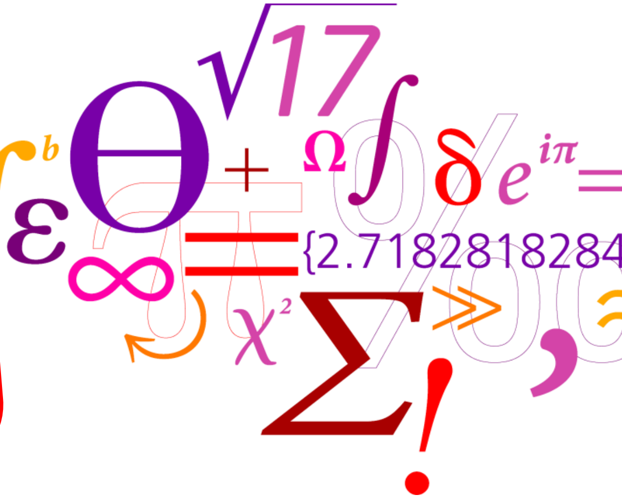
# Software Engineering 2

## A practical course in software engineering

Ekkart Kindler

**DTU Informatics**
Department of Informatics and Mathematical Modeling

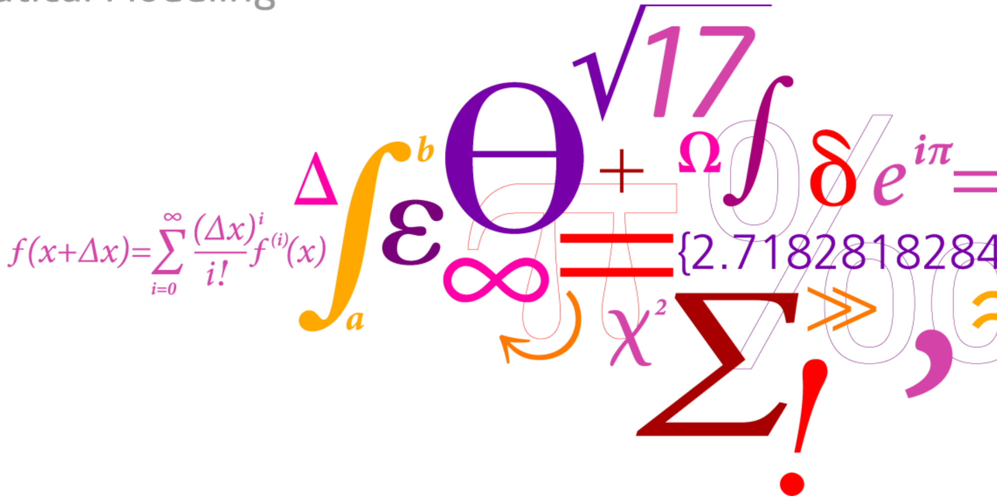
**Note**: These slides are a selection of the slides from lecture 3



# III. Specifying Software

**DTU Informatics**

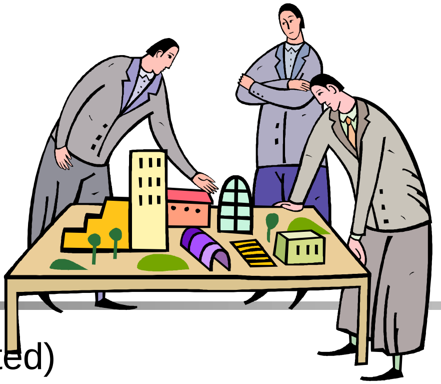
Department of Informatics and Mathematical Modeling

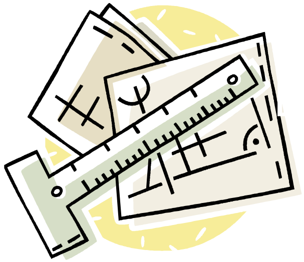

$f(x+\Delta x)=\sum_{i=0}^{\infty}\frac{(\Delta x)^i}{i!}f^{(i)}(x)$
$\Delta\int_a^b \varepsilon\, \Theta^{\sqrt{17}} + \Omega \int \delta e^{i\pi} =$
$\infty$
{2.7182818284
$\chi^2$
$\Sigma!$
≫

# III. Specifying Software

Goals:

- Defining what the software should do (before it is really there)
- **C**ustomer and **D**eveloper agree on what should be delivered
- Effort (resources and time) can be planned based on that (contract will be based on that).



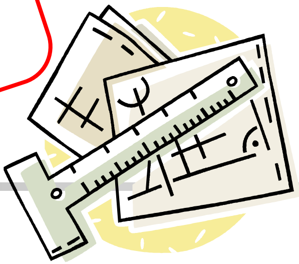


# Specifying Software

what

- Project Definition
- Requirements Specification
  - rough
  - detailed
- Systems specification
- Complete Models
- Implementation, Documentation Handbook

Actually, handbook is "what"; it could be part of the requirements specification.



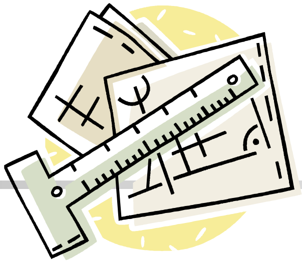
how

# Specifying Software

- Project Definition
- Requirements Specification
  - rough
  - detailed
- Systems specification
- Complete Models
- Implementation, Documentation Handbook

rough

detailed



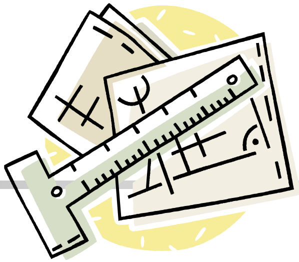
# Specifying Software

- Project Definition
- Requirements Specification
  - rough
  - detailed
- Systems specification
- Complete Models
- Implementation, Documentation Handbook

low cost

↓

high cost



# Specifying Software

Goals:

- Defining what the software should do before it is really there
- **C**ustomer and **d**eveloper agree on what should be delivered
- Effort (resources and time) can be planned based on that (contract will be based on that).

On which kind of document will (can) the cost calculation and the contract be based?

Trade off:
earlier: lower cost / higher risk
later: higher cost / lower risk

# Project Definition: Contents

what

rough

Use examples, how things could look like in the final product.

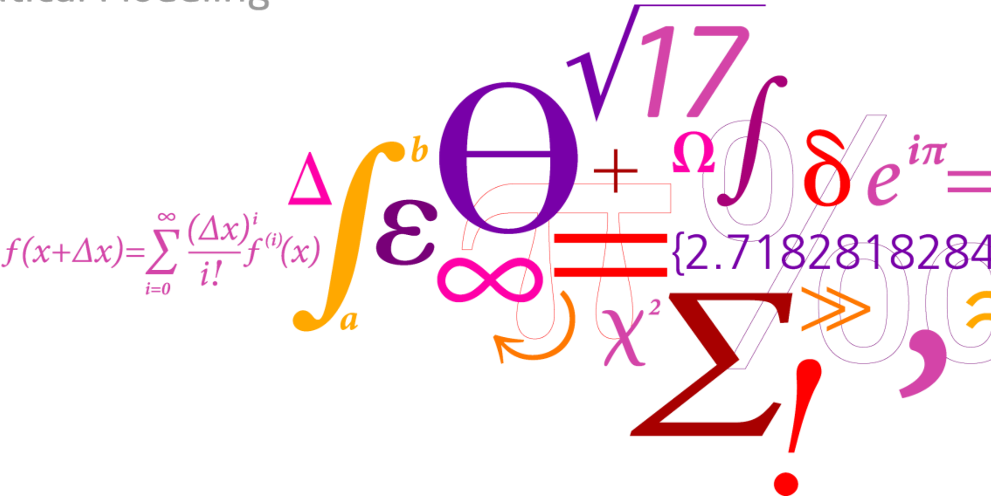
- Partners
- Context
- Objective
- Scope
  (in particular, what is NOT to be done)
- **Functionality** (from the end-users point of view)
  - Users
  - Use cases (as text, not necessarily as diagrams)
  - Main data (in our case ”modelling concepts”, ”extra 3D info”)
- Platform (HW/SW)
- Glossary of main terms

→ inductive vs deductive writing!

# 2. Requirements Specification

**DTU Informatics**

Department of Informatics and Mathematical Modeling


$f(x+\Delta x)=\sum_{i=0}^{\infty}\frac{(\Delta x)^i}{i!}f^{(i)}(x)$
$\sqrt{17}$
{2.7182818284

# Specifying Software

- Project Definition
- Requirements Specification
  - rough
  - detailed
- Systems specification
- Complete Models
- Implementation, Documentation Handbook

# What do we need?

“Why”

- What should be achieved by the product?

- How is it used?
- Which functions does it have?
- Which data are there?
- What interfaces should be there?

“What”

- In which quality?

- On which platform or technology?

“how”

# Requirements Spec

Partners: Customer & Developer

1. Objectives
2. Product use
3. Product functions
4. Product characteristics (non-functional req.)
   - Platform
   - Performance
   - Security
   - …
5. Glossary
   (could be included somewhere else)

This can be done on different levels of detail: Project proposal, requirements specification, systems specification, final documentation.

# Differences

- Project definition / idea
- Text (possibly sketch of screen shots); not complete

- Requirements specification
  - Rough
  - Use cases (named), glossary, rough domain model
  - detailed
  - Use cases modelled and explained, complete domain model, GUI design (sketch), acceptance tests

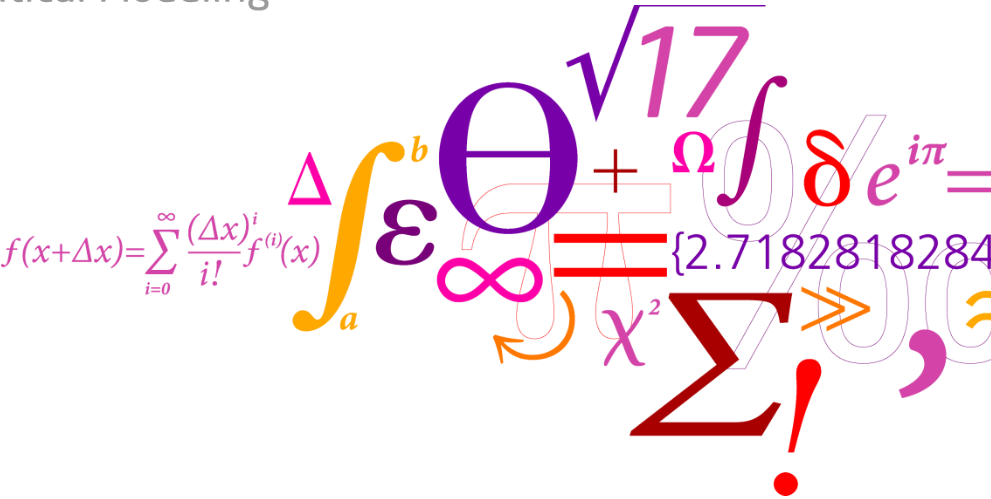
The exact definition of different specification types varies: structure, level of detail, models, …

- Systems specification
- Architecture & design of Software, detailed models, software models

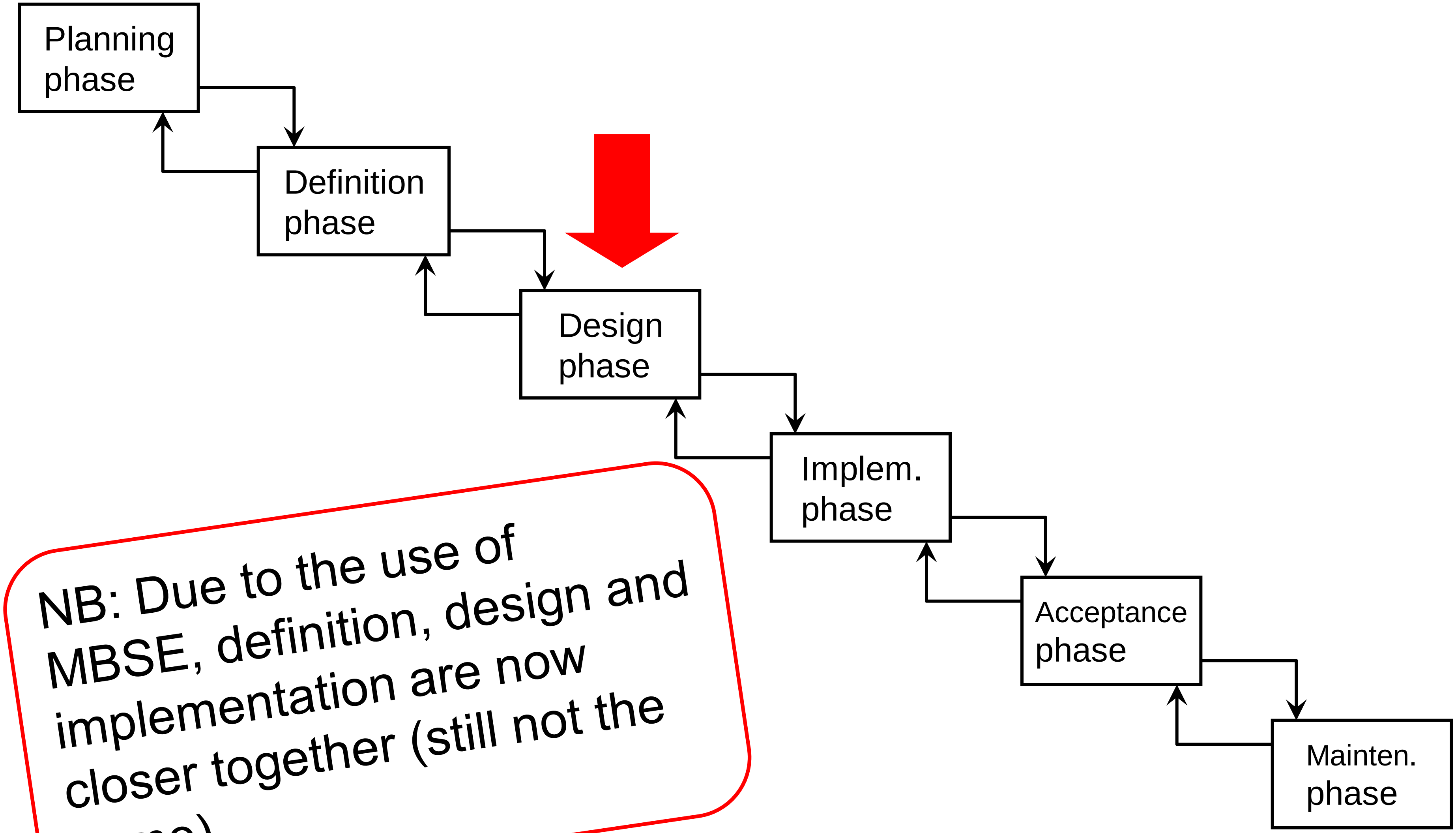
# 3. Software Specification

**DTU Informatics**

Department of Informatics and Mathematical Modeling



# Design Phase



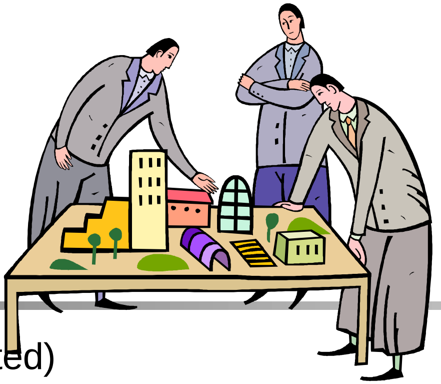
NB: Due to the use of MBSE, definition, design and implementation are now closer together (still not the same).

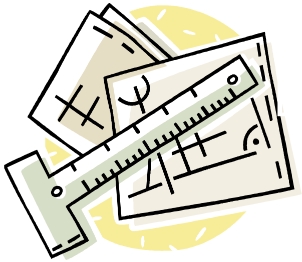
# Specifying Software

Recapitulation
(→ p. 3)

Goals:

- Defining what the software should do (before it is really there)
- **C**ustomer and **d**eveloper agree on what should be delivered
- Effort (resources and time) can be planned based on that (contract will be based on that).





# Specifying Software

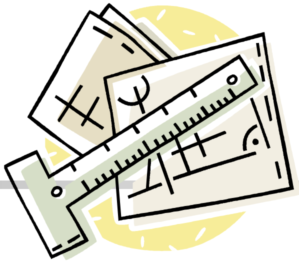
Recapitulation
(→ p. 4-6)

- Project Idea
- Requirements Specification
    - rough
    - detailed
- Systems specification
- Complete Models
- Implementation, Documentation Handbook

what

↓

how

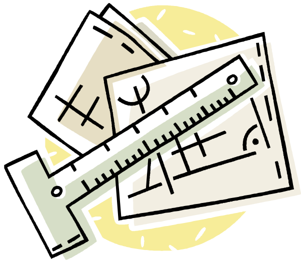


# Now: Software Specification

**Goals**:

- Defining **how** the software should be technically realized
- In such detail that the implementation is “details only”

C-requirements



**D-requirements**

# Main issues

"programming in the large"

- Software architecture / implementation architecture
- auxiliary systems and infrastructure persistent storage of data (→DB and DAL)
- GUI

With EMF, much of the auxiliary structure comes for free. As do a simple form of "persistence" (e.g. XMI serialisation) and some parts of the GUI.

- and the relation between them (and the domain model).

# Architecture

- Software architecture:
  - Main components and sub-components of the system
  - Interfaces (provided and required) of the components
- Implementation architecture:
  - Software architecture +
  - Platform, technology, and language specific details

# Architecture

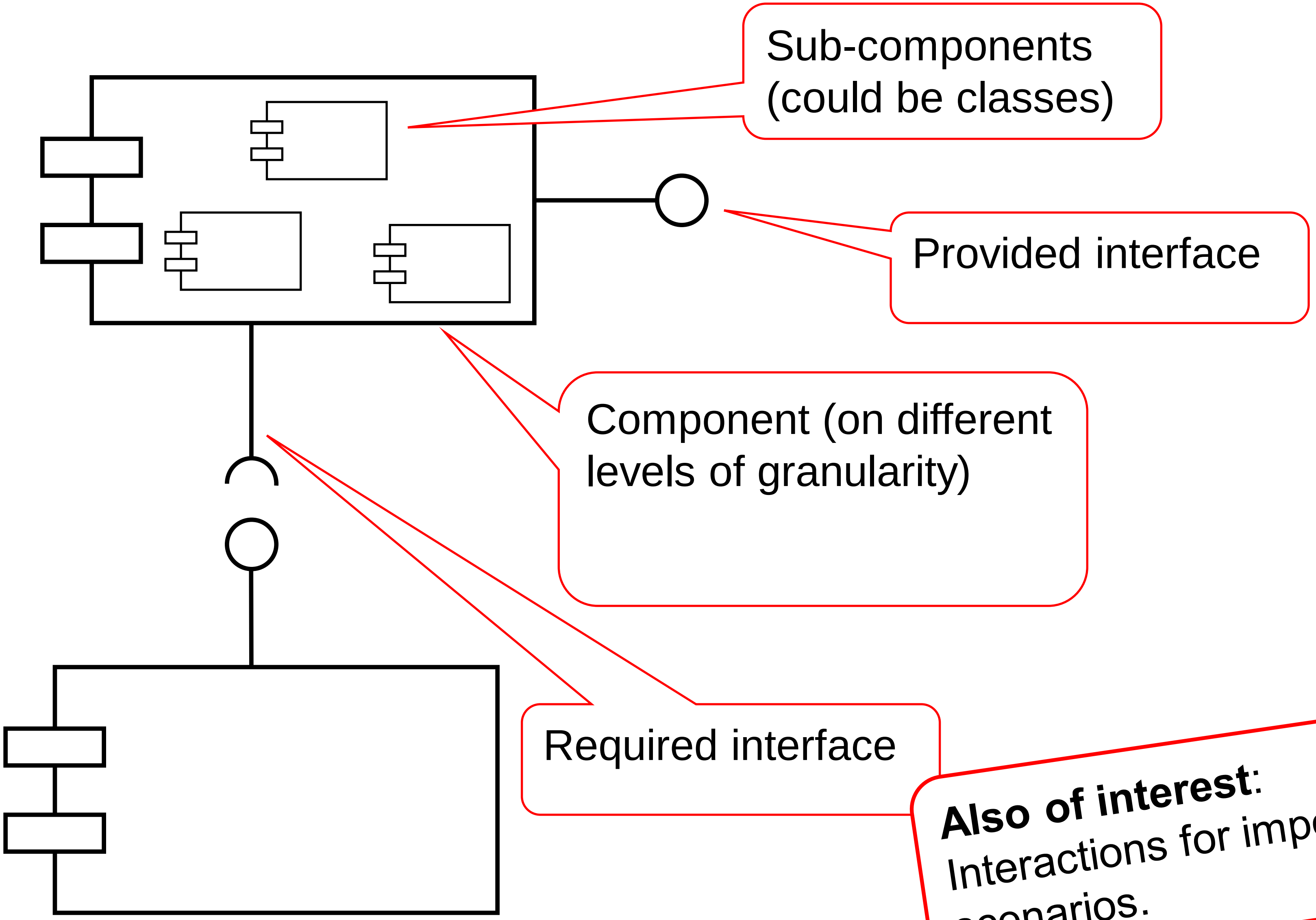
Use cases (refined) + activity diagrams should also be contained in the systems specification.

- Notations:
  - Component diagrams
  - Class diagrams (refined)
  - Design patterns & their terminology
  - Sequence diagrams + state machines

Behaviour at interfaces

Behaviour of important components or classes

# Reminder: Component Diagrams



Sub-components (could be classes)

Provided interface

Component (on different levels of granularity)

Required interface

**Also of interest**: Interactions for important scenarios.

# Design principles

- Clearly identified functionality
- Simplicity of interfaces
- Loose coupling between different components
- Performance / efficiency

# Model refinements

- Naming conventions
- Directions of associations
- Relaxed cardinalities
- Proper containments (→ serialization)
- Visibilities of attributes and references
- “Characteristics” (→ EMF generation)
- Auxiliary attributes, classes, and associations (in EMF often generated automatically)
- DB Schema

# Excursion / Discussion

(OO) Analysis vs. (OO) Design

# GUI

Screenshots (or mock-up screenshots) help writing a readable text on the functionality from a user point of view.

- Sketch GUI visually
- Associate GUI elements with model elements
- Discuss main use cases in terms of GUI (hand book)

# Req Spec vs. Swt Spec

1. Objectives
2. Product use
3. Product functions
4. Product characteristics (non-functional req.)
   - Platform
   - Performance
   - Security
   - …
5. …
6. Glossary

**Systems spec =**
**Requirements Spec +**

- Database Schema
- GUI
  (more detailed → Handbook)
- Architecture
- Refined models (from technical perspective)