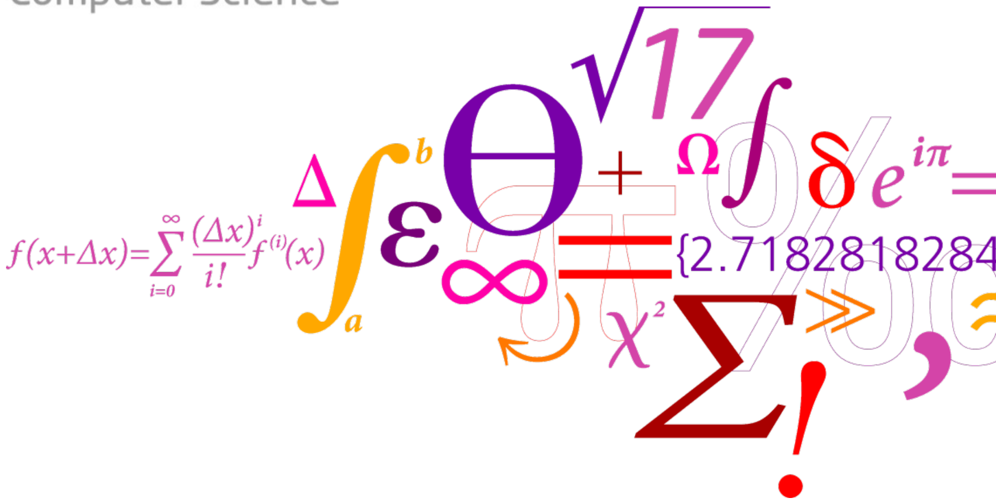
Software Engineering 2
A practical course in software engineering

Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science

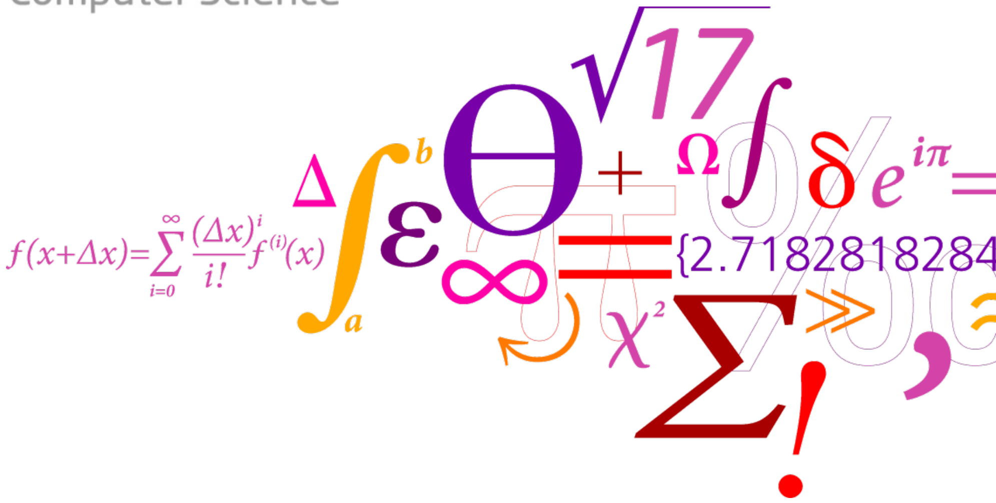


VI. Specifying Software

In agile, writing and documentation is not so much in the focus!

But, documentation is part of our releases (and the final submission)!
→ Learning objectives of this course

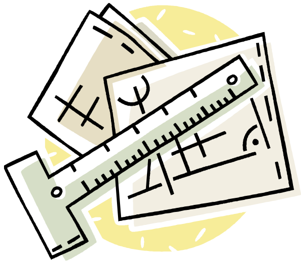
DTU Compute
Department of Applied Mathematics and Computer Science



Specifying Software

Goals (of “software documents”):

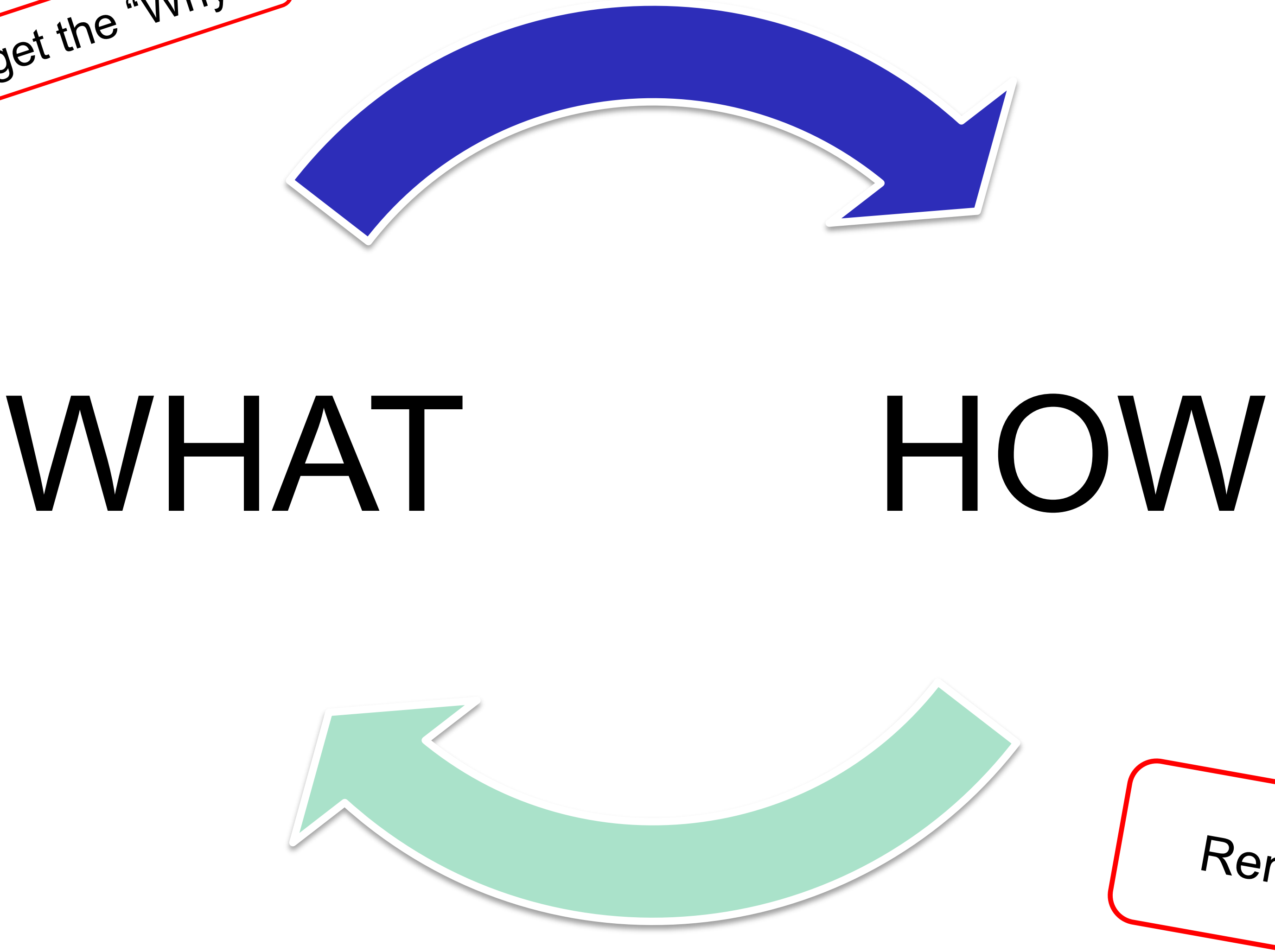
- Defining what the software should do (before it is really there)
- **C**ustomer and **D**eveloper agree on what should be delivered
- Effort (resources and time) can be planned based on that (contract will be based on that).



NB: Writing them is part of the process of understanding what the software should do!

Why, What and How

Don’t forget the “Why”!



Reminder

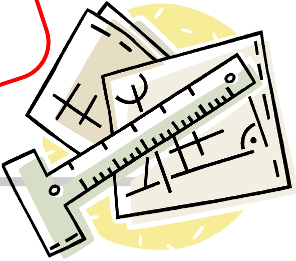
Specifying Software

what

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

Might concern both ”what” or ”how”

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



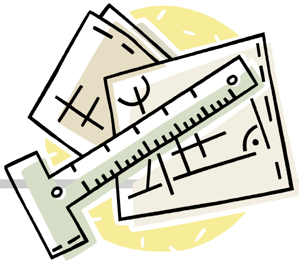
how

Specifying Software

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

rough

↓



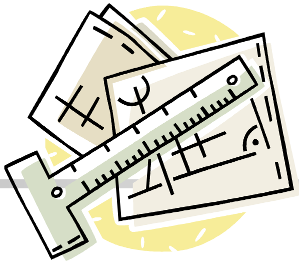
detailed

Specifying Software

low cost

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

Maintainability



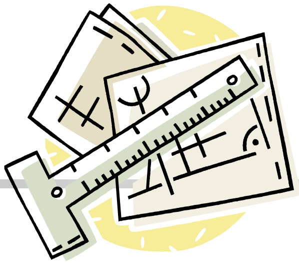
high cost

Specifying Software

informal

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

formal



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).

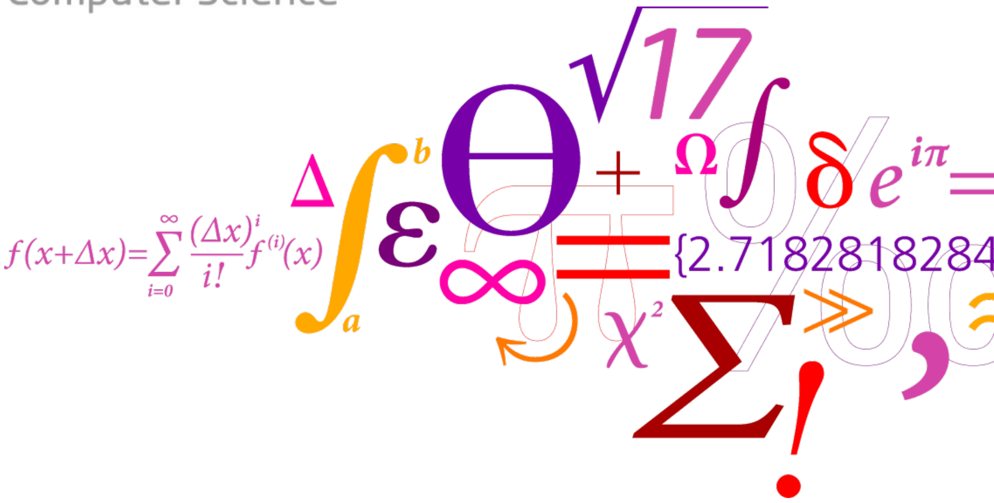
On which kind of document will (can) the cost calculation and the contract be based?

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

1. Project Definition

DTU Compute

Department of Applied Mathematics and Computer Science



Specifying Software

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

Project Definition: Contents

why, what

rough

Readable without any other documents.

But enough details to get an idea of the full picture.

- Partners
- Context
- Objective
- Scope
  (in particular, what is NOT to be done)
- **Use of product** (from the end-user’s point of view)
  - Examples (from this derive)
  - Users
  - Use cases (as text, not necessarily as diagrams)
  - Main data (in our case ”indoor climate domain”)
- Platform (HW/SW)
- Glossary of main terms

What do we have already. What are the extensions.

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)
- **Use of product** (from the end-user’s point of view)
  - Examples (from this derive:)
  - Users
  - Use cases (as text, not necessarily as diagrams)
  - Main data
- Platform (HW/SW)
- Glossary of main terms

→ inductive vs deductive writing!

Project Definition: Contents

what

rough

- Partners
- Context
- Objective
- Scope
  (in particular, what is NOT to be done)
- **Use of product** (from the end-user's point of view)
  - Examples (from this derive:)
  - Users
  - Use cases (as text, not necessarily as diagrams)
  - Main data
- Platform (HW/SW)
- Glossary of main terms

Example: Get a feeling/idea of things. Then sum up formally.

Project Definition: Contents

- Partners
- Context
- Objective
- Scope
  (in particular, what is NOT to be done)
- **Use of product** (from the end-user's point of view)
  - Examples (from this derive:)
  - Users
  - Use cases (as text, not necessarily as diagrams)
  - Main data
- Platform (HW/SW)
- Glossary of main terms

why, what

rough

Scope might be more reasonable after "Use of product".

Glossary: Make it a working document!

Project Definition: Contents

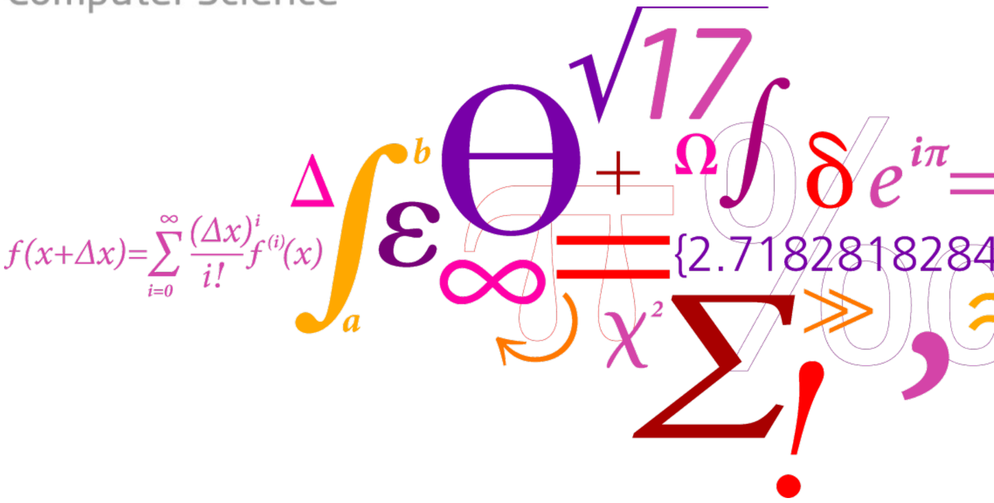
- Project definition could contain project plan
  - External milestones (delivered to customer)
  - Final deliverables

- Since these are fixed for this project and we work agile, this was not required for the project vision in this course.

2. Requirements Specification

DTU Compute

Department of Applied Mathematics and Computer Science



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”

Examples!

- In which quality?

- On which platform or technology?

“how”

Basic outline

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 definition, requirements specification, systems specification, final documentation.

Objective: purpose

- Why is the software developed?
- What should be achieved by using this software?
(requires to set the context)

Frequent **mistake (!)**: "The goal of this project is to develop software!"

Objective: structure

Note the difference:
Purpose of document /
purpose of product

- Purpose of this document
- Context & main (!) terms
- Objectives of this product
- Overview of this document

Should not be too long
(in project: less than a page)

Product use: purpose

- **Basic** understanding of how the product is used!

Not: how it is implemented!

Product use: structure

- Main concepts
- Types of users
- Usage scenarios
- Domain model
  (no design/implementation details)
- Main tasks
- Platform & Interfaces

“Glossary as a class diagram”!
Only concepts from domain!

Hint: Don’t be too detailed (see product functions use cases)

Excursion / Discussion

(OO) Analysis vs. (OO) Design

When using code generation from models, domain models tend to contain some aspects of design already!

Product functions: purpose

- Understanding of all functions of the product (as seen by the end user)

Product functions: structure

- Use cases + use case diagrams
- Example dialogs (GUI)
- Outline of steps for every use case
- Exceptions
- Variations

Dependent on level of detail:
Could contain "screenshots".

Product charact.: purpose

- How will the software run?
- In which environment?

Product charact.: structure

- Usability
- Platform
- Standards
- Performance
- Maintenance / portability
- Security

Frequent mistake: “Empty phrases”; characteristics that are not provable/checkable

Other information

- Used development methods / notations
- Used tools
- Used programming language

Glossary

Now it should be complete!

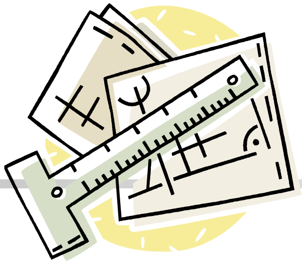
- List of all important concepts / terms of the problem domain along with a brief explanation

Frequent mistakes:

- Glossary only created in the end! → should be a living document (e.g. wiki)
- Mixing meta-terms with domain-terms

Specifying Software

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



Differences

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

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

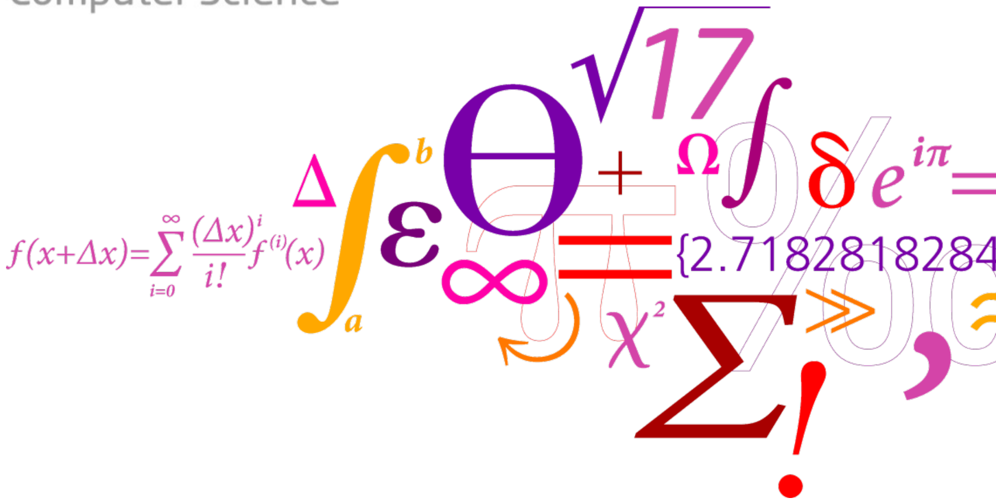
The exact definition of different specification types varies: structure, level of detail, models, …

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

3. Software Specification

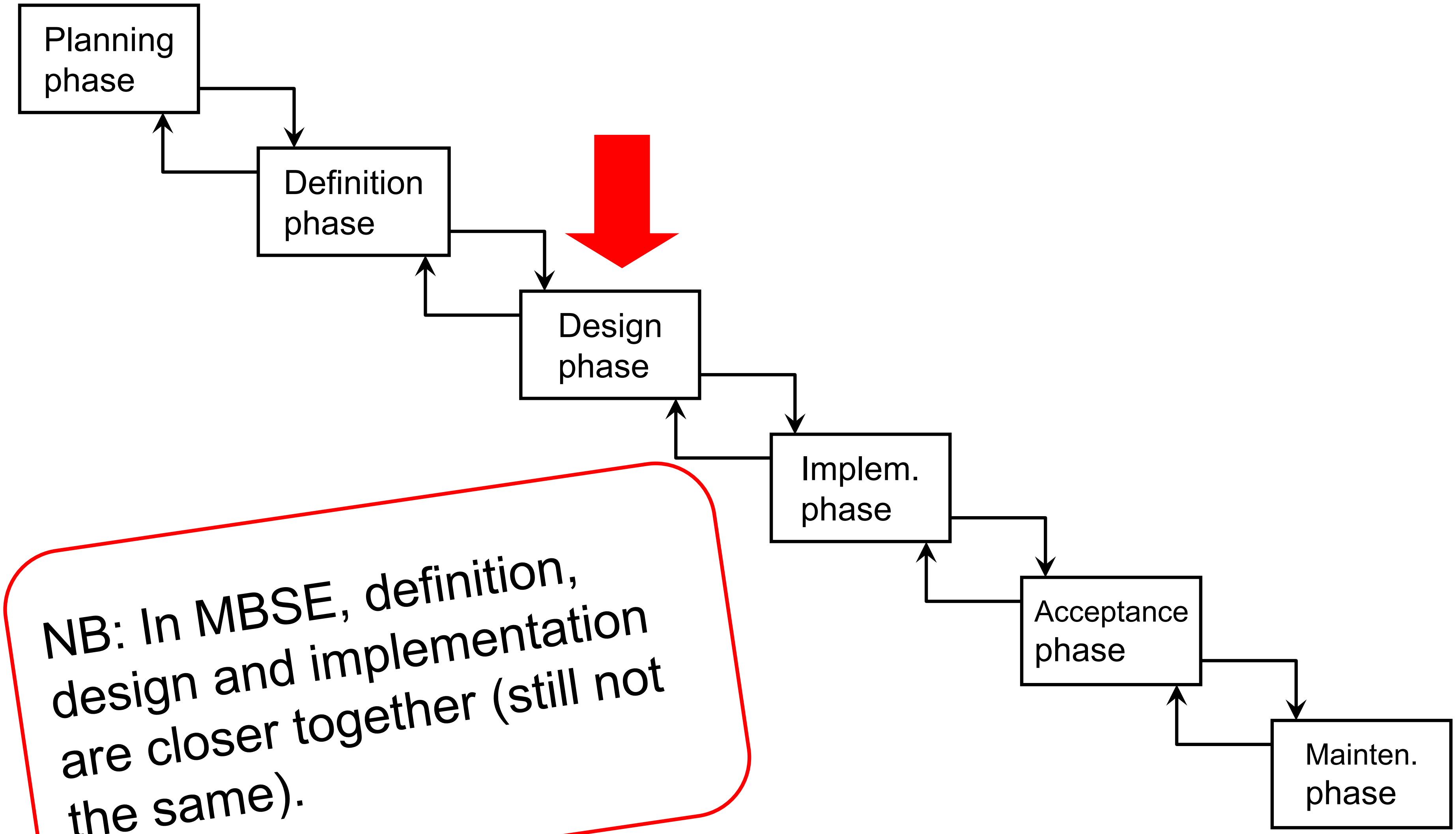
DTU Compute

Department of Applied Mathematics and Computer Science



$f(x+\Delta x)=\sum_{i=0}^{\infty}\frac{(\Delta x)^i}{i!}f^{(i)}(x)$
$\sqrt{17}$
$e^{i\pi}$
{2.7182818284
χ^2

Design Phase



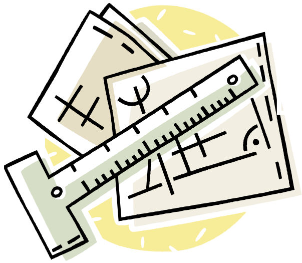
NB: In MBSE, definition, design and implementation are closer together (still not the same).

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).



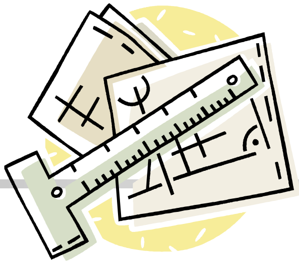
Specifying Software

Recapitulation
(→ p. 5-8)

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

what

how

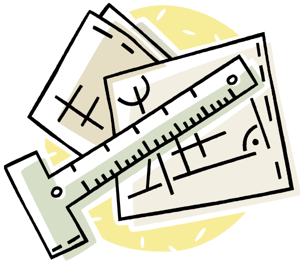


Now: Software Specification

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

With MBSE technologies, much of the auxiliary structure comes for free (or added by tagging or annotating models). As does a simple form of "persistence" (e.g. XMI serialisation) and some parts of the GUI.

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

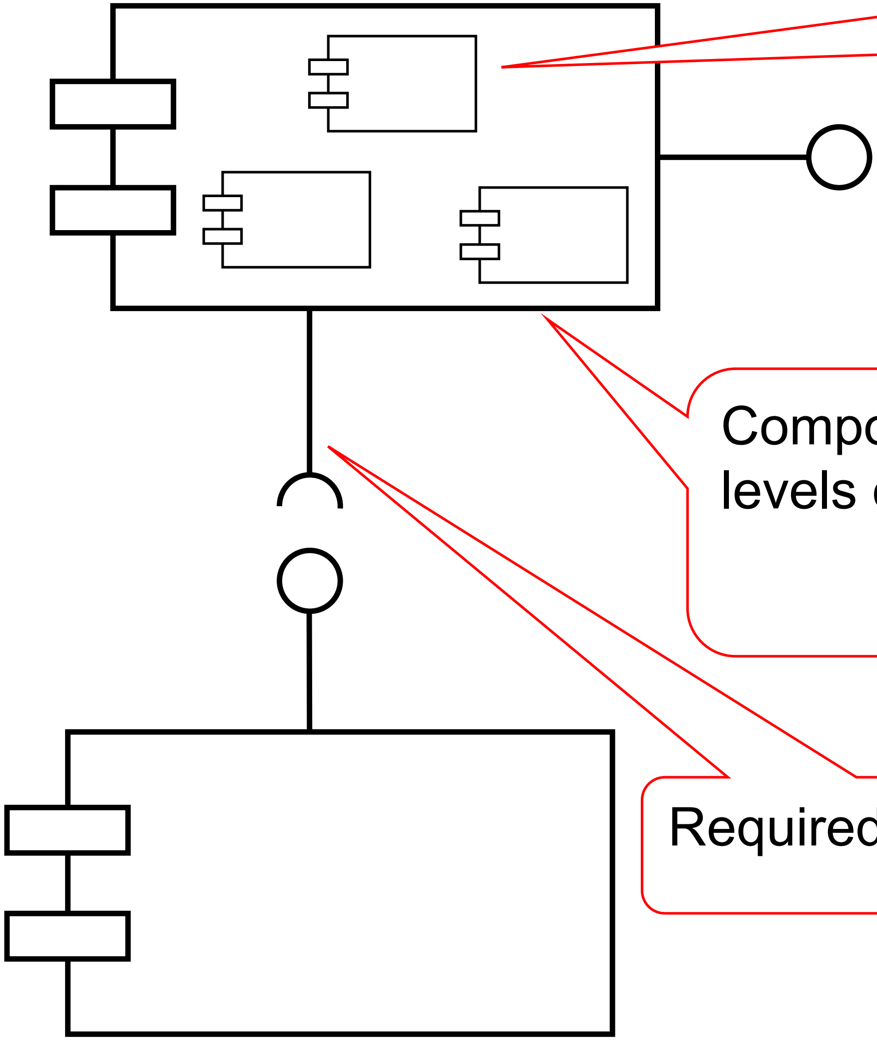
NB: Use cases (refined) + activity diagrams should also be contained in the systems specification (but not under "Architecture".

- Notations:
  - Component diagrams
  - Class diagrams
    (refined domain model + software models)
  - 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
- Auxiliary attributes, classes, and associations (in EMF often generated automatically)
- DB Schema

Excursion / Discussion

(OO) Analysis vs. (OO) Design

Conceptualize domain (example models)

Conceptualize (make abstract view of) software: Model + View + Controllers, Design Patterns

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

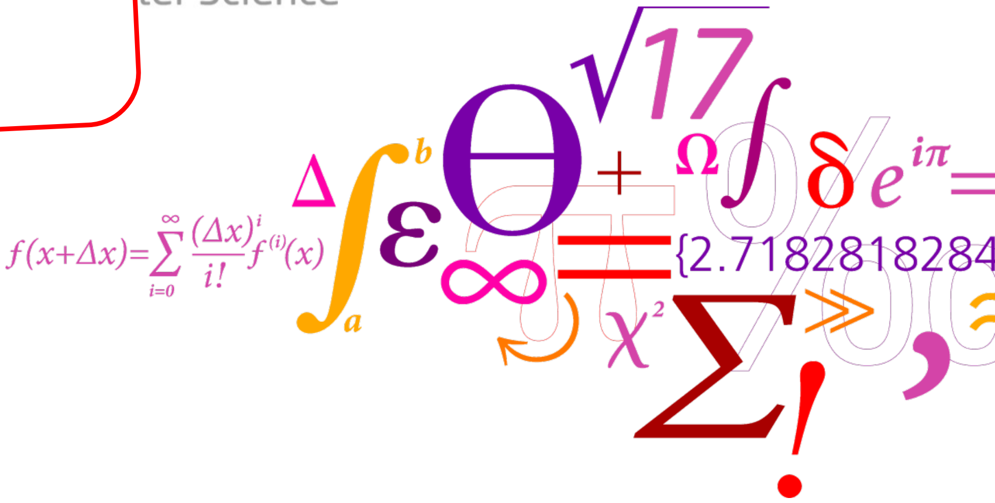
Systems spec =
Requirements Spec +

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

4. On Writing Well

Headline "borrowed" from the book William Zinsser: On Writing Well (ed. from 1976 - 1998)

ter Science



Motivation

- Writing good texts is hard work!

- Most of it can learned and is more on about the writer’s attitude than on talent:
  - What is the purpose?
  - What do I want to achieve?
  - Who is the reader?
  - How do I achieve my goals?

Motivation

Problems

- The readers can’t ask the writer
- The writer must foresee possible questions and misunderstandings
  (and take care of them)

- The writer should not assume too much
- The writer should not make implicit assumptions or conclusions

Rule of thumb:
Don’t assume anything. But, don’t tell the reader that he/she is stupid.

Comprehensibility

When is a text comprehensibility?

Are there criteria for comprehensibility?

Langer, Schulz von Thun, Tausch:
„Sich verständlich ausdrücken!“

Criteria

- Simplicity (-- - 0 + ++)
  - simple words
  - simple sentences
  - short sentences
  - concrete (e.g. by example)

→ Inductive vs. deductive

- Structuring (-- - 0 + ++)
  - one idea after the other
  - form and content are coherent
  - conclusive

Criteria

- Conciseness (-- - 0 + ++)
  - shortness
  - focussed on essentials
  - no empty words and sentences

- Inspiring Additions (-- - 0 + ++)
  - motivating
  - interesting
  - diversified

Important issues

- Set the scene / context:
  Don’t assume anything (except readers’ pragmatics) for granted
- Different levels of abstraction:
  Typical student mistake: always on the lowest level!!
- Guide the reader:
  Why do you say what you are saying
- Bring the point (argument) home – **completely**!

- ”**Spiralform writing**”: Writing linearly about a complex network of concepts

Black board discussion

More rules (of thumb)

- important stuff first / high-lighted
- strong verbs (avoid adjectives / adverbs)
- short sentences
- use singular whenever possible
- familiar terms and expressions
- use “active” wherever possible
- clear headlines
- …

Be critical about your own texts!

Comprehensibility

- The above criteria hold for almost all texts
- For scientific texts:
  - consistent terminology (same term for same concept throughout the text):

    My favourite **counter example** „Deutscher Fußballreporter“:
    Ball, Rund, Kulle, Leder, Ding, …
  - Same structure for alike structured content