

12.24196

# Introduction to Embedded Systems

Prof. Dr.-Ing. Stefan Kowalewski | Julius Kahle, M. Sc.  
Summer Semester 2025

Part 5

## Embedded Software Development & Design

# Agenda

IES Part 7 – Embedded Software Development & Design

- I. Development Process Models
- II. Analysis of Functional Requirements
- III. Analysis of Quality Requirements
- IV. Architecture Design



# Part I

## Development Process Models

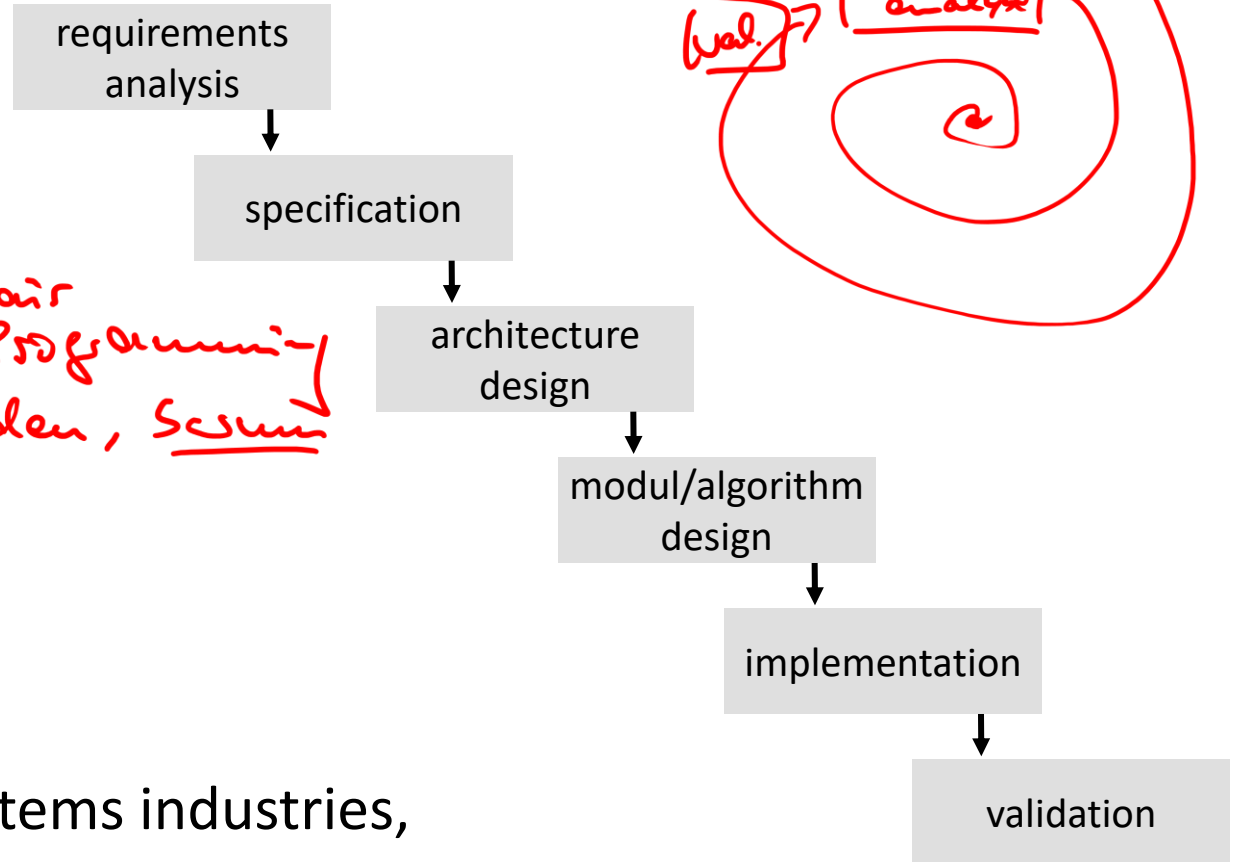
## A process model

- ▶ structures the development and maintenance process into distinguishable **steps**
- ▶ defines **work products** of the steps
- ▶ specifies the possible **sequences and repetitions** of the steps
- ▶ defines **roles** for the participants of the development and maintenance process and their **responsibilities**

# Process models /2

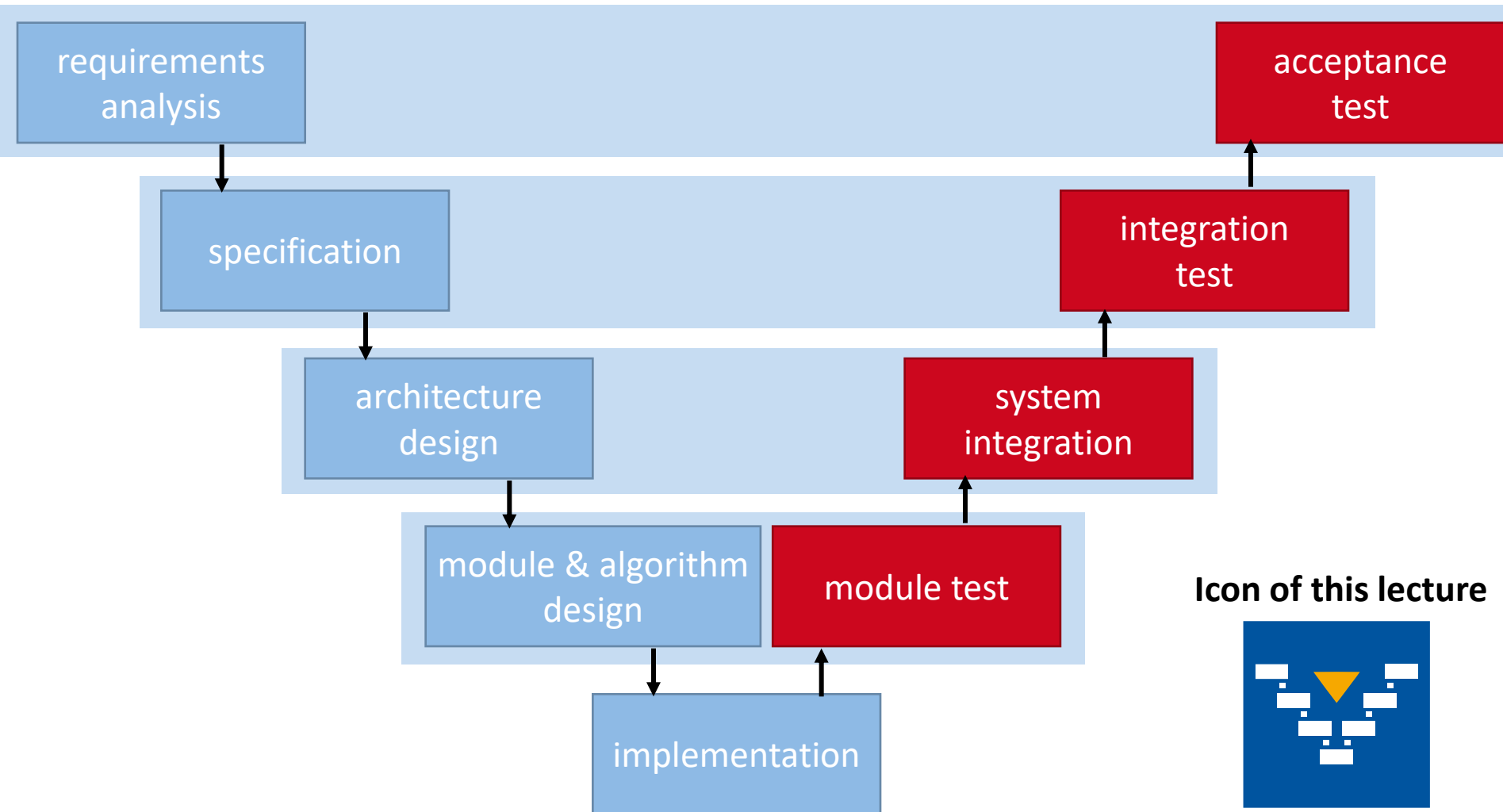
## Examples:

- ▶ Waterfall
  - ▶ Spiral
  - ▶ Extreme Programming
- Pair Programming*  
*Agile Methoden, Scrum*

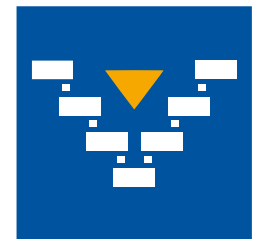


In the embedded systems industries, the most popular process model is the **V model**.

# V model



Icon of this lecture



## ▶ Requirements analysis

- Functional reqs (e.g., control loop performance)
- non-functional reqs (e.g., safety, reliability, maintainability)

## ▶ Architecture design

- Systematic approach
- Current standards/concepts  
(e.g., AUTOSAR, CAN/Flexray)

## ▶ Module/algorithm design

- Model-based approaches (e.g., Simulink/Stateflow)
- ~~Controller design/tuning~~

# Requirements analysis

## ► Objective:

- Determine and document the required functionality and properties of the system (**What** and **how good, not how**)
- Determine and document how the achievement of the functionality and properties can be validated or measured, i.e. the **test cases** for the acceptance test

## ► Results:

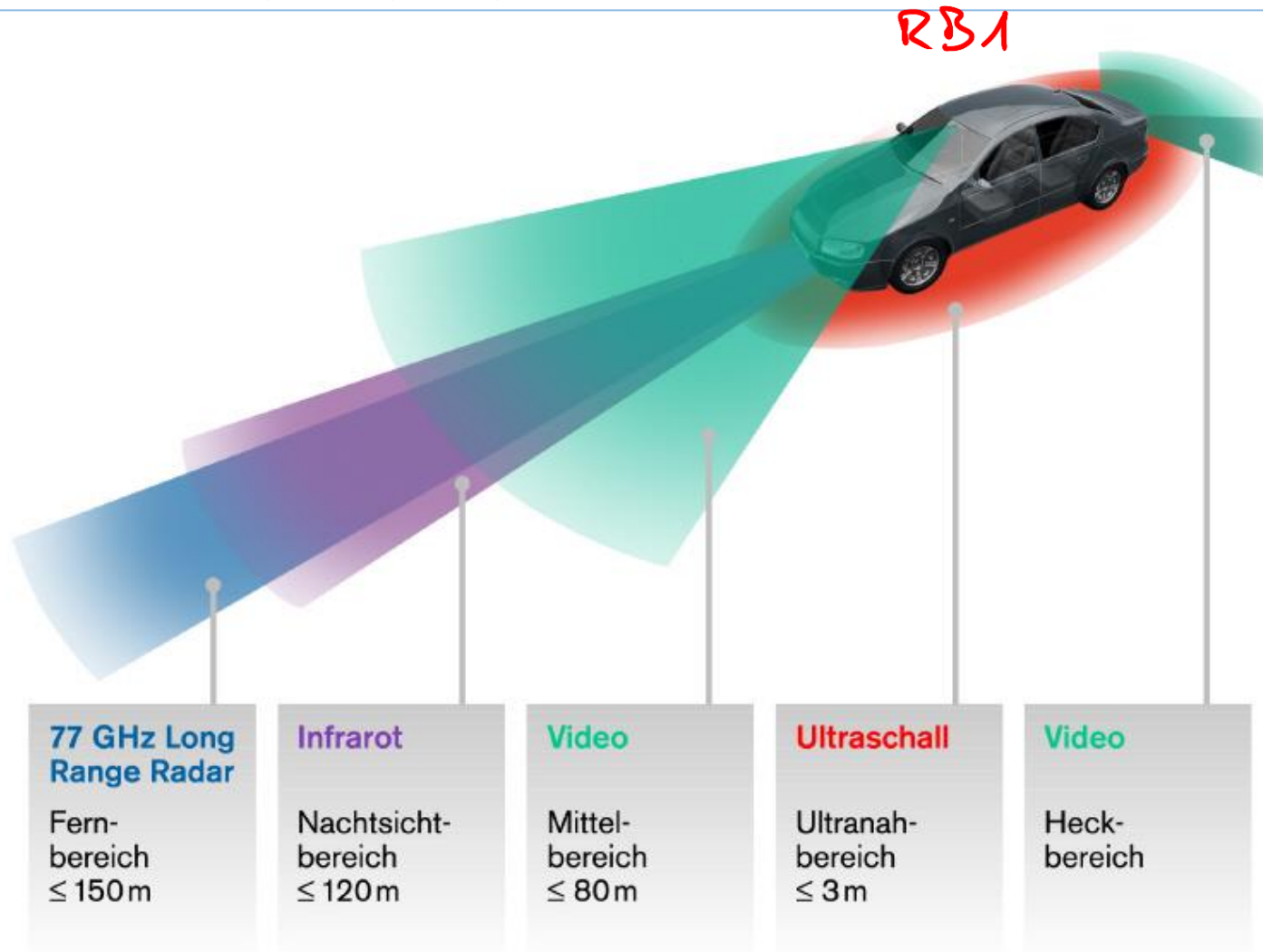
- Specification document
- Acceptance test plan
- Possibly first version of system manual

↗ Anforderungen → Lastenheft  
Spezifikation → Pflichtenheft



# Example: Adaptive Cruise Control (ACC)

## Context: Car Periphery Supervision



Graphics: Bosch

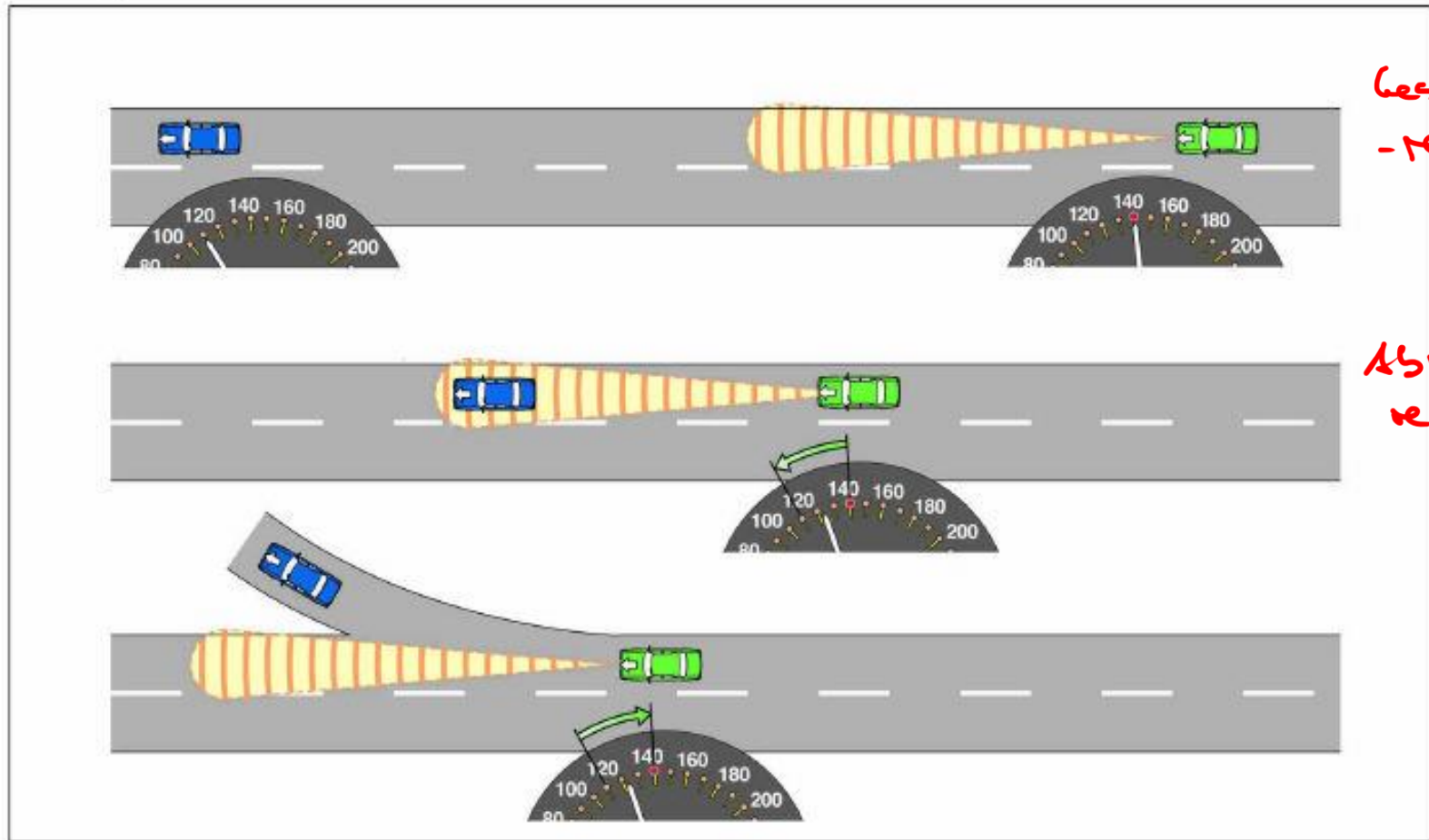
# Example ACC: Functionality

## Adaptive Cruise Control ACC von Bosch

Tempomat

Geschwindigkeitsregler

Abstandsregler



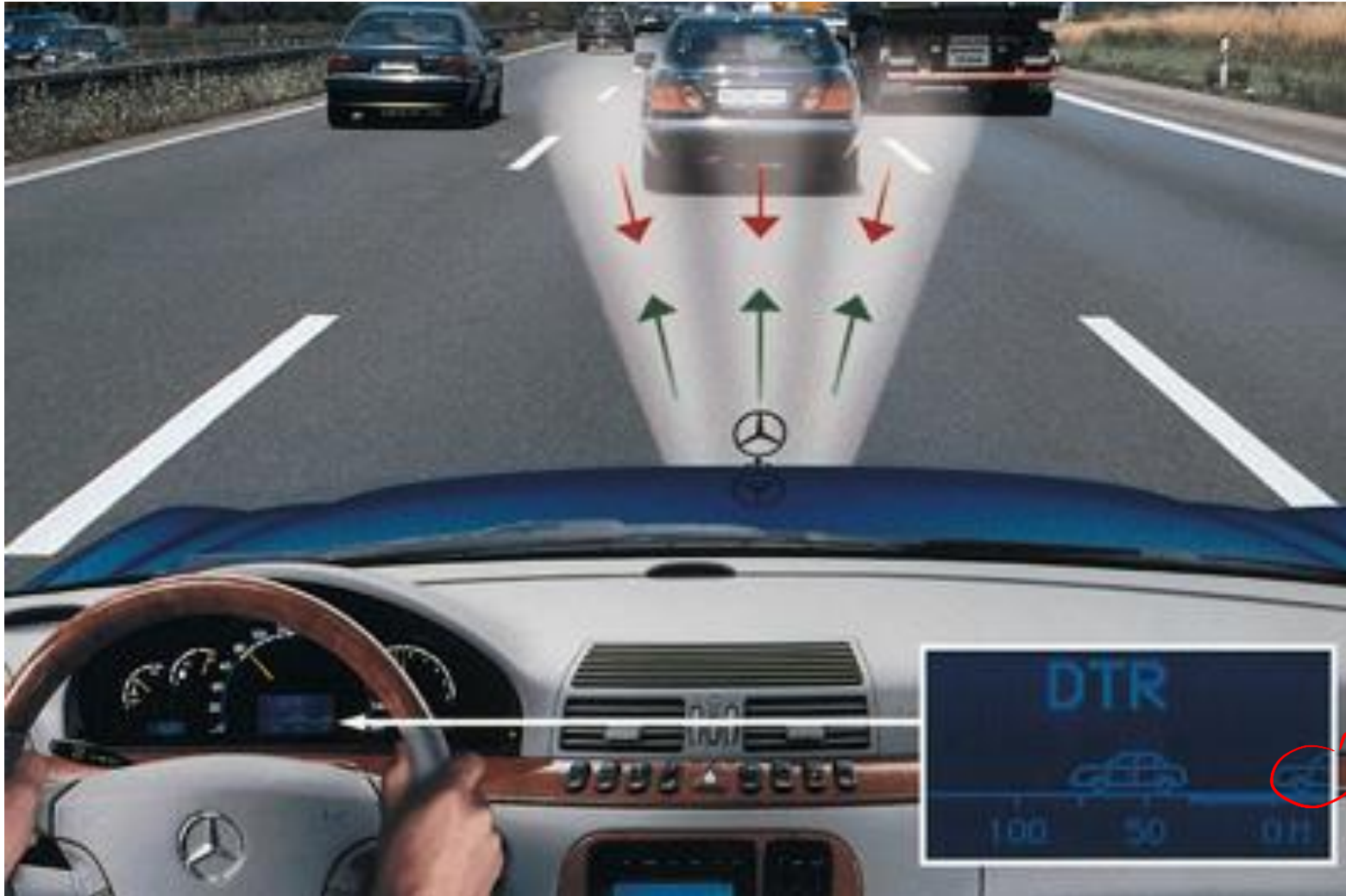
**BOSCH**



Nachdruck honorarfrei mit Vermerk "Foto: Bosch".

Pressebild-Nr. 9185

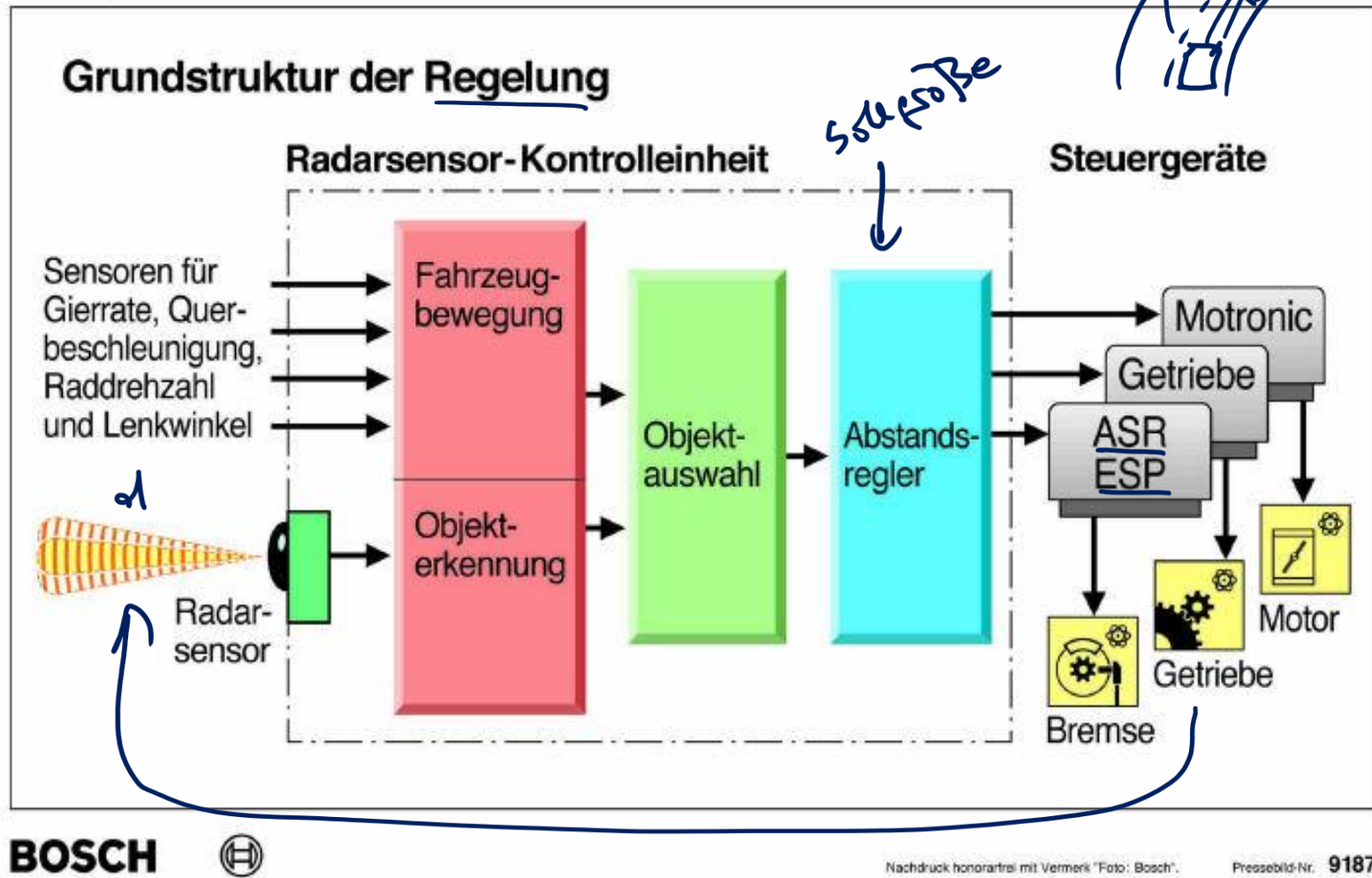
# Example ACC: User Interface



Graphics: DaimlerChrysler

# Example ACC: Structure

## Adaptive Cruise Control ACC von Bosch

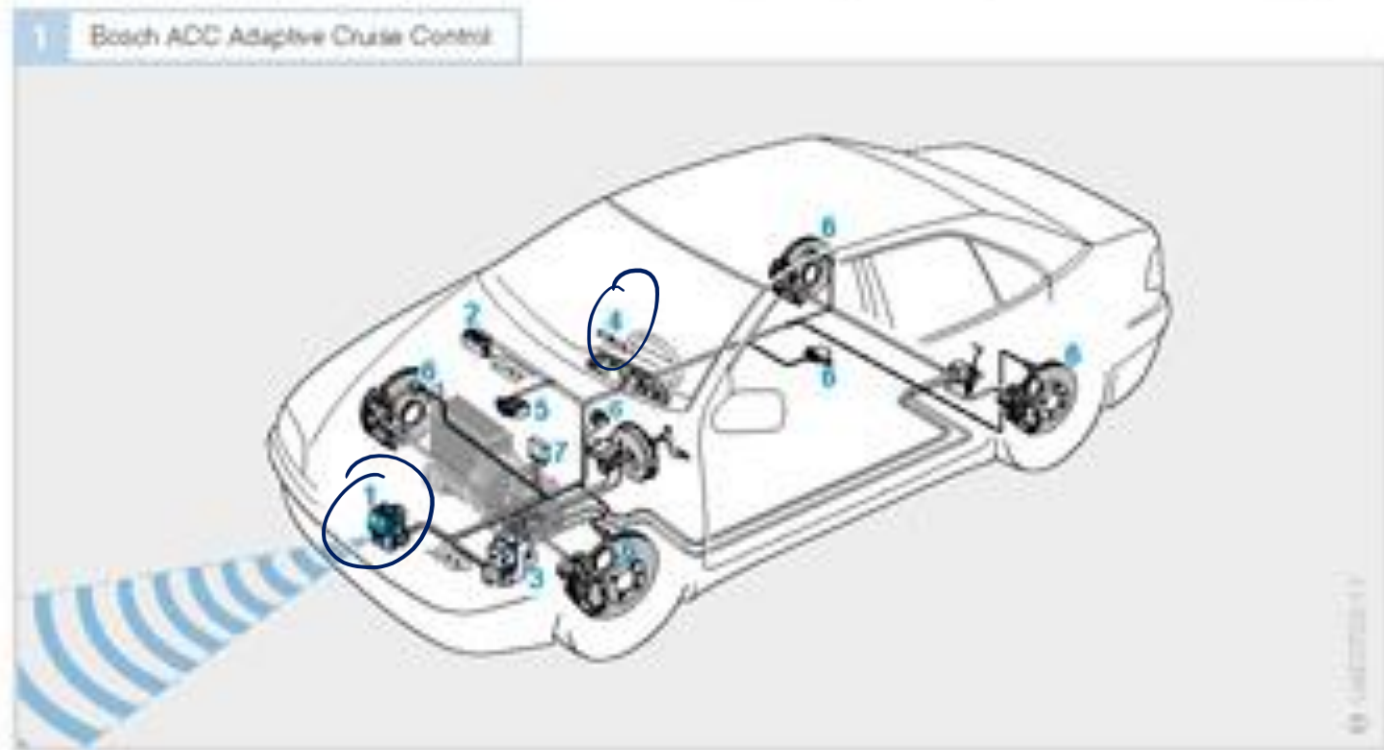


Graphics: Bosch

# Example ACC: Structure /2

Fig. 1

- 1 ACC sensor & control unit
- 2 Engine management ECU
- 3 Active intervention in braking via ESP
- 4 Controls and display
- 5 Intervention at the engine via EM system with ETC (Electronic Throttle Control or EGAS) (gasoline engines) or EDC (diesel engines)
- 6 Sensors
- 7 Transmission intervention (optional)



Graphics: Bosch



# Example ACC: Sensor



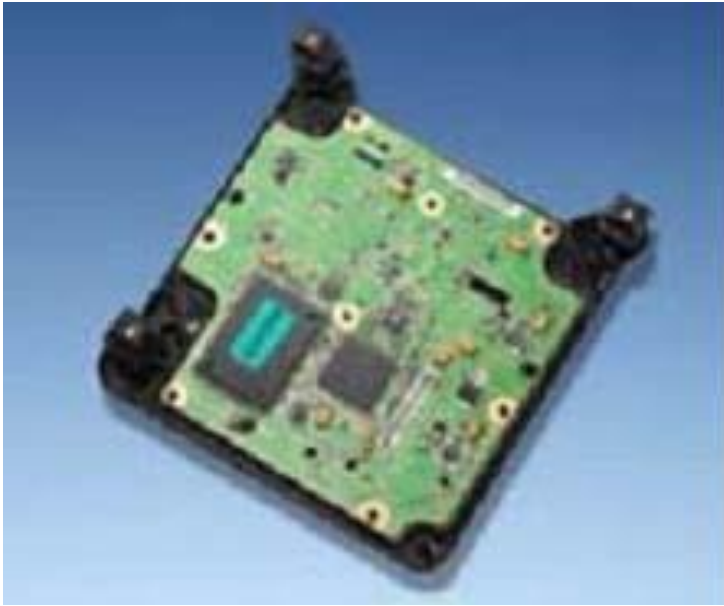
Photo: Bosch

# Example ACC: Sensor /2

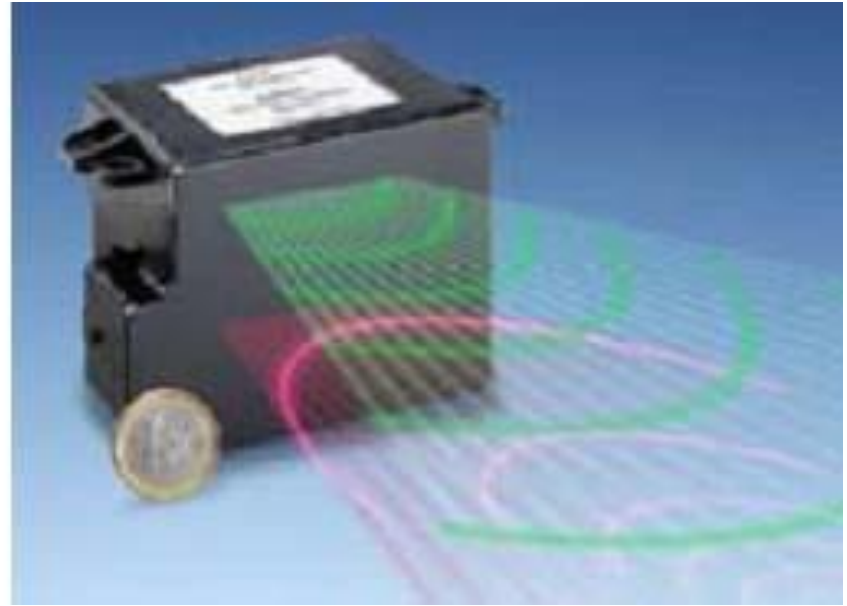


Photo: Bosch

# Exampe ACC: Sensor /3



**24 GHz Short Range Radar**



**Infrared**

Graphics: Continental



# Requirements for ACC /1

- ▶ ACC shall detect vehicles in front of the car in a distance from 0 to 120 meters and in an angle of  $\pm 4^\circ$  relative to the centerline of the car.
- ▶ ACC shall determine the velocity of detected vehicles
- ▶ If the road is free, ACC will keep the car at a specified cruise velocity.
- ▶ If a slower vehicle is in front of the car, ACC will reduce speed until a specified distance to the front vehicle is achieved. ACC will then keep the distance constant until the car either vanishes or accelerates above the specified cruise velocity.
- ▶ ACC can be activated between 30 and 180 km/h.

ACC stop & go  
0 - 30 km/h

# Requirements for ACC /2

- ▶ In distance control mode, deviations from the reference distance must not exceed 5% for more than 2 seconds assuming a maximal acceleration or deceleration of the front vehicle of 0.5 g.
- ▶ The distance control algorithm shall be a PI controller.
- ▶ If ACC is malfunctioning, it has to switch off automatically and indicate by a red light and a short beep that it is out of service. It must never indicate a safe distance to the front vehicle when this is not the case.
- ▶ ACC must not be out of function for more than  $10e-7$  hours per year of operation time.
- ▶ Distance and cruise speed selections must be well readable for the driver.

# Requirements for ACC /3

---

- ▶ Speed and direction angle information are provided by the ESP system via the CAN bus.
- ▶ For customer A, the ACC software has to run on a Freescale MPC5200. Customer B wants to integrate the ACC software into his already existing engine control unit.
- ▶ Manufacturer A wants a 77 Ghz radar sensor, manufacturer B an infrared sensor, manufacturer C wants to combine radar and video information.
- ▶ Marketing plans to offer ACC Stop&Go with an additional 24 GHz short range radar sensor for next year.

# Requirements Engineering

---

**Requirements Engineering** consists of three parts:

▶ Requirements **Elicitation**

- Collect the requirements from customers, marketing, system engineers etc.

▶ Requirements **Analysis**

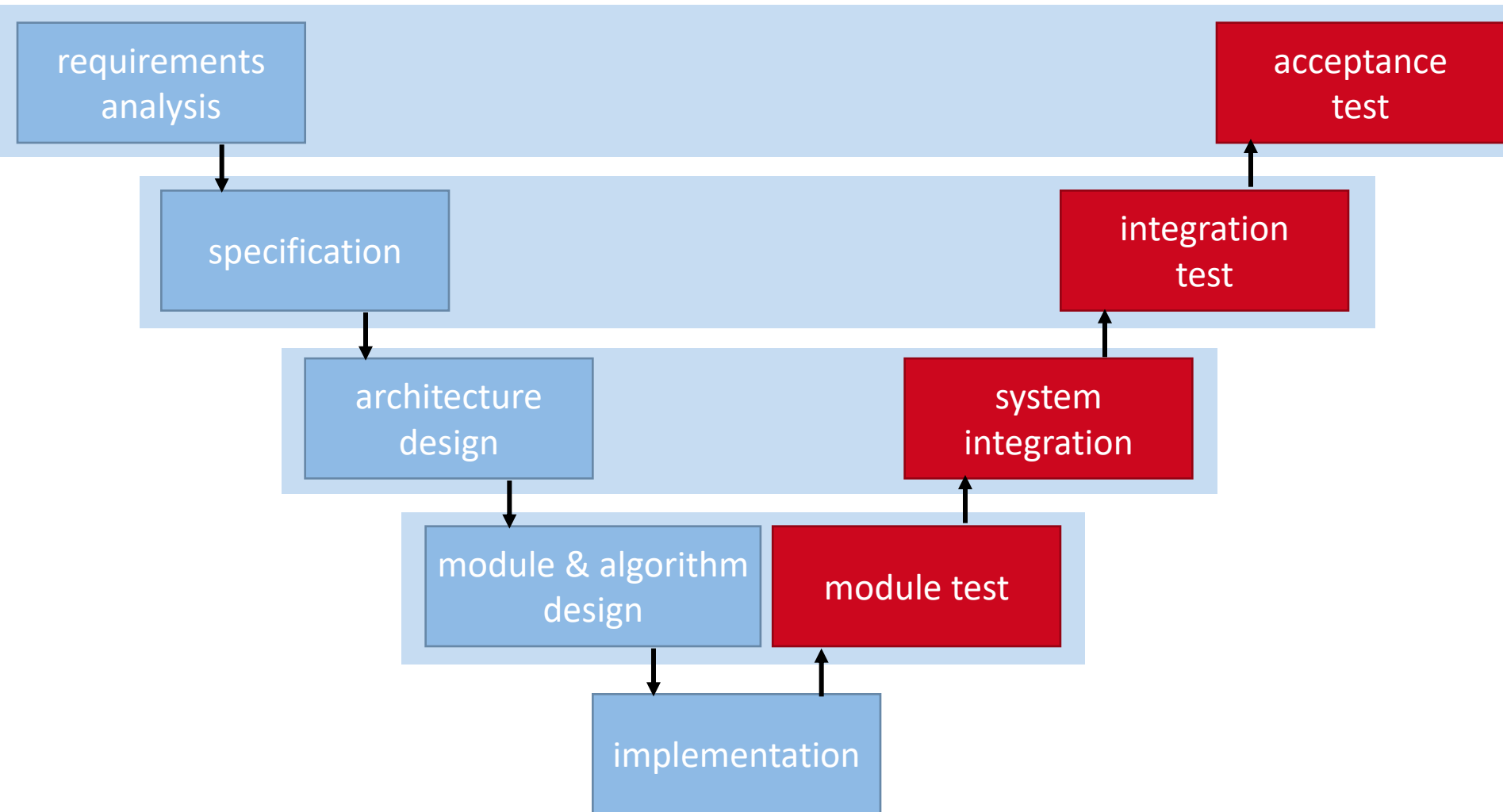
- Analyse whether the requirements are actually what the customers, marketing, system engineers etc. want.

▶ Requirements **Management**

- Manage changes of the requirements (document, estimate costs, check technical possibility, delegate to developers, charge customer)

**To which phase of the V-model does Requirements Management belong?**

# V model



# What do you need to know about requirements elicitation and analysis?

## ▶ There are **techniques** for requirement elicitation:

- Guided interviews
- Brainstorming
- Role plays, etc.

(→ literature)

## ▶ There are **models** for requirement analysis:

- Use cases
- Sequence diagrams
- Data flow diagrams, etc. (→ dynamic systems lecture)

## ▶ For successful elicitation and analysis of requirements, it is most important to know a few **basic concepts** and **typical problems**.

# Requirements vs. solutions

---

- ▶ A requirement should state **what** the system shall do or **how good** it shall do it, not **how**.
- ▶ Often, customers tend to provide **solution** aspects with the requirements.
- ▶ Example:  
**The detected objects shall be stored in a linked list.**
- ▶ Often the customer is not aware that he/she demands a particular solution and agrees on replacing it by a proper requirement.
- ▶ Example:  
**The detected objects have to be stored such that they can be processed efficiently.**

# Requirements vs. constraints

---

- ▶ If a demanded solution cannot be replaced by a requirement, it is called a **constraint**.
- ▶ Example:  
**The system must use Microsoft Windows CE as operating system.**
- ▶ Behind every constraint there is a rationale which can not be questioned by the developer.
- ▶ Example:  
**The management of the company signed a strategic partnership with Microsoft.**



# Functional vs. Quality requirements

*non-functional*

- ▶ Difference between **what** the system shall do and **how good** it shall do it.

- ▶ Examples:

Function (**what**):

- The system must keep the distance to the vehicle in front of the car constant.

Quality (**how good**, also **non-functional requirements**):

- The driver must be able to adjust the desired distance without taking the hands from the steering wheel.

(**Usability**)

- Deviations from the desired distance must not exceed 5%.

(**Reliability**)

- The distance control algorithm must be easily replaced with a customer-owned algorithm.

(**Modifiability, Integrability**)

# Sample Qualities

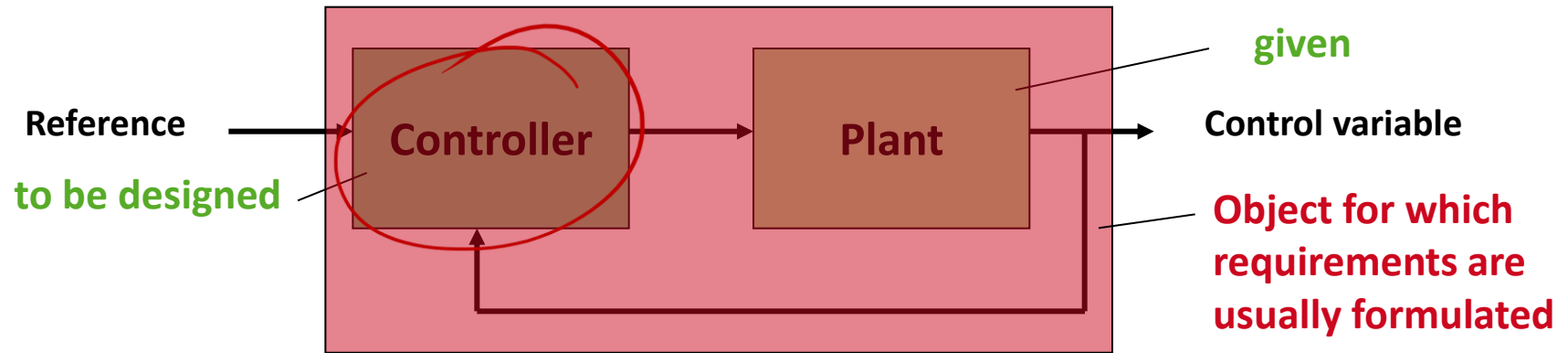
---

- ▶ Maintainability
  - Modifiability
  - Integrability
- ▶ Usability
- ▶ Dependability
  - Reliability
  - Availability
  - Safety
  - Security

## Particular for embedded systems:

- ▶ System Cost
- ▶ Mounting space
- ▶ Power consumption

# Closed loop vs. Controller requirements



- ▶ Distinguish between requirements for **embedded** system and **embedding** system.
- ▶ Particular for embedded software:

**The presumed properties and behaviour of the plant (environment) must be elicited and analysed in the requirements phase!**

**ACC example?**

# Requirements must be checkable

- ▶ Both functional and non-functional requirements must be formulated such that their fulfillment can be checked.
- ▶ Examples:
  - **Bad:** The ACC system shall react properly when the car in front becomes too slow.
  - **Good:** If the speed of car in front becomes smaller than 30 km/h, ACC shall stop controlling the distance, release control over brake and accelerator back to the driver, and indicate this situation with a loud beep.
  - **Bad:** The software shall be prepared for a change of the measurement principle (e.g., infrared instead of radar)
  - **Good:** The necessary software adaptations for changing the principle of the sensor must be performable by one developer in one week.

# Requirements must understandable by the developers

---

- ▶ In embedded systems:

How much **domain knowledge** can be expected from the developers?

- ▶ Example:

**For the distance control, overshoot shall be below 5% and rise time below 3 seconds.**

- ▶ Trade-off between presumed domain knowledge of developer and size/complexity of requirements.
- ▶ A **dictionary** is always helpful.

# Concise requirements require sufficient domain knowledge of the developers

---

- ▶ A famous example from a famous requirements analysis book (D.C. Gause, G.M. Weinberg: Are your lights on? Adapted from Philip Koopman, CMU):
  - Shortly after a road tunnel there is a scenic-view overlook with a parking area.
  - Before the tunnel there is a sign asking the car drivers to switch on the lights
  - Result: Many drivers go through the tunnel, turn into the parking area, forget to switch off the lights, and have flat batteries when they return.
  - Therefore the highway department intends to erect a sign after the tunnel which **specifies the appropriate behavior** with respect to switching off the lights.



**Turn your lights off**

- ▶ But what about those who did not switch their lights on in the tunnel?



**If your headlights are on,  
turn them off**

- ▶ But what if it is night time?





**If it is daytime and  
your headlights are on,  
turn them off**

- ▶ But what if there is fog and visibility is reduced during daytime?



**If your headlights are on,  
and they are not required  
for visibility, turn them off**

- ▶ But what about modern (US) cars which are designed such that the headlights are on whenever the motor is running?



**If your headlights are on,  
and they are not required  
for visibility,**

**and you can turn them off,  
then turn them off**

**What would you do?**



**Are your lights on?**

## Lessons from this example?

# Summary: Basic requirements issues

---

Basic concepts and typical problems during requirements elicitation:

- ▶ **Solutions** instead of requirements
- ▶ Difference between **solutions** and **constraints**
- ▶ Difference between **functional** and **non-functional (quality)** requirements
- ▶ **Closed loop** vs. **Controller** requirements
- ▶ Assumed **environment (plant) behavior** and properties must be elicited
- ▶ Requirements must be **checkable**
- ▶ Requirements must be **understandable**

# Requirements for ACC /1

---

- ▶ ACC shall detect vehicles in front of the car in a distance from 0 to 120 meters and in an angle of  $\pm 4^\circ$  relative to the centerline of the car.
- ▶ ACC shall determine the velocity of detected vehicles
- ▶ If the road is free, ACC will keep the car at a specified cruise velocity.
- ▶ If a slower vehicle is in front of the car, ACC will reduce speed until a specified distance to the front vehicle is achieved. ACC will then keep the distance constant until the car either vanishes or accelerates above the specified cruise velocity.
- ▶ ACC can be activated between 30 and 180 km/h.

# Requirements for ACC /2

---

- ▶ In distance control mode, deviations from the reference distance must not exceed 5% for more than 2 seconds assuming a maximal acceleration or deceleration of the front vehicle of 0.5 g.
- ▶ The distance control algorithm shall be a PI controller.
- ▶ If ACC is malfunctioning, it has to switch off automatically and indicate by a red light and a short beep that it is out of service. It must never indicate a safe distance to the front vehicle when this is not the case.
- ▶ ACC must not be out of function for more than  $10e-7$  hours per year of operation time.
- ▶ Distance and cruise speed selections must be well readable for the driver.

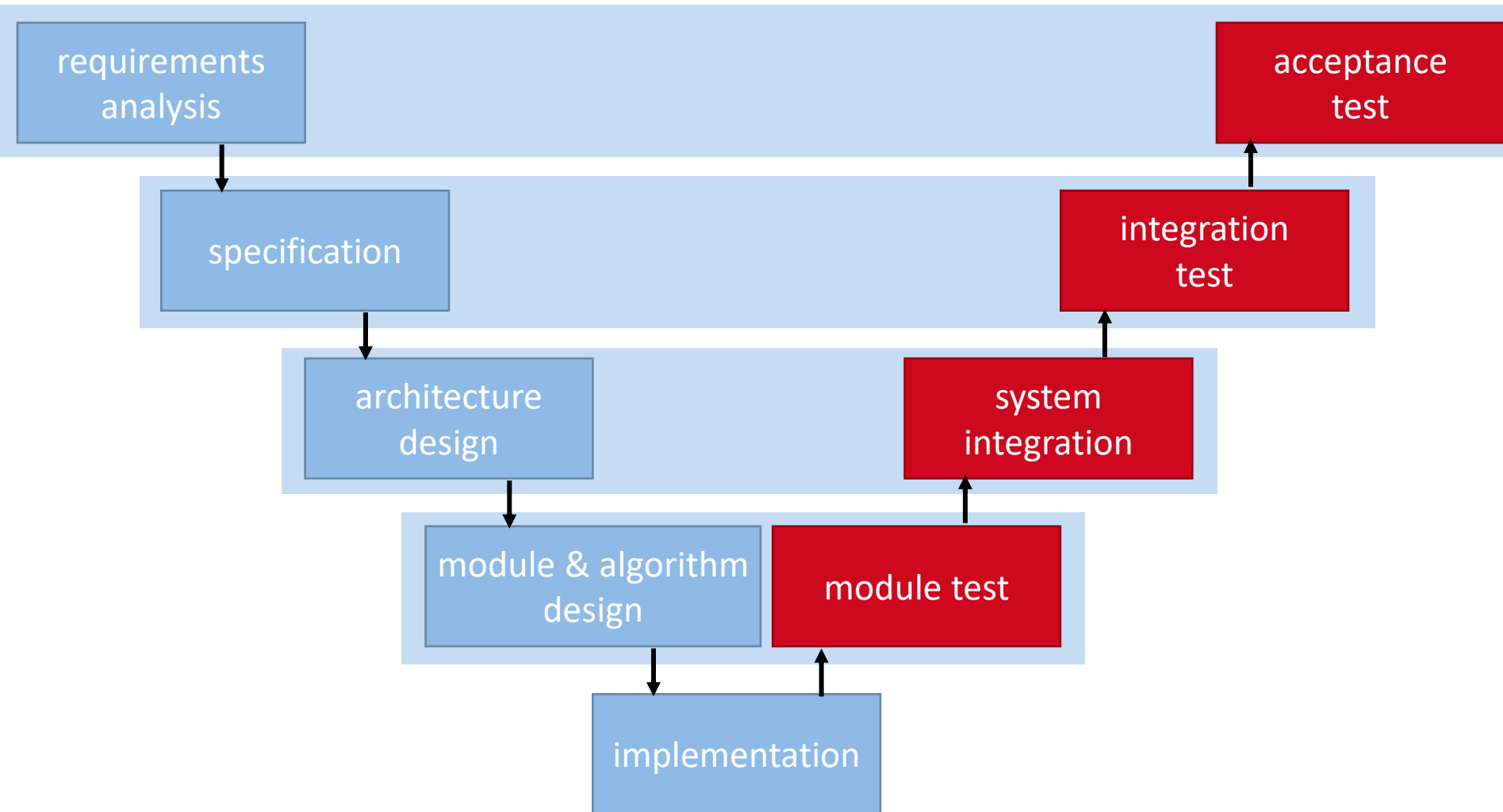
# Requirements for ACC /3

---

- ▶ Speed and direction angle information are provided by the ESP system via the CAN bus.
- ▶ For customer A, the ACC software has to run on a Motorola MPC4200. Customer B wants to integrate the ACC software into his already existing engine control unit.
- ▶ Manufacturer A wants a 77 Ghz radar sensor, manufacturer B an infrared sensor, manufacturer C wants to combine radar and video information.
- ▶ Marketing plans to offer ACC Stop&Go with an additional 24 GHz short range radar sensor for next year.



# Reminder: V model



# How to do requirements analysis

Reminder:

## ► Requirements **Elicitation**

- Collect the requirements from customers, marketing, system engineers etc.

## ► Requirements **Analysis**

- Analyse whether the requirements are actually what the customers, marketing, system engineers etc. want.

- “Online”
- Includes first ad-hoc analysis

- “Offline”
- Results will be presented to the customer after analysis
- Several iterations possible

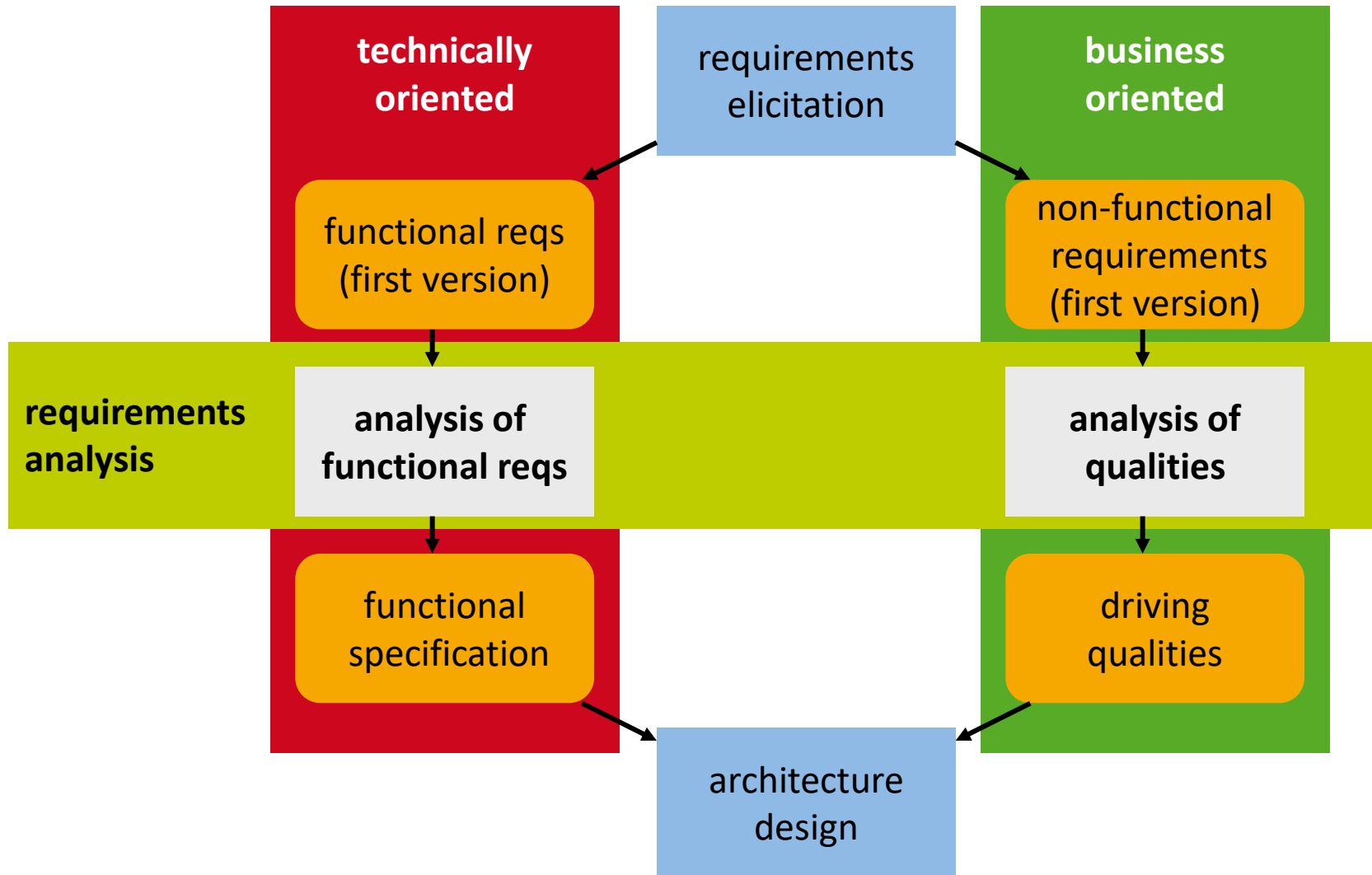
# Reminder: Basic requirements issues

---

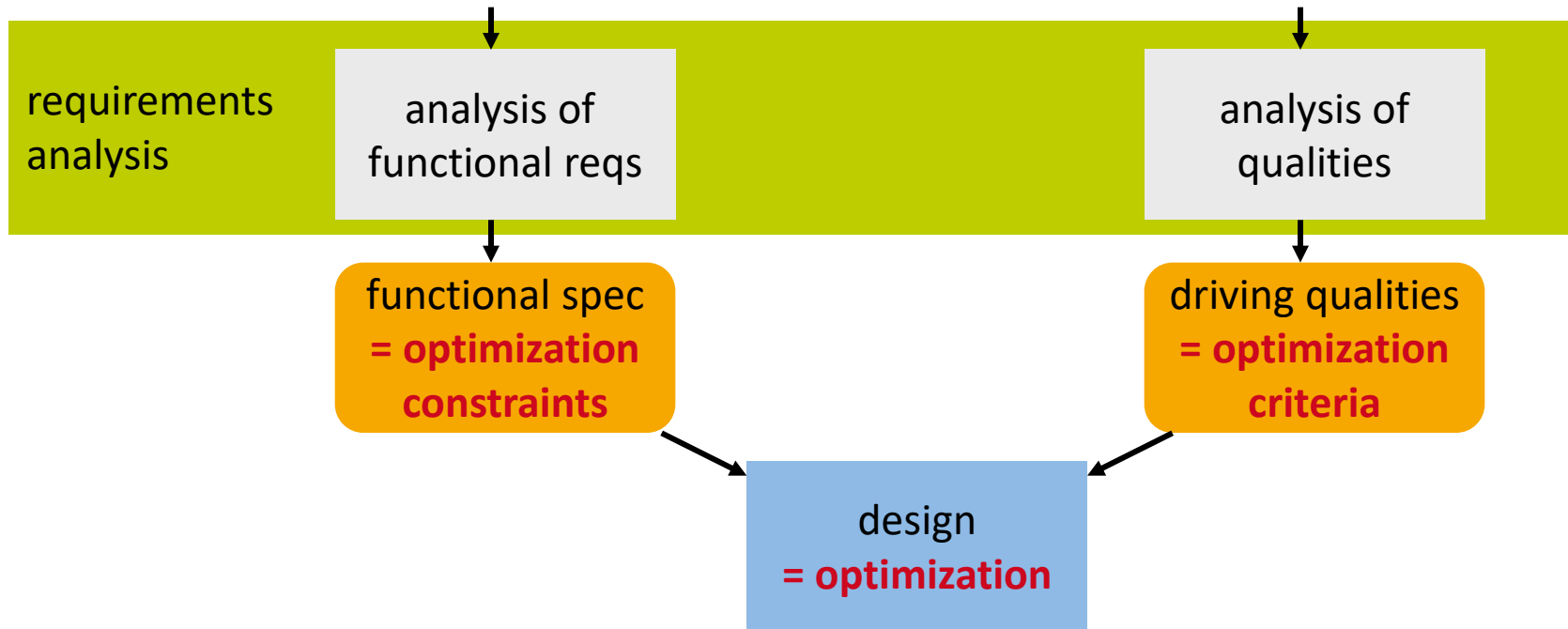
Basic concepts and typical problems during requirements elicitation:

- ▶ **Solutions** instead of requirements
- ▶ Difference between **solutions** and **constraints**
- ▶ Difference between **functional** and **non-functional (quality)** requirements
- ▶ **Closed loop** vs. **Controller** requirements
- ▶ Assumed **environment (plant) behavior** and properties must be elicited
- ▶ Requirements must be **checkable**
- ▶ Requirements must be **understandable**

# Two analysis paths



# Design viewed as an optimization problem



Find best or good enough solution (measured by qualities) among all admissible solutions (given by functional spec.)

# Part II

## Analysis of Functional Requirements



# Analysis of Functional Requirements

- ▶ Manifest your understanding of the elicited requirements (make them clear to you).
- ▶ Structure and present them in different form to the customer (e.g., build models or derive test cases), such that the customer can then compare your understanding with his/her original intentions.
- ▶ Do not begin designing the system!
- ▶ **Why?**
  - Requirements may change.
  - Customer will discuss design issues with you.
  - Design solution will become part of specification document and reduce degree of design freedom.
- ▶ **How?**
  - Regard system as black box (in this phase).

# Analysis of Functional Requirements – First step: Define System Boundaries

---

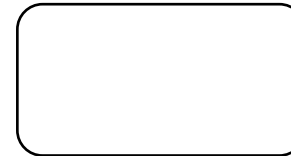
- ▶ Interfaces between system and environment
- ▶ What must be designed, what is given?
- Needed for agreement on content of the project
- One possible model: **Context diagram**



# Context diagram

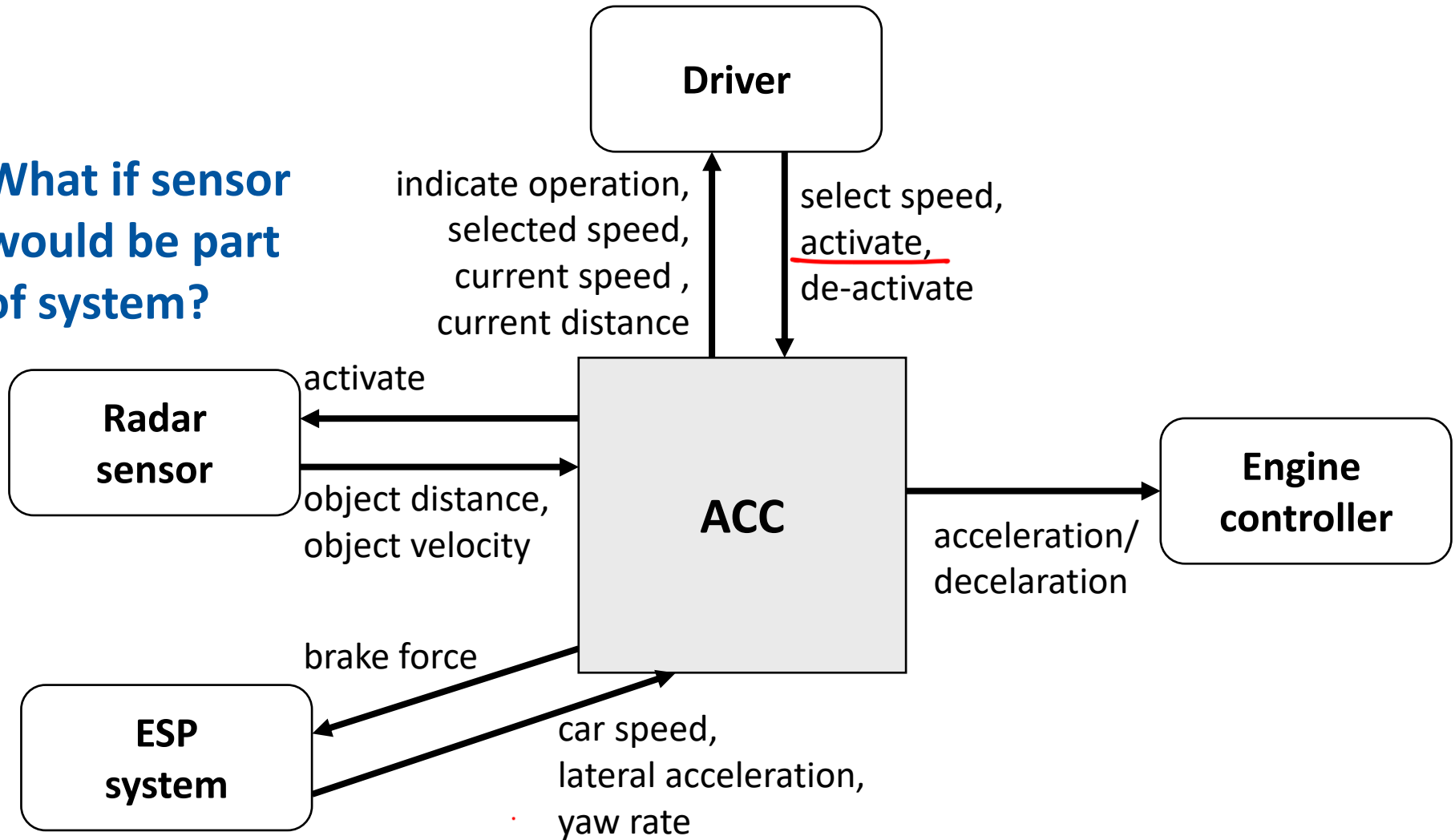
## Elements:

- ▶ System (Function)
- ▶ Partner in the environment (Interface)
- ▶ Flow/Exchange of
  - Data
  - Information
  - Energy
  - Mass
  - ...



# Context diagram: Example ACC

What if sensor  
would be part  
of system?



# Analysis of Functional Requirements –

## Second Step: Understand „Black Box“ behavior

---

- ▶ Capture functional requirements as they manifest themselves at the interface between system and environment.
- One possible tool: **Use Cases**
- ▶ **Use case = A coherent piece of functionality of the system that is visible (in black-box form) from outside the system.**
- ▶ Elements outside the system which are involved in the use case are called **actors**.
- ▶ In embedded systems, actors are mostly **sensors** and **actuators** (but also users).
- ▶ OOA Definition of use case (see OOSC, Prof. Lichter):  
A **sequence** of interactions between an actor/actors and a system **triggered** by a specific actor which produces a **result** for an actor.  
**(applicable to continuous control functions?)**

# Use cases: Example ACC

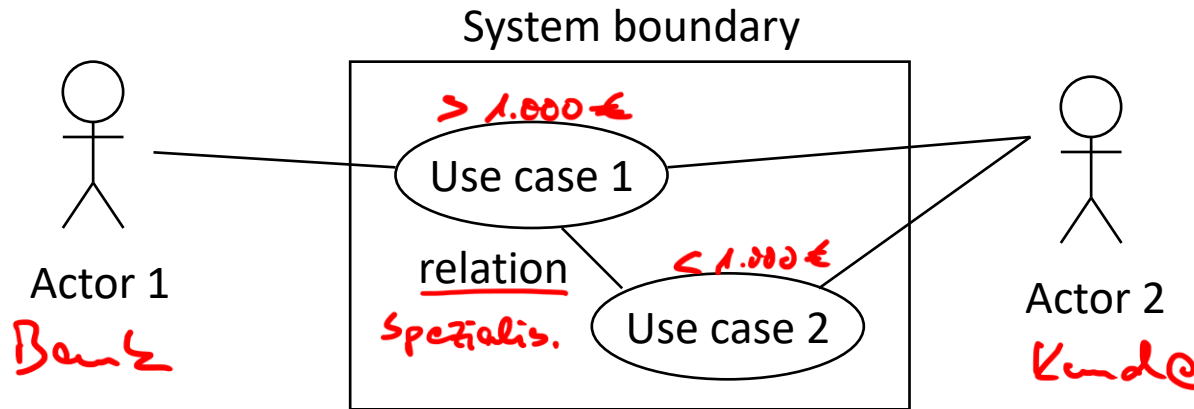
---

## Use cases:

- ▶ Activation
- ▶ De-activation
- ▶ Keeping speed constant
- ▶ Keeping distance to front object constant

# Use case relations/ use case diagrams

- ▶ In OOA, use cases can have relations:
  - includes
  - extends
  - generalizes (with inheritance of interaction relations)
- ▶ This offers possibility to **reuse** use cases.
- ▶ Aim: Structure functionality.
- ▶ **Use case diagrams:**



# Textual description of use cases

---

- ▶ Several templates in literature.
- ▶ From lecture OOSC by Prof. Lichter:
  - **Name**
  - **Aim**
  - **(Level)**
  - **Pre-condition**
  - **Post-condition**
  - **Post-condition in case of failures**
  - **Actors**
  - **Trigger**
  - **Standard sequence (steps, actions/events)**
  - **Alternatives/exceptions**

# Textual description of use cases:

## Example ACC activation

---

- ▶ **Name:** Activating ACC
- ▶ **Aim:** ACC must be operational
- ▶ **Pre-condition:** ACC is not activated; car is running; radar sensor is available; ESP, engine ECU are running; CAN is available; (cruise speed is selected); current speed is in [30 ; 180 km/h]
- ▶ **Post-condition:** ACC operation is indicated to driver
- ▶ **Post-condition in case of failures:** failure to activate ACC is indicated to driver
- ▶ **Actors:** driver, radar sensor, ESP
- ▶ **Trigger:** driver operates ACC activation interface

# Textual description of use cases:

## Example ACC activation /2

---

### ► Standard sequence (steps, actions/events):

- 1 Driver operates ACC activation interface
- 2 ACC activates radar sensor
- 3 radar sensor acknowledges availability
- 4 ACC checks that current speed is in [30 ; 180 km/h]
- 5 ACC indicates operability

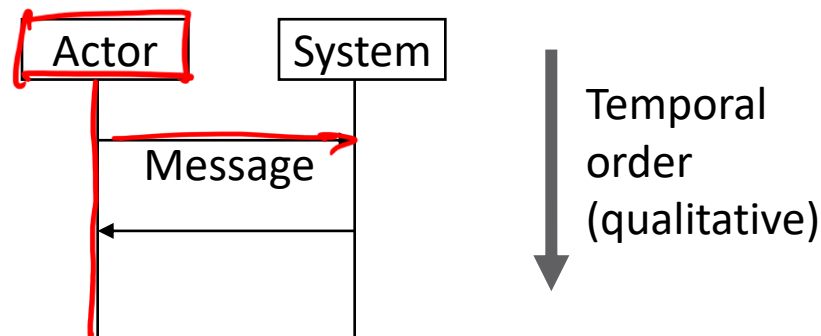
### ► Alternatives/exceptions:

- 3a1 no response from radar sensor
- 3a2 ACC indicates failure
- 4a1 current speed is not in [30 ; 180 km/h]
- 4a2 ACC de-activates radar
- 4a3 ACC indicates failure



# Sequence of actions/events in use cases can be represented graphically by Sequence diagrams/MSCs

- ▶ Basic idea:  
Represent interaction between system and actors by a temporally ordered exchange of messages
- ▶ Fundamental elements:
  - System, actors: vertical lines
  - Message exchange: horizontal arrows



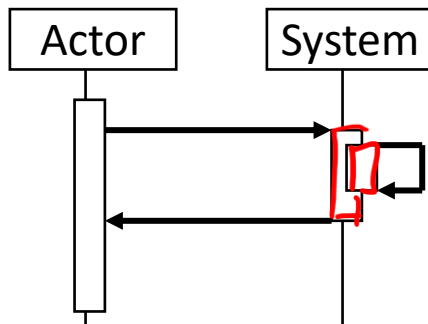
- ▶ Sequences are only possible, not a complete behavioral specification!

# Graphical representation of use cases:

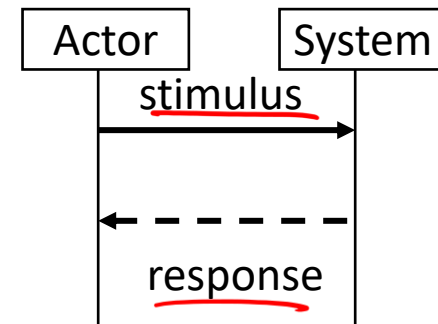
## Sequence diagrams/MSCs /2

- ▶ Origin:
  - OSI Time Sequence Diagrams (standardized 1991!)
  - ITU-T SG 10: Message Sequence Charts, 2000
  - ITU-T SG 10: Algebraic semantics of MSCs, 1996
- ▶ UML adopted MSCs, renamed them to **sequence diagrams** and changed/added some elements/representations
  - Examples:

Lifelines for objects

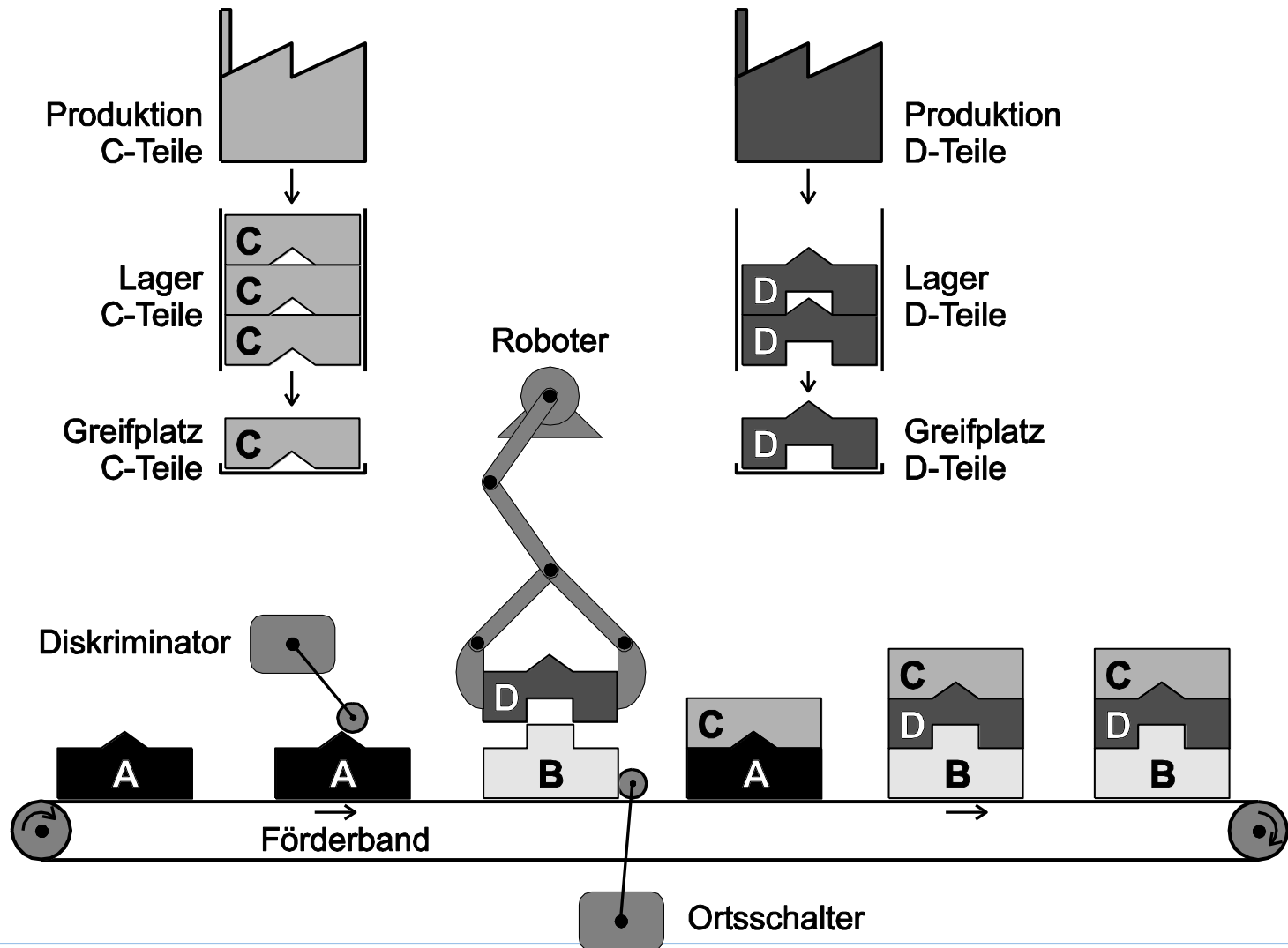


stimulus and response



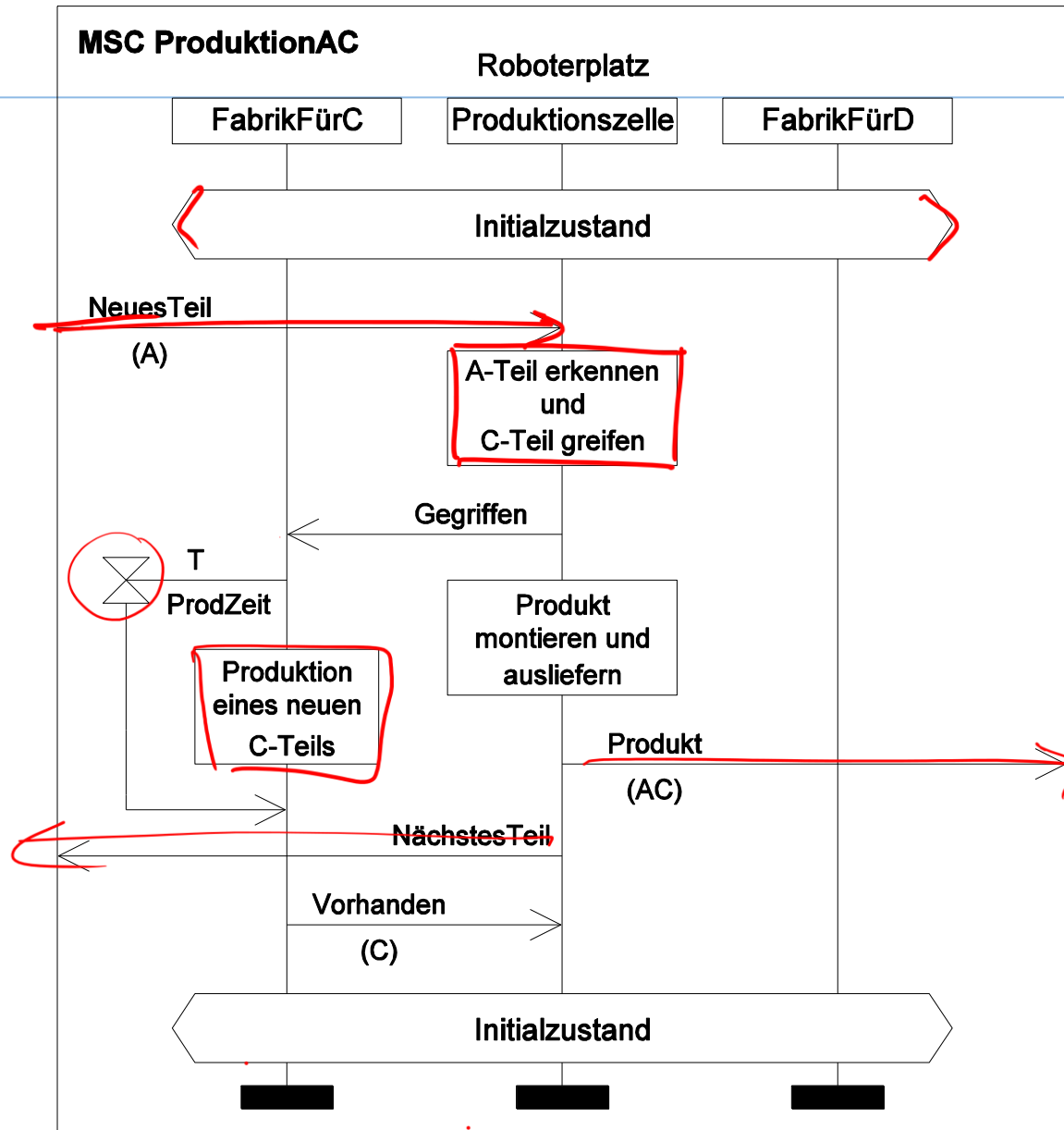
# MSC Example: production plant

(from Kowalewski, at-Automatisierungstechnik, 9/2001)



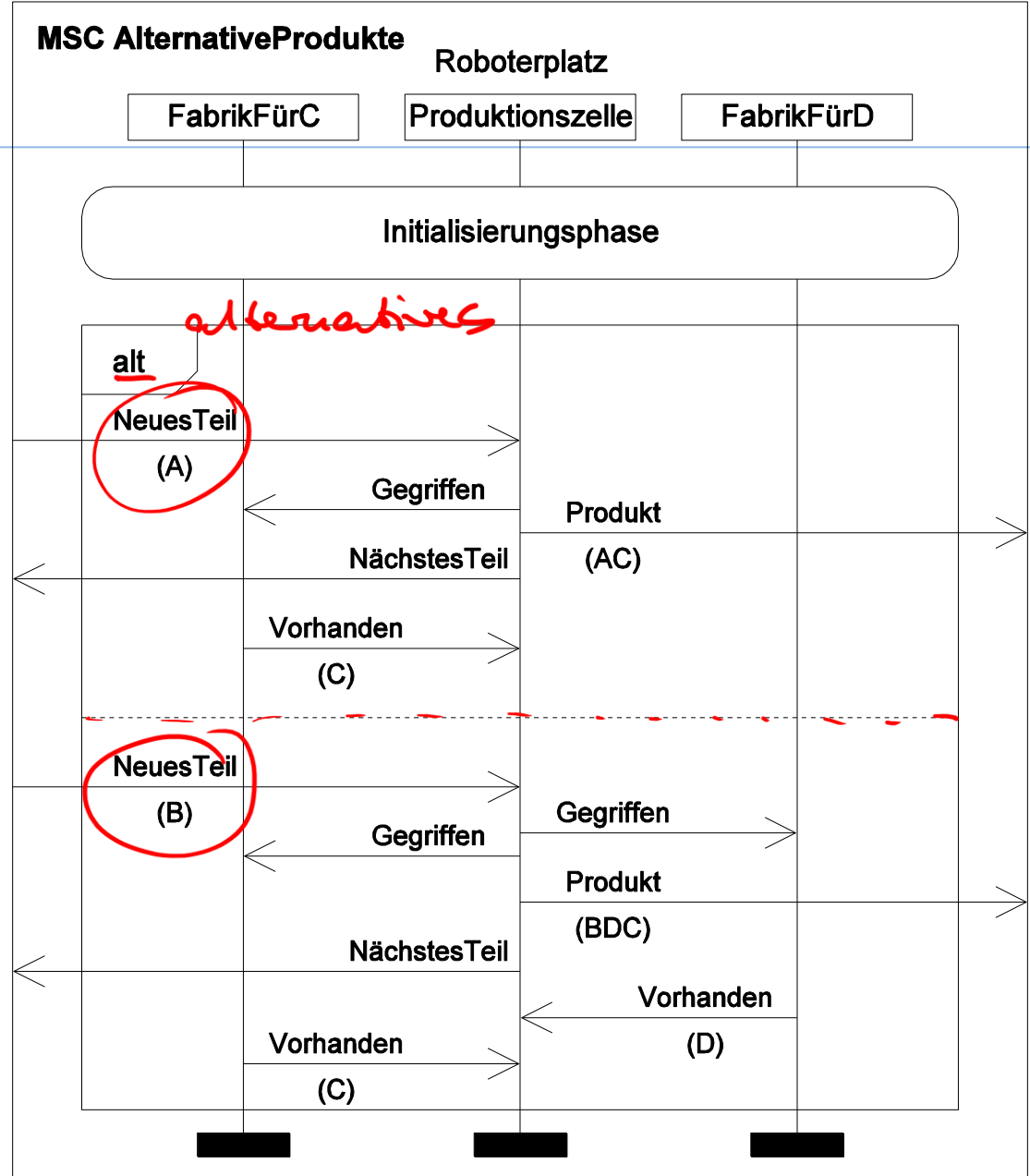
# MSC Example /2: basic MSC

(from Grabowski et al,  
at-Automatisierungstechnik, 12/2001)



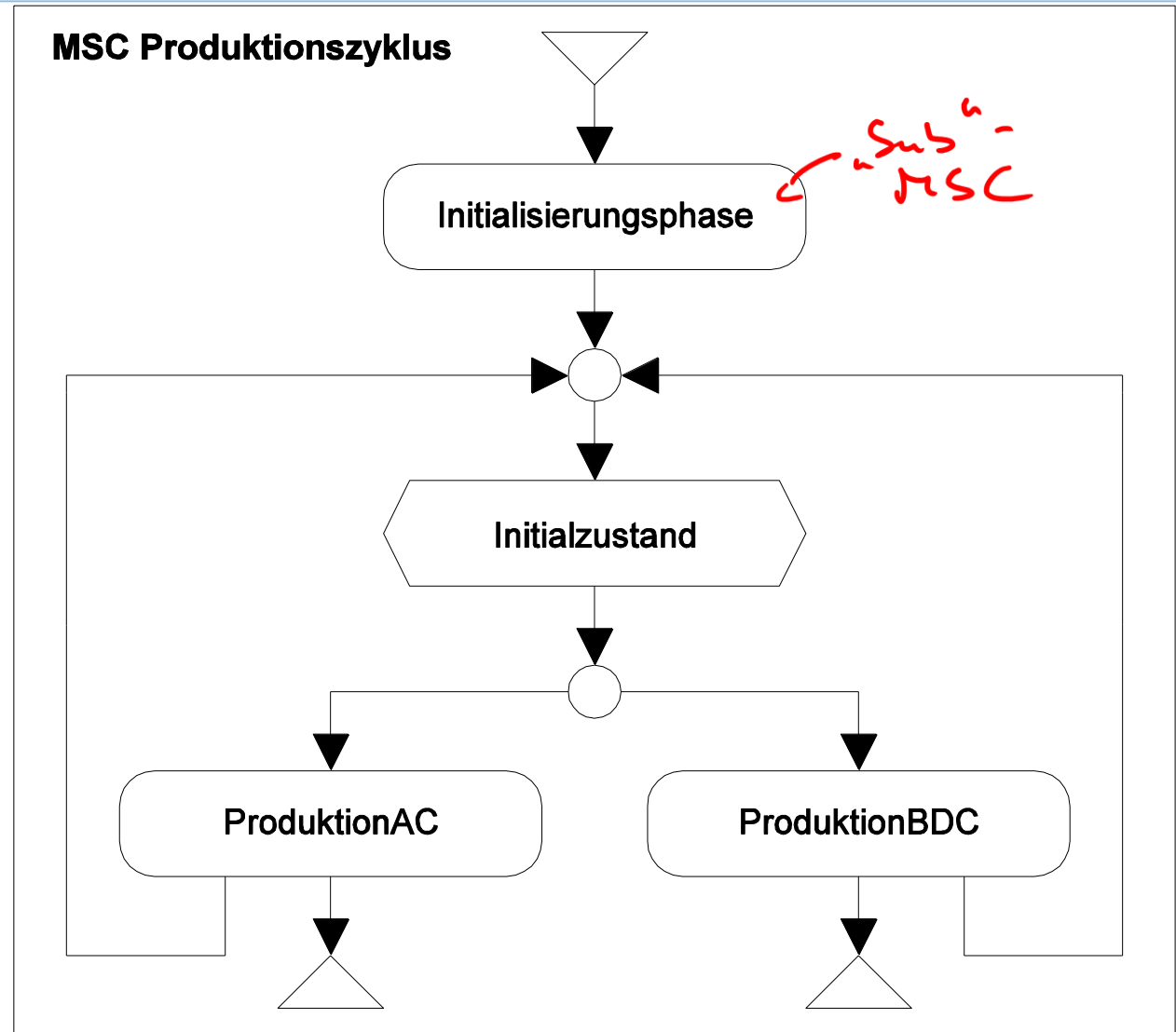
# MSC Example /3: inline expressions

(from Grabowski et al, at-  
Automatisierungstechnik, 12/2001)



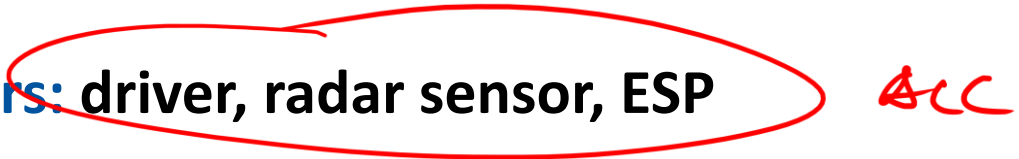
# MSC Example /4: High level MSCs

(from Grabowski et al, at-Automatisierungstechnik, 12/2001)



# Example for sequence diagram:

## Use case ACC activation

- ▶ **Name:** Activating ACC
- ▶ **Aim:** ACC must be operational
- ▶ **Pre-condition:** ACC is not activated; car is running; radar sensor is available; ESP, engine ECU are running; CAN is available; cruise speed is selected; current speed is in [30 ; 180 km/h]
- ▶ **Post-condition:** ACC operation is indicated to driver
- ▶ **Post-condition in case of failures:** failure to activate ACC is indicated to driver
- ▶ **Actors:** driver, radar sensor, ESP 
- ▶ **Trigger:** driver operates ACC activation interface

# Example for sequence diagram:

## Use case ACC activation (2)

---

### ► Standard sequence (steps, actions/events):

- 1 Driver operates ACC activation interface
- 2 ACC activates radar sensor
- 3 radar sensor acknowledges availability
- 4 ACC checks that current speed is in [30 ; 180 km/h]
- 5 ACC indicates operability

### ► Alternatives/exceptions:

3a1 no response from radar sensor

3a2 ACC indicates failure

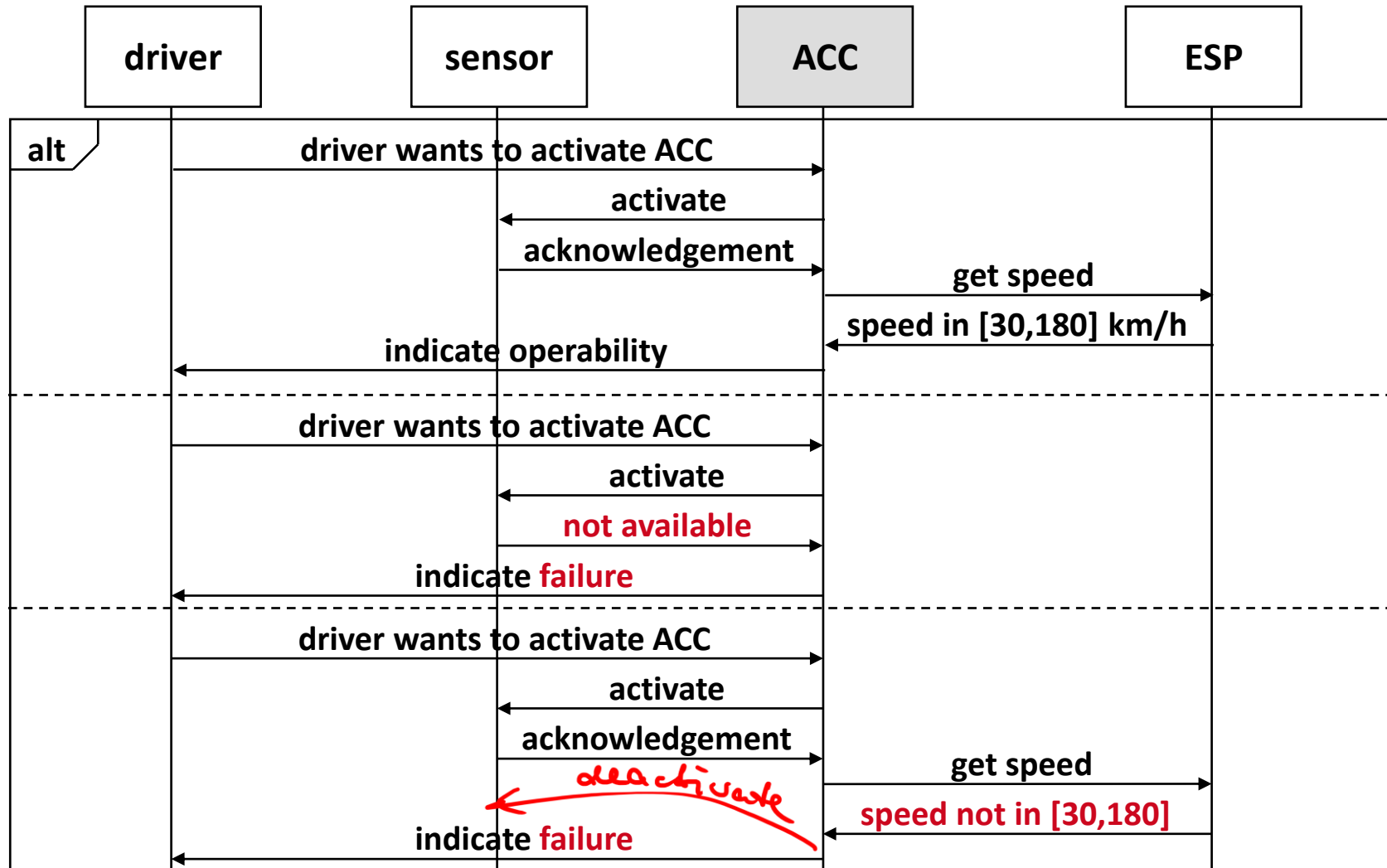
4a1 current speed is not in [30 ; 180 km/h]

4a2 ACC de-activates radar

4a3 ACC indicates failure



# MSC for use case „ACC activation“



# What about control loop reqs?

---

## Use cases:

- ▶ Activation
- ▶ De-activation
- ▶ Keeping speed constant
- ▶ **Keeping distance to front object constant**

# First try:

## Use case for Keeping distance to front object constant

---

### ▶ Name:

- Keeping distance to front object constant

### ▶ Aim:

- Keeping distance constant without driver interaction

### ▶ Pre-condition:

- ACC is active

### ▶ Post-condition:

- current distance is displayed, distance is as specified, in speed control if necessary

### ▶ Post-condition in case of failures:

- indication of failure, de-activate ACC

### ▶ Actors:

- engine CU, ESP, driver, sensor

### ▶ Trigger:

- detection of a front object

## First try:

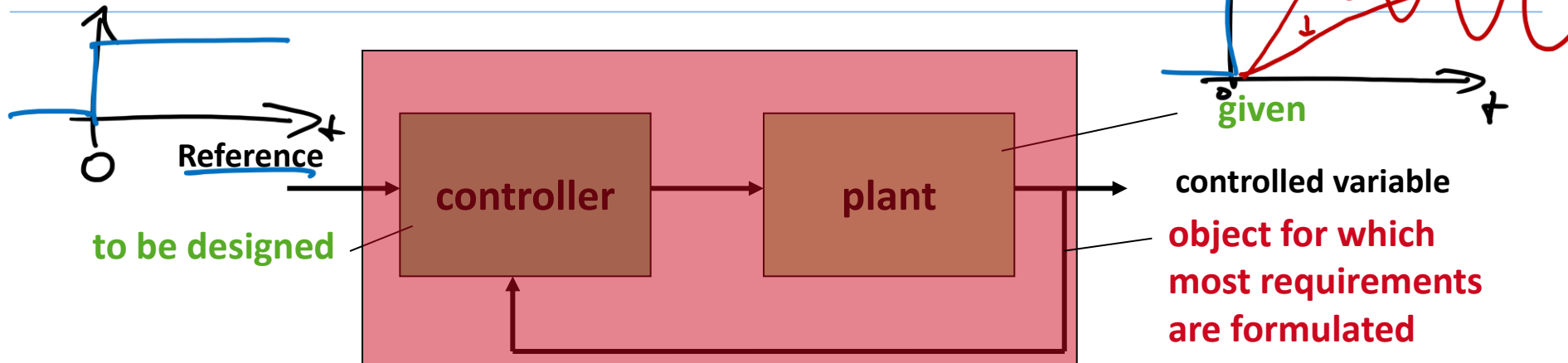
### Use case for Keeping distance to front object constant

---

- ▶ **Standard sequence (steps, actions/events):**
  - 1 **Get distance and object speed**
  - 2 **Get own speed, (lateral acceleration, yaw rate)**
  - 3 **Compare current with desired distance**
  - 4 **Adjust speed accordingly**
  - 5 **Display distance**
  - 6 **Go back to 1 as long as object is in front**

**This is not how control loop reqs are formulated!**

# Excursus: Basic control engineering



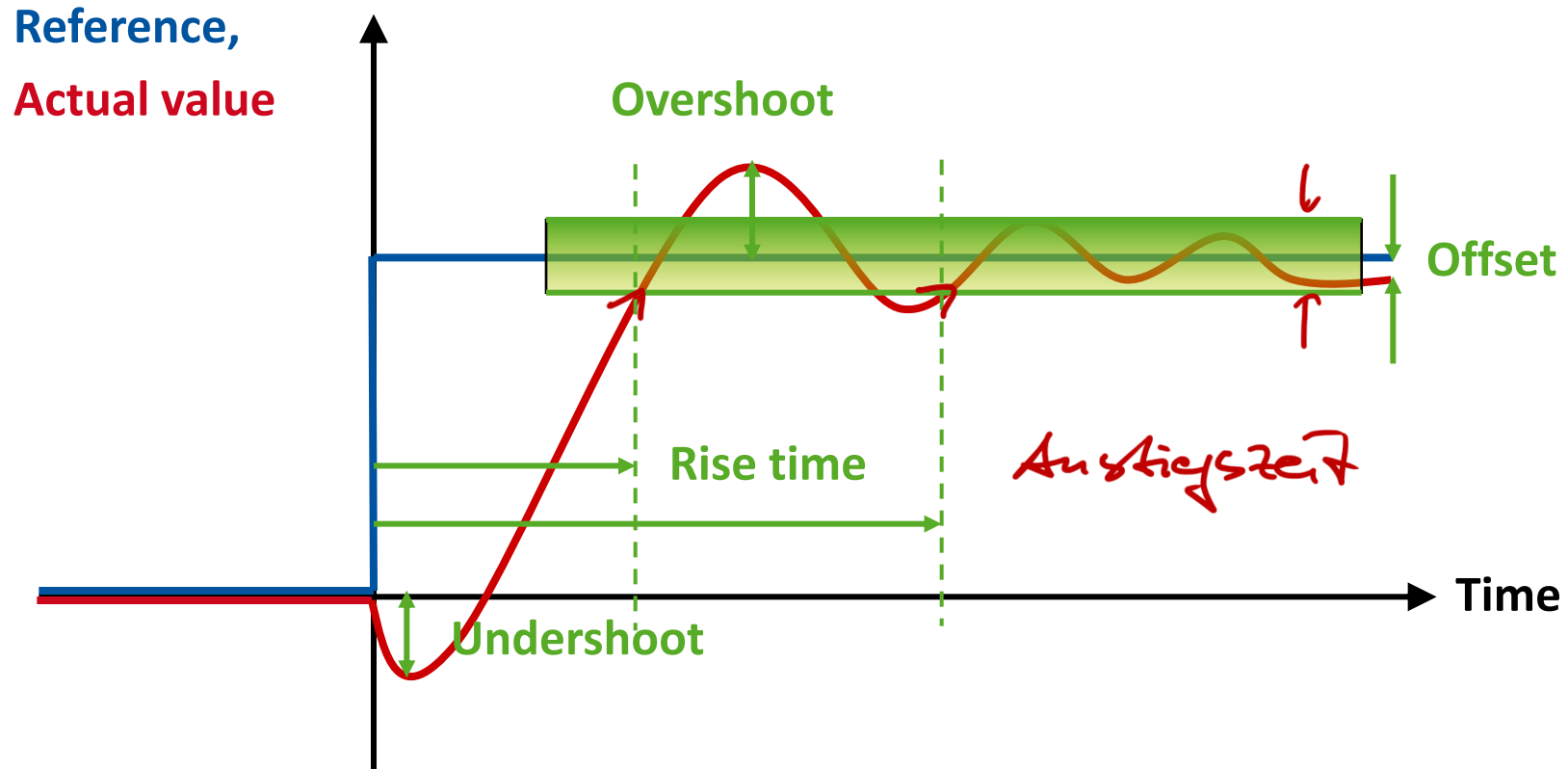
- ▶ Main requirements formulated for controlled variable as output of closed loop system:
  - stability
  - no permanent oscillation
  - quick and close following of reference signal
- ▶ Not always trivial to achieve: dynamic behavior of plant must be considered

# Closed-loop requirements

---

- ▶ Stability
- ▶ Speed of compensation
- ▶ Transient errors
- ▶ Stationary accuracy
- ▶ Remaining oscillations
- ▶ → Step response

# Requirements specification for control loops by means of the step response



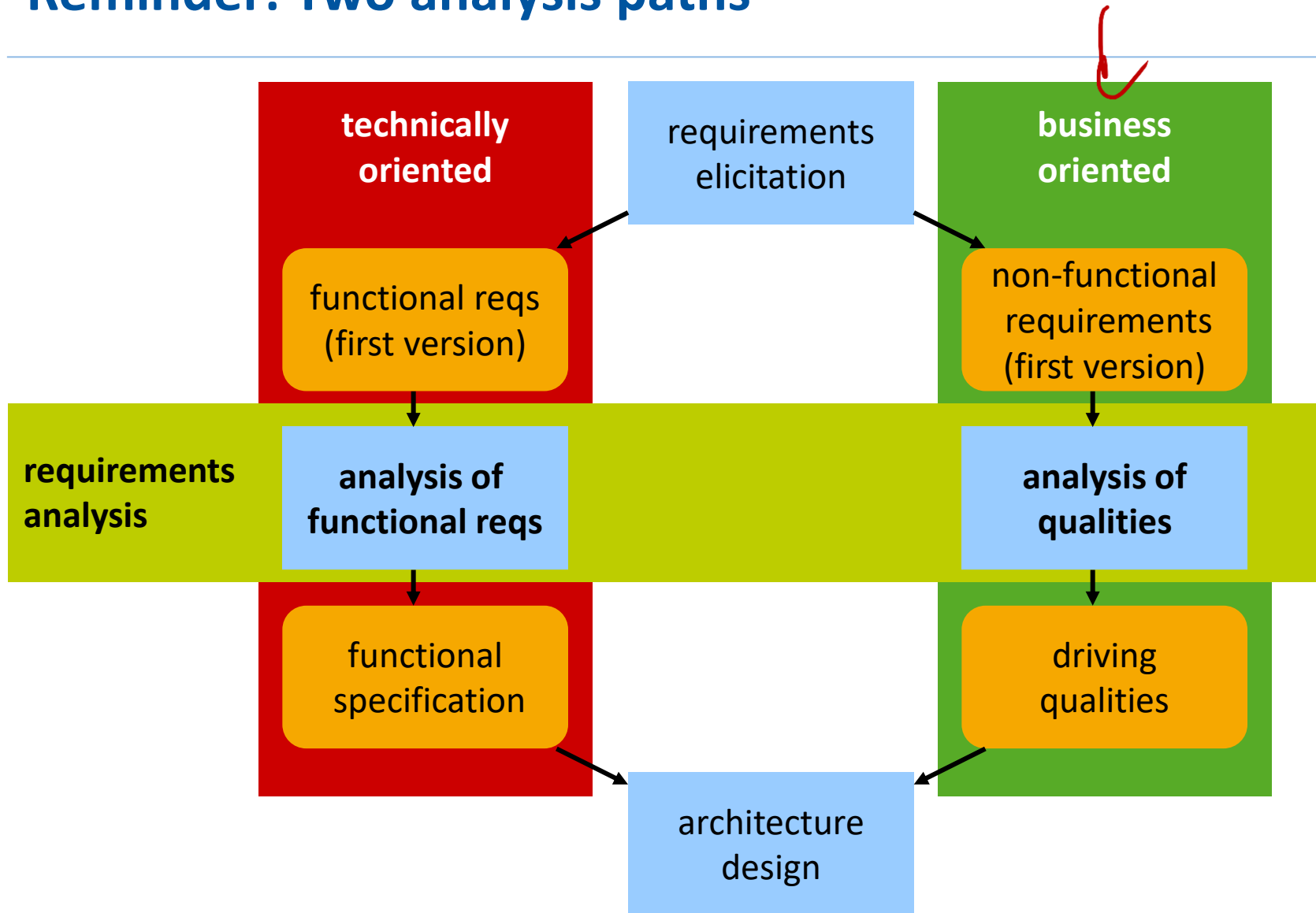


# Part III

## Analysis of Quality Requirements



# Reminder: Two analysis paths



## Thought experiment:

- ▶ Two companies A and B want to develop and market an ACC system with exactly the same functionality

# Company A

---

- ▶ We are the biggest supplier of driver assistance systems in Europe with a long standing reputation for quality products. Under no circumstances can we risk to loose this reputation by frequent failures or even recalls.
- ▶ To achieve reliability we developed an elaborated diagnosis and fault tolerance concept involving watch dogs and graceful degradation.
- ▶ If faults will appear in the field, we will take care to remove them quickly during the regular inspections at the dealer.
- ▶ Our strategy is to offer an ACC product line for the medium and high end market which will stay in market at least for the next five years. This includes compatibility with new sensor technologies.
- ▶ In this market segment, cars usually have plenty of processor capabilities. To reduce costs for customers, we will offer ACC functionality as a pure software product, too.

# Company B

---

- ▶ We are new in the driver assistance systems business. Our only chance to get into the market is to be there first and with competitive prices.
- ▶ We want to keep our prices low by three main strategies:
  1. High volumes for cheap hardware.
  2. A “one-size-fits-all” approach with a standard hardware configuration and a standard radar sensor. Adaptations should be restricted to the interfaces to other ECUs and the HMI.
  3. Off-shore development of non-critical parts of the software.
- ▶ Since we believe to be first on the market, it is important to protect our ACC algorithm from being re-engineered by competitors.
- ▶ Of course, our system has to function properly. A recall would mean the end of our young company. We will therefore test extensively.

# Legal constraints

---

- ▶ Safety regulations and standards require that the operation of driver assistance systems must not increase the risk of damage to life and health of the car passengers and other people.
- ▶ All electronic systems must store failures during operation in a special memory which must be accessible by a standard testing device.

# Quality analysis

## ▶ Aim:

- Understand the meaning and importance of the quality requirements.
- Prepare for checking that quality requirements will be achieved.

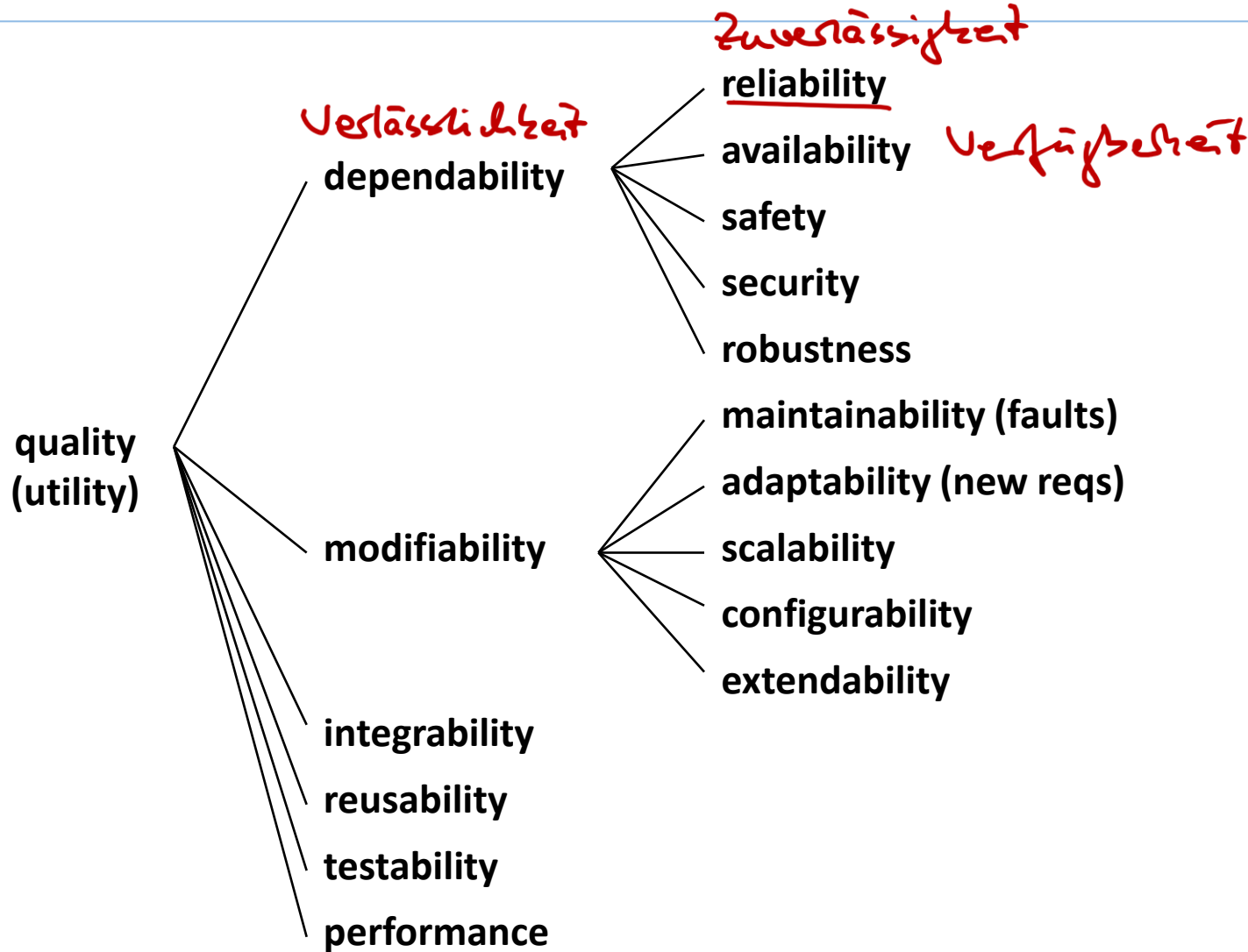
## ▶ Tools: Quality trees and scenarios

## ▶ Literature:

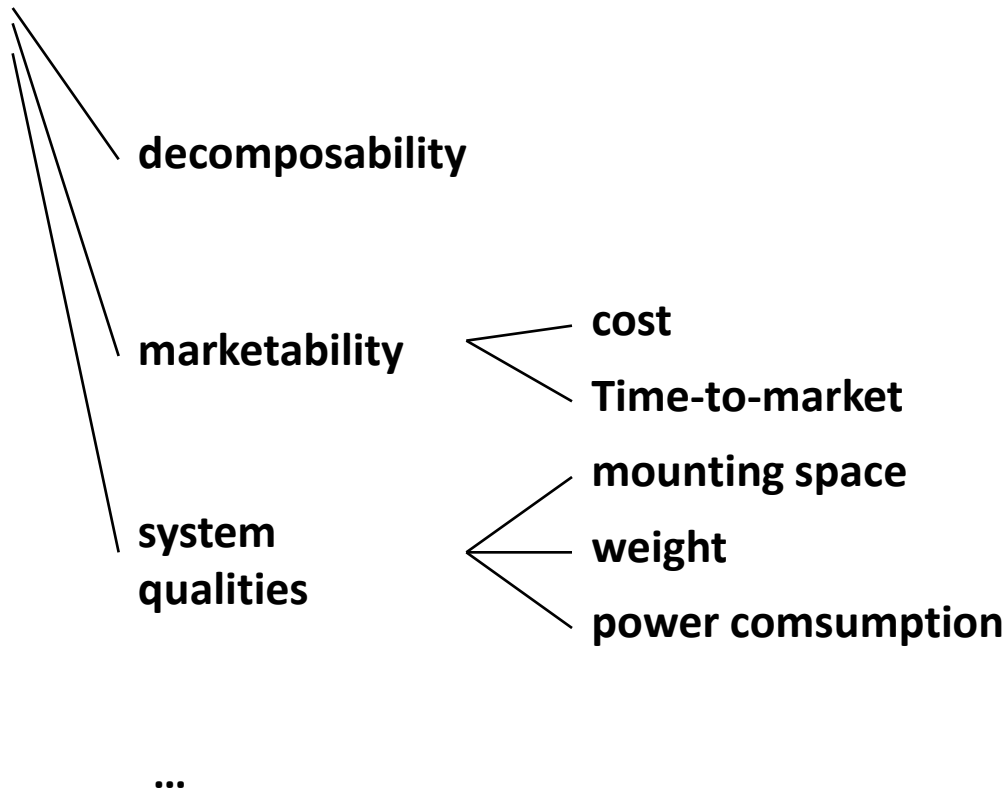
- Len Bass, Paul Clements, Rick Kazman:  
Software Architecture in Practice, 2nd Ed.  
Addison-Wesley, 2003

Software Eng. Inst.  
(SEI), CMU. Pittsburgh

# Quality tree (general)



# Quality tree (general) /ctd.





# Quality analysis by a quality tree

---

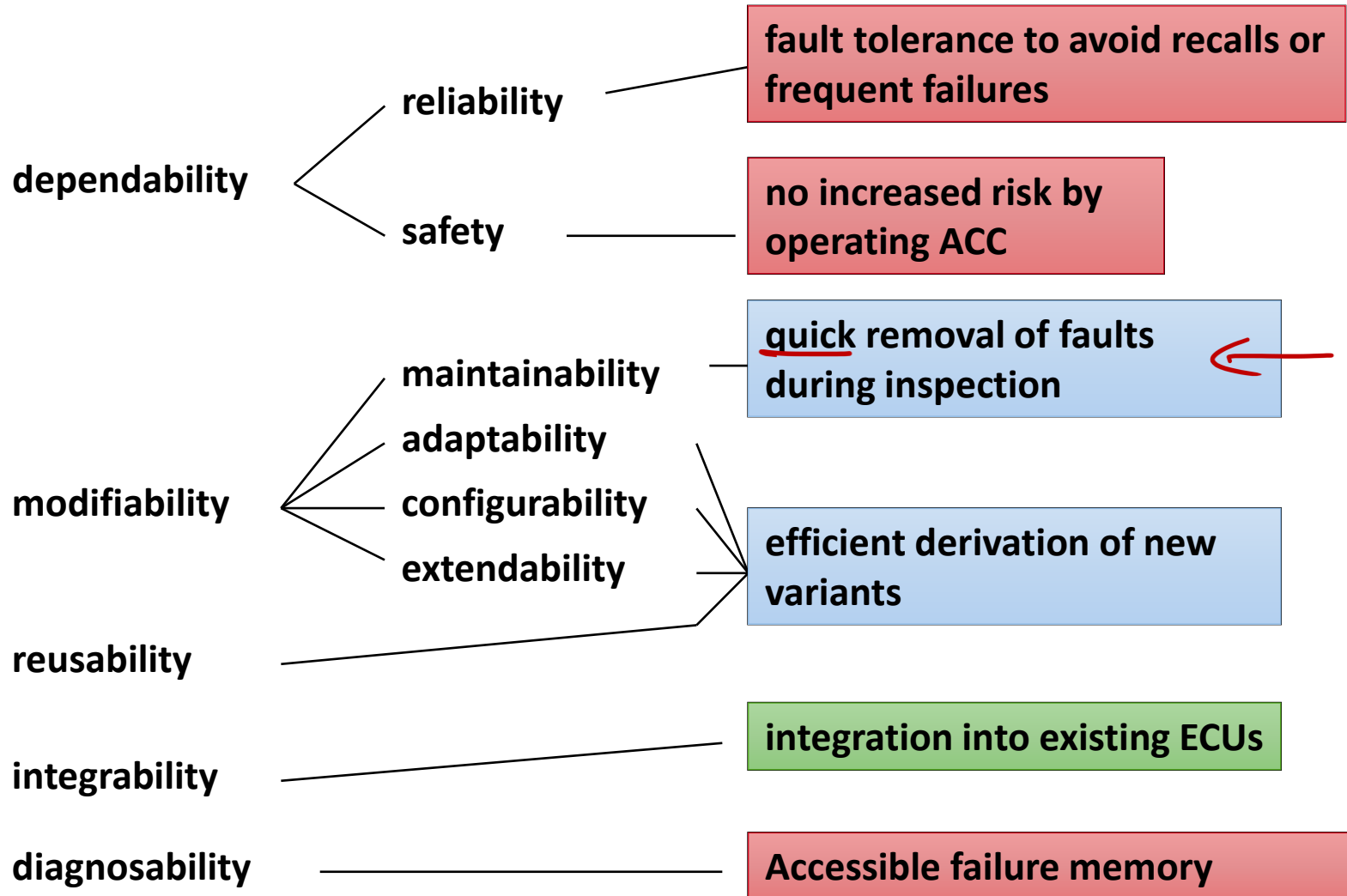
- ▶ List only relevant qualities
- ▶ Refine qualities
- ▶ Classify low level qualities according to their importance (e.g. \*\*\* crucial, \*\* important, \* nice-to-have)
- ▶ (Later: Assign a scenario to each low level quality)

# Company A

---

- ▶ We are the biggest supplier of driver assistance systems in Europe with a long standing reputation for quality products. Under no circumstances can we risk to loose this reputation by frequent failures or even recalls.
- ▶ To achieve reliability we developed an elaborated diagnosis and fault tolerance concept involving watch dogs and graceful degradation.
- ▶ If faults will appear in the field, we will take care to remove them quickly during the regular inspections at the dealer.
- ▶ Our strategy is to offer an ACC product line for the medium and high end market which will stay in market at least for the next five years. This includes compatibility with new sensor technologies.
- ▶ In this market segment, cars usually have plenty of processor capabilities. To reduce costs for customers, we will offer ACC functionality as a pure software product, too.

# Reminder: Quality tree for company A



# Quality trees are not sufficient

---

- ▶ Quality trees help to get a good overview of the quality requirements
  
- ▶ Open questions:
  - How do we find the most relevant qualities?
  - How can we check later that quality requirements are met (e.g. by the architecture)?

→ Further refinement needed.

→ **Scenarios.**

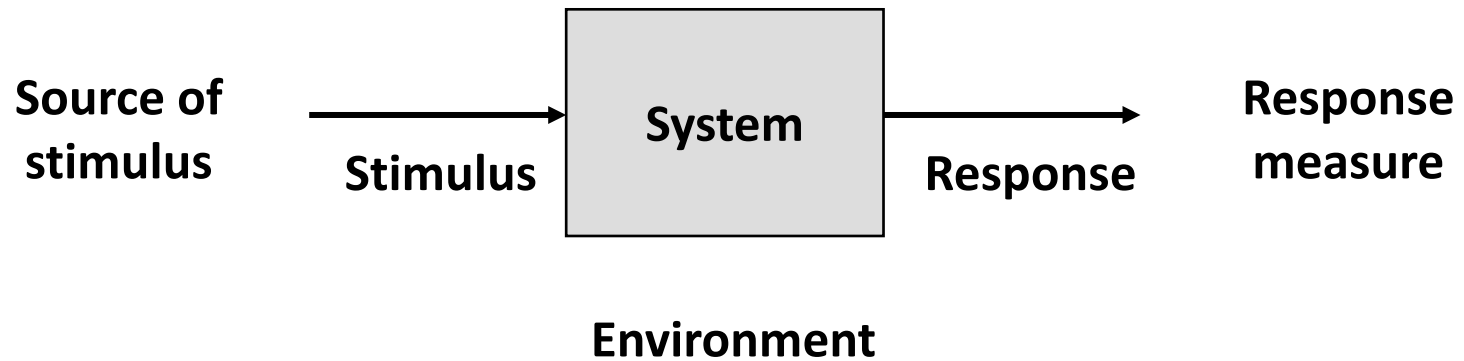
# SEI Definition of “Scenario”

---

- ▶ "Short stories" that describe a system interaction with respect to some quality attribute. Typically, the focus would be on three kinds of scenarios:
  - *use case*: anticipated uses of the system,
  - *growth*: anticipated changes to the system, and
  - *exploratory*: unanticipated stresses to the system (uses and/or changes).

# Quality attribute scenarios (Bass, Clements, Kazman)

---



# Example scenario: maintainability for company A

---

Source of stimulus

External, user or customer

Stimulus

User or customer reports fault

Environment

- System is in use, fault occurred during run-time
- all running systems are only accessible at regular inspections
- fault is not safety-critical

Response

Fault will be removed

Response measure

Fault is removed in an arbitrary garage within at most one hour and for <100€ per car.

SPIEGEL ONLINE - 12. Mai 2004, 10:18

URL: <http://www.spiegel.de/auto/werkstatt/0,1518,299464,00.html>

## Bremsversagen

# DaimlerChrysler ruft 680.000 Mercedes SL und E zurück

**Ein Elektronikproblem bei der E- und SL-Klasse macht Mercedes zu schaffen: Die SBC-Bremse kann sich plötzlich abschalten. Besonders gefährdet sind Fahrzeuge mit hoher Laufleistung in Verbindung mit sehr häufiger Bremsbetätigung wie Taxis.**

Stuttgart/Auburn - Die Probleme mit der SBC-Bremse (Sensotronic Brake Control) führen zu einem Auto-Rückruf in großem Maßstab: Weltweit ruft DaimlerChrysler rund 680.000 Fahrzeuge der Mercedes-Benz-Klassen E- und SL in die Werkstätten. In Deutschland seien rund 225.000 Fahrzeuge betroffen, sagte ein Unternehmenssprecher am Dienstagabend auf Anfrage in Stuttgart.

Zurückgerufen werden E-Klasse-Limousinen ab dem Baujahr März 2002 und T-Modelle ab dem Baujahr März 2003 sowie die SL-Klasse ab dem Baujahr Oktober 2001. Der Zeitaufwand beträgt eine Stunde. Über Kosten macht Mercedes keine Angaben. Experten gehen aber von rund 50 Millionen Euro aus.



T-Modell der Mercedes-E-Klasse: Die beanstandeten Fahrzeuge waren hauptsächlich Taxis



## A recent example /2

---

„Bei den Fahrzeugen kann sich die elektronische SBC-Bremskontrolle auf Grund "unvorhergesehener Signale" abschalten. Mercedes zufolge steuert ein Mikrocomputer das System, während Sensoren das Verhalten von Fahrzeug und Fahrer überwachen. Dadurch sollen sich unter anderem die Bremswege verkürzen. Bei Ausfall des SBC-Systems sind Verzögerungen nur noch über die hydraulische Reserve möglich. Sie erfordert einen höheren Kraftaufwand, und der Pedalweg wird länger.

Die Zahl der Autos, deren Bremsanlage möglicherweise zu beanstanden seien, belaufe sich nur auf ungefähr zwei von tausend Fahrzeugen, sagte ein Sprecher. Dabei handele es sich hauptsächlich um Fahrzeuge mit hoher Laufleistung in Verbindung mit sehr häufiger Bremsbetätigung. Dies treffe beispielsweise auf Taxis zu.“

## A recent example /2 (English translation)

---

In the vehicles, the Sensotronic Brake Control (SBC) may switch off because of “unpredicted signals”. According to Mercedes, a microcomputer is controlling the system while sensors supervise the behavior of vehicle and driver. This shall lead, among others, to a reduction of the effective braking distance. In the case of a SBC failure, decelerations are only possible using the hydraulic fall-back path. This one requires extended force, and the pedal travel will be longer.

The number of cars with potentially objectionable braking systems is approximately only 2 of 1000 vehicles, a spokesman said. These are mainly vehicles with high mileage in connection with very frequent usage of the brakes, e.g. taxis.

# A recent example - discussion

---

- ▶ What went wrong?
- ▶ In which development phase was the fault (probably) made?
- ▶ Which qualities are impaired?
  - Reliability/Availability
  - Safety
  - Usability
- ▶ Which quality does the system obviously have?
  - Estimated cost of recall: 50 million €
  - Number of cars: 680 000
  - ⇒ **Cost per car: € 73,53 !!**
  - ⇒ **Maintainability must be very good** (or the cost estimation is bad)
  - ⇒ Also: most probably a software fault

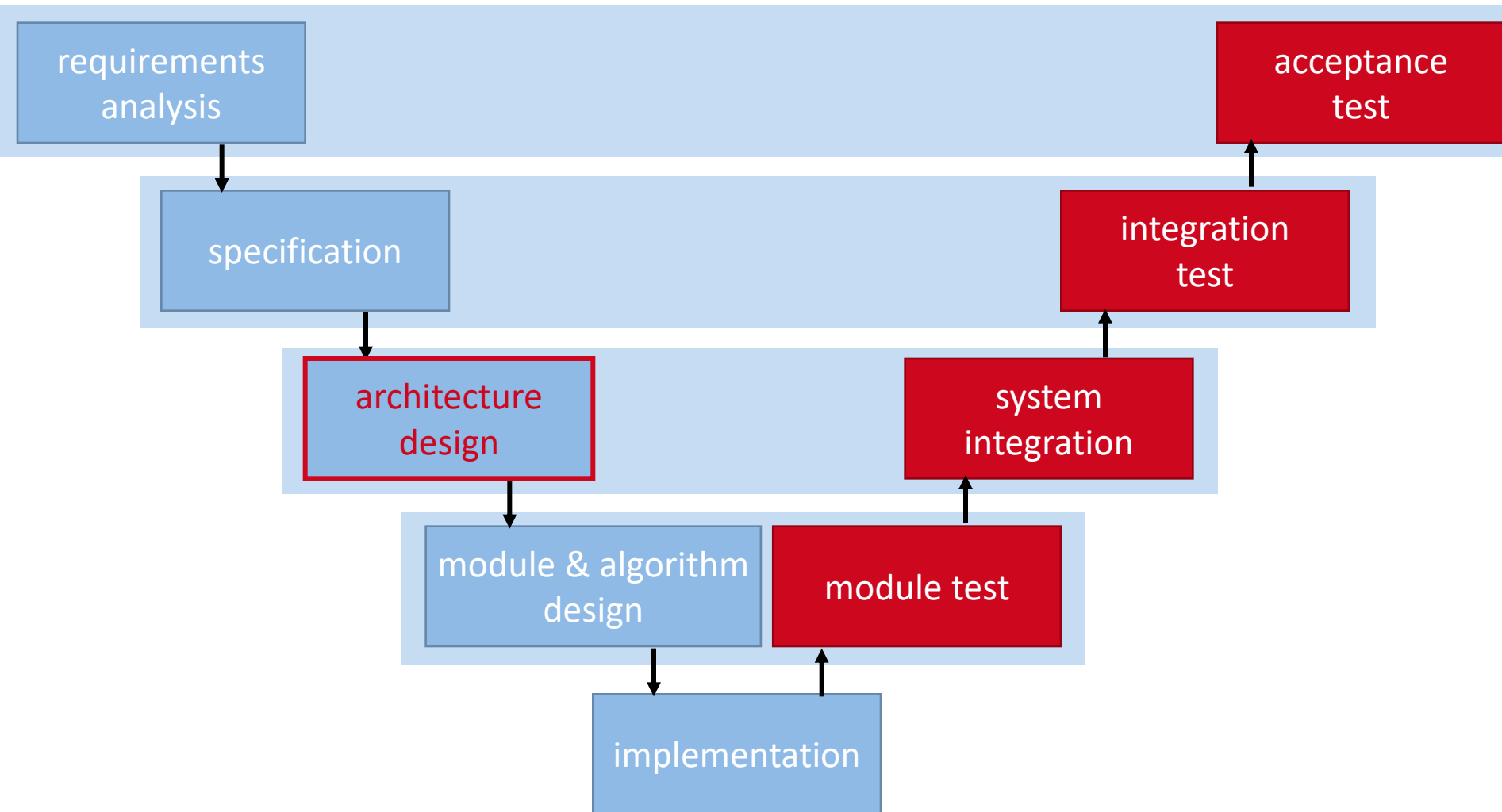


# Part IV

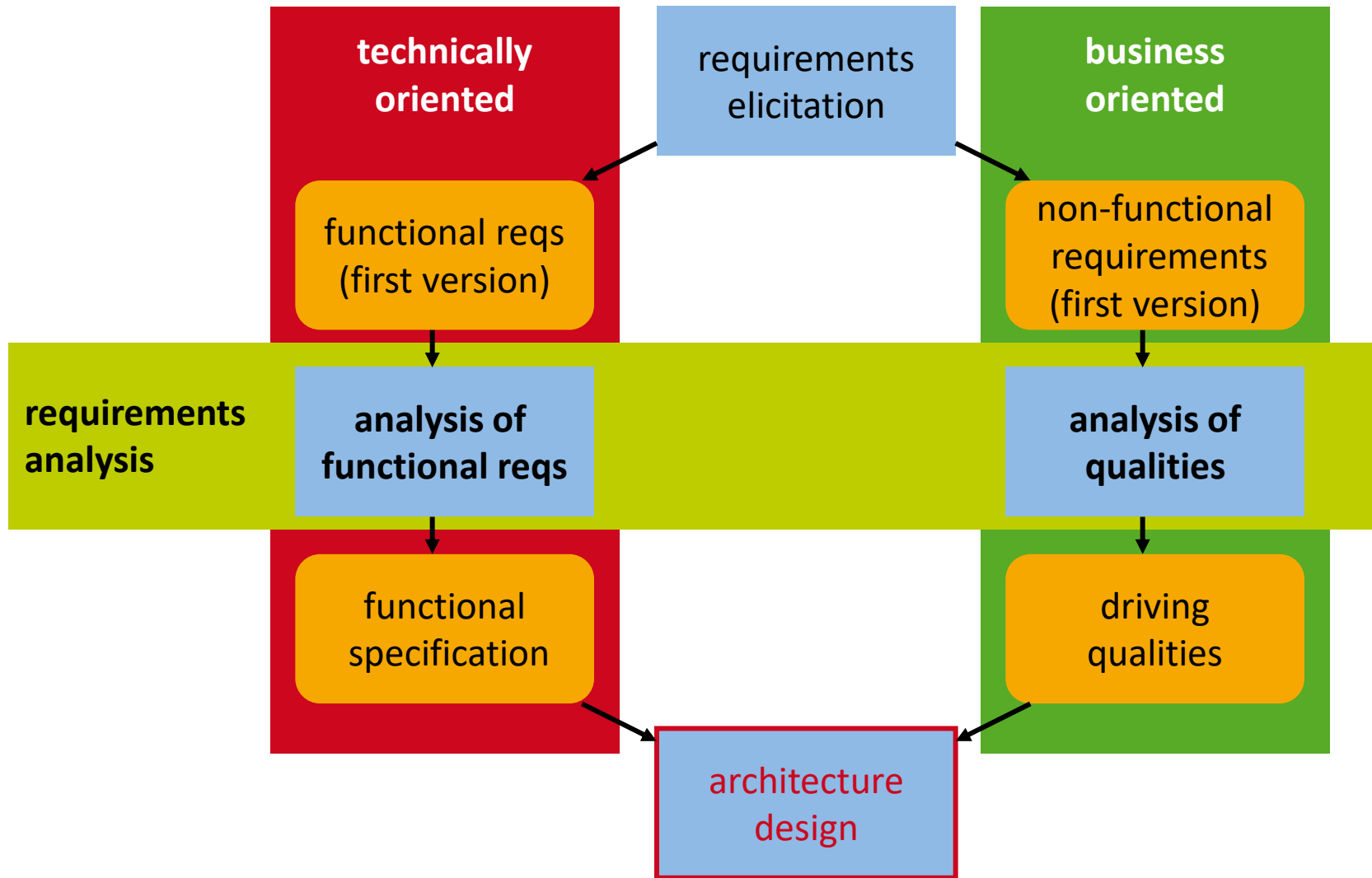
## Architecture Design



# Reminder: architecture design in the V model



# Architecture design needs functional spec and driving qualities



# What is architecture?

---

Bass, Clements, Kazman, 2003 (modified):

**The architecture of a system is the structure or the structures of the system, which comprise elements, the externally visible properties of those elements, and the relationship among them.**

► **The architecture defines elements of the system.**

- Architecture design is the first phase in which the system is no longer a black box.
- The designer begins to structure the system into parts.
- BCK: Architecture manifests the earliest design decisions.
- SK: Architecture is the blueprint for system integration.

# What is architecture? /2

---

Bass, Clements, Kazman, 2003 (modified):

**The architecture of a system is the structure or the structures of the system, which comprise elements, the externally visible properties of those elements, and the relationship among them.**

- ▶ The architecture is only one step further in refinement.
  - Now the elements are black boxes.
  - The architecture specifies what the elements do and how they interact **from an outside (system's) perspective** (often regarded as the element's responsibilities)
  - Central concept of architectures: **Interfaces**.



# What is architecture? /3

---

Bass, Clements, Kazman, 2003 (modified):

**The architecture of a system is the structure or the structures of the system, which comprise elements, the externally visible properties of those elements, and the relationship among them.**

- ▶ A system can have and usually has more than one structure.
  - Examples:
    - Design time elements (files, components, modules)
    - Run-time elements (processes, tasks, threads)
    - Behavioral elements (states, messages, queues)
  - The designer must consider different **architectural views**.

# Importance of architecture

---

Bass, Clements, Kazman, 2003 (again):

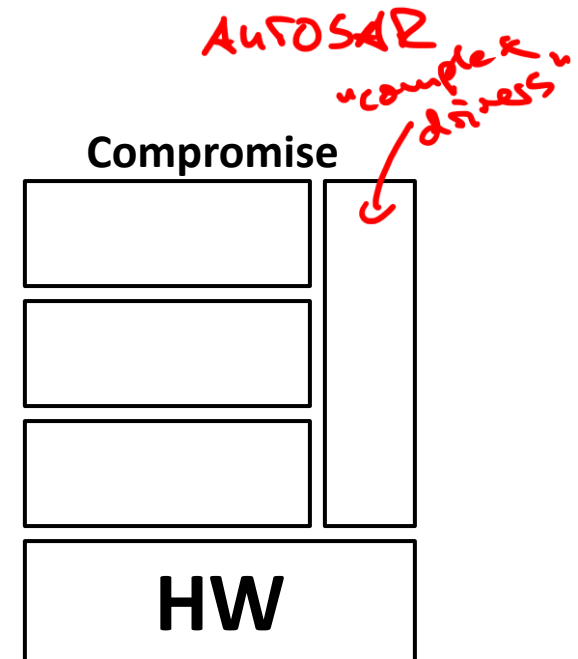
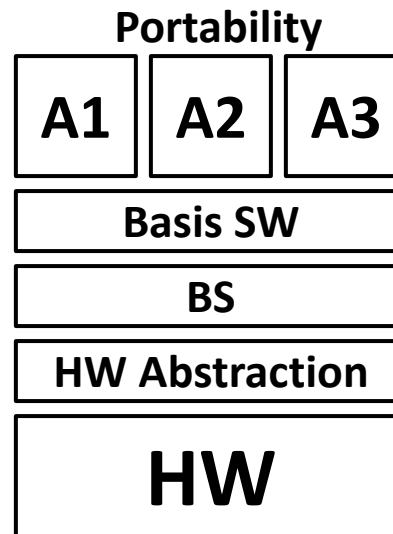
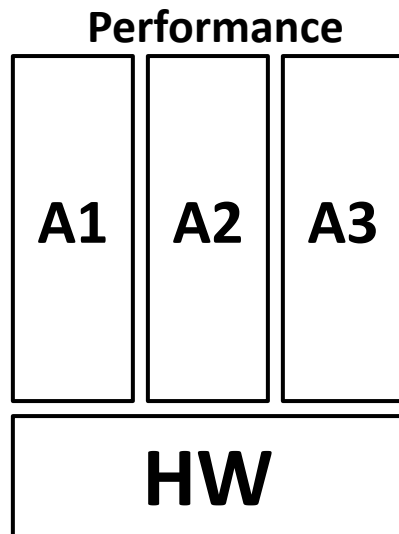
- ▶ Architecture represents earliest design decisions.
- ▶ They are the most difficult to get correct and the hardest to change later in the design process.
- ▶ They have the most far-reaching effects.

## Why?

- ▶ Architecture defines constraints on implementation.
- ▶ Architecture dictates organizational structure.
- ▶ Architecture inhibits or enables a system's quality.
- ▶  $\Rightarrow$  It is possible to predict system qualities by studying the architecture.

# Architecture design

- ▶ Take most important qualities (SEI: „architectural drivers“)
- ▶ Apply structuring principles which support these qualities
- ▶ Find a compromise, if qualities require contradicting structuring principles



# Example: Maintainability

- ▶ **Maintainability** is the property of a system which describes how well faults can be removed.

## General Scenario:

- ▶ Stimulus:
  1. A fault is detected
  2. Requirements change such that previous (implemented) design decisions become wrong
- ▶ Response:
  - Change is made such that fault is removed without introducing new faults
- ▶ Response measure:
  - Effort for removing fault
  - ~~Probability of introducing new faults~~

# Achieving Maintainability

- ▶ **General principle** (SEI: “Tactics”) to achieve maintainability:

## Keep changes local

- With respect to where the error can be found
  - With respect to where the change is done
  - With respect to which other parts are effected by the change.
- ▶ Concrete structuring principle:

## Information Hiding

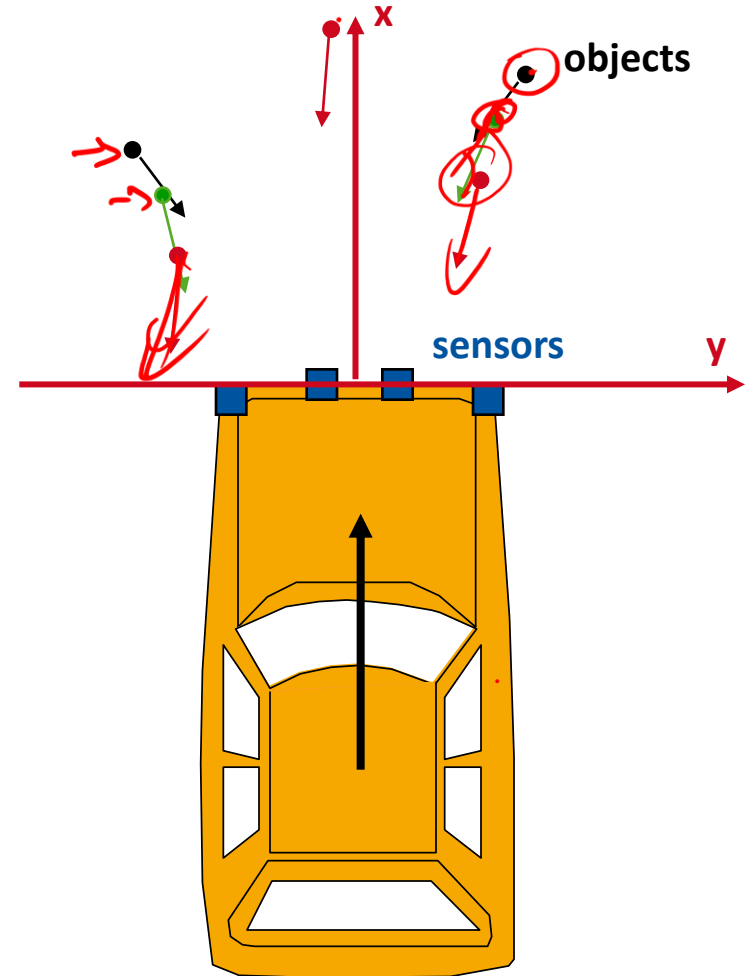
David Parnass, 1970s

# Example: A pre-crash sensing system

- ▶ The function of the pre-crash sensing system (PCSS) is pre-processing of sensor data to provide information about potentially colliding objects to pre-crash system (PCS), which will then decide what to do.

## Required sequence:

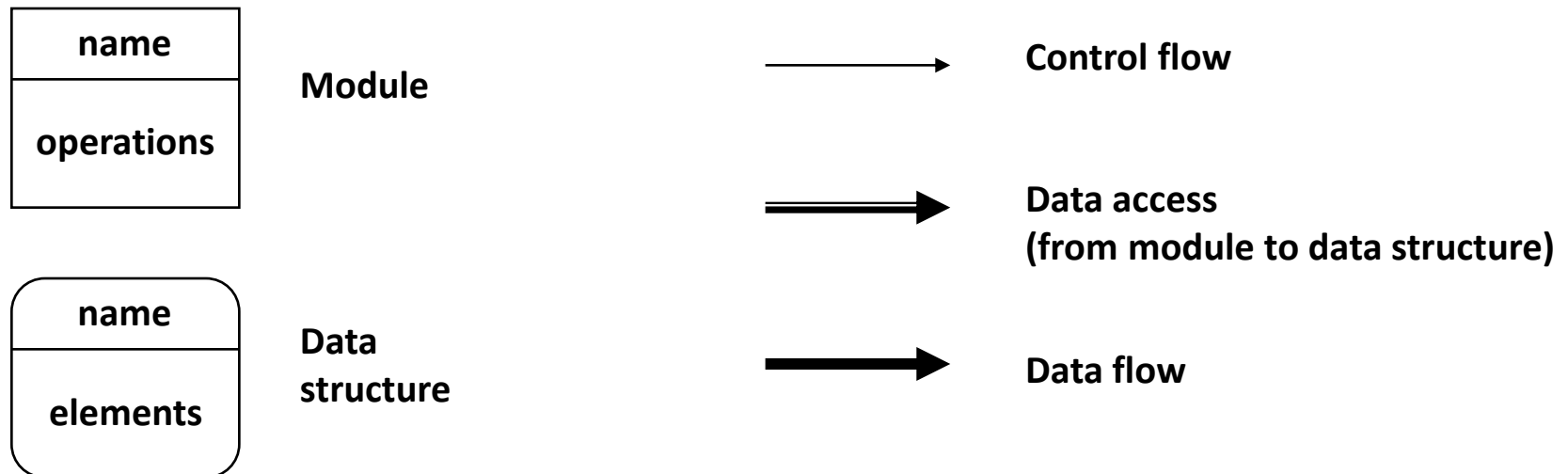
1. Read sensor data
2. Update list of objects (location, rel. speed, time of last measurement)
3. Identify dangerous objects
4. Send information to PCS



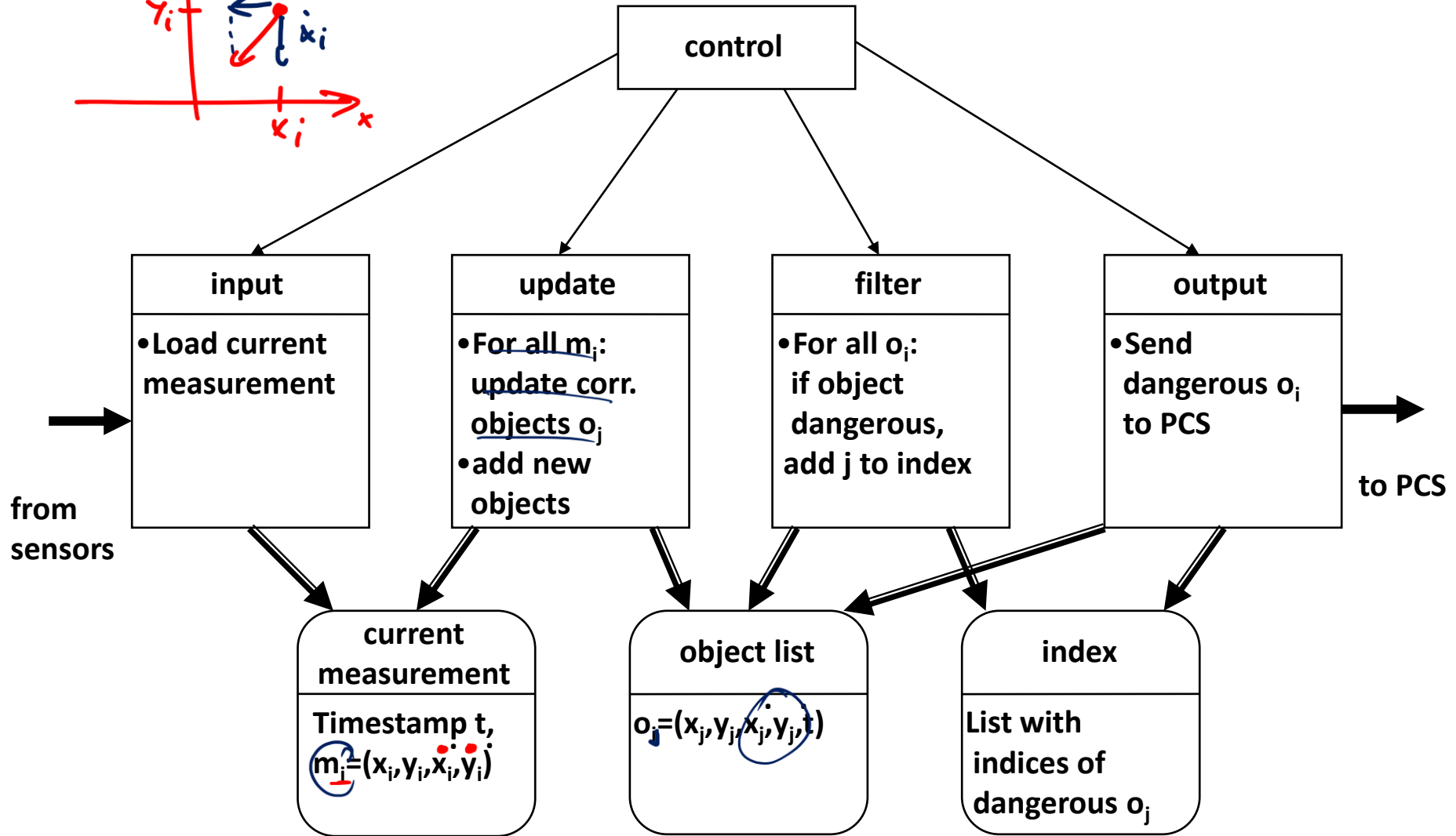
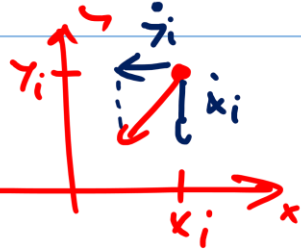
# Two alternative structures

- ▶ In the following two alternative structures will be presented.
- ▶ The first one is based on the functional sequence, the second one on information hiding.

## Representation elements:

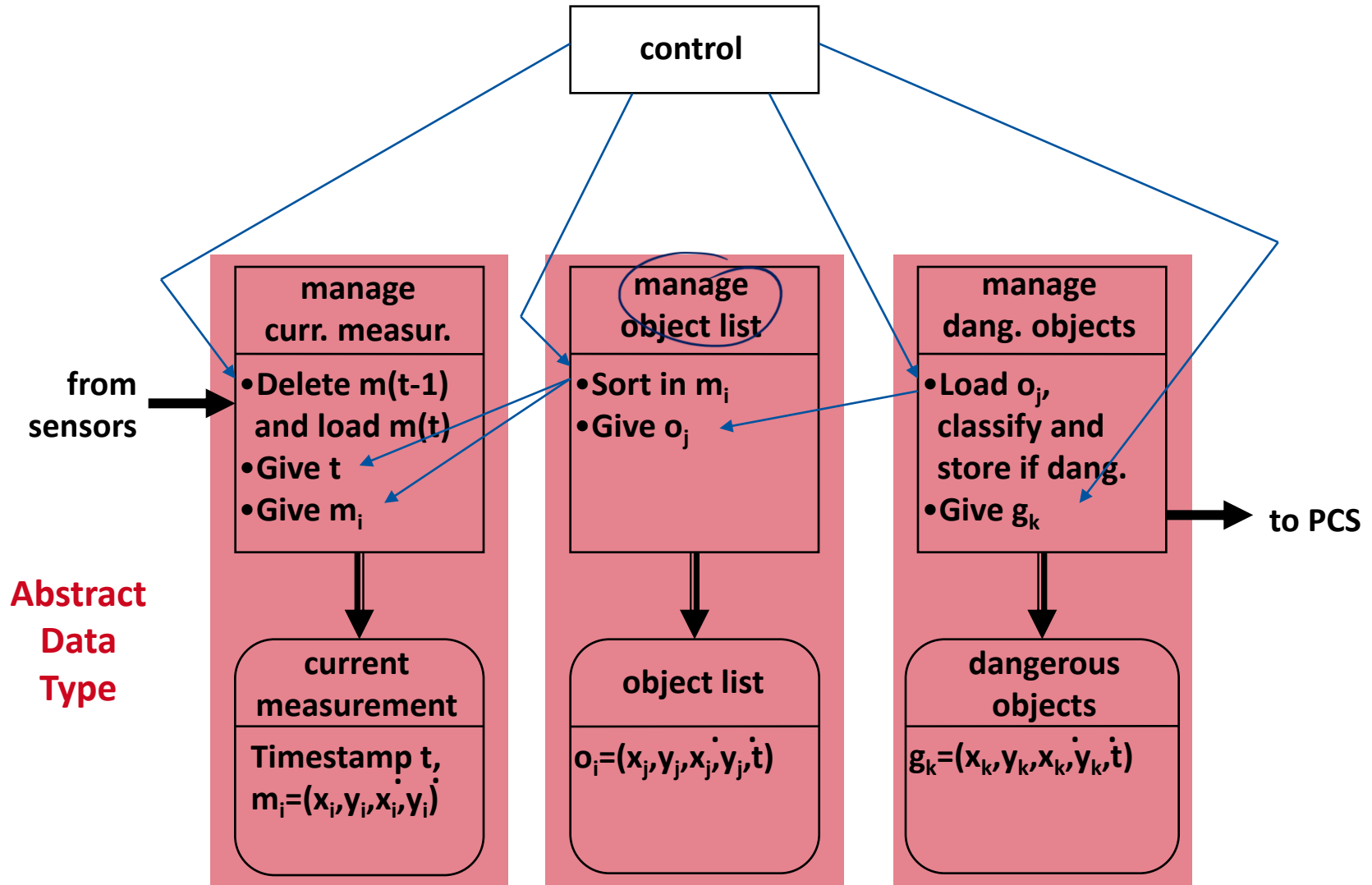


# First structure: based on functional sequence



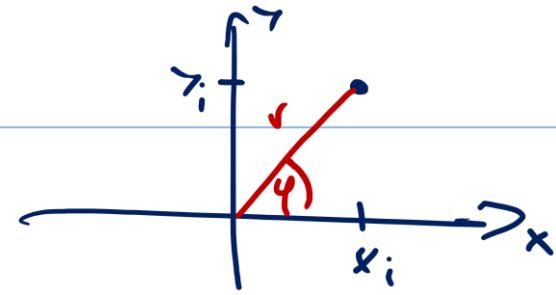


# Second structure: information hiding

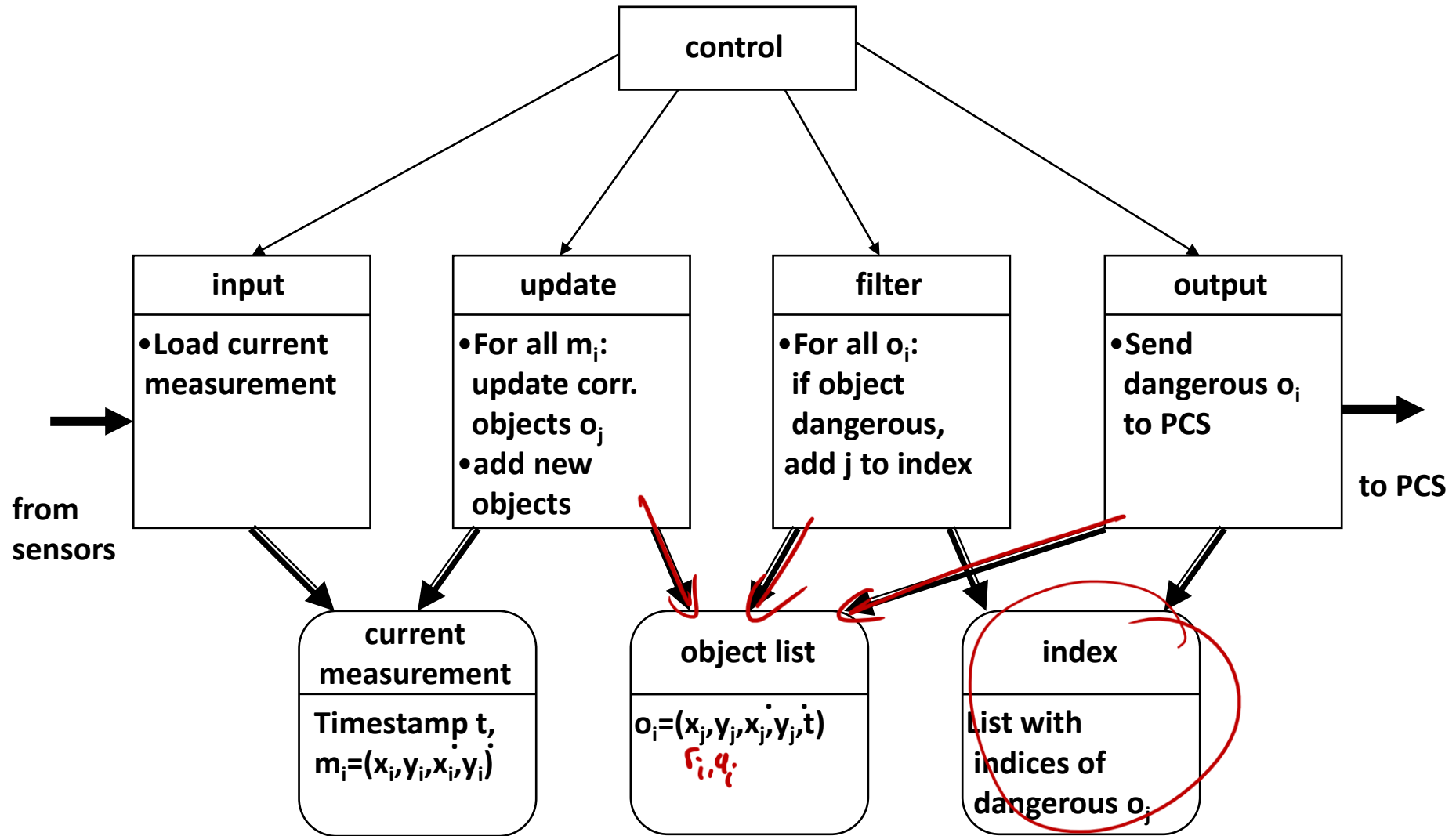


# Which is the better structure?

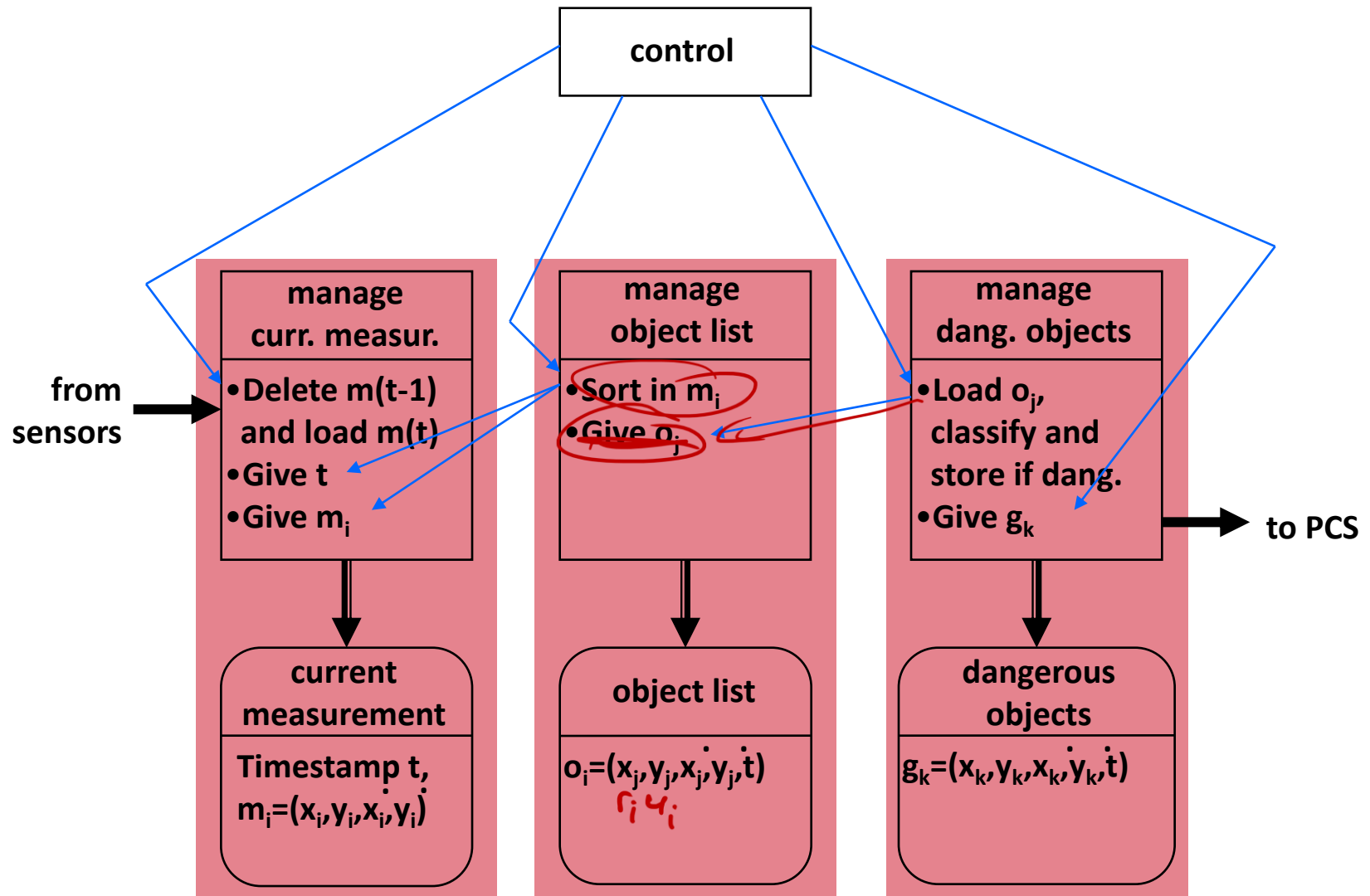
- ▶ Depends on criteria.
- ▶ Parnas, 1972:
  - Best criterion for modularization is maintainability/modifiability, i.e. the support of changes.
  - Changes mostly affect data structures
- ▶ Example:
  - Objects shall be stored in polar coordinates instead of cartesian coordinates.
  - Changes in first structure: 3 Modules
  - Changes in second structure: 1 Module



# First structure: based on functional sequence



# Second structure: information hiding



# Applying information hiding

---

1. Identify design decisions which are likely to change
2. Assign each of these decisions to one module, if possible.
3. Encapsulation: For each module, hide the design decision behind an abstract interface such that the other modules do not have to know about it. (“Secret of the module”)

## Information hiding supports:

- Modifiability
- Maintainability
- Reusability

## Information hiding does not support:

- Performance (e.g. memory efficiency)

# Other design principles

---

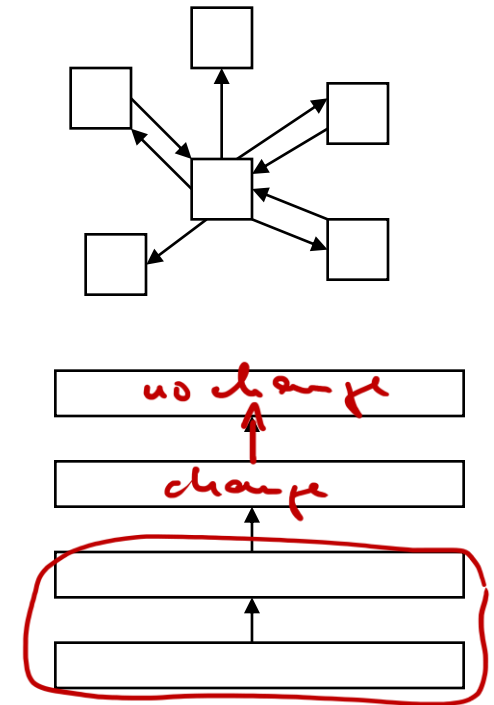
- ▶ **Portability** is the property which describes how well the system can be moved to a different platform (hardware, operating system).

## General Scenario:

- ▶ **Stimulus:**  
A new platform shall be used
- ▶ **Response:**
  - Changes are made such that the system is running on the new platform
- ▶ **Response measure:**
  - Effort for performing changes incl. testing

# Structuring principle supporting portability?

- ▶ **Layered architecture**
- ▶ General principle similar to information hiding: keep changes local
- ▶ Difference:
  - Information hiding is flat (abstraction direction from inside to interface of modules)
  - Layered architecture is hierarchical (modules on different abstraction levels, abstraction direction from bottom to top)



## ► Reliability

- **Terminology:**

Reliability is the property which describes how much we can expect the system to fulfill its specified function under certain conditions (time period, environment conditions)

- **Aim:** Make it sufficiently probable that the system will fulfill its specified function under the specified conditions

- Basic approach?

→ Redundancy!

→ Functional Safety &  
System Reliability



► Booming area of software engineering research

► Architecture Description Languages (ADLs):

→ <http://www.sei.cmu.edu/architecture/adl.html>

UML } orig  
SysML

► Main distinction:

■ Models for structure:

- Class diagrams, object diagrams
- Signal/data flow diagrams, block diagrams
- Deployment diagrams
- etc.

■ Models for behavior:

- State Charts, Petri nets