



# Real-time Software Systems Engineering (2IN70)

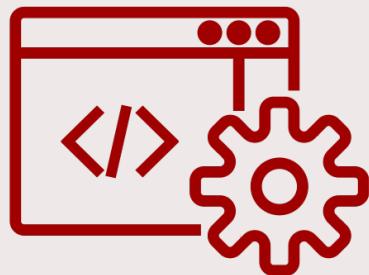
## Automotive Software Engineering Autumn 2022

### WEEK7 – Testing and Integration

11 JANUARY 2023

- Ion Barosan – [i.barosan@tue.nl](mailto:i.barosan@tue.nl)
- Mathematics and Computer Science – SET

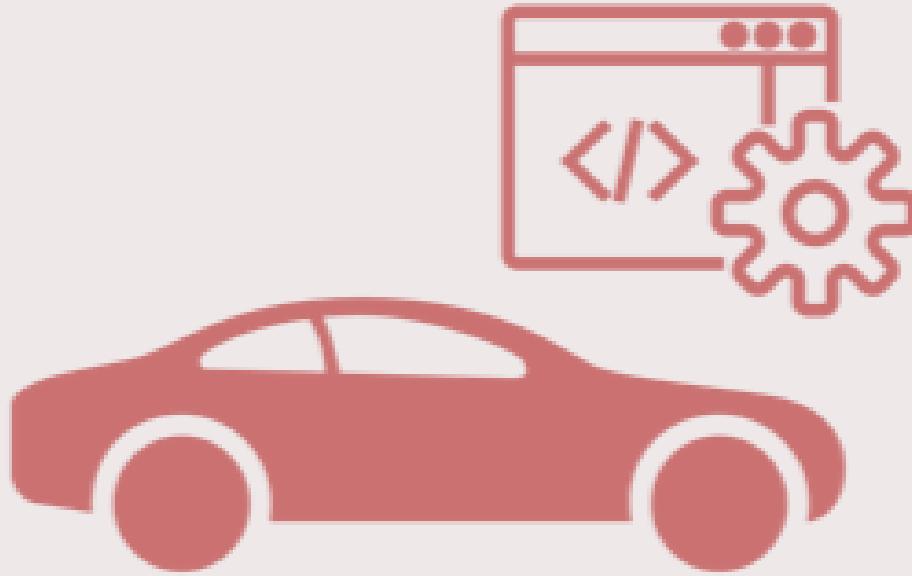
# Content



Automotive  
Software  
Development

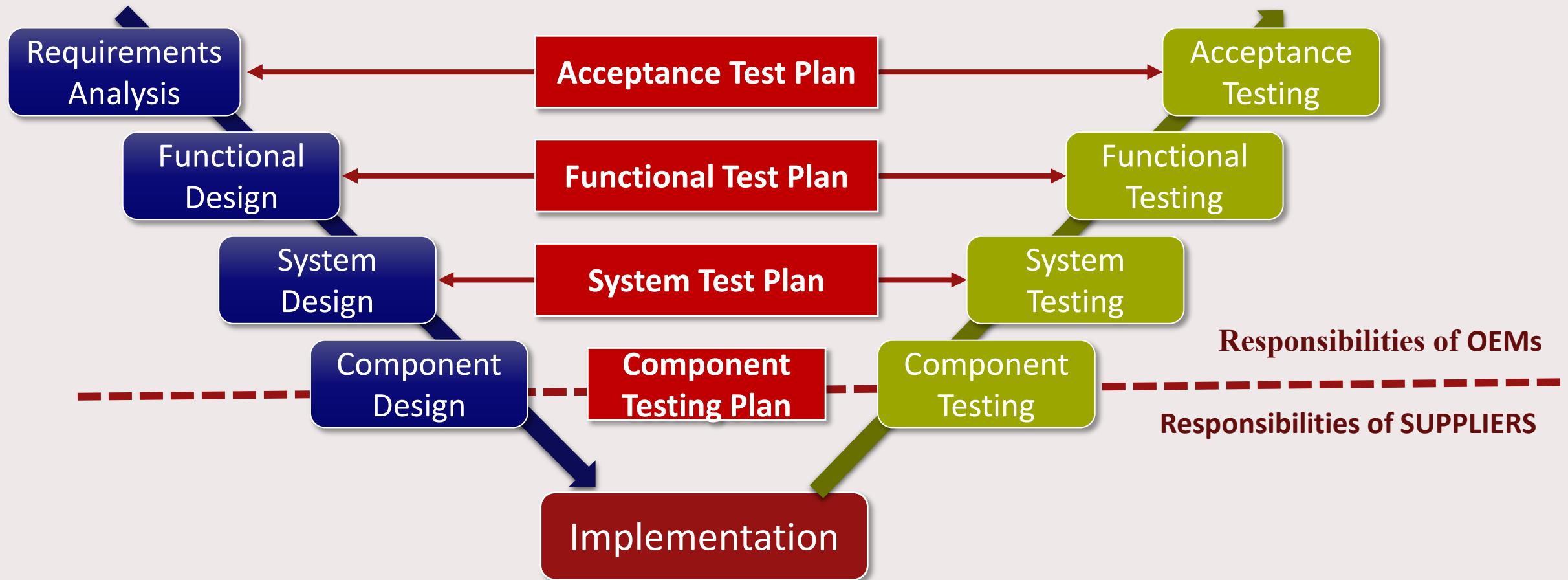
1. Automotive Software Development Process
2. Variant Management
3. Integration Stages of Software Development
4. Testing Strategies
5. Construction Database in Automotive Software Engineering

# 1

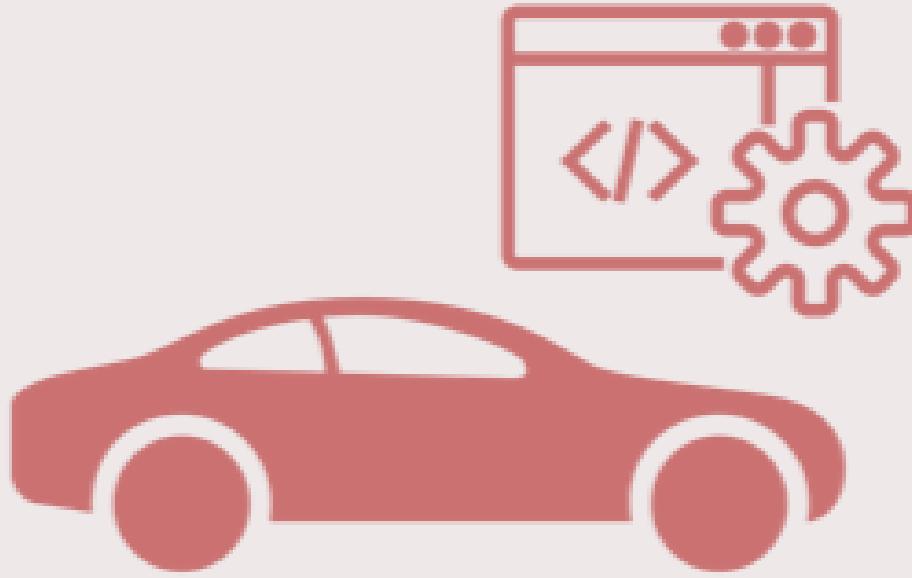


## Automotive Software Development Process

# V-shaped model of software development process in automotive software development



# 2



## Automotive Software Development Process Variant Management

# Automotive Software Engineering

## Variant Management



Automotive market is based on **Variability** - i.e. *the locations in the product (software) where it can be configured.*



A good database of requirements and construction elements is **key to success**.



The **customers** expect the **ability to configure** the car with the latest and greatest **features of hardware, electronics and software**.

# Automotive Software Engineering

## Variant Management

**Configuration**—when we configure parameters of the software without modifying its internal structure

### Used

In the non-safety critical functions

**Example** – the engine calibration or in configuring the availability of functions (e.g. rain sensors).

**Compilation**—when we change the internal structure of the software, compile it and then deploy on the target ECU

### Used

To ensure that the software always behaves in the same way

**Example** - the availability of the function for collision avoidance by breaking.

# Automotive Software Engineering

## Variant Management - Configuration

Configuration is often referred to as runtime variability

Changing the software can be done after the software is compiled

The code for the software component is compiled once

Two different configuration files are used when deploying the software

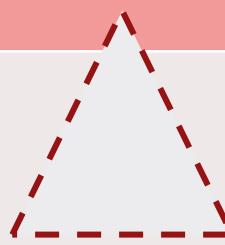
The software must be tested using multiple scenarios

The software designers need to be able to prevent use of the software component with invalid configurations

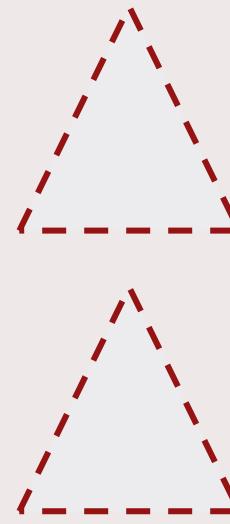
# Automotive Software Engineering

## Variant Management - Configuration

Software component  
(e.g. windshield wipers controller)



Variability point: speed controller



Configuration 1: no rain sensor

Configuration 2: with rain sensor

Compile once the software component.

Use **two different** configuration files when deploying the software.

# Automotive Software Engineering

## Variant Management - Compilation

**The Software Component cannot be modified (configured) after its compilation, during runtime.**

**Compilation**  
so- called *Design Time Variability*

The designers must decide during design which variant is being developed

**Different code bases for the software component**

The development of the variants can be decoupled from each other

The designers have to maintain different code bases at the same time

**Advantages**

The assurance that the code is not tampered with in any way after the compilation

The code can be tested, and once deployed there is no way that an incorrect configuration can break the quality of the component.

**Disadvantage**

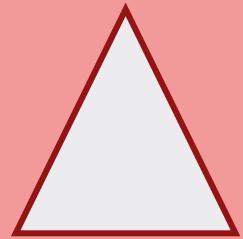
High cost of maintenance of the code base—parallel maintenance.

# Automotive Software Engineering

## Variant Management - Compilation

Software

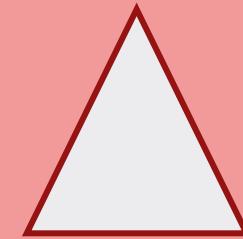
Component var 1  
(e.g. windshield  
wipers controller)



Variant 1: no rain sensor

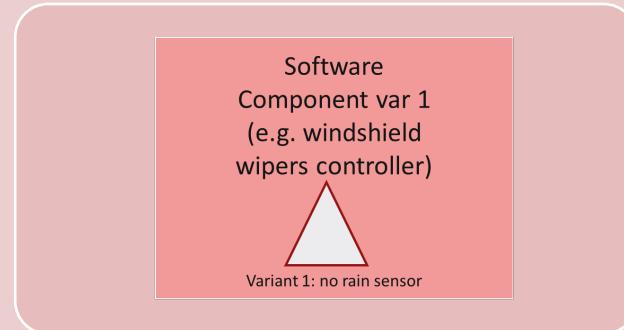
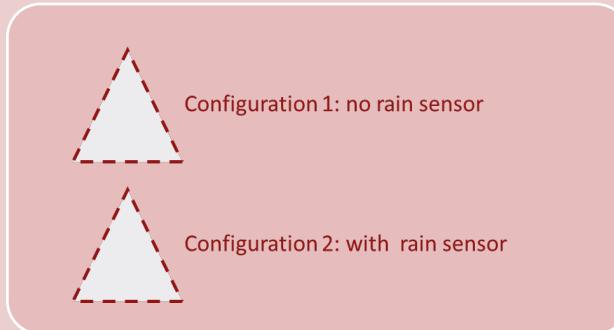
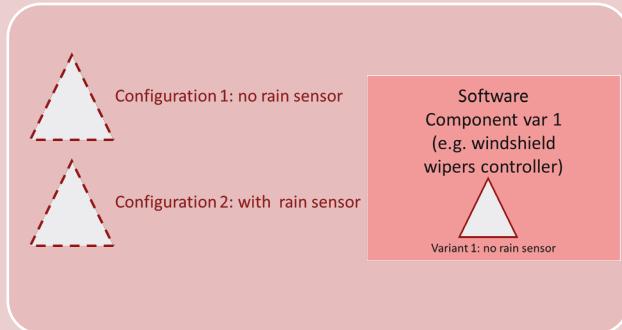
Software

Component var 2  
(e.g. windshield  
wipers controller)



Variant 2: with rain sensor

# Practical Variability Management



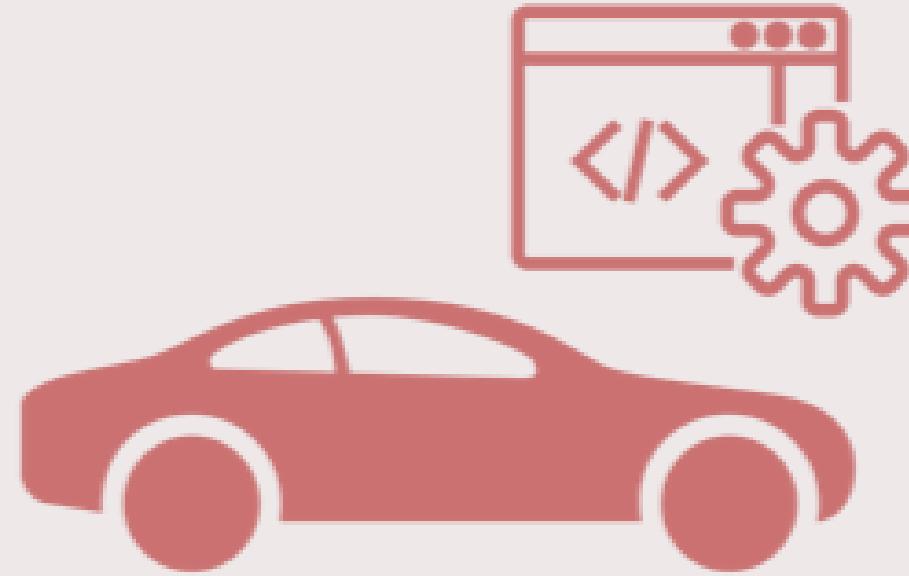
**Both variability management mechanisms are used in practice.**

**Compile time Variability - used when the software is an integral part of an ECU**

**Configuration Variability- used when the software can be calibrated to different types of configurations during deployment**

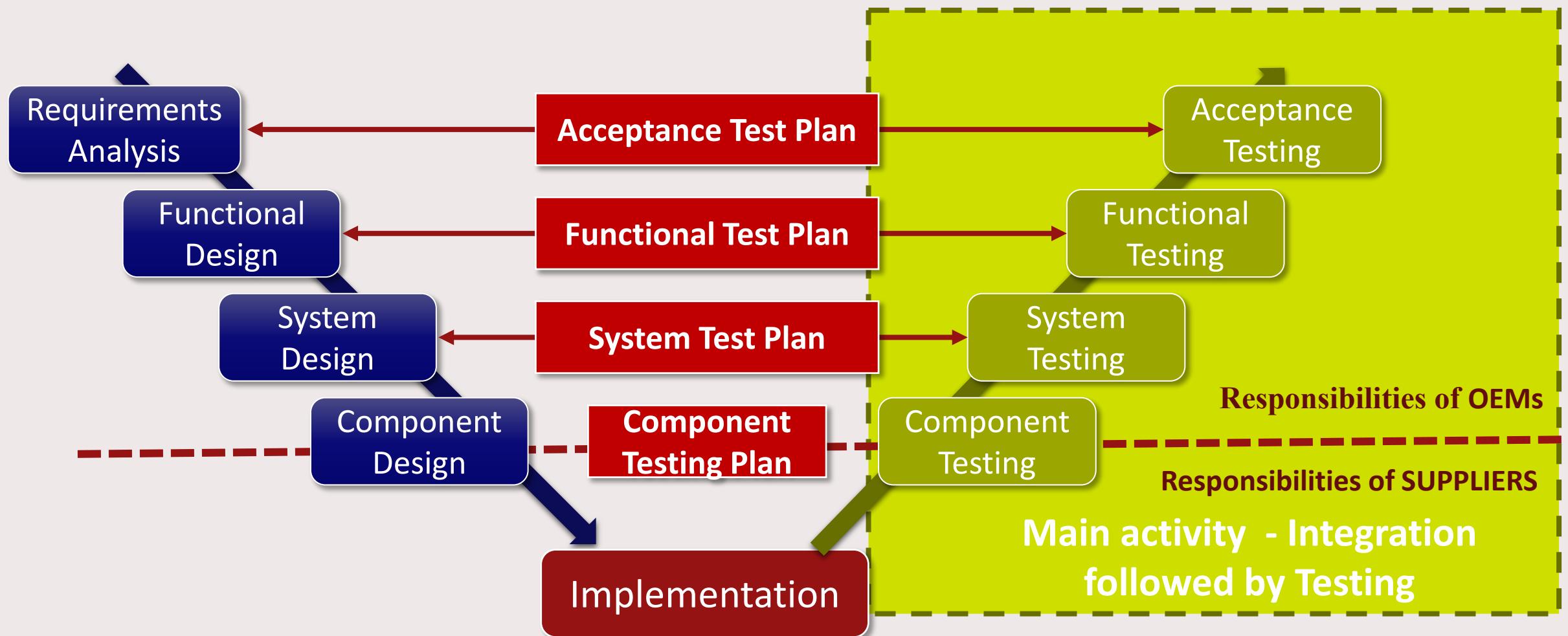
Example: *configuration on the assembly line, calibration of the engine and gearbox depending on the powertrain*

# 3

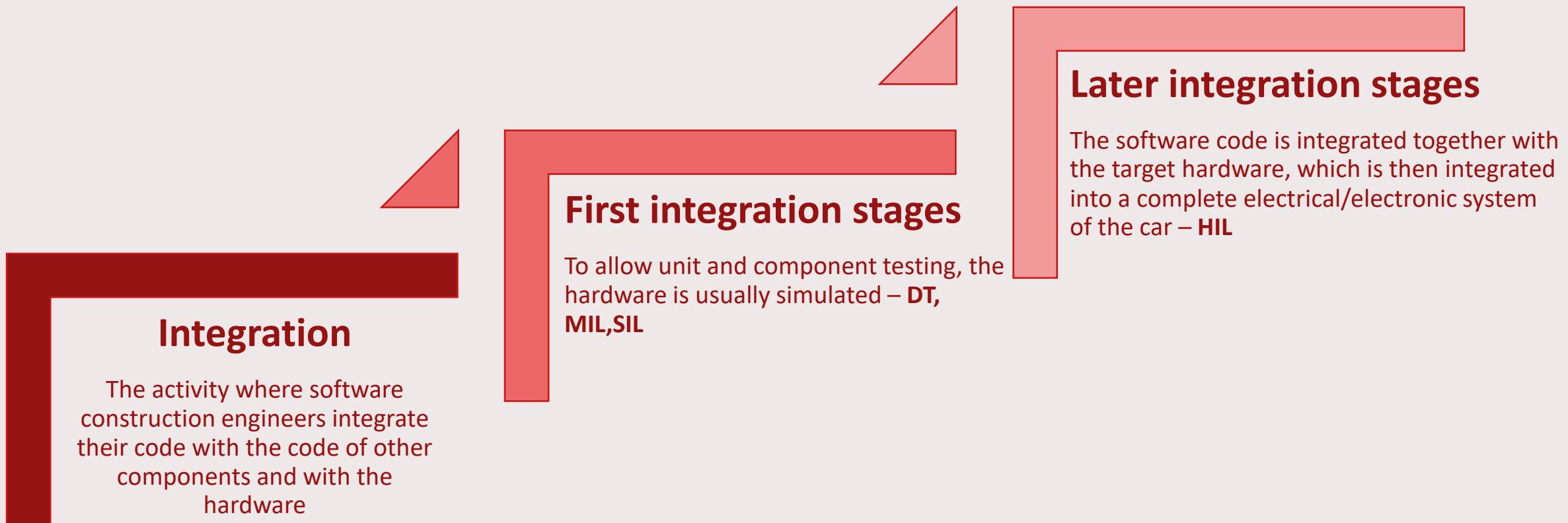


## Automotive Software Development Process Integration Stages of Software Development

# Integration Stages of Software Development



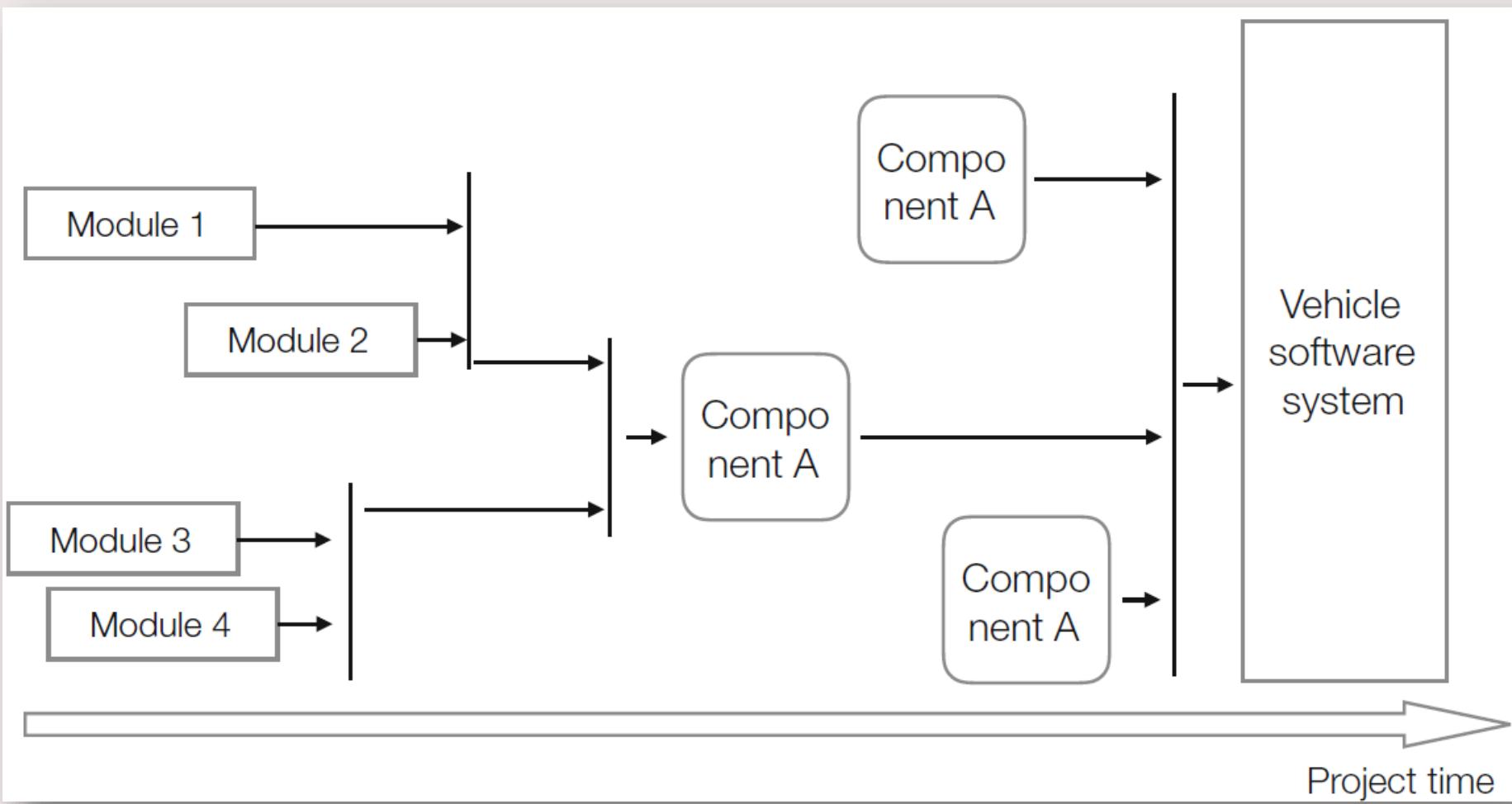
# Integration Stages of Software Development



# Types of integration

Software Integration	Software-Hardware Integration	Hardware Integration
<p><b>Two (or more) software components are put together and their joint functionality is tested.</b></p> <p>Integration depend on the what is integrated:</p> <ol style="list-style-type: none"><li><i>Merging of the source code if the integration is on the source code level</i></li><li><i>Linking of two binary code bases together</i></li><li><i>Parallel execution to test interoperability.</i></li></ol>	<p><b>The software is integrated (deployed) to the target hardware platform.</b></p> <p>The focus is on the ability of the complete ECU to be executed and the main testing type is <b>component testing</b>.</p>	<p><b>The focus is on the integration of the ECUs with the electrical system.</b></p> <p>The focus is on the <b>interoperability</b> of the nodes and basic functionality, such as <b>communication</b>.</p> <p>The testing related to this type of integration is <b>system testing</b>.</p>

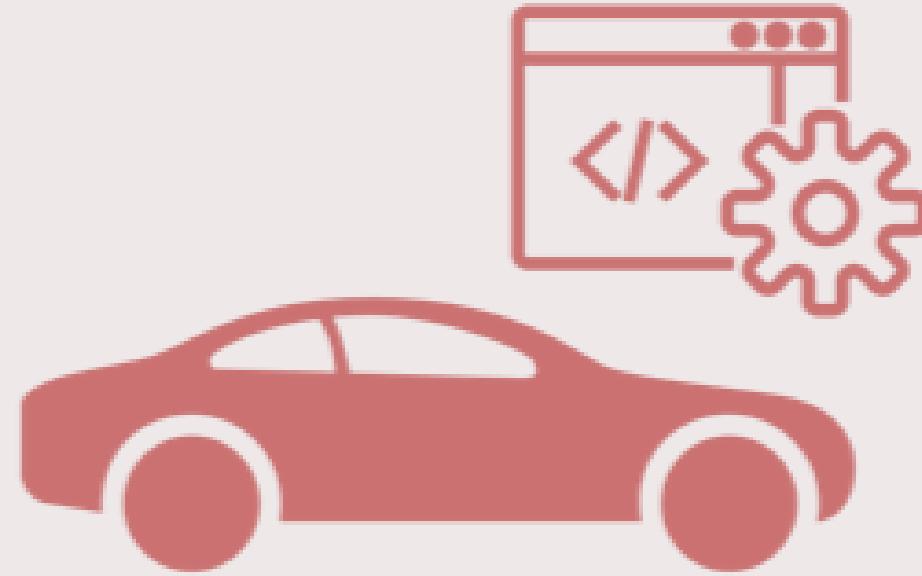
# Software integration with integration steps



The integration steps (vertical solid black lines) are not synchronized as the development of different software modules is done at different pace. Each integration cycle is done several times during the project.

*Automotive Software Architectures  
Miroslav Staron*

# 4



## Automotive Software Development Process *Testing Strategies*

# Overview of Software Testing

“Testing shows the presence, not the absence of bugs.” —Edsger W. Dijkstra

A fault, also called “defect” or “bug,” is an erroneous hardware or software element of a system that can cause the system to fail!

## Test-Driven Development (TDD)

Every step in the development process must start with a plan of how to verify that the result meets a goal

The developer should not create a software artifact (a system requirement, a UML diagram, or source code) unless they know how it will be tested

A test case is a particular choice of input data to be used in testing a program

A test is a finite collection of test cases

## A key tradeoff of testing

Testing as many potential cases as possible while keeping the economic costs limited

## Why Testing is Hard

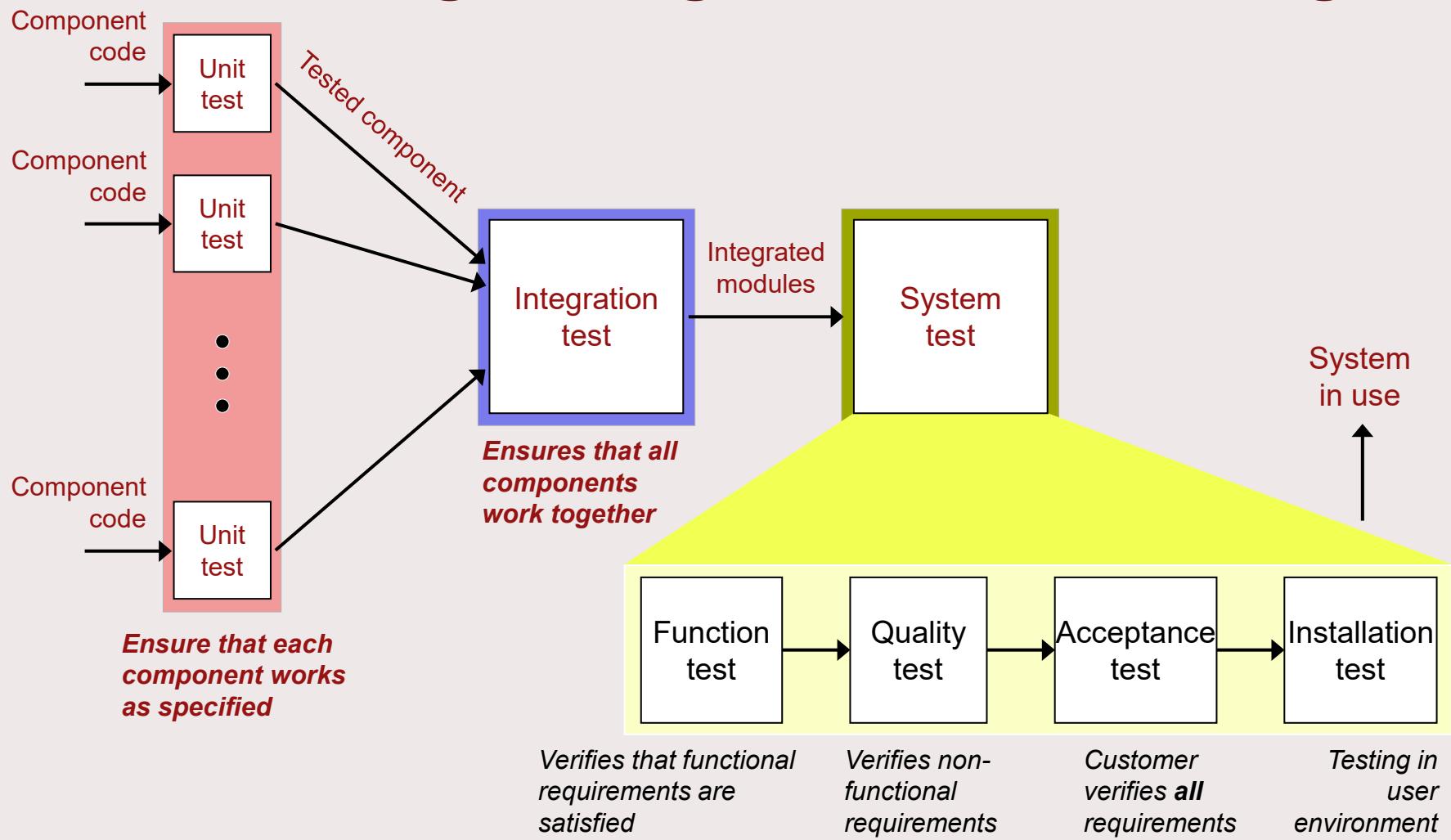
Our goal is to find faults as **cheaply** and quickly as possible.

Ideally, we would design a single “right” test case to expose each fault and run it

In practice, we have to run many “**unsuccessful**” test cases

Tests which do not expose any faults

# Logical Organization of Testing





# Test Coverage

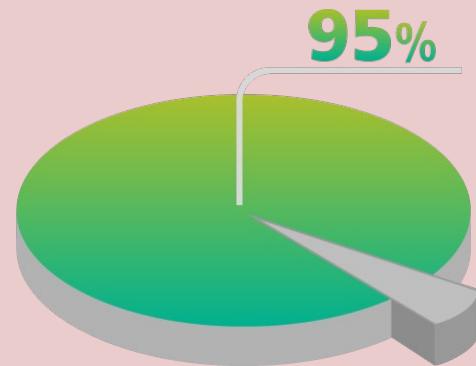
## Test Coverage

measures the degree to which the *specification or code* of a software program has been exercised by tests



## Code Coverage

measures the degree to which *the source code of a program has been tested*



## Code Coverage Criteria

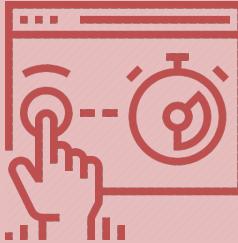
equivalence testing

boundary testing

control-flow testing

state-based testing

# Automotive Software Testing Strategies

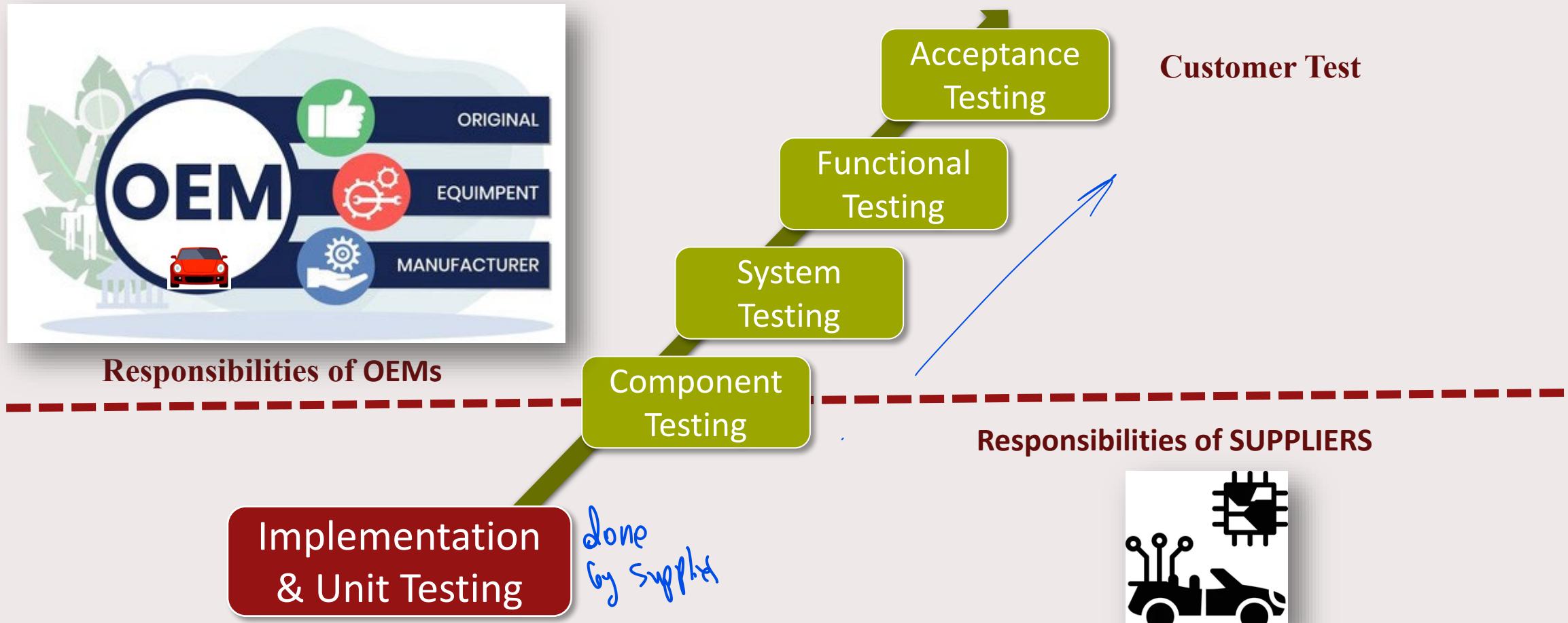


Requirements engineering progresses from higher abstraction levels towards more detailed, lower abstraction levels

**Testing** the software starts from the most atomic type of testing—unit testing—where they test each function and each line of code

**Testing** gradually progress by testing **entire components** (i.e. multiple units linked together), then the **entire system** and finally each function.

# Testing phases in automotive software development



Johnson  
meta  
domain

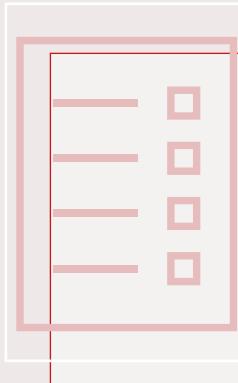
problem  
domain

use r  
domain

# Unit Testing

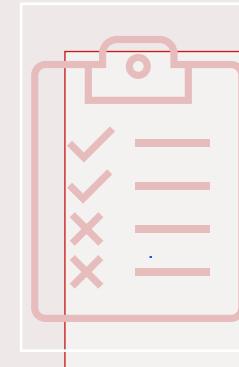
- testing any  
class any method  
or function

object  
functional  
one (property  
initialize)



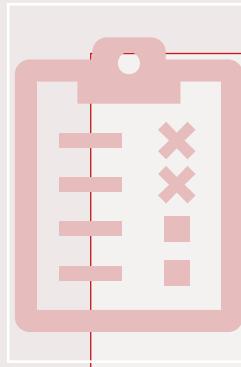
## Unit test is the basic test

Is performed on individual entities of software such as classes, source code modules and functions.



## The goal of unit testing

To find defects related to the implementation of atomic functions/methods in the source code.



test = activity of diagram.

## The basic scheme of unit testing

1. Create automated test cases which combine individual methods with the data that is needed to achieve the needed quality.
2. The result is then compared to the expected result, usually with the help of assertions.

- testing the boundaries  
- testing control flow

Any system have primary behaviour & secondary behaviors.

Engine: primary = spir (Ad)  
secondary = algorithm

# Unit Testing

Example unit test for testing the status of windshield wiper module

```
1  using System;
2  using Microsoft.VisualStudio.TestTools.UnitTesting;
3  using WindshieldSimulator;
4
5  namespace WindshieldTest
6  {
7      [TestClass]
8      public class BasicSuite
9      {
10         // unit test method
11         [TestMethod]
12         public void TestCreationInitialState()
13         {
14             // arrange
15             WindshieldWiper pWiper;
16
17             // act
18             pWiper = new WindshieldWiper();
19
20             // assert
21             Assert.AreEqual(pWiper.Status,
22                             position.closed,
23                             "Initial status should be /closed/");
24         }
25     }
26 }
```

The most interesting are lines 21–23 since they contain the so-called assertion.

The assertion is a condition which should be fulfilled after the execution of the test code. In our example the assertion is that the status of the newly created object (line 21) is “closed” (line 22). If it is not the case, then the error message is logged in the testing environment (line 32) and the execution of the new test cases continues.

Unit testing is most often automated. Frameworks like CppUnit, JUnit or Google test framework can orchestrate the execution of unit tests, allowing us to quickly execute the entire set of tests (called test suites) without the need for manual intervention.

# Component Testing

Put component together at larger components.  
Test → in environment → simulate this environment



Called **integration testing**, as the goal of this type of testing is to test the integrations, i.e. links, between units of code within one of many components.

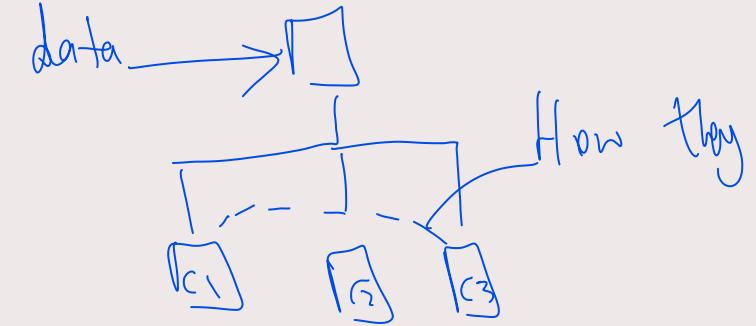
We use **stubs** to simulate the environment of the tested component or the group of the components

Component tests focus on the interaction between the stubs and the component under test.

The goal of this type of testing is to verify that the structure and behavior of the interfaces is implemented correctly.

- interaction between the environment
- hardware simulators
- digital twins

# Component Testing



## Automotive systems component testing

- **Simulation Environment** using models (the so-called Model-In-the-Loop or MIL testing)
- **Hardware simulators** (the so-called Hardware-In-the-Loop or HIL testing).
- **Digital Twins** *are physical component of the car & virtual environment.*

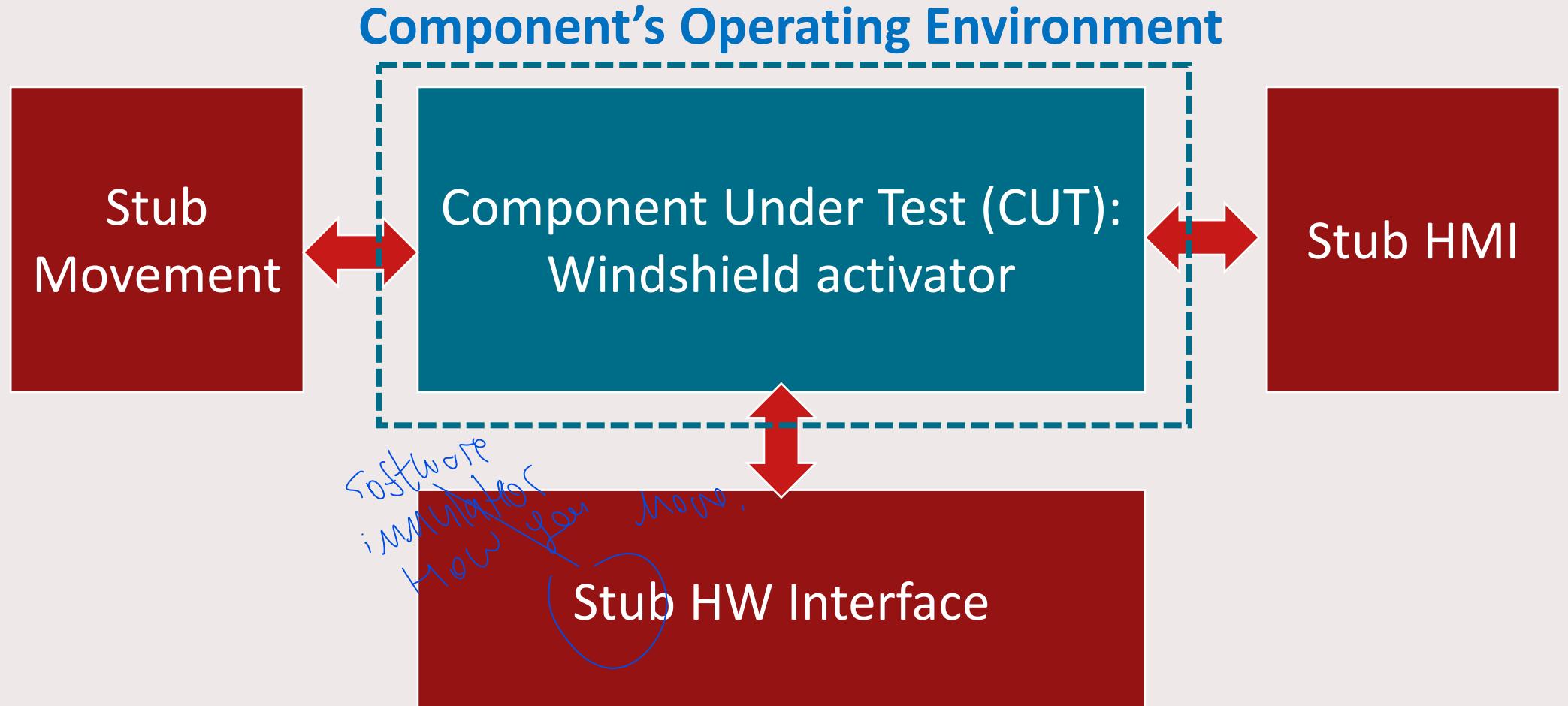


Simulation Models

Hardware

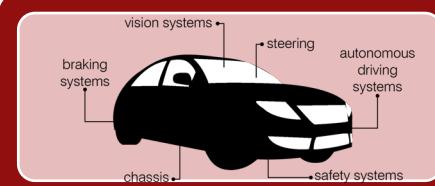
Copyright: © Copyright 2022 dSPACE GmbH

# Component under test with the simulated environment



have to place the system into  
the operating environment

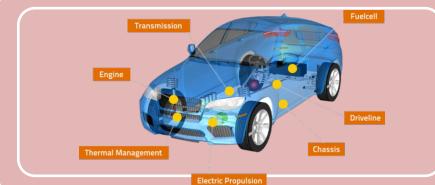
# System Testing



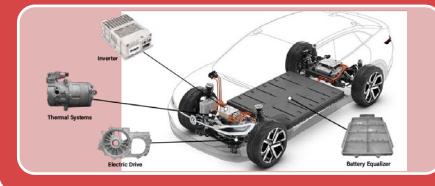
The entire system is assembled and tested as a whole



The focus of system testing is on checking whether the system fulfills its specifications in several ways



System Testing is a very costly way of testing and very inefficient, as fixing the defects found in this phase requires frequent changes in multiple components.



In the automotive software this type of testing is often done using the so-called “box cars”—the entire electrical system being set up on tables without the chassis and the hardware components.

# System Testing – Verifying aspects



# Functional Testing

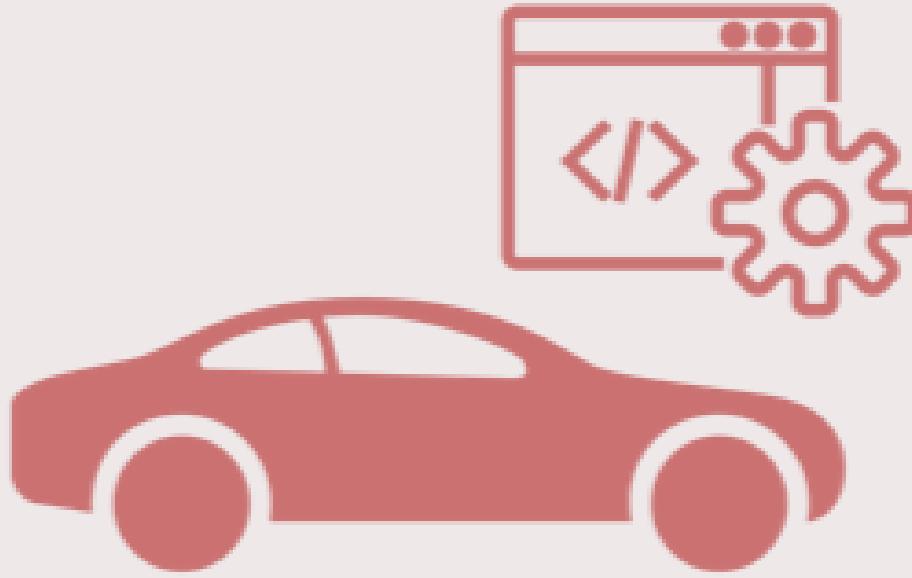
Verifies that the functions of the system work according to their specification

Tests correspond to the functional requirements in the form of use cases and are quite often specified according to the use cases

Tests recreate the situations when the system could be activated

In case of functional test fails, it is difficult to find the defect; many components that take part in the interaction

# 5



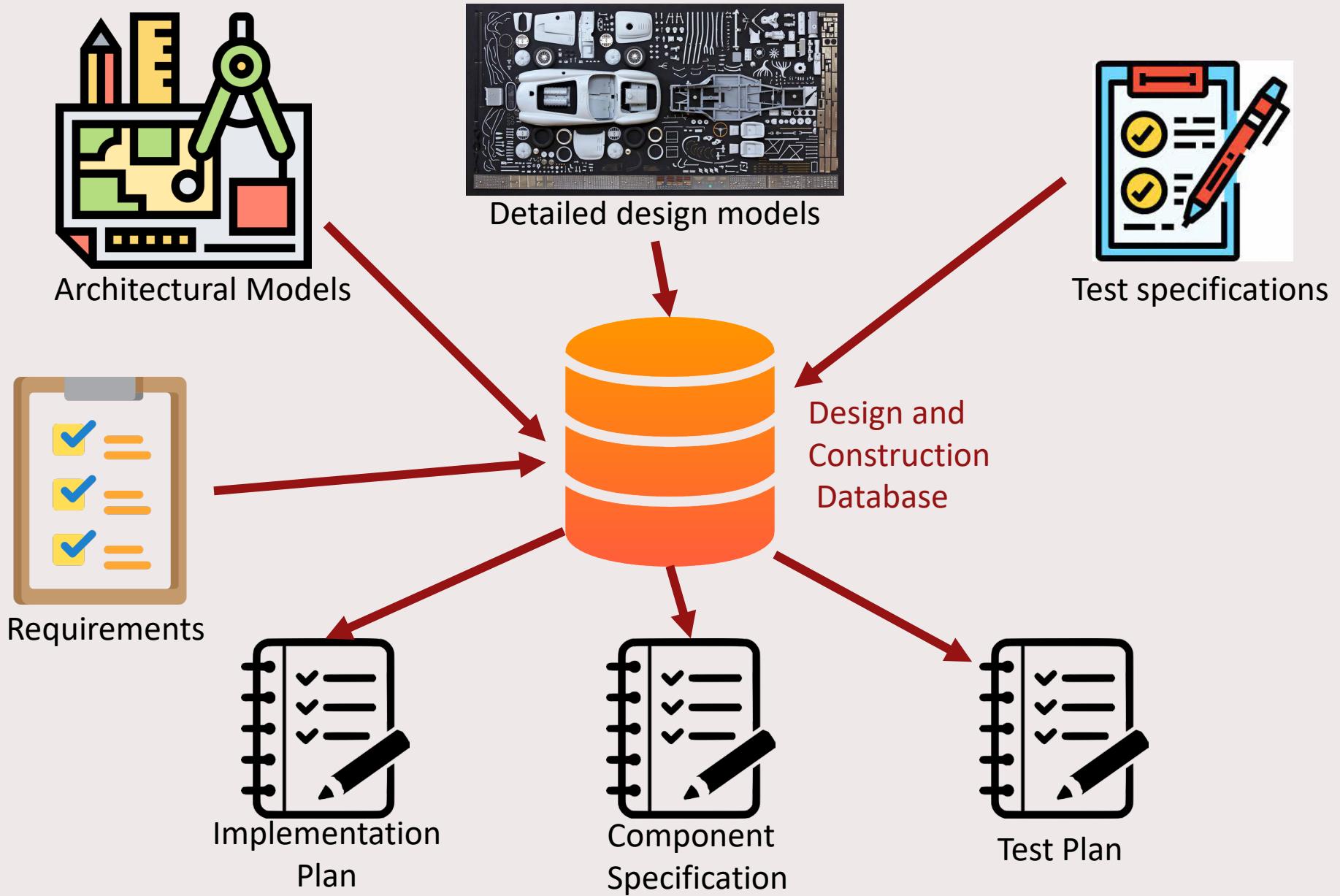
## Construction Database and Its Role in Automotive Software Engineering

# Construction Database and Its Role in Automotive Software Engineering

The requirements need to come together somehow and that's why we have the process and the infrastructure for requirements engineering.

## The Construction Database

- Contains all elements of the design of the electrical system of the vehicle— components, electronic control units, systems, controllers, etc. The structure of such a database is hierarchical and reflects the structure of the vehicle.
- Grows over time and is version-controlled as different versions of the same elements can be used in different vehicles (e.g. different year models of the same car or different cars).
- Each of the elements in the database has a set of requirements linked to it.
- The requirements are also linked to one another to show how they are broken down.



# Tools

- Doors + IBM Rhapsody
- SystemWeaver - Systemite

# Working with Function Requirements

File   Welcome   Dashboard   Items   Projects

Back   Forward   Open item   New item

Copy   Open   Overview   Attributes

Find   Versions   New issue   Add note

CM   Issues   Notes

Libraries   Start Server   Impact analysis

Move items...   Manage libraries   Functions Specification

Navigation   Items   Edit   Find   Complete Status   Issues and Notes   Notes   Security   Server   System Requirements

**B SystemWeaver - demonstration(1)**   **B Example(1)**

**B Baseline Architecture - Default**   Edit

Name	Status	V
SystemWeaver - demonstration	Work	
SystemWeaver - demonstration	Work	
Function requirements	Work	
Adaptive cruise control	Work	
Requirements	Work	
Adjust speed	Work	
Emergency braking	Work	
<b>Maintain speed</b>	Work	
Audio control	Work	
Climate control	Work	
Constant cruise control	Work	
Indicator Lamp	Work	
Low speed emergency braking	Work	
Radio	Work	
Seat heating	Work	
Washer wiper	Work	
Design architecture	Work	
Design functions	Work	
Adaptive cruise	Work	
CruiseControlManager	Work	
Requirements	Work	
CCThrottleReq	Work	
CCBrakeReq	Work	
UnConnected123	Work	
EmergencyBrake	Work	
AdaptiveSpeed	Work	
CruiseControlActive	Work	
CruiseSpeed	Work	
RadarFailure	Work	
VehicleSpd	Work	

**Impact analysis** | Maintain speed | Function requirement

Baseline: SystemWeaver - demonstration (1)

**Test**

**Design**

**Function**

The diagram illustrates the impact analysis for the 'Maintain speed' function. It shows three main components: 'Adaptive cruise control', 'Adaptive cruise', and 'Cruise control'. The 'Adaptive cruise control' component contains a 'Requirements' section with a red 'Maintain speed' requirement. This requirement has arrows pointing to several test cases: 'Test jerk', 'Cruise speed under current speed', 'Cruise speed over current speed', 'Cruise control from stand still', and 'Adaptive speed with moving obstacle'. The 'Adaptive cruise' component also contains a 'Requirements' section with a red 'CC Regulate speed' requirement, which has arrows pointing to the same five test cases. The 'Cruise control' component contains a 'Test jerk' requirement, which also points to the same five test cases.

# Viewing Function Specifications - Adaptive cruise control

Screenshot of the SystemWeaver application interface showing the function specification for Adaptive cruise control.

The left sidebar shows the Baseline Architecture - Default tree structure:

- SystemWeaver - demonstration (Work, 1)
- SystemWeaver - demonstration (Work, 1)
- Function requirements
  - Adaptive cruise control (Work, 2) [highlighted with a red box]
  - Audio control (Work, 2)
  - Climate control (Work, 1)
  - Constant cruise control (Work, 1)
  - Indicator Lamp (Work, 1)
  - Low speed emergency braking (Work, 1)
  - Radio (Work, 1)
  - Seat heating (Work, 1)
  - Washer wiper (Work, 1)
- Design architecture (Work, 1)
- Hardware architecture (Work, 1)
- Test (Work, 1)

The right panel displays the Overview for the selected "Adaptive cruise control" function:

Last Changed	Last Changed By	Creation Date	Access	Owner	Status	Version
2017-11-06	test1	2017-07-19	Read/Write	Martin Ivarsson	Work	(2)

**Attributes**

**Description**

Automatically adjusts vehicle speed to maintain a driver-selected distance from the vehicle ahead in the same lane. It then returns to the set speed when traffic clears.



# Overview of a requirement, you can view the requirement's default attributes, such as ID and Rationale in the example

B SystemWeaver demo(1)

B Baseline Architecture - Default Edit

Name	Status	Version
B SystemWeaver demo	Work	(1)
SystemWeaver demo	Work	(1)
Function requirements	Work	(1)
Adaptive cruise control	Work	(1)
§ Adjust speed	Work	(3)
§ Emergency braking	Work	(1)
§ Maintain speed	Work	(1)
§ Test Req	Work	(1)
Constant cruise control	Work	(1)
Climate control	Work	(1)
Seat heating	Work	(1)
Washer wiper	Work	(1)
Low speed emergency braking	Work	(1)
Radio	Work	(1)
Design architecture	Work	(1)
Design functions	Work	(1)
AllocatedDesign	Work	(1)
Hardware architecture	Work	(1)
Safety analysis	Work	(1)
Test	Work	(1)

Overview

Last Changed	Last Changed By	Creation Date	Access	Owner	Stat
2015-11-12	Martin Ivarsson	2015-11-11	Read/Write	Martin Ivarsson	Wor

**Attributes**

Default

ID: CRU-3 v1

Rationale:

**Description**

When the cruising speed has been reached, the speed shall be maintained within 4%

Adaptive speed

Vehicle speed

Obstacle distance

# Function Specification Report

Variability demo database[sys60] - SystemWeaver Collaborative Environment

File Welcome Dashboard Items Projects

Back Forward Open item New item Copy Overview Attributes Find Versions Status New issue Add note Libraries Move items... Impact analysis Navigation Items Parts Filter on search result CM Issues Notes Complete status Manage libraries Functions Specification

**B SystemWeaver demo(1)**

**B Baseline Architecture - Default** Edit

Name	Status	Version
SystemWeaver demo	Work	(1)
SystemWeaver demo	Work	(1)
<b>Function requirements</b>	Work	(1)
Adaptive cruise control	Work	(1)
Adjust speed	Work	(1)
Emergency braking	Work	(1)
Maintain speed	Work	(1)
New Requirement	Work	(1)
Constant cruise control	Work	(1)
Climate control	Work	(1)
Seat heating	Work	(1)
Washer wiper	Work	(1)
Low speed emergency braking	Work	(1)
Radio	Work	(1)
Design architecture	Work	(1)
Design functions	Work	(1)
AllocatedDesign	Work	(1)
Hardware architecture	Work	(1)
Safety analysis	Work	(1)
Test	Work	(1)

Print... Print Preview... Functions Specification Baseline SystemWeaver demo (1) Document type Owner Last changed Status SystemWeaver Id Specification Martin Ivarsson 2015-09-14 14:55 by Database administrator x04000000000000000000000000000000

Function requirements

## 1 Function requirements

### 1.1 Adaptive cruise control



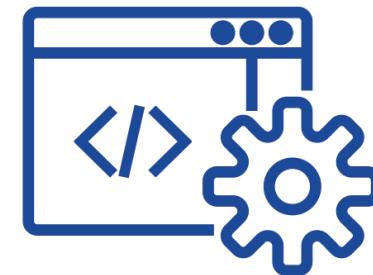
A red arrow points from the 'Functions Specification' button in the top menu bar to the 'Function requirements' section in the main content area.

# Impact Analysis View - Assess Impact of a Requirement Change

The screenshot shows the SystemWeaver application interface. On the left is a navigation bar with File, Welcome, Dashboard, Items, Projects, and various search and management tools. Below it is a tree view of the Baseline Architecture under SystemWeaver - demonstration(1). A red box highlights the 'Maintain speed' requirement under the Adaptive cruise control function requirements. The main area displays the Impact analysis view for this requirement. The title bar says 'Impact analysis' with a red arrow pointing to it. The view shows a 'Function' section with 'Maintain speed' and a 'Design' section with 'Adaptive cruise'. Arrows indicate dependencies between requirements like 'CC Regulate speed' and 'ACC Regulate adaptive speed', and between design elements like 'Test jerk' and 'Cruise speed under current speed'. A red box also highlights the 'Test' section at the top right of the impact analysis view.

When a function requirement is selected, it is possible to use the **Impact Analysis** view to see the effects of the requirement in the system. This can, for example, be an efficient way of assessing the impact a change of that requirement would have on design requirements and test specifications.

# Questions?



תודה

Dankie Gracias

Спасибо شکرًا

Merci Takk

Köszönjük Terima kasih

Grazie Dziękujemy Děkujeme

Ďakujeme Vielen Dank Paldies

Kiitos Täname teid 谢谢

**Thank You** Tak

感謝您 Obrigado Teşekkür Ederiz

Σας Ευχαριστούμ 감사합니다

Bedankt Děkujeme vám

ありがとうございます

Tack