ecu.test

# Accelerating Virtual Validation using FMI-based Test Automation: From Component Testing to Digital Twins

Matthias Berthold

tracetronic GmbH (Germany)

# Introduction

- **tracetronic**
  - Founded 2004 in Dresden, Germany
    - (Hangzhou is twin town/sister city since 2009!)
  - ~500 employees (incl. neocx as a VW joint venture)
  - https://www.tracetronic.com
  - Products: ecu.test (test automation), one:cx (CI/CD collaboration platform), …
  - Engineering, support, consulting (automotive, finance, agriculture…)
  - Offices Germany: Dresden (HQ), Munich, Ingolstadt, Stuttgart and Hamburg
  - Offices international: China (Shanghai, Hefei), Korea (Seoul), Japan (Yokohama) and USA (Detroit)





- **tracetronic China**
  - Founded 2023
  - Offices in Hefei and Shanghai
  - Engineering and on-site support

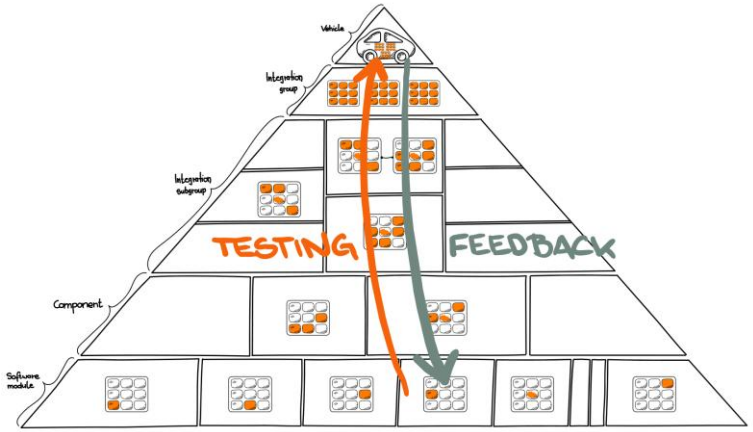




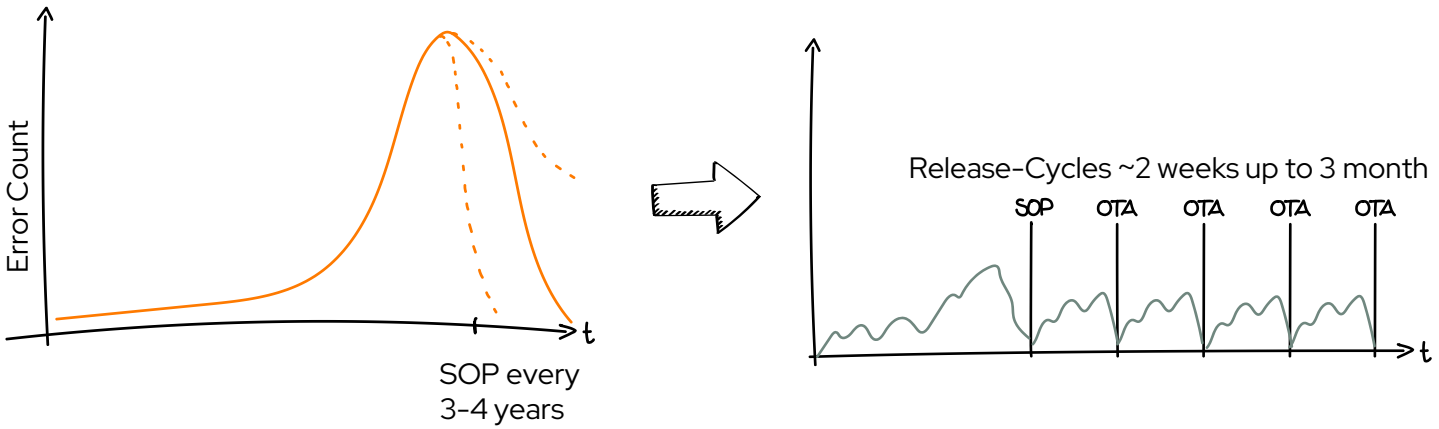
- **Matthias Berthold (Dipl.-Inf.)**
  - Joined tracetronic 2014
  - Product Owner ecu.test
  - Member of FMI Design Group
  - matthias.berthold@tracetronic.de
  - https://www.linkedin.com/in/matthias-berthold-dresden

# Motivation

- **Title is already the solution – but what's the problem?**
  - More and more software and shorter development cycles → early & continuous test and integration required ("shift left")
  - HIL systems and prototypes often available too late or are too expensive and inflexible
  - **Goal:** Working integration and tests are needed for efficient software development long before hardware exists



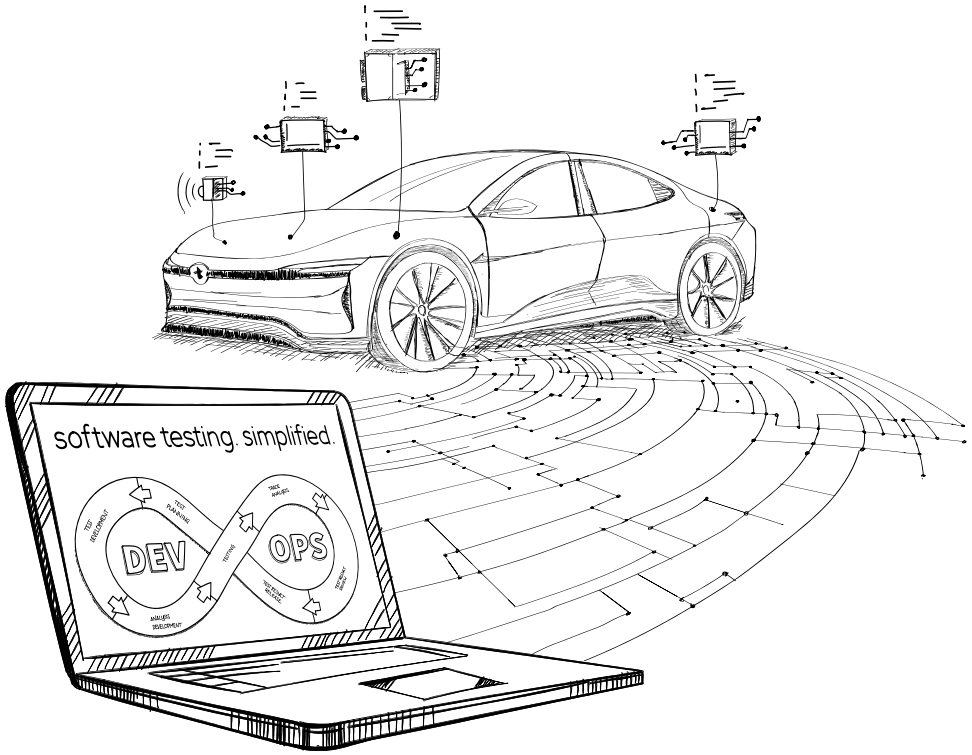

- **Problem is not related to FMI, but FMI is a possible solution!**



# Agenda

- **Only 20 min → high level overview**
  - Please feel free to chat during conference
  - Or contact me or my colleagues any time afterwards
  - Download link for slides on last page

- **1st part:**
  - Usage of FMI in testing
  - FMI, SSP, FMI-LS-BUS/XCP
- **2nd part:**
  - Insights, hints, lessons learned from the last years…
  - HIL, AI, manual testing



# Usage of FMI in testing

# The vECU Levels

- **vECU** = virtual electronic control unit (automotive) = model = FMU
- **prostep ivip White Paper** (PDF download):
  - *Smart Systems Engineering - Requirements for the Standardization of Virtual Electronic Control Units (V-ECUs)*
  - Good paper, especially for 2020 (and the time before during creation)
  - Helps for common understanding what vECUs are and how to use it and why → recommended reading
  - Each level can be interpreted as a digital twin → each with a different focus (testing, performance, effort in general)
- **For test automation and FMI:**
  - These levels are **not** important → FMI provides **one** interface for all levels (with FMI LS and SSP)
  - FMI can run on Win, Linux, Mac and x86, Arm… → Tools should support this (import and export)
- **More important: abstraction and consistency**
  - Test automation users should focus on testing
  - Component, integration and system testing (V model)
  - Not on technology, standards, levels…

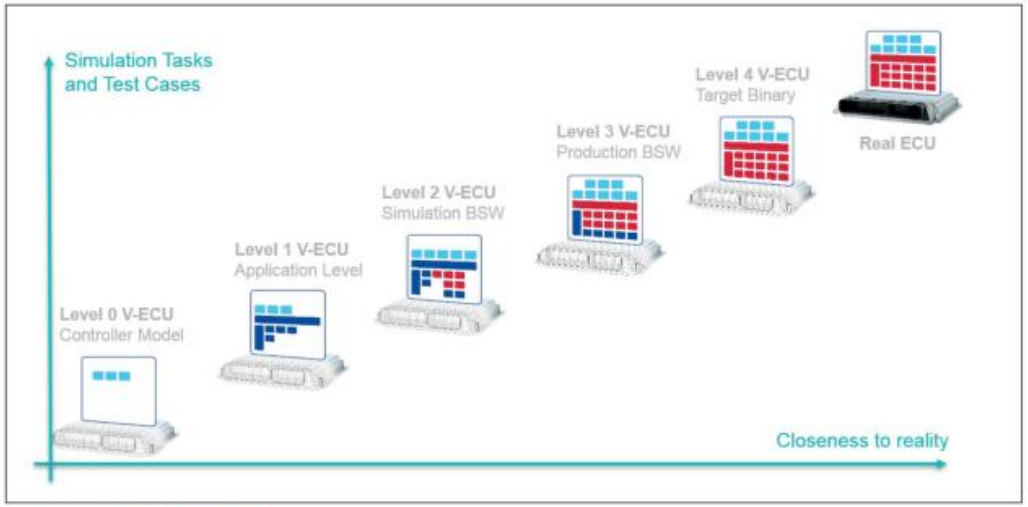
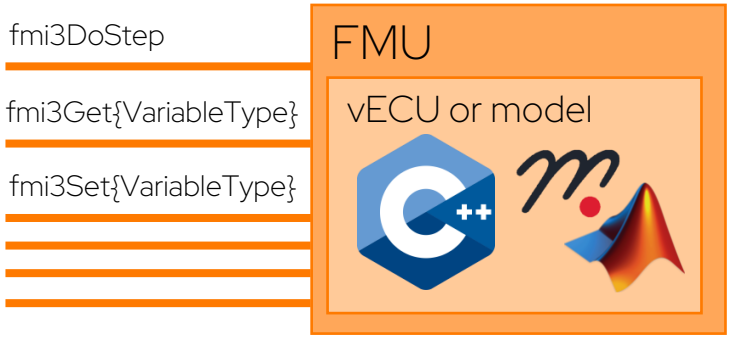



Figure 8 Proposed V-ECU levels

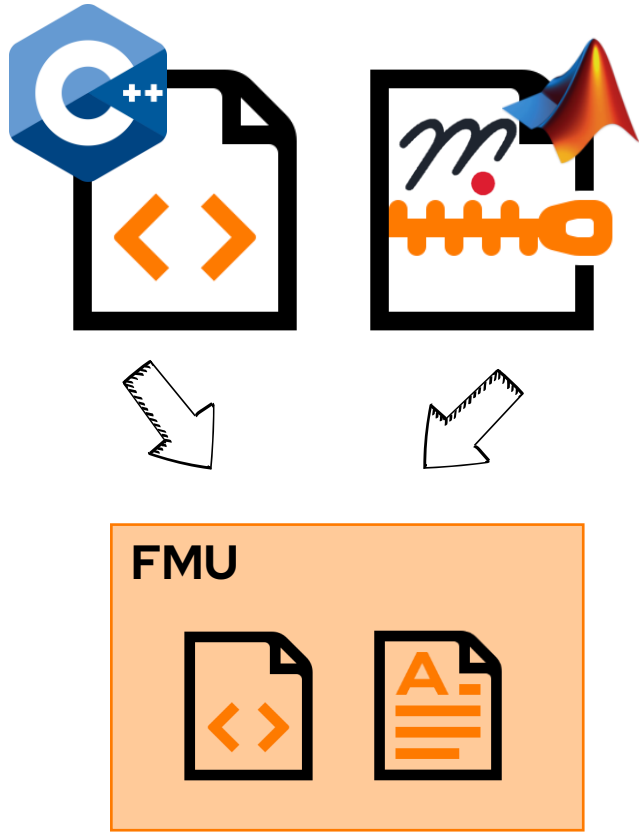
Source: Smart Systems Engineering - Requirements for the Standardization of Virtual Electronic Control Units (V-ECUs) (prostep ivip White Paper , P. 7)




Interfaces of an FMU → suitable for all vECU levels

# Component Testing Using FMI

- **1st step in SIL: One FMU for component test**
  - Easy start, quite common nowadays (FMI 3, CS)
  - Complete vECU or part of it (Should be executable by itself, no environmental models necessary)
  - Never tried it out? → Just use FMPy (open source) and some available examples or AI to create your first experience!
- **Starting point: tested code from a developer**
  - Source testing: natively using google test etc. or e.g. a Simulink model with Simulink test
  - Why not just a unit test? → white-box test becomes black-box test with a defined interface (FMI)
    - That interface is later reused by other components → foundation for every next step, no big bang in integration phase
    - Isolated FMU as an executable artifact, tested for later stages (regardless if called "vECU" or not)



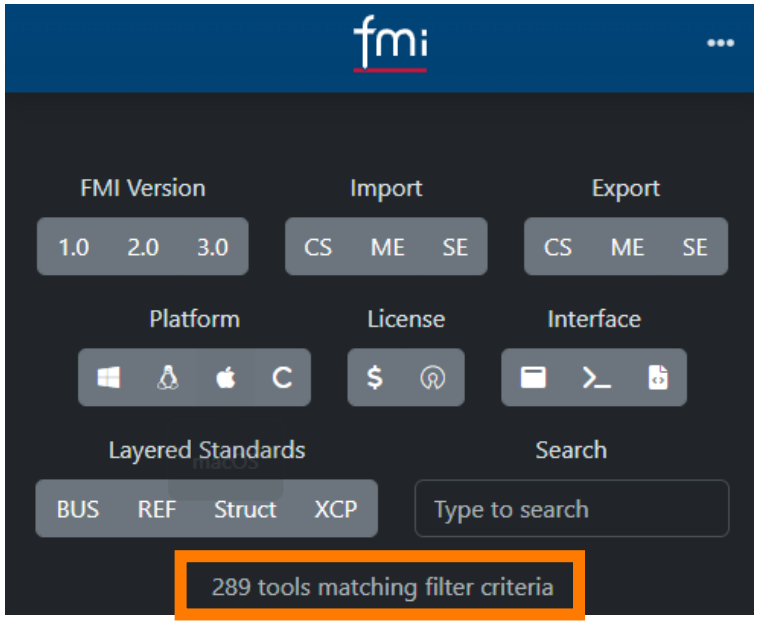

Creating an FMU incl. model (dll, so, ..) and model description (XML)

# Component Testing Using FMI

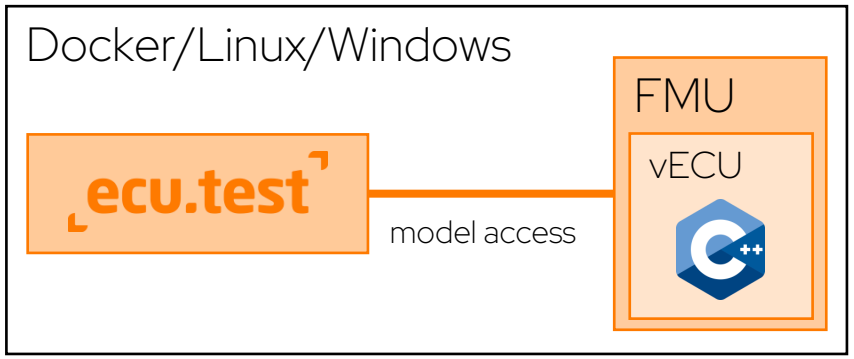
- Creating an FMU:
  - Build (“export”) with used tool (> 150 tools)
  - use custom build scripts and glue code for custom code (e.g. C/C++)
- Using a single FMU :
  - Execution (“import”) with over 200 tools

→ **flexible for individual needs** → in contrast to a single tool or tool landscape → no vendor lock-in

- **Test focus: standardized interfaces**
  - “In the loop” and “open loop” (replay)
  - Don’t repeat white box testing




Source: https://fmi-standard.org/tools/

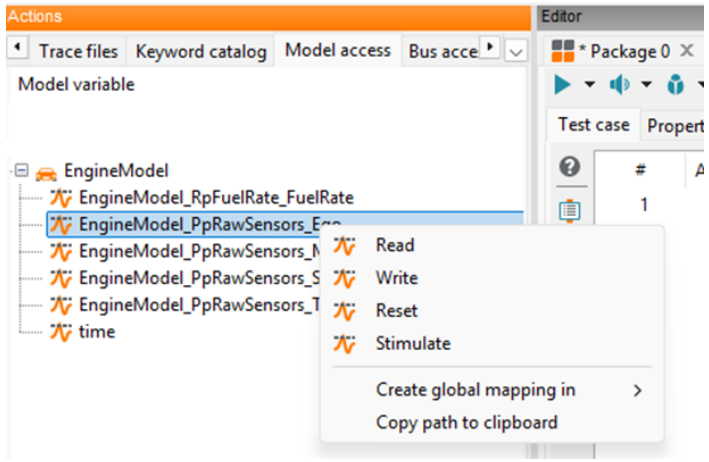



ecu.test: Example of the usage of an FMU

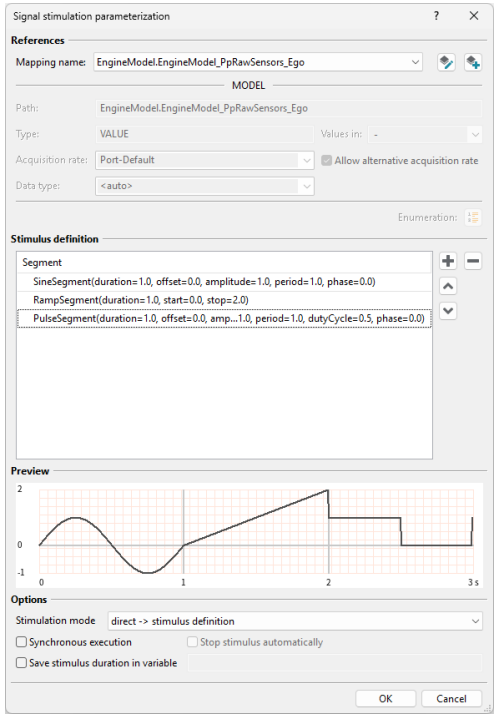
# Component Testing Using FMI

Using FMI with **abstraction and consistency!**

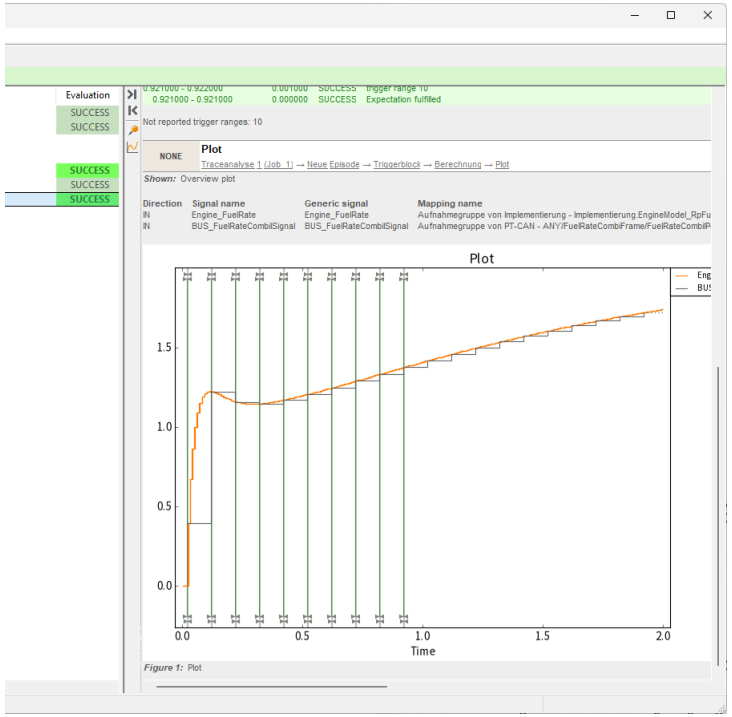
- Easy model access for interface tests: Read, write, stimulate (individual formats, ASAM STI/STZ or recorded files)
- Trace analysis and reporting
- GUI or Python Interface (e.g. for VS Code)
- No FMI visible but thanks to FMI:
  - Test a component
  - Create more complex systems
  - Exchange models




ecu.test: Access model variables of a single FMU of an engine model for in-the-loop and open-loop test case creation




ecu.test: Stimulation of an FMU variable with an individual curve incl. preview




ecu.test: Trace analysis and reporting of recorded FMU variables after a test execution

# Integration Testing Using SSP

- SSP → Standard to combine 2 or more FMUs (signal based) in testing
  - environmental models (most vECUs needs complex (realistic) models to work properly, e.g. residual bus, stimulation not sufficient) → testing still a Component
  - Interaction between models/vECUs → testing the integration of the several parts
- Not that much tool support → https://ssp-standard.org/tools/
  - 17 in total (15 import, 12 export)
  - Workaround: "Container-FMU"
    - (or other proprietary or individual solutions to combine several FMUs into a single FMU)
    - but: inflexible (e.g. rebuilds for different combinations) and limited (e.g. signal exposure or Bus)
- SSP standards supports much more:
  - Parameterization
  - Layered Standard Traceability → very useful for process definition, additional simulation information and OEM supplier integration

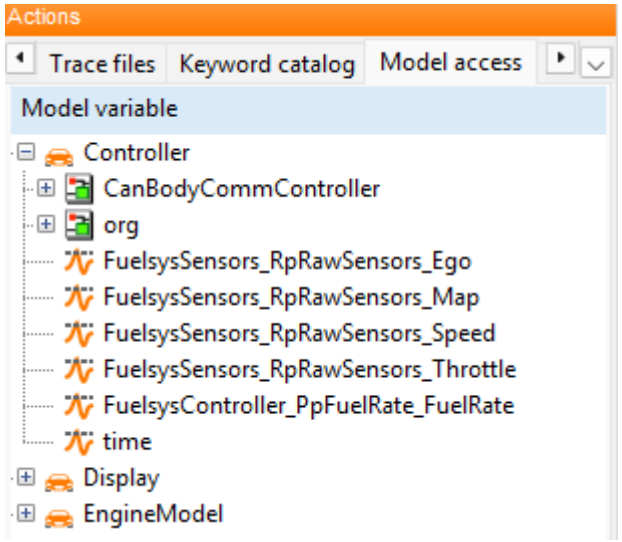



**Tools that support SSP**

| Name | Vendor | Description | License | Import | Export |
|---|---|---|---|---|---|
| SYNECT Model Management | dSPACE | SSP Add-On for SYNECT | commercial | 1.0 | 1.0 |
| OMSimulator | OpenModelica | FMI & TLM based simulator | free | 1.0 | 1.0 |
| Model.CONNECT | AVL | A model integration and co-simulation platform connecting virtual and real components | commercial | 1.0 | 1.0 |
| FMI Bench | PMSF | Flexible FMI integration and customization workbench | commercial | 1.0/2.0 | 1.0/2.0 |
| easySSP | eXXcellent solutions | orchideo \| easySSP is a platform for credible simulations, enabling traceable architecture design, execution, and efficient workflows based on open standards like SSP and FMI. | free + commercial | 1.0/2.0 | 1.0/2.0 |
| Simcenter System Architect | Siemens | Simcenter System Architect is a platform to support the process of architecture driven development of simulation models via model integration with execution/cosimulation capabilities | commercial | 1.0 | 1.0 |
| Simcenter Studio | Siemens | Simcenter Studio is an application for generating and evaluating system architectures during the early concept phase | commercial | 1.0 | 1.0 |
| Dymola | Dassault Systèmes | Dymola is a model development and simulation tool for Modelica and FMI. | commercial | 1.0/2.0 | 1.0/2.0 |
| CATIA Magic | Dassault Systèmes | CATIA Magic is a model-based system engineering and system of systems engineering solution and offering. | commercial | no | 1.0 |
| OpenMCx | Eclipse Foundation | OpenMCx is an extendable, tool-neutral co-simulation framework based on Modelica Association Standards. | free | 1.0 | no |
| Simulator Suite | TLK-Thermo | Simulator Suite enables our customers to efficiently evaluate models based on the Functional Mock-up Interface (FMI) standard or compiled Dymola code. | commercial | 1.0 | no |
| FMIGo! | Tjoppen | FMIGo! is a set of Free Software tools aimed at the Functional Mockup Interface (FMI) standard. | free | 1.0 | no |
| Ecos | NTNU Ålesund | Ecos (Easy co-simulation) is a fast, efficient and very easy to use co-simulation engine written in modern C++. | free | 1.0 | 1.0 |
| SysAI.simulate | SysAI | Simulation environment providing automated FMI/SSP simulation model templating from SysMLV2 and other system architectures. | commercial | no | 1.0 |
| FMU Manipulation Toolbox | Groupe Renault | FMU Manipulation Toolbox is a python library and applications which modify or create Functional Mock-up Units (FMU) without recompilation. | free | 1.0 | no |
| Scilab/Xcos | Dassault Systèmes S.E. | Provided with Scilab, Xcos is a graphical editor to design hybrid dynamical systems models (with SSP and FMU support). | free | 2.0 | 2.0 |
| ecu.test | tracetronic | ecu.test is an intuitive test tool for creating test cases for automotive software and other domains in any development phase and for their automatic execution, supporting a variety of test execution and test process tools. | commercial | 1.0/2.0 | no |

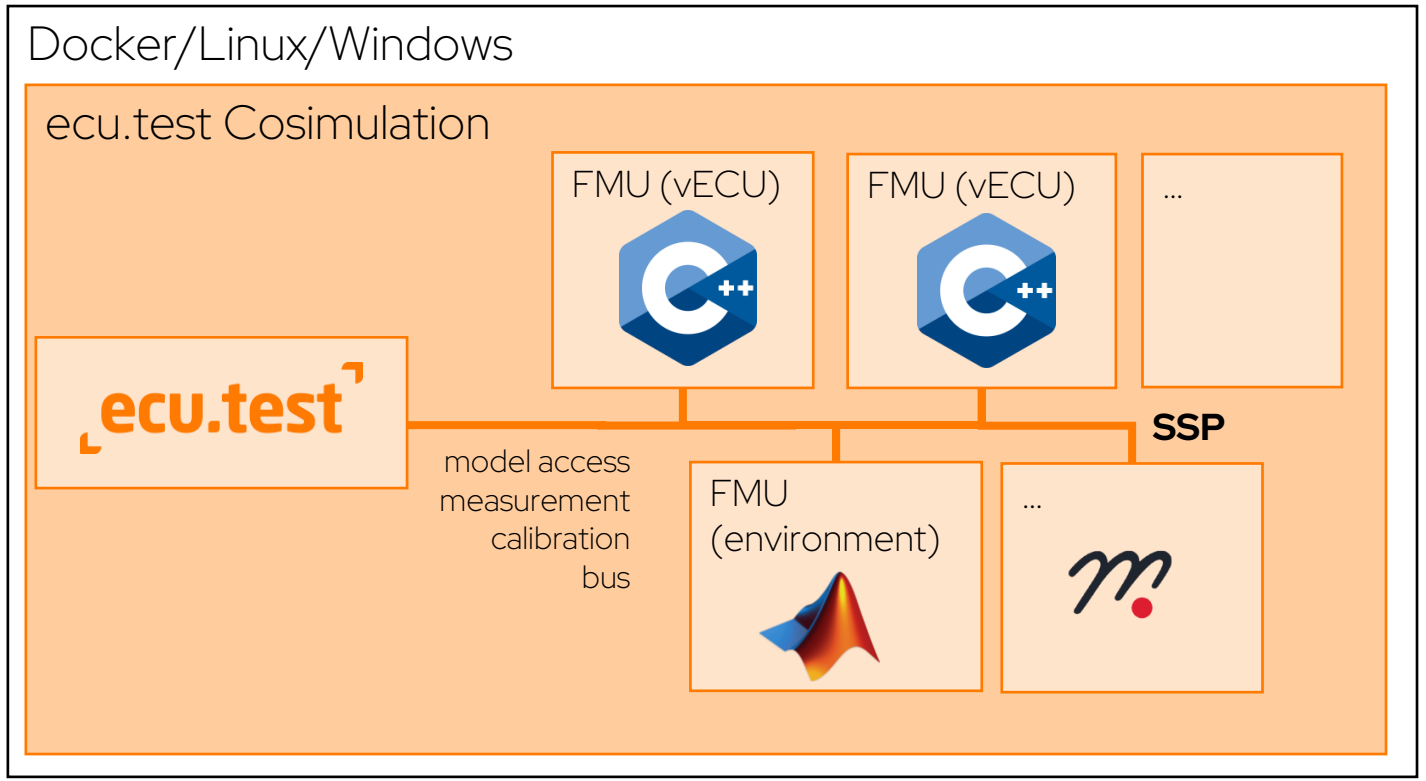
Source: https://ssp-standard.org/ & https://ssp-standard.org/tools/

# Integration Testing Using SSP




ecu.test: Access to model variables ("Fuelsys…") of several FMUs ("Controller", "Display", "EngineModel") combined with SSP to a complex system

- **Thanks to SSP** a realistic simulation of more than one component can be realized tool independent!
- Used with previously tested FMUs (component → integration)



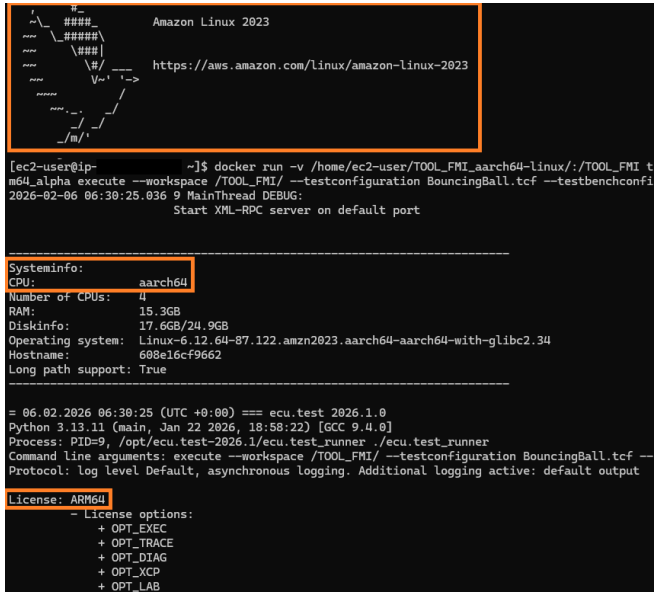

Co-Simulation algorithm for SSP inside ecu.test

# Advanced (Automotive) Testing Using Layered Standards

- **Two major Layered Standards:**
  - FMI-LS-BUS for Network Communication → bus based integration and communication (in contrast to SSP which is signal based)
  - FMI-LS-XCP for measurement & calibration with XCP → e.g. A2L based vECU memory access

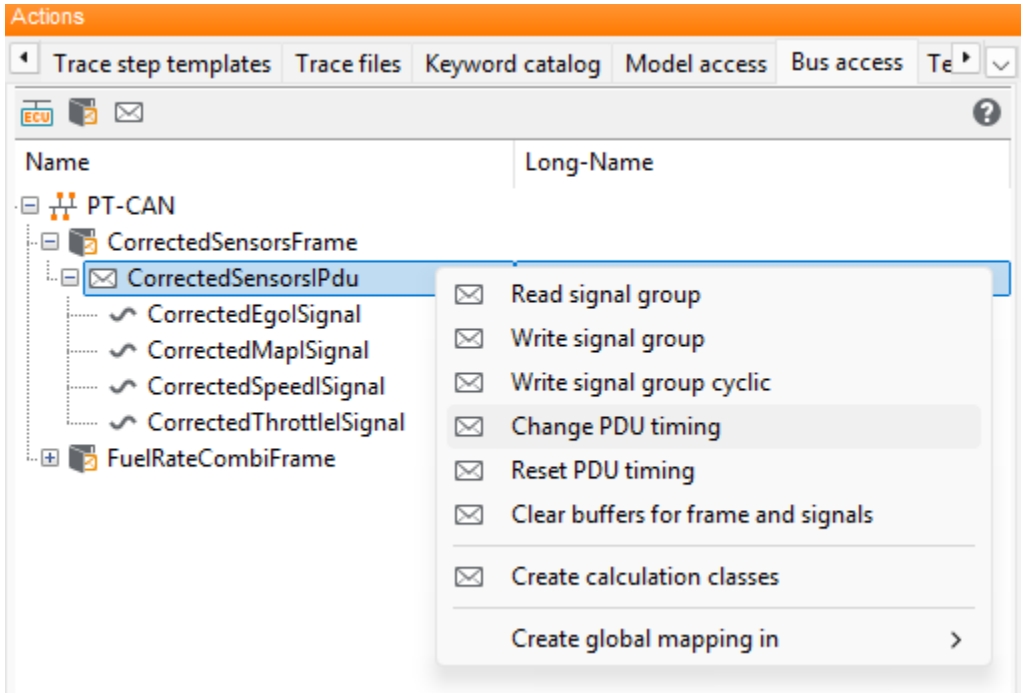
- Not that many tools (9 for XCP, 11 for BUS, 5 for both) but rising during design and implementation phase (some major tools already have support)
  - Only CAN is released, others are in development (will be released soon)

- **vECU Level 2 & 3: a "more realistic" digital twin**
  - Integrate basic software and use vECU or several vECUs incl. Bus and XCP → (complete) SIL system simulation → realistic automotive behavior
  - Necessary (or at least helpful) for later integration in HIL/VIL → Bus systems and XCP available and required
- **vECU Level 4:** Component or system executed with target code
  - But it is also the one with the most efforts and highest performance requirements → check if really necessary or helpful
  - E.g. Arm: AWS (Graviton), Azure (Cobalt), Alibaba Cloud (Yitian)…

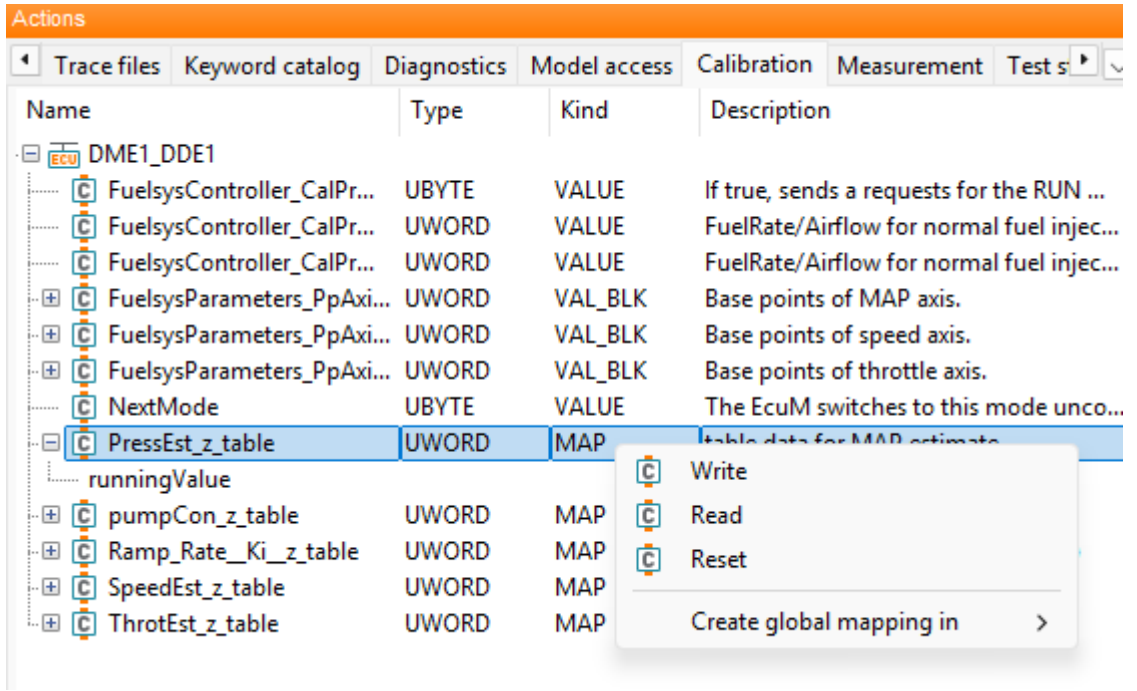



Running ecu.test on an Arm system (here: AWS Graviton) to execute Arm compiled code without slow emulation

# Advanced (Automotive) Testing Using Layered Standards




ecu.test: Access to bus variables (based on DBC) for bus communication testing




ecu.test: XCP access (based on A2L) for measurement and calibration usage

Source: "ASAM Demo" from FMI Design Group (see also YouTube)

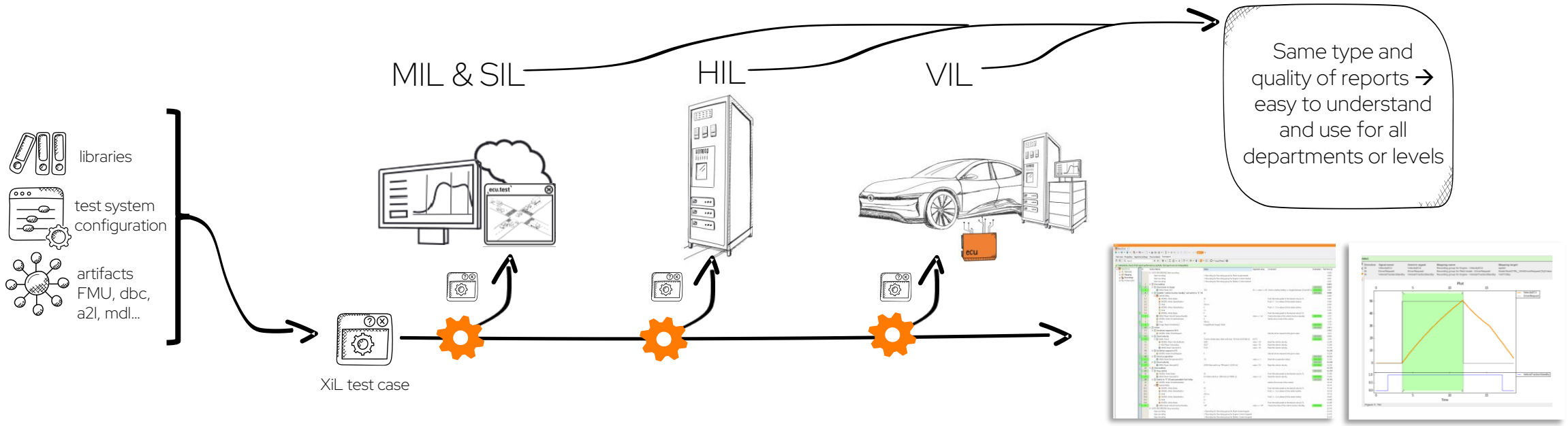
## Thanks to FMI-LS-BUS/XCP:

- Test components or systems of components with a single, standardized interface (vECU level 2 to 4)
- Outcome: Validation & Verification of the "digital twin" before HIL/VIL (+ reusage of test cases)

# Insights, Hints, Lessons Learned…
# About FMI and Beyond

# Reuse tests (and models) with HIL or VIL systems

- Challenge: Migration from SIL (FMI) to HIL/VIL?
- FMI, SSP and FMI LS BUS/XCP were the base to create tests and models → Reuse it!
- Reason for abstraction during SIL:
  - Exchange configurations and mappings → rerun test cases, compare results, improve tests and/or models
  - Used (environmental) models (FMUs) can be reused in HIL environments
  - ASAM XIL in HIL comparable technology abstraction like FMI in SIL
- **Important for you:** The used test automation tool should only change configs and mappings!

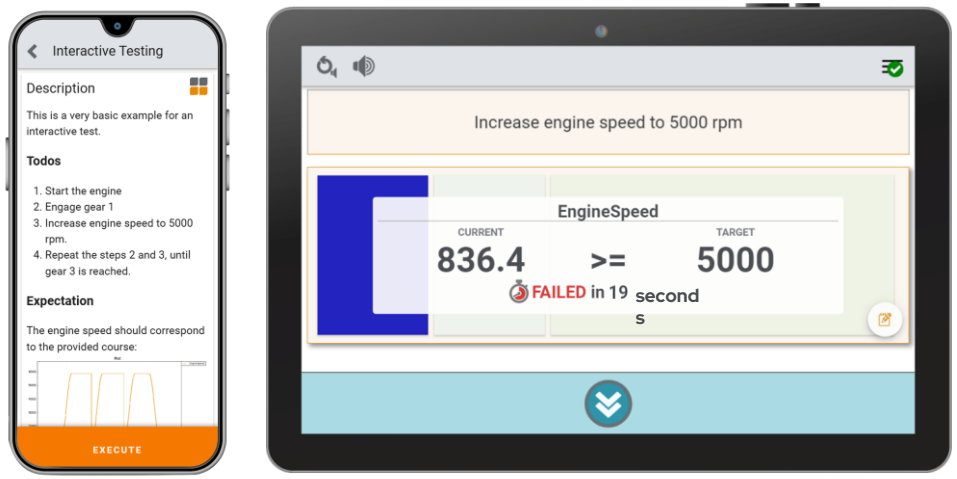


# Manual & AI-driven exploratory testing starts in SIL

- Next to test automation manual (or exploratory) testing may help to ensure software quality.
- Starting in SIL (with FMUs) with realtime execution → can be transferred to HIL and VIL
- Today "manual testing" also means that AI agents executes exploratory testing without test spec or requirement (without realtime)
  - This must be persisted as a test case afterwards (regression)



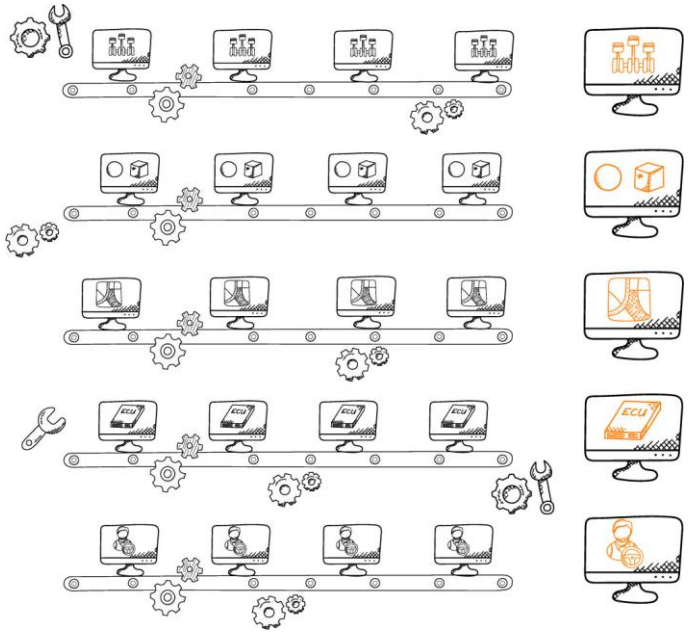

ecu.test lab: Creating and using dashboards for monitoring, controlling and visualization

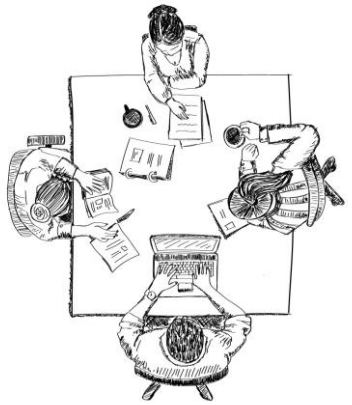



ecu.test drive: Executing tests with user interaction and feedback

# Tool Flexibility ≠ Tool Chaos: One Toolchain for All XIL Stages

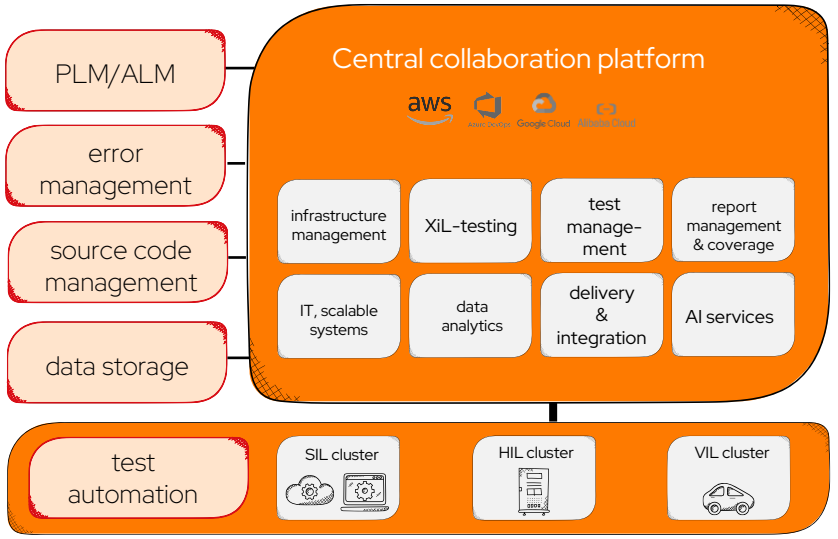
- Tool flexibility with hundreds of tools in SIL is great, but…
- Challenge: How to use several tools or toolchains with different reporting, test environments, traceability usage, operating systems…?
  - Or from a department POV: How to collaborate with your colleagues?
  - And what about later stages like HIL?
- Essential parts for efficient collaboration: Identical execution and reporting with complete traceability
- **Important for you:** Try to find tools or toolchains which fits for all (most) stages and departments (in best case: one)
  - Maintainability, usability and user acceptance is also important (next to license costs)



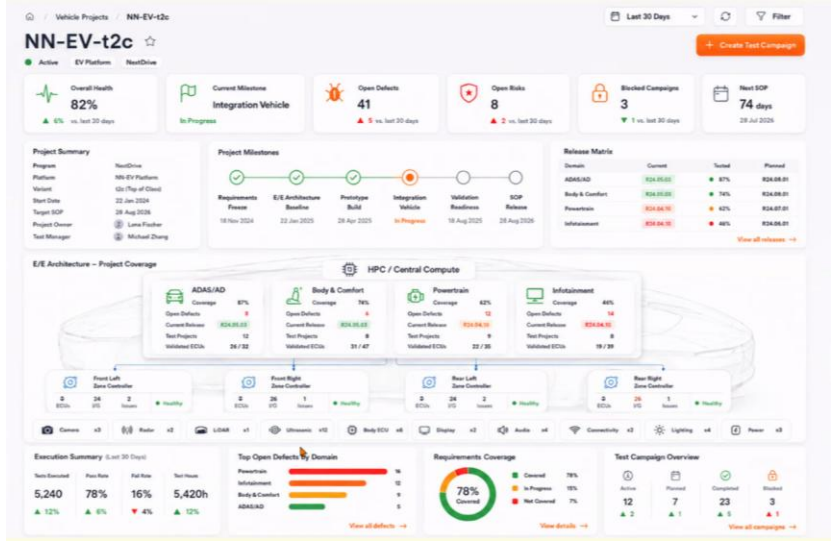



# Central Data Management

- SIL, HIL and VIL testing creates a lot of data
- Central data management to keep overview
- No silo thinking or workarounds for data exchange like Excel, Jira, file system…
- Reuse data:
  - Improve FMUs after test execution in HIL or VIL
  - Create virtual HIL as FMUs based on HIL/VIL data
  - Only execute small part of SIL tests after code change (change impact analysis)

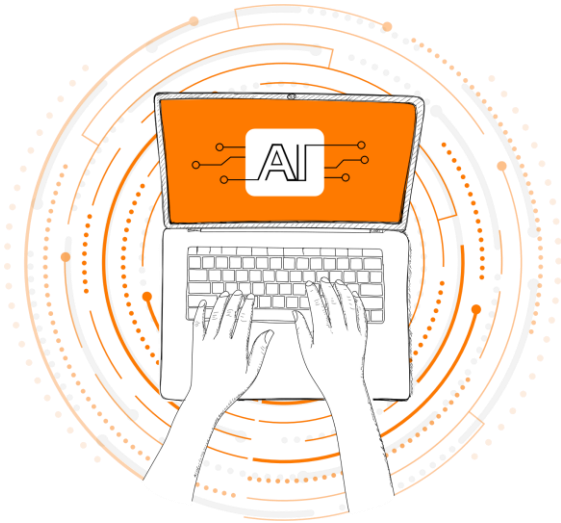



one:cx: Data storage as one essential part of effective collaboration



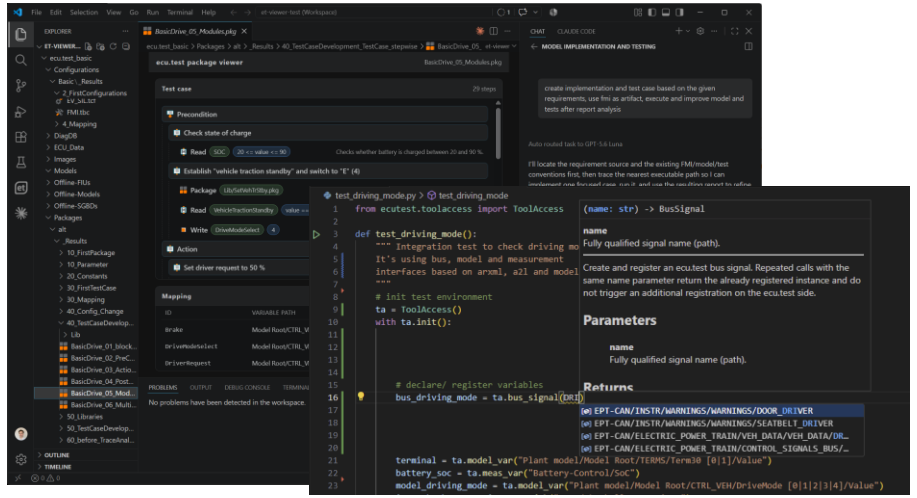

one:cx: Customizable dashboards for data visualization

# Usage of AI in Context of FMI




- FMI, FMI-LS and SSP are open standards:
  - AI can use this very well and generate code, headers, build scripts etc. in a very high quality!
  - Generate models: environment, residual bus, vECU dummies/mocks…
  - Generate interfaces, test and execute models, create SSPs or interactive editors for SSP creation…
  - Helps to integrate local or proprietary solutions in that open environment → smoother migration
  - **FMI is stable and solid foundation in this vibe coding era!**

- Model Generation with, e.g., Machine Learning
  - Suitable and fast models from already available environmental data (see: central reporting system → virtual HIL)
  - e.g. ONNX2FMU or PythonFMU with PyTorch
  - It is not always necessary to have most accurate models (which are slow and have high efforts to create)

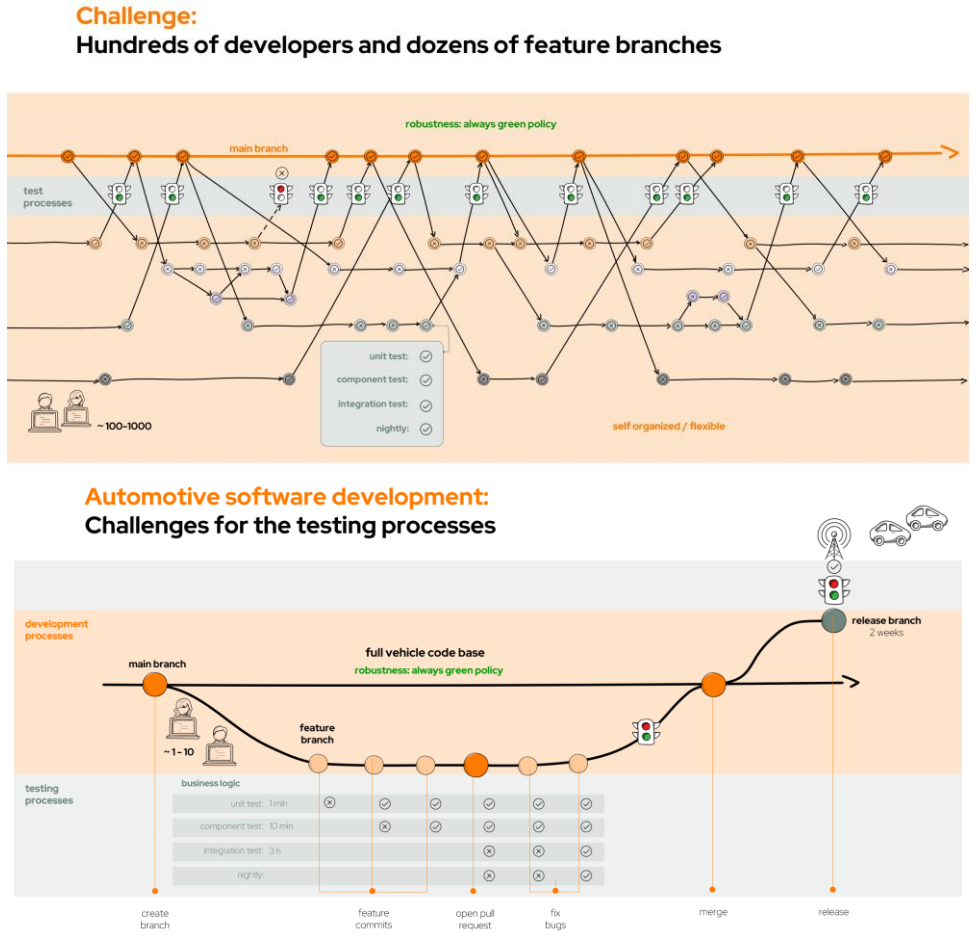


ecu.test in agent based workflows (e.g. VS Code) at AWS Hackathon: Requirement → create implementation → export as FMU → use FMU in ecu.test → create test cases (Python or GUI) → execution → analyze reports → improve model (or tests or requirements)

# Journey to Fully Automated DevOps Using FMI

- **Remember the goal: Shift left to fully automated testing!**
  - Release often, without manual efforts
  - Find defects in early stages using FMI

- Starting in SIL? Currently only HIL or VIL tests?
  - Create your first FMU (e.g. for a component)
- Integration problems in HIL?
  - Integrate it using SSP and/or LS Bus
- Too much (expensive) simulation tools?
  - Try to reduce, focus on speed and maintenance
- Limited HIL capacity?
  - Improve SIL models or create virtual HIL environments




ecu.test and one:cx: Improve software development using organized processes based on FMI and related artifacts

# Public Resources

- Active, open environment → use it!
  - From beginners to special cases in implementations
- Official FMI Implementers' Guide
  - FMI 3.0 Implementers' Guide
  - FMI LS BUS Implementers' Guide
- JSAE guide with introduction and lot of tool related examples
  - FMI LS BUS Implementers' Guide (Ver. 2.0.2)
  - FMI Application Guide (Appendix, Ver. 2.0.2)
- Complete code example for FMI for all SIL Levels (incl. SIL Kit)
  - SIL ECU Virtualization – Level 1 vECU to FMU Co-Simulation (GitHub)
- „Learn Modelica & FMI" by Dr. Clément Coïc
  - FMI 3.0 & FMPy Cheatsheet
  - Learn Modelica & FMI (also on LinkedIn)
- Stack Overflow, LinkedIn, Modelica Newsletter…



# Summary and Outlook

# Summary

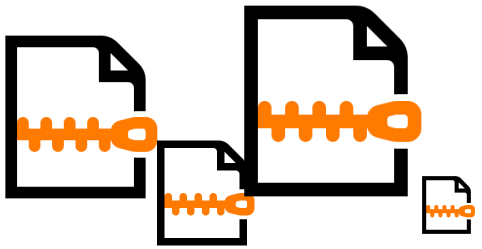
- **You can't be wrong when using FMI!**
  - FMI, FMI-LS and SSP offer a suitable and open system to test in SIL without any proprietary dependencies.
  - All major modeling and automotive tool vendors support FMI
    → no need to use proprietary formats anymore
  - At last FMI conference someone said *"FMI is like a marketplace"*
    → because of the free choice every tool must offer best possible support (or won't be used anymore in long term).



- **With proper tooling testing from first model or vECU as FMU over HIL to vehicle is possible.**
  - Complete V-model validation and verification → as much reuse as possible (FMUs, tests)
  - Tool interchangeability thanks to FMI (and a lot of cross testing)



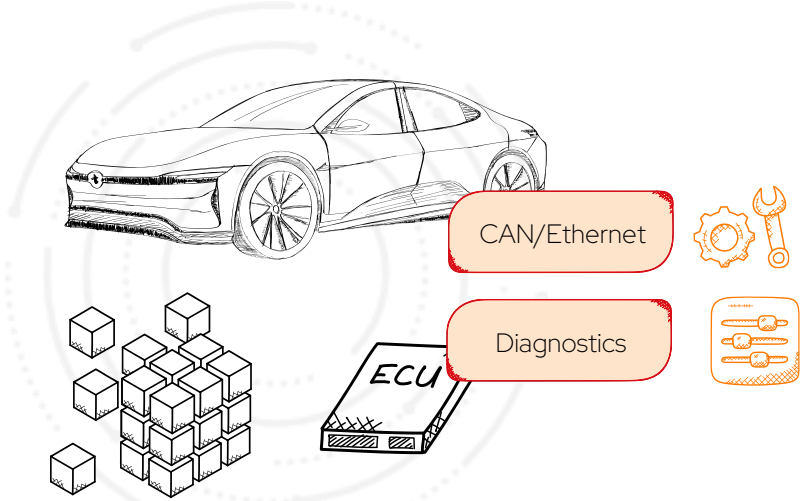
- **There is no THE digital twin and no THE process for SIL testing across all vECU levels.**
  - Act on the integration level or vECU level which fits best for you.
  - Less is better than nothing → improve over time
  - Check performance and complexity

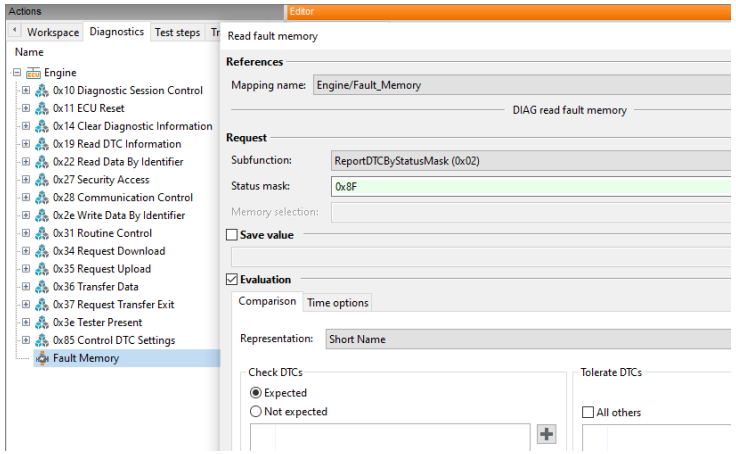


# Outlook 1/2

- More LS Bus (especially Ethernet):
  - More cross testing for LS Bus in general
    - Cross testing in plain FMI well established
    - FMI Design Group focus on LS Bus right now
  - Diagnostics: Adoption as layer above Bus for realistic system tests
  - Differentiation and/or limitations regarding Vector SIL Kit
  - More adoption from users → it's your turn now!

- FMI-LS-STRUCT for structured data like vectors and maps: Not final yet, first tools have to support it soon (incl. ecu.test)




ecu.test: Access to diagnostics like fault memory or diagnostic services (e.g. based on UDS)

# Outlook 2/2

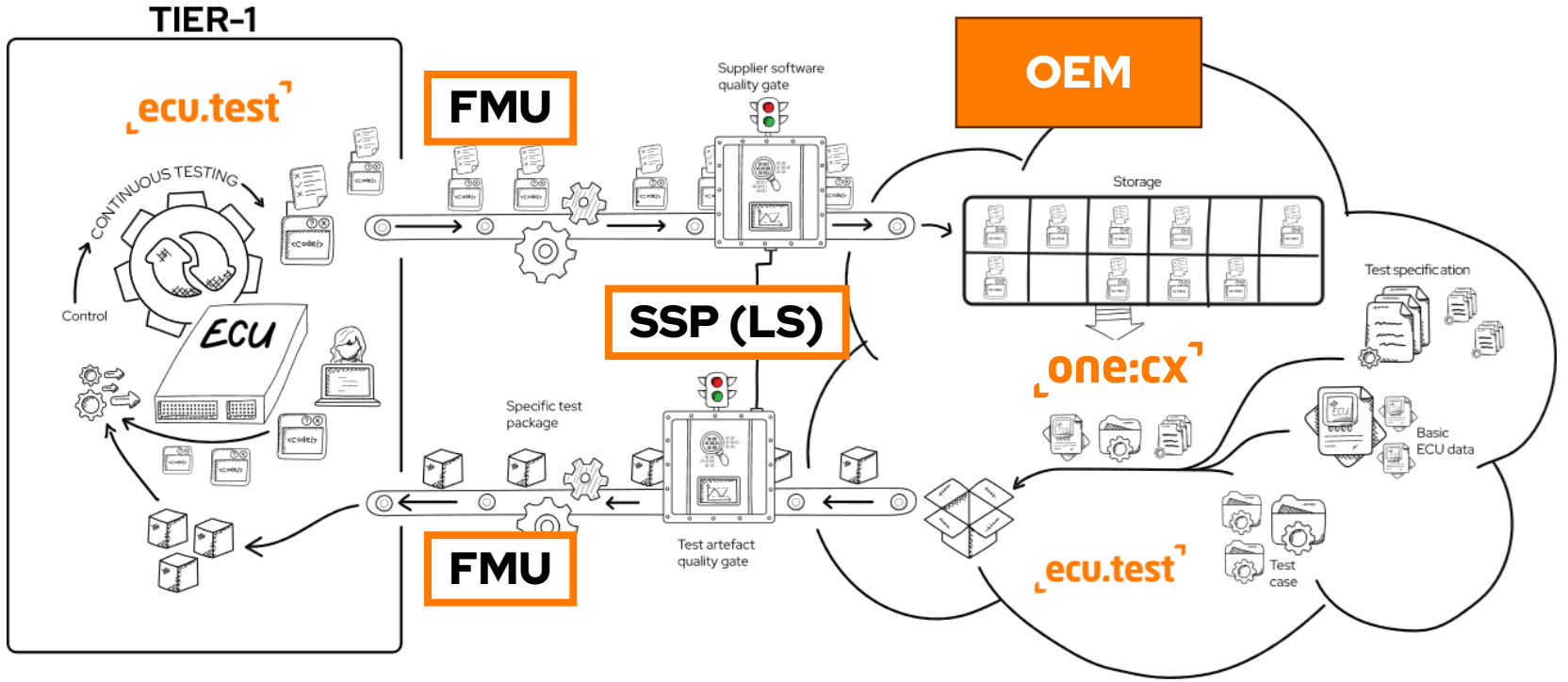
SSP incl. Layered Standard Traceability could be a solution for
**OEM and Tier-1 cross collaboration**:

- Exchange from models (FMUs) and systems of models (SSP)
- Process definitions (LS)
- Test specifications and reports (LS)
- …

**Same idea as FMI:**

- Everyone uses local tooling which fits best.
- Artefact exchange (supplier integration) with standardized artefacts.

→ not every OEM and Tier-1 tool chain and artefact combination has to be maintained (n*m combinations).




one:cx and ecu.test: Vision of SSP (incl. LS) based OEM and Tier-1 collaboration

# Thanks for Your Attention

- Any questions?

https://www.linkedin.com/in/matthias-berthold-dresden

matthias.berthold@tracetronic.de

Slide download