System Design – Introduction & Initial Steps [01]

Hochschule
für Technik
Stuttgart

Marcus Deininger

WS 2021/22

Overview

- Definitions for Design / Architectures
- Design in the Software Development
- The Design Dilemma
- Initial Design Steps

- Excursus: UML / Enterprise Architect
- Exercise

Definitions (1)

Design

The process of defining the software architecture, components, modules, interfaces, and data for a software system to satisfy specified requirements.

The result of this process.

ISO/IEC/IEEE 24765-2010 –
Standard Glossary of Software Engineering Terminology

Logical Design

An abstract representation of the data flows, inputs and outputs of the system

Physical Design

The actual input and output processes of the system. … [it] can generally be broken down into three sub-tasks: User Interface Design, Data Design and Process Design.

https://en.wikipedia.org/wiki/Systems_design

Definitions (2)

Architecture

Fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution.

ISO/IEC 15288 – Systems and software engineering
– System life cycle processes

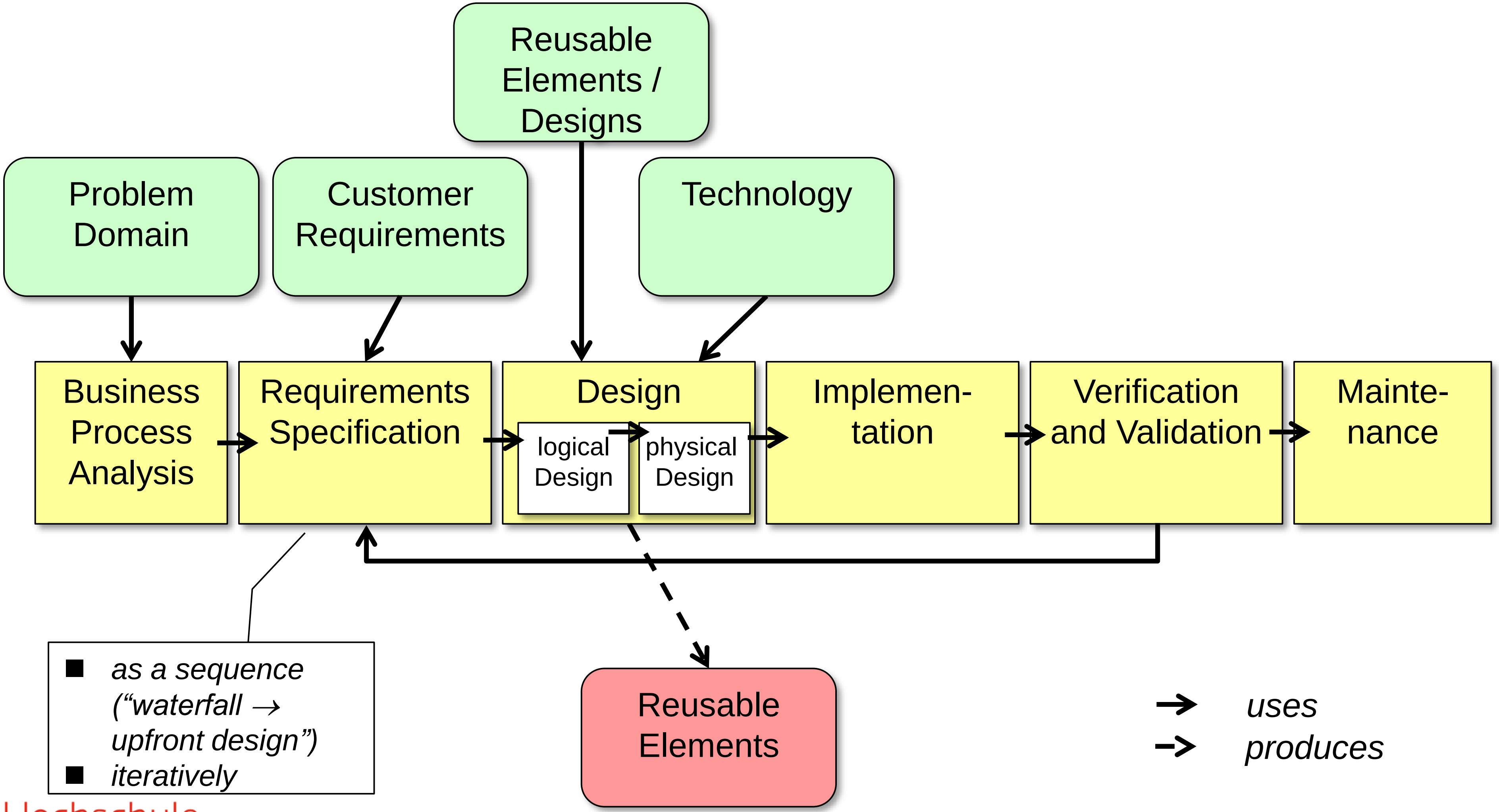
The structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Bass, L., P. Clements, R. Kazmann (2003):
Software Architecture in Practice, 2nd Edition, Addison-Wesley.

Structure

- Structure is the sum of relationships between its entities which stay invariant.
- A software architecture may be defined by several structures (usually called “views”)
- There is no “optimal” (i.e. best) design – only the most satisfying design under the given circumstances (time / budget / personal experience / state of the art)
- A good design fits the needs within the given restrictions

Design in the Software Development



Reusable Elements / Designs

Problem Domain

Customer Requirements

Technology

Business Process Analysis

Requirements Specification

Design

logical Design

physical Design

Implementation

Verification and Validation

Maintenance

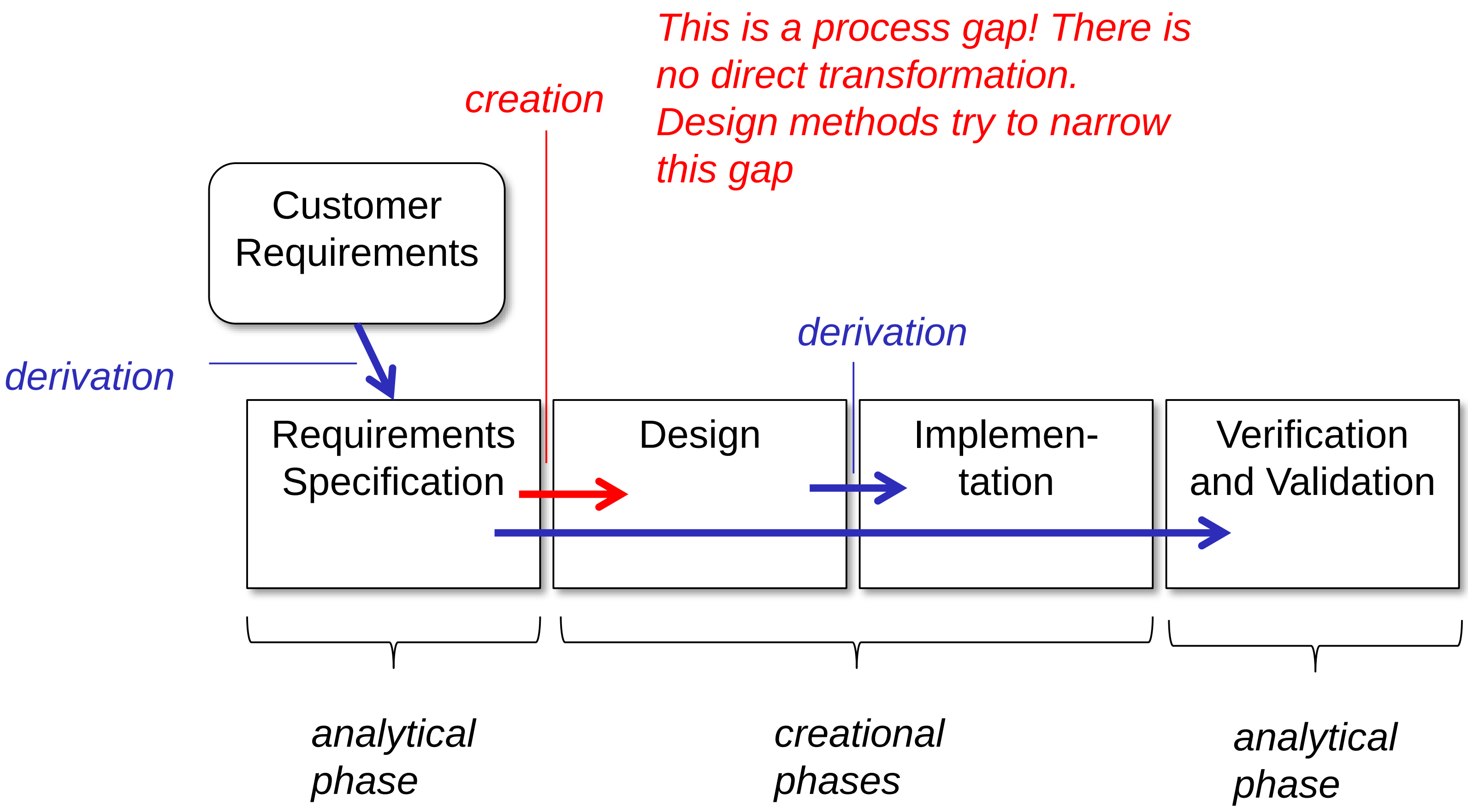
Reusable Elements

- as a sequence ("waterfall → upfront design")
- iteratively

→ uses

--> produces

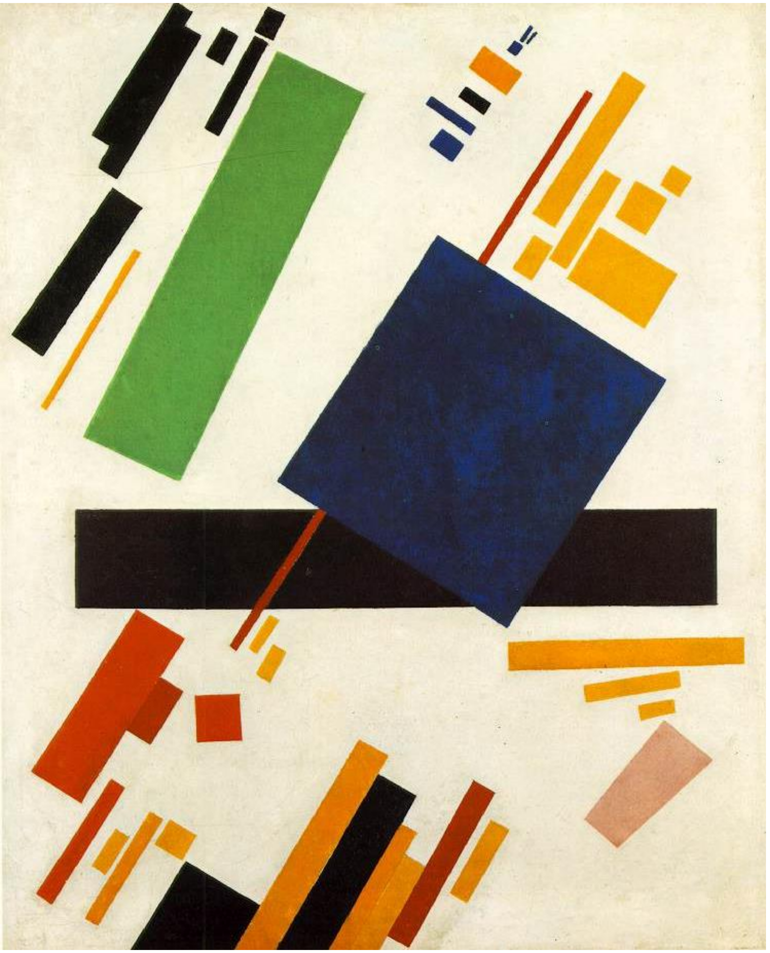
Design is Difficult


This is a process gap! There is no direct transformation. Design methods try to narrow this gap
creation
Customer Requirements
derivation
derivation
Requirements Specification
Design
Implemen-tation
Verification and Validation
analytical phase
creational phases
analytical phase

Hochschule
für Technik
Stuttgart

Design is an Art

- Design is the creative transformation of requirements to a future system structure.
- There is no automated design, only heuristics with supporting principles

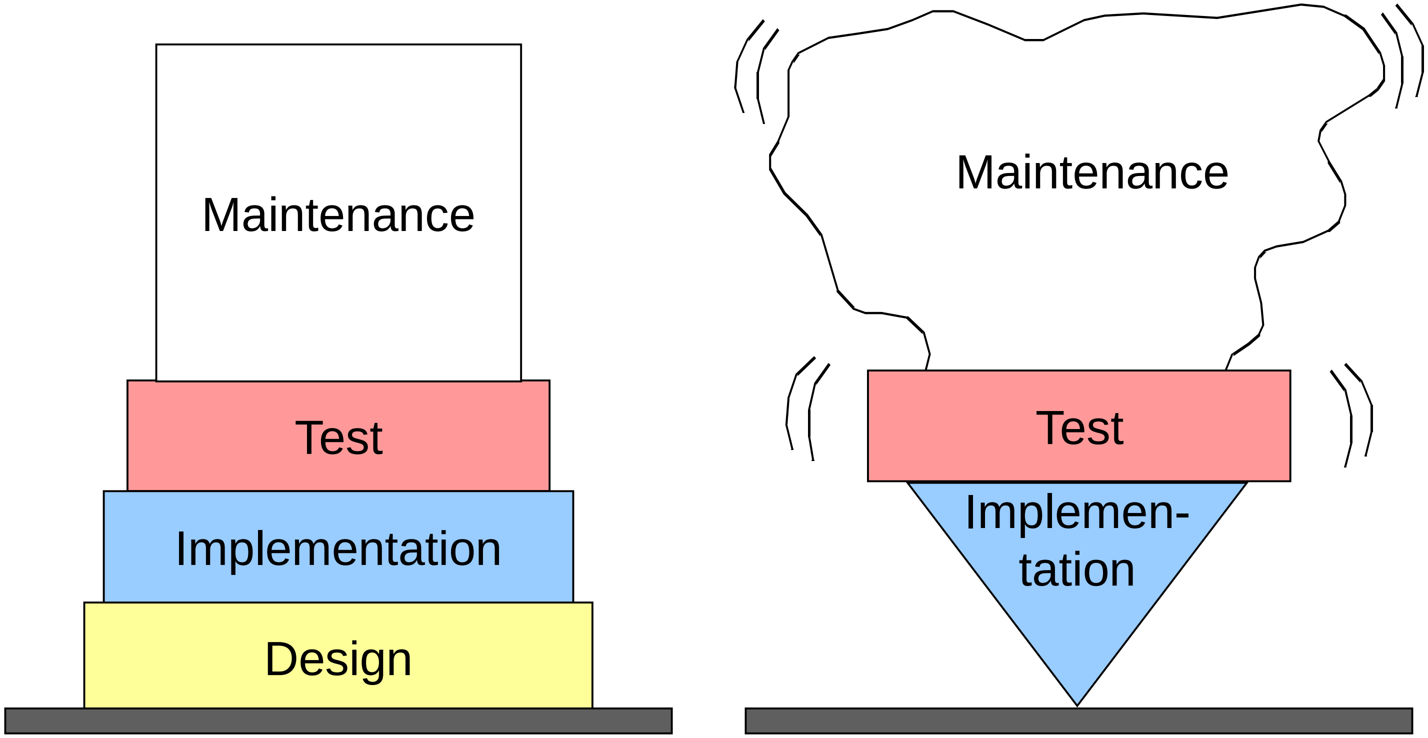


Kasimir Malevich, “Suprematism“
1916-17; Oil on Canvas, 80 x 80 cm;
Fine Arts Museum, Krasnodar

Hochschule
für Technik
Stuttgart

Design is Crucial

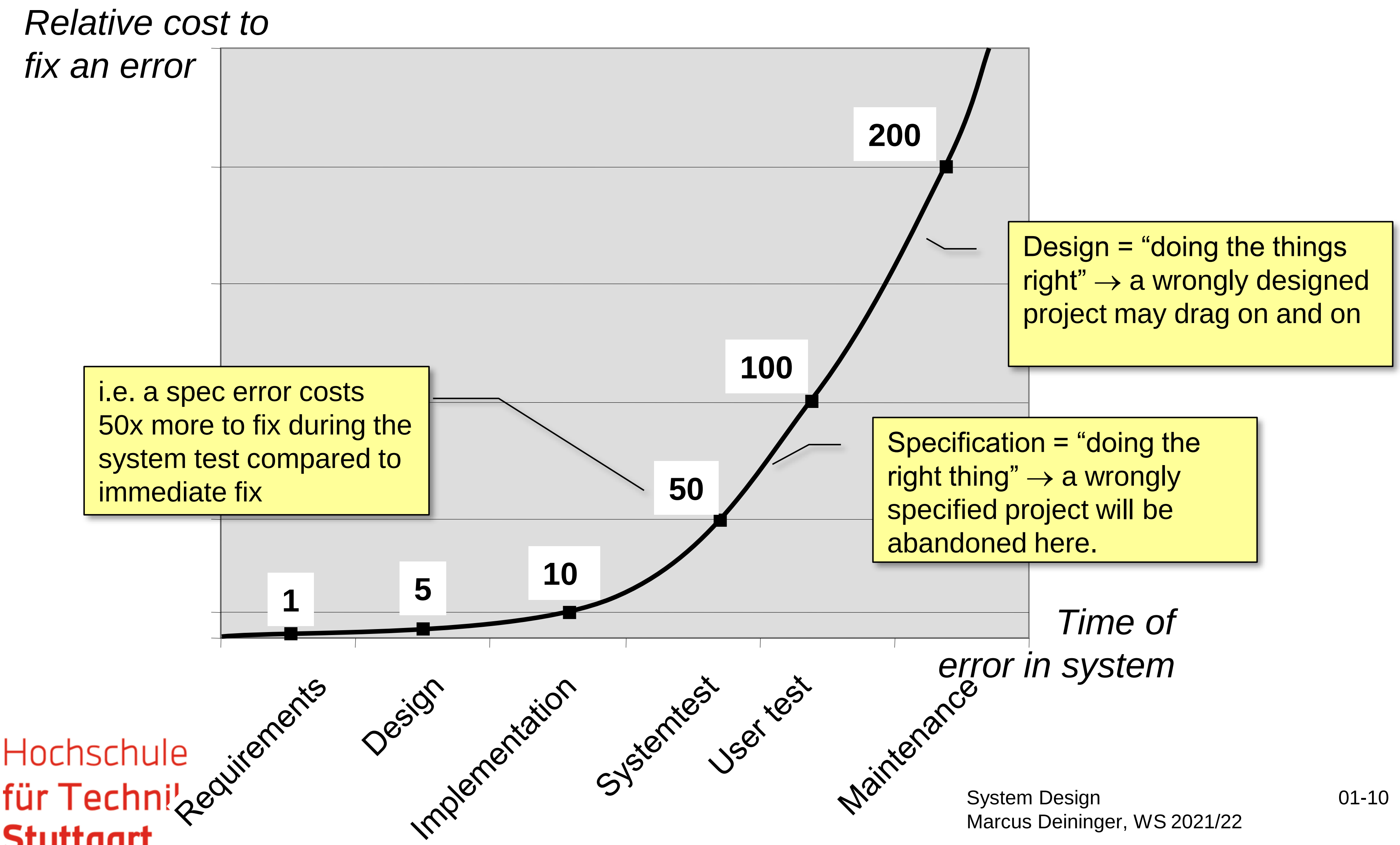
- It lays the foundation of the system ("the founding structure")



with design

without / bad design

The Cost of a Bad Design



i.e. a spec error costs 50x more to fix during the system test compared to immediate fix

Design = “doing the things right” → a wrongly designed project may drag on and on

Specification = “doing the right thing” → a wrongly specified project will be abandoned here.

The Design Dilemma – 1

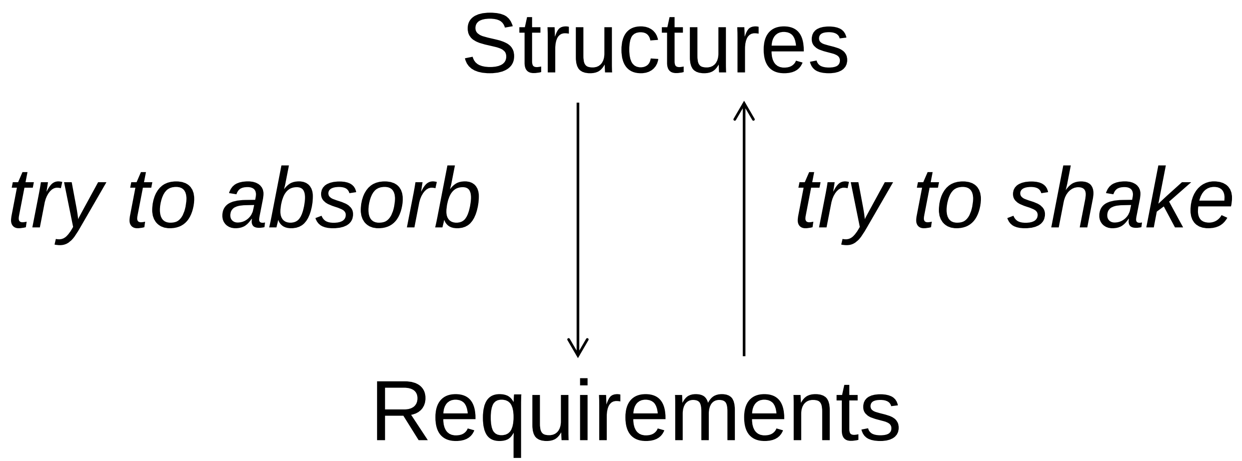
Structures are stable

pixabay.com

Requirements are volatile

> "… systems must be continually adapted or they become progressively less satisfactory."
> Lehman's Law of Continuing Change

The Design Dilemma – 2



Structures

try to absorb

try to shake

Requirements

Architectures should provide stability ***and*** flexibility

- If a design is good, change can be adopted easily
  → the system will adapt and survive
- If a design is bad, change is expensive
  → the system will stop to evolve and die

Enduring Structures

Beetle, 1937

Beetle, 1986

Photos:
www.spiegel.de,
www.flickr.com

Enduring Structures

The original IBM PC in 1981 used an 8088 processor with a 20 bit address bus, allowing access to 1MB of memory. MS-DOS split the memory into a 640k for users and 384k “Upper Memory” for devices like graphics adapters. Unfortunately, these limits got baked into a lot of DOS software.

- Workarounds like Expanded memory (EMS) and Extended memory (XMS) were introduced, but software needed to be programmed to utilize the extra memory.
- Newer CPUs addressed more memory, and the Windows OS on top of DOS let programs use memory outside that limit. However, DOS native items still were bound by those limits.
- Windows 3.0 added DOS Protected Mode Interface in 1989 which resolve the bulk of the 1981 issues. However, programs hard-coded to the limit lingered for years. By 1989, any new code had no issue.

Windows NT completely wiped out all of those limits when first released in 1993.

https://www.quora.com/Why-was-there-a-640k-memory-barrier-with-PC-s-that-took-almost-30-years-to-fix

Initial Design Steps

Start with a one-page vision of the system, describing*:

- Major tasks and domain elements
- Usage and users of the system
- User interface and system interfaces
- Data access and organization
- System control

Add views

- Static Context View: most abstract, top level
- Static Top-down Refinements of the context view
- if needed add dynamic and data views

* *In an ideal world this has been done in the business process / requirements analysis*

Based on Starke, G.: Effektive Software-Architekturen. Ein praktischer Leitfaden. 4th Edition. Hanser, 2009.

Major Tasks and Domain Elements

Describe the major task

- two or three positive sentences
- use domain terms
- list the five most important domain terms

Agree on this with your customer

Usage and Users of the System

What type of usage should be achieved? (More than one aspect may apply)

- interactive & operational → transform current data, highly available and performance (“CRUD”)
- decision support → read / evaluate current data (copies), lower available and performance
- batch → pre- or post-processing, interaction with other systems
- embedded → hardware integration
- real-time → guaranteed execution times

Stakeholders

- Users of the system and their roles
- Negative stakeholders

User Interface And System Interfaces

User interfaces

- form-based / object-oriented / command-line / special devices
- adaptable to experience / user groups
- installer / user

System interfaces

- to / from other systems, performance, stability, changeable
- data-interface
- functional interface: synchronous / asynchronous, definition, fault-tolerance

Data Access and Organization

Storing data in

- main memory
- files
- database management systems (DBMS)

Think about

- needed persistence, license costs, size of data, performance, extendibility, parallelism, integrity, transactions, security, recovery

Typically you will need a professional DBMS

System Control

Ways to control the system

- procedural → a main component controls a sequential flow, minimal user interaction
- event driven / reactive → graphical user interfaces, client/server, feedback control systems
- parallel → independent interacting components, client/server
- rule-based → rules executed by a rule-engine

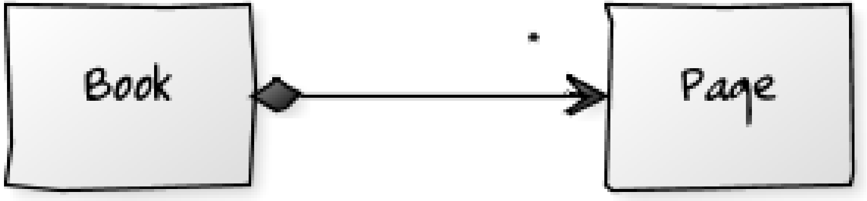
Design Views

- Context View
  - most abstract, top level static view
- Static View
  - top-down refinement of the context view
- if needed, Dynamic View
  - behavior of (some) components
- if needed, Deployment View
  - physical distribution of the components
- if you only design the Database: Data View
  - ER-Diagrams explain/refine data structures

View-Notation

For describing the views we use a simplified UML

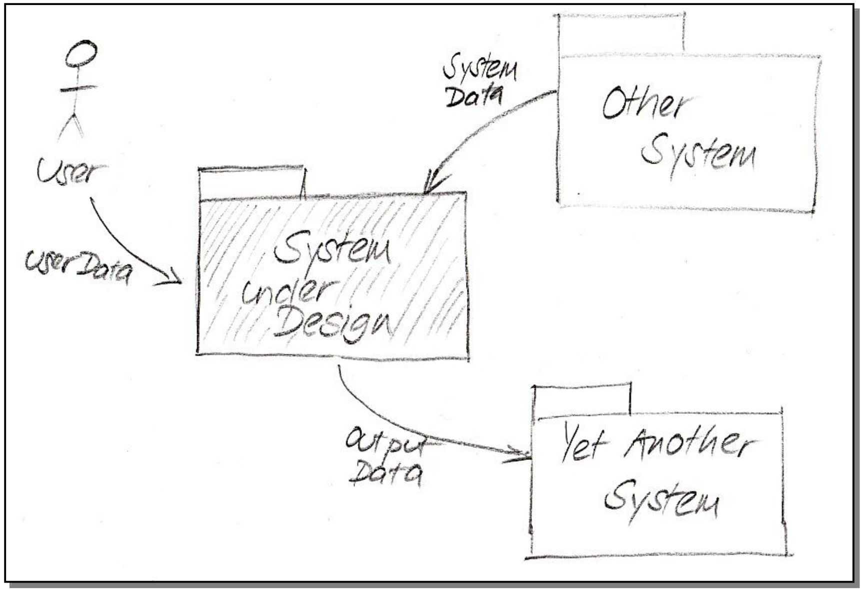
- Advantages:
  - simple to understand and to use
  - concentration on basics, neglecting the details
  - Ideally some "scribbled UML"
- Disadvantages:
  - No round-trip-engineering
  - important details may be neglected



http://yuml.me

Context View – Top View on your System

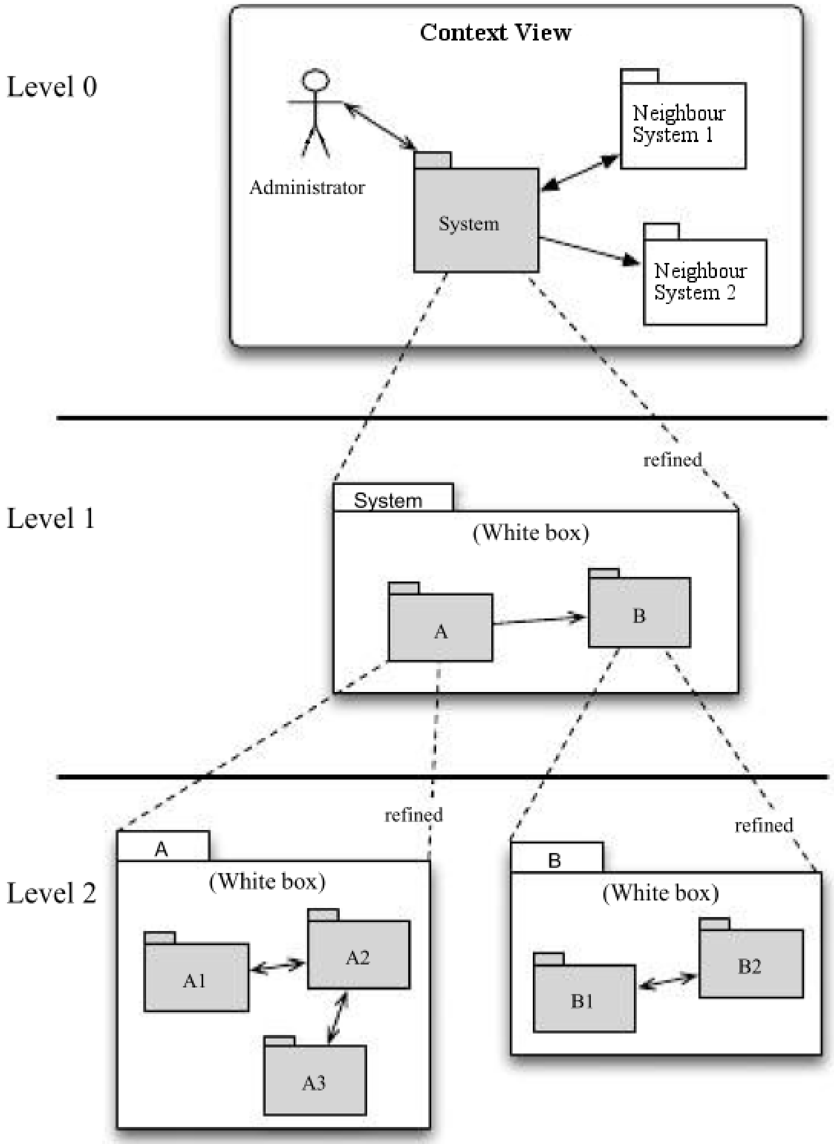
- The system is seen as black box
- it is connected to the environment
  - other systems
  - users
- Use a package-diagram-like representation



Static View

- The system is seen as related components
- Components
  - may use each other within one level
  - may be refined hierarchically into lower levels
- Components may be (depending on the level)
  - Packages, Classes, Modules, Tables, Functions, Libraries, …
- Notation
  - (simplified) Class- / Package-Diagrams

Static View – Example



Source: Gernot Starke: Effektive Software Architekturen

Hochschule für Technik Stuttgart

Static View – How to Achieve

- Start at the domain and do a top-down refinement
- For each component document
    - its responsibility
    - its inputs and outputs
    - use-type (member/dependency | call/return | synchronous/asynchronous | push/pull | notify/update)
    - reasons for this decomposition
- always link to already existing structures (DRY: Don't repeat yourself)

Static View – Remarks

- This is the most important view – expect the most effort here
- Maintain higher levels (avoid too much details)
- Check for reusable components (which you can reuse)
- Various design methods
  - provide refinement-/decomposition-decisions
  - additional/special elements
  - additional/special notations

Dynamic View

- The system is seen as executable units, which
  - may transfer control
  - may transfer data
- Notations
  - Sequence Diagrams

Rules

- try to stick to one type of notation
- usually there are less levels of detail (if any)