Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

Designing Software Architectures Part 1/2

Dr. Martin Girschick

Software Engineering in Industrial Practice

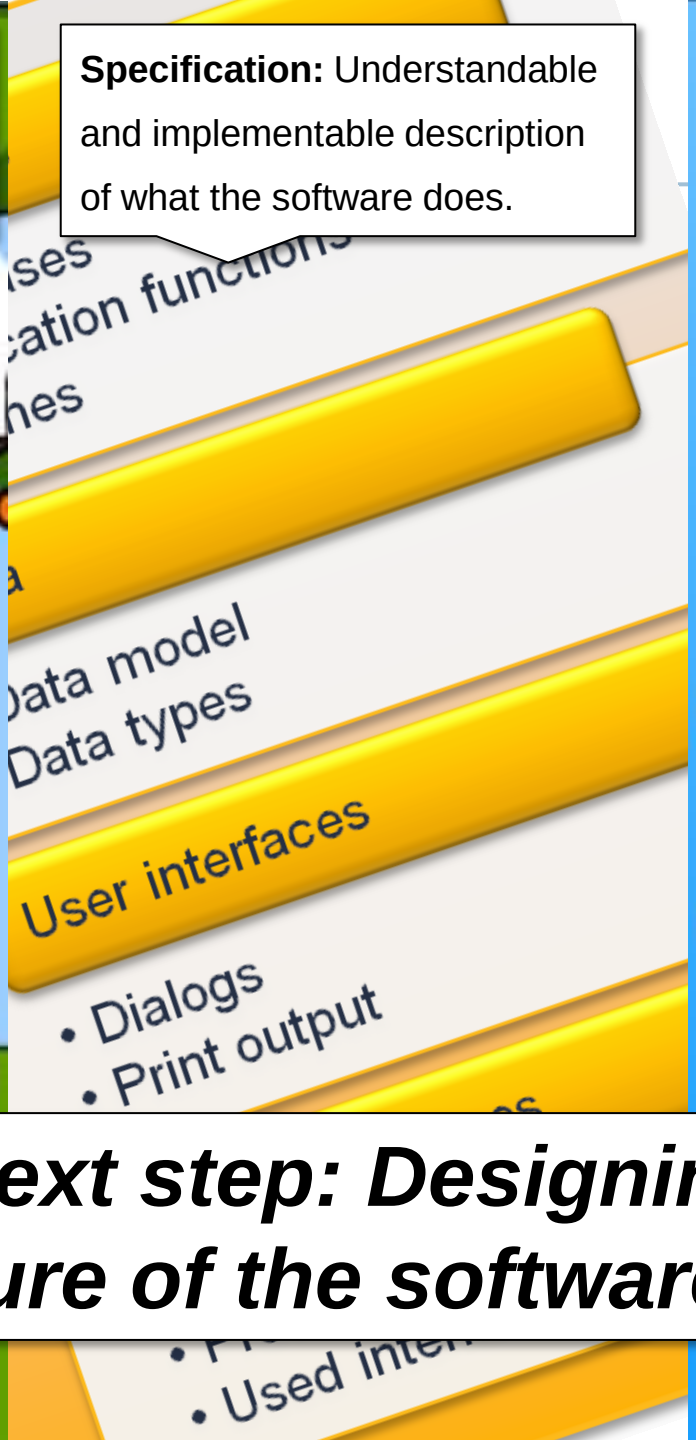
WS2015/2016

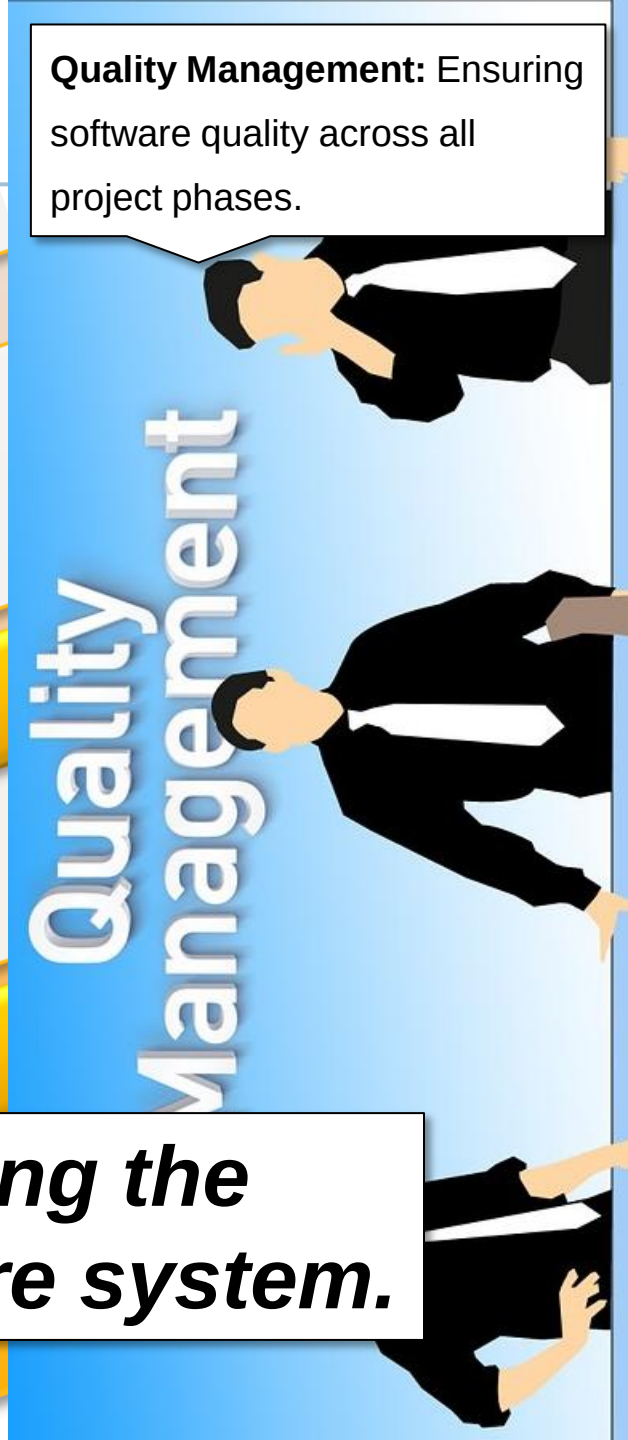
TOGETHER. FREE YOUR ENERGIES.

Requirements analysis: A testable, coherent description of the problem to be solved.

Specification: Understandable and implementable description of what the software does.

Quality Management: Ensuring software quality across all project phases.





The next step: Designing the architecture of the software system.

AGENDA

- What is the purpose of software design?
- How to get from specification to construction?
- How do you produce good designs?
- The architectural principles
- Thinking in terms of components and interfaces
- Architectural views and layers
- Design best practices

AGENDA

- **What is the purpose of software design?**
- How to get from specification to construction?
- How do you produce good designs?
- The architectural principles
- Thinking in terms of components and interfaces
- Architectural views and layers
- Design best practices

The purpose of the design is …

…to master the complexity of a software system

- To break a problem down into manageable units
- To separate concerns, which is the most important guiding principle
- To consider the system from different points of view

…to make it plannable and realizable

- To make it possible to plan the system and verify its feasibility
- To facilitate cost-effective, parallel development
- To make the system comprehensible and communicable
- To ensure maintainability and extensibility

…to meet the requirements

The result is the software architecture, the "construction plan" for software development

- Defines how an information system is **organized and structured**
  - The static aspects
  - The dynamic aspects
  - how the requirements are matched to the architecture
- Determines to a large extent the system's **quality characteristics**
  - The adherence to non-functional requirements
  - The reusability of system components

Good design documents are…

- **Complete** and **suitable** – providing answers at the right time
- **Not an end in themselves** – what doesn't get read doesn't have to be written
- **Easy to understand** for those they are written for, as concise as possible
- **Based** on the requirements
- **Practical** in terms of the development resources available (expertise, components, tools)
- **Consistent** and **up-to-date**

Example

Who is going to read the design document for what reason?

Discuss and post your ideas on twitter under the hashtag #seiipTUD

AGENDA

- What is the purpose of software design?
- **How to get from specification to construction?**
- How do you produce good designs?
- The architectural principles
- Thinking in terms of components and interfaces
- Architectural views and layers
- Design best practices

In the specification phase we define *what* we are going to build. In the construction phase we define *how* we are going to do it.

Specification:
external view of the system

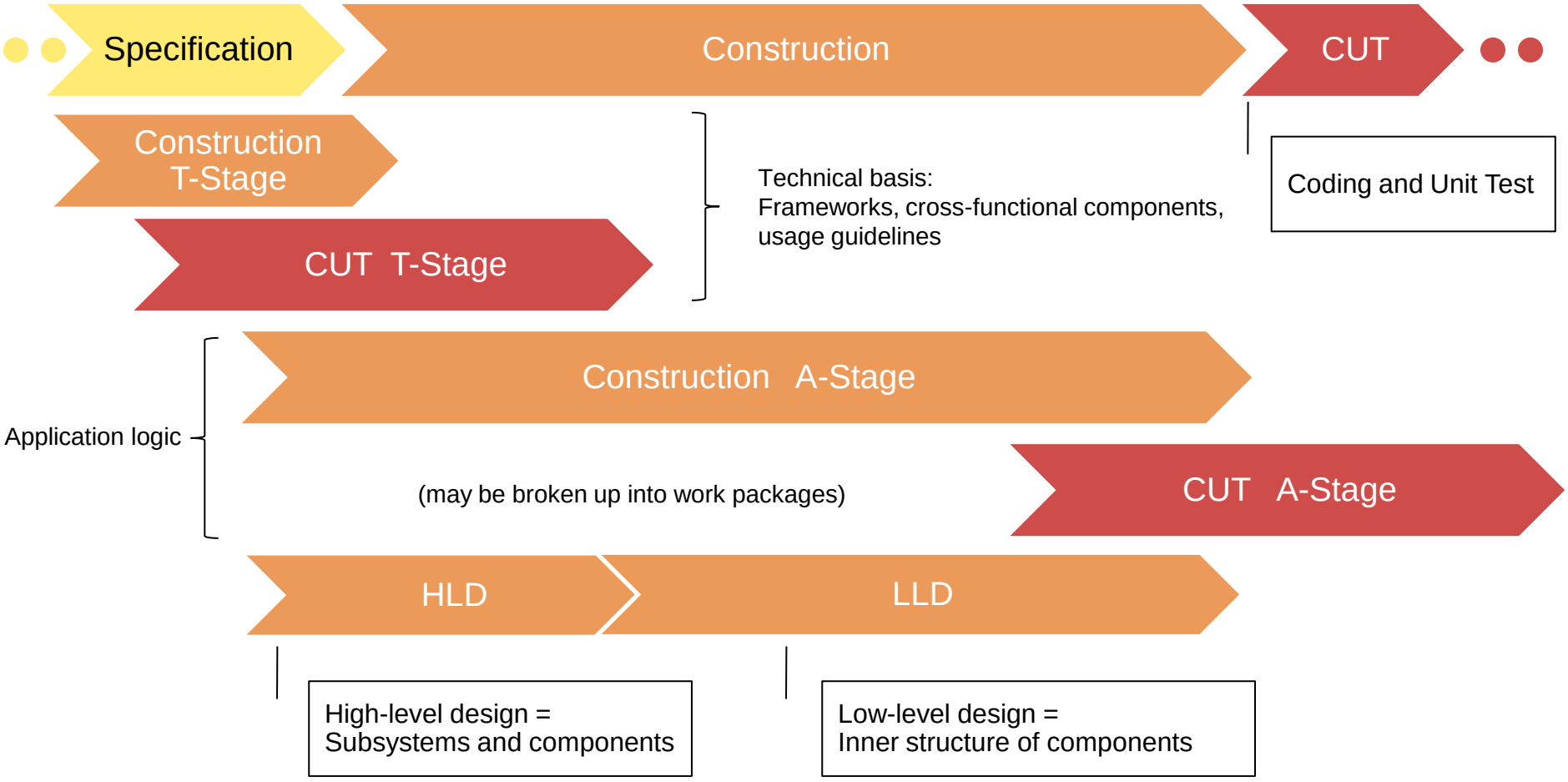
- Functional and non-functional requirements
- **Target groups**: users, management, architects, developers
- **Models**: business use cases, dialog designs, logical data model, etc.

Construction:
internal view of the system

- Logical and technical structure
- **Target groups**: developers, architects, operations staff, (management, etc.)
- **Models**: architecture views, component, class, sequence and ER diagrams, etc.

In large-scale projects, we usually follow a modified waterfall model.

Typical plan of a large-scale project



In large-scale projects, we usually follow a modified waterfall model.

Typical plan of a large-scale project

HANDOUT

- The typical large-scale project involves some kind of waterfall layout:
  - After writing the specification, and before entering implementation (CUT), we construct the system in a dedicated construction phase.
- In truth, this looks much more intertwined:
  - We start with construction and implementation of the technical basis already before the specification is finished. We call this the T-Stage.
  - In order to use time efficiently, construction and implementation of the business logic will also start before the specification is finished. We call this the A-Stage.
  - As soon as the required input for the next stage of a work package is stable enough, work of the next stage can begin. The “waterfall” gets split up into many rivulets that are heavily overlapping in the time frame.
- The construction of the A-Stage itself can be divided into two separate activities:
  - The high-level design (HLD), in which we divide the system in to components.
  - And the low-level design (LLD), which per component describes how it is set up from parts.
- Parallelism can be increased for LLD and CUT of the A-components. HLD should be done as a whole.

Notice: the best projects are those, where the T-Stage is already finished before we start (i.e. can be re-used).

AGENDA

- What is the purpose of software design?
- How to get from specification to construction?
- **How do you produce good designs?**
  - The architectural principles
  - Thinking in terms of components and interfaces
  - Architectural views and layers
  - Design best practices

We master the complexity of system design by not taking care about everything at once

The architectural principles

Refinement stages

- **Application landscape**
- **System design**
  - Layers and stages
  - Components
  - Separation of business logic from technical aspects
- **Component design**
  - Entity classes
  - Determination of the interfaces
  - Use of design patterns
- **Class and method design**

External / coarse-grained

Internal / fine-grained

Architectural views

Application architecture
business components

Technical architecture
“virtual machine” for the application

Technical infrastructure
hardware, platform, network, etc.

AGENDA

- What is the purpose of software design?
- How to get from specification to construction?
- How do you produce good designs?
- **The architectural principles**
- Thinking in terms of components and interfaces
- Architectural views and layers
- Design best practices

An architectural principle is a generally valid, elementary guideline that can be followed on all levels of the software design process.

Definition architectural principle

Architectural principle

Guideline

- Easy to comprehend
- Instruction to be followed in the design
- Based on accumulated experience

Generally valid

- Independent of technology, business logic, and project size

Scale-invariant

- Applicable on all levels of software construction (application landscape, application, component internal structure,…)

Altogether, the architectural principles must be applied with sense and judgment. Exaggeration of single principles will not lead to well-conceived architectures.

An architectural principle is a generally valid, elementary guideline that can be followed on all levels of the software design process.

Definition architectural principle

HANDOUT

An architectural principle is a guideline for the design of an application landscape, an application or a component – it is a generally valid, elementary procedural instruction that is applicable at every level of software development.

A single architectural principle is not enough. A good architecture can only be created by applying all the principles sensibly and appropriately. In contrast, the exaggerated use of a single principle will not result in a well-conceived architecture.

We identify 10 architectural principles that support our design objectives.

Architectural principles

Elementary principles	
SoC	Separation of concerns
MoD	Minimization of dependencies
IH	Information hiding
HG	Homogeneity
ZR	Zero redundancy

We identify 10 architectural principles that support our architectural objectives.

Architectural principles

HANDOUT

Architectural principle 1: **Separation of concerns**

Subdivide the system into software elements: Every software element is intended to perform precisely one task. This task is also only performed by this software element. The task must be defined precisely and clearly and the delineation from other software elements must be clearly identifiable.

Architectural principle 2: **Minimization of dependencies** *(strong cohesion & loose coupling)*

Minimize the dependencies between the software elements, e.g. by combining software elements or decoupling elements. Avoid cyclical dependencies.

Architectural principle 3: **Information hiding**

Encapsulate the internal knowledge in a software element and do not reveal it to other software elements. The users of the software element learn only what is necessary for them or obtain a special view of the functionality in the software element which is used by them.

We identify 10 architectural principles that support our architectural objectives.

Architectural principles

HANDOUT

Architectural principle 4: **Homogeneity**

Use similar solutions to solve similar problems and use similar dimensions and units within one and the same structure level.

Architectural principle 5: **Zero redundancy**

Implement identical technical or logical functionality that is used by different software elements in only one location. Identify and isolate shared functionality in the software elements and generalize this functionality so that it can be re-used easily by other software elements.

Group discussion: What are the benefits when applying the first five principles?

Discuss and post your ideas on twitter under the hashtag #seiipTUD

We identify 10 architectural principles that support our design objectives.

Architectural principles

Elementary principles	
SoC	Separation of concerns
MoD	Minimization of dependencies
IH	Information hiding
HG	Homogeneity
ZR	Zero redundancy

Secondary principles	
SC	Differentiation of software categories
SiL	Subdivide into Layering
DbC	Design-by-contract
DS	Data sovereignty
RU	Re-use

We identify 10 architectural principles that support our architectural objectives.

Architectural principles

HANDOUT

Architectural principle 6: **Differentiation of software categories**

Assign precisely one software category to each software element. Here, we distinguish between the three fundamental software categories:

A software: The software element implements logic requirements and possesses no knowledge of the technical characteristics of the system.

T- software: The software element shapes the technical character of the system. It possesses no knowledge of the logical requirements to be implemented.

0 software: The software component possess no knowledge of either the system's logical requirements or of its technical composition.

Software components which can be assigned multiple software categories should be avoided.

We identify 10 architectural principles that support our architectural objectives.

Architectural principles

HANDOUT

Architectural principle 7: **Layering**

Subdivide the system into layers. Each layer then contains software elements with a similar structure or similar basic functionality. A layer may only call software elements in its own layer or a lower-level layer – never software elements in the higher-level layer.

Architectural principle 8: **Design by contract**

Fully describe the system in terms of a set of interface contracts.

Architectural principle 9: **Data sovereignty**

Make sure that exactly one software element is responsible for each element in the data resources. This software element defines the structure and modeling of a section of the data resources and ensures the integrity and consistency of this section when the data is used.

Architectural principle 10: **Re-use**

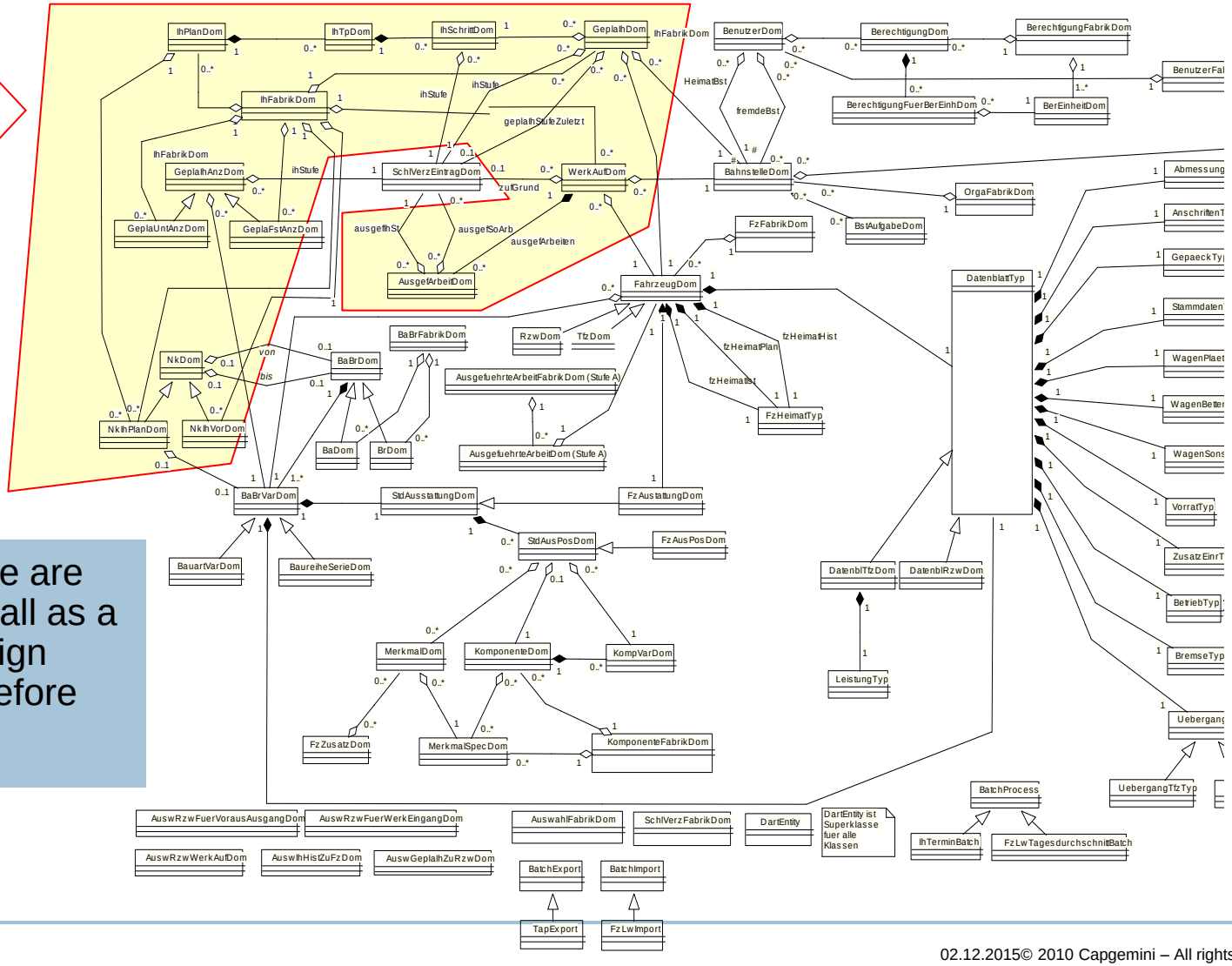
Use existing (standard) solutions, design patterns and reference architectures when implementing the system's functionality. Find reference solutions for the typical logical and technical problems posed by the application and make sure that these are used.

AGENDA

- What is the purpose of software design?
- How to get from specification to construction?
- How do you produce good designs?
  - The architectural principles
  - **Thinking in terms of components and interfaces**
  - Architectural views and layers
  - Design best practices

Example: Classes affected by a change request

Here



Classes alone are much too small as a software design unit and therefore unsuitable!

The idea of the software component

"Software that changes at different rates is subdivided into different modules."

D. L. Parnas:
On the Criteria to be Used in Decomposing Systems into Modules, CACM
1972

*„… **We propose instead that one begins with a** list of difficult design decisions or design decisions which are likely to change**.***
Each module is then designed to** hide such decisions from the others**. …”

Separation of concerns (business logic and technical aspects)

Hiding of design decisions (principle of information hiding)

We need a component definition that supports our architectural principles and an efficient project implementation.

Component – definition objectives

Objectives

Architectural principles

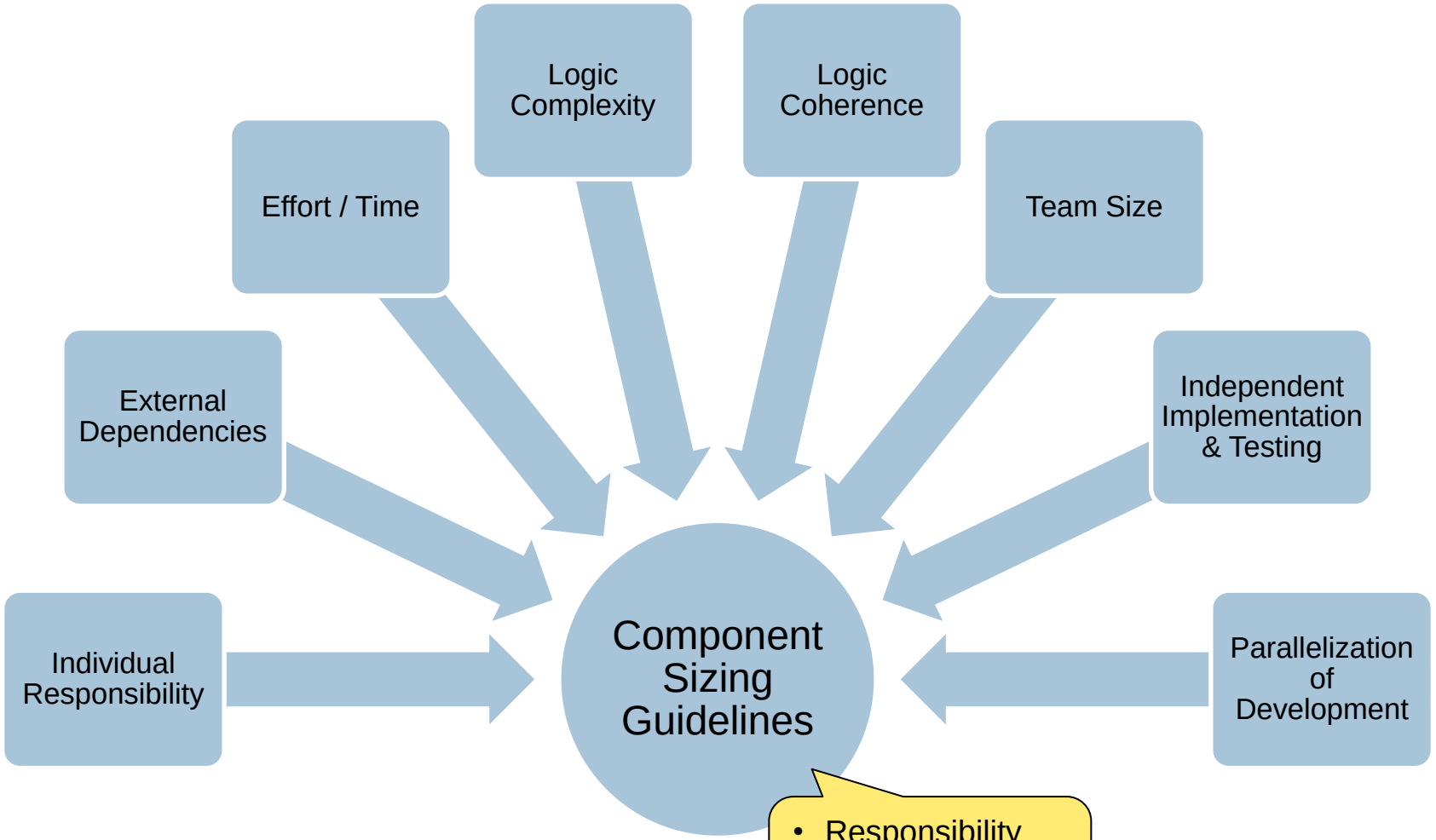
1. Separation of concerns
2. Minimization of dependencies
3. Information hiding
4. Homogeneity
5. Zero redundancy
6. Software categories
7. Layering
8. Design-by-contract
9. Data sovereignty
10. Reuse

Efficient project implementation

- Staffing
- Planning
- Controlling
- Responsibility

The size of the components must be chosen to support effective implementation and maintenance

Component size – determining factors



The size of the components must be chosen to support effective implementation and maintenance

Component granularity

HANDOUT

Component size aims

- Individual responsibility (“my component”)
- Component implementation as a task that can be planned
- Implementation and test not depending on other components
- Parallelization of development

Component size guidelines

Planning aspect:

- No more than 3 developers at any one time
- Per developer: effort not more than 1/3rd of project implementation duration

Complexity aspect:

- For data components: not more than 15-20 entities / component
- If not only very simple logic (CRUD), then rather aim for 5-10 entities / component

Coming up next week

components and interfaces

Architectural views and layers

Design best practices