Fraunhofer IESE

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

SSA05 – Design Process

Dr. Pablo Oliveira Antonino
pablo.antonino@iese.fraunhofer.de

TU Kaiserslautern, SS2018
Lecture "Software and System Architecture (SSA)"

Organization

Lecture

- **2018-06-18: Architecture Evaluation** will be given by Domink Rost from Fraunhofer IESE.

Exercises

- Original plan -> 2018-06-06: Modeling with tools
  - ***(NEW DATE 13.06.2018).***

- Original planned -> 2018-06-13: Evaluation of Architecture documentation
  - ***(NEW DATE 20.06.2018)***

Discussion

- **RECAP LAST LECTURE**

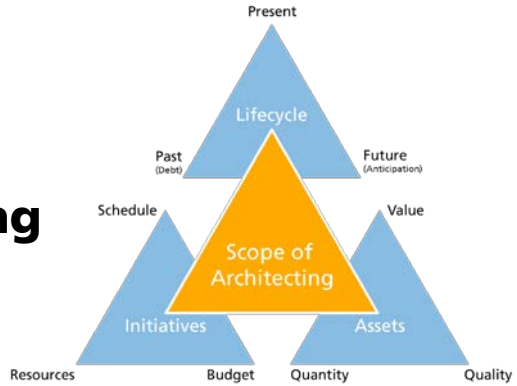
- Explain the contents of the last lecture
  - What were the topics?
  - Why do we need it?
  - How does it work?
  - How is it created, used, and/or evolved?

Design

- It is relatively easy to design for **the perfect cases**, **when everything goes right**, or **when all the information required is available in proper format.**
  [Donald Norman]

- The most difficult part of building software is not coding;
  it is the decisions you make early at the design level.
  Those **design decisions** live with the system for the rest of its lifetime.
  [Pattern-Oriented Analysis and Design - Composing Patterns to Design Software Systems]

The Goals of Design



- **Balancing the Bermuda triangle of architecting** (lifecycle, initiatives, assets)
- **Creation** of a plan or convention for the construction of a software system
  - **Decomposition** of the problem into smaller pieces
  - **Control** of the complexity
    - Coming to a solution and communicating it
  - **Prediction** of effort, quality, impacts, etc.
- **Risk Mitigation** of the impact of change on a software system
  - Change is the inevitable characteristic of any successful software system
  - Prepare **anticipated** changes (before the change is required)
  - Address **unforeseeable** changes (when it occurs)

Architecture vs. Design

- **"Architecture is design, but not all design is architecture"**

  (G. Booch)

  - It depends on the criticality of the decision: is it **architecturally-significant?**
  - Principle of locality
  - Depends on scope and is relative

- We call things architecturally-significant if they are…
  - **Costly to change**
  - **Risky**
  - **New**

Challenges of Architecture Design

- Understand the **domain** and its specifics
- Work on **incomplete and changing** requirements
- Find adequate **solutions**
- Achieve adequate **confidence** that solutions work
- Design a solution **involving many experts** for specific topics
- Refine and adjust architecture while implementation is already ongoing
- Judge technologies for their adequacy
- Select and use **technologies** appropriately
- Become aware of **drift**

Generic Decomposition Steps

- Identify **elements**
  - Identify responsibilities
  - Identify unique, self-contained roles/elements
  - Identify types of composed/aggregated elements
- Identify **relationships**
  - Identify data/information exchange requirements
  - Identify interfaces
  - Identify dependency types
  - The dependency graph of elements must have no cycles
- **Increase semantic cohesion** between elements and **reduce coupling** of elements
  - Elements that change together are grouped together
  - Elements that are used together are grouped together
  - Elements that are owned by the same group are grouped together

Design: Essential Principles

Abstraction

- Extraction of the essentials

Separation of Concerns

- Hierarchical Decomposition
- Divide & Conquer (Top-down)
- Divide & Conquer (multi-dimensional)
- Modularization
- Localization of concerns

Encapsulation

- Information hiding
- Coupling & Cohesion
- Redundancy Free

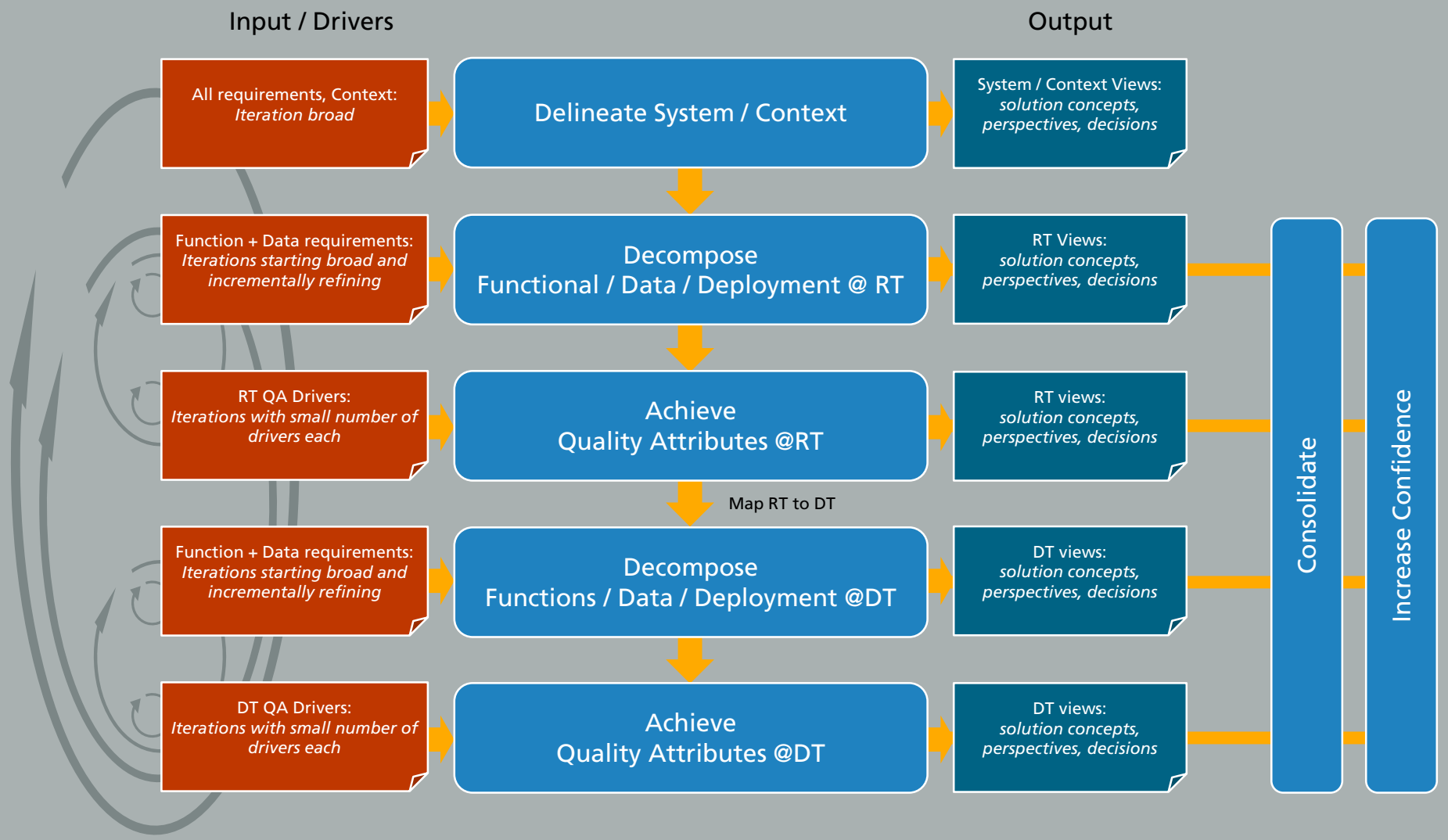
Uniformity

- Common Design Principles

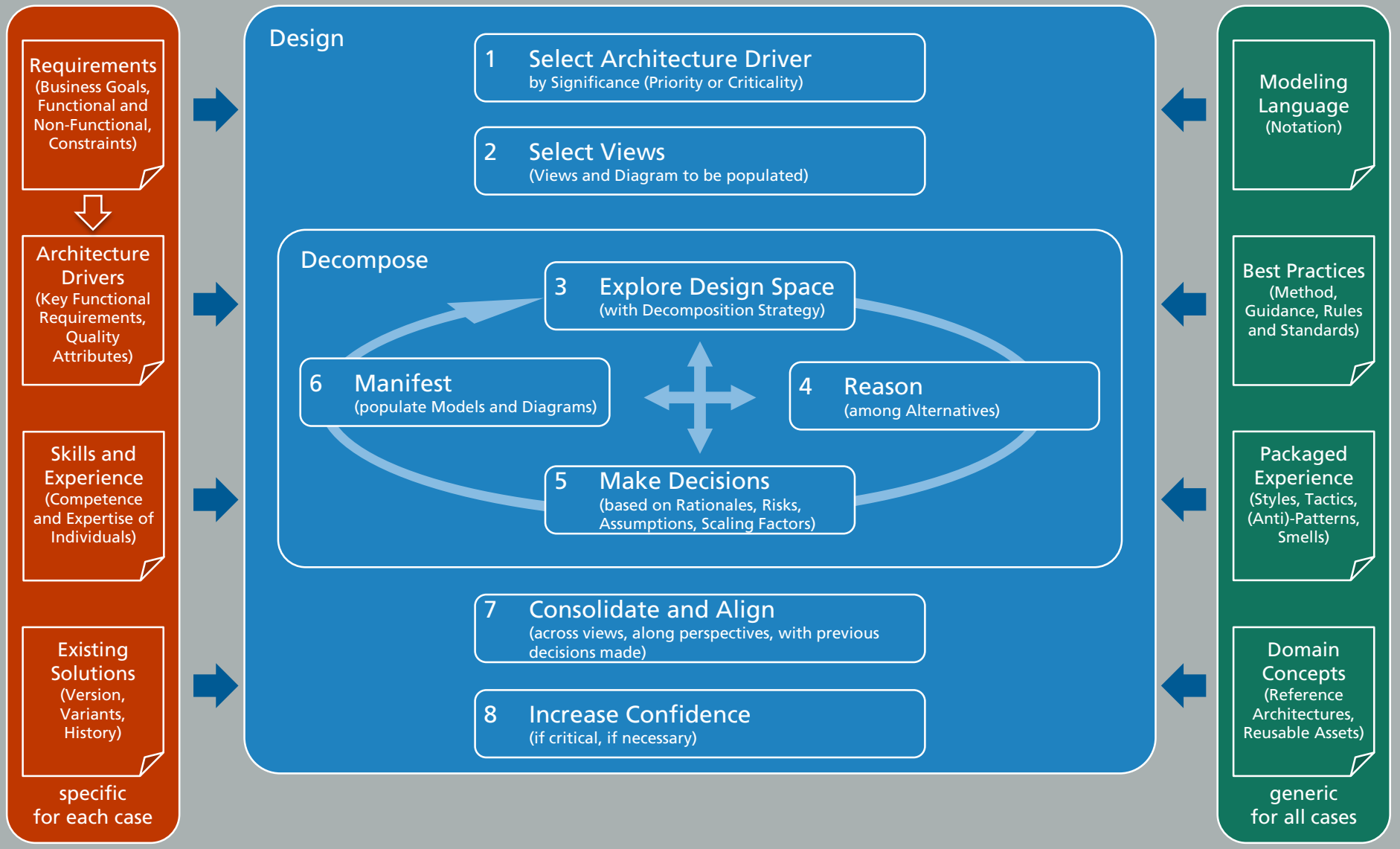
Conceptual "Tools" for Architecture Design

- **Creativity**
- **Classification**
  - Element types, relation types
- **Abstraction**
  - Simplifications, aggregations, processes, end-to-end usages
- **Categorization**
  - Group distinct facets of a solution concepts
    - E.g., functionality, data, data flow, information flows, control flows, deployment, interfaces, physical constraints, technologies
- **Experience**
  - … since there is no "detailed guide to creativity" ☺

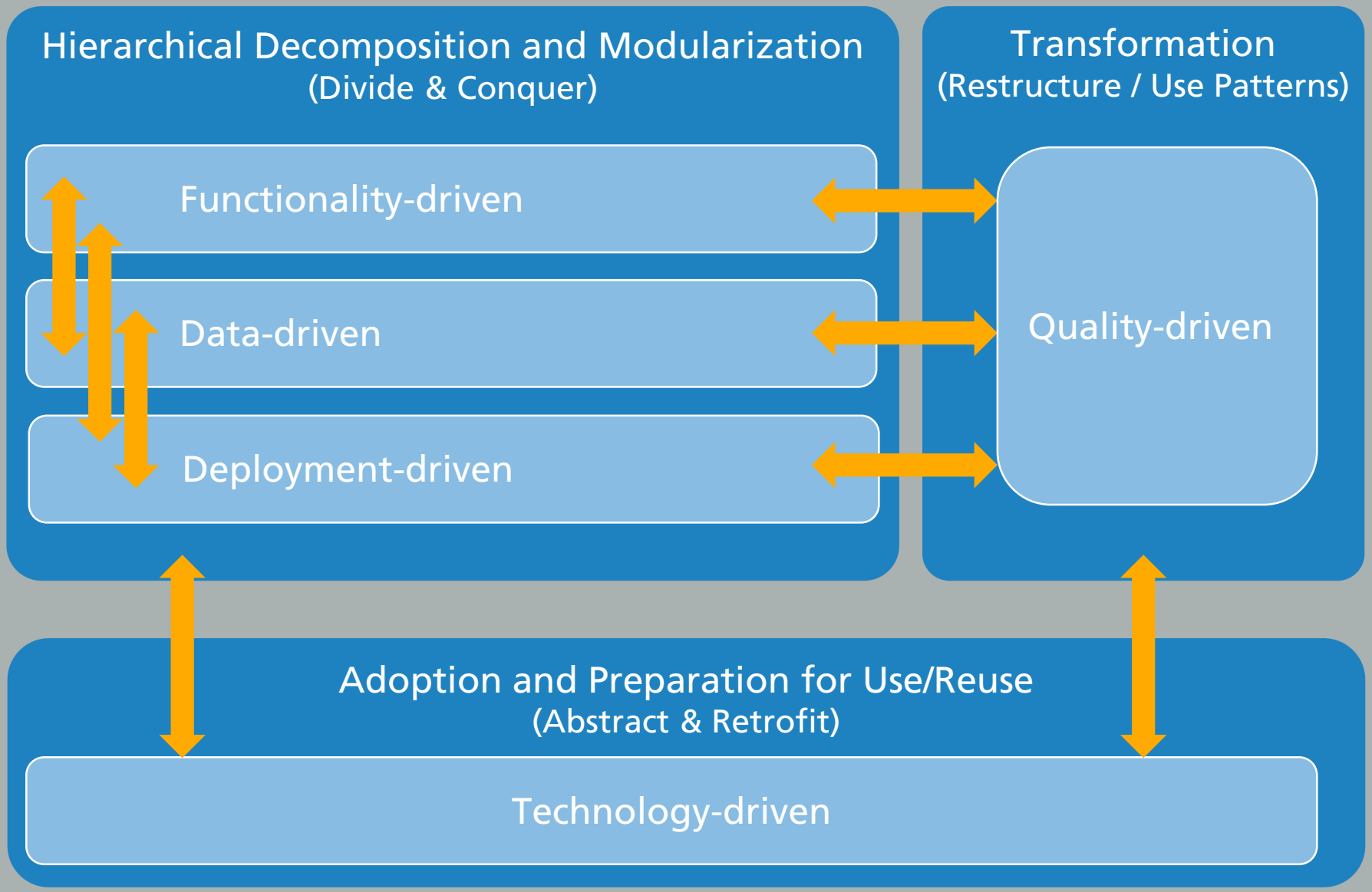
Architecture Design Process



Architecture Design Iteration



Architecture Decomposition Strategies



Context View

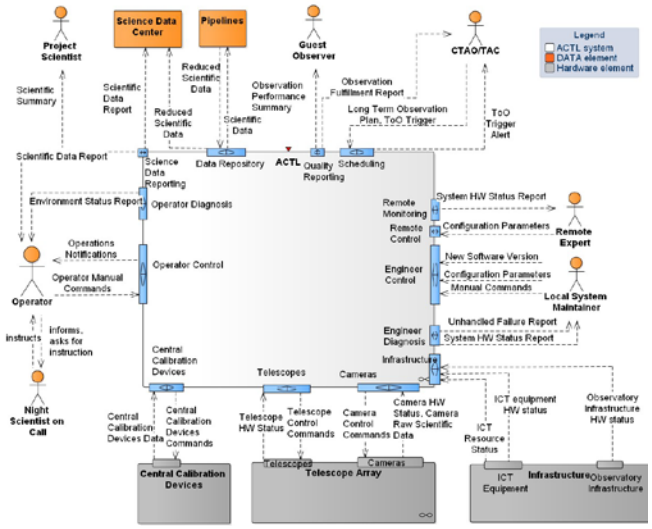
1 - Delineate System and its context

Architecture Design

Context

Main Purposes

- Definition of System Boundary
- Identification of Context:
  - Humans interacting with System
  - Context Systems
  - Information flow
- System seen as a black box!



Aspects to think about

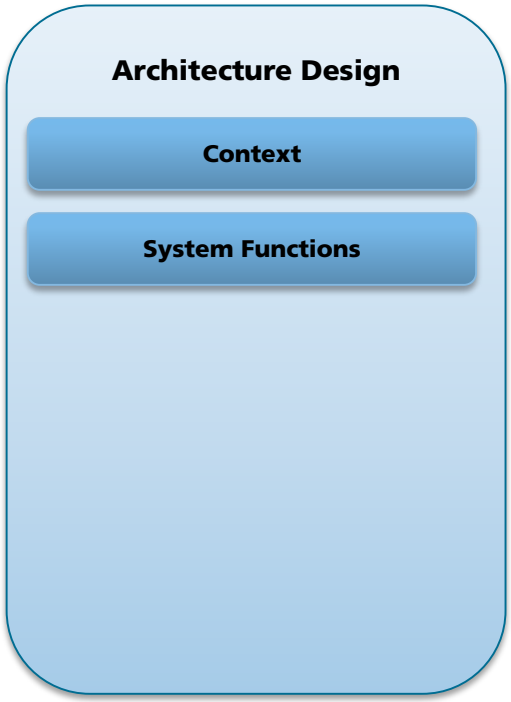
- What does the system do?
- What is the interaction with the environment?
- What data is exchanged?

Fraunhofer IESE

Functional View

1 - Delineate System and its context

2 - Functional decomposition and orchestration to grant the services



Main Purposes

- Identification and decomposition of systems functionality
- Linking (functional) requirements to later design steps
- Understanding of functional interrelationships
- Identify missing (functional) requirements

Functional View

1 - Delineate System and its context

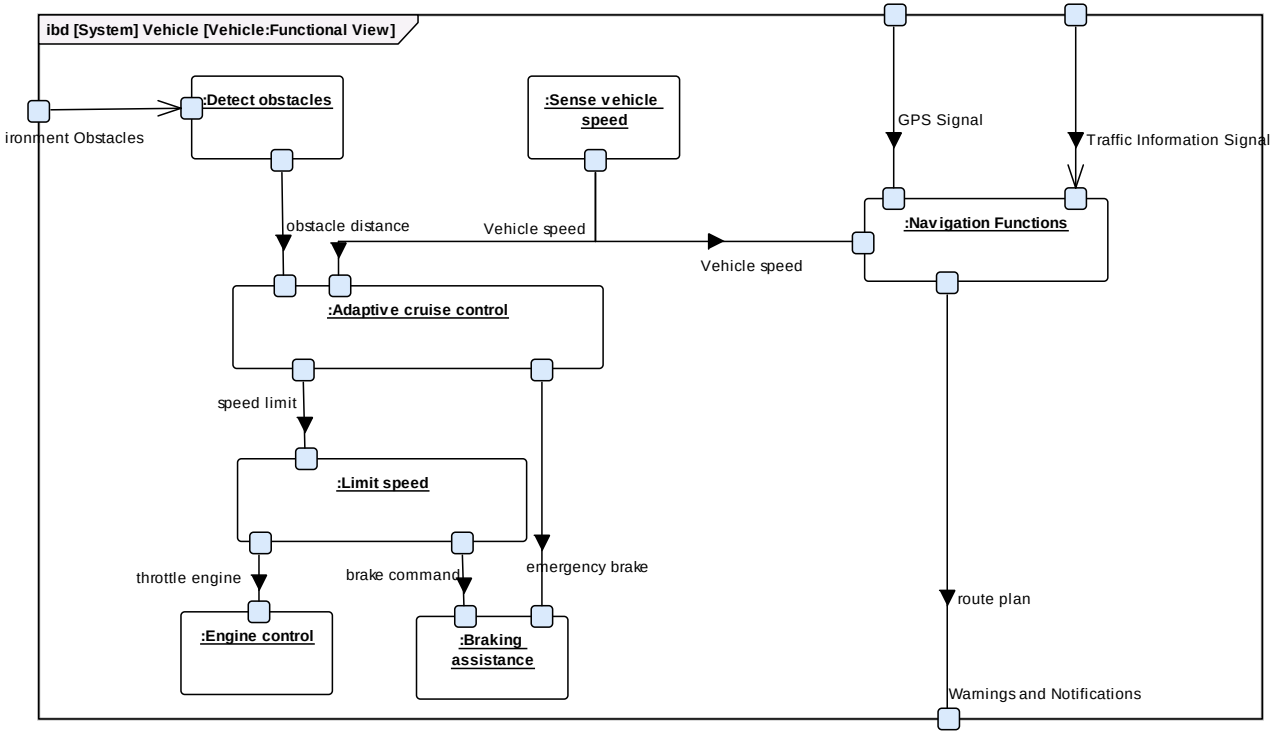
2 - Functional decomposition and orchestration to grant the services

Architecture Design

Context

System Functions

- Decomposition of functionalities.



Fraunhofer IESE

Functional View

1 - Delineate System and its context

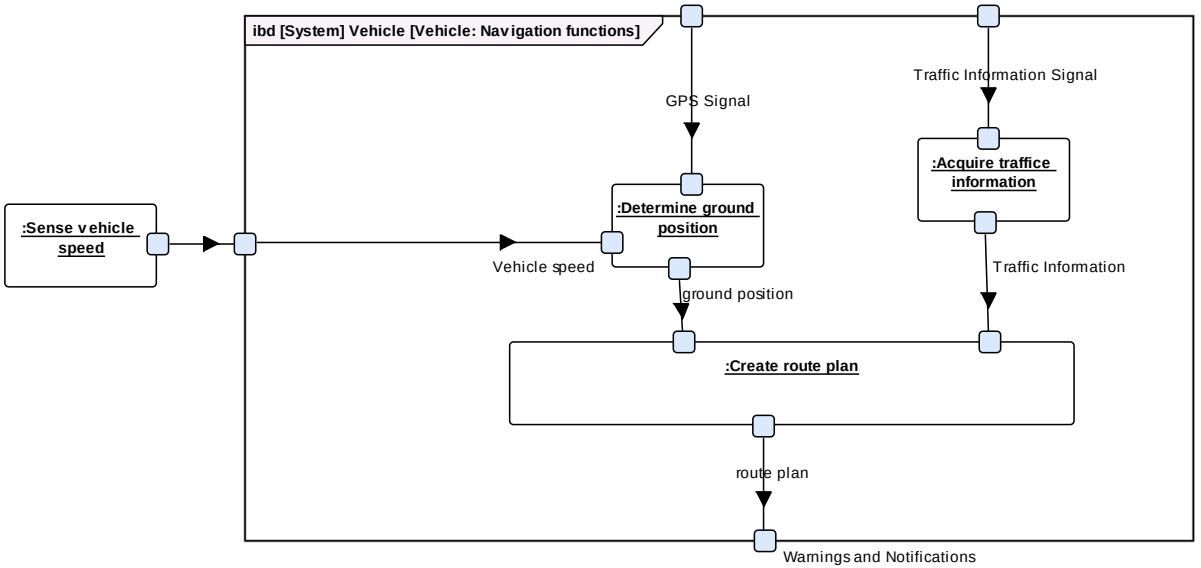
2 - Functional decomposition and orchestration to grant the services

Architecture Design

Context

System Functions

- Decomposition of functionalities.



Fraunhofer IESE

Functional View

1 - Delineate System and its context

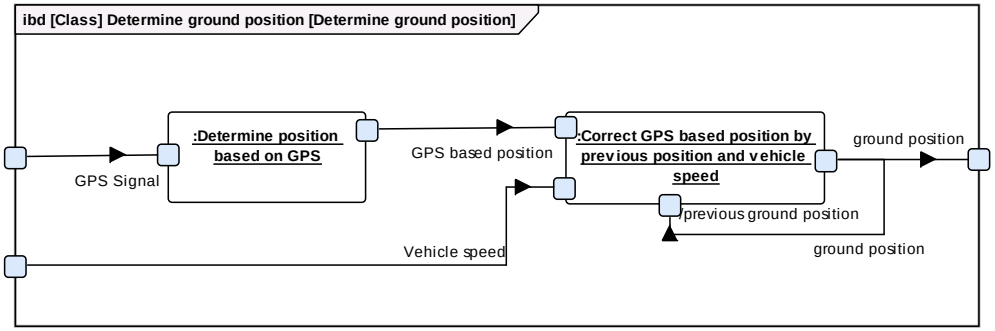
2 - Functional decomposition and orchestration to grant the services

Architecture Design

Context

System Functions

- Decomposition of functionalities.



Fraunhofer IESE

Functional View

1 - Delineate System and its context

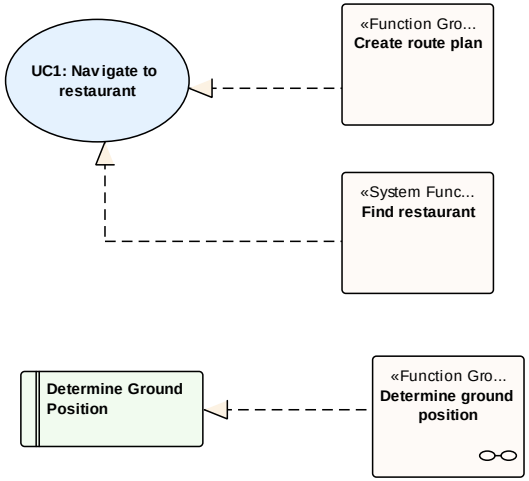
2 - Functional decomposition and orchestration to grant the services

Architecture Design

Context

System Functions

- Linking to functional requirements



Fraunhofer IESE

Functional View

1 - Delineate System and its context

2 - Functional decomposition and orchestration to grant the services

Architecture Design

Context

System Functions

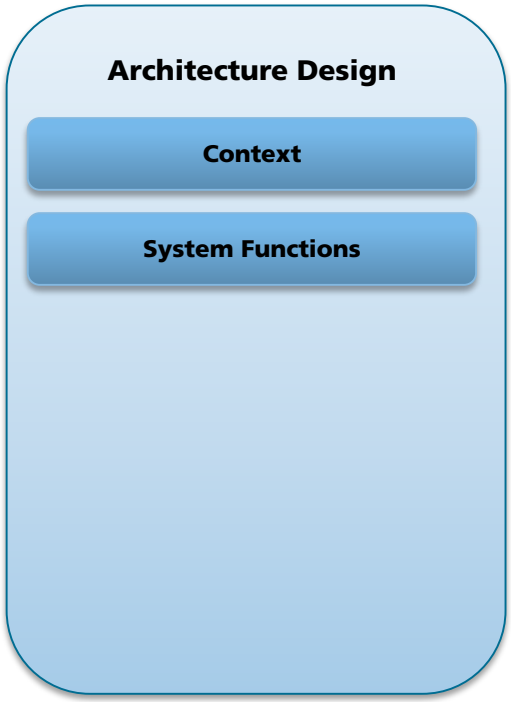
- Consolidate Requirements & Functional View

Source \ Target	Brew Coffee	Interrupt Brewing	Observe Clock Time	Warm up Coffee
Control boiler heater	↑			
Control relief valve	↑	↑		
Indicator Light				↑
Observe plate sensor	↑	↑		↑
Observe start button	↑			
Observe water tank	↑			
Plate Heater		↑		

Functional View

1 - Delineate System and its context

2 - Functional decomposition and orchestration to grant the services



Main Purposes

- Identification and decomposition of systems functionality
- Linking (functional) requirements to later design steps
- Understanding of functional interrelationships
- Identify missing (functional) requirements

Aspects to think about

- Which functions are responsible for Data
  - Creation
  - Transportation
  - Processing and Storage
- Which functions communicate?
- What data is exchanged?

Logical View

1 - Delineate System and its context

2 - Functional decomposition and orchestration to grant the services

3 – Reason on the realization (technology-independent)

Main Purposes

- Describe the internal logical structure of the SUD
- Partition the system into communicating logical components
- Allocate desired functions to cohesive logical units
- Support the reuse of already existent logical components
- Define the total behavior of the system

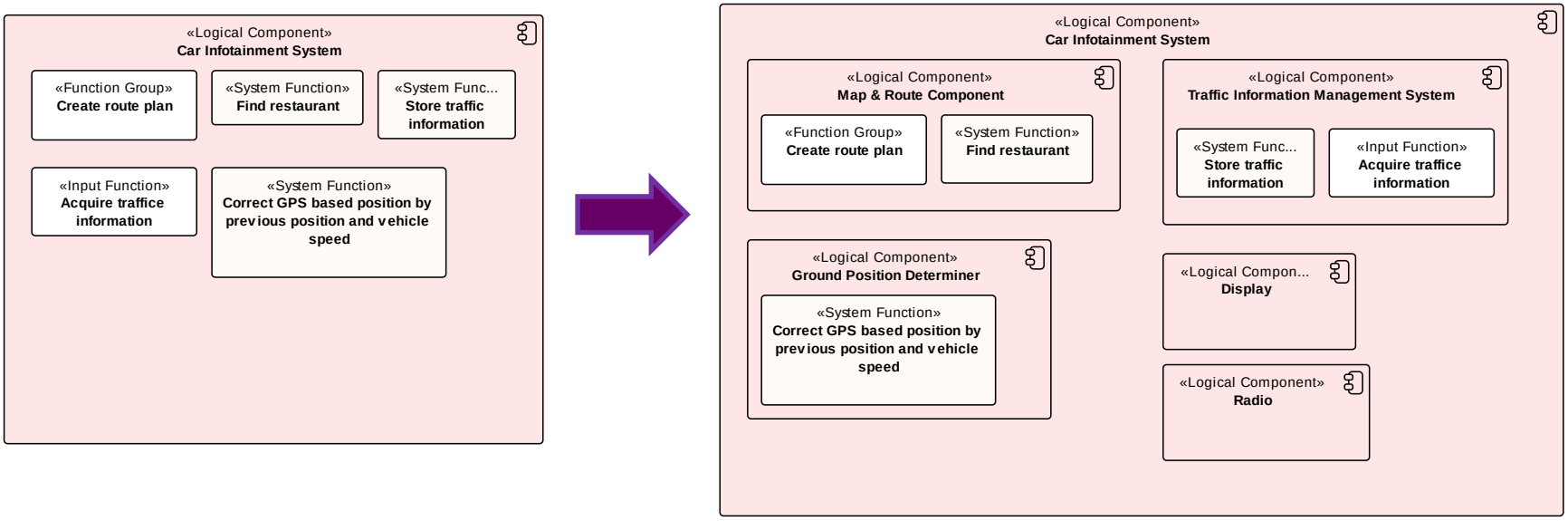
Logical View

1 - Delineate System and its context

2 - Functional decomposition and orchestration to grant the services

3 – Reason on the realization (technology-independent)

- Mapping of functions to logical components



Fraunhofer IESE

Logical View

1 - Delineate System and its context

2 - Functional decomposition and orchestration to grant the services

3 – Reason on the realization (technology-independent)

- Mapping of functions to logical components: Aspects to consider
  - Increase cohesion: Semantically similar functions go together
    - Development Expertise can be concentrated in one component
    - Similar functions often have the same ASIL-level. Less components can be developed using more rigor. See e.g. Microkernel approach
  - Decrease Coupling: Functions with many interactions go together
    - Reduce communication overhead
    - Reduce testing effort
    - Reduce integration effort

Logical View

1 - Delineate System and its context

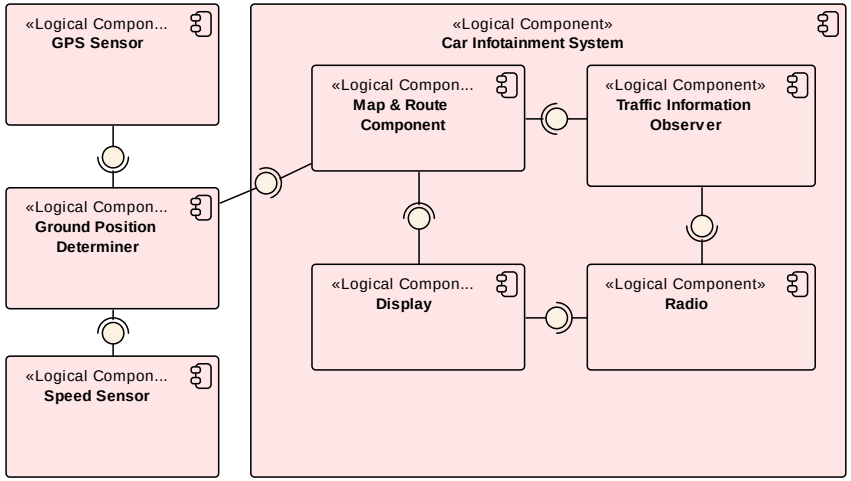
2 - Functional decomposition and orchestration to grant the services

3 – Reason on the realization (technology-independent)

Architecture Design

- **Context**
- **System Functions**
- **Logical Components**

- Identification of Interfaces between logical components



Fraunhofer IESE

Logical View

1 - Delineate System and its context

2 - Functional decomposition and orchestration to grant the services

3 – Reason on the realization (technology-independent)

Aspects to think about

- How should functionality be allocated to executable components?
- How can components be refined into sub-components?
- Are there commonalities between components?
  - So that redundant code can be prevented
- What data is exchanged between the components
- What are the interfaces to be used?
- What Behavior does a component have (statemachine, activity, …)?

Software View

- **Main Purposes**
- Decomposition of software components into sub-components and classes
- Definition of Software Interfaces
- Identification of necessary software datatypes

Aspects to think about

- How are logical components realized by software components?
- How should components be implemented by implementation units
- What are the interfaces to be used
- Which component provides/requires an interface
- What datatypes should be used for data-exchange?

Software View

1 - Delineate System and its context

2 - Functional decomposition and orchestration to grant the services

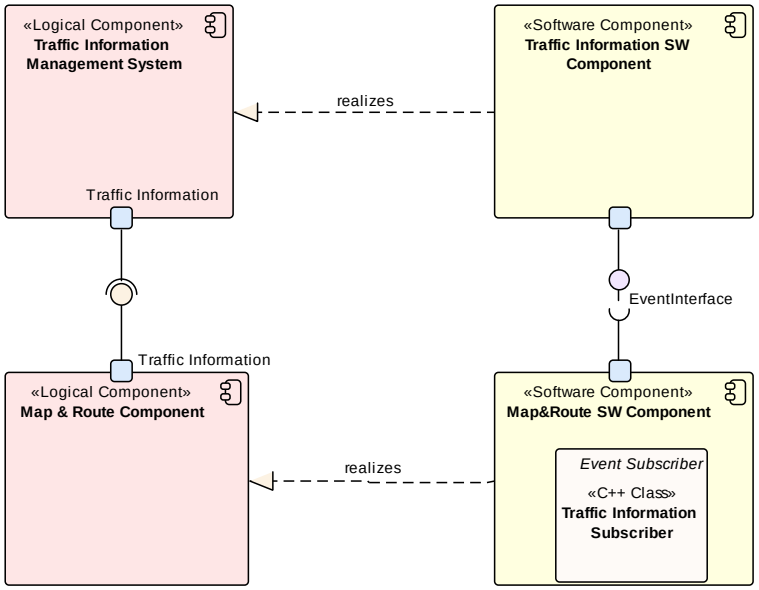
3 – Reason on the realization (technology-independent)

4 - Software Realization of Functions

Architecture Design

- **Context**
- **System Functions**
- **Logical Components**
- **Software Entities**

- How are logical components realized by software components?



Fraunhofer IESE

Software View

1 - Delineate System and its context

2 - Functional decomposition and orchestration to grant the services

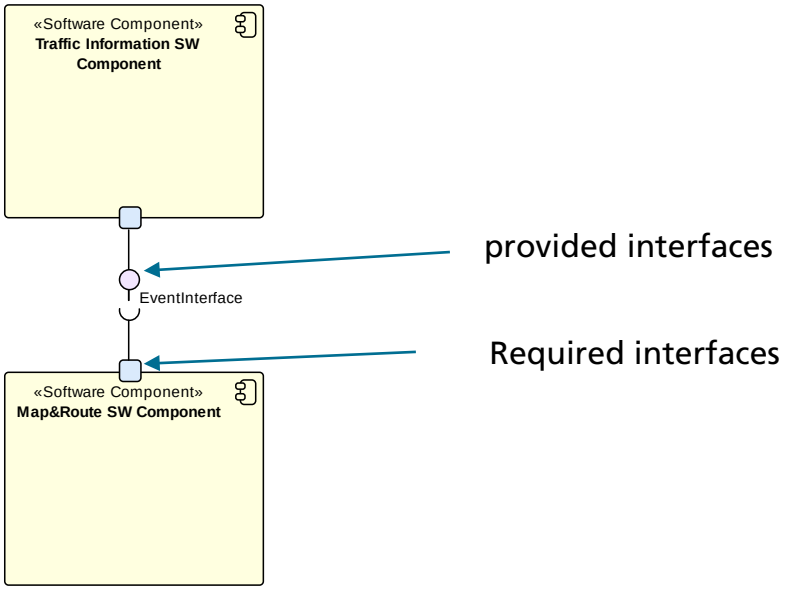
3 – Reason on the realization (technology-independent)

4 - Software Realization of Functions

Architecture Design

- **Context**
- **System Functions**
- **Logical Components**
- **Software Entities**

- Which component provides/requires an interface



Fraunhofer IESE

Software View

1 - Delineate System and its context

2 - Functional decomposition and orchestration to grant the services

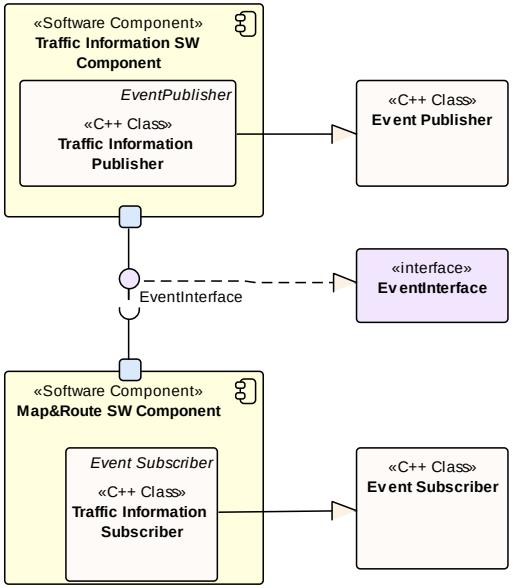
3 – Reason on the realization (technology-independent)

4 - Software Realization of Functions

Architecture Design

- **Context**
- **System Functions**
- **Logical Components**
- **Software Entities**

- How are interfaces implemented?



Fraunhofer IESE

Software View

1 - Delineate System and its context

2 - Functional decomposition and orchestration to grant the services

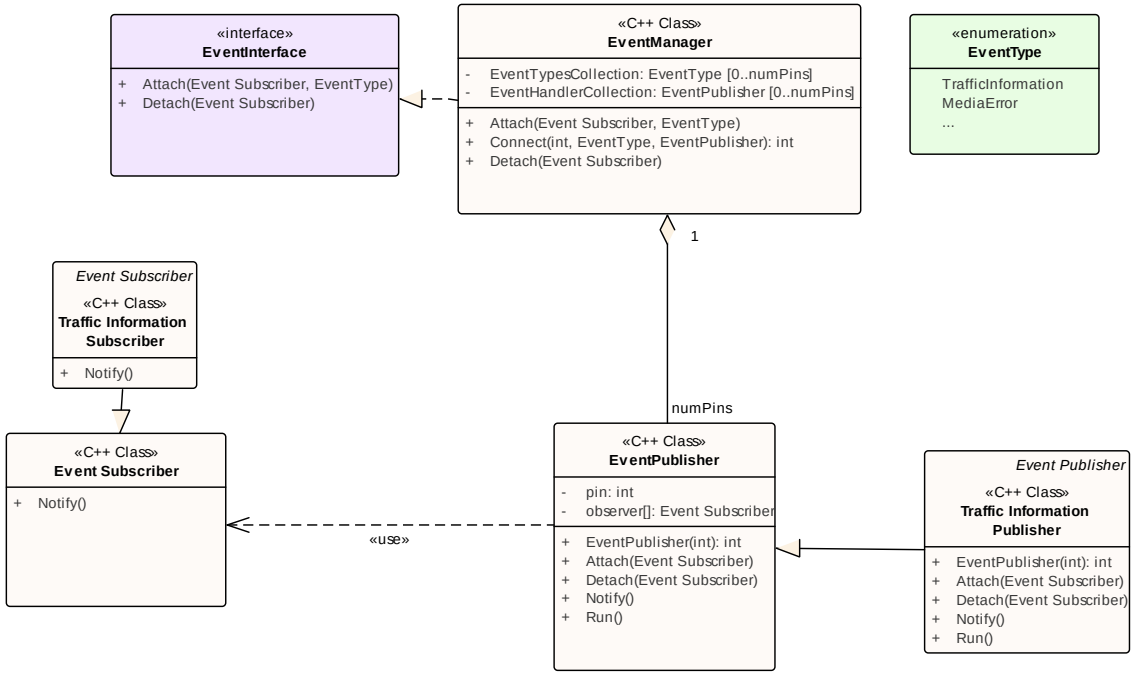
3 – Reason on the realization (technology-independent)

4 - Software Realization of Functions

Architecture Design

- **Context**
- **System Functions**
- **Logical Components**
- **Software Entities**

- Architecture relevant classes, interfaces, datatypes



Fraunhofer IESE

Hardware View

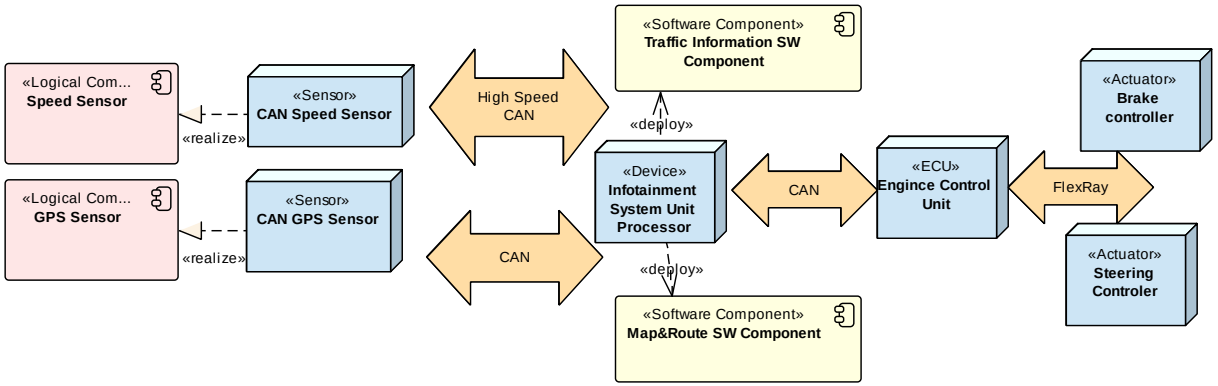
1 - Delineate System and its context → **2 -** Functional decomposition and orchestration to grant the services → **3 –** Reason on the realization (technology-independent) → **4 -** Software Realization of Functions → **5 -** Hardware realization of functions and Deployment of SW Entities

Architecture Design

- **Context**
- **System Functions**
- **Logical Components**
- **Software Entities**
- **Hardware & Networks**

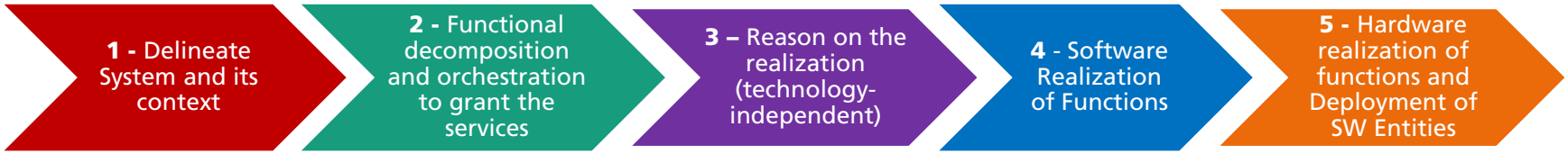
Main Purposes

- Identification of Sensors, actuators, ECUs
- Identification of Communication buses
- Deployment of software components
- Functional realization by hardware entities



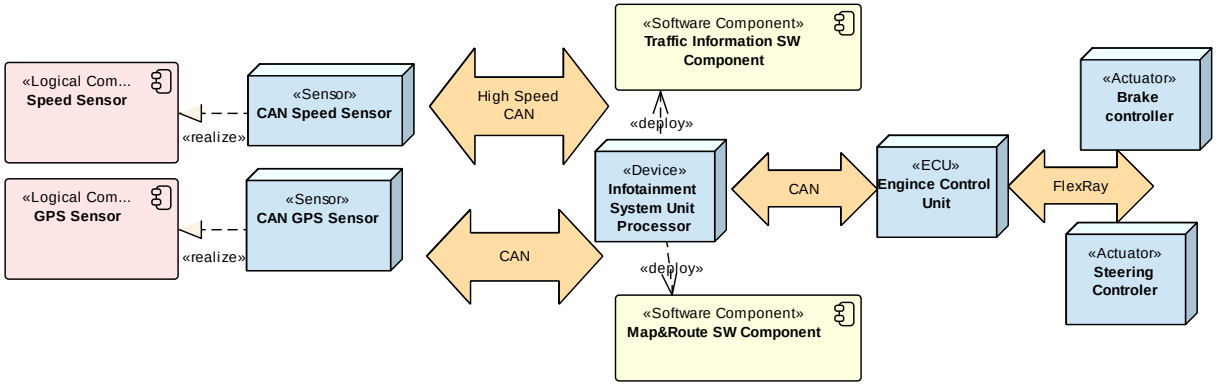
Fraunhofer IESE

Hardware View



Aspects to think about

- How should software components be deployed to computing nodes?
- What bus systems are used to exchange data?
- Which logical components should be implemented in hardware?
  - Which hardware manufacturer/type should be chosen?



Logical vs. HW/SW View

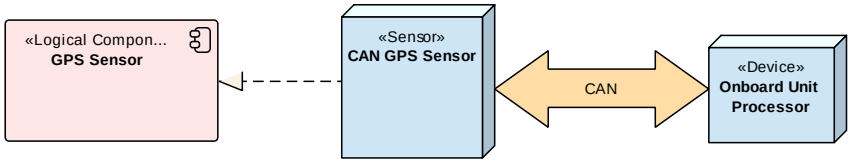
1 - Delineate System and its context → 2 - Functional decomposition and orchestration to grant the services → 3 – Reason on the realization (technology-independent) → 4 - Software Realization of Functions → 5 - Hardware realization of functions and Deployment of SW Entities

Architecture Design

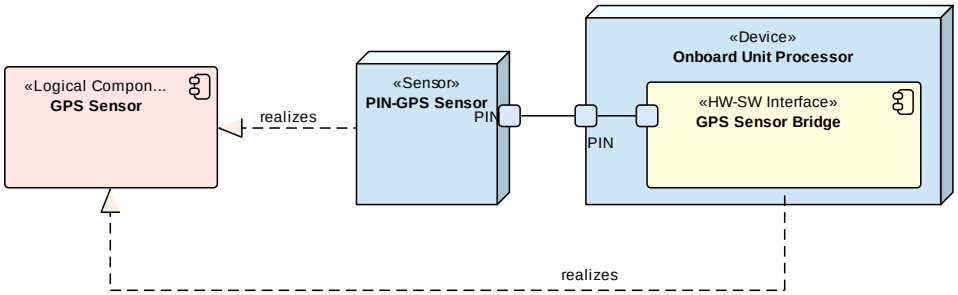
- Context
- System Functions
- Logical Components
- Software Entities
- Hardware & Networks

- Logical View should be Technology-independent
  - Defer design decision as late as possible

Alt.1



Alt.2



Fraunhofer IESE

When to Stop Working at the Architecture Level?

- You should stop when…
  - … you addressed the key requirements and quality attributes
  - … you can explain how they are addressed
  - … and have enough **confidence** that they can be achieved
  - … and you can assign work units to developers
  - … and you can control the parallel development and integration
- You might temporarily leave the architecture level…
  - … to collect information, get more confidence
  - … if you do not exactly know what you abstract from and have to try out
  - … in prototyping activities and technical evaluations
- You always have to come back to the architecture level…
  - … to integrate your lessons learned
  - … to judge the results in the context
  - … to reason about change

When to Stop Designing?

- You have covered the most important things when…
  - … you addressed the key requirements and quality attributes
  - … you can explain how they are addressed
  - … and have enough confidence that they can be achieved
  - … and you can assign work units to developers
  - … and you can control their parallel development and integration

- You continue designing during development because …
  - … you refine architectural decisions
  - … you design the methods, data structures
  - … you implement solutions (source code and test cases)
  - … you make the system work
- **But you need to make sure not to break the architecture!**


architecture
software
development
systems
requirements
Architecture
time
product
quality
technologies
decisions
system
support
architectures
competencies
key
activities
context
evaluation
SAVE
control
products
architecture-centric
Systems
organizations
IESE
technology
architectural
challenges
aspects
basis
Tekla
making
particular
practices
like
design
generation
business
services
compliance
provide
Fraunhofer
processes
evolution
projects
make
clear
level
results
many
models
customers
information
usage
need
implementation
organization
packages
goals
Therefore
engineering
use
one
Engineering
new
order
terms
documentation
help