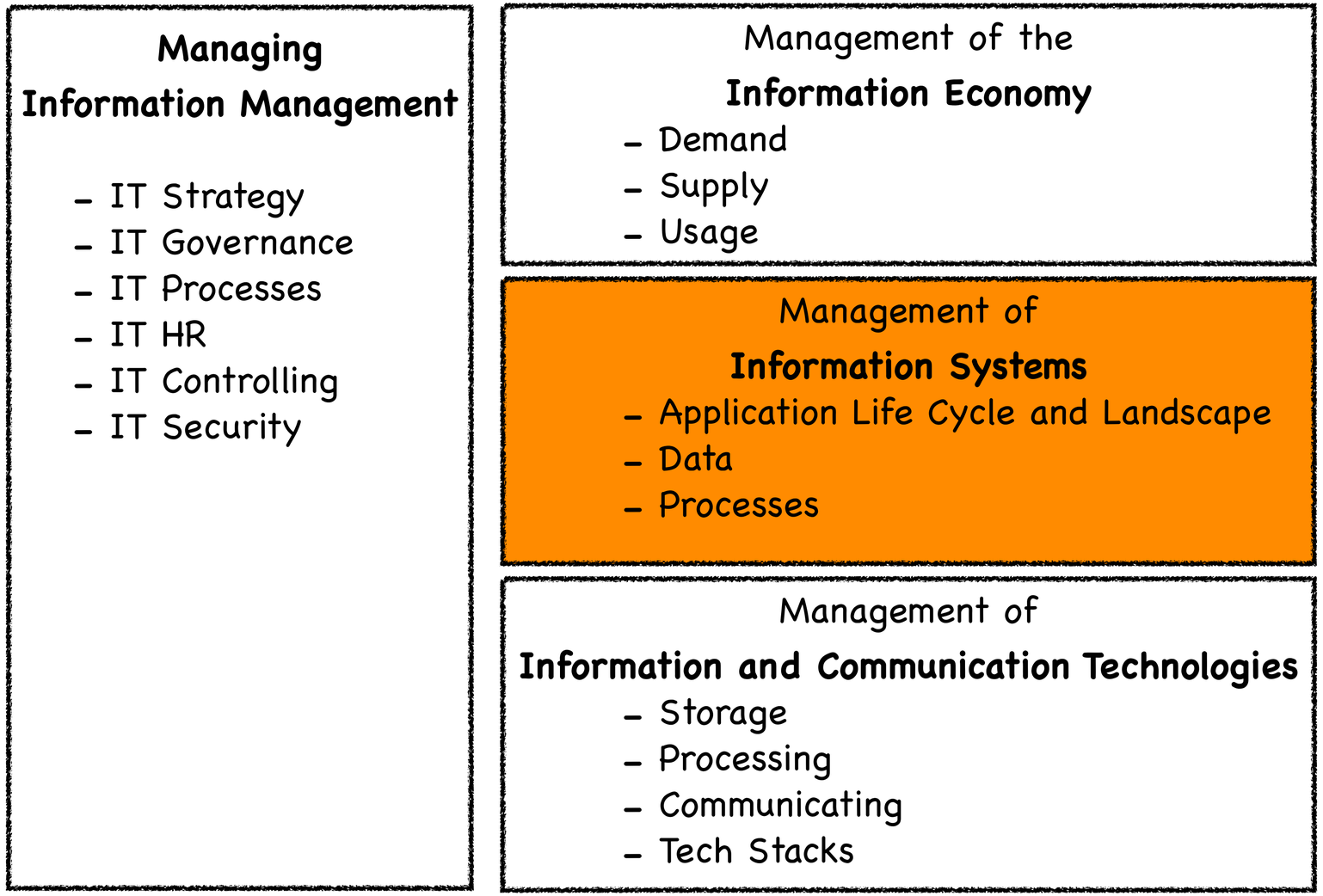
Chapter 3: Management of Information Systems

Topics: Software Engineering Lifecycles - Sourcing Models - Communicating Behavior - Communicating Concepts & Structure

Fachhochschule Südwestfalen
University of Applied Sciences

Information Management



Krcmar, Helmut. "Informationsmanagement." Informationsmanagement. Springer Gabler, Berlin, Heidelberg, 2015, p.94.

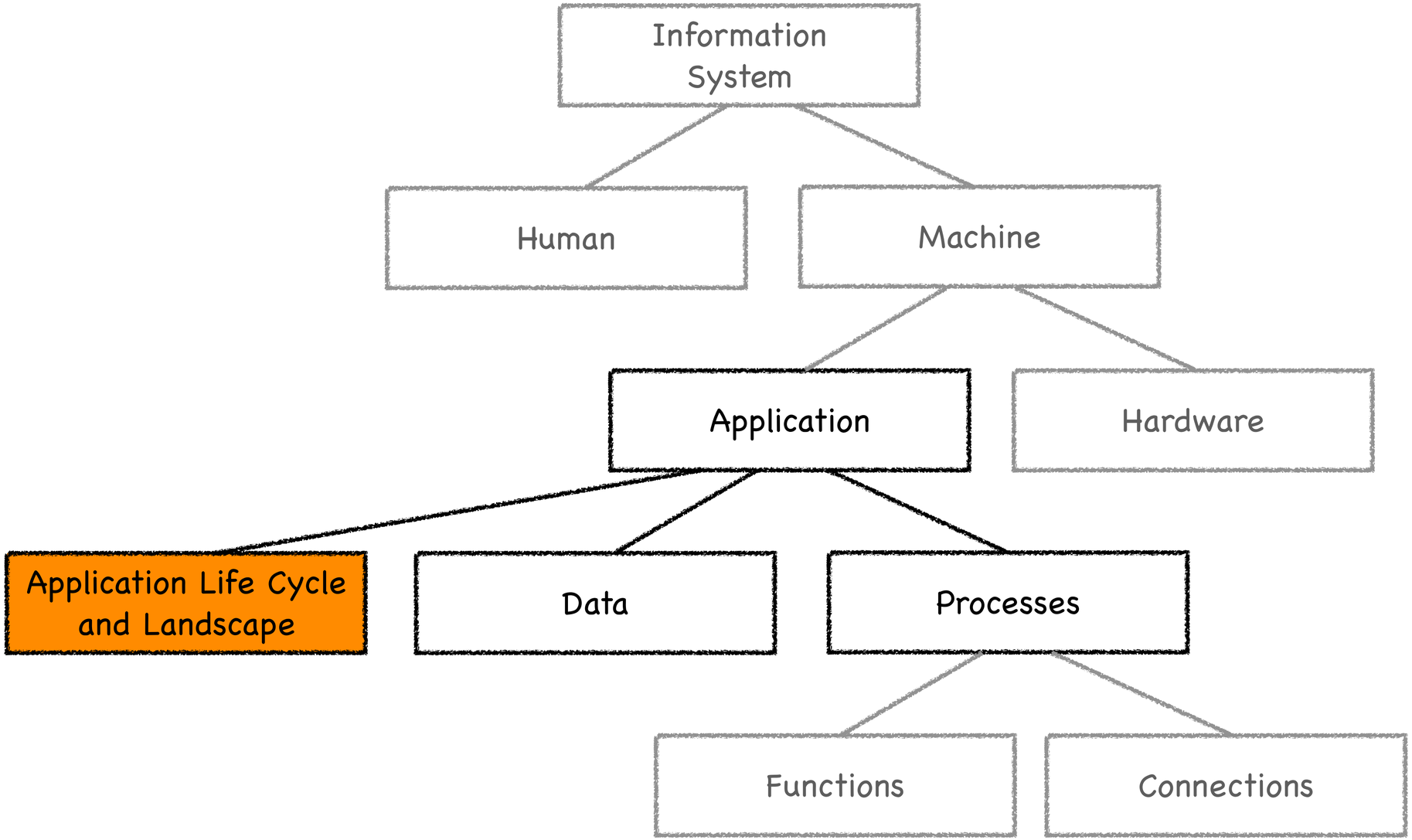
For your case, identify a central software system.

Chapter 3.1: Management of Information Systems - Application Life Cycle

Topics: Project vs Service Model - Software Engineering Lifecycle Models - Sourcing Models

- **Scrum**
- **Kanban**
- **SAFe**

Fachhochschule Südwestfalen
University of Applied Sciences


Information System
Human
Machine
Application
Hardware
Application Life Cycle and Landscape
Data
Processes
Functions
Connections

Three fundamental SE process decisions that every CIO and IT manager needs to make:

1. Project *or* Service
2. Waterfall *or* incremental-iterative
3. Inhouse *or* outsource (offshore *or* nearshore)

Question 1:
Project or Service

Project vs. Service

- Defined goal
- Defined time span
- (usually run once)

DIN69901

- Defined goal
- Defined target group
- Running indefinitely

Project vs. Service

Fundamental Differences

- Project Manager vs. Service Owner
- Focus on project lifetime vs. Long-term vision
- Projects need justification every fixed period
- Projects a great for hypothesis testing (validations, prototypes etc.) but tend to be not very good at knowledge transfer and documentation across project lifetime

Choose your fighter:

Virtuelles Krankenhaus VKh.NRW

URBANMAKER

WAELZHOLZ

?

N.B.: Waterfall
= Sequencial

Question 2:
Waterfall *or* Iterative-Incremental?

Waterfall - What it does **not** mean...

Waterfall - Where the name came from...

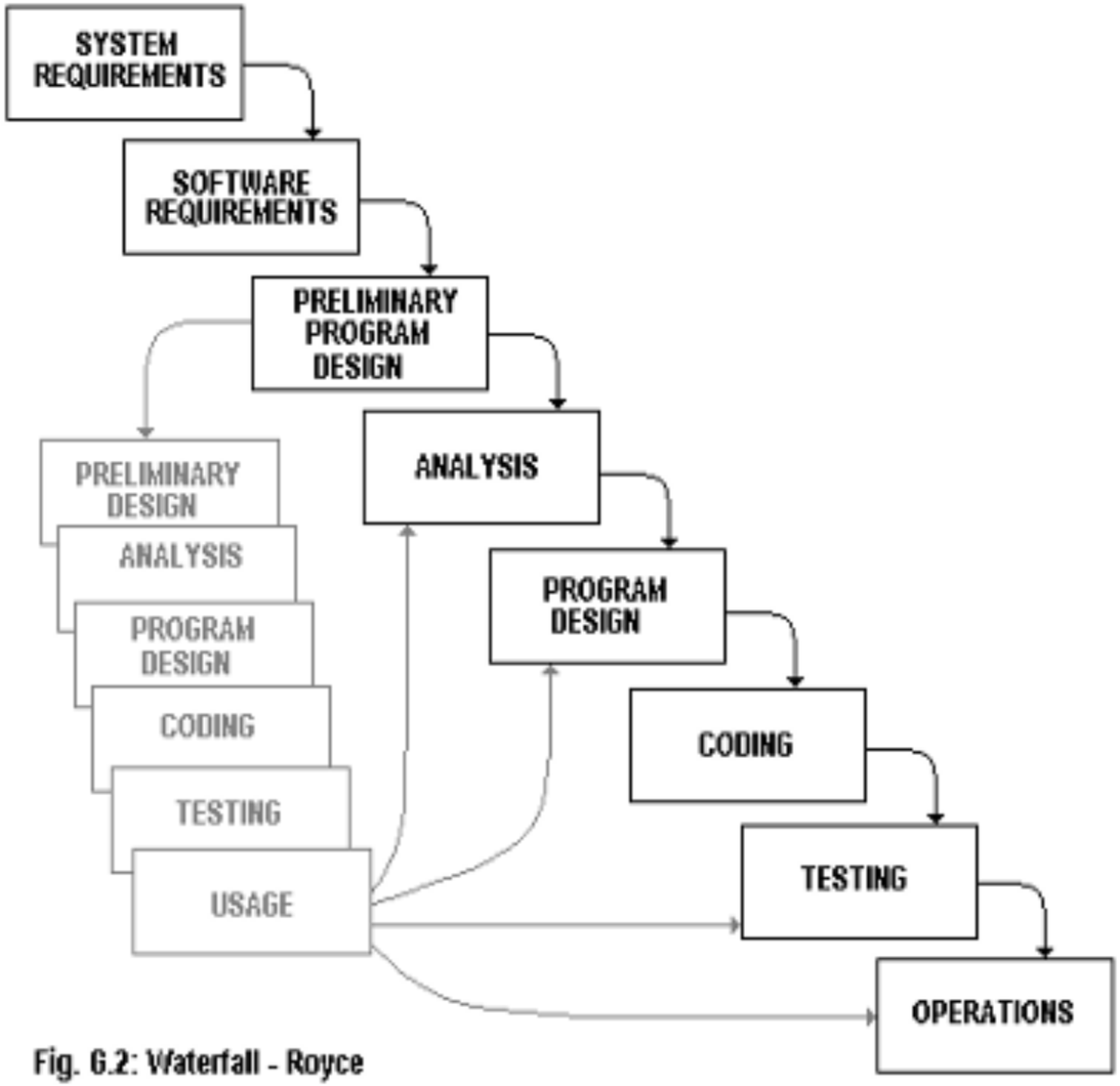
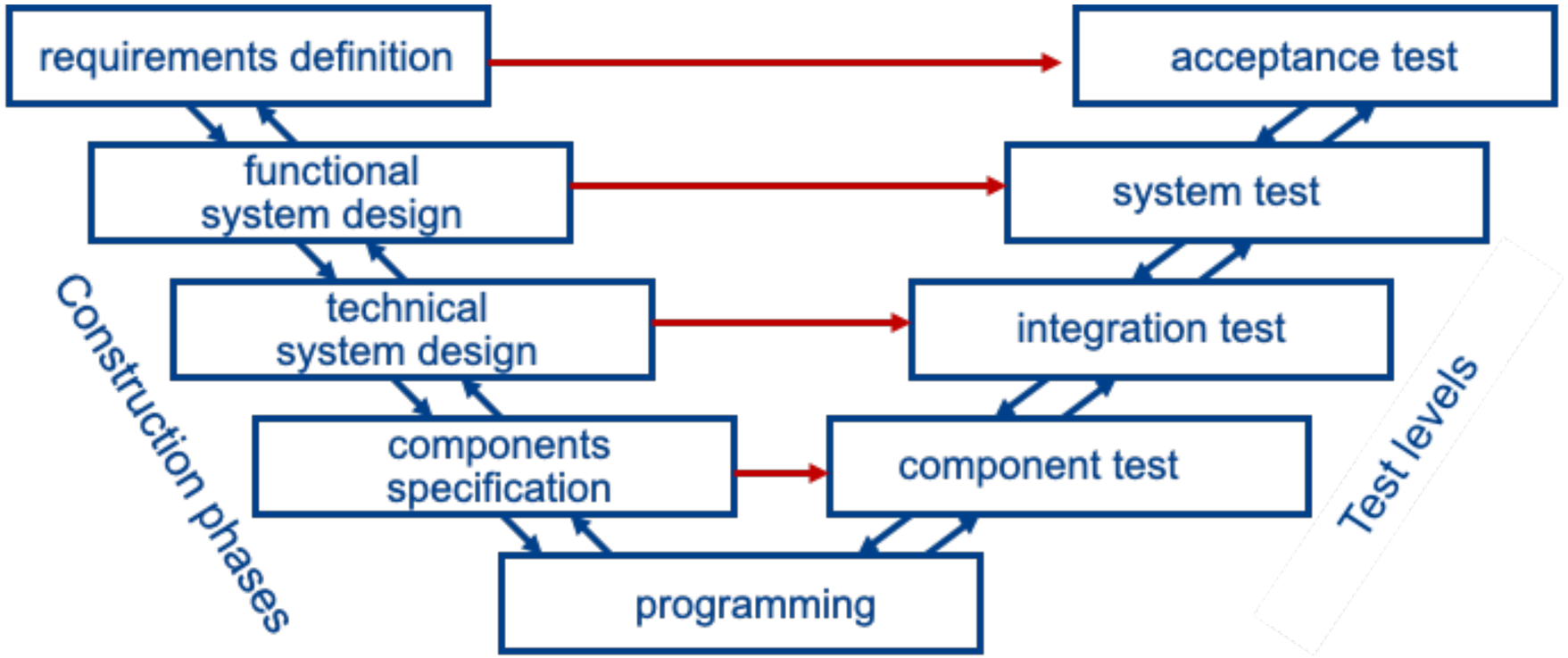


Fig. 6.2: Waterfall - Royce

Royce, W. W.:
Managing the Development of Large Software Systems: Concepts and Techniques
Proc. WESCON, IEEE Computer Society Press, Los Alamitos, CA, 1970.

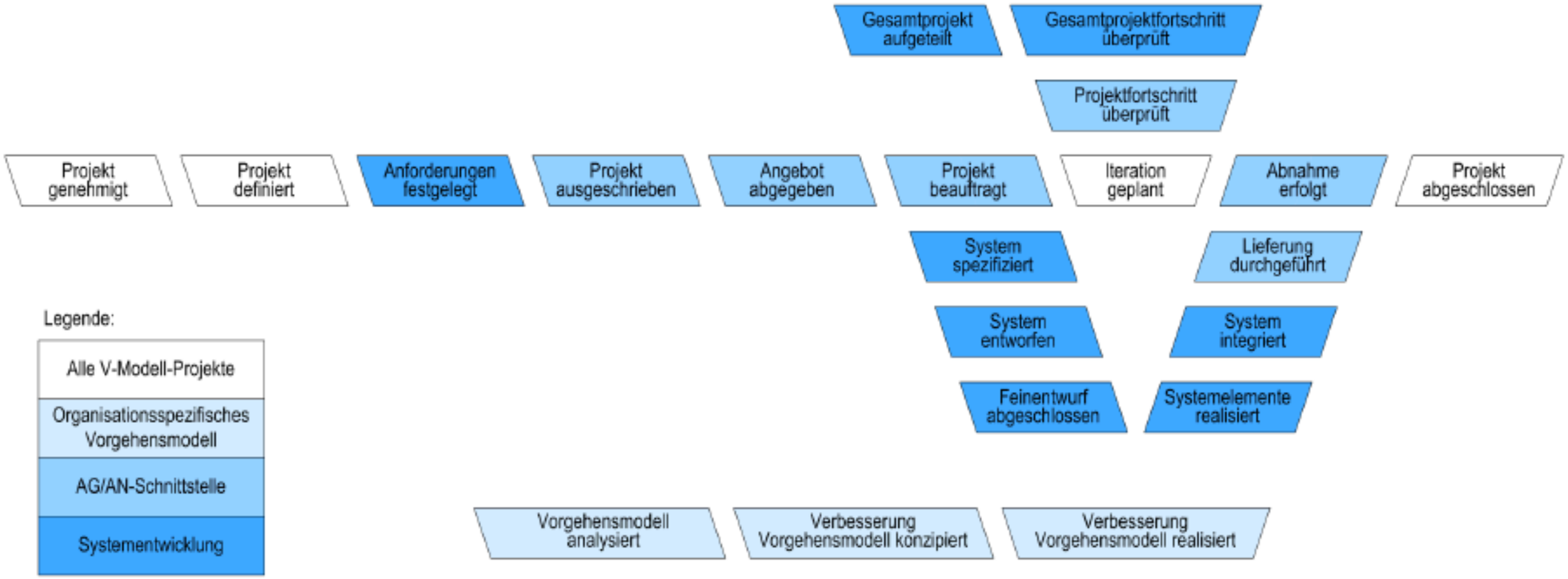
Reprinted at the ICSE'87, Monterey, California, USA. March 30 - April 2, 1987

V-Model


requirements definition
acceptance test
functional
system design
system test
technical
system design
integration test
components
specification
component test
programming
Construction phases
Test levels

V-Modell XT (since 2005)

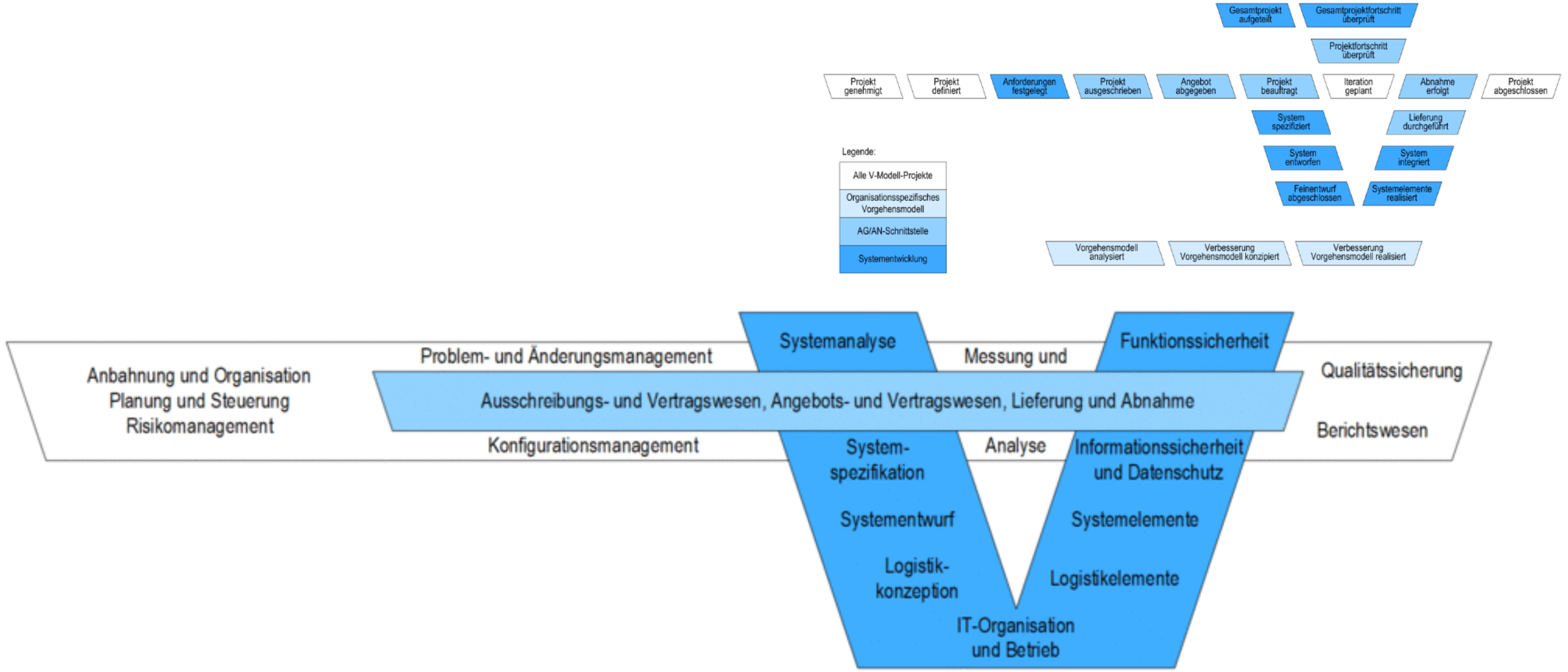
XT = eXtreme Tailoring



https://www.cio.bund.de/Web/DE/Architekturen-und-Standards/V-Modell-XT/vmodell_xt_node.html

V-Modell XT (since 2005)

XT = eXtreme Tailoring



https://www.cio.bund.de/Web/DE/Architekturen-und-Standards/V-Modell-XT/vmodell_xt_node.html

Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

Kent Beck
Mike Beedle

James Grenning
Jim Highsmith

Robert C. Martin
Steve Mellor

https://agilemanifesto.org

Want to Know More? Listen to "Uncle Bob" Martin telling his version of the creation of the agile manifesto.
https://www.retrotimepodcast.com/23-a-brief-history-of-the-agile-manifesto-with-uncle-bob-martin/

Principles behind the Agile Manifesto

We follow these principles:

Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.

Business people and developers must work together daily throughout the project.

Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

Working software is the primary measure of progress.

Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.

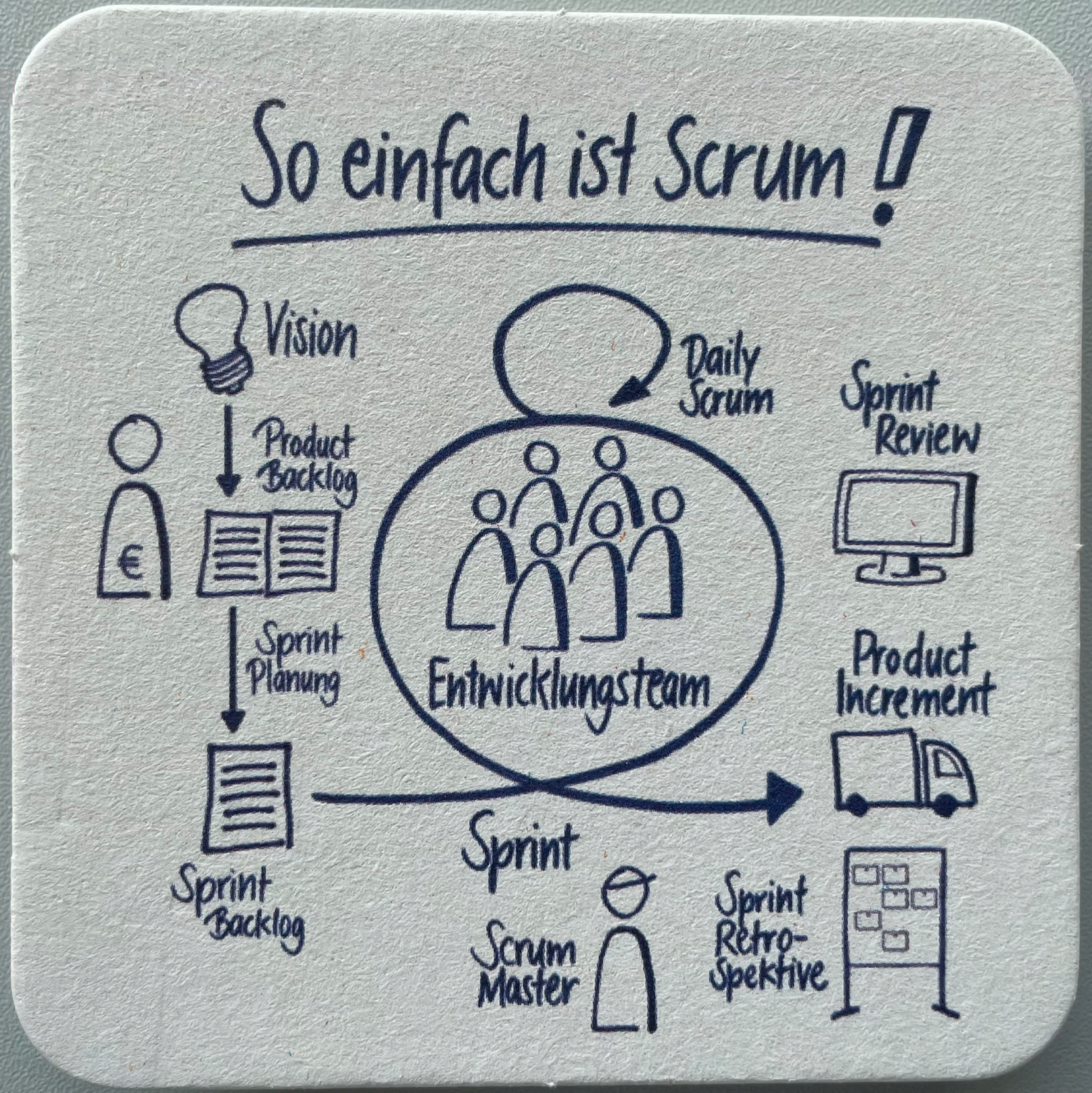
Continuous attention to technical excellence and good design enhances agility.

Simplicity--the art of maximizing the amount of work not done--is essential.

The best architectures, requirements, and designs emerge from self-organizing teams.

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

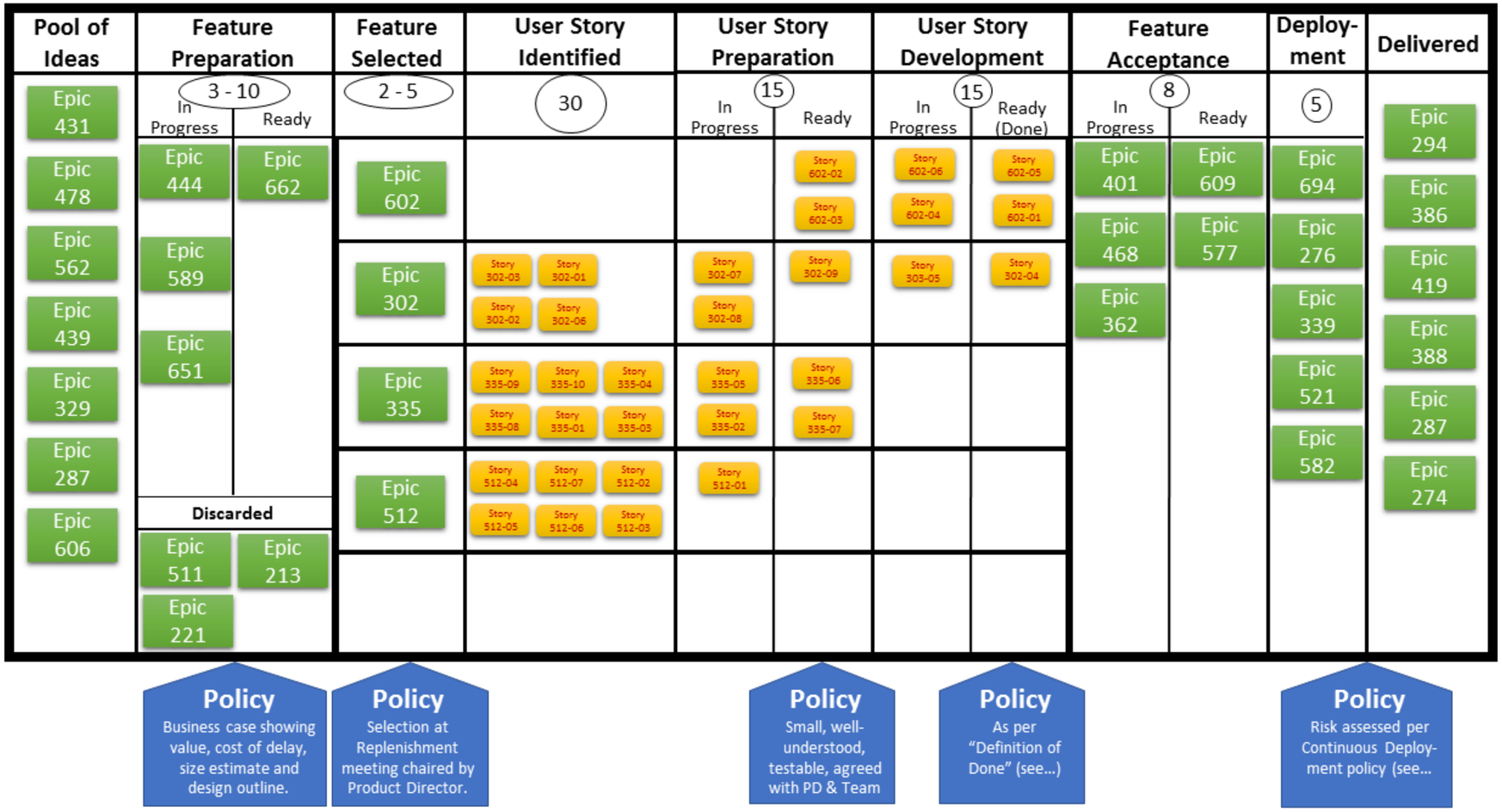
Return to Manifesto



https://www.it-agile.de/fileadmin/PDF/Scrum_Intro.pdf

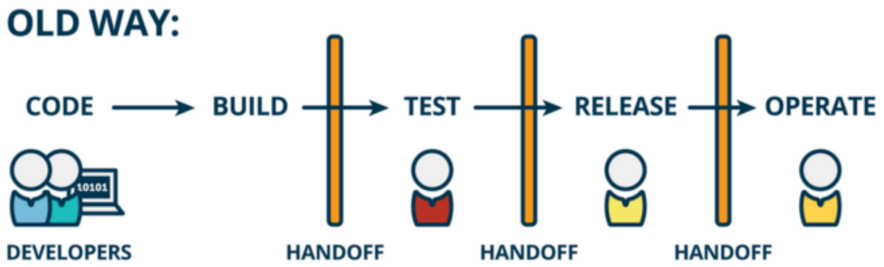
Kanban (看板, meaning signboard or billboard)

- History: Toyota (1947): Supermarket principle: Pick an item for each task, identify and improve "empty shelves"
- David J. Anderson (Microsoft) brought Kanban to software development
- Core principle: Limited WIP (work in progress) increases throughput time
- Fantastic for process improvement & service maintenance



SAFe® in 5 Minutes

With Inbar Oren, SAFe Fellow



DevOps

DevOps is a collaborative and multidisciplinary effort within an organization to automate continuous delivery of new software versions, while guaranteeing their correctness and reliability.

Leite, Leonardo, et al. "A survey of DevOps concepts and challenges." ACM Computing Surveys (CSUR) 52.6 (2019): 1-35.

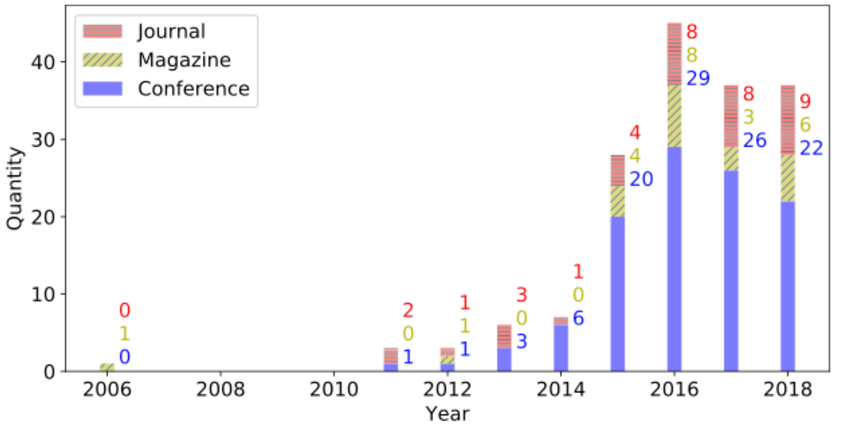


Fig. 2. Publications by source types and publication year.

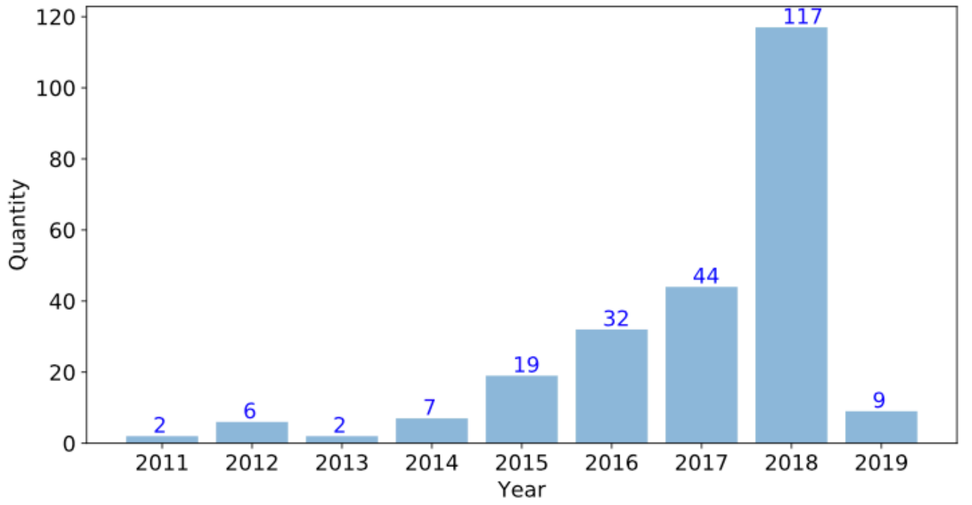
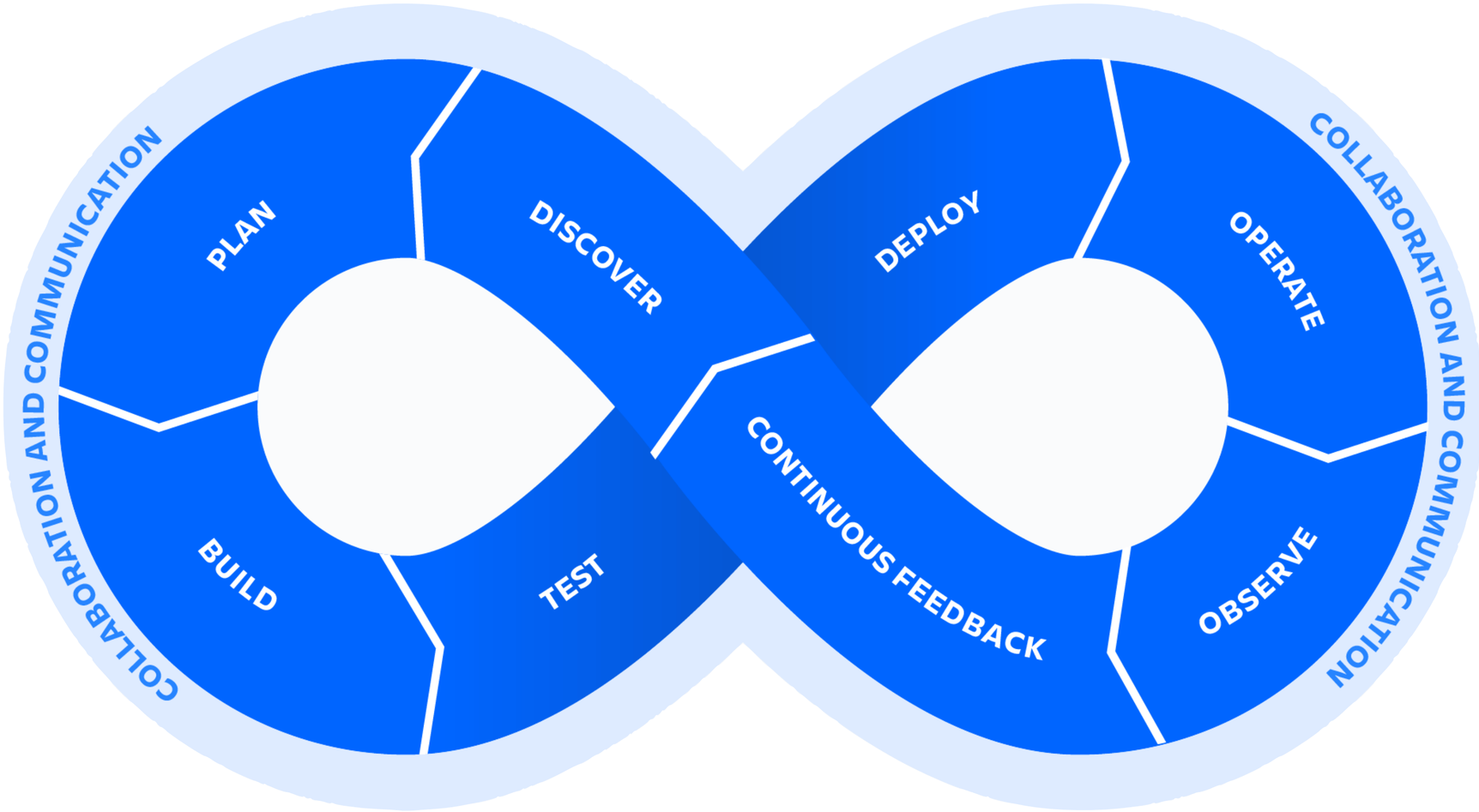


Fig. 3. Distribution of books about DevOps by publication year according to Amazon.com.

https://victoria-guido.medium.com/devops-101-55acdbd3067

Leite, Leonardo, et al. "A survey of DevOps concepts and challenges." ACM Computing Surveys (CSUR) 52.6 (2019): 1-35.



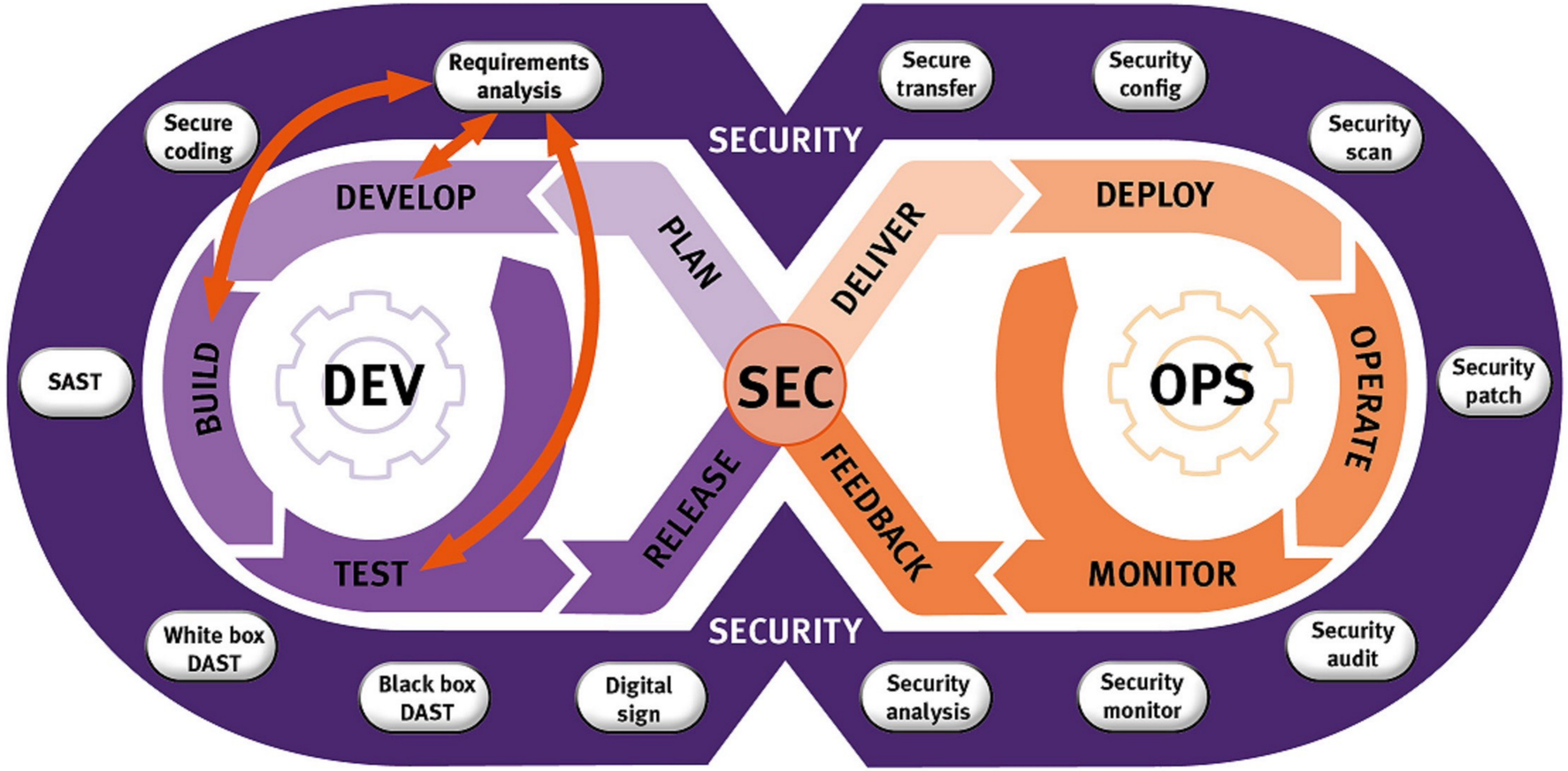
https://www.atlassian.com/de/devops

Table 4. Major DevOps Tools Related to Actors and DevOps Concepts

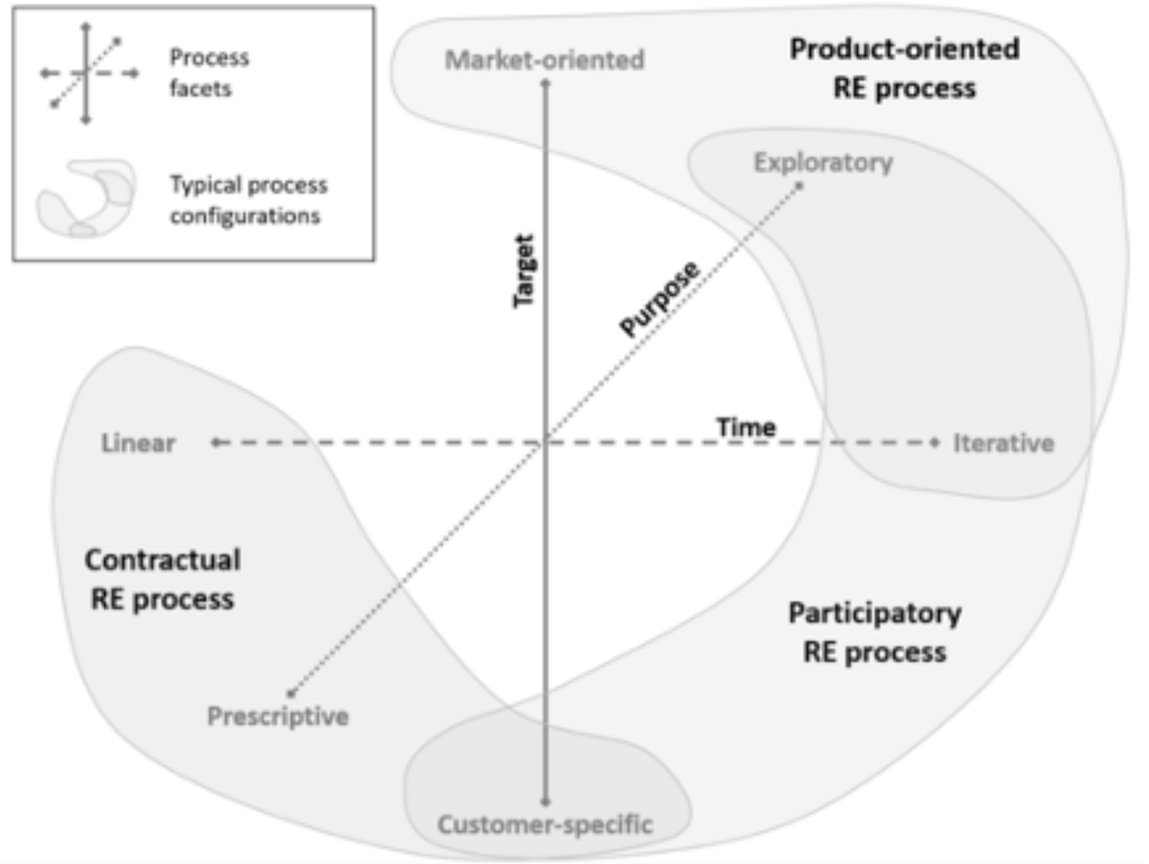
Category	*Examples*	*Actors*	*Goals*	*Concepts*
Knowledge sharing	Rocket Chat GitLab wiki Redmine Trello	Everyone	Human collaboration	Culture of collaboration Sharing knowledge Breaking down silos Collaborate across departments
Source code management	Git SVN CVS ClearCase	Dev/Ops	Human collaboration Continuous delivery	Versioning Culture of collaboration Sharing knowledge Breaking down silos Collaborate across departments
Build process	Maven Gradle Rake JUnit Sonar	Dev	Continuous delivery	Release engineering Continuous delivery Automation Testing automation, Correctness Static analysis
Continuous Integration	Jenkins GitLab CI Travis Nexus	Dev/Ops	Continuous delivery	Frequent and reliable release process Release engineering Continuous integration Deployment pipeline Continuous delivery, Automation Artifact management
Deployment automation	Chef, Puppet Docker Heroku Open Stack AWS Cloud Formation Rancher Flyway	Dev/Ops	Continuous delivery Reliability	Frequent and reliable release process Release engineering Configuration management Continuous delivery Infrastructure as code Virtualization, Containerization Cloud services, Automation
Monitoring & Logging	Nagios Zabbix Prometheus Logstash Graylog	Ops/Dev	Reliability	You built it, you run it After-hours support for Devs Continuous runtime monitoring Performance, Availability, Scalability Resilience, Reliability, Automation Metrics, Alerting, Experiments Log management, Security

Leite, Leonardo, et al. "A survey of DevOps concepts and challenges." ACM Computing Surveys (CSUR) 52.6 (2019): 1-35.

DevSecOps



https://www.embedded.com/devsecops-brings-defense-in-depth-to-embedded-security/



Glinz, Martin, et al. "Handbook for the CPRE Foundation Level according to the IREB Standard." International Requirements Engineering Board (2020).

What are the main differences between traditional and agile engineering?

Highly generalized, where should you apply what?

Choose your fighter:

Virtuelles Krankenhaus VKh.NRW

URBANMAKER

WAELZHOLZ

?

Question 3:
Inhouse *or* Outsource?

Options and Definitions

- Outsource vs Inhouse
- Offshore vs. Onshore vs. Nearshore

Outsourcing the process of paying to have part of a company's work done by another company

Cambridge Dictionary

Offshoring the practice of basing a business or part of a business in a different country, usually because this involves paying less tax or other costs

Cambridge Dictionary

Nearshoring is relocation of business processes to (typically) lower cost foreign locations, but in close geographical proximity

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

Which advantages would you expect from offshoring?

Which challenges do you see?

Reality Check: Outsourcing benefits

Expectations and Achievements: A Longitudinal Study on an Offshoring Strategy

Darja Šmite
Blekinge Institute of Technology
Karlskrona, Sweden
darja.smite@bth.se

Daniela S. Cruzes
NTNU, SINTEF
Trondheim, Norway
dcruzes@idi.ntnu.no

***Abstract*—Offshore software development has gained momentum and most of software companies today have implemented offshore strategies of some sort. Many of these strategies are enforced by corporate top management and driven by assumptions that lower development wages guarantee cheaper and better software development. In practice, offshore software development is associated with many risks, and achievement of the expected benefits is not as straightforward as the rumor has it. In this paper we explore an implementation of an offshore strategy in a Swedish software company that opened its offshore branch in Russia. Based on extensive documentation analysis we create an overview of the initially expected benefits and obstacles that prevailed among onshore product and development unit managers. Years after implementation of the offshore insourcing strategy we asked these managers about the achievement of their expectations. We observed that the company documented various expected benefits when implementing an offshoring strategy and also concerns that some of these benefits might not be achieved. Seven years after its implementation, the offshoring strategy was overall considered working, however the expected benefits were not fully achieved. More importantly, several gaps were identified, that suggest that the enforced strategy has resulted in a stable but not beneficial collaboration from the onshore perspective.**

Keywords — Offshoring, Insourcing, Global software development, Expected benefits, Perceived benefits, Concerns, Empirical case study, Longitudinal study

I. Introduction

Contemporary software engineering is trending more and more towards globally distributed collaborations, in which software engineers working on the same project are spread across multiple locations, often thousands of kilometers and several time zones away from each other. The popularity of global software engineering (GSE) is fueled by the rumors of significant cost savings. Software companies are attracted by the perceived benefits of cheaper, faster and better development [1] and hence offshoring today is practiced not only by multinational collaborations, but also by small and medium enterprises [2]. Despite its popularity, a systematic literature review on empirical evidence in global software engineering indicates that there are more problem reports than success stories [3]. An experienced reader of research material in the field, however, would most likely be accustomed to varying opinions ranging from significant success to total failure. For example, in an industry report Burger claims that his company saves several millions of euros per year [4], while lessons learned from a multiple-case study conducted at three Scandinavian companies suggest that first outsourcing contracts in all cases failed due to low productivity, poor quality and other collaboration troubles [2].

One of the reasons for the varied opinions on the phenomenon is that the analysis of success or failure in global software engineering in general is not a straightforward task. Although it is often referred to as one research field, GSE relates to a variety of different types of software development projects [5]. The differentiating attributes of these projects have been captured in a recent taxonomy [6] and include: location (onshore versus offshore), legal entities (outsourcing versus insourcing), geographic distance (close versus distant for onshore projects, and near versus far for offshore projects), as well as temporal distance (similar versus different for onshore projects, and small or large for offshore projects). Notably onshore projects (those distributed within one country) are often included as a type of global projects, while they can hardly qualify to be global in its common sense. Because global projects are enabled through a number of diverse scenarios, it is fair to assume that what works in one context does not necessarily apply in another [7]. Therefore deriving average success rates or cost saving rates from heterogeneous samples, as proposed in [8], is meaningless.

In addition to varying contexts, success or failure of GSE projects can be measured differently. In a systematic review, on dispersion, coordination and performance in global software teams, the authors identified 28 papers on empirical studies in the area [9] and concluded that all of the studies that used perception-based measurement of performance (for example, interviews with developers about performance of global projects) show a negative or neutral impact of geographical and temporal dispersion on performance. While perception-based studies report a pessimistic view of the impact of dispersion on team performance, studies on direct measurement show positive results.

Another essential aspect of measuring success of global software projects is related to the time perspective. Carmel and Tjia [10] attempt to systemize evaluation of economic benefits related to implementing an offshore strategy. Their approach suggests that it takes time for the economic benefits to realize as the "breakeven" is reached when the sum of direct cost savings reaches the sum of initial investments. It is fair to assume that the same rationale applies to other benefits, such as

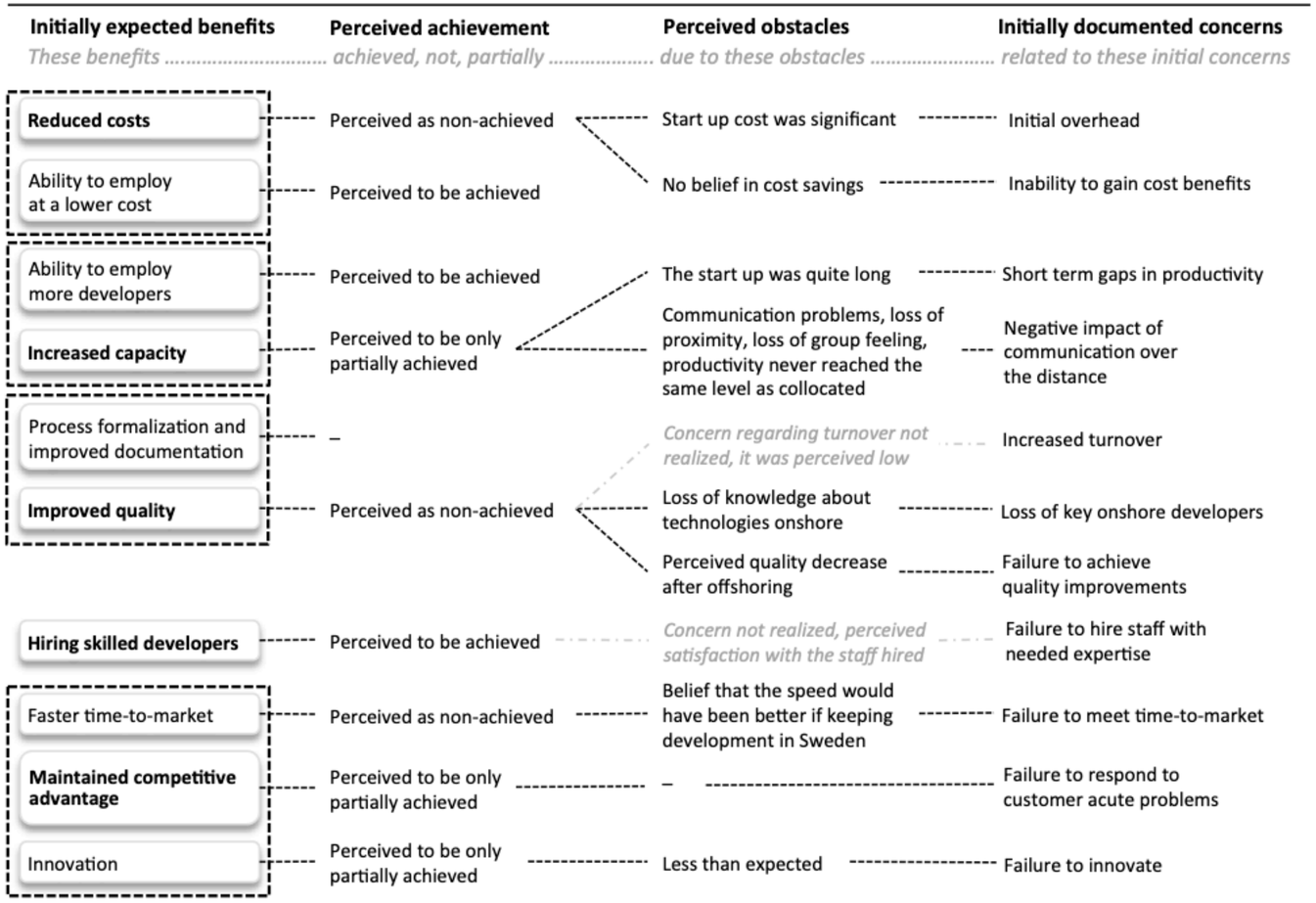


Fig. 4. Comparison of initially documented expectations and their perceived realization

Šmite, Darja, and Daniela S. Cruzes. "Expectations and achievements: a longitudinal study on an offshoring strategy." Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE, 2013.

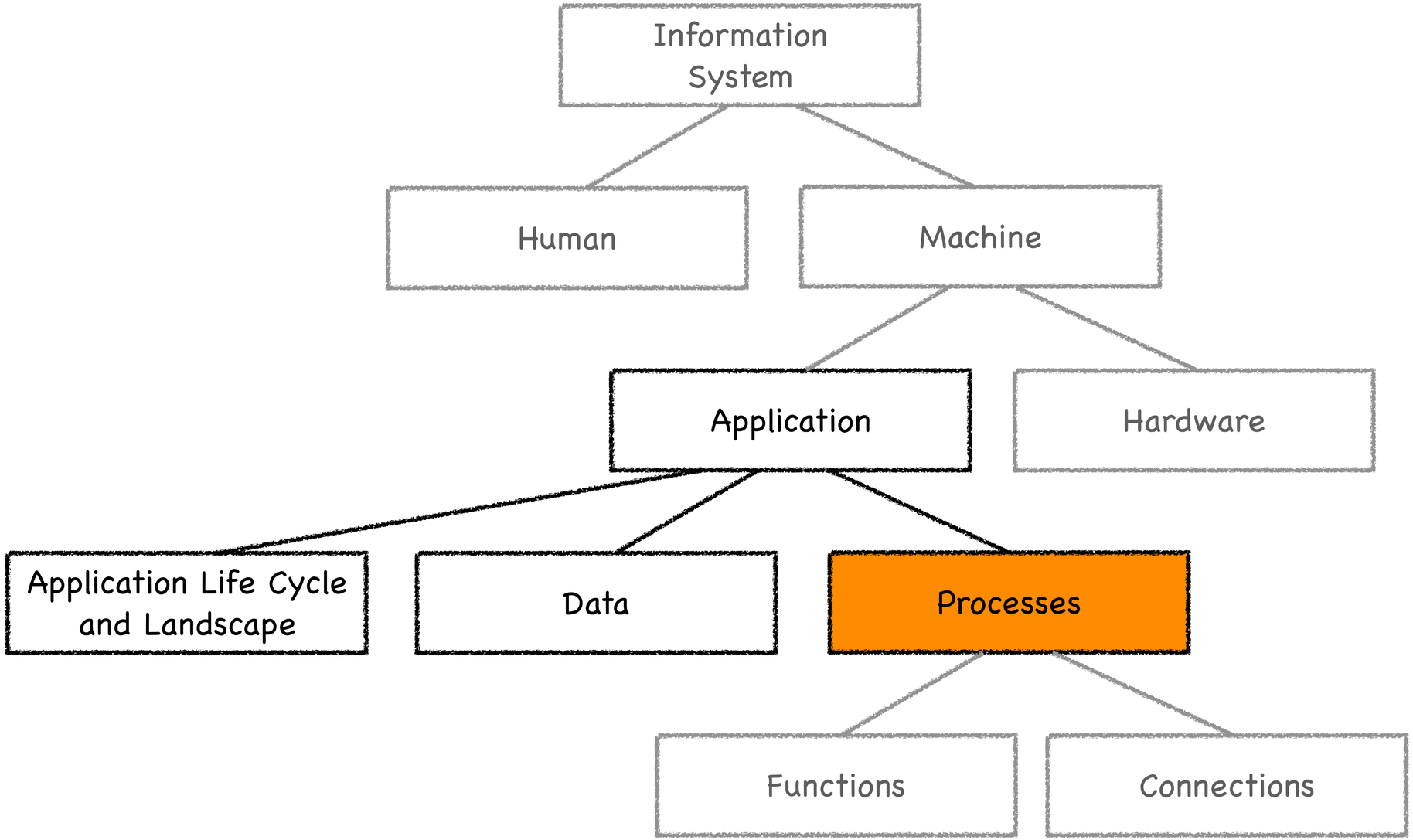
Three fundamental SE process decisions that every CIO and IT manager needs to make:

1. Project *or* Service
2. Waterfall *or* incremental-iterative
3. Inhouse *or* outsource (offshore *or* nearshore)

Chapter 3.2: Management of Information Systems - Processes

Topics: Purposes of Models - History of UML - Modeling Use Cases - Modeling Processes

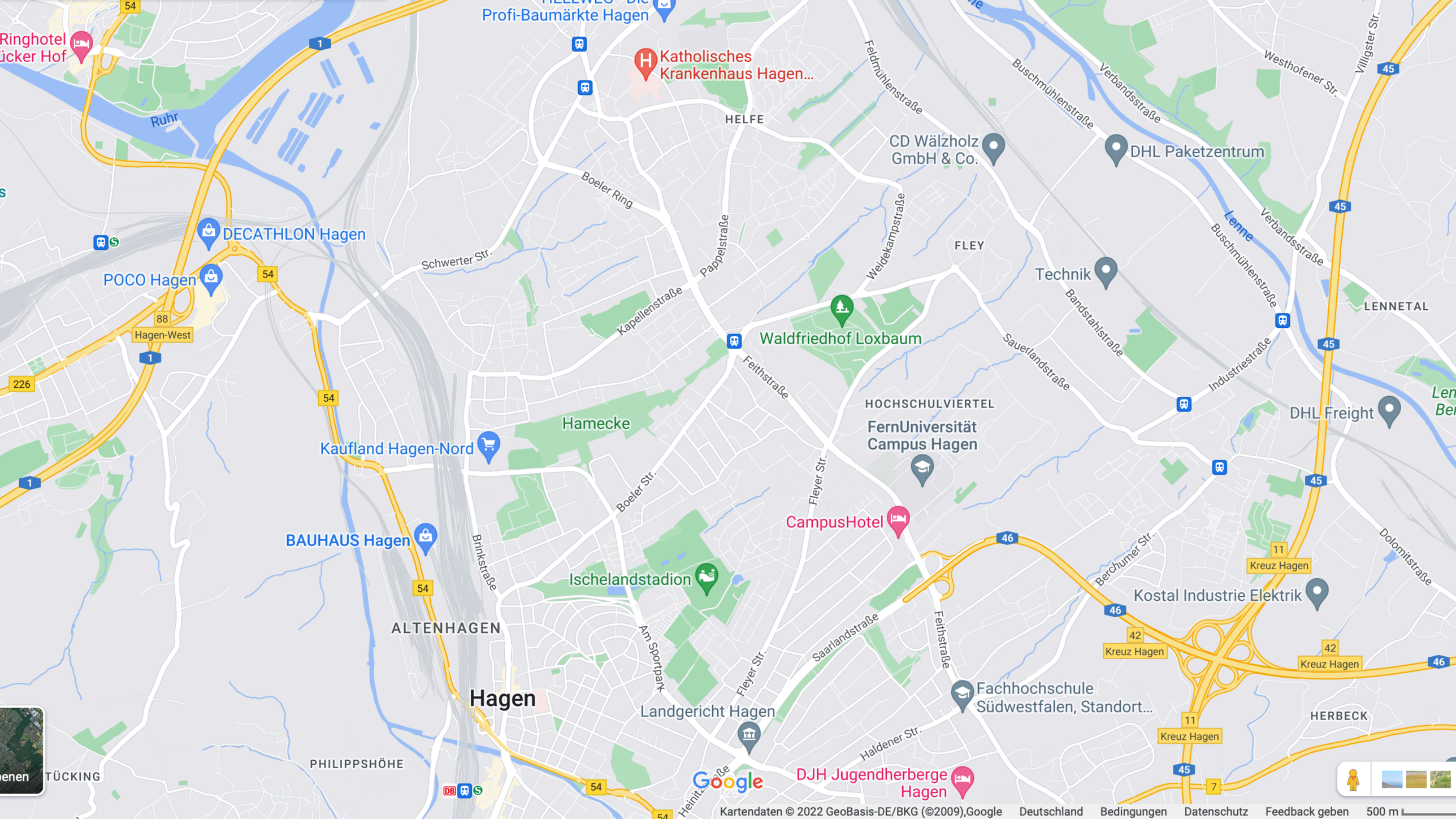
- **UML Use Case Diagrams**
- **UML Activity Diagrams**


Information System
Human
Machine
Application
Hardware
Application Life Cycle and Landscape
Data
Processes
Functions
Connections

Mgmt of Information Systems is often about Communication

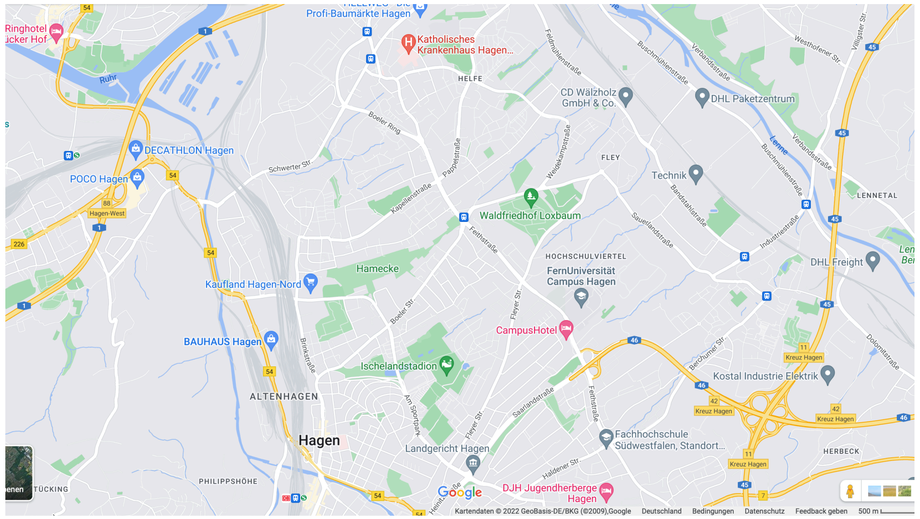
Means for Communication

- *Speak* vs. *Specify*
- *Natural Language* vs. *Models*
  - *A model is an **abstraction** from reality with a **specific purpose** or for a **specific use**.*
  - Models in general capture not all attributes of the original, they filter all attributes except for those relevant to the model creator/user
  - There might be different models for the same part of reality


Ringhotel
ücker Hof
Profi-Baumärkte Hagen
Katholisches
Krankenhaus Hagen...
HELFE
Ruhr
CD Wälzholz
GmbH & Co.
DHL Paketzentrum
Boeler Ring
DECATHLON Hagen
POCO Hagen
Schwerter Str.
Pappelstraße
Weidekampstraße
FLEY
Technik
Lenne
Verbandsstraße
Buschmühlenstraße
LENNETAL
Kapellenstraße
Waldfriedhof Loxbaum
Bandstahlstraße
Sauerlandstraße
Industriestraße
Hagen-West
Feithstraße
HOCHSCHULVIERTEL
DHL Freight
Hamecke
FernUniversität
Campus Hagen
Kaufland Hagen-Nord
Boeler Str.
Fleyer Str.
CampusHotel
BAUHAUS Hagen
Brinkstraße
Ischelandstadion
Berchumer Str.
Kreuz Hagen
Kostal Industrie Elektrik
Dolomitstraße
ALTENHAGEN
Am Sportpark
Saarlandstraße
Hagen
Landgericht Hagen
Fachhochschule
Südwestfalen, Standort...
HERBECK
Haldener Str.
PHILIPPSHÖHE
TÜCKING
Google
DJH Jugendherberge
Hagen
Kartendaten © 2022 GeoBasis-DE/BKG (©2009),Google
Deutschland
Bedingungen
Datenschutz
Feedback geben
500 m

What is a model?

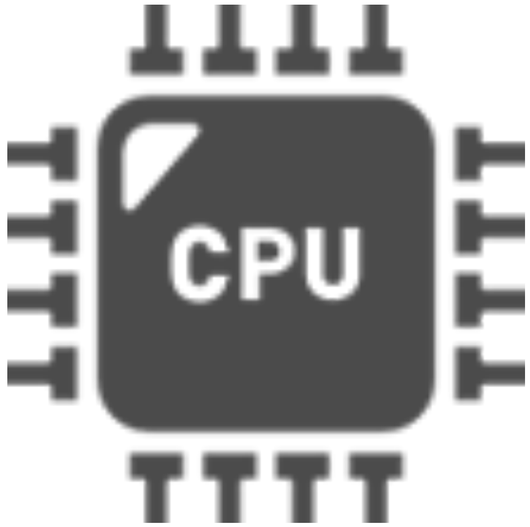
Properties according to Stachowiak

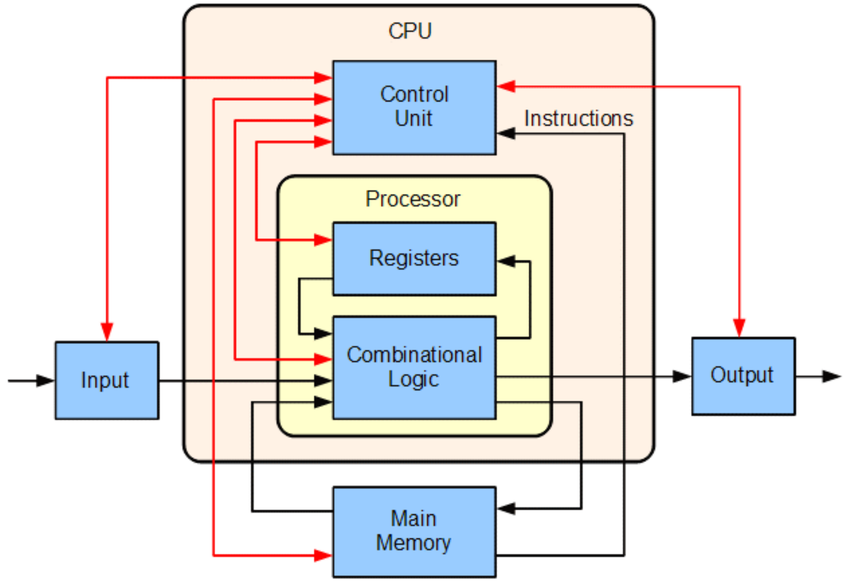


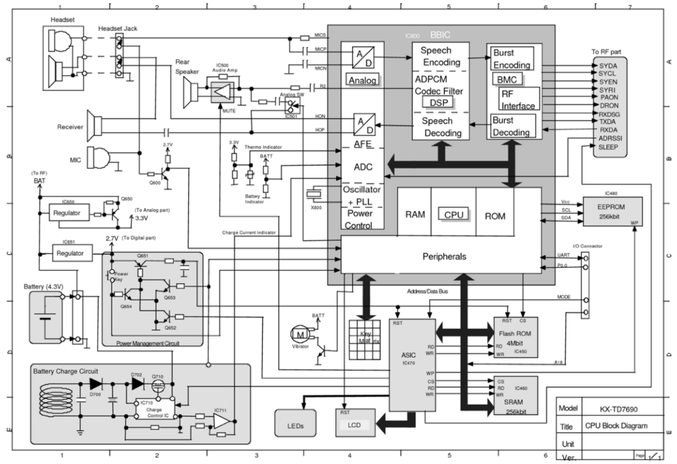
- **Mapping**: Models map the reality
- **Reduction**: Models reduce the represented reality
- **Pragmatics**: Models are constructed for a specific purpose

Herbert Stachowiak: Allgemeine Modelltheorie, 1973
(CC-BY 3.0) Ennepetaler86 via wikimedia.org

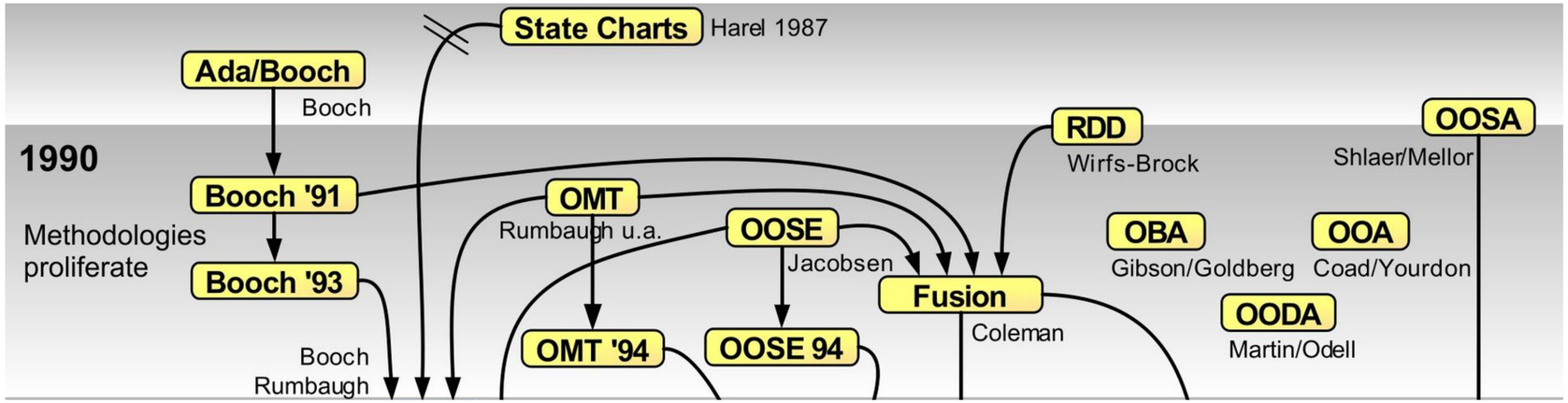
Models and Reality





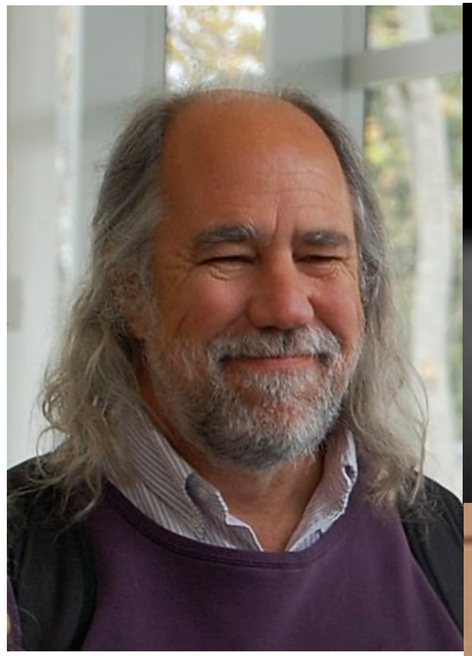


Modeling in Software in the 80ies-90ies



The Unified Modeling Language (UML)

Purpose and History

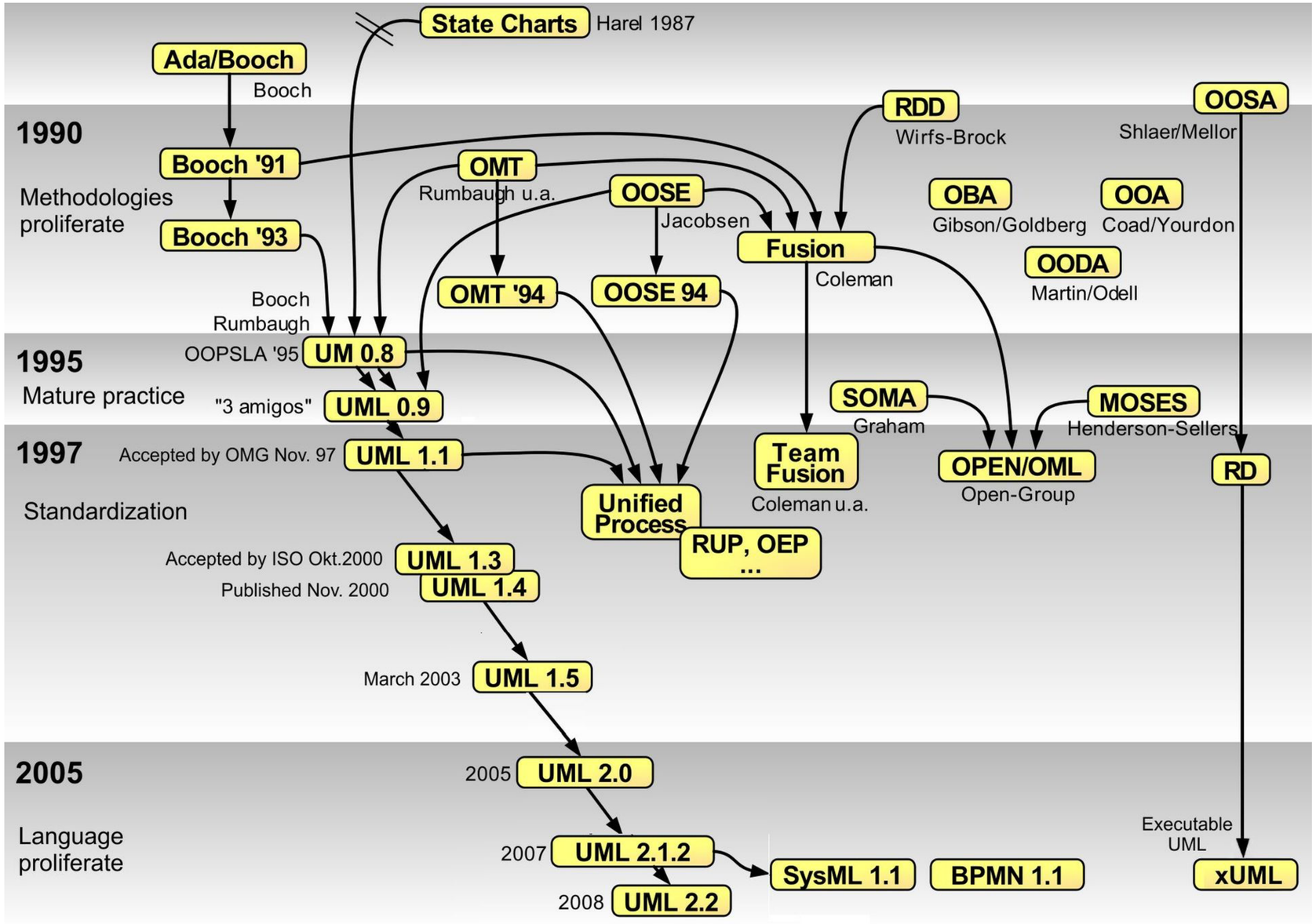


UNIFIED MODELING LANGUAGE™

- Three of the most prominent heads (Booch, Jacobsen, Rumbaugh, "3 Amigos") wanted to create a consistent language
- §Unified Modeling Language (UML) is defined by the Object Management Group (OMG)
- UML provides many diagram types

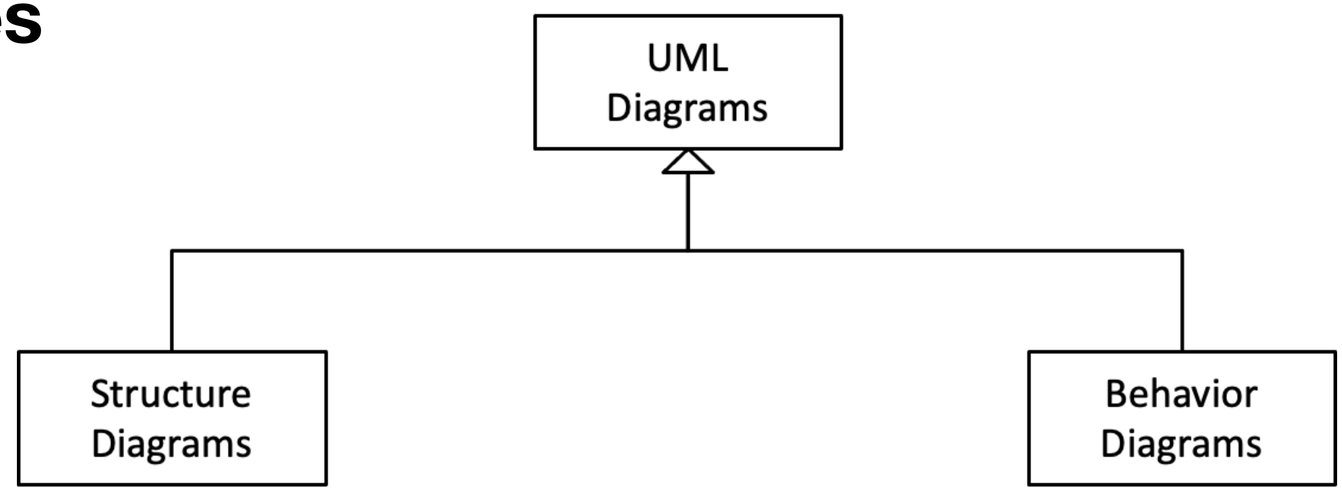
Grady Booch, The Future of Software Engineering. ICSE 2015 Keynote. https://youtu.be/h1TGJJ-F-fE?t=377

Modeling in Software in the 80ies-90ies



The Unified Modeling Language (UML)

Diagram Types

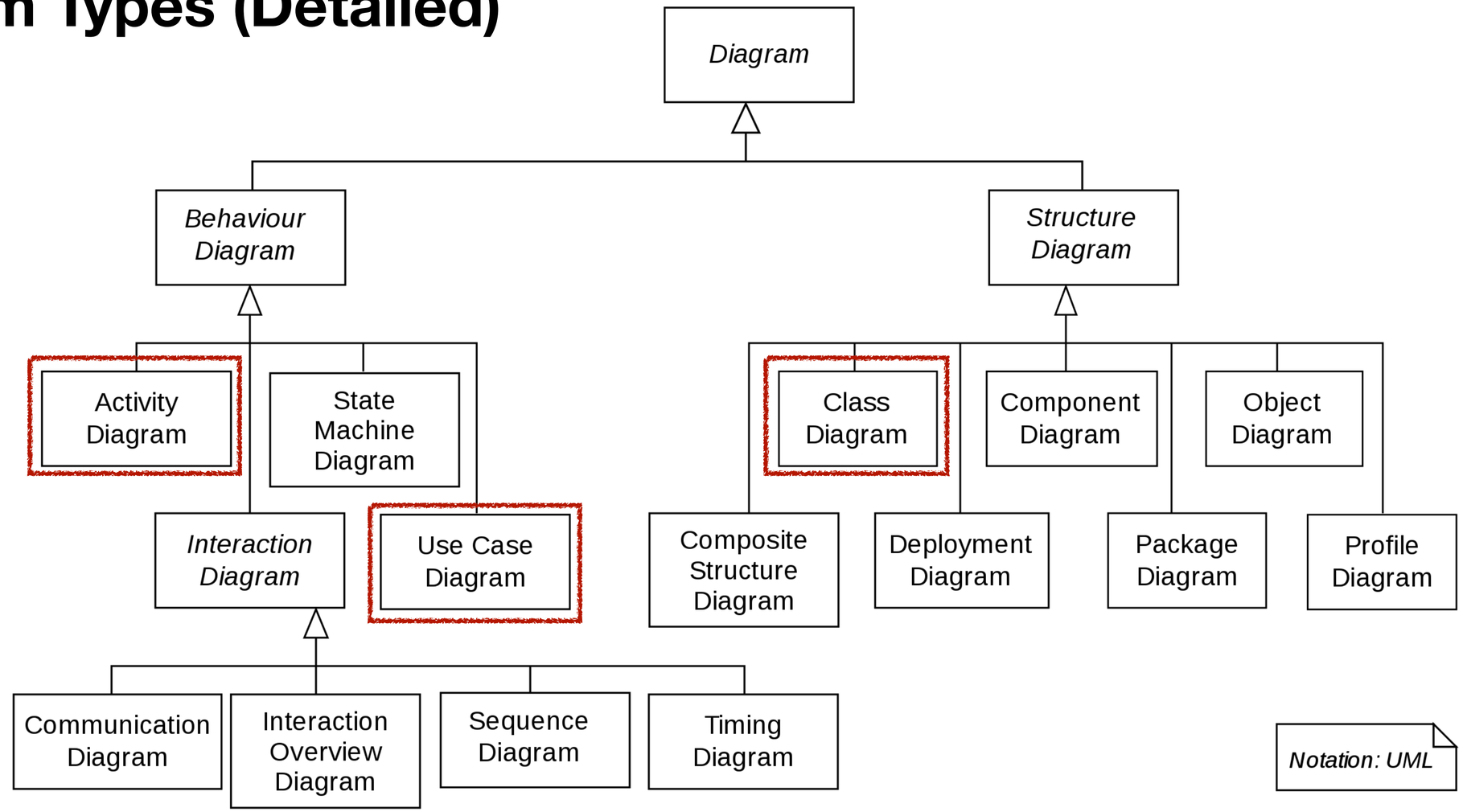


- What are the **elements** that make up our system?
- What are the syntactic **interfaces** between the elements?
- What are possible **configurations** of the system elements?

- Which **stakeholders** and **actors** are involved with the system?
- Which **use cases** and scenarios shall the system support?
- What is the **input/output behavior** of the system (elements)?
- Which **actions** are performed by the system (elements)
- What **states** can the system (elements) be in and when does it transition from one to the next?

The Unified Modeling Language (UML)

Diagram Types (Detailed)

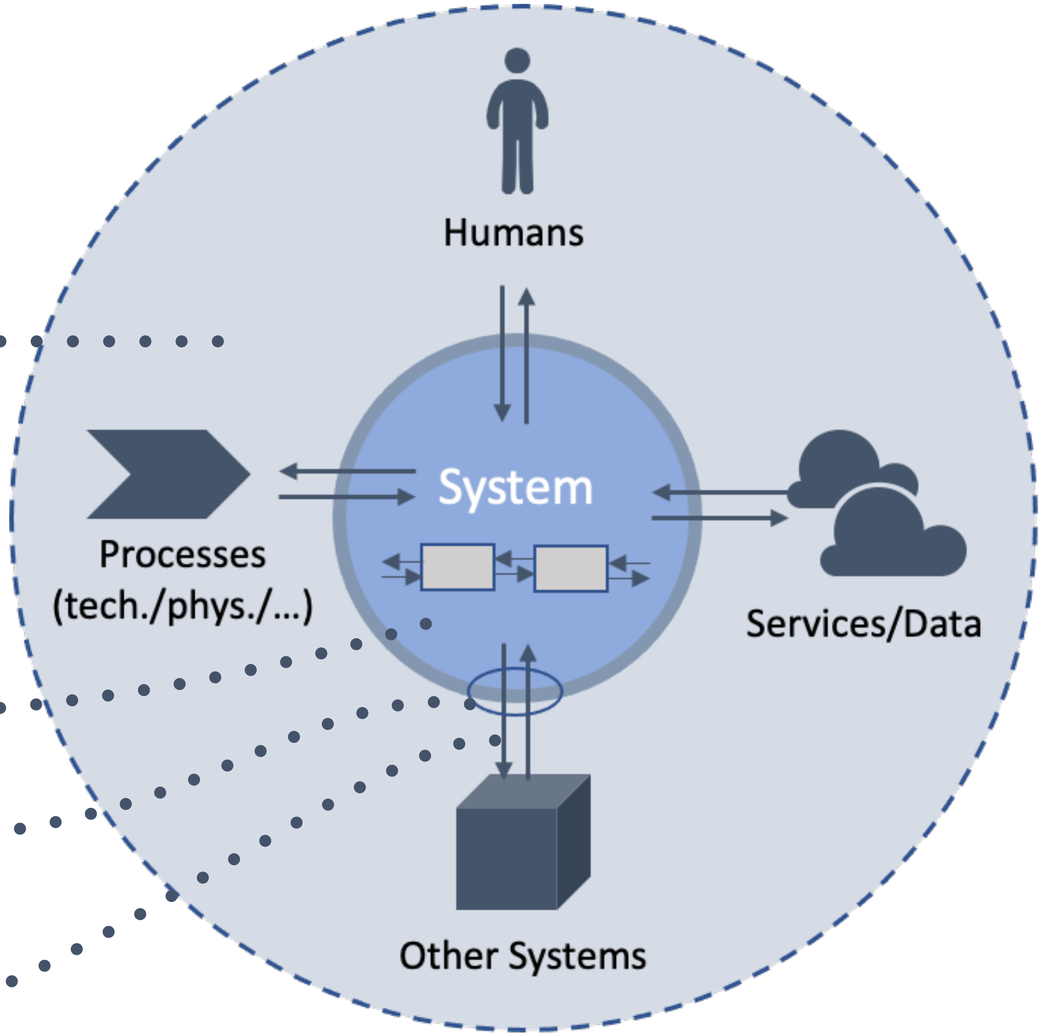


https://upload.wikimedia.org/wikipedia/commons/thumb/e/ed/UML_diagrams_overview.svg/2560px-UML_diagrams_overview.svg.png

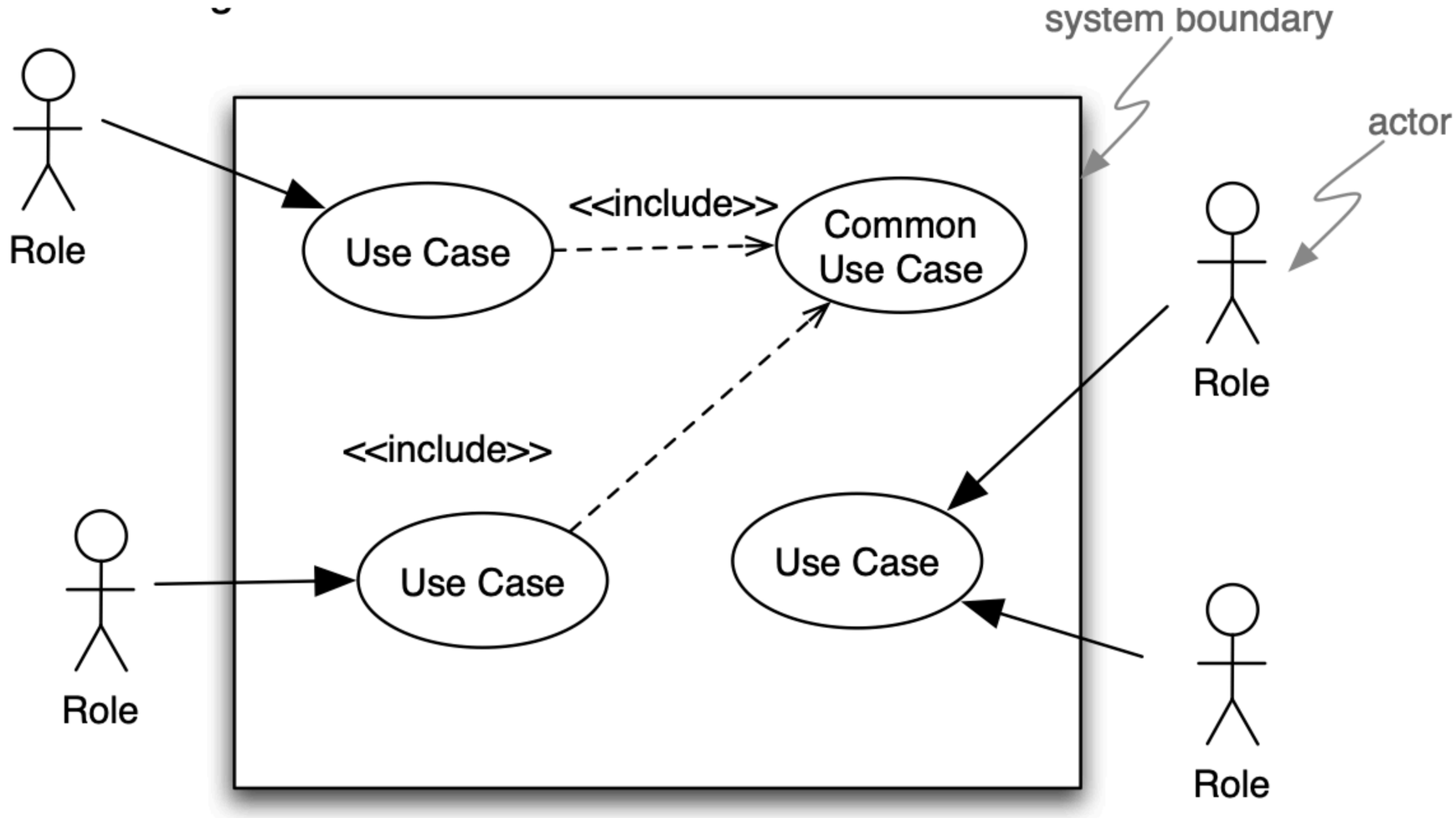
Definition: What's a system?

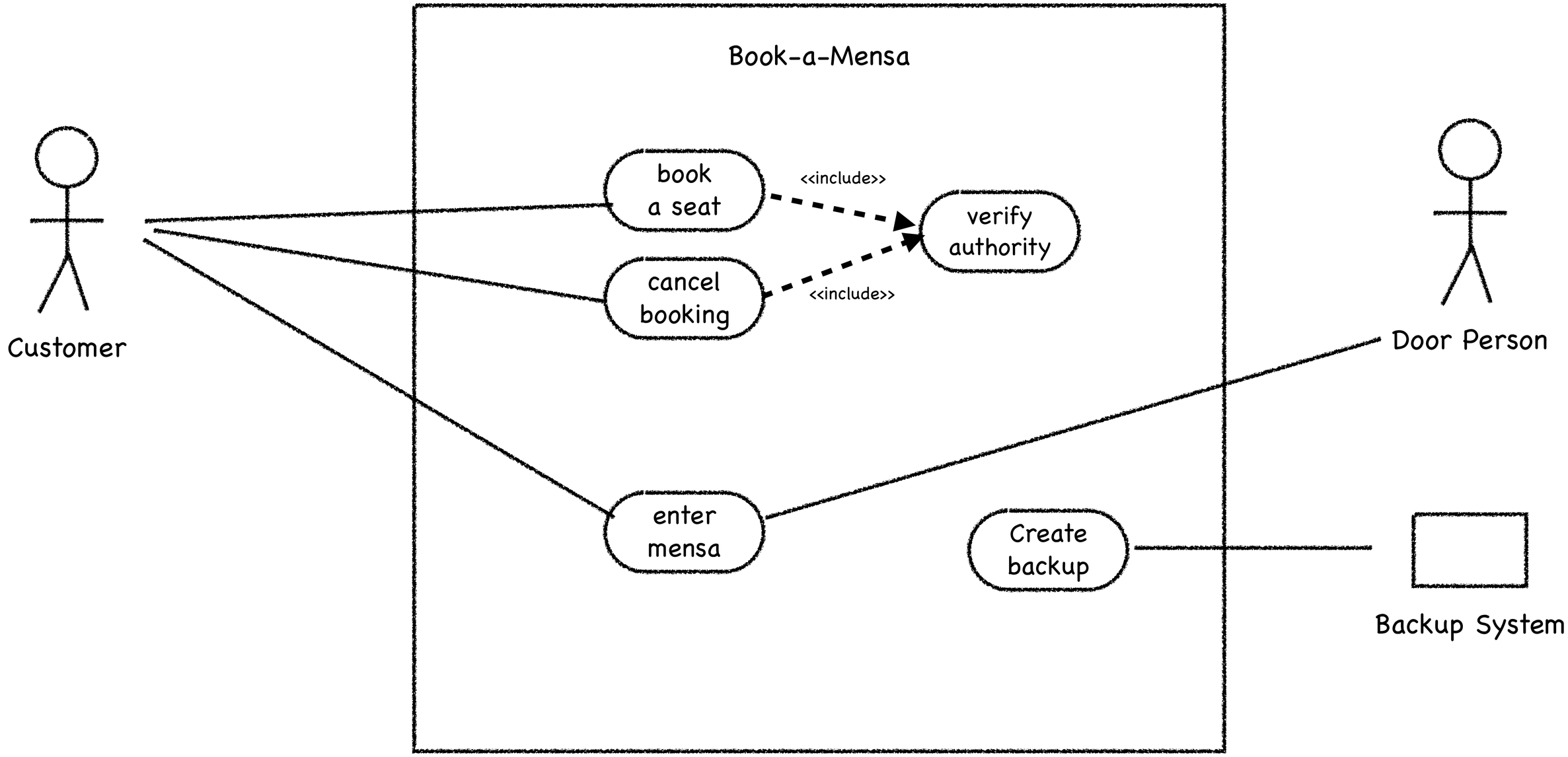
A System (System under Development - SuD) has:

1. An **operational context** that influences or is influenced by the SuD
2. The **context boundary** separates the operational context from non relevant environment
3. An **inner structure** of related elements
4. **Interfaces** that clearly distinguish the SuD from its context and define the system boundary
5. **Behavior** that is observable at these interfaces



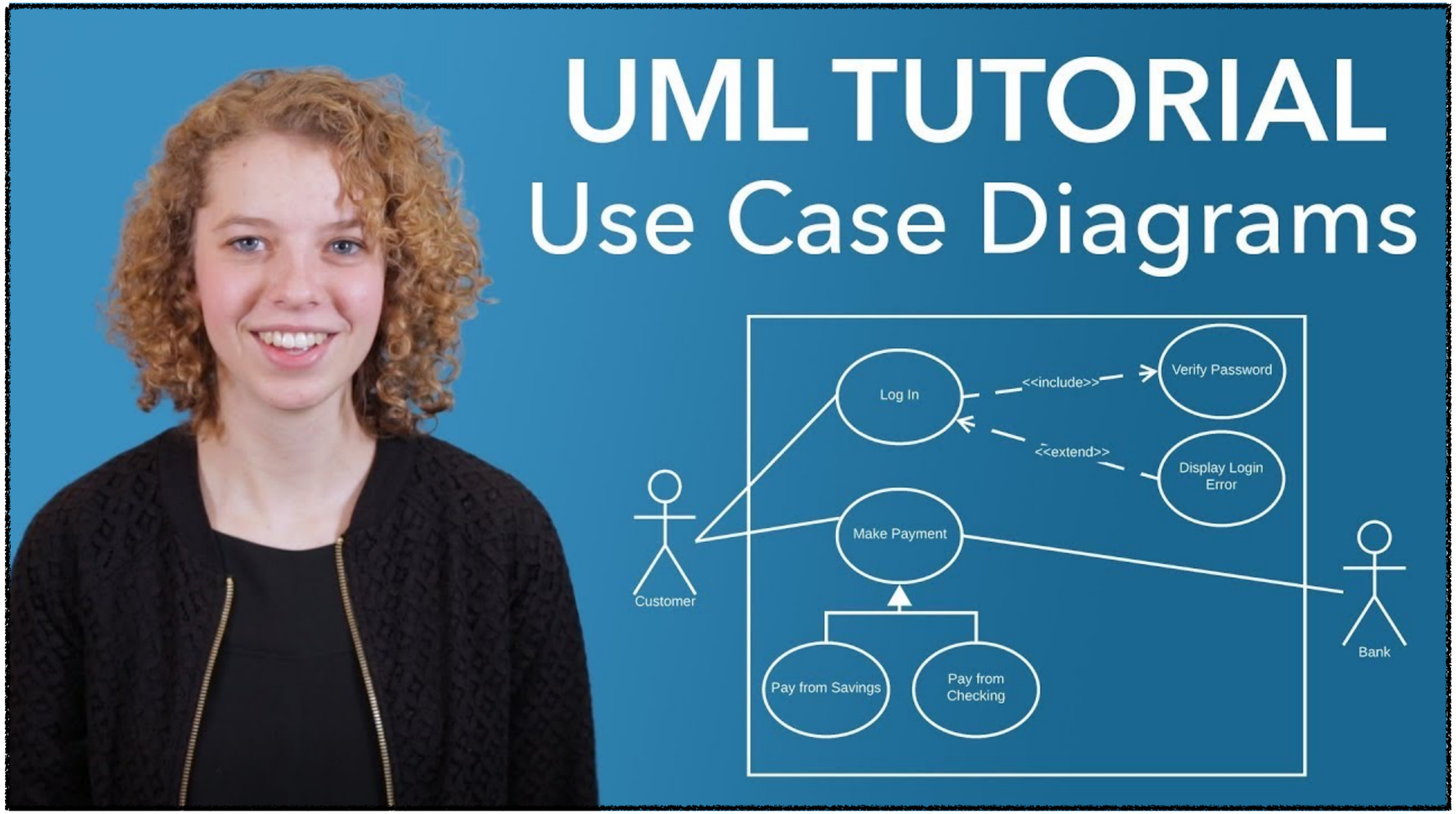
UML Use Case Diagrams




Book-a-Mensa
Customer
book
a seat
<<include>>
verify
authority
cancel
booking
<<include>>
enter
mensa
Create
backup
Door Person
Backup System

UML Use Case Diagrams: Recap

Was this too fast for you? Check the video.



LucidChart. UML Use Case Diagram Tutorial
https://www.youtube.com/watch?v=zid-MVo7M-E

Detailed Use Case: Cockburn Template (excerpt)

Use Case Name: *Order bike to user*
Goal in Context: As a user, I want to order a bike to come to my position, so that I do not have to search for a bike.
Precondition: User position can be reached by the bike
Success End Condition: The user gets a bike at their position.
Failed End Condition: The user is informed about the problem.

Description / Main Success Scenario:
Step 1: The user requests a bike to come to the users current position.
Step 2: The bike acknowledges.
Step 3: The bike drives to the user position.
Step 4: The user receives the bike.

Alternatives:
Step 1: The user requires a bike to come to a determined positions.
Step 2: The bike acknowledges
Step 3: The bike drives to the determined position.
Continue with Step 4.

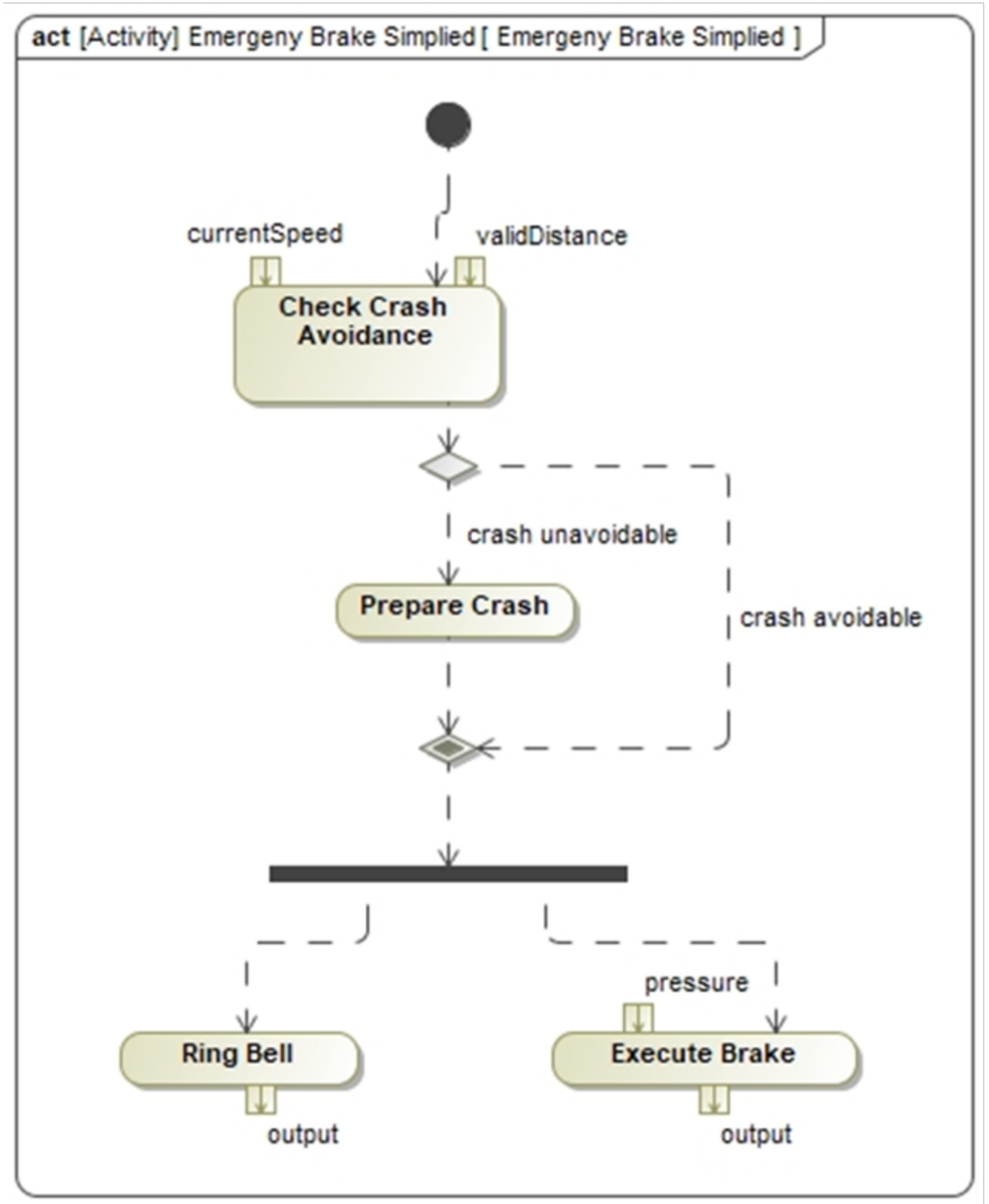
UML Activity Diagrams

An Activity Diagram visualizes:

- Activities, which specify a transformation from input to output via a sequence of actions.
- Building blocks of an activity: Actions
- Optionally: Inputs and outputs of each action
- Optionally: Who is executing which action? ("swimlanes", next slide)

Model forks in a process:

- Decision/merge node: Only one outgoing edge is taken and only if the edge's guard is fulfilled
- Fork/join node: Potentially all outgoing edges are taken

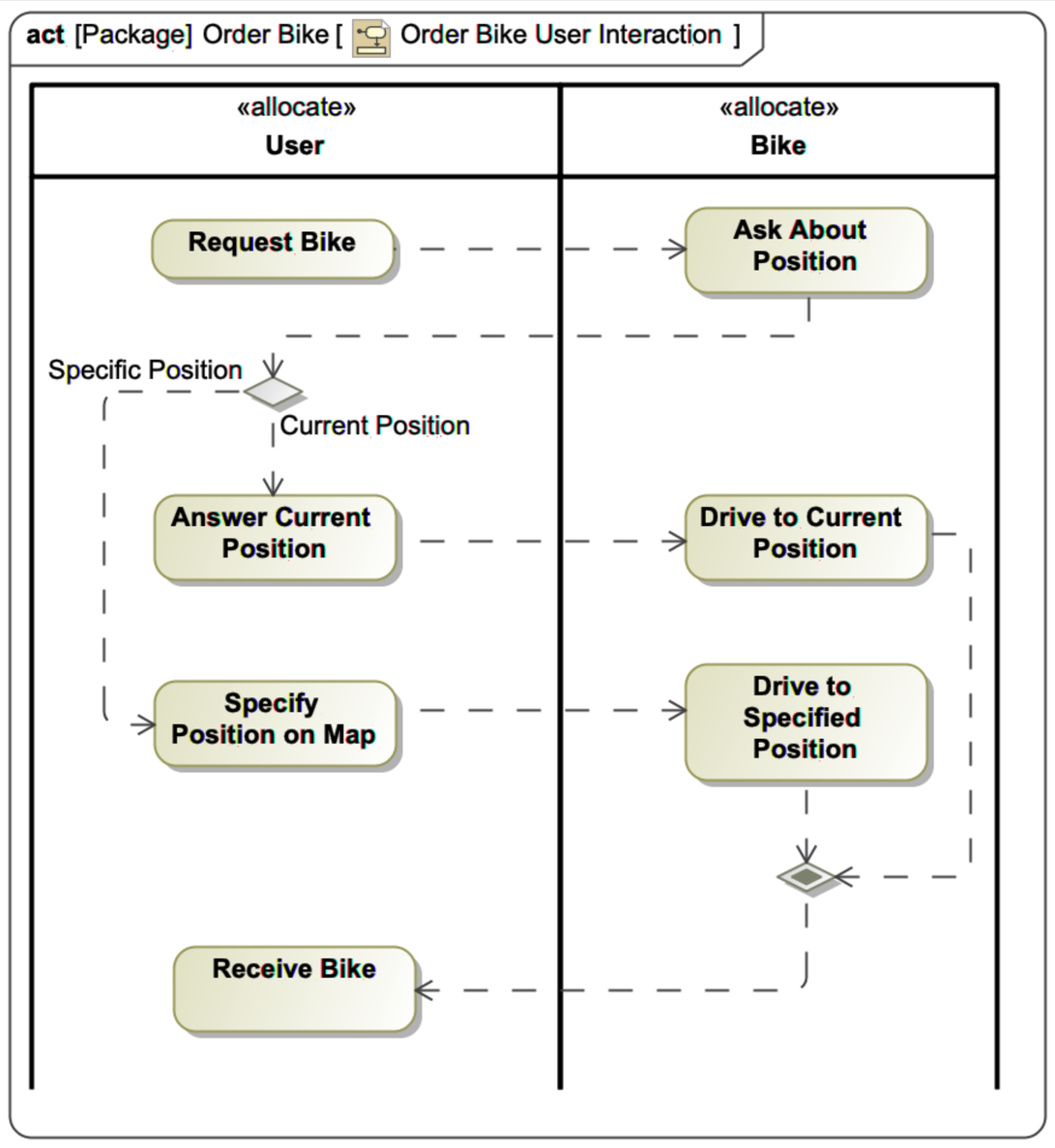


UML Activity Diagrams for Describing Use Cases

Use Case Name: *Order bike to user*
Goal in Context: As a user, I want to order a bike to come to my position, so that I do not have to search for a bike.
Precondition:
User position can be reached by the bike
Success End Condition:
The user gets a bike at their position.
Failed End Condition:
The bike could not reach the user position

Description:
Step 1: The user requests a bike to come to the users current position.
Step 2: The bike acknowledges.
Step 3: The bike drives to the user position.
Step 4: The user receives the bike.

Alternatives:
Step 1: The user requires a bike to come to a determined positions.
Step 2: The bike acknowledges
Step 3: The bike drives the to determined position.
Continue with Step 4.

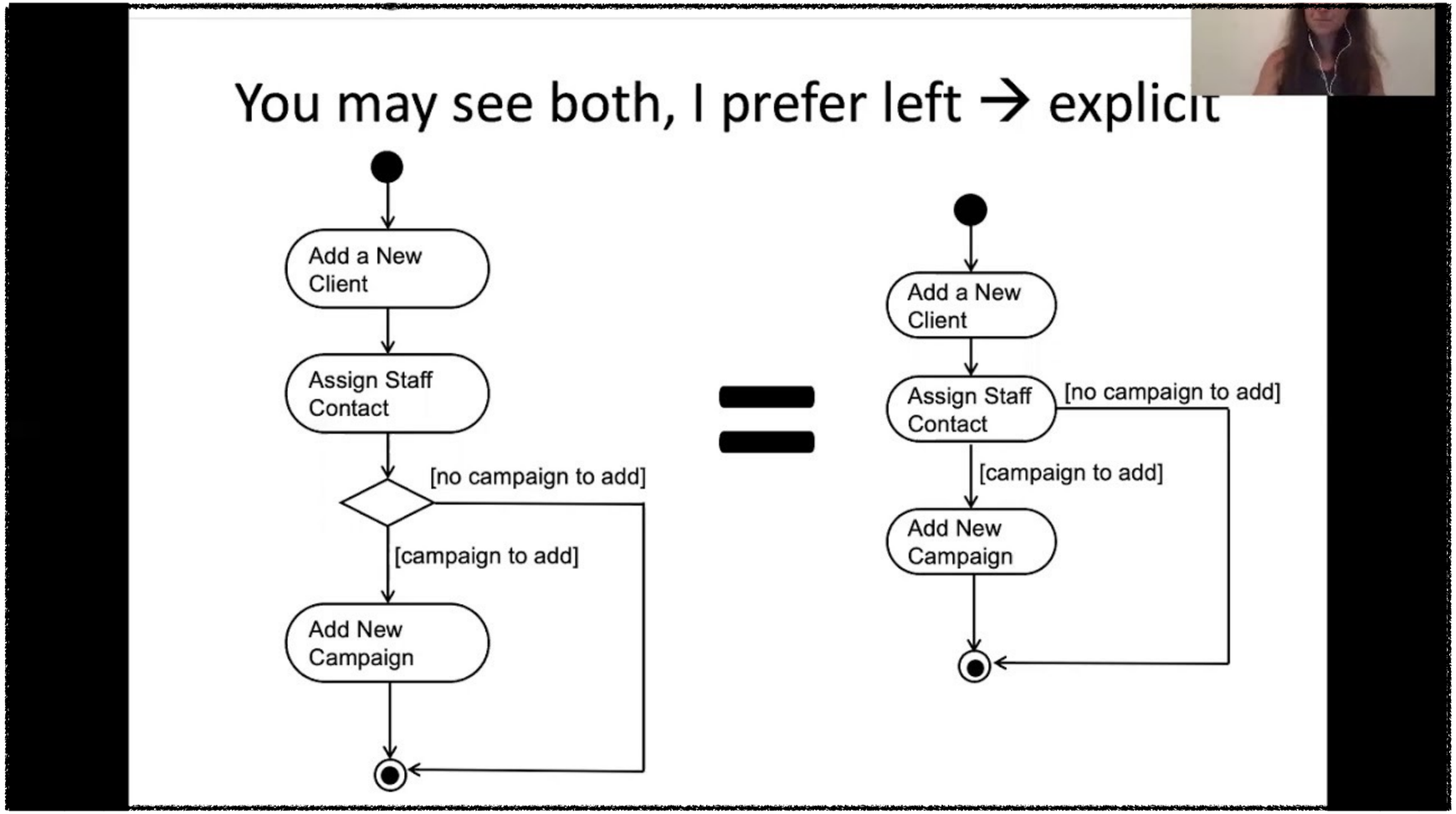


Choose your fighter:

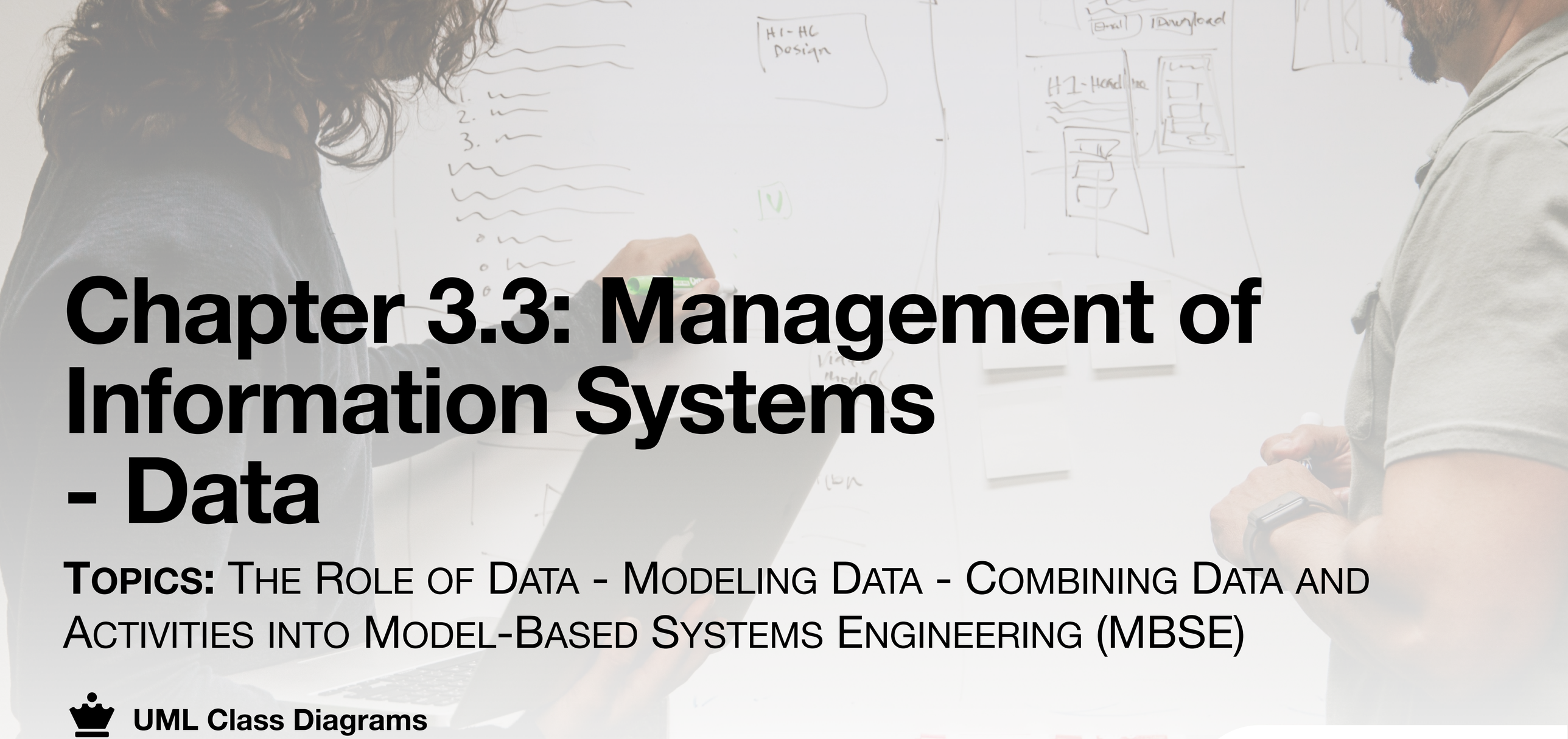
	Munich RE	URBANMAKER
OpenAI	WAELZHOLZ	?

UML Activity Diagrams: Recap

Was this too fast for you? Check the video.



Birgit Penzenstadler. Behaviour Modeling: Activity diagrams https://www.youtube.com/watch?v=m8iwrEEP7A4

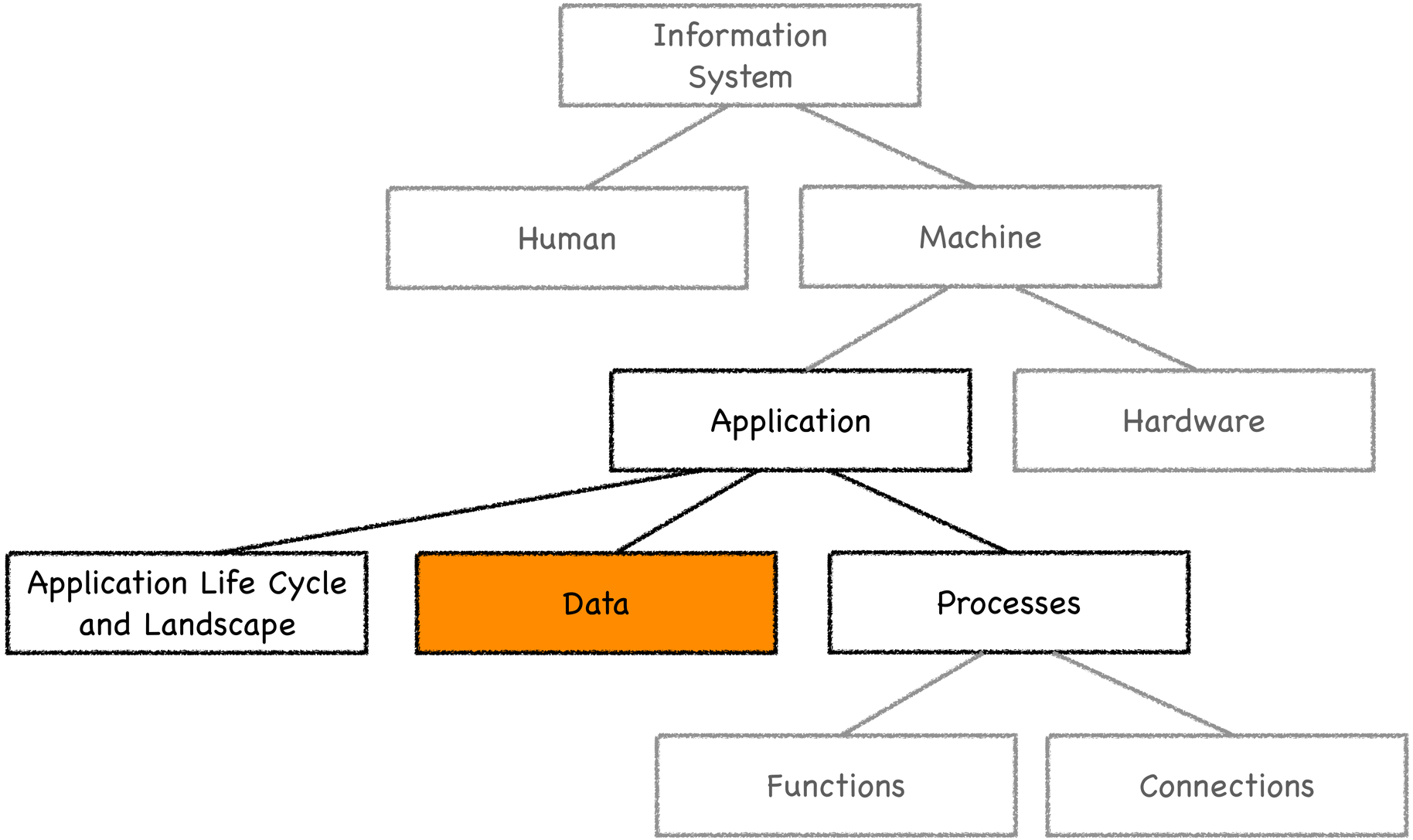


Chapter 3.3: Management of Information Systems - Data

Topics: The Role of Data - Modeling Data - Combining Data and Activities into Model-Based Systems Engineering (MBSE)

- **UML Class Diagrams**
- **Model-Based Systems Engineering**

Fachhochschule Südwestfalen
University of Applied Sciences


Information System
Human
Machine
Application
Hardware
Application Life Cycle and Landscape
Data
Processes
Functions
Connections

The Mental Model

Mental model: A mental model is the organized understanding and mental representation of knowledge about key elements of the system under consideration.

Donald A Norman. Some observations on mental models. Mental models, 1983.

- Mental Models form the fabric of communication, architecture, databases, project planning, contract negotiation ...
- Modern approaches (e.g. Domain Driven Design) re-focus on the importance of a common terminology

Evans, E., 2004. Domain driven design : tackling complexity in the heart of software. Addison-Wesley.

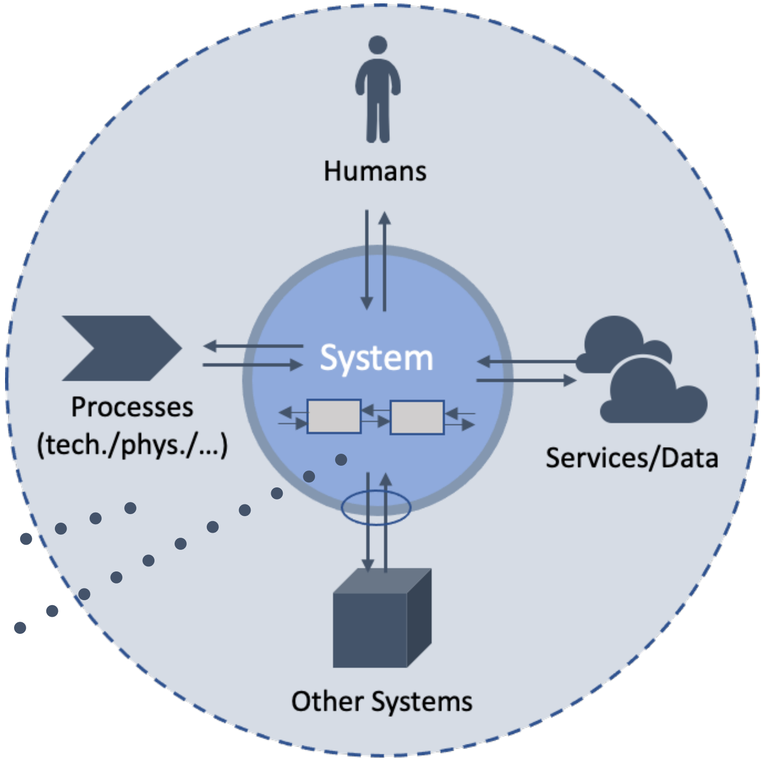
Eric Evans, 2019. What is DDD?. DDD Europe 2019. https://www.youtube.com/watch?v=pMuiVlnGqjk

Modeling: The **what** and **why**

Modeling Object:
What is modeled?

Domain Models as description of the **application domain** of the system to be built.

System Models as abstract representation of the system and its architecture.



Modeling Purpose:
What is it modeled for?

Conceptual Models for stakeholders, developers, testers, users, etc. prioritizes *understandability* for the target users over *formality*.

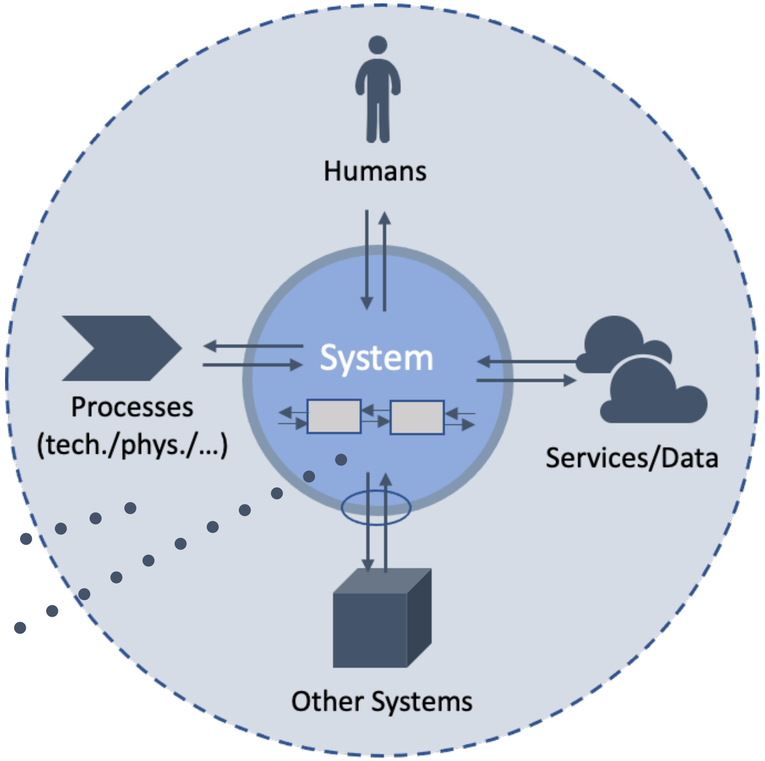
Specification Models allow to generate source code or test cases for the final system and therefore must be *formally precise*.

Modeling: The **what** and **why**

Modeling Object:
What is modeled?

Domain Models as description of the **application domain** of the system to be built.

System Models as abstract representation of the system and its architecture.

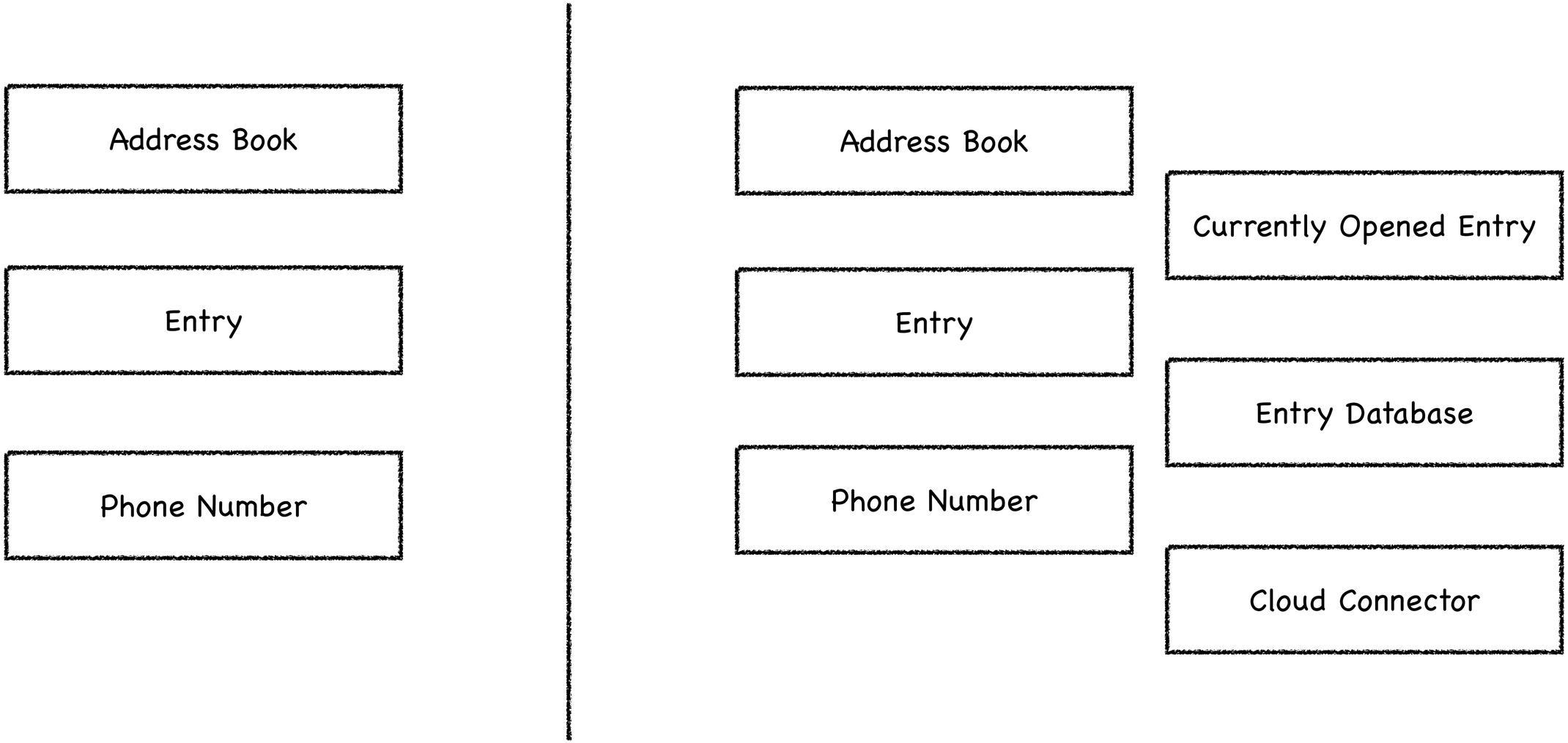


Modeling Purpose:
What is the model for?

Conceptual Models for stakeholders, developers, testers, users, etc. prioritizes *understandability* for the target users over *formality*.

Specification Models allow to generate source code or test cases for the final system and therefore must be *formally precise*.

Domain Models vs. System Models

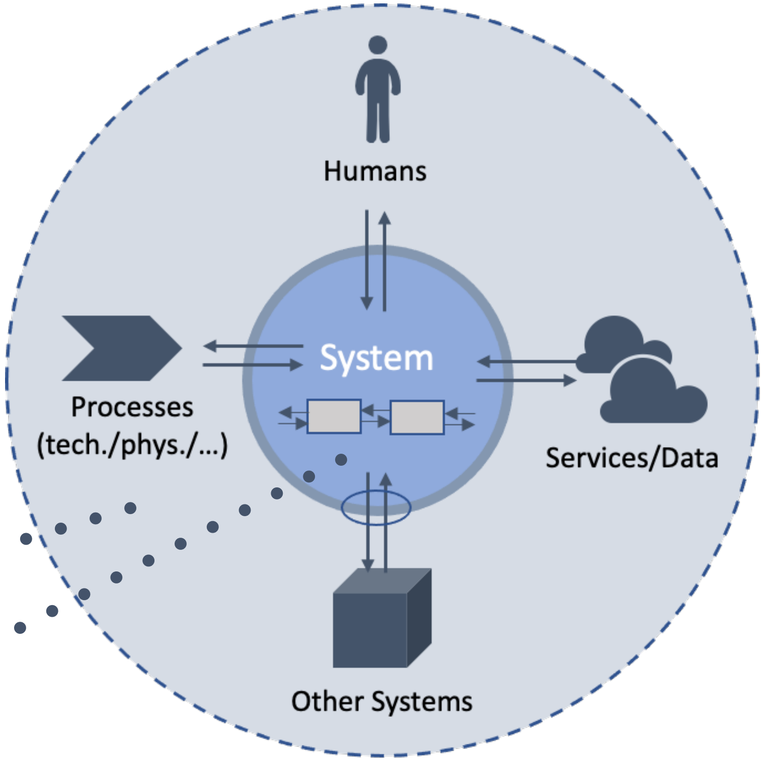


Modeling: The **what** and **why**

Modeling Object:
What is modeled?

Domain Models as description of the **application domain** of the system to be built.

System Models as abstract representation of the system and its architecture.

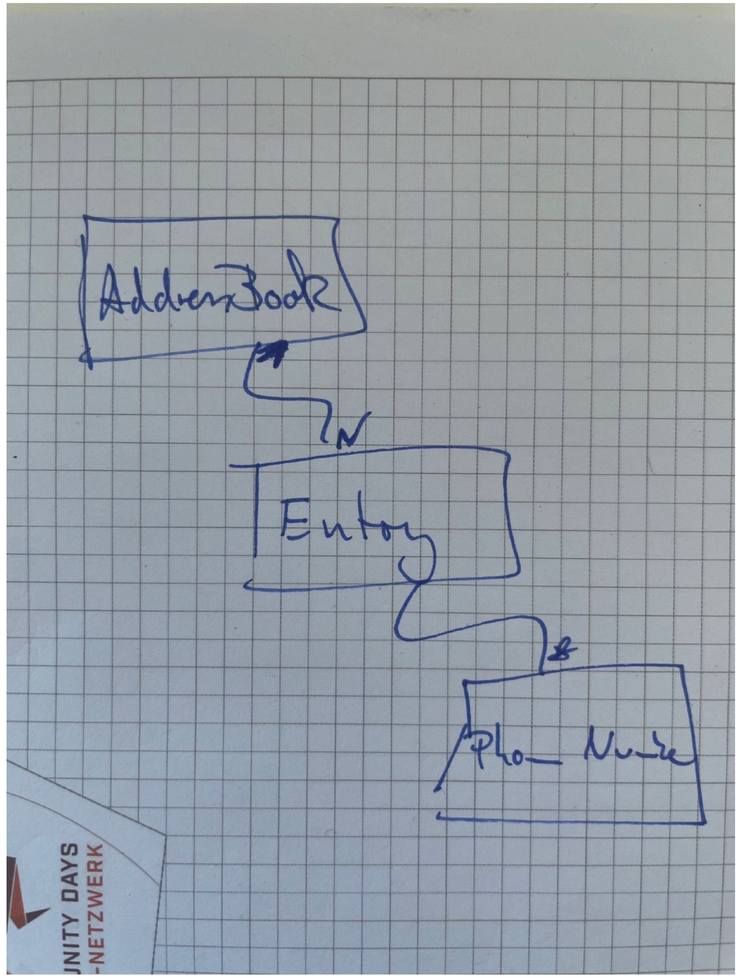


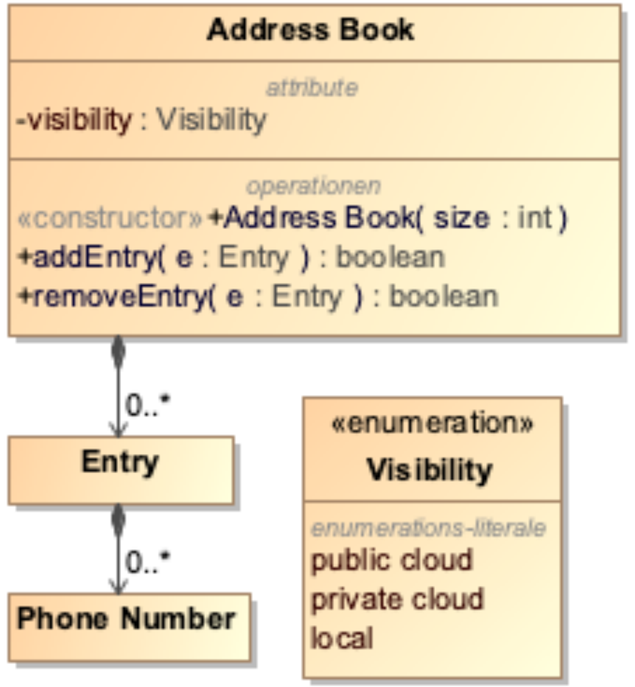
Modeling Purpose:
What is the model for?

Conceptual Models for stakeholders, developers, testers, users, etc. prioritizes *understandability* for the target users over *formality*.

Specification Models allow to generate source code or test cases for the final system and therefore must be *formally precise*.

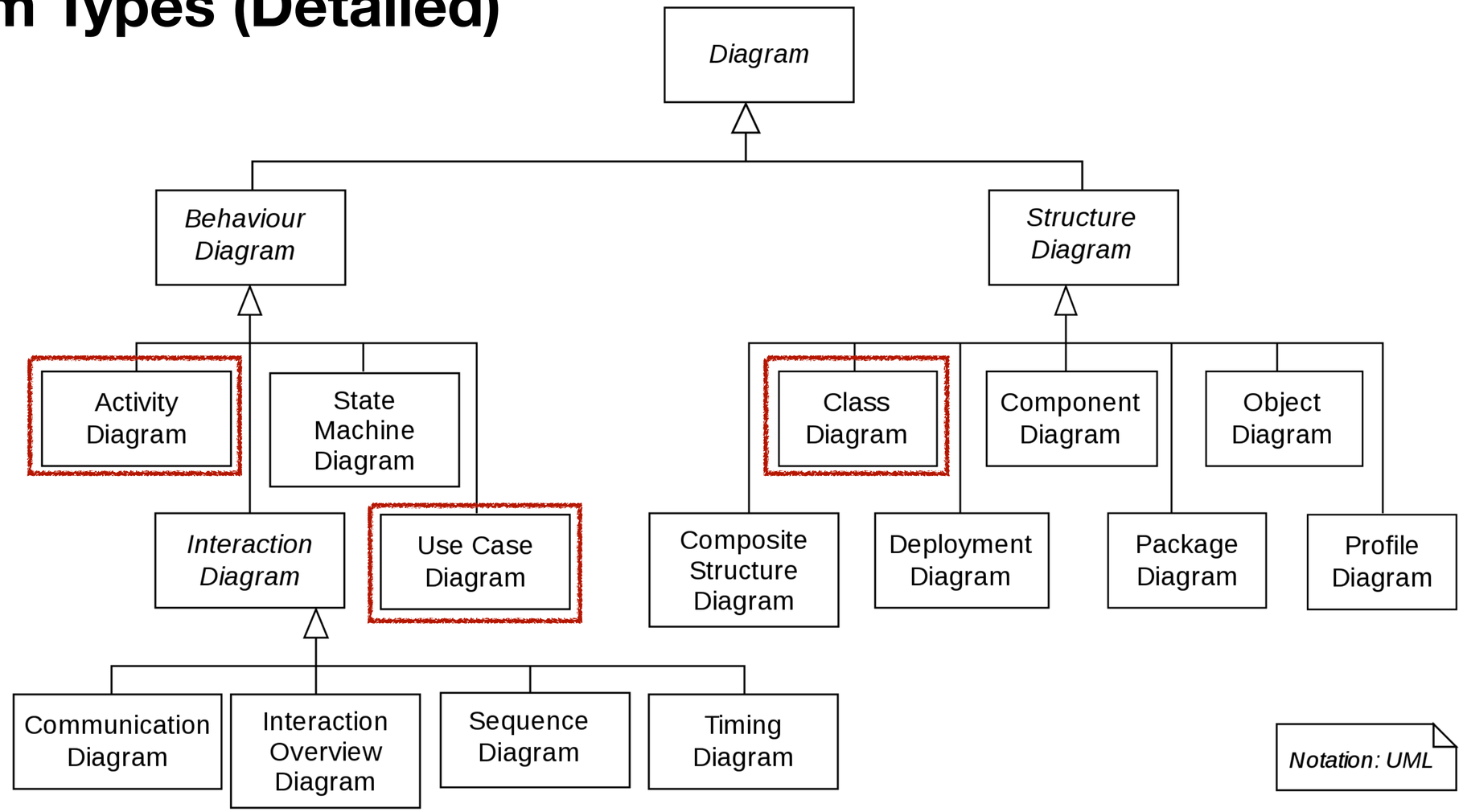
Modeling for Communication vs. Specification





The Unified Modeling Language (UML)

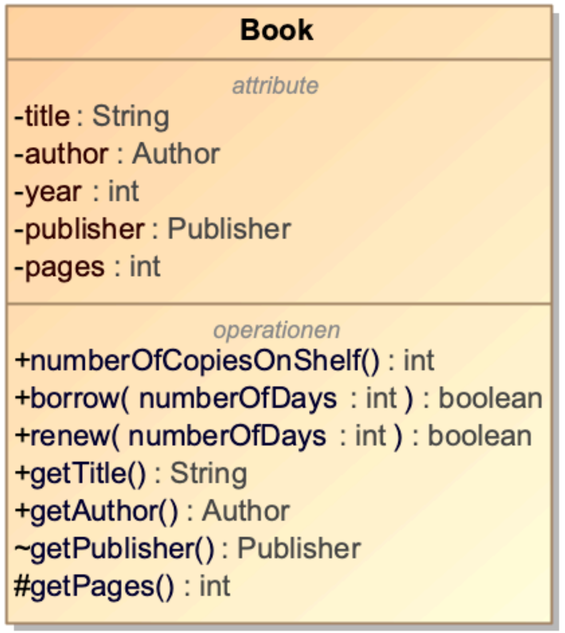
Diagram Types (Detailed)



https://upload.wikimedia.org/wikipedia/commons/thumb/e/ed/UML_diagrams_overview.svg/2560px-UML_diagrams_overview.svg.png

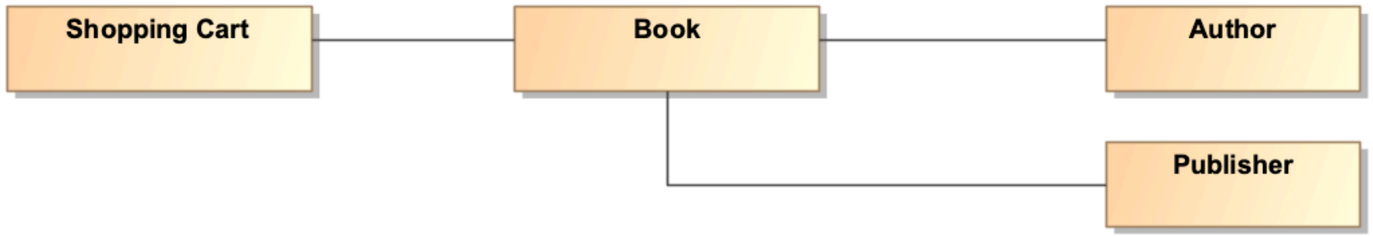
UML Class Diagrams: Classes

- *Objects* are concrete instances of *classes*,
  e.g., a specific copy of Harry Potter and the Philosopher's Stone
- Classes are generalized *types of objects*,
  e.g., the class Book
- In UML class diagrams, classes are represented through rectangles with a name
- Compartments for *attributes* and *operations* may be added
- Syntax for attributes is:
  <+/-/#/~> <name> : <type>
- Syntax for operations is:
  <+/-/#/~> <name> (<parameters>:<types>) : <return type>



UML Class Diagrams: Associations

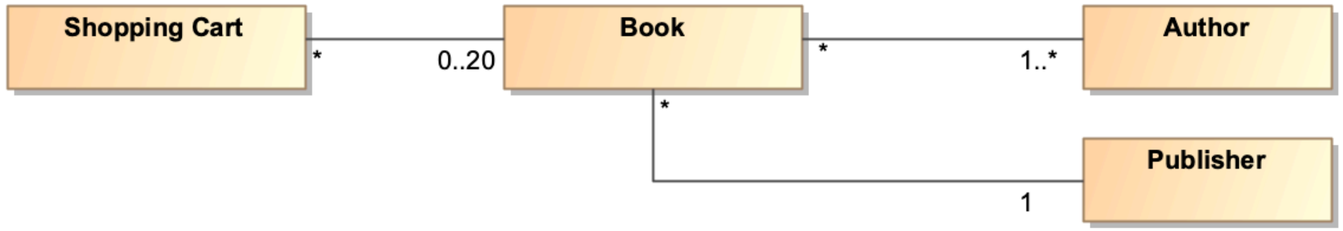
- An association connects two (or rarely more) classes.
- A *link* is an instance of an *association*.
- *Rolenames* show the roles that objects play in an association.

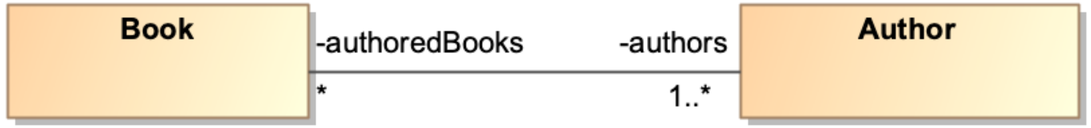


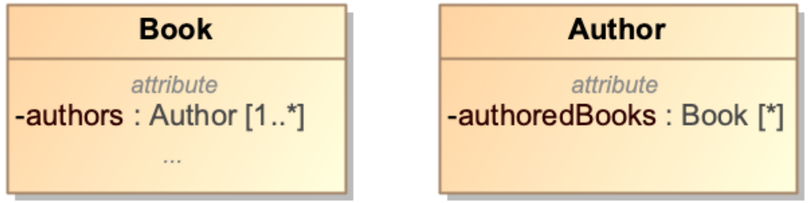
UML Class Diagrams: Multiplicities

The most common multiplicity indications:

- **0..1**: The property may or may not have a value.
- **1..1**: The property has exactly (and always) a value. Usually denoted as “1” only.
- **0..*** or shorthand *: The property has zero or more values.
- **1..***: The property one or more values.
- **n..m**: The property has at least n and at most m values.








```
public class Book {
   private Author[] authors;
   ...
}
public class Author {
 private Book[] authoredBooks;
 ...
}
```

How to Create A Data Model from a Specification

A Simple Method

Simple Method: look for noun phrases in the system description!

Then remove things which are:
▶ Redundant ▶ Attributes ▶ Outside scope ▶ Operations and events
▶ Vague ▶ On a different level of detail (*System Model* vs. *Domain Model*)

Similarly, can use *verb phrases* to identify operations and/or associations.

Books and Journals: The library contains books and journals. It may have several copies of a given book. Some of the books are for short term loans only. All other books may be borrowed by any library member for three weeks. Members of the library can normally borrow up to six items at a time, but members of staff may borrow up to 12 items at one time. Only members of staff may borrow journals.

Each record company has a unique name; an email address and telephone number are also kept on each record company. A record company publishes one or more albums; every album is published by exactly one record company. An album is identified by its album ID, and other attributes are title, price, and number of songs. Each album contains one or more songs. Each song can be included on one or more albums and is performed by exactly one musician. Every song is identified by its song ID and in addition its song title and length are also kept as attributes. A musician may perform one or more songs and he/she is uniquely described by a musician ID. We also know each musician's name and address (combination of street, city and ZIP). A musician receives a separate royalty check for each of his songs yearly from the record company. Each check is identified by its check number, and we also keep track of the date and amount of each check.

What's the relationship between authors, customers, and publishers?

Generalization

Inheritance is a binary relationship:

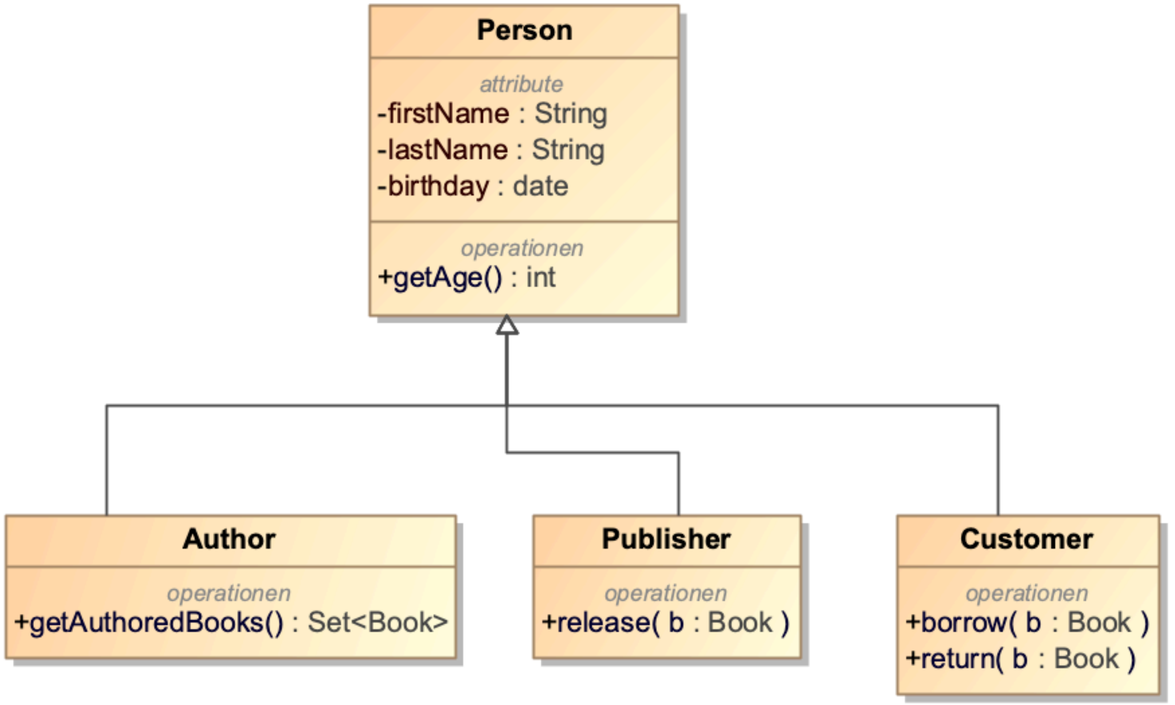
- Subtype: A subgrouping of the objects in a class that has attributes distinct from those in other subgroupings
- Supertype: A generic class that has a relationship with one or more subtypes

Attribute Inheritance:

- Subtype entities inherit values of all attributes of the supertype
- An instance of a subtype is also an instance of the supertype

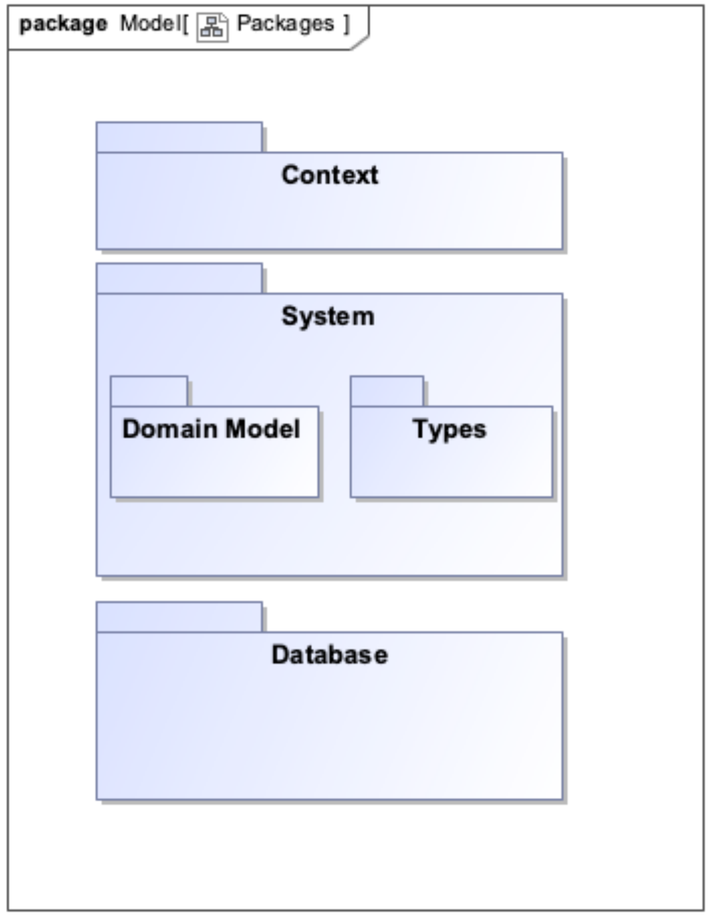
Association Inheritance:

- Associations at the supertype level indicate that all subtypes will have this association
- Associations at the subtype level only refer to the subtype



UML Class Diagrams: Packages

- Packages define regions
- Think of them as folders, defining the path to elements inside the package



UML Class Diagrams: Access Operators

- Syntax for attributes is:
  <+/-/#/~> <name> : <type>

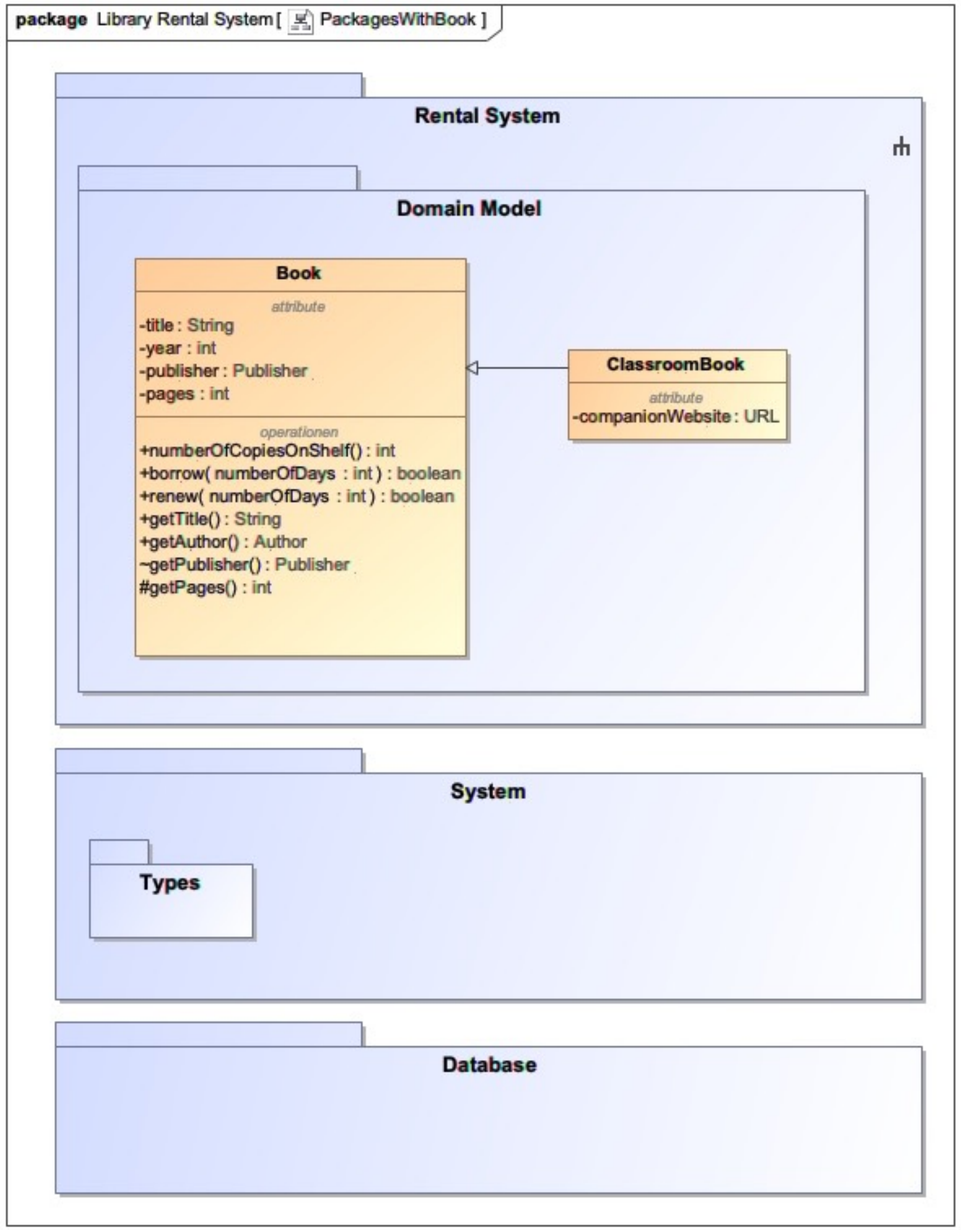
- Syntax for operations is:
  <+/-/#/~> <name> (<parameters>:<types>)
  : <return type>

+ (public): accessible from all classes with access to this package

~ (package): accessible only inside this package

(protected): accessible only to this and to all subclasses

- (private): accessible only to this class



Choose your fighter:

Virtuelles Krankenhaus VKh.NRW

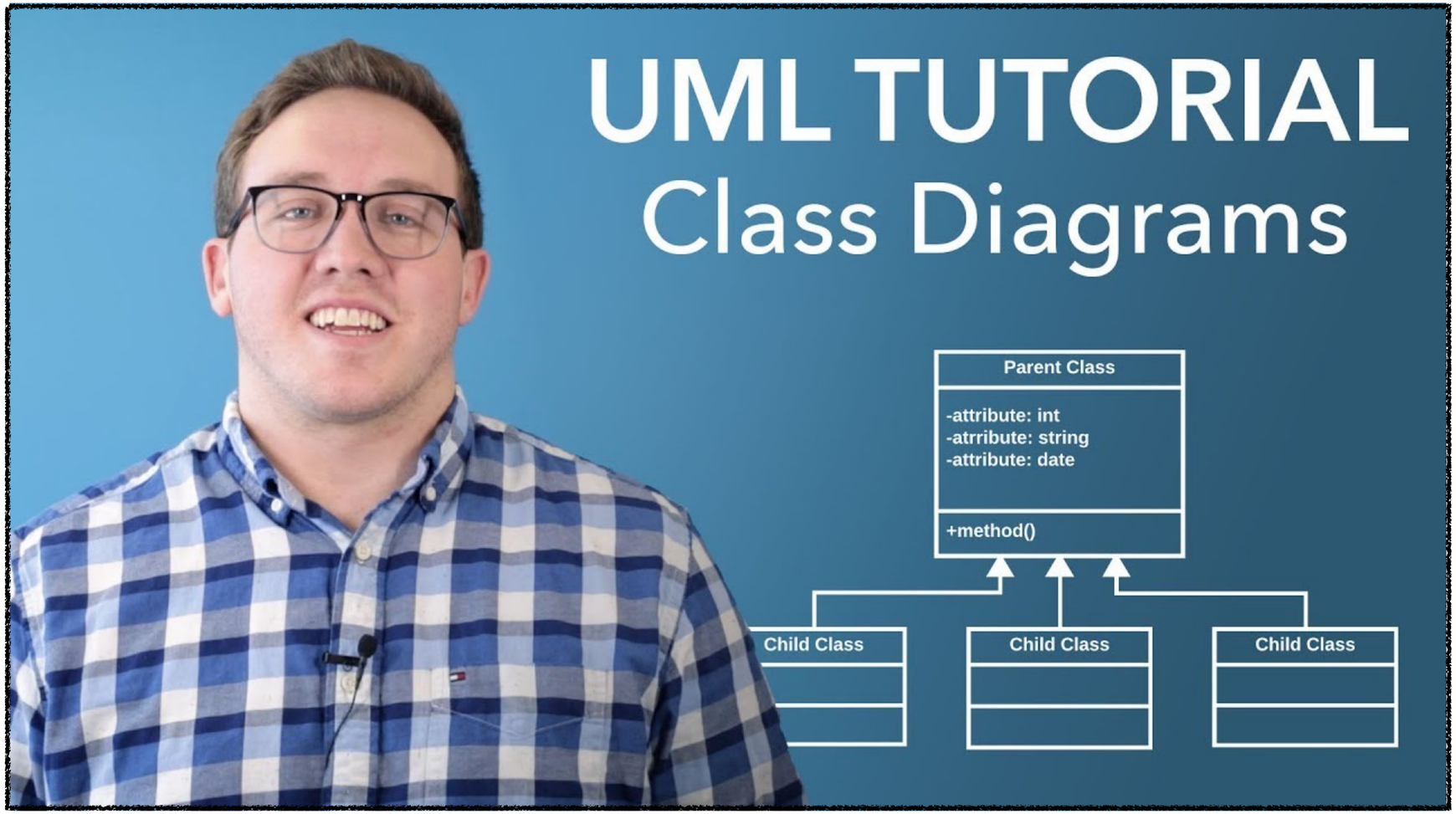
URBANMAKER

WAELZHOLZ

?

UML Class Diagrams: Recap

Was this too fast for you? Check the video.

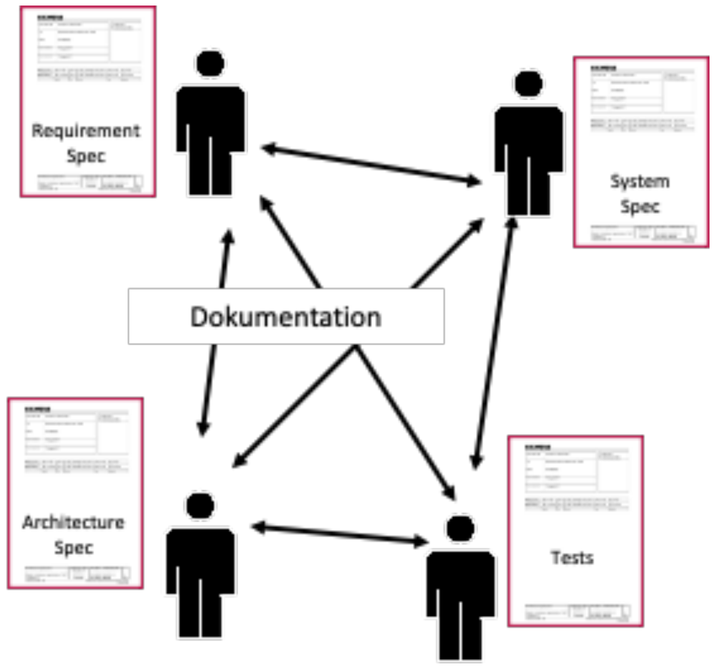


LucidChart. UML Class Diagram Tutorial
https://www.youtube.com/watch?v=UI6lqHOVHic

Model-based Systems Engineering

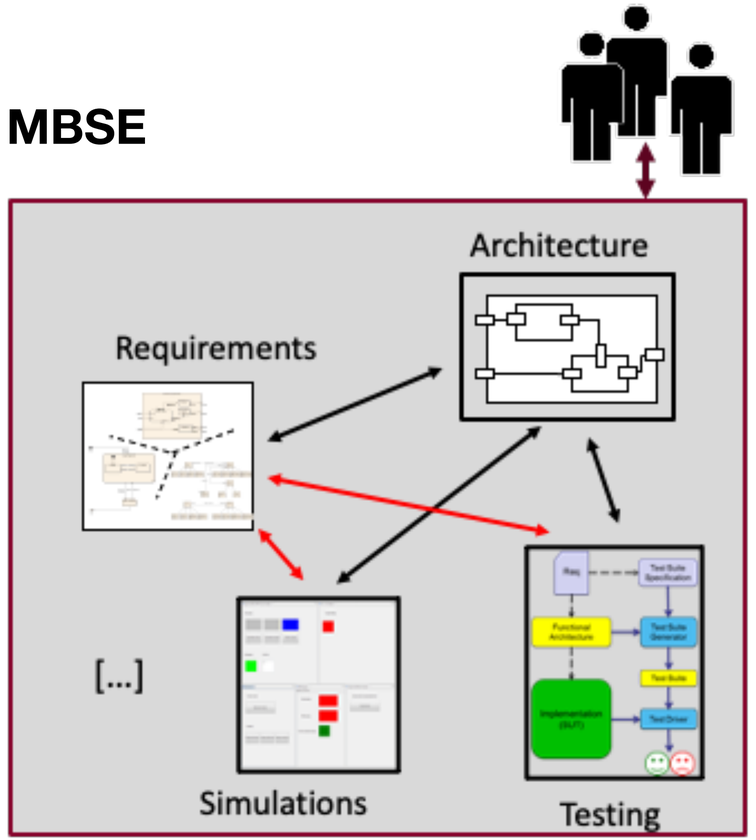
Going from Documents to Models

Document-based Engineering



Document Types and Document Structure

MBSE



Types, Relations, Traces, Information
Architecture, Representation, Usage

Of Models and Diagrams...

(System) Model:

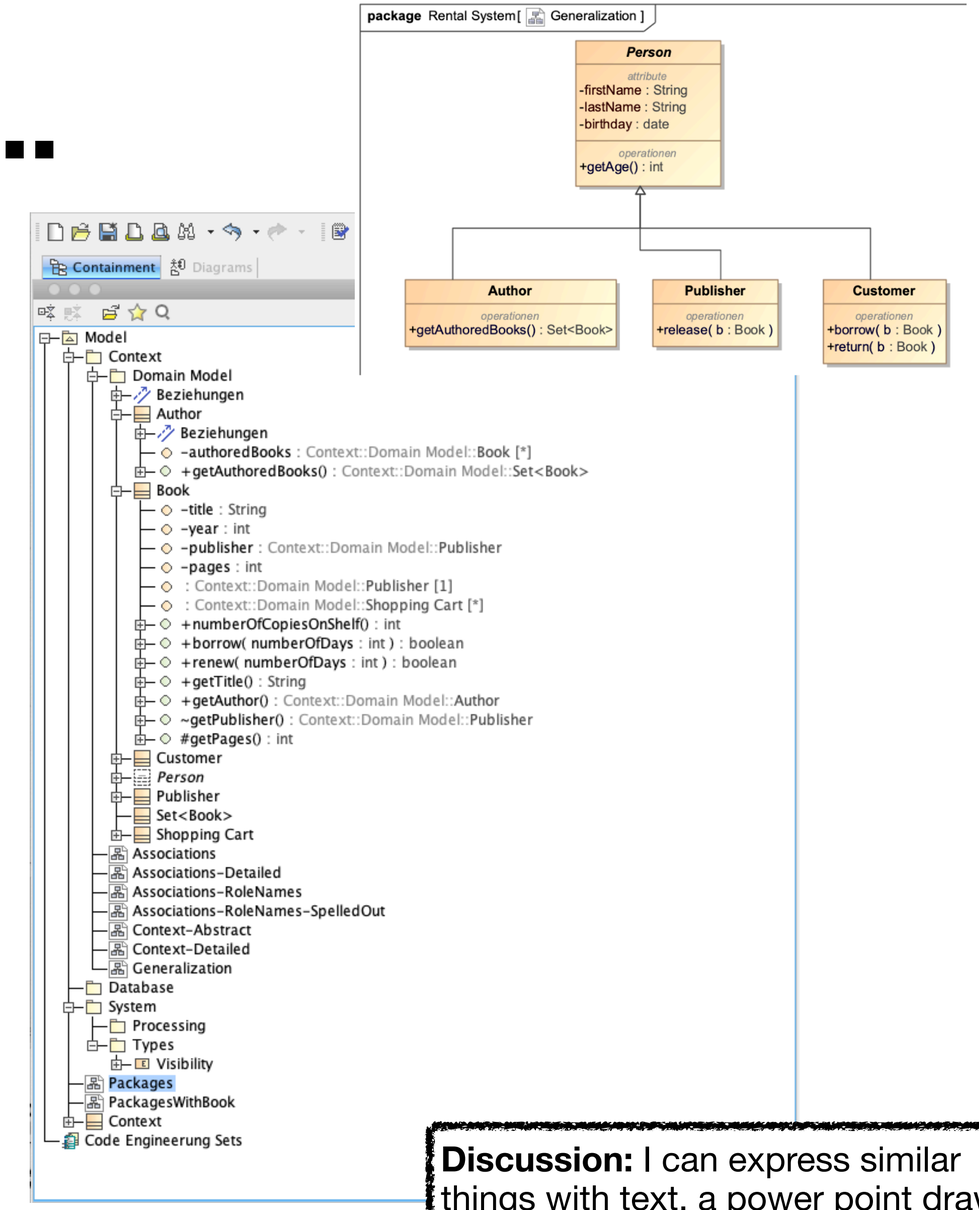
- Contains all system elements and their relations
- Is structured in a hierarchic tree (think: file system), but the tree does not necessarily represent system structure

Diagram:

- Visualizes a selection of system elements and a selection of their relations
- Provides a graphical view of a part of the model

Keep in mind:

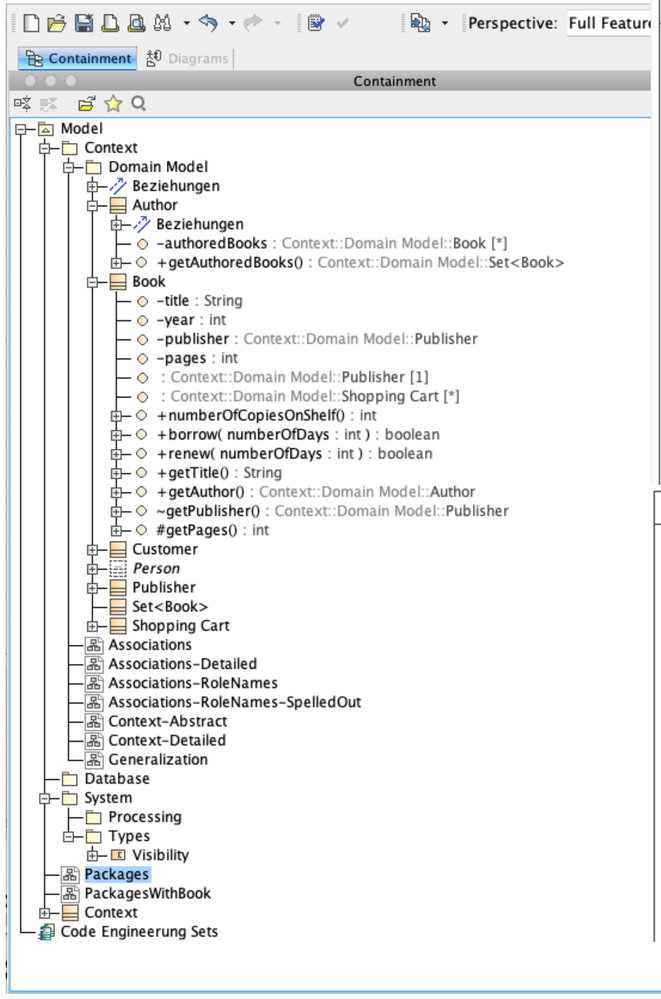
- A model element can be visualized **in many diagrams**
- Even if a model element is not shown in a diagram **it may still be part of the model**
- You can create a model **without a single diagram**

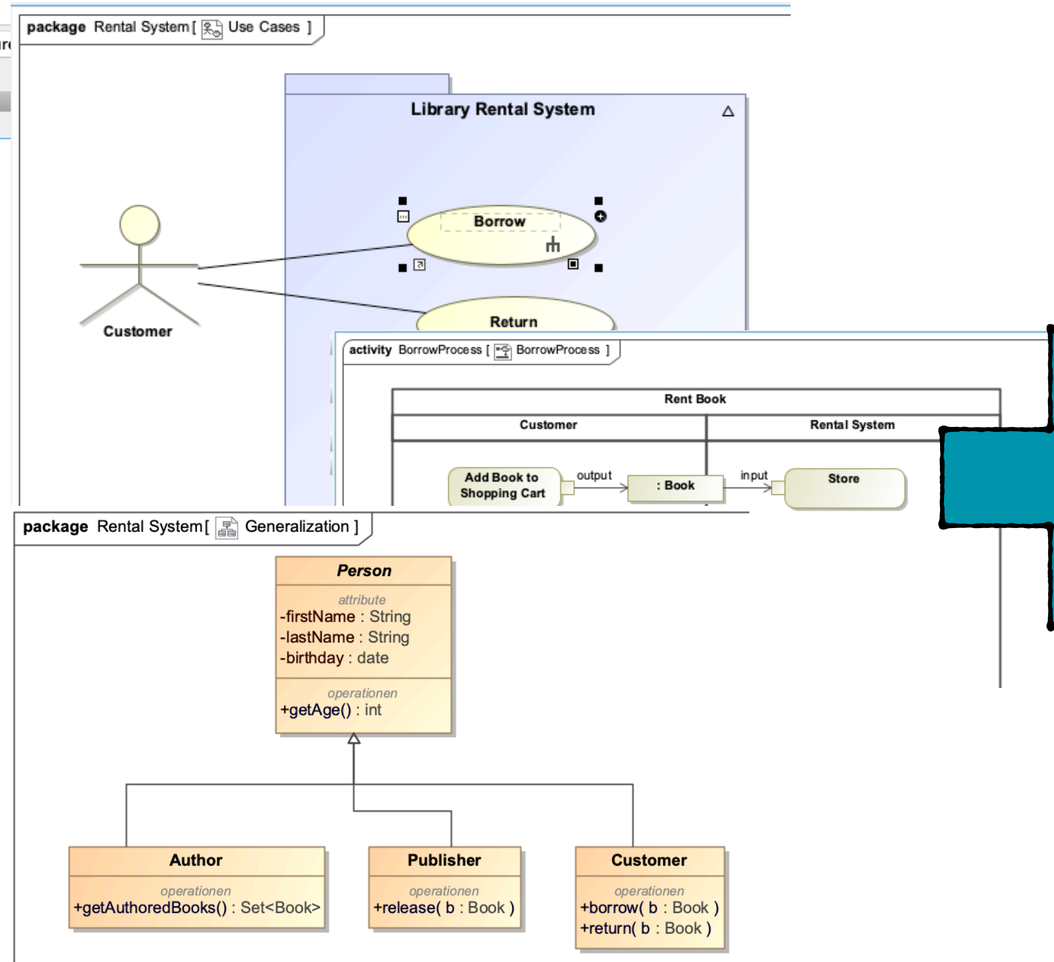


Discussion: I can express similar things with text, a power point drawing, a diagram and a model. *What does each step add?*

Model-based Systems Engineering

How it all comes together





Validate requirements

Analyze impact of change

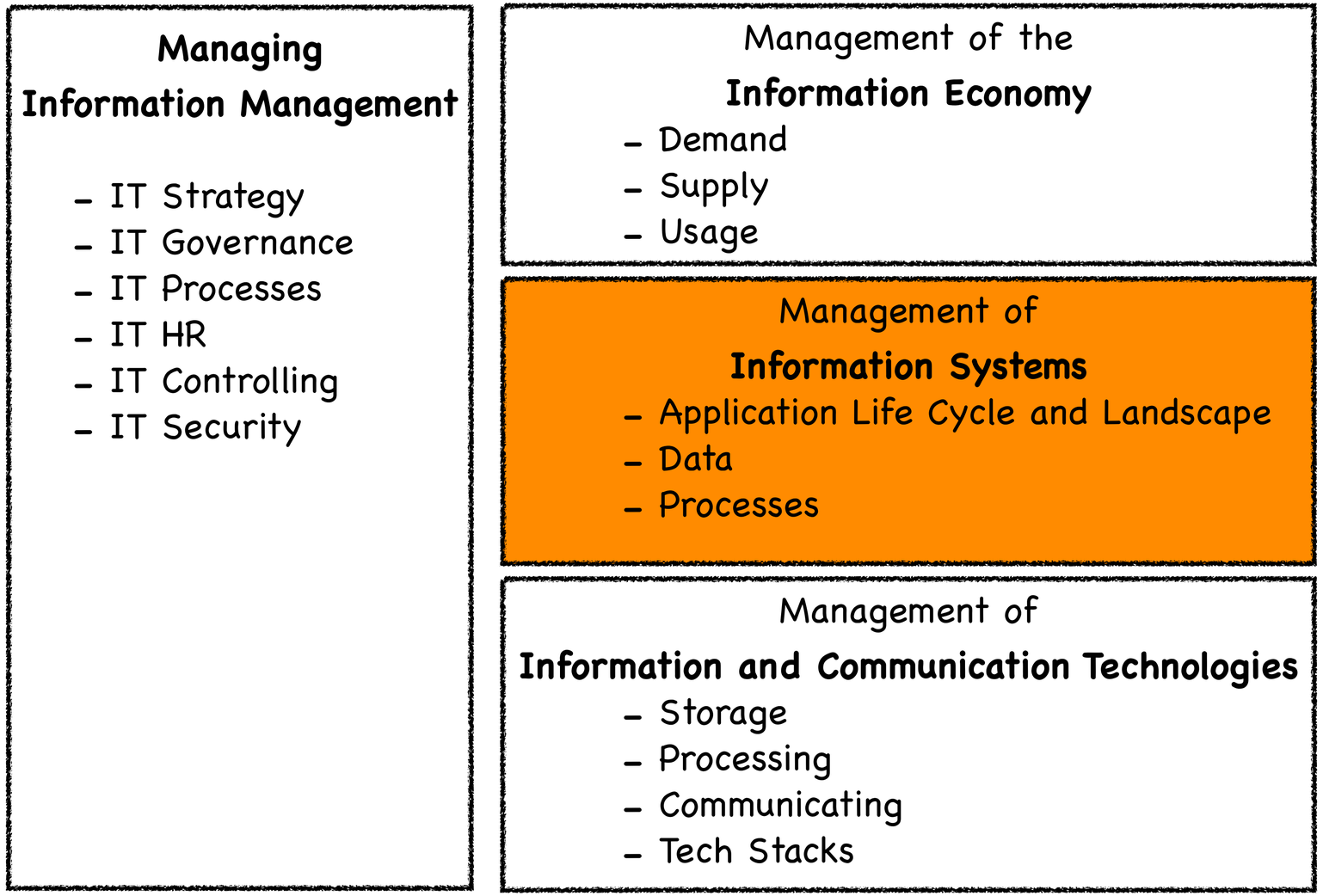
Verify design

Generate artifacts

Model-based Systems Engineering

How it all comes together: Demo

Information Management



Krcmar, Helmut. "Informationsmanagement." Informationsmanagement. Springer Gabler, Berlin, Heidelberg, 2015, p.94.