# Virtueller Test in der praktischen Anwendung

M.Sc. Felix Strauß, Dipl. Ing. (FH) Alexander Merkel, Nic Eckstein

Electronics and Virtual Testing Solutions
Bertrandt AG
Birkensee 1
71139 Ehningen
Felix.Strauss@bertrandt.com
Alexander.Merkel@bertrandt.com
Nic.Eckstein@bertrandt.com

**Abstract:** Der Trend zum Software-Defined-Vehicle (SDV) erhöht zunehmend die Komplexität der Software im Fahrzeug. Dies erschwert auf der einen Seite die gewohnten linearen Absicherungs- und Integrationsprozesse. Auf der anderen Seite lässt sich die Komplexitätszunahme kaum noch durch den Einsatz von hardware-basierten Testplattformen beherrschen. Eine weitere Hardware-Skalierung erscheint bei wachsender Komplexität unrealistisch sowie unwirtschaftlich. Speziell der virtuelle Test mittels virtueller Steuergeräte (vECUs) bietet hier Abhilfe. Hervorzuheben ist, dass virtueller Test weit mehr als klassisches Software-in-the-Loop-Testing (SiL) bedeutet. Vielmehr kann über eine Integration von Basissoftware und ggf. einer zusätzlichen Chip-Emulation ein sehr realitätsnahes Abbild von Steuergeräten hergestellt werden. Eine ganzheitliche Absicherung von Fahrzeugsoftware bieten diverse Integrationsstufen virtueller Steuergeräte. Jede Stufe bietet sowohl Chancen als auch entsprechende Limitierungen. Diese müssen entsprechend bekannt und bewertet sein, um einen digitalen Absicherungsprozess aufbauen zu können. Das Hauptaugenmerk muss dabei auf einem holistischen Integrationsprozess liegen.

## 1 Motivation

In den letzten Jahren befindet sich die Automobilindustrie in einem tiefgreifenden Wandel. Die Megatrends Connected, Autonomous, Shared und Electric (CASE) transformieren das Fahrzeug hinsichtlich seiner Funktionen zu einem grundlegend neuen Produkt. Kundenanforderungen, die durch die Smartphone-Welt und autonome Fahrfunktionen geprägt sind, treiben die fortschreitende Digitalisierung und eine signifikante Erhöhung der Softwarekomplexität voran. Darüber hinaus erfordert der Übergang zur Elektromobilität Neuentwicklungen der soft- und hardwareseitigen Komponenten des Antriebsstrangs und bedingt neue Fahrzeugplattformen.

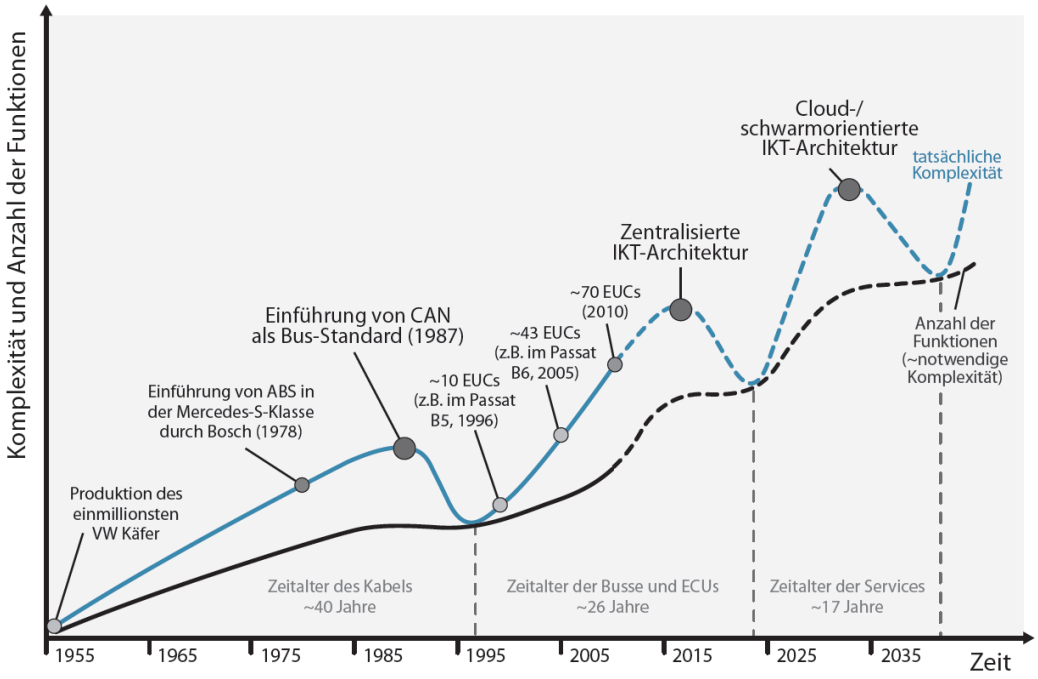


Abbildung 1: Komplexität und Anzahl Funktionen im Fahrzeug [FOR11]

Diese Trends führen zu einer Zunahme der Komplexität auf allen Ebenen (Abbildung 1): Die Steuergeräte werden leistungsfähiger und stärker vernetzt, die Softwarefunktionen umfangreicher. Die Anzahl der notwendigen Codezeilen nimmt exponentiell zu. Es ist zu beobachten, dass klassische Ansätze der E/E-Absicherung der notwendigen Geschwindigkeit und Menge nicht mehr Stand halten. Im Zuge dieser Entwicklungen manifestiert sich jedoch auch ein Umdenken in der E/E-Absicherung. Bertrandt leistet für zukünftige Absicherungsstrategien einen wertvollen Beitrag, indem der Fokus vor allem auf das Potential der virtuellen Absicherung gelenkt wird.

# 2 Stand der Technik

## 2.1 Virtualisierung

Unter Virtualisierung ist die Nachbildung eines Hard- oder Softwareobjekts durch Hilfe einer Abstraktionsschicht zu verstehen. In vielen Fällen spricht man auch von Hardwareabstraktion. Hiermit lassen sich virtuelle (d.h. nicht-physische) Geräte oder Dienste wie emulierte Hardware, Betriebssysteme oder Netzwerke erzeugen. Primäres Ziel der virtuellen Bereitstellung ist eine Unabhängigkeit von häufig sehr spezifischen und dadurch limitierten und teuren Zielgeräten zu erreichen. Vielmehr erfolgt die Bereitstellung auf weit verbreiteten Verbraucherendgräten, die in aller Regel aus einer x86-Prozessorarchitektur und einem Windows- oder Linux-Betriebssystem bestehen. Abbildung 2 zeigt diesen Prozess als Prinzip-Skizze.

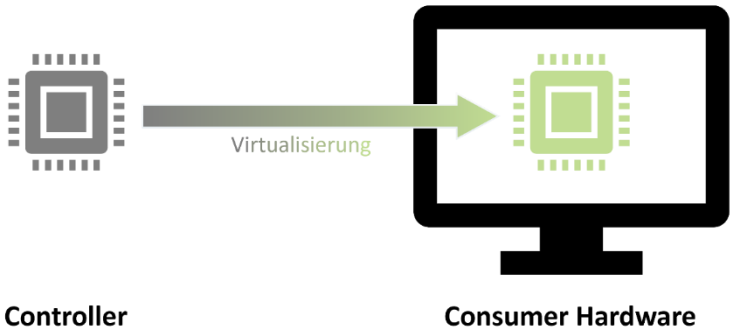


Abbildung 2: Prinzip-Skizze Virtualisierung

## 2.2 AUTOSAR Classic

Zur Herleitung der Steuergeräte-Virtualisierung, lohnt sich zunächst ein Blick in den AUTOSAR (AUTomotive Open System ARchitechture) Classic Standard. AUTOSAR ist eine Initiative verschiedener OEM und Zulieferer, die seit der Gründung im Jahr 2003 eine Referenzarchitektur für die Softwareentwicklung im Automotive-Bereich geschaffen hat [AUT22]. Diverse Ansätze der Abstraktion ermöglichen eine flexible Softwareentwicklung über Steuergerätegrenzen hinweg.

In erster Näherung definiert AUTOSAR eine Schichtarchitektur bestehend aus der *Applikationsschicht*, dem *Runtime Environment (RTE)* und der *Basissoftware (BSW)*. Die Applikationsschicht implementiert im Rahmen der Applikationssoftware (ASW) logische Fahrzeugzusammenhänge und Regelalgorithmen. Idealerweise ist sie hardwareunabhängig. Das *Runtime Environment*, eine Art Middleware, dient als Kommunikationsschicht zwischen einzelnen Softwarekomponenten (SWC) sowie zwischen *Applikationsschicht* und *Basissoftwareschicht*. Letztere stellt diverse Dienste zur Verfügung, die in drei Abstraktionsschichten unterteilt werden: *Service Layer*, *ECU Abstraction Layer* und *Microcontroller Abstraction Layer (MCAL)*.

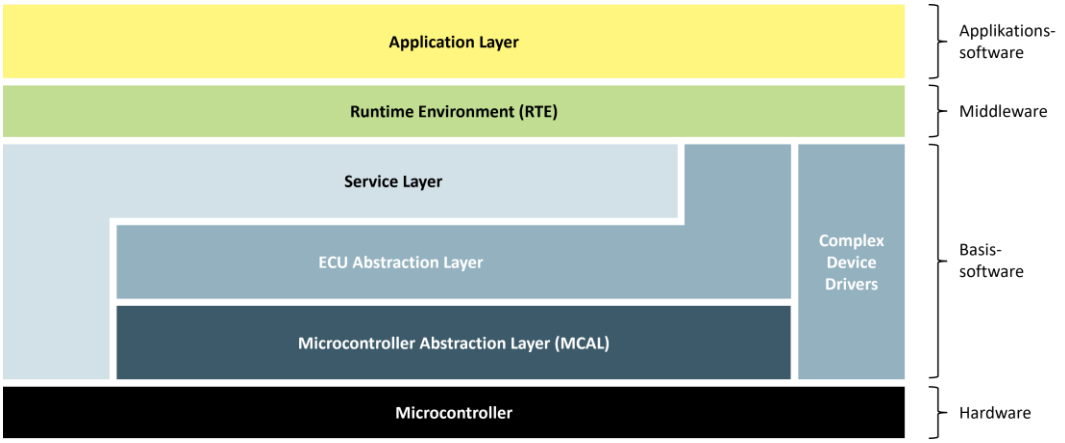


Abbildung 3: AUTOSAR Classic Architektur [AUT22]

Ein entscheidender Aspekt für die Steuergeräte-Virtualisierung ist die Trennung von möglichst hardwareunabhängigen Schichten und der darunterliegenden Hardware-Abstraktion, dem *MCAL*. Ein zweiter wichtiger Aspekt ist die strikte Abgrenzung von Funktionalitäten in Dienste und die hieraus resultierenden Schnittstellendefinitionen. Entlang dieser Schnittstellen können Funktionsblöcke gut abgegrenzt und logisch „geschnitten“ werden. Eine Ausnahme im Standard stellen die sogenannten *Complex Device Drivers (CDD)* dar. Diese ermöglichen die Vermischung der entkoppelten Schichten und stellen ein teilweise schwer zu überwindendes Hindernis für die Virtualisierung dar.

## 2.3 Klassifizierung virtueller Steuergeräte

Virtuelle Steuergeräte haben zum Ziel produktiven Code des Steuergeräteprojekts zu nutzen und diesen für eine, von der Zielhardware (Target) verschiedene, Plattform bereitzustellen. In aller Regel ist das Target des Steuergeräts ein Mikrocontroller und die Zielplattform des virtuellen Steuergeräts ein Windows oder Linux-System mit x86-Prozessorarchitektur. Ausgehend von AUTOSAR wurde folgende theoretische Klassifizierung von virtuellen Steuergeräten abgeleitet [PRO20] und kann als Orientierung gesehen werden:

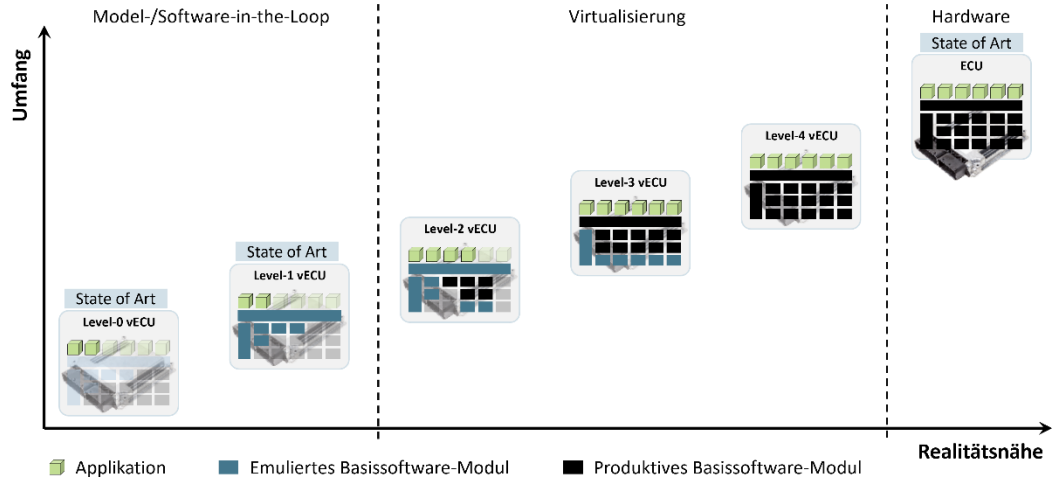


Abbildung 4: vECU-Level aus [PRO20]

### Level-0 vECU

Level-0 ist vor allem im Modultest im Rahmen des Model-in-the-Loop (MiL) zu finden. Hierbei besteht die vECU aus einem oder mehreren Reglermodellen, welche zur Entwicklung der Algorithmen selbst genutzt werden, nicht jedoch aus produktivem Code. In der Softwareabsicherung spielen diese in aller Regel keine wesentliche Rolle und sollen hier vernachlässigt werden.

**Level-1 vECU**

Das Level-1, auch mit *High Cut* [VDA22] bezeichnet, wird durch einen logischen Schnitt entlang der RTE erreicht. Es handelt sich um eine Komposition aus einigen SWCs und der entsprechenden RTE. Einsatz findet hierbei nur produktiver Code. Im Rahmen dieser vECU werden zusätzlich sehr rudimentäre *System Services* emuliert, welche Speicherbereiche und eine Ablaufsteuerungen von Prozessen (Scheduling) bereitstellen. Diese vECU wird in der Regel für die funktionale Absicherung der spezifischen Systemfunktionen entworfen.

**Level-2 vECU**

Mit dieser vECU-Form wird eine erweiterte Emulation der Basissoftwareschicht integriert. Es gibt jedoch keine klare Definition für eine logische Schnittebene. Vielmehr orientiert sich die Emulation der BSW am Anwendungsfall. Einige Beispiele der Erweiterung zur Level-1 vECU sind: COM-Stack, Bus-Interfaces (CAN, Ethernet, etc.), non-volatile Memory oder Diagnosedienste. Die Level-2 vECU ermöglich bereits in sehr frühen Phasen Tests auf der Protokollebene.

**Level-3 vECU**

Das auch mit *Low Cut* bezeichnete Level-3 wird oberhalb der Hardwareschicht geschnitten und beinhaltet die gesamte Kommunikationsschicht der Basissoftware. Technisch gesehen setzt sich eine Level-3 vECU aus allen hardwareunabhängigen Anteilen der produktiven Software sowie einer Virtualisierung des MCAL und eines anteilig virtualisierten Betriebssystems zusammen. *CDDs* müssen häufig gesondert betrachtet und angepasst werden. Die Kommunikation dieser vECU erfolgt auf Grundlage der jeweiligen Busprotokolle. Die Level-3 vECU erweitert die Reichweite für die Absicherung deutlich bis in die Basissoftware hinein.

**Level-4 vECU**

Die *Binary Target vECU* ist binär identisch zum realen Steuergerät. Dies wird durch eine Emulation des Chipsatzes der Zielplattform (Target) auf einem klassischen x86-System erreicht. Diese stellt den Maschinenbefehlssatz für die Hardwaretreiber zur Verfügung. Erreicht wird das unter anderem mit sogenannten Kernel-based virtual machines (KVM). Bei der Level-4 vECU gibt es keinen Code-Unterschied zwischen virtuellem und realem Steuergerät.

# 3 Praktische Anwendungen

## 3.1 Potentiale und Limitierungen virtueller Steuergeräte

Jedes vECU-Level bedingt durch den logischen Schnitt unterschiedliche Potentiale und Limitierungen. Ausgehend von der technischen Kenntnis der unterschiedlichen Level können Potentiale und Limitierungen nun hergeleitet werden.

Einzelne Potentiale und Limitierungen sind in der folgenden Tabelle farblich nach dem Ampelsystem bewertet:

| | Level-1 | Level-2 | Level-3 | Level-4 |
|---|---|---|---|---|
| Debugging durch Brake-Points / Anhalten der Simulation | 🟢 | 🟢 | 🟢 | 🟢 |
| Skalierung - Simulationszeiten (Echtzeitfaktor) | 🟢 | 🟢 | 🟡 | 🟡 |
| Skalierung - Parallelisierung | 🟢 | 🟢 | 🟡 | 🟡 |
| Zeitaufwand der Auslieferung | 🟢 | 🟡 | 🟡 | 🟢 |
| Funktionaler/logischer Schnitt von Steuergeräten | 🟢 | 🟡 | 🟡 | 🟡 |
| Basisfunktionen: Diagnose, Security und Busprotokolle | 🔴 | 🟡 | 🟢 | 🟢 |
| Determinismus / Realitätsnähe | 🟡 | 🟡 | 🟡 | 🟢 |
| Kosten | 🟢 | 🟢 | 🟡 | 🟡 |
| Kopplung mit Hardware-Setups | 🔴 | 🟡 | 🟢 | 🟡 |
| Fehlersimulation von Hardwarefehlern (SW-Rückfallebene) | 🔴 | 🔴 | 🟢 | 🟡 |
| Einsatz bei Treiberentwicklung und Compiler-Analyse | 🔴 | 🔴 | 🔴 | 🟢 |
| Flashen, Analyse von Speicher- und Prozessorauslastungen | 🔴 | 🔴 | 🔴 | 🟢 |
| Integration Non-Autosar/Legacy Code, CDDs | 🔴 | 🔴 | 🔴 | 🟢 |

Tabelle 1: Potentiale und Limitierungen

Mit der Kenntnis der Potentiale und Limitierungen kann bewertet werden, welches vECU-Level für welchen Anwendungsfall genutzt werden kann oder definitiv ausgeschlossen werden sollte. Bei der Bewertung der einzelnen Level ist festzustellen, dass verschiedene Level sich sinnvoll ergänzen können. Im Rahmen einer Absicherungsstrategie muss bewertet werden, welche Anwendungsfälle einer großen Skalierung bedürfen und wo sich deswegen eine Virtualisierung lohnt. Dies kann beispielsweise bei einer Vielzahl von abzusichernden Varianten der Fall sein. Ebenso finden sich viele Testszenarien, bei denen die Absicherung an hardware-basierten Testplattformen eine Verschwendung der Ressource darstellt, da nur die reine Softwarefunktion getestet wird. Häufig ist eine gestaffelte Absicherung mit einer Level-1 vECU, einer Level-3 und einer realen ECU sehr zielführend. Je nach Softwareprojekt kann jedoch auch eine Level-4 vECU unabdingbar sein. Dies kann beispielsweise bei Mikroprozessorarchitekturen und AUTOSAR Adaptive Anteilen der Fall sein.

## 3.2 Einsatz virtueller Steuergeräte

Der Einsatz virtueller Steuergeräte entlang der Integrationsstufen im V-Modell lässt sich aus den Potentialen und Limitierungen ableiten und ist in Abbildung 5 prinzipiell dargestellt. Level-4 findet sowohl im Entwurf als auch im Systemtest Einsatz. Level-1 und Level-3 decken gemeinsam nahezu den gesamten rechten Teil des V's ab.

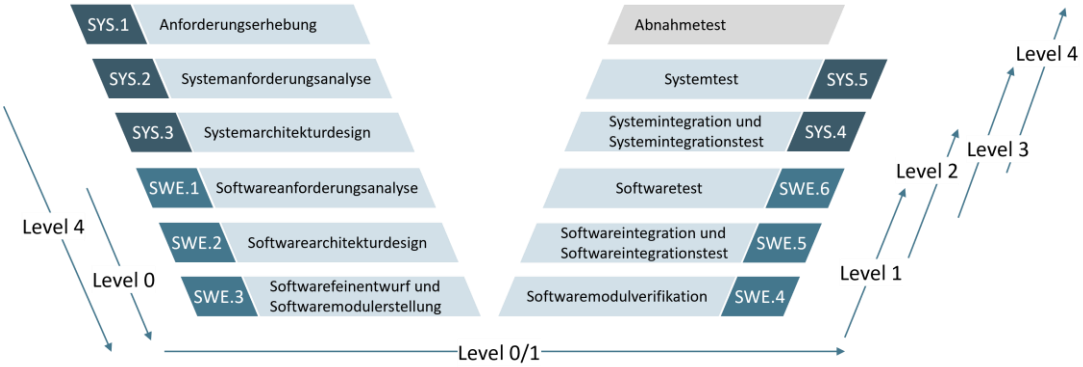


Abbildung 5: Einsatz virtueller Steuergeräte im V-Modell

## 3.3 Anwendungsbeispiele

Die zentrale Frage, die beim Einsatz von virtuellen Steuergeräten zu beantwortet ist, ist die folgende: Was ist der hauptsächliche Testfokus: *die funktionale Softwarewirkkette*, am Ende „nur" über den Hardwaretreiber an ein Kommunikationsmedium (Bus) angeschlossen oder doch der *Hardwaretreiber und die verbundene Physik selbst*? In aller Regel (außer in der Treiberentwicklung) ist ersteres die Antwort. Betrachtet man die Wirkkette innerhalb eines AUTOSAR-Steuergeräts (Abbildung 6), lassen sich solche Fälle mit virtuellen Steuergeräten ab Level-3 ideal absichern, da maximal die Treiberschicht virtualisiert wird.

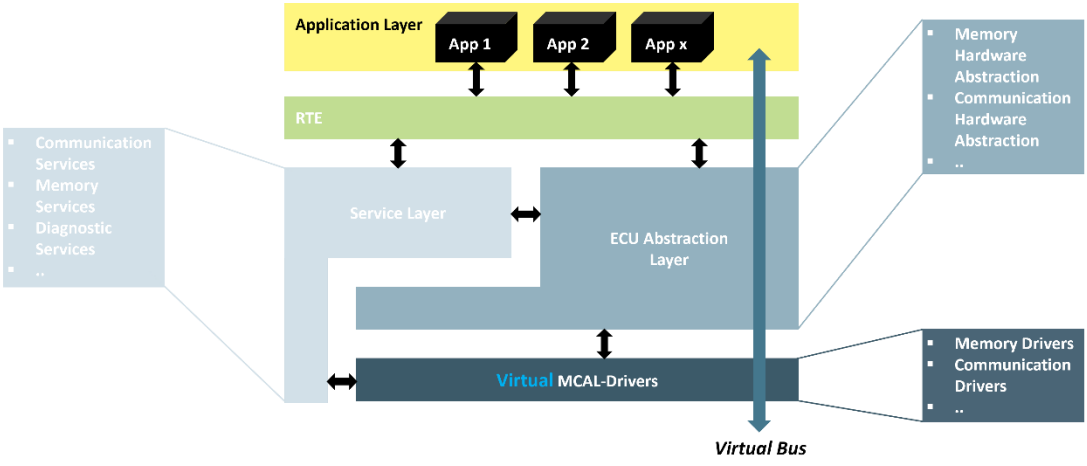


Abbildung 6: Prinzip-Skizze virtualisierter MCAL im Level-3 (CDD vernachlässigt)

Im Folgenden werden drei anschauliche Fälle beschrieben. Zur Vorbereitung wird das virtuelle Steuergerät jeweils in eine Simulation eingebettet, welche alle notwendigen Schnittstellen bedienen kann. Im Falle von Sensorschnittstellen wird der Regelkreis durch die Simulation der Regelstrecke geschlossen. Busschnittstellen werden mit virtuellen Bussen verbunden und mit einer Restbussimulation (Mock) ergänzt.

a) **Komponententest: Diagnose und Buskonformität**
Der erste Fall, siehe Abbildung 7a), beschreibt einen Komponententest von Basis- bzw. Querschnittsfunktionen wie Diagnose oder Buskonformität. Hierbei werden im ersten Fall automatisiert die Diagnose-Parameter (basierend auf dem ODX) getestet. Der Bustreiber kann hier nicht mit abgesichert werden, da dieser virtualisiert ist. Das ist jedoch nicht weiter schlimm, da der Treiber nur „Mittel zum Zweck“ ist und nicht Teil des eigentlichen *System-under-Tests*. Bei den Buskonformitätstests müssen Abstriche bei Buskurzschlüssen oder Ähnlichem gemacht werden, die Protokolltests sind jedoch ausnahmslos möglich (bspw. Service-Discovery).

b) **Fehlerspeichertest**
Beim Fehlerspeichertest werden häufig Fehlerzustände hervorgerufen und anschließend die Reaktion im Fehlerspeicher überprüft. Der relevanteste Anteil dieser Funktion findet nicht in RAM oder ROM statt, sondern in den Services der höherliegenden Schichten. Somit sind auch diese Tests mit virtuellen Steuergeräten durchführbar. Ganz besonders gut lassen sich die Simulationen von beliebigen Fehlerzuständen darzustellen, welche mit Realteilen oder im Fahrzeug deutlich schwieriger ist.

c) **Service Oriented Vehicle Diagnostics (SOVD)**
Service Oriented Vehicle Diagnostics ermöglichen Szenarien wie Remote-Diagnose. Hierzu wird ein Fahrzeug oder das entsprechende Connectivity-Steuergerät mit dem OEM-Backend verbunden. Solche Zugriffe erfordern beim Test einen sehr robusten kabellosen Zugriff in den Testumgebungen. Mit virtuellen Steuergeräten lassen sich solche robusten Zugriffe aus dem Backend sehr einfach umsetzen. Da es sich bei SOVD um ein reines Softwarefeature handelt ist es sehr gut virtuell testbar.

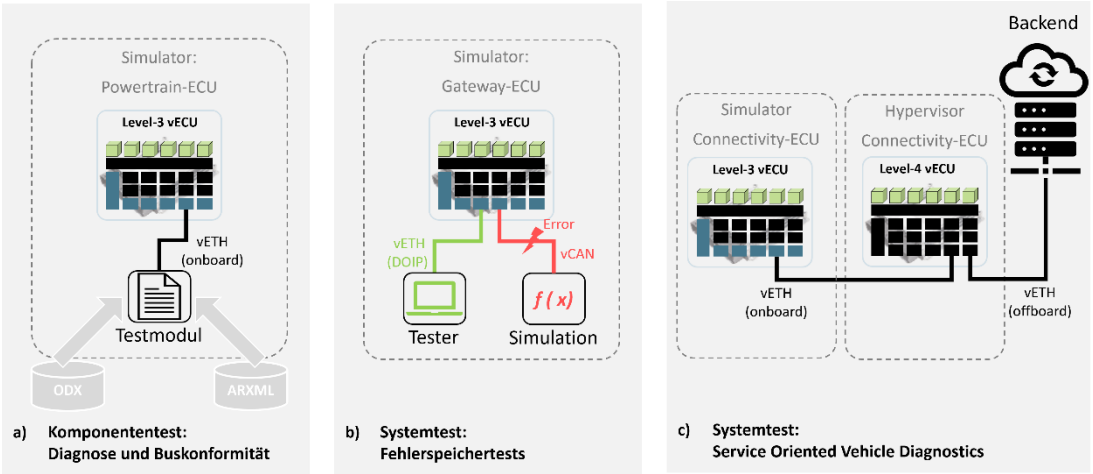


Abbildung 7: Prinzip-Skizze beispielhafter Anwendungen

## 4 Zusammenfassung und Ausblick

In diesem Beitrag wurde die Technologie virtueller Steuergeräte (vECUs) detailliert hergeleitet. Eine zentrale Rolle bei der Entwicklung dieser virtuellen Steuergeräte spielt der AUTOSAR-Standard. Verschiedene Stufen von vECUs bieten unterschiedliche Potenziale und Limitierungen. Anhand praxisnaher Beispiele wurde demonstriert, dass die virtuelle Absicherung ein erhebliches Potenzial über die Absicherung von Applikationsfunktionen hinaus bietet. Insbesondere bei der Absicherung von Basisfunktionen zeigen vECUs großes Potenzial. Mit vECUs ab Level-3 können viele Basisfunktionen frühzeitig und in hoher Qualität abgesichert werden. Dies trägt nicht nur zur Qualität der nachfolgenden Integrationsstufen bei, sondern kann auch die Entwicklung der Basissoftware beschleunigen. Entscheidend ist, dass die Absicherung der Funktionen bereits während des Entwurfs der Softwarearchitektur berücksichtigt wird.

Neben der vECU-Technologie ist die Implementierung eines durchgängigen und konsistenten Absicherungsprozesses über ein entsprechendes Testkonzept und Testmanagement von zentraler Bedeutung. Dieser Prozess sollte einem umfassenden Ansatz folgen. Es ist essenziell, bereits im Testkonzept zu entscheiden, welche Anforderungen mit virtuellen Steuergeräten getestet werden können und welche den Einsatz von Hardware erfordern. Um diese Potenziale zu nutzen, ist ein ganzheitlicher Entwicklungsansatz notwendig. Durch konsequente Verfolgung dieser Ansätze lassen sich nicht nur signifikante Kosteneinsparungen realisieren, sondern auch die Entwicklungsgeschwindigkeit erheblich steigern. Vor diesem Hintergrund kann das virtuelle Steuergerät zu einem entscheidenden Faktor in der zukünftigen Fahrzeugentwicklung werden.

## 5 Literaturverzeichnis


[AUT22] AUTOSAR. (kein Datum). *autosar.org*. Abgerufen am 18. 3 2023 von https://www.autosar.org/

[FOR11] ForTISS GmbH. (2011). *Mehr Software (im) Wagen: Informations- und Kommunikationstechnik (IKT) als Motor der Elektromobilität der Zukunft.* ForTISS GmbH. Abgerufen am 18. 3 2024 von https://mediatum.ub.tum.de/doc/1287138/270280.pdf

[PRO20] prostep IVIP. (2020). Requirements for the Standardization of Virtual Electronic Control Units (V-ECUs). *Smart Systems Engineering*.

[VDA22] Verband der Automobilindustrie e. V. (VDA). (2022). VDA 710 Software-in-the-Loop (SiL) Standardisierung (Version 07/2022). Verband der Automobilindustrie e. V. (VDA).