

---

- Decision support techniques
  - PMI (Plus-Minus-Interesting) categorization of potential conflict resolution decisions
  - Decision matrix (Matrix with a row per interesting criterion and a column per potential resolution alternative. The cells contain relative weights which can be summarized per column and then compared)

# Acceptance testing

---

DEFINITION. **Acceptance** – The process of assessing whether a system **satisfies all its requirements**.

DEFINITION. **Acceptance test** – A test that assesses whether a system satisfies all its requirements.

# Requirements and acceptance testing

---

Requirements engineering and acceptance testing are naturally **intertwined**

- For every requirement, there should be **at least one acceptance test case**
- Requirements must be written such that acceptance tests can be written to **validate** them
- Acceptance test cases can serve
  - for **disambiguating** requirements
  - as **detailed specifications** by example

# Choosing acceptance test cases

---

Potential coverage criteria:

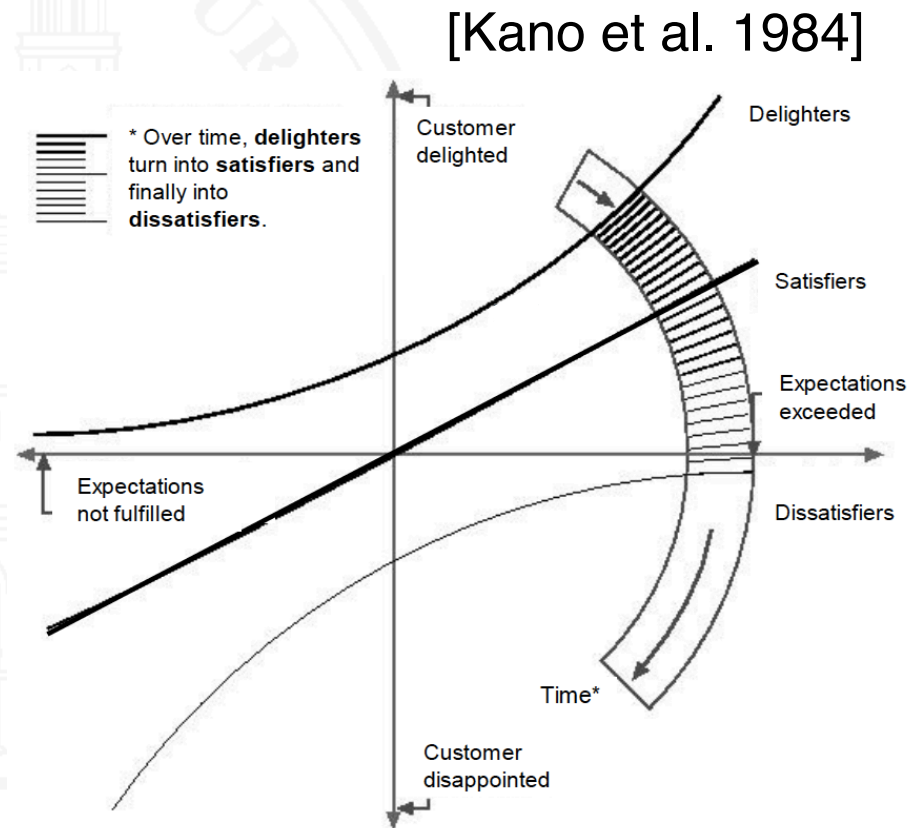
- **Requirements** coverage: At least one case per requirement
- **Function** coverage: At least one case per function
- **Scenario** coverage: For every type scenario / use case
  - All actions covered
  - All branches covered
- Consider the **usage profile**: not all functions/scenarios are equally frequent / important

# 12 Innovative requirements

Satisfying stakeholders is not enough  
(see Principle 8 in Chapter 2)

○ Kano's model helps identify...

- what is **implicitly expected** (dissatisfiers)
- what is **explicitly required** (satisfiers)
- what the stakeholders don't know, but would **delight** them if they get it: **innovative requirements**





# How to create innovative requirements?

---

## Encourage **out-of-the-box thinking**

- Stimulate the stakeholders' **creativity**
  - Imagine/ make up scenarios for possible futures
  - Imagine a world without constraints and regulators
  - Find and explore metaphors
  - Study other domains
- **Involve solution experts** and **explore what's possible** with available and future technology
- Involve **smart people without domain knowledge**



[Maiden, Gitzikis and Robertson 2004]  
[Maiden and Robertson 2005]

# 13 Requirements management

---

- Organize
  - Store and retrieve
  - Record metadata (author, status,...)
- Prioritize
- Keep track: dependencies, traceability
- Manage change



# 13.1 Organizing requirements

---

Every requirement needs

- a **unique identifier** as a reference in acceptance tests, review findings, change requests, traces to other artifacts, etc.
- some **metadata**, e.g.
  - Author
  - Date created
  - Date last modified
  - Source (stakeholder(s), document, minutes, observation...)
  - Status (created, ready, released, rejected, postponed...)
  - Necessity (critical, major, minor)

# Storing, retrieving and querying

---

## Storage

- Paper and folders
- Files and electronic folders
- A requirements management tool

## Retrieving support

- Keywords
- Cross referencing
- Search machine technology

## Querying

- Selective views (all requirements matching the query)
- Condensed views (for example, statistics)

## 13.2 Prioritizing requirements

---

- Requirements may be **prioritized** with respect to various criteria, for example
  - Necessity
  - Cost of implementation
  - Time to implement
  - Risk
  - Volatility
- Prioritization is done by the **stakeholders**
- Only a **subset** of all requirements may be prioritized
- Requirements to be prioritized should be on the **same level of abstraction**



# Simple prioritization (by necessity)

---

Ranks all requirements in three categories with respect to **necessity**, i.e., their **importance for the success** of the system

- **Critical** (also called essential, or mandatory)  
The system will **not be accepted** if such a requirement is not met
- **Major** (also called conditional, desirable, important, or optional)  
The system **should meet** these requirements, but not meeting them is **no showstopper**
- **Minor** (also called nice-to-have, or optional)  
Implementing these requirements is **nice**, but **not needed**

# Selected prioritization techniques

---

## Single criterion prioritization

- Simple ranking

Stakeholders rank a set of requirements according to a given criterion

- Assigning points

Stakeholders receive a total of  $n$  points that they distribute among  $m$  requirements

- Prioritization by multiple stakeholders may be consolidated using weighted averages. The weight of a stakeholder depends on his/her importance

# Selected prioritization techniques – 2

---

## Multiple criterion prioritization

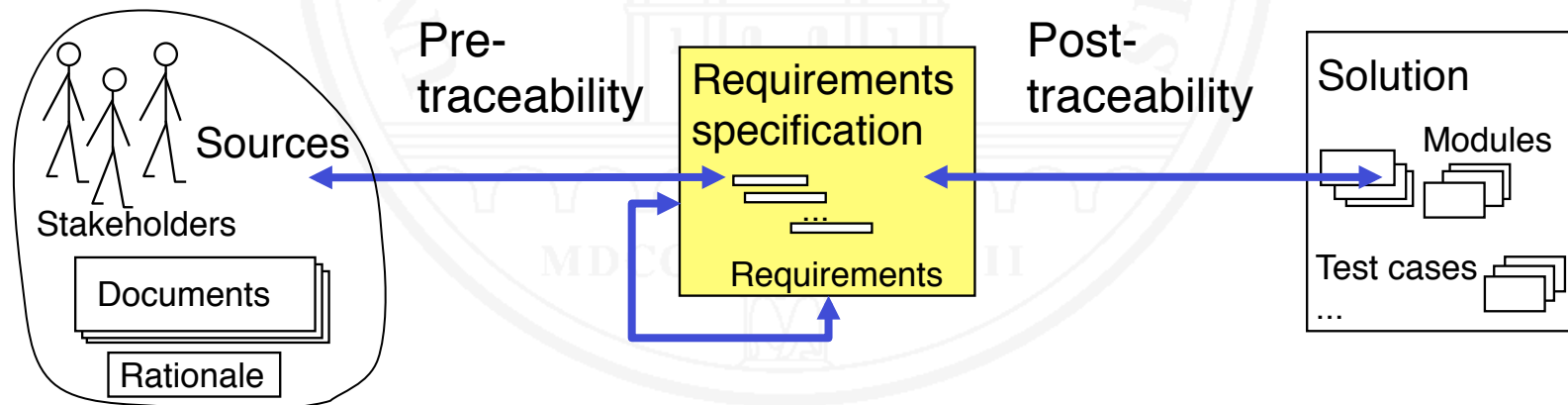
- **Wieggers' matrix [Wieggers 1999]**
  - Estimates relative benefit, detriment, cost, and risk for each requirement
  - Uses these values to calculate a weighted priority
  - Ranks according to calculated priority values
- **AHP (Analytic Hierarchy Process) [Saaty 1980]**
  - An algorithmic multi-criterion decision making process
  - Applicable for prioritization by a group of stakeholders



# 13.3 Traceability

[Gotel and Finkelstein 1994]

DEFINITION. **Traceability** – The ability to trace a requirement  
(1) back to its origins,  
(2) forward to its implementation in design and code,  
(3) to requirements it depends on (and vice-versa).  
Origins may be stakeholders, documents, rationale, etc.



# Establishing and maintaining traces

---

## ○ Manually

- Requirements engineers explicitly create traces when creating artifacts to be traced
- Tool support required for maintaining and exploring traces
- Every requirements change requires updating the traces
- High manual effort; cost and benefit need to be balanced

## ○ Automatic

- Automatically create candidate trace links between two artifacts (for example, a requirements specification and a set of acceptance test cases)
- Uses information retrieval technology
- Requires manual post processing of candidate links

## 13.4 Requirements evolution

---

The **problem** (see Principle 7 in Chapter 2):  
Keeping requirements **stable**...  
... while permitting requirements to **change**

Potential **solutions**

- Agile / iterative development with short development cycles (1-6 weeks)
- Explicit requirements change management

Every solution to this problem further needs **requirements configuration management**

# Requirements configuration management

---

## Keeping track of changed requirements

- **Versioning** of requirements
- Ability to create requirements **configurations, baselines** and **releases**
- **Tracing** the reasons for a change, for example
  - Stakeholder demand
  - Bug reports / improvement suggestions
  - Market demand
  - Changed regulations

# Requirements change management

---

## Adhering to a strict **change process**

- (1) Submit change request
- (2) Triage. Result: [OK | NO | Later (add to backlog)]
- (3) If OK: Perform impact analysis
- (4) Submit result and recommendation to Change Control Board
- (4) Decision by Change Control Board
- (5) If positive: make the change, create new baseline/release,  
(maybe) adapt the contract between client and supplier

**Change control board** – A committee of **client** and **supplier** representatives that **decides** on **change requests**.

# Requirements change in agile development

---

In agile and iterative development processes, a **requirements change request ...**

- ... **never affects the current sprint / iteration**, thus ensuring **stability**
- ... is added to the **product backlog**

**Decisions** about change requests are made when prioritizing and selecting the requirements for the subsequent sprints / iterations