UNIVERSITÄT DES SAARLANDES

30 YEARS DFKI Human Centric AI.

Introduction & Basic Concepts

Architectural Thinking for Intelligent Systems

Winter 2019/2020

Prof. Dr. habil.Jana Koehler

Agenda

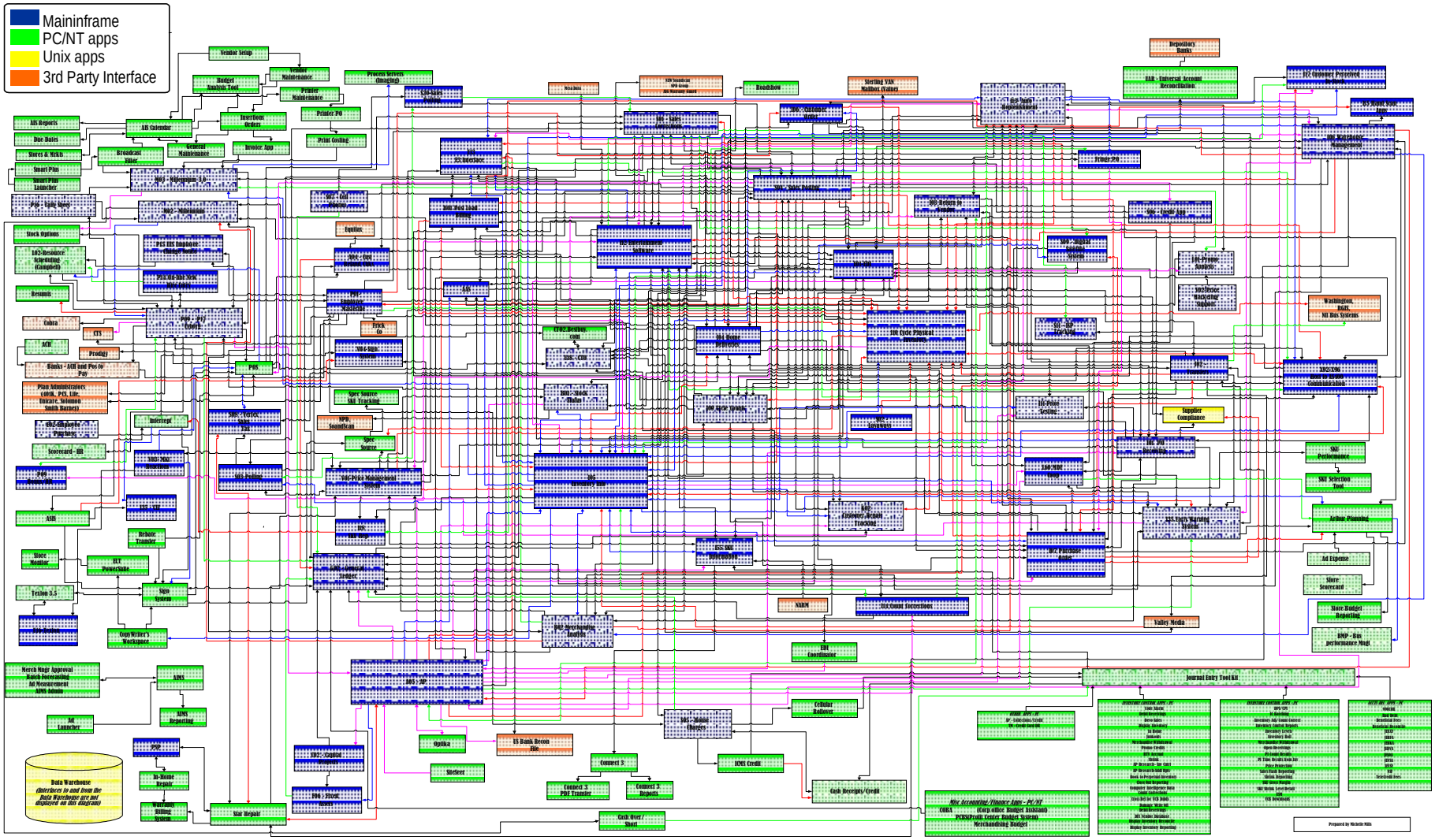
- Big Ball of Mud
- Organization of this lecture
- Solution/Application architecture vs. other architectural disciplines
- Architectural Thinking and the 3 C of success
- Architecture vs. Design

If you think good architecture is expensive,

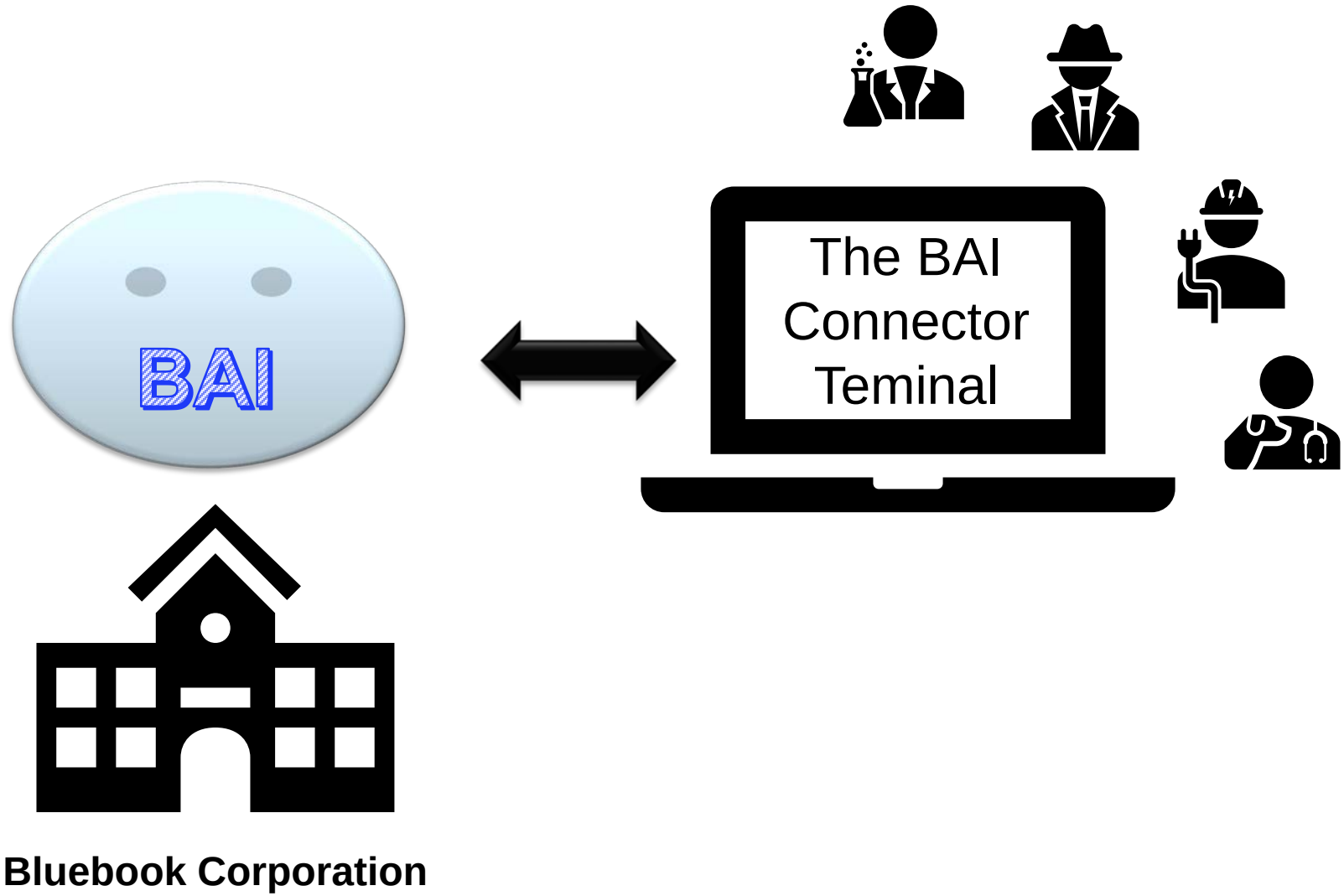
try bad architecture

Brian Foote and Joseph Yoder: “Big Ball of Mud”

http://www.laputan.org/mud/



Let´s Build a System …



BIG BALL OF MUD: The most frequently deployed Software Architecture

- A casually, even haphazardly, structured system
- Its organization, if one can call it that, is dictated more by expediency than design
- Several patterns describe the forces that encourage the emergence of a BIG BALL OF MUD:
  - PRESSURE TO DELIVER
  - THROWAWAY CODE
  - PIECEMEAL GROWTH
  - KEEP IT WORKING
  - SHEARING LAYERS
  - SWEEPING IT UNDER THE RUG
  - RECONSTRUCTION

PRESSURE TO DELIVER

- SPAGHETTI CODE
- You need to deliver quality software on time, and under budget
- *Therefore*, focus first on features and functionality, then focus on architecture and performance

THROWAWAY CODE

- QUICK HACK, SCRIPTING, KILLER DEMO, PERMANENT PROTOTYPE

- You need an immediate fix for a small problem, or a quick prototype or proof of concept

- *Therefore*, produce, by any means available, simple, expedient, disposable code that adequately addresses just the problem at-hand

PIECEMEAL GROWTH

- ITERATIVE-INCREMENTAL DEVELOPMENT
- Master plans are often rigid, misguided and out of date. Users' needs change with time
- *Therefore*, incrementally address forces that encourage change and growth
- Allow opportunities for growth to be exploited locally, as they occur
- Refactor unrelentingly

KEEP IT WORKING

- VITALITY, BABY STEPS, DAILY BUILD, DO NO HARM
- Maintenance needs have accumulated, but an overhaul is unwise, since you might break the system
- *Therefore*, do what it takes to maintain the software and keep it going. Keep it working

SHEARING LAYERS

- GLUE CODE, ADAPTERS, FACADES, INTERFACES
- Different artifacts change at different rates
- *Therefore*, factor your system so that artifacts that change at similar rates are together

SWEEPING IT UNDER THE RUG

- POTEMKIN VILLAGE, PRETTY FACE, QUARANTINE, HIDING IT UNDER THE BED, ENCAPSULATION

- Overgrown, tangled, haphazard spaghetti code is hard to comprehend, repair, or extend, and tends to grow even worse if it is not somehow brought under control

- *Therefore*, if you can't easily make a mess go away, at least cordon it off (isolate). This restricts the disorder to a fixed area, keeps it out of sight, and can set the stage for additional refactoring

RECONSTRUCTION

- TOTAL REWRITE, THROWAWAY THE FIRST ONE, START OVER

- Your code has declined to the point where it is beyond repair, or even comprehension

- *Therefore*, throw it away and start over

➢ To prevent such a situation:

➢ ***Refactoring: Improving the Design of Existing Code*** by Martin Fowler

Goals of this Course

- Learn essential elements of architectural thinking
- Understand the relationship of architecture – design – code
- Deepen and expand learned knowledge for building software systems
- Understand architectural thinking as a method to control project risk

➢ **Systematically learn, apply, and deepen architectural knowledge by working out an architectural solution concept for a specific project**

Lecture Plan

In red: tutorial plan & date for assignments (upload day before until 5.59pm)

Week 1 21.10.	Week 2 28.10.	Week 3 4.11.	Week 4 11.11.	Week 5 18.11.	Week 6 25.11.	Week 7 2.12.
A1: Introduction to Architectural Thinking - Big Ball of Mud - Organization of this lecture - Solution/Application architecture vs. other architectural disciplines - Architectural Thinking - Architecture vs. Design	**A2: Modeling for Architects I** - Capturing architectural concepts with UML 2 - Sequence diagrams - Package & Component diagrams - State machines - Use case diagrams	**A3: Modeling for Architects II** - Analyzing business processes with BPMN 2.0 - Understanding Business Object Lifecycles **1) UML**	**A4: Modeling for Architects III** - Understanding forces and concerns - Architectural concerns and decisions in ISO 42010 - Architecture documentation, Enterprise Architecture Frameworks **2) BPMN 2.0**	**A5:System Functionality** - Negotiating functional requirements - Goal hierarchies - Writing good use cases and user stories **3) Forces, Concerns, Architectural Decisions**	**A6: System Qualities** - Importance of non-functional requirements - Making qualities measurable with scenarios **4) Goal Hierarchies and Acceptance Tests**	**A7: System Vision, Idea, and Views** - Formulating the System Idea and Vision - Views & Viewpoints - Operational model **5) Scenarios**
Week 8 **9.12.**	**Week 9** **16.12.**	**Week 10** **13.1.**	**Week 11** **20.1.**	**Week 12** **27.1.**	**Week 13** **3.2.**	**Week 14** **7. 2.** **(Friday!)**
A8: Domain-Driven Design - Understanding the business domain - Domain elements & bounded contexts - DDD context maps and the big ball of mud **6) System Vision, System Idea, Views**	**A9: Principles & Tactics** - 10 principles: Loose Coupling,High Cohasion, Design for Change, Separation of Concerns, Information Hiding, Abstraction, Modularity, Traceability, Self documentation, Incrementality - Tactics **7) DDD & Revision of Decisions**	**A10: Architectural Styles** - Layers, Tiers - Peer2Peer - Client-Server - Pipes & Filters - SOA, Microservices - Blackboard - Onion,Clean, Lambda **8) Principles & Tactics**	**A11: Architectural Patterns** - Enterprise Application Integration (EAI) - EAI Pattern - File Transfer - Shared Database - Remote Procedure Invocation - Messaging **9) Architectural Styles**	**A12: Evaluation of Architectures** - Architecture Tradeoff Analysis Method ATAM - Scenarios - Risks and sensitive points **10) EAI Pattern**	**A13: AI Architectures** - AI agent model - Shakey Layers, Belief-Desire-Intention, - Brooks Subsumption Architecture, - SOA Cognitive Architecture **11) ATAM**	**A14: Summary** - Challenges & Risks in architectural thinking - Architect profession and career paths **A15:Examen Preparation**

How we Work in this Course

- **Monday afternoon in Week n**:
  - Learn about method in lecture
- **Until Sunday 6 pm in Week n**:
  - Apply method to own project based on questions from tutorial working guidelines document
  - Upload your solution
- **Monday morning in Week n+1**:
  - Selected teams present solution concepts in class
  - Discuss solutions
  - All submitted solutions are shared among all participants

Team Organization

- We use a paper-based inscription list
- Form teams of 2-3 people

- Put the full name of all team members on the list
- Remember your team number

- Until Week 4 add a short name and description of your project on the list
  - project-specific assignments start in Week 4

The Team Project

- Choose a system, for which an architecture needs to be devised, such a system can be
  - a known AI system already in existence (e.g. Google search, Alexa, a subsystem of Facebook, a Natural-Language based application, etc.)
  - a specific system or app, that a team wants to build, is building, or has built in the past
  - any other app or system (including non-AI), a team wants to use for practicing the methods taught in this course
- Create complete architectural description document by working through all tutorial questions

Submit Your Assignment Documentation

- Name your file: <AssignmentNo>-<Team-No.>
  - 1-6.pdf – Assignement 1, Team 6
  - 11-3.pdf Assignment 11, Team 3

- One representative of the team uploads the solution to the ATIS CMS

Participation Requirements & Admission to Examen

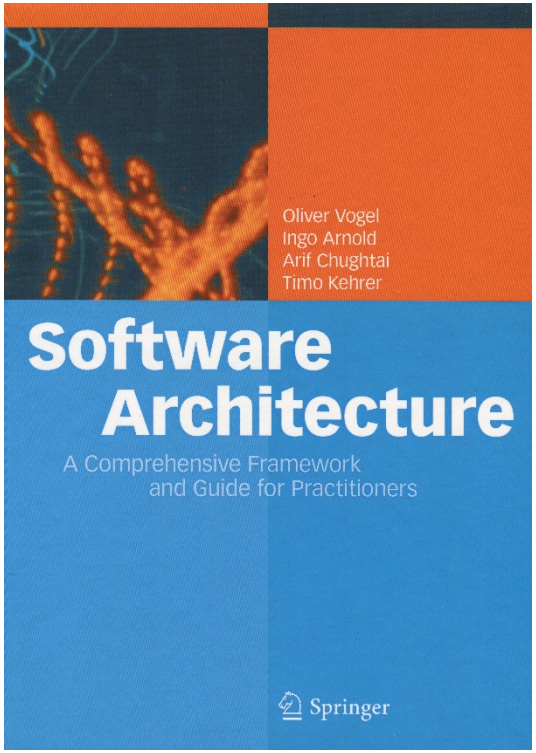
- At least 1 member of each team must be present in each tutorial
- Each student must participate in 8 out of 11 tutorial sessions
- All submitted solutions must be formally accepted (=11 out of 11)
  - Formal acceptance until Monday evening 8pm for solutions submitted on Sunday until 5:59pm
  - If not accepted, 1 week for improvement & resubmission
  - Selection of presentation is independent of submission status
- Each team presents at least 3 times

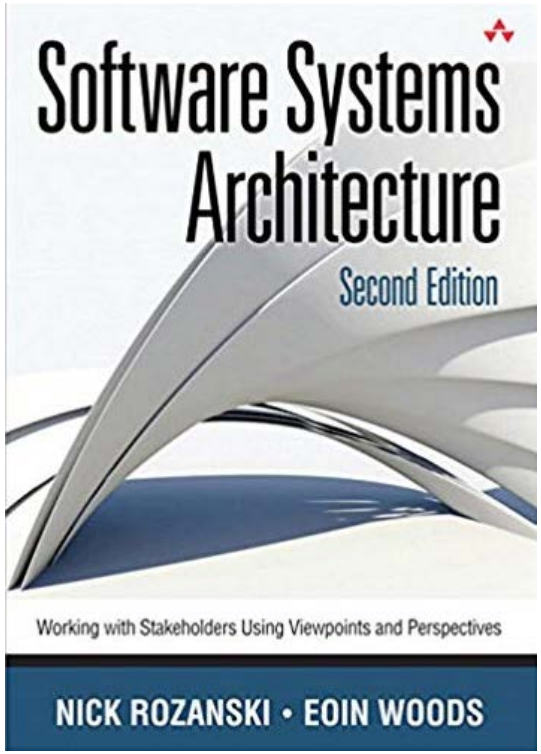
Examen

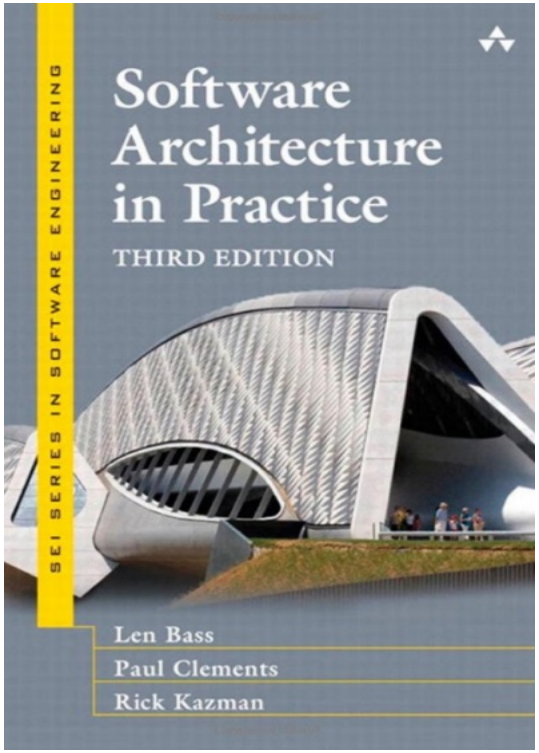
- Written Examen 90 minutes
- Closed book
- Only paper and pen allowed
  - No pencils, no electronic devices
- Focus is on applying methods
- Check slide deck 15 for more information

- If passed → Great ☺
- If failed, you will have to wait one year until the next iteration
  - No re-exam!

Main Literature

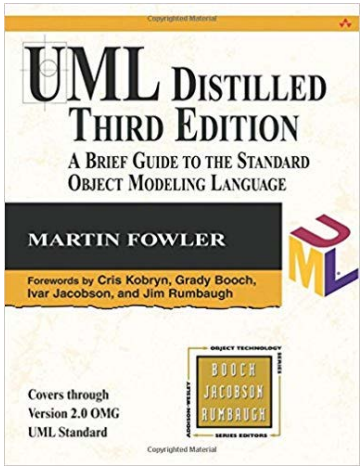


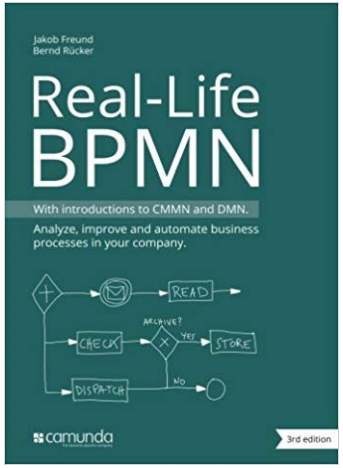


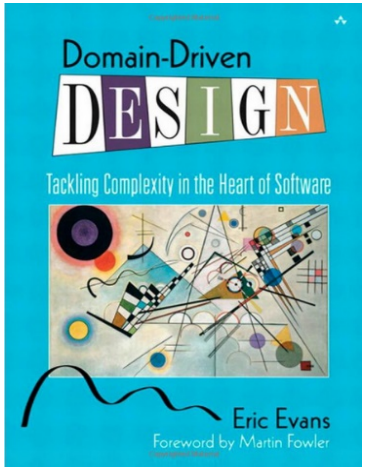


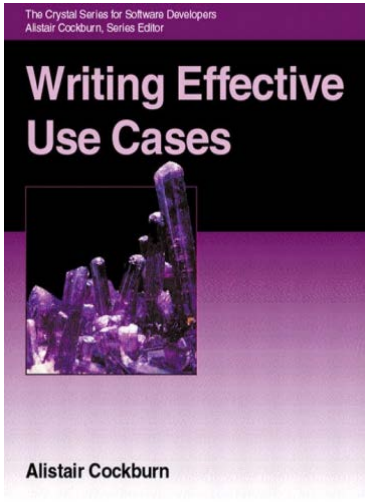
Coherent and systematic presentation of all concepts
Practical insights, numerous examples and patterns

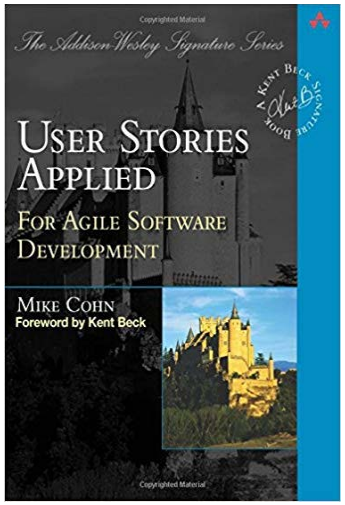
Additional Sources for Specific Methods

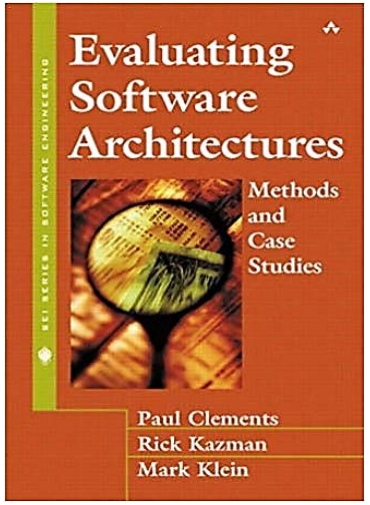










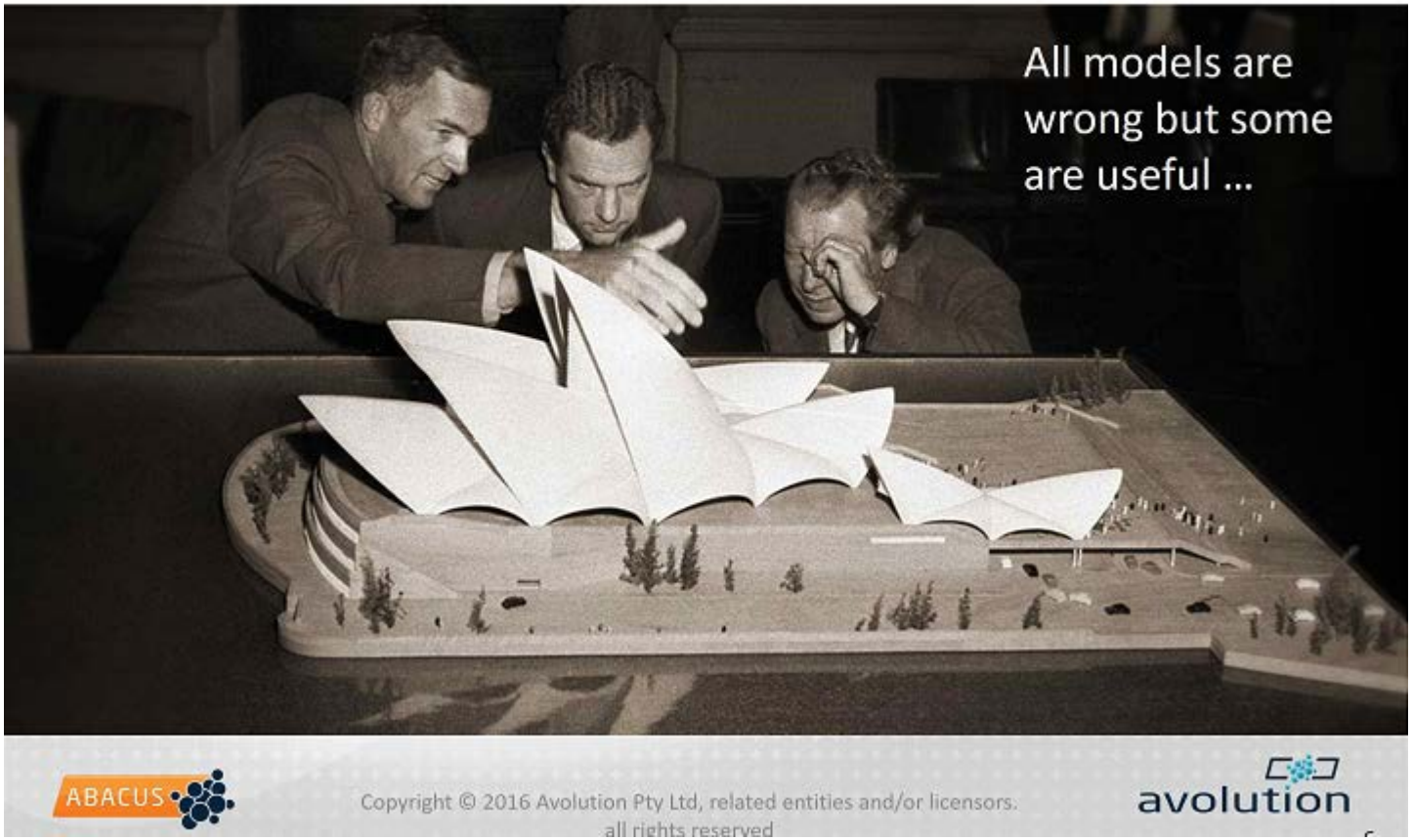


Literature in German

Prerequisites

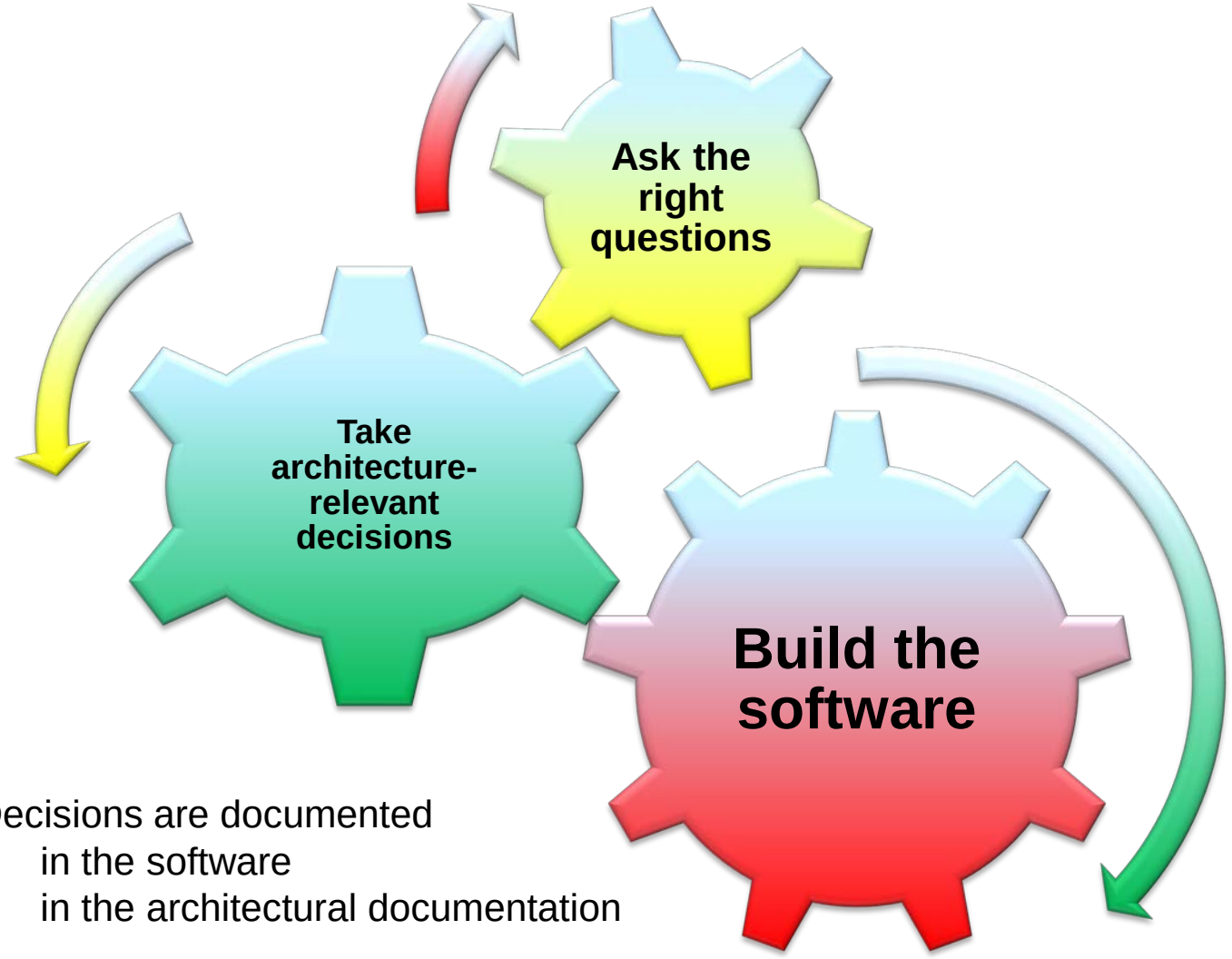
- Software and Systems Modeling
- Software Engineering & Development
- Agile software development methods (Scrum)
- Software project management

1. Something … is better than nothing



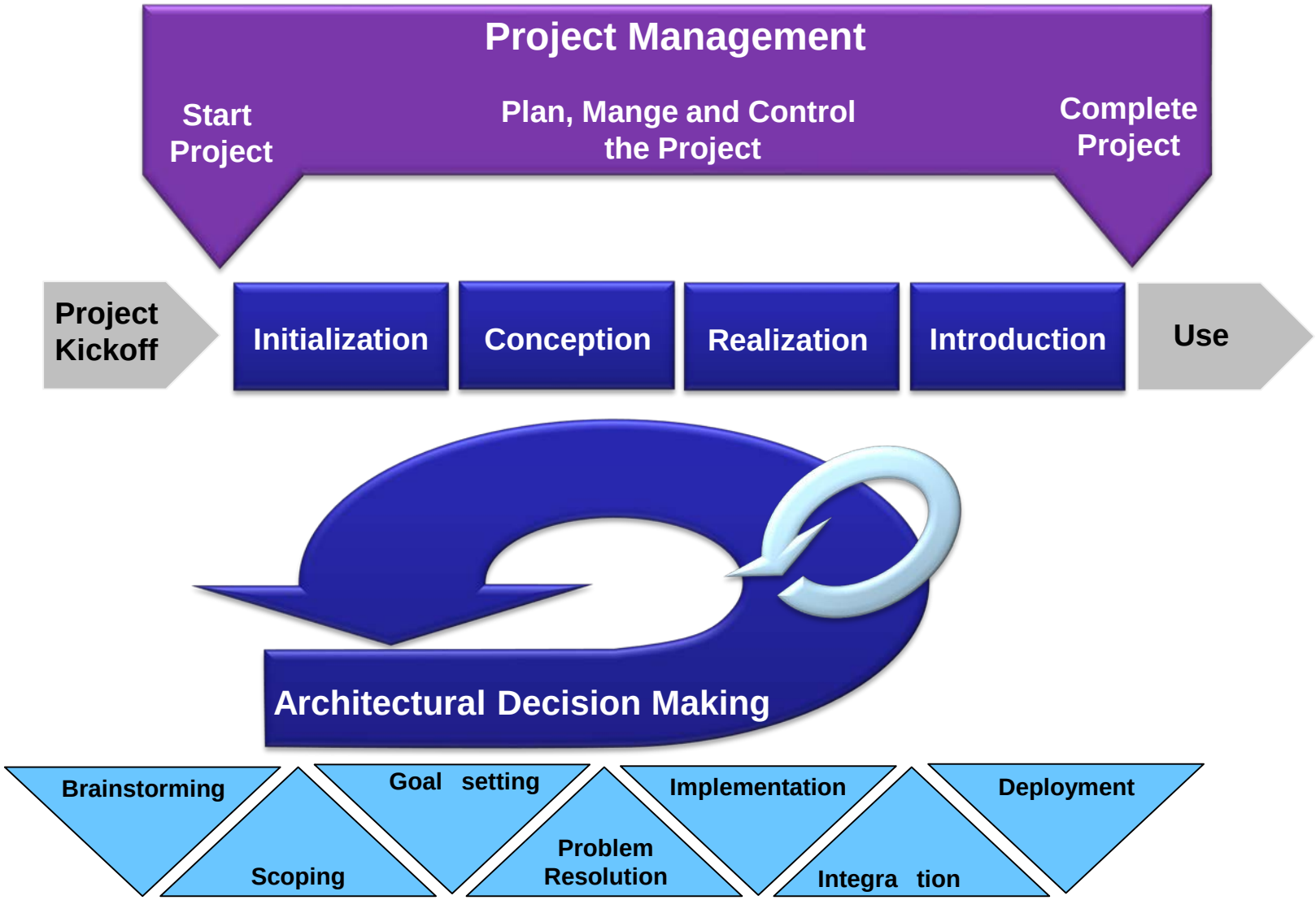
Open Group Webinar: 5 Quick Wins That Give Enterprise Architects an Edge

Software Architecture and Agility

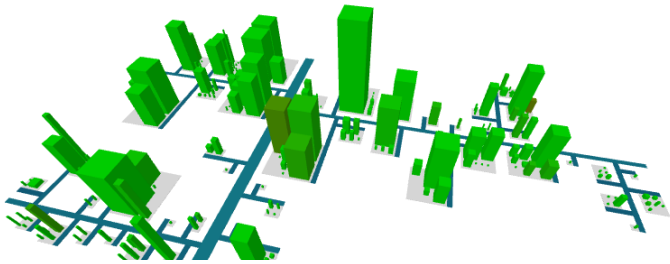


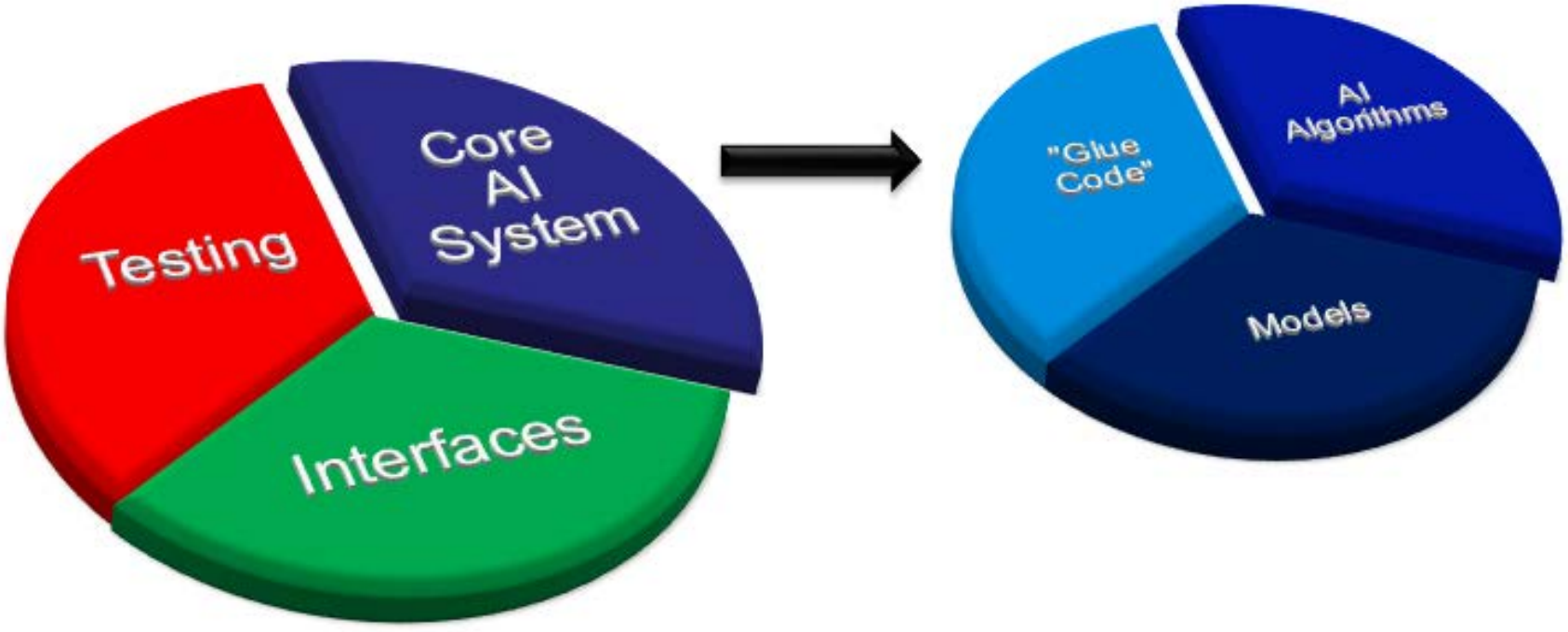
Decisions are documented
- in the software
- in the architectural documentation

Architecture and Agile Development

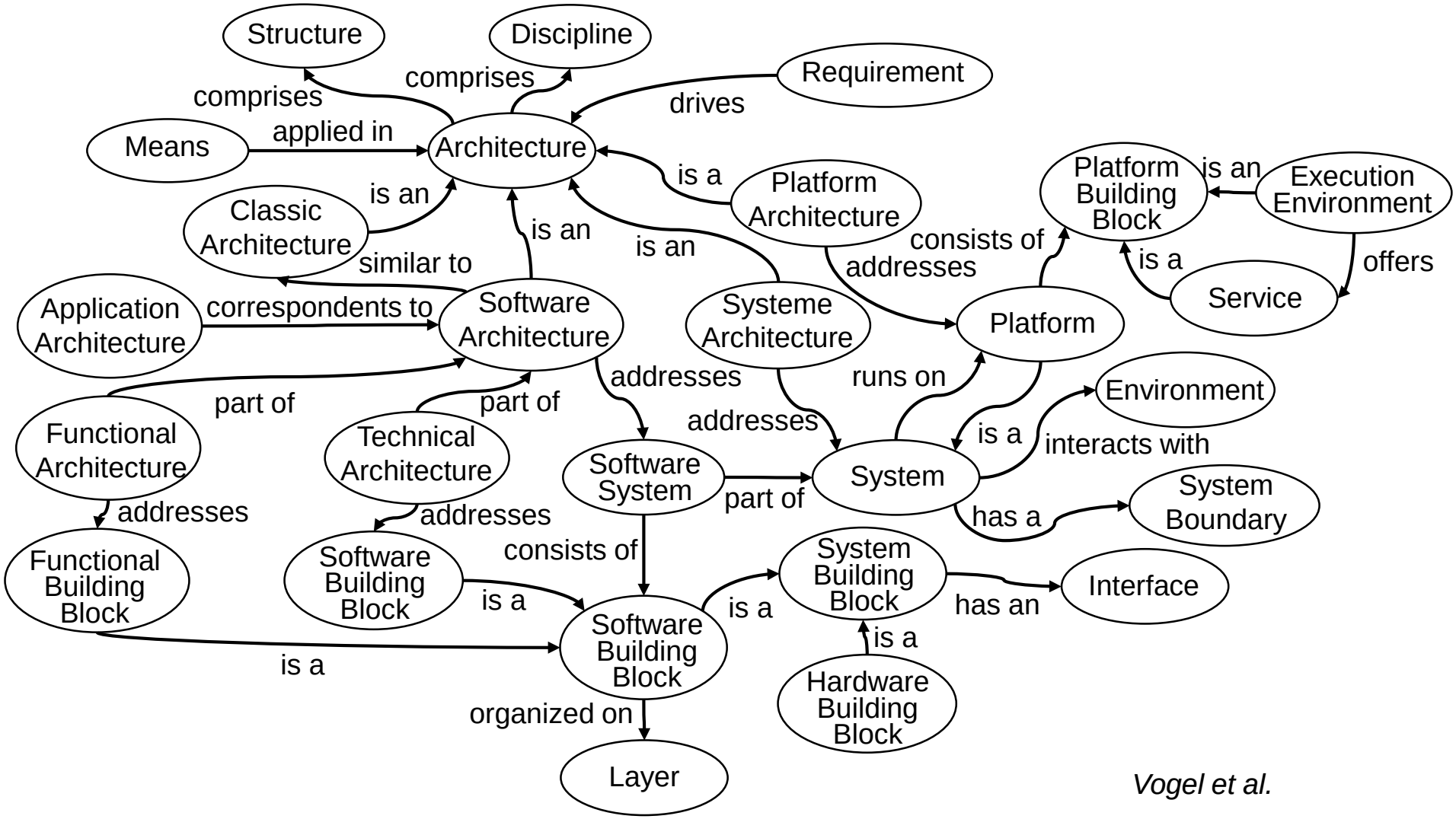


Software Architecture and Artificial Intelligence



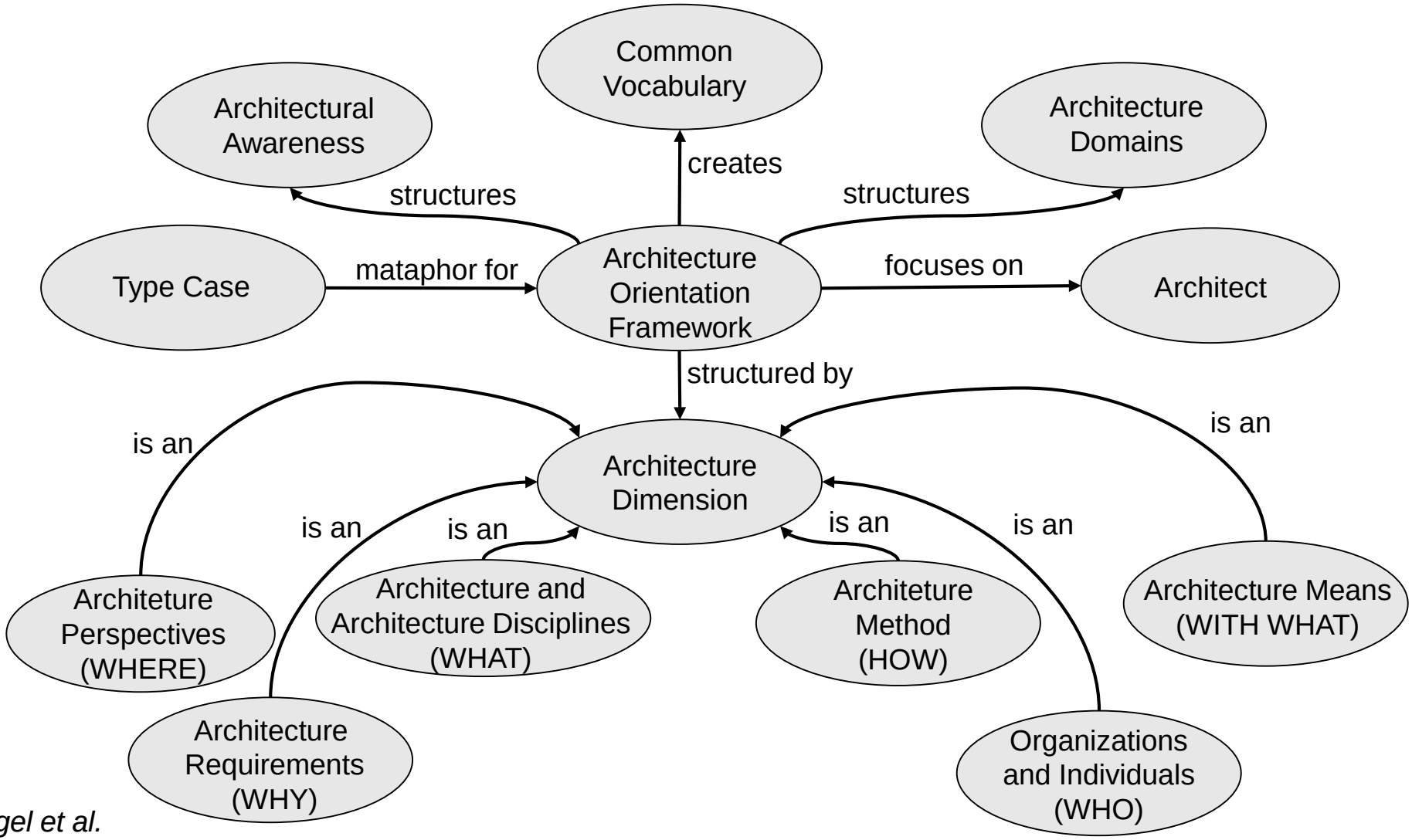


The Language of Architects



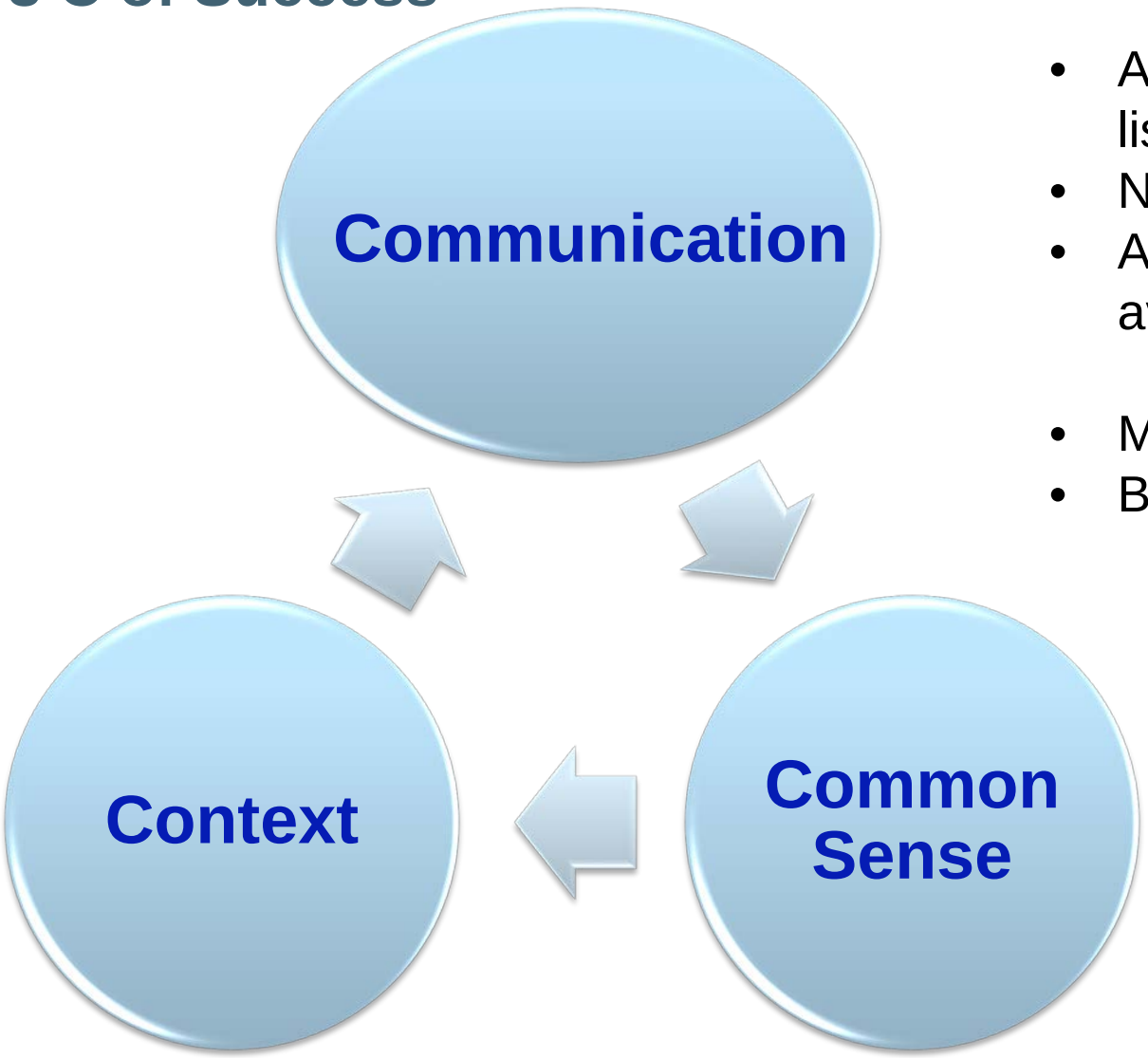
Vogel et al.

Basic Concepts



Vogel et al.

3 C of Success



- Ask the right questions and listen carefully
- Negotiate – don’t clarify!
- Apply proven methods, avoid known errors

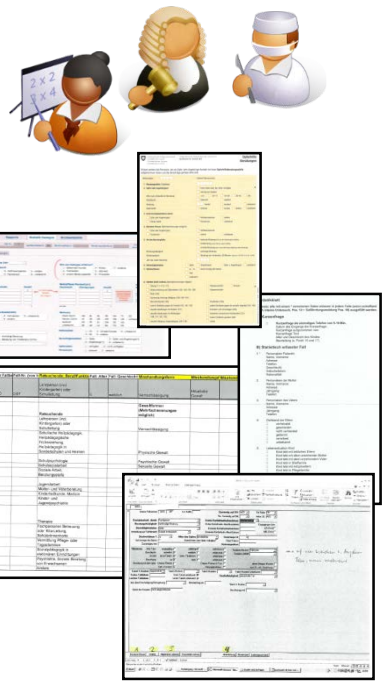
- Manage Risks
- Build on experience

Architectural Concerns in a Data Integration Project

UNIL | Université de Lausanne
Observatoire maltraitance envers les enfants

Lucerne University of Applied Sciences and Arts
HOCHSCHULE LUZERN

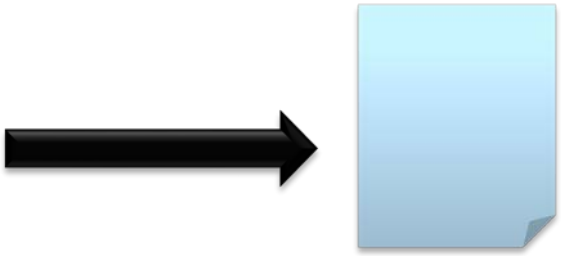
Institutions



Data in different Formats and Tools

Optimus Study, cycle 3

The magnitude of legal, health and child protective services response to child maltreatment in Switzerland



Study Team

Overview

AI Architecture of an Elevator Control System

How do you define "Software Architecture"?

Architecture Definition I

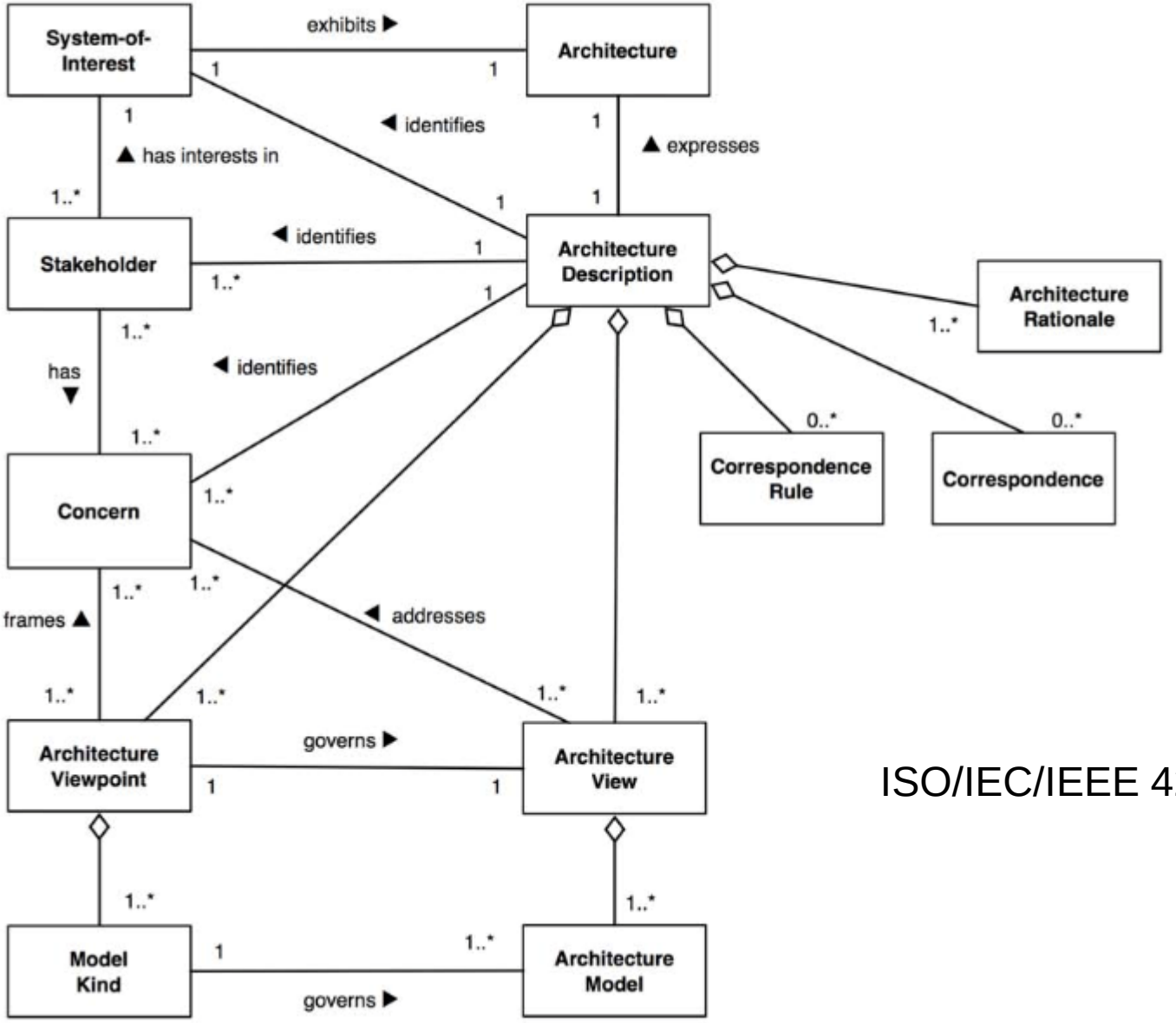
The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both.

Bass,Clements,Katzman: Software Architecture in Practice, Addison Wesley 2003 and ISO/IEC/IEEE 42010:2011, Systems and software engineering — Architecture description

- Based on the notion of a system
  - Elements organized by structure and relationships in a system with a given purpose to achieve goals and a clear boundary separating it from its environment

Implications of this Definition

- Architecture is
  - a set of software structures
  - an abstraction
- Every software system has a software architecture
- Architecture includes behavior



ISO/IEC/IEEE 42010:2011

Architecture must …

- … define the components of a system
- … describe its essential (externally visible) features
- … characterize the relations between these components

- Static aspects: building plan
- Dynamic aspects: work plan

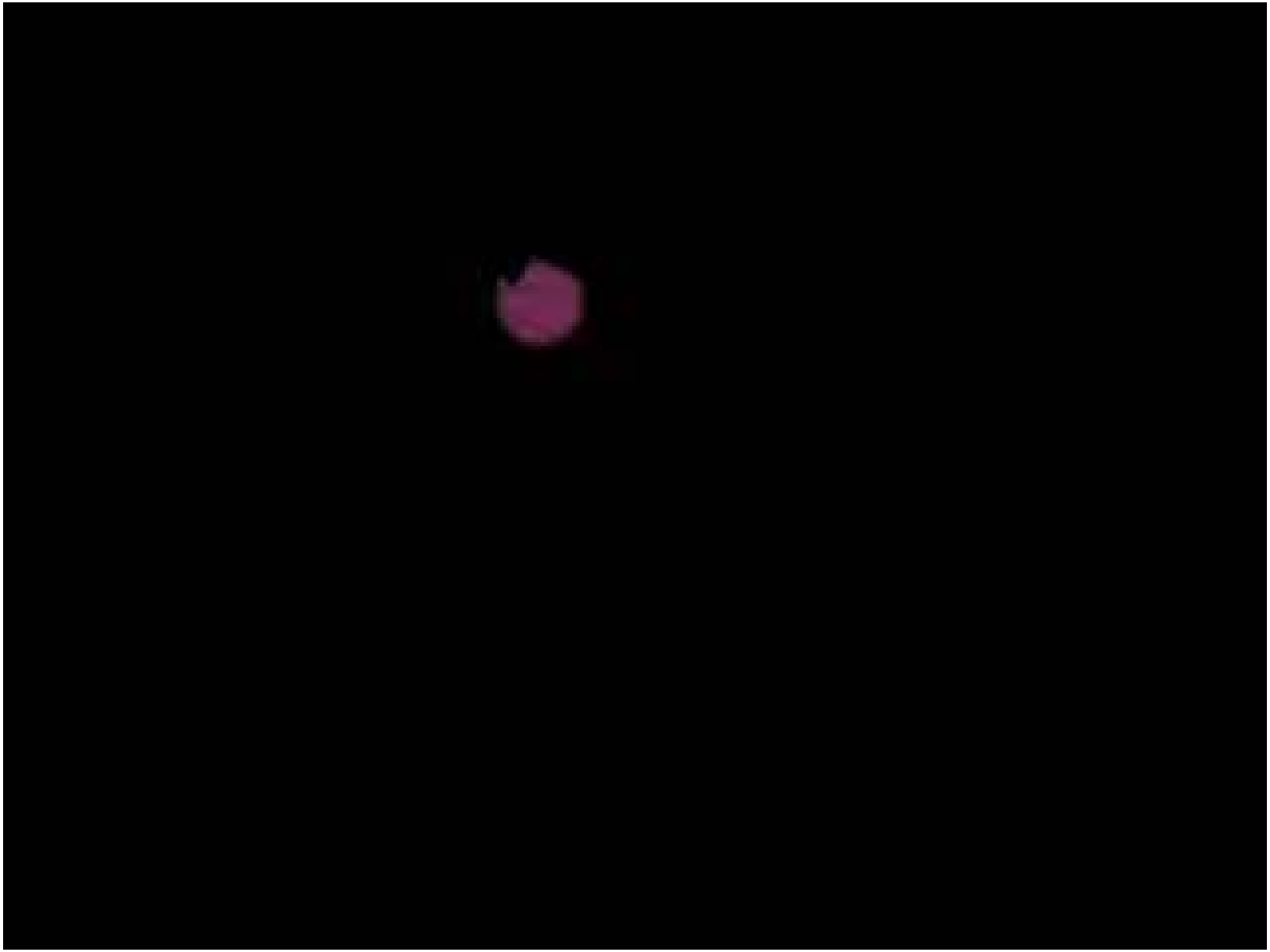
- Architecture as a scientific discipline and profession
  - Methods and principles to create an architecture

Our Focus - Solution/Application Architecture

Infrastructure Architecture *«Supply Network"*

Solution Architecture *«House Plan"*

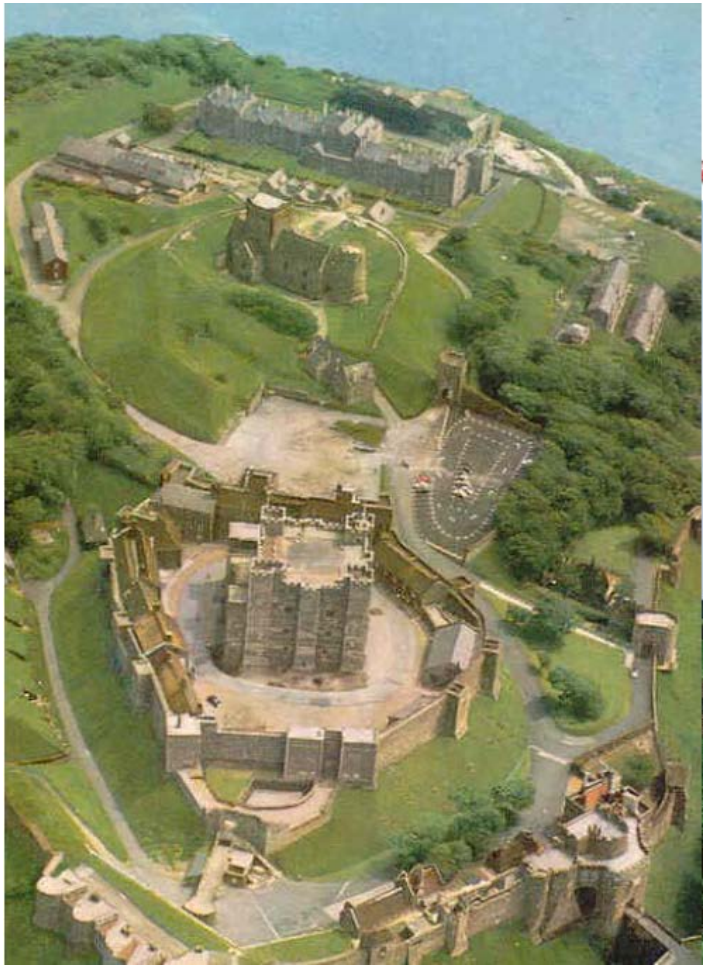
Enterprise Architecture *«City Plan"*



Building Architecture

- Requirements determine style

Impress



Find and reuse substructures

Defend

There is no such thing as the "best" solution – be fit for a purpose

Ensure Mobility

Question: Software Architecture – Building Architecture

a) How far carries the analogy?

b) What do both architectural disciplines have in common?

c) Where are important differences?

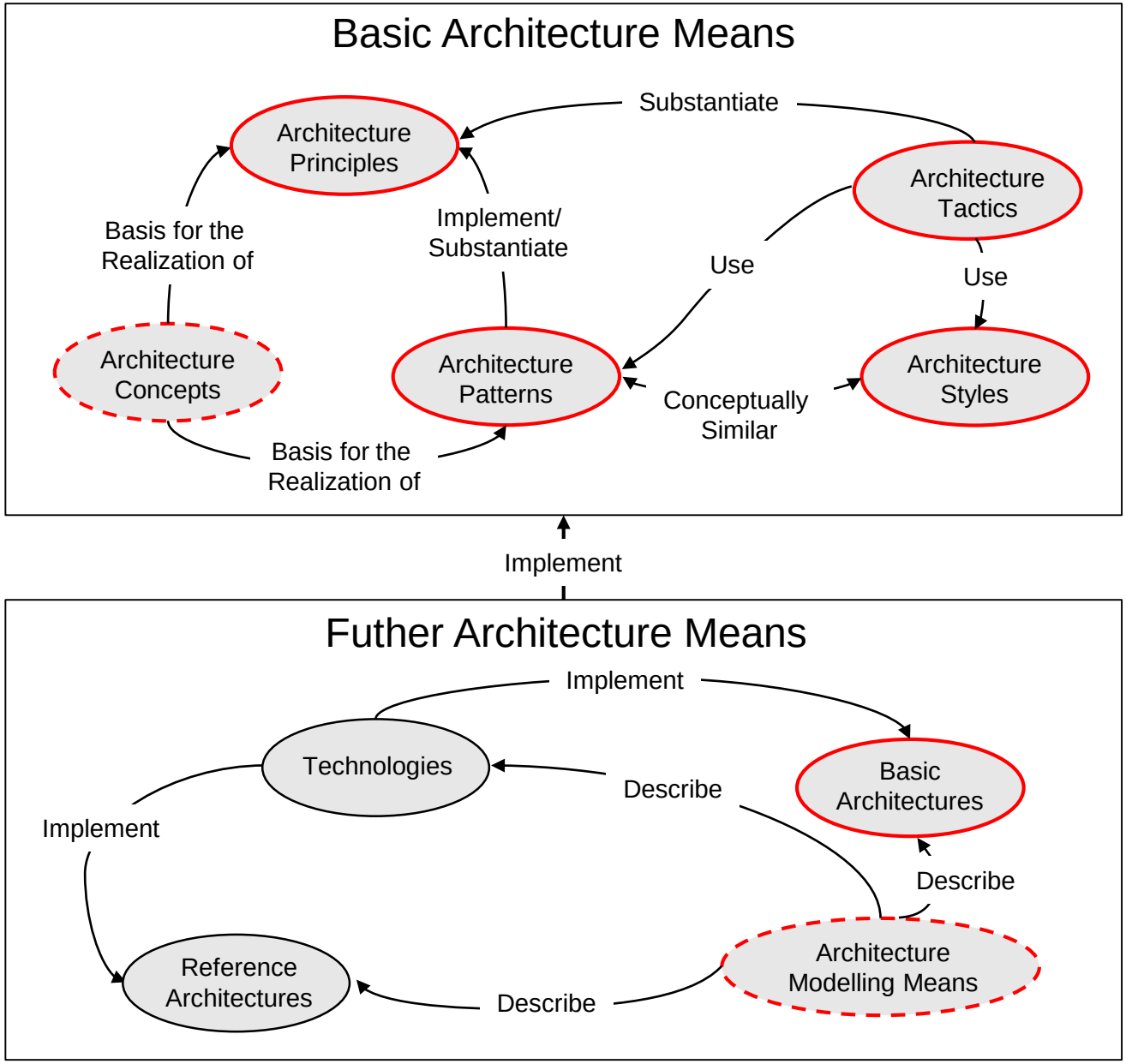
d) Which decisions are taken at the software architectural, design, implementation level?

Most Relevant Goal: Complexity Reduction

- by decomposition and structuring
- by abstraction
- by reuse
- by good documentation

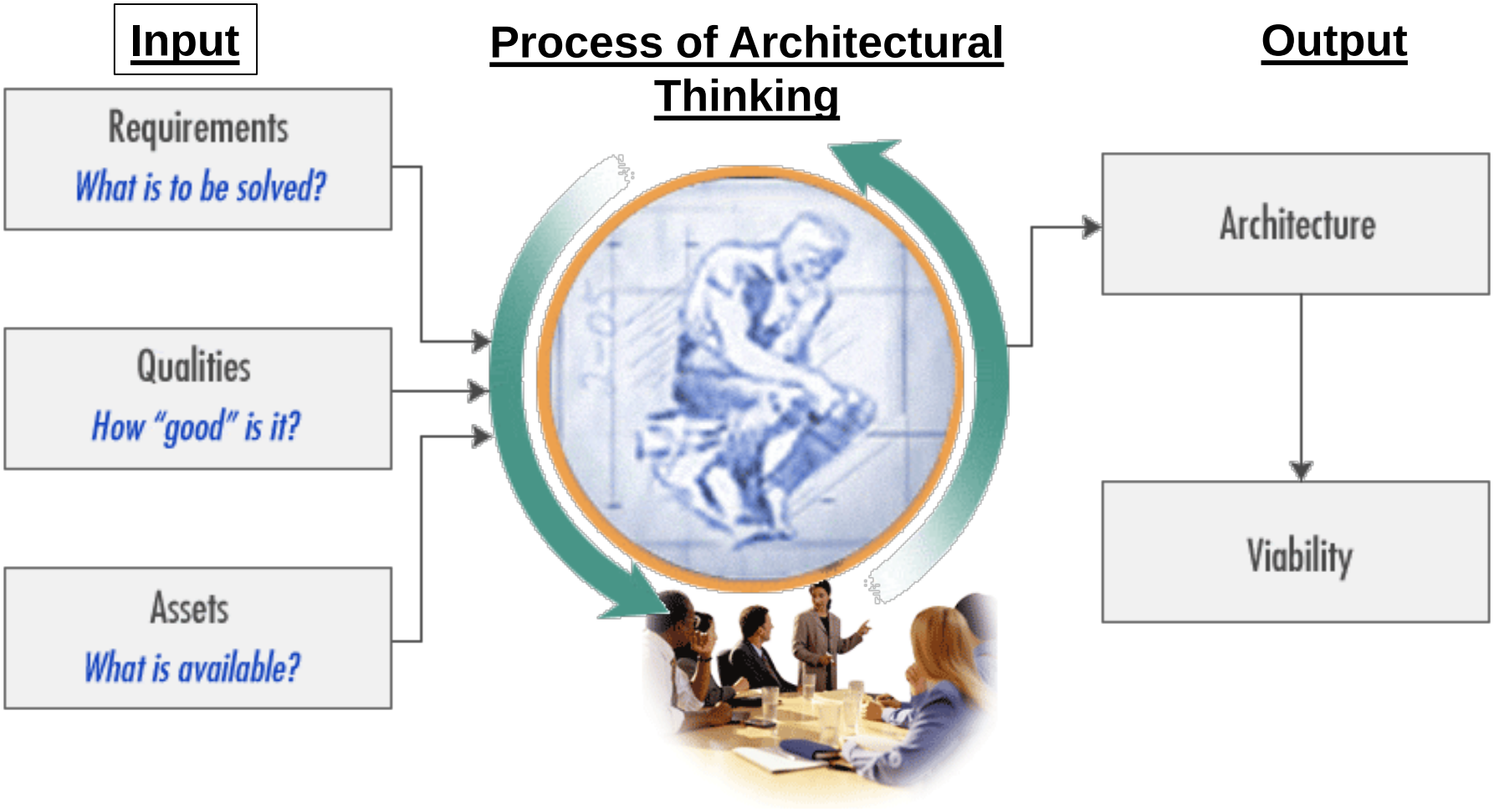
- The importance of architectural thinking is growing
  - Today's systems have to be transformed rapidly to meet changing requirements
  - More heterogeneous systems (various technologies, different interfaces, evolving functionality, platform independence)
  - Increased interoperability with other systems

Basic Concepts in the Focus of this Lecture



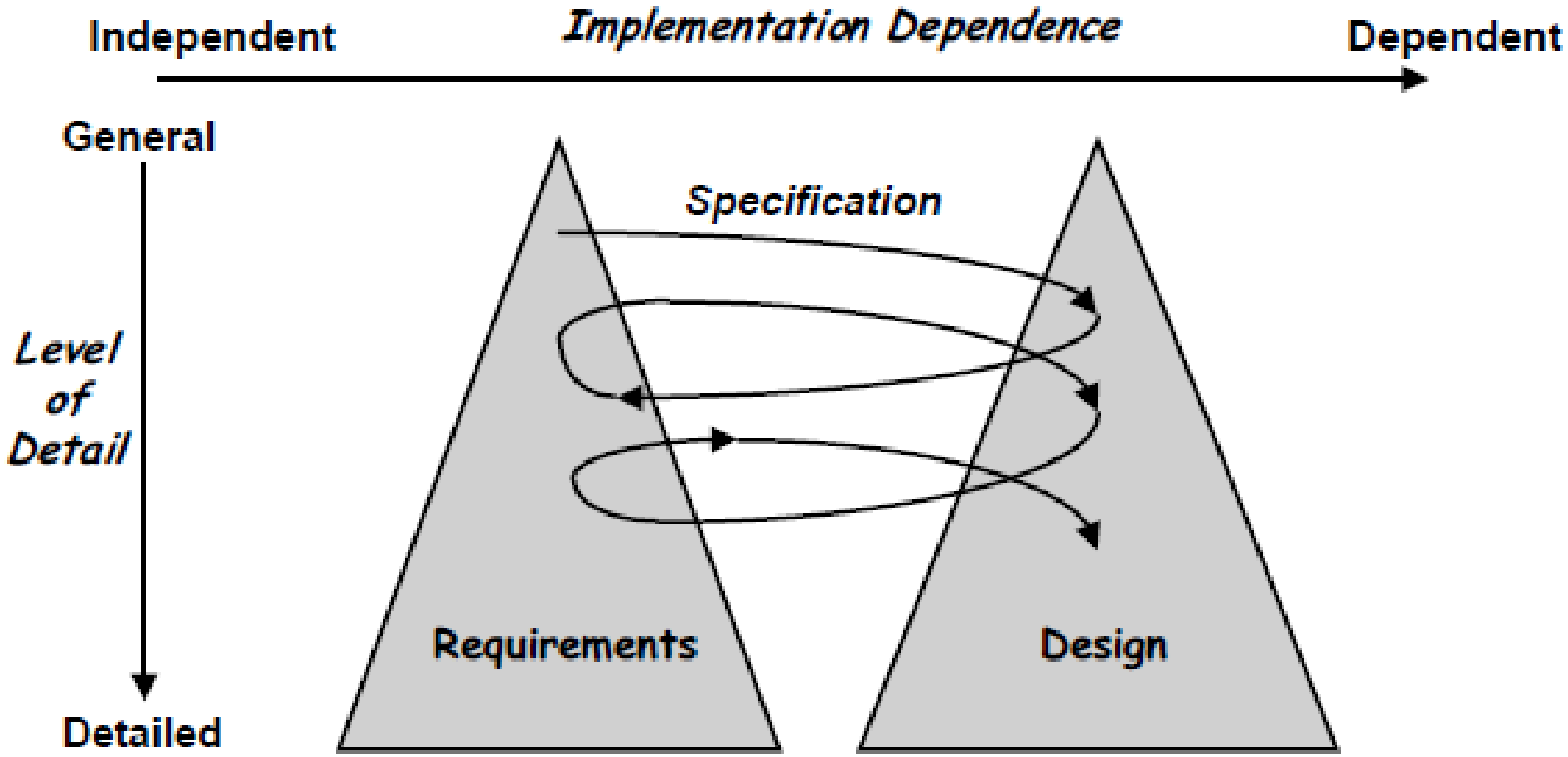
Vogel et al: Software Architecture

Key Elements of Architectural Thinking



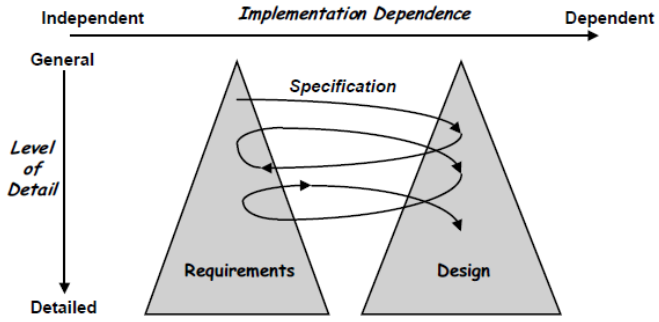
Source: IBM Architectural Thinking

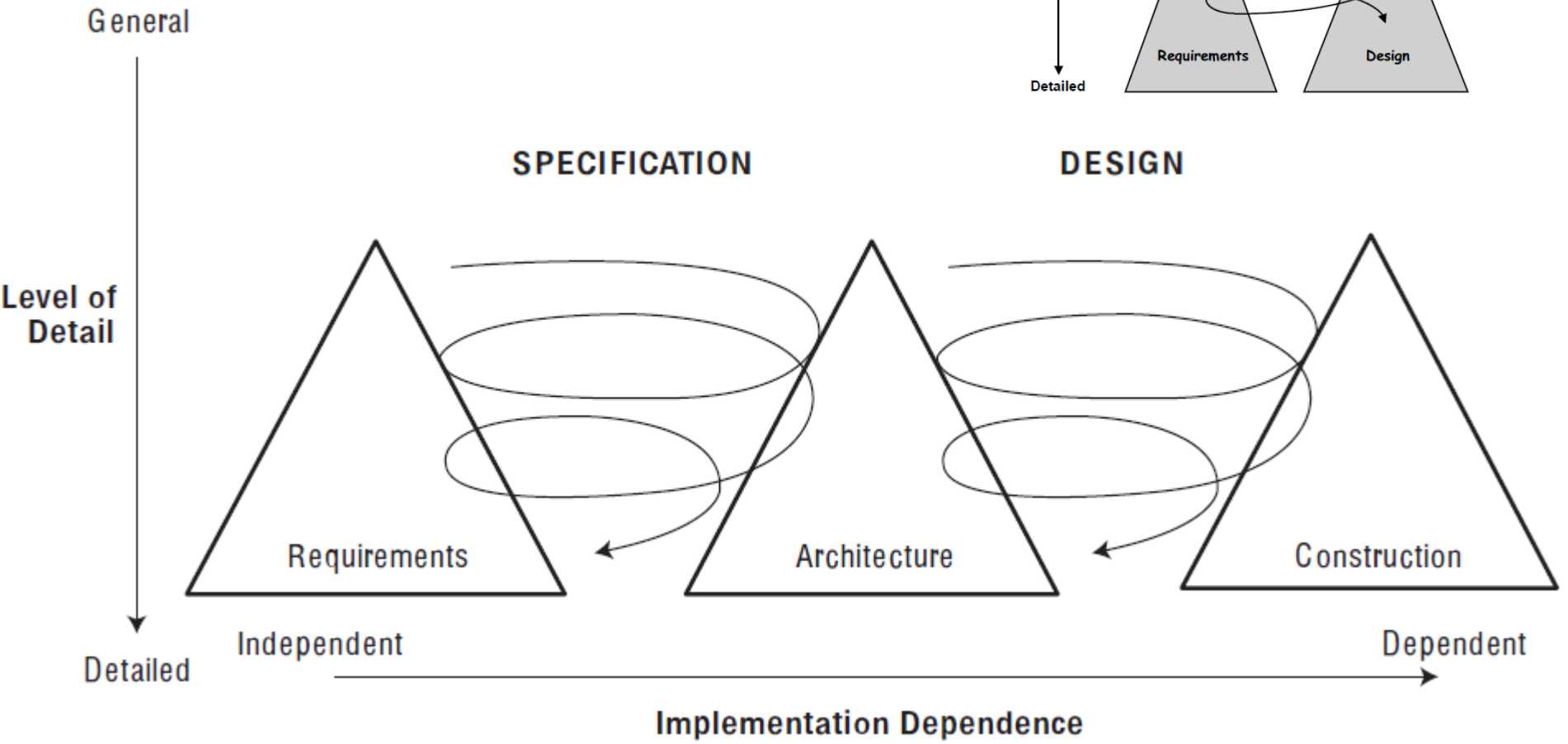
Twin Peaks Model (Nuseibeh, 2011)



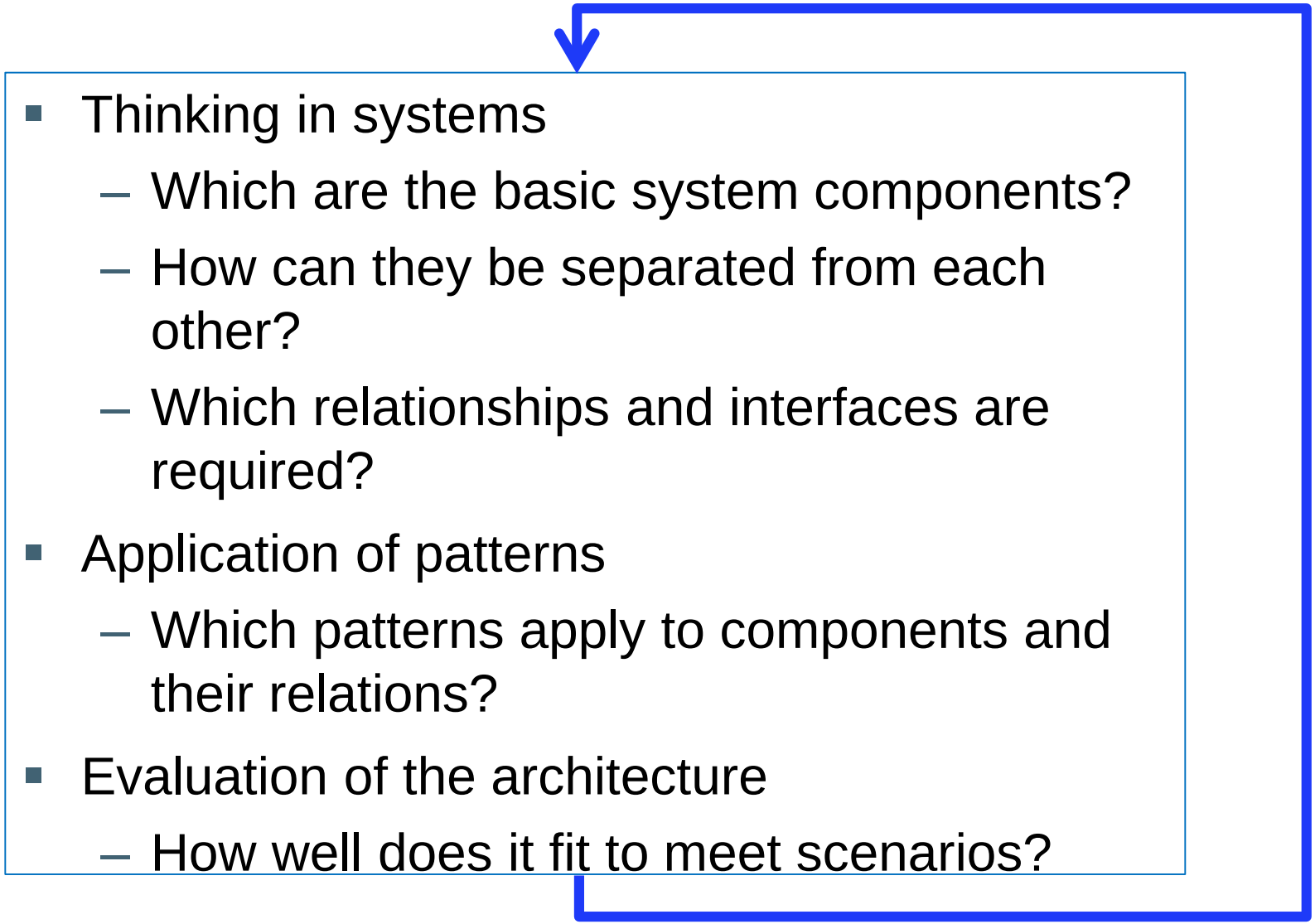
Enhanced Twin Peaks Model (Woods & Rozanski, 2010)

- Architecture as mediator between requirements and code





Iterative Evolution of a System´s Architecture



- Thinking in systems
  - Which are the basic system components?
  - How can they be separated from each other?
  - Which relationships and interfaces are required?
- Application of patterns
  - Which patterns apply to components and their relations?
- Evaluation of the architecture
  - How well does it fit to meet scenarios?

A well-defined and interative Process

- Analysis of requirements and resolution of conflicts
- Application of principles & tactics & patterns
- Taking decisions & making compromises
- Evaluation of alternatives
- Documentation of views for stakeholders
- Documentation and evaluation of decisions

Facts

- Architecture as the key factor defining the quality of a system – WHAT is realized HOW?
- Every system has an architecture
  - even if there is no documentation beyond the code

Challenges

- Technological mismatch between conception and implementation
  - Modeling vs. programming
- Architecture is difficult to reconstruct from the implementation code

Summary

- Architecture as a structural system of components, their relationships, and responsibilites
- Why do specific components and relations exist?
- Architectural thinking asks different questions than design and leads us to different answers
- Architecture develops in an incremental process
- Architectural thinking gives you a plan to control project risks!

Working Questions

1. How do you define the architecture of a software system?
2. Which elements and essential steps constitute architectural thinking?
3. Why is complexity reduction the most important goal when designing an architecture?
4. Why is a structured, iterative, and incremental approach necessary?
5. How can a Scrum team deal with architectural questions?
6. Which subdisciplines of software architecture do you know?
7. Which challenges are we facing when developing the architecture of a system?