Karlsruhe Institute of Technology

Softwaretechnik II

Oliver Hummel, IPD

Topic 4
Requirements Engineering

SOFTWARE DESIGN AND QUALITY GROUP
INSTITUTE FOR PROGRAM STRUCTURES AND DATA ORGANIZATION, FACULTY OF INFORMATICS

sdq.ipd.kit.edu

KIT – University of the State of Baden-Wuerttemberg and
National Research Center of the Helmholtz Association

www.kit.edu

Overview on Today's Lecture

- Content
  - Brief Repetition & Foundations
  - User Stories
  - Requirements Priorization
    - Story Maps
  - Requirements Validation

- Learning Goals
  - Get an overview of software requirements elicitation
    - understand some selected requirements capturing approaches
  - Be aware of the subtle problems that might arise with them
    - be able to develop a solution in practice

Who‘s that Guy?

- *The indispensable first step to getting the things you want out of life is this: decide what you want.*

  -- Ben Stein

[Neshan Naltchayan, Wikipedia]

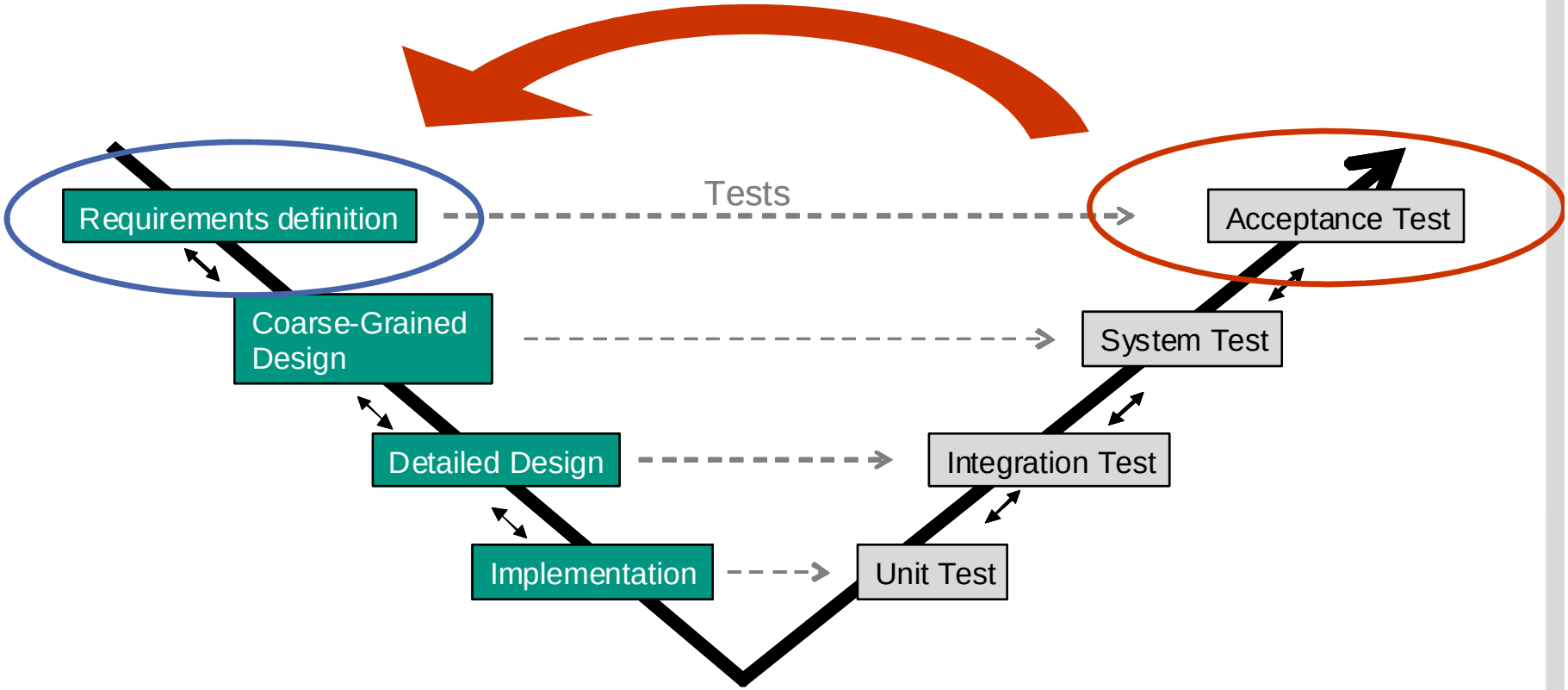
Some Motivation

- ~60% of software defects are coming from incorrect requirements
  [Boehm, 1981]
  - ➔ *Requirements are missing*
  - ➔ *Requirements are wrong or missunderstood*
- Solving these problems during later phases is very expensive
  - Boehm estimates a **factor of ten per development phase**

- *You might want to listen to the* Software Engineering Radio — The Podcast for Professional Software Developers
  - *http://www.se-radio.net/2008/10/episode-114-christof-ebert-on-requirements-engineering*
  - ➔ *Christof Ebert's 3 biggest risks in requirements engineering*
    - wrong requirements
    - missing requirements
    - changing requirements

Software Requirements Specification

- The software requirements specification is both the starting point and end point for a software development project
  - probably the most important artifact in a project



Repetition

- What are requirements?
  - something required, wanted or needed

Requirement:
(1) A condition or capability needed by a user to solve a problem or achieve an objective.

(2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.

(3) A documented representation of a condition or capability as in (1) or (2).

[IEEE 610.12-1990]

Software Requirements

- Ideally, software requirements are described in a way that they are –
  1. _____
  2. _____
  3. _____
  4. _____
  5. _____
  6. _____
  7. _____

[IEEE]

- We distinguish three kinds of requirements, namely –
  1. _____
  2. _____
  3. _____

The Requirements Engineering Process

- Cooperative, iterative and incremental process of –
  - Requirements Elicitation
    - Documentation of Goals
    - Scenarios
    - Problem Analysis
  - Requirements Documentation
    - Create Software Requirements Specification (SRS)
  - Cross cutting actions:
    - Requirements Validation
    - Requirements Management

➔ determine and comprehend **all** relevant requirements in necessary degree of detail

➔ achieve acknowledgement of requirements by involved stakeholders

Stakeholder

- A person or organisation that (in)directly influences the requirements of a system
  - Users of the system
    - labor union?
  - Operator of the system
  - Purchaser / Sponsor / Controller
  - Software Developer
  - Software Architects
  - Tester
- Identification of relevant Stakeholders and their relationships is critical for RE success
  - missing Stakeholders may lead to missing requirements
- Also take care of potential political interests of your stakeholders!

The Requirements Engineer

- Central role in the development process
  - translates between users and developers
  - sometimes also called business analyst
- Needs methodological skills
  - thinks analytically
  - has empathy
  - has communication skills
  - is conflict solving
  - has discussion moderation skills
  - is selfconcious
  - is cogent (überzeugend)
- Is responsible for requirements elicitation and documentation
  - maintains the requirements document as well

Requirements Elicitation Techniques

- Which techniques for finding requirements do you know?

Werbung…

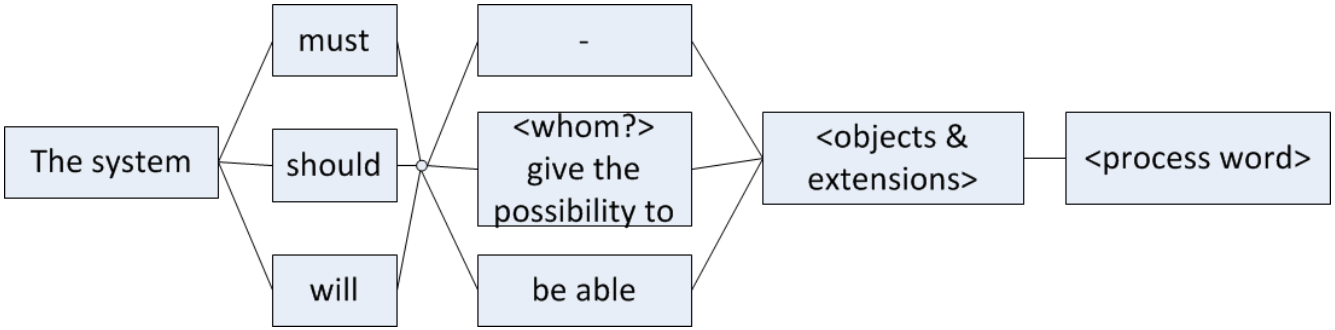

Karlsruher Institut für Technologie
Interview-Day
Der direkte Weg zum Traumjob!
TRAUMJOB
Bewerbungsgespräche direkt
auf dem Campus
28.11.2013
im Festsaal | Studentenhaus
Bewerbung bis 04.11.2013
Infos und Kurzbewerbung unter
www.careerservice.kit.edu
KIT
1&1
HOMAG
Palantir
[ipt]
KHS
Filling and Packaging – Worldwide
UNISERV
zeb/

Basic Writing Recommendations

- Most requirements are (initially) captured in natural language
  - as they need to be understandable by users
- Thus, try to follow the following writing guidelines –
  - Short sentences, one clear requirement per sentence
    - *BAD: The Navigation systems navigates to a destination with comfortable usability*.
    - **GOOD: The navigation system must allow in all modes to set the destination.**
  - Use active language that makes clear who is responsible for what
    - *BAD: When the PIN was validated, money can be withdrawn from the account.*
    - **GOOD: When the ATM has validated the PIN, the customer can withdraw money from his account.**
  - Avoid „weak“ words, such as –
    - *effective, user-friendly, easy, quickly, timely, reliable, appropriate*
  - Maintain a glossary of terms
    - and use them conforming to it

Basic Writing Recommendations II

- Use sentence templates, such as –



- or: [Trigger] Actor Action Object [Condition] [Intel]
  - Trigger: *When the Navigation System is set into on road mode*
  - Actor: *the Navigation System*
  - Action: *displays*
  - Object: *the distance to the destination*
  - Condition: *until another mode is chosen.*

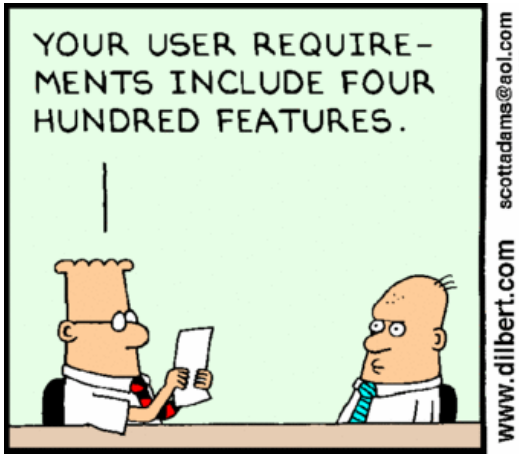
- *or others… see e.g. [Ebert 2008, p. 148]*

Traditional Feature Lists

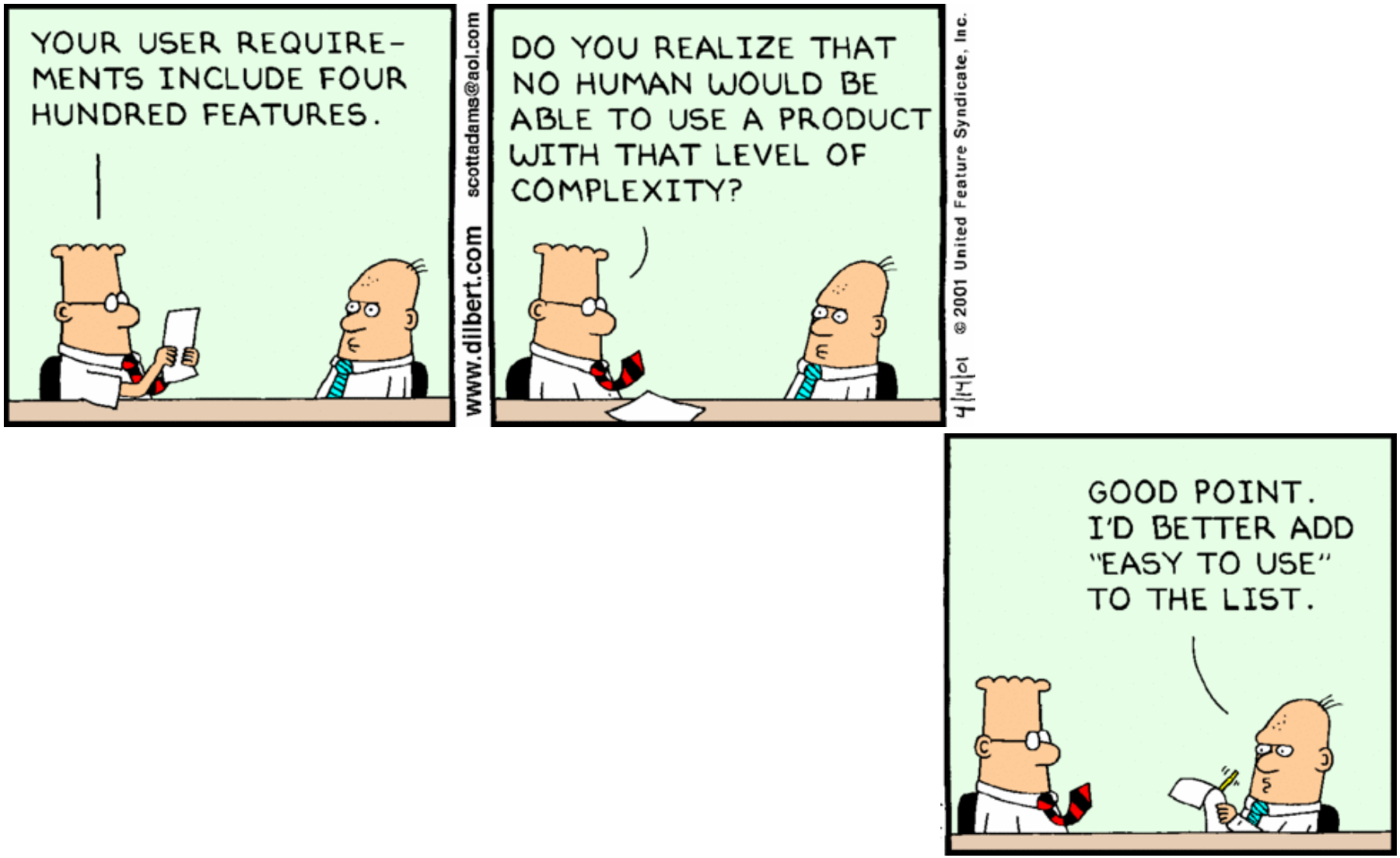
- Traditional requirements analysis methods typically result in detailed, low-level feature lists

ID	Feature
FEAT1.9	The system shall accept entry of item identifiers
….	
FEAT2.4	The system shall log credit payments to the accounts receivable system
….	

- Especially in the context of enterprise applications such long, detailed lists have some drawbacks as they –
  - do not describe requirements in a cohesive way
  - are unstructured and have the feel of a “laundry list”

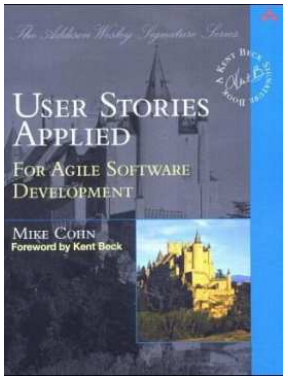


The Dilbert Solution



Modern Requirements Capture

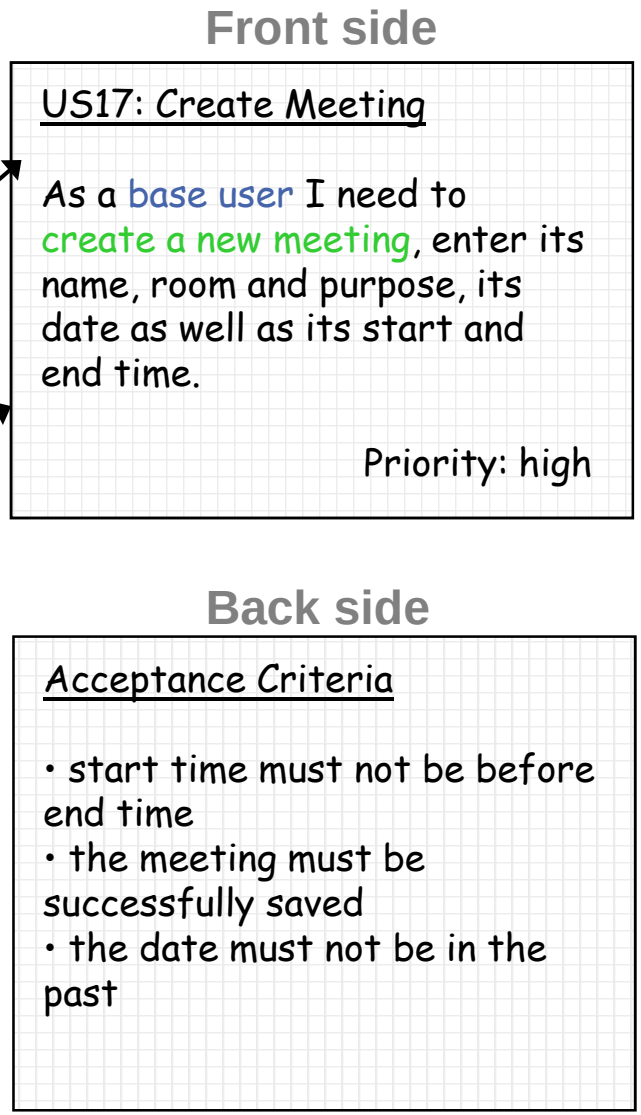
- Today's most common approaches for writing down requirements are –
  - user stories (agile) & use cases (model-based)
- Both focus on interaction of the user and the software
  - and place the requirements in the context of the stories and goals of using the system
  - chief virtues are simplicity and utility
- Brief history
  - use cases were introduced in 1986 by Ivar Jacobson
  - fleshed out in "Writing Effective Use Cases" by Alistair Cockburn
  - user stories are the predominant approach in agile processes



- Both are not always the most appropriate approach
  - some applications still call for a feature-driven approach
    - e.g. application servers, data products, middleware or back end systems

… with User Stories [Cohn04]

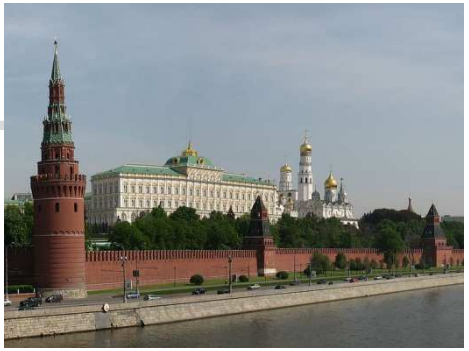
- User Stories are collected on index cards
  - not suitable for UI requirements
- They describe requirements from the **end-user's point of view**
  - they comprise
    - a **name** (and ideally an ID)
    - a brief **textual description**
    - **acceptance criteria**
- The textual description should contain –
  - the user's role
  - the goal of the story
  - optionally the benefit of the requirement



Agile Requirements Elicitation

1. The Product Owner initially populates the product backlog with **coarsely described requirements**
   - derived from the product concept and discussed with the customer
   - the goal is to quickly collect some requirements
2. The Product Owner **clusters** the requirements according to their overall themes
3. Product Owner and team **prioritize** the requirements
   - usually according to business value and perhaps risk
4. The Product Owner **refines** the requirements with the highest priorities
   - in requirements workshops with the customer
     - and the team as far as possible
   - the product backlog should now contain enough information to start with the first sprint
5. The requirements are updated, refined or even removed as needed

Priority: MoSCoW



[Минеева Ю., Wikipedia]

- … is the capital of Russia ☺
- … but also a way of prioritizing software requirements
  1. MUST have
     - a critical requirement that must be present in the product
  2. SHOULD have
     - important, but not absolutely necessary requirement
  3. COULD have
     - nice to have requirement that could increase customer satisfaction
  4. WON‘T have / WOULD like
     - requirement of relatively low importance that still increases business value

➔ [IEEE-830]: *Mandatory, Optional, Nice to have…*

Priority: Cost Value Analysis

- One dimensional priorization is often not sufficient
  - contrastive pairs are often more helpful
  - using value/importance vs. –
    - cost / time
    - risk
    - volatility
    - etc…
- for example

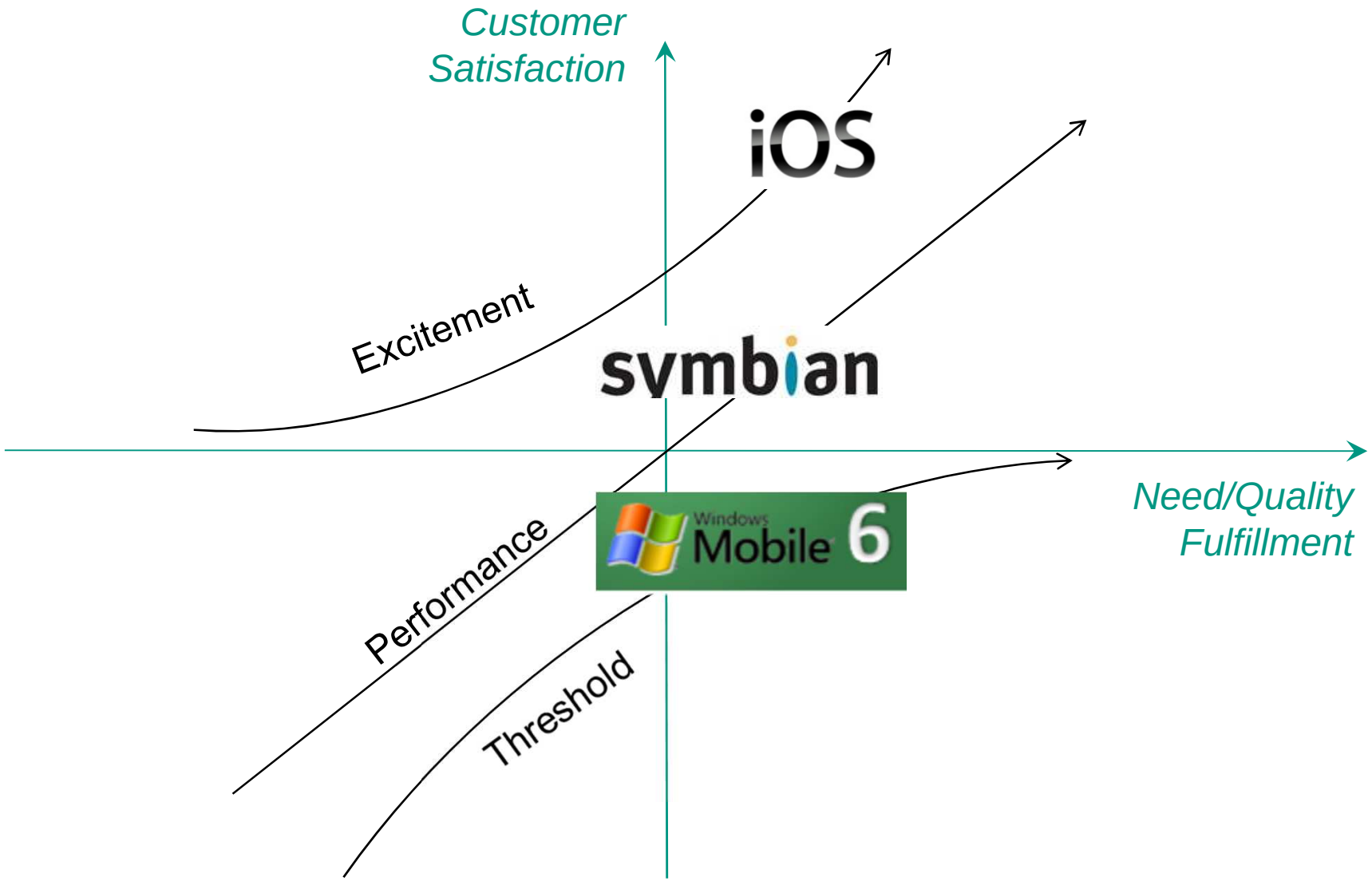
Reqmnt.	Value	Risk	-> Prio.
A	4	2	2
B	3	3	1
C	3	2	1.5

➔ *approach is context-dependent*

Priority: The Kano Model [Kano84]

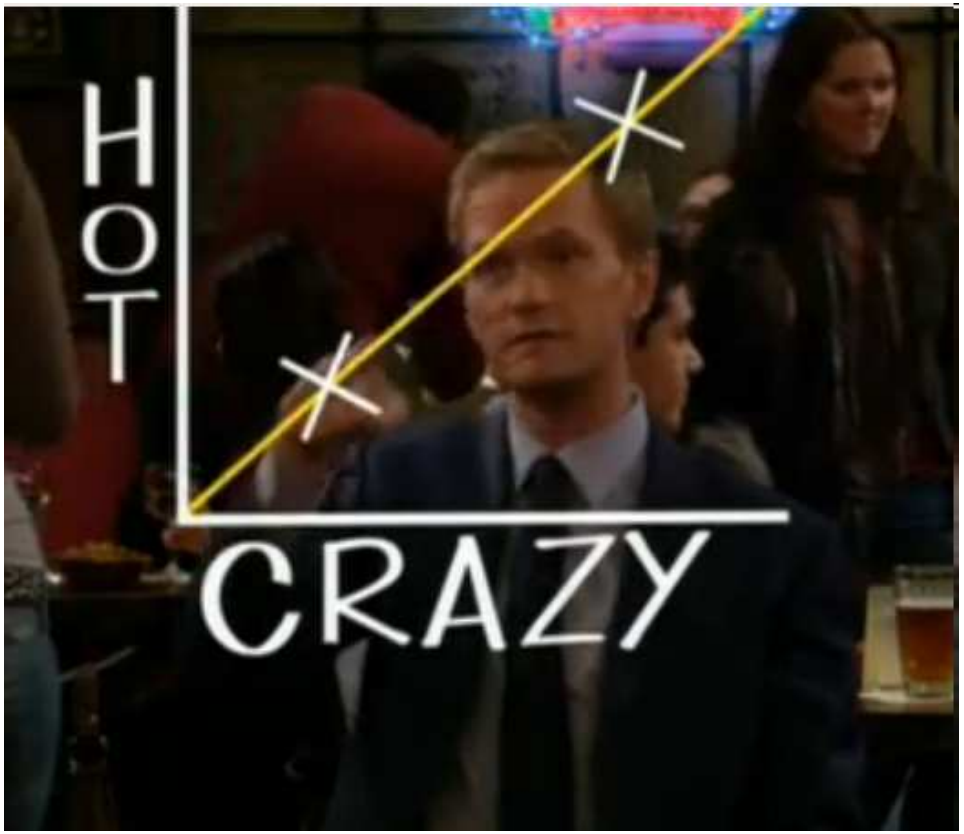
- Categorisation for requirements
  - Attractive Quality *(Excitement)*
    - features that make a system attractive to use
    - but are not expected by the user
  - One-dimensional Quality *(Performance)*
    - critical key functionality
    - explicit requirements that are often advertised
  - Must-be Quality *(Threshold)*
    - basic attributes taken for granted
    - often implicit requirements that are not spoken out
  - Indifferent Quality
    - features that do not influence customer satisfaction
  - Reverse Quality
    - people have different tastes

The Kano Model



Looks Familar..? ☺

- The Hot Crazy Scale



[How I met your mother]

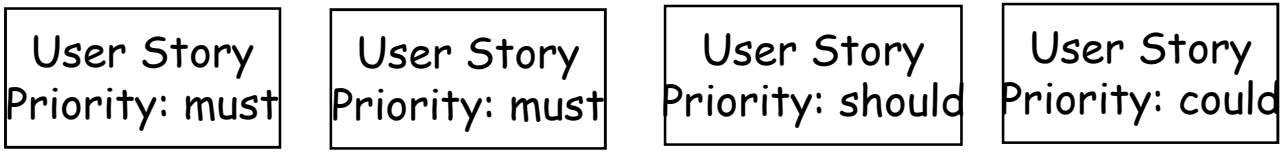
- http://www.youtube.com/watch?v=rNfXdHJ6Knc

Release Planning Again

- Does building the highest priority features first really deliver the best system increments?
  - or in other words: *how to assign requirements to meaningful releases?*

➔ A priority-driven assignment of requirements often leads to unusable intermediate releases

- since low-priority requirements are often needed to “hold the software together”

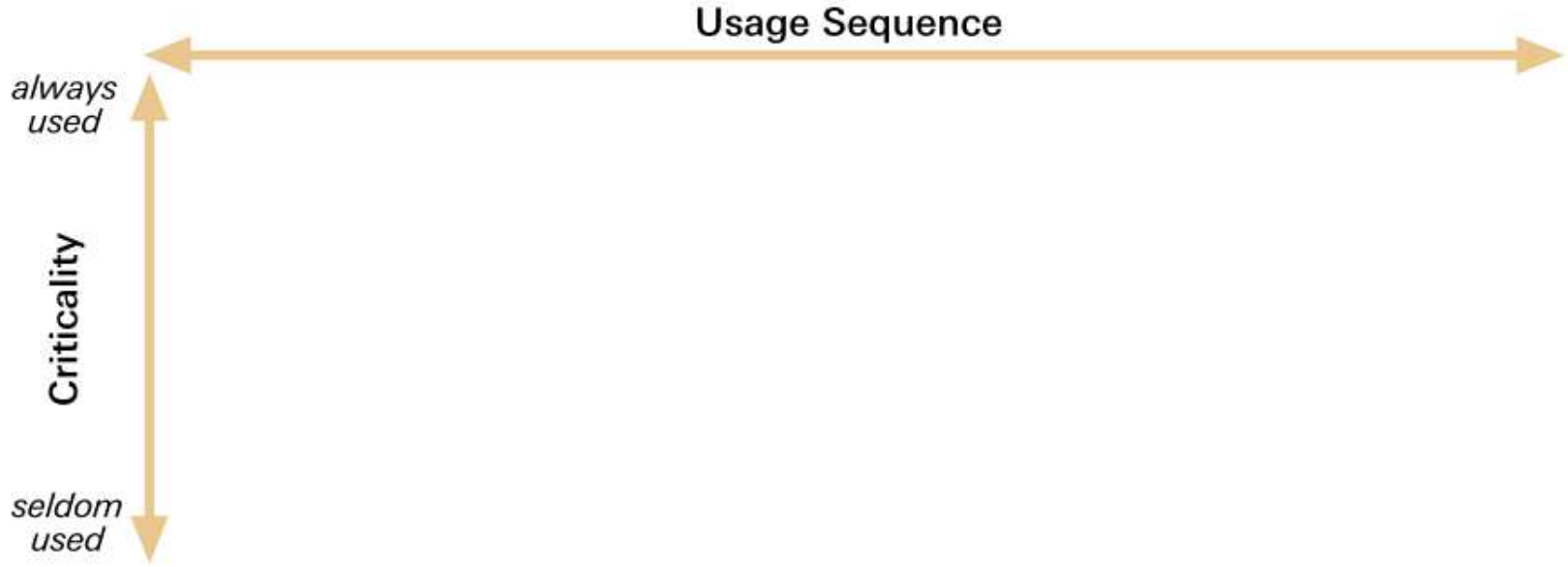


➔ The one-dimensional product backlog may not be sufficient to organize releases

➔ *“Design your project in working layers [of user stories] to avoid half-baked incremental releases”* [Patton05]

Story Maps [Patton05]

- The idea of story mapping is to arrange features (user stories) in two dimensions
  - according to their –
    - priority / criticality
    - natural usage sequence



Story Mapping Example

1. Start by collecting user stories as usual
   - for example for a software for small retailers
     1. Create purchase order for vendor
     2. Receive shipment from vendor
     3. Print price tags for received items
     4. Sell items
     5. Return and refund items
     6. Analyze sales
2. Detail and prioritize these user stories

   User Story 1: As a merchandise buyer I would like to create a purchase order (po) for a vendor.

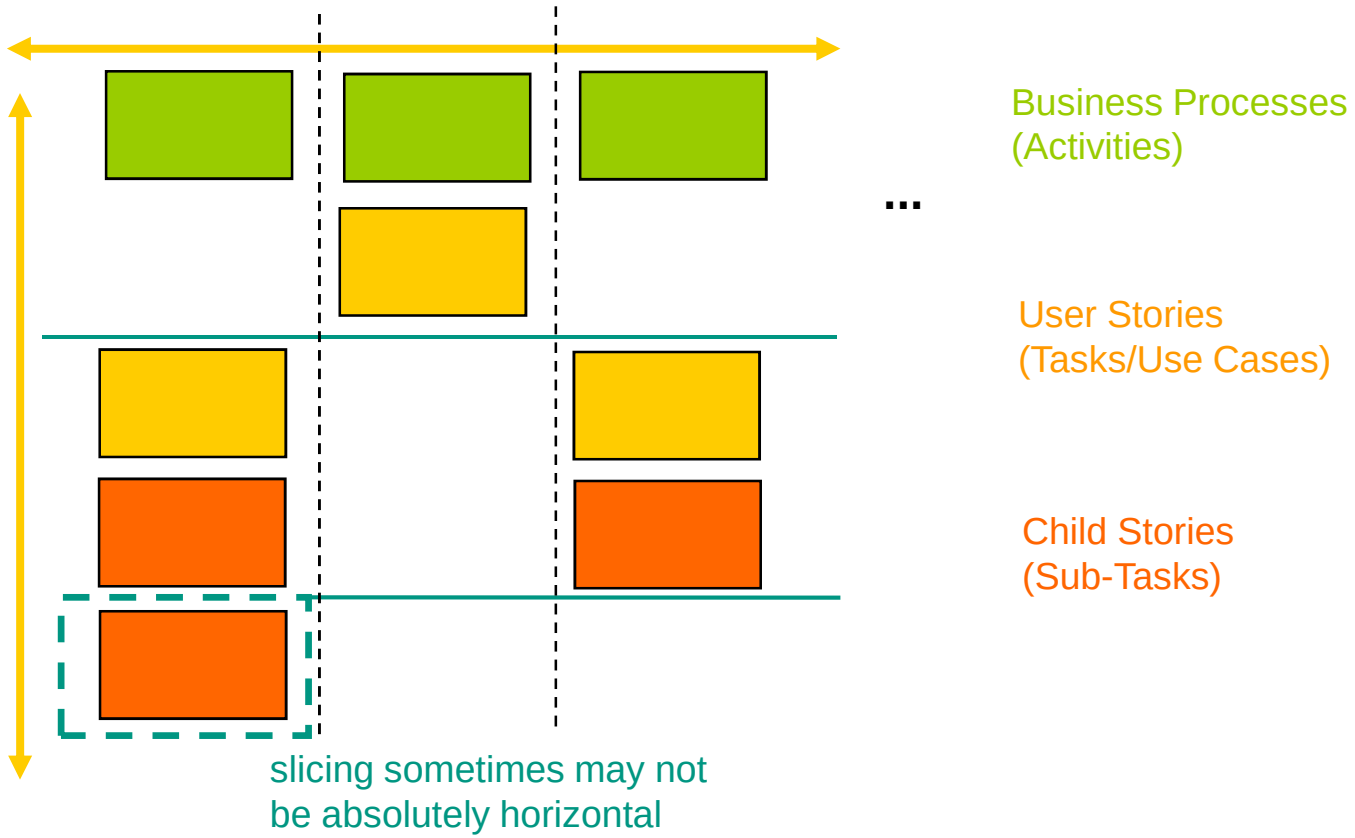
   frequency: weekly
   value: medium

   -> in the context of a shop management system

3. Arrange them in their natural sequential order

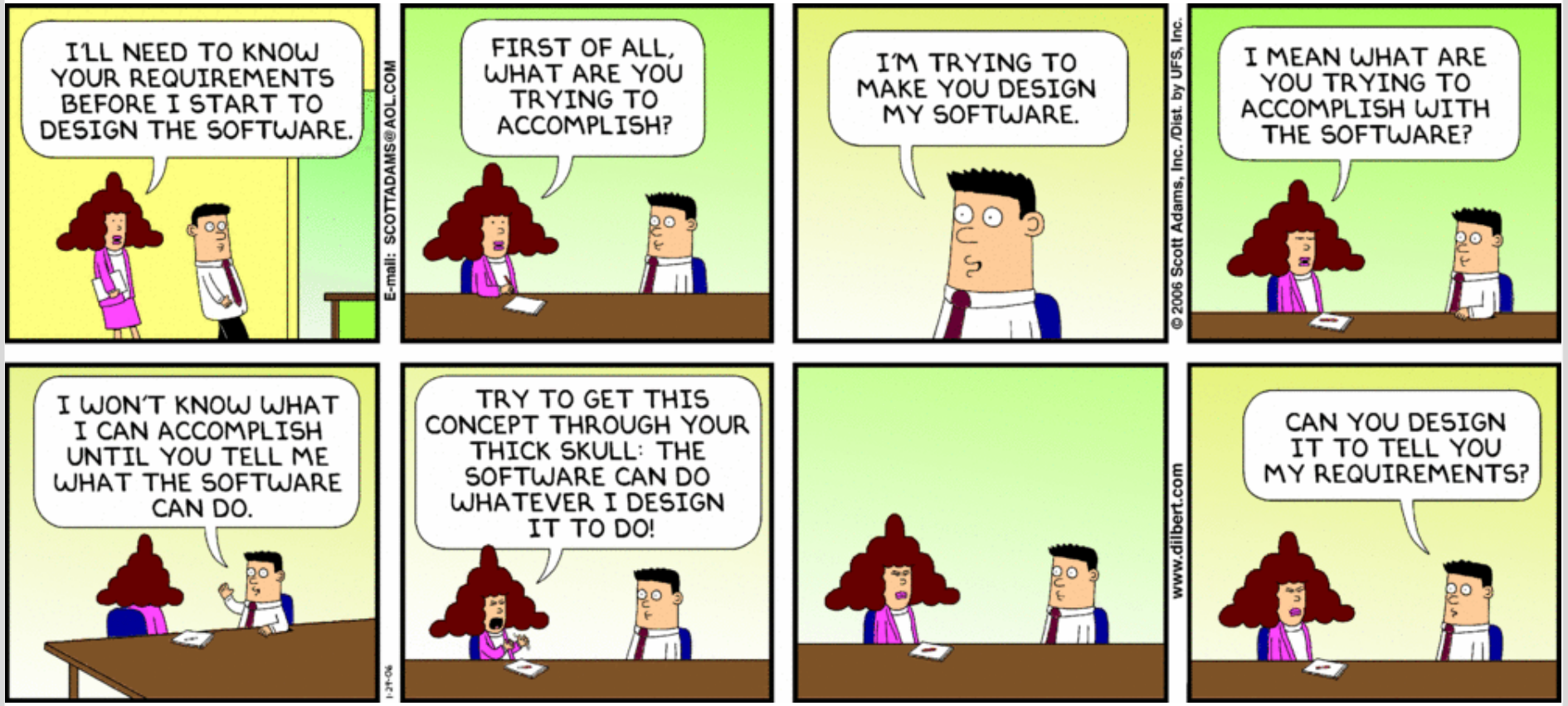
Practical Hints

- You may need to decompose user stories into child stories
  - use differently colored cards for them



➔ this model may be helpful for other I&I approaches as well

Requirements Elicitation Techniques



Requirements Validation

- *Remember: Requirements errors are expensive*
  - find errors in requirements as early as possible
- Validation: *Am I building the right system?*
- Verification: *Am I building the system right?*
  - correctness (relation between two documents)
- Validation of requirements artefacts (output)
  - Ambiguities, incompleteness, inconsistencies
- Validate context aspects (input)
  - Wrong or missing context information
- Validate RE process
  - Missing steps, stakeholders

➔ *Without a formal req. model, only reviews and prototypes help!*

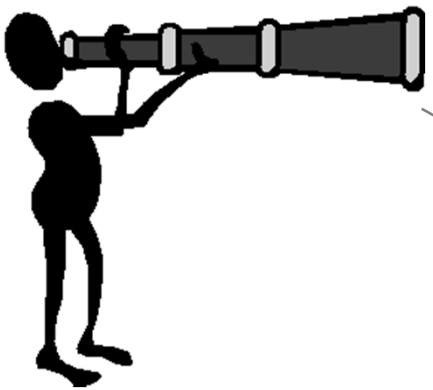
- *Simulation and verification when formal model is available*

Requirements Validation Techniques

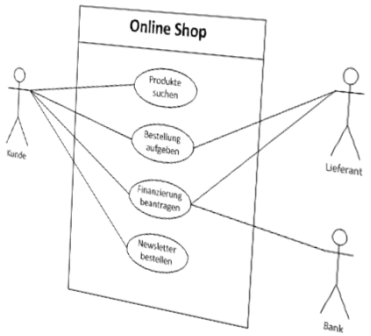
- Inspection, Reviews, Walkthroughs
  - find errors manually
- Simulation
  - simulate selected aspects of a system
- Prototyping
  - oriented towards design model
  - stakeholders test selected scenarios in a prototype
    - of special importance for UI design (usability testing)
- Creation of system test cases
- Model Checking
  - formal verification of used models (e.g. FSMs)

Conclusion

- Various techniques for capturing requirements are available today
  - agile approaches usually use user stories
- Prioritizing and validating requirements are important aspects
  - however, you always have to balance different aspects and influences
- Thank you for your attention!



Use Case Writing



Bibliography

- S. Ambler
  The Object Primer: Agile Model-Driven Development with UML 2.0
  Cambridge University Press, 2004
- Davis, A. M. (1993), Software Requirements: Objects, Functions and States, Prentice Hall PTR, New Jersey
- Patton, J. (2005) How You Slice it, Better Software.
  http://agileproductdesign.com/writing/how_you_slice_it.pdf
- Pohl, K. (2007), Requirements Engineering: Grundlagen, Prinzipien, Techniken, dpunkt, Heidelberg
- Lamsweerde, A. V. (2001), Goal-oriented requirements engineering: A guided tour
- Kano, N.; Seraku, N.; Takahashi, F. & Ichi TSUJI, S. (1984), 'Attractive Quality and Must-Be Quality',