

Hochschule Karlsruhe
Technik und Wirtschaft
UNIVERSITY OF APPLIED SCIENCES

EIT Fakultät für Elektro-
und Informationstechnik

Hochschule Karlsruhe - Technik und Wirtschaft
Fakultät für Elektro- und Informationstechnik

Bachelor-Thesis

Calibration System Testautomatisierung

von
Thomas Wagner

Matrikel-Nr.: 48389

Referent:	Prof. Dr.-Ing. Philipp Nenninger
Korreferent:	Prof. Dr. rer. nat. Thorsten Leize
Arbeitsplatz:	Robert Bosch GmbH, Schwieberdingen
Betreuer am Arbeitsplatz:	Dipl.-Ing. (FH) Andreas Aberfeld
Zeitraum:	15.04.2018 - 30.08.2018
Version vom:	20. August 2018

Eidesstattliche Erklärung

Ich versichere, die von mir vorgelegte Arbeit selbstständig verfasst zu haben. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder nicht veröffentlichten Arbeiten anderer entnommen sind, habe ich als entnommen kenntlich gemacht. Sämtliche Quellen und Hilfsmittel, die ich für die Arbeit benutzt habe, sind angegeben. Die Arbeit hat mit gleichem Inhalt bzw. in wesentlichen Teilen noch keiner anderen Prüfungsbehörde vorgelegen.

Karlsruhe, 15. August 2018

Zusammenfassung

Damit ein Motorsteuergerät seine Funktion als elektronische Regelung in vollem Leistungsumfang entfalten kann, muss es optimal an den jeweiligen Fahrzeugtyp angepasst werden. Zu diesem Zweck werden Parameter der generischen Softwarefunktionen, die in möglichst vielen Fahrzeugtypen Verwendung finden, im Entwicklungsschritt der Applikation, für den aktuellen Fahrzeugtyp angepasst und erprobt. Für diesen Vorgang ist eine leistungsfähige Schnittstelle zum Motorsteuergerät nötig. Eine Toolkette aus verschiedenen Software- und Hardwareelementen stellt diese Schnittstelle zur Verfügung.

Eine solche Toolkette unterliegt ständiger Änderungen in Soft- und Hardware. Jede Änderung setzt eine erneute Freigabe der Toolkette für die Applikation voraus. In der folgenden Arbeit wird betrachtet, wie sich dieser testbasierte Freigabeprozess automatisieren lässt.

Auf Basis unterschiedlicher vorhandener Vorgehensweisen für die Freigabe, wird eine vereinheitlichte, durch Automatisierung unterstützte, Vorgehensweise entwickelt und in einem entsprechenden Tool zusammengeführt.

Abstract

To enable an engine control unit to fulfill its duty as an electronic control effectively, it has to be adapted to the vehicle type it is working in. This is achieved by adjusting parameters of generic software functions, which are used in a variety of vehicle types. The step of development this is done in is called application. Adjusting and validating these parameters requires a high-performance interface to the control unit. A tool-chain, consisting of multiple software and hardware components, provides this interface.

Such a tool-chain is subject to constant changes in hard- and software. Each change requires a renewed release of the tool-chain for application. The following thesis takes a look at whether the needed test based release process can be automated.

Different existing release processes will be used as a base to develop a unified and automated process which then will be implemented in a common tool.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Arbeitsumfeld	1
1.2	Motivation	1
1.3	Aufgabenstellung der Arbeit	1
1.4	Aufbau der Arbeit	2
2	Grundlagen	3
2.1	Applikation	3
2.2	Toolkette	4
2.2.1	Calibration ECU	5
2.2.2	Zugriffsvariante 1: über XCP	5
2.2.3	Zugriffsvariante 2: über ETK	6
2.2.4	Calibration Software	7
2.2.5	Weitere Varianten	7
2.2.6	Flashen und Speicherseitenverwaltung	7
2.3	Test und Freigabe	8
2.3.1	Terminologie des Testens	8
2.3.2	Freigabe	9
2.4	Testvorgehen	9
2.4.1	Testverfahren	10
2.4.2	IEEE 829	11
2.5	V-Modell	11
2.6	ISO 26262	12
2.7	INCA	13
2.8	ECU-Test	14
2.9	Python	16
2.10	Testautomatisierung	16
2.10.1	Testaufbau	17
3	Konzeption	19
3.1	Use-Case-Analyse	19
3.1.1	Use-Case 1: Freigabe ETK-System	19
3.1.2	Use-Case 2: Freigabe XCP-System	19
3.1.3	Use-Case 3: Flashvorgang	20
3.1.4	Zusammenfassung und Bewertung	20
3.2	Ist-Stand-Analyse	21
3.2.1	Toolsysteme	21
3.2.2	Ablaufdefinition	22
3.2.3	Dokumentationssystem	23
3.2.4	Zusammenfassung und Bewertung	24
3.3	Ergebnis	26
4	Testfallbeschreibung	27
4.1	Anwendung IEEE 829	27
4.2	Analyse Freigabechecklisten	30
4.3	Erfassung weiterer Testfälle	31
4.4	Testfallsammlung	33

5	Architektur	35
5.1	Treiber und Schnittstellen	35
5.1.1	Netzteil	35
5.1.2	Switchbox	37
5.1.3	INCA	38
5.1.4	SQLite	39
5.1.5	Registry	40
5.1.6	A2L-Parser	40
5.2	Testsuite	42
5.2.1	Testkonfiguration und Testbenchkonfiguration	42
5.2.2	Testreport	43
5.2.3	Testablauf	45
6	Umsetzung	49
6.1	ETK-System	49
6.1.1	Umsetzung: Konfiguration und Setup	49
6.1.2	Umsetzung: Testfälle	51
6.1.3	Validierung der Testfälle	55
6.2	XCP-System	55
6.2.1	Anpassung der Zustandsmaschine	55
6.2.2	Anpassung im Versuchsaufbau in HW und SW	55
6.2.3	Validierung der Funktion der harmonisierten Testfälle	56
6.2.4	Umsetzung: XCP-spezifische Testfälle	56
6.3	Dokumentation	57
7	Zusammenfassung und Ausblick	58
7.1	Ergebnis	58
7.2	Ausblick	60
	Literatur	62
	Abbildungsverzeichnis	64
	Tabellenverzeichnis	64
	Abkürzungsverzeichnis	65
	Anhang	a

1 Einleitung

Das nachfolgende Kapitel gibt einen Überblick über die Randbedingungen dieser Arbeit. Es wird auf das Arbeitsumfeld (Kapitel 1.1) eingegangen, die Motivation (Kapitel 1.2) hinter der Arbeit und die genaue Aufgabenstellung (Kapitel 1.3). Zudem wird der allgemeine Aufbau (Kapitel 1.4) erläutert.

1.1 Arbeitsumfeld

Der Geschäftsbereich *Powertrain Solutions - Electronic Control* (PS-EC) der Robert Bosch GmbH entwickelt Steuergeräte (engl. Engine Control Unit - ECU), welche international für sowohl Verbrennungsmotoren als auch alternative Antriebe (Hybride- und Elektroantriebe) genutzt werden. Dem Hauptsitz dieses Geschäftsbereiches in Schwieberdingen gehört u.a. die Abteilung *Engineering Service Business Tools and Methods* (EBT3) an. Diese Abteilung ist für die Entwicklung von Tools und Methoden zuständig, die den Zugriff auf Steuergeräte für die Applikation ermöglichen bzw. erleichtern.

1.2 Motivation

In Serienfahrzeugen findet der Zugriff auf die ECU über ein Controller Area Network (CAN) Bussystem statt. Obwohl hierbei typischerweise eine reservierte Schnittstelle nur für den ECU-Zugriff von extern zur Verfügung steht, lassen sich damit bei 1Mbaud höchstens Nutzdatenraten von 50kByte/s erreichen [Doc13], da die maximale Busauslastung begrenzt ist. Während solche Datenraten für das Messen und Verstellen oder das Flashen einer neuen Software in Serienfahrzeugen ausreichend sind, werden bei der Applikation von ECUs während der Produktentwicklung deutlich höhere Datenraten benötigt. Solche hohen Datenraten, bis zu 30MByte/s, werden erreicht, indem das Steuergerät in der Entwicklung um zusätzliche Hardware- und Softwarekomponenten erweitert wird. Es ergibt sich eine Software- und Hardwaretoolkette, welche bei jeder Änderung einer ihrer Komponenten erneut geprüft und freigegeben werden muss.

„One of the saddest sights to me has always been a human at a keyboard doing something by hand that could be automated.“ [Bei95, S.232]

Diese Aussage motiviert auch den Ansatz, die Freigabe automatisiert zu unterstützen. Vor allem da einige Schritte sehr repetitiv und zeitlich aufwendig sind. Manuelles Testen ist in der Regel teurer als automatisiertes Testen, da meist mehr als nur ein Spezialist mit entsprechendem Know-How benötigt wird. Ein weiterer Grund für die Automatisierung ist die Wiederholung. Jedes Mal wenn ein Test fehlschlägt muss er, nach der Fehlerbehebung, erneut durchgeführt werden. Ein Ablauf der sich einige Mal wiederholen kann. Zudem machen Menschen Fehler, erst recht wenn sie gleichbleibende Abläufe mehrfach abarbeiten müssen. [Bei95]

Aufgrund dieser Argumente wird also die Möglichkeit einer Testautomatisierung untersucht.

1.3 Aufgabenstellung der Arbeit

Konkret beinhaltet der Freigabeprozess einen Ablauf von zu untersuchenden Funktionen, welcher im Zuge dieser Arbeit untersucht und automatisiert umgesetzt werden soll. Hierzu

sollen erst die einzelnen Use-Cases untersucht und dann bereits bestehende Lösungen und Ideen analysiert werden. Eine Testfallbeschreibung und Ergebnisausgabe ist festzulegen. Die erfassten Testfälle sollen dann auf ihre Automatisierbarkeit untersucht werden. Die Automatisierung kann hierbei:

- auf Application Programming Interfaces (APIs) zurückgreifen, um diverse Softwaretools zu steuern,
- die serielle Schnittstelle nutzen, um Hardware wie z.B. ein Labornetzteil zu steuern und
- auf eine Datenbank mit diversen Softwareartefakten für die ECU zuzugreifen, um Versionen und Kompatibilität zu prüfen.

Auf der Basis dieser Untersuchungen soll eine Testfallbibliothek erstellt werden. Ein Tool soll eine Testsuitespezifikation¹ einlesen und entsprechend dieser die geforderten Testfälle aus der Bibliothek abrufen und prüfen. Es ist ein Format für eine solche Testsuitespezifikation zu definieren, um möglichst allgemein viele Tests abdecken zu können. Ebenso muss ein Ausgabeformat definiert werden, um das Ergebnis der Prüfung in einem aussagekräftigen Report darzustellen.

Um die weitere Nutzung des Tools sicherzustellen, soll es modular aufgebaut und gut dokumentiert sein, sodass es von anderen Entwicklern weiter gepflegt werden kann.

1.4 Aufbau der Arbeit

Diese Arbeit teilt sich in fünf Abschnitte: Einleitung, Grundlagen, Konzept, Umsetzung, Auswertung. Kapitel 1 führt zum Thema und zur Aufgabenstellung dieser Arbeit hin und legt die Motivation dar. In Kapitel 2 werden der aktuelle technische Stand und die notwendigen technischen Grundlagen erläutert.

Der dann folgende Teil der Arbeit, der das Konzept, die Umsetzung und die Auswertung bespricht, ist entsprechend dem V-Modell (siehe Kapitel 2.5) aufgebaut. Damit spiegelt er auch das Vorgehen der praktischen Arbeit wieder. Bezogen auf das V-Modell liegt der Teil der Konzeption in der analytischen Phase. Er beinhaltet die Kapitel 3, Konzeption, Kapitel 4, Testfallbeschreibung, und Kapitel 5, Architektur. Letzteres fällt zudem zusammen mit Kapitel 6 in die Umsetzung, im V-Modell der Wendepunkt („Erstellung, Implementierung“). Es folgt die Auswertung in Kapitel 7. Sie gehört zusammen mit einem Teil des Kapitels 6 in die synthetische Phase des V-Modells.

¹eine Testsuite beschreibt einen Ablauf einer Sammlung von Testfällen (vgl. 2.3.1)

2 Grundlagen

In diesem Kapitel sind die theoretischen Grundlagen gesammelt und erläutert, welche für das Verständnis der weiteren Arbeit relevant sind. Zuerst wird in Kapitel 2.1 der Entwicklungsschritt der Applikation allgemein erläutert, bevor dann in Kapitel 2.2 die, für die Applikation benötigte, Toolkette genauer erläutert wird. Wie eingangs erwähnt, geht es in dieser Arbeit um das automatisierte Testen dieser Toolkette, weshalb die Grundlagen für Freigabe und Test in Kapitel 2.3 erläutert werden, sowie die nötige Terminologie. Kapitel 2.4 geht auf theoretische Grundlagen von Testabläufen ein. Das in Kapitel 2.5 beschriebene Vorgehensmodell ist Grundlage für u.a. den Aufbau der in Kapitel 2.6 beschriebenen ISO-Norm. Es folgen die Kapitel 2.7, 2.8 und 2.9, welche auf die Software eingehen, die für diese Arbeit relevant ist. Im abschließenden Kapitel 2.10 wird dann auf die zur Verfügung stehenden Hardware- und Softwareelemente eingegangen, mit welchen bereits Teilautomatisierungen für die Tests umgesetzt wurden und die in dieser Arbeit genutzt werden.

2.1 Applikation

„Die Applikation bzw. das Kalibrieren von Steuergeräten ist ein iterativer Prozess aus Messen und Verstellen zur Laufzeit, um die Parameter von Steuergeräte-Algorithmen (Funktionen) optimal [auf den Fahrzeugtyp] abzustimmen.“ [Vec18]

ECU-Software wird so ausgelegt, dass deren Regel-, Steuer- und Diagnosefunktionen, durch das Kalibrieren von Verstellgrößen, auf eine Vielzahl von Fahrzeugen angepasst werden können. [ETA18a]

Unter *Applikation* ist also ein Schritt der ECU-Softwareentwicklung zu verstehen. Dieser folgt der *Funktionsentwicklung*. Die Funktionsentwicklung ist für die Entwicklung von verschiedenen Funktionen bzw. Funktionsbausteinen zuständig (z.B. Reglerbausteine). Aus diesem Pool von Funktionsbausteinen werden, je nach Bedarf und Projektanforderung, durch die *Softwareentwicklung*, Programmstände erstellt. Programmstände beinhalten alle nötigen „Gesamtfunktionen“ für die Steuerung und Regelung des Antriebsstranges. Im letzten Entwicklungsschritt wird die gesamte Software, durch die *Applikation*, dem jeweiligen Projekt weiter angepasst. Allgemein zusammengefasst versteht man unter *Applikation* also das Anpassen von generischer Software.

Ein Beispiel hierfür ist in Abbildung 1 dargestellt. Einzelne Funktionsbausteine f_1 und f_2 wurden in eine Reihenschaltung zusammengeführt um eine bestimmte Funktion zu erfüllen. Funktionsbaustein f_1 könnte beispielsweise zur Messwertskalierung genutzt werden während f_2 einen PID-Regler implementiert.

Durch Anpassen der Verstellgrößen a , b und c unter Zuhilfenahme von Messgrößen x , y und z als Feedback kann die Gesamtfunktion, bestehend aus der Reihenschaltung der beiden Funktionsblöcke f_1 und f_2 , entsprechend der Gegebenheiten des Projektes abgestimmt werden. Die angepassten Verstellgrößen werden dann in einem Datenstand hinterlegt. Das abgeschlossene Projekt beinhaltet dann einen Programmstand und einen Datenstand.

Für diesen Applikationsvorgang wird eine Toolkette aus Hardware- und Softwarekomponenten genutzt, die einen schnellen Zugriff auf Verstell- und Messgrößen ermöglicht. Abbildung 1 zeigt wie ein solcher Zugriff auf Verstell- und Messgrößen zudem für Ra-

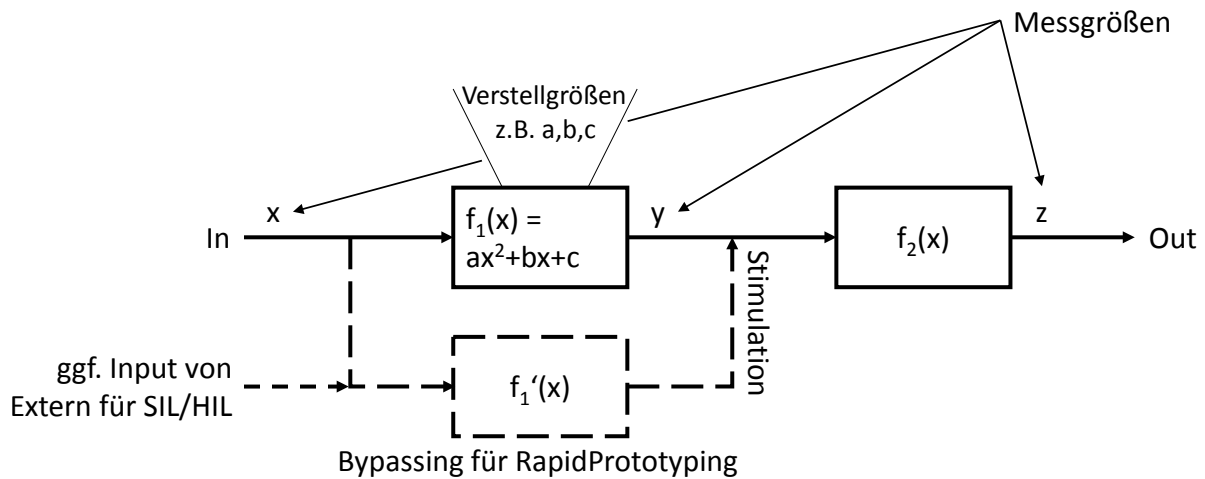


Abbildung 1: ECU-Funktionen und ihre Größen

pid Prototyping (durch Bypassing) und SIL/HIL (Software und Hardware In the Loop) genutzt werden kann. Bypassing ermöglicht Funktionsentwicklern neue Funktionen, die noch nicht auf einer ECU implementiert sind, mit externer Hardware an einem realen Aufbau zu testen. SIL- und HIL-Aufbauten ermöglichen es Entwicklern, mittels Modellen, Parameter zu beobachten, wenn das Zielsystem nur teilweise oder gar nicht zur Verfügung steht.

2.2 Toolkette

In diesem Kapitel werden zwei Toolkettenvarianten erläutert. Beide Varianten nutzen eine spezielle *Calibration ECU* sowie ECU externe *Calibration Hardware* und entsprechende Software auf dem Rechner des Applikateurs (vgl. Abbildung 2). Sie unterscheiden sich durch die Schnittstelle, über die auf die ECU zugegriffen wird.

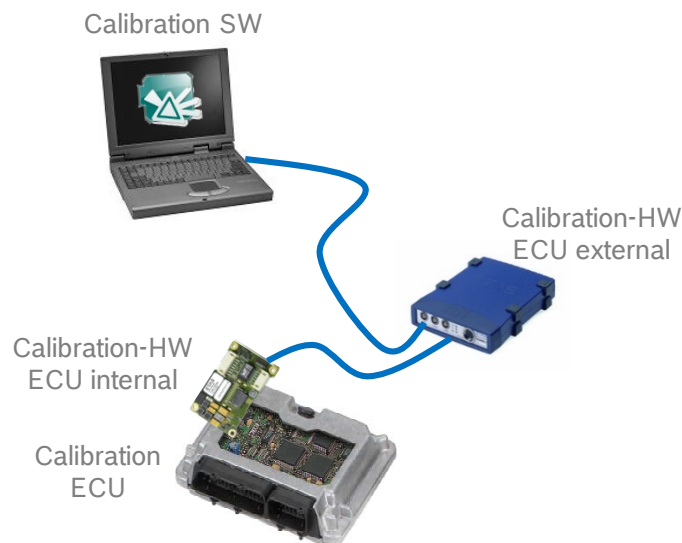


Abbildung 2: Schematische Darstellung der Toolkette (Quelle: Intranet Bosch)

2.2.1 Calibration ECU

Bei der Applikation werden ECUs genutzt, welche sich in ihrem Hardwareaufbau von Serien-ECUs unterscheiden. Hierbei wird meist das Production Device (PD) durch ein Emulation Device (ED) ergänzt. Meist handelt es sich bei dem PD und dem ED um zwei getrennte Mikrocontroller. Das ED stellt dem Gesamtsystem hauptsächlich zusätzlichen RAM (Random Access Memory) zur Verfügung. Es kann sich aber auch um Funktionen wie eine erweiterte Debugschnittstelle kümmern, um eine höhere Bandbreite für den Speicherzugriff über die Debugschnittstelle zu ermöglichen. Teilweise wird auch das Gesamtsystem als Engineering Device (ebenfalls als ED abgekürzt) bezeichnet. Wird die Abkürzung ED auf das Gesamtsystem bezogen, handelt es sich innerhalb der ECU um nur einen Mikrocontroller, der im Vergleich zu einem PD-Mikrocontroller um zusätzlichen RAM und Register für Debugging erweitert wurde. In dieser Arbeit wird ED aber immer als Abkürzung für Emulation Device genutzt und steht somit für einen zusätzlichen Mikrocontroller der parallel zum PD arbeitet.

Da Verstellgrößen im Serienfahrzeug Konstanten sind, werden sie im Flash-Speicher abgelegt. Im Applikationsfall ist der Zugriff auf einzelne Größen im Flash-Speicher allerdings, aufgrund der Charakteristik von Flash-Speicher², ungünstig. Durch das Laden von Verstellgrößen in den zusätzlichen RAM des ED wird diese Problematik umgangen. Es gibt unterschiedliche Ansätze und Vorgehensweisen um diese Transition, vom Flash-Speicher in den ED-RAM, zu realisieren, sowie alle entsprechenden Zugriffe „umzuleiten“. Aufgrund ihrer Irrelevanz für diese Arbeit wird auf diese Vorgehensweisen nicht weiter eingegangen. (Weiterführende Informationen [PZ16, S.79 ff.])

2.2.2 Zugriffsvariante 1: über XCP

Beim direkten Zugriff auf die ECU über das XCP-Protokoll fällt die ECU interne *Calibration Hardware* insofern weg, dass lediglich eine zusätzliche CAN³- oder Ethernet-Schnittstelle benötigt wird (vgl. Abbildung 2). Das Protokoll wird direkt auf dem Mikrocontroller implementiert und nutzt diese CAN- oder Ethernet-Schnittstelle. Da in diesem Fall die ECU den Datenverkehr bearbeiten muss, ist die so erreichbare Geschwindigkeit vergleichsweise gering.

Im Falle von XCP-on-CAN ist ECU externe *Calibration Hardware* nötig, um die Umsetzung auf z.B. XCP-on-Ethernet⁴ vorzunehmen. Erst so kann der Rechner des Applikateurs die Verbindung über seinen Ethernet-Port herstellen.

XCP Das *Universal Measurement and Calibration Protocol* (XCP) ist eine Weiterentwicklung des *CAN Calibration Protocol* (CCP). Es ist durch die ASAM MCD-1 standardisiert [ASA17]. Die Hauptanwendung von XCP ist, wie bei CCP, das Messen und Verstellen von internen ECU-Parametern. Das Protokoll ermöglicht ein ereignissynchrones Erfassen von Messwerten der ECU. XCP wurde hierbei, anders als CCP, so entwickelt, dass es auf einer Vielzahl von physikalischen Layern arbeiten kann wie z.B. CAN, CAN FD⁵,

²Flash-Speicher kann nur begrenzt oft wieder beschrieben werden, bevor es durch Löschvorgänge zerstört wird

³freie CAN Schnittstellen sind oft sogar ohne Erweiterung verfügbar

⁴je nach Hardware ist zudem USB, Ethernet mit TCP/IP oder UDP/IP möglich

⁵CAN Flexible Data-Rate (FD) ist eine Erweiterung von CAN, die höhere Datenraten erlaubt. Diese werden ermöglicht, indem das Datenframe von 8 byte auf 64 byte angehoben wurde und die Datenphase mit einer höheren Datenrate übertragen wird, als die Arbitrierungsphase.

FlexRay, USB (Universal Serial Bus) oder Ethernet. Diese Unabhängigkeit von einem bestimmten physikalischen Layer wurde durch eine Trennung des Protokolls in Protokoll Layer und Transport Layer erreicht (OSI-Modell).

XCP ist ein Protokoll, welches dem Master-Slave Prinzip folgt. Das Protokoll ermöglicht dem Master adressorientierten Schreib- und Lesezugriff auf Speicheradressen des Slaves.

Die Kommunikation über das XCP-Protokoll ist in zwei Bereiche unterteilt: Command Transfer Objects (CTO) und Data Transfer Objects (DTO). CTOs nutzen eindeutige Nummern, die der Master als Befehle an den Slave senden kann. Der Großteil dieser Befehle ist nur optional zu implementieren. Lediglich Befehle wie „Connect“ oder „Get_Status“ sind nicht optional. Während bei der Kommunikation über CTOs normalerweise jedem Befehl des Masters eine Antwort des Slaves folgt, ist es über DTOs möglich, dass der Slave ohne diesen Overhead⁶ Messdaten an den Master schickt. Der Master definiert zu Beginn welche Messwerte er empfangen will und bei welchen Events. Mit einem Aktivierungsbefehl schickt nun der Slave, ohne Rückmeldung des Masters entsprechend der Auslöseevents, die Messwerte und stoppt diese zyklische Übertragung erst auf einen weiteren Befehl des Masters hin. [PZ16, ASA17]

2.2.3 Zugriffsvariante 2: über ETK

Der Zugriff auf die ECU über einen ETK stellt die Möglichkeit des Applikationszugeschlusses mit der momentan höchst möglichen Geschwindigkeit dar. Über spezielle Trace-ETKs lassen sich bis zu 30MByte/s erreichen. [Doc13]

ETK ETK steht für Emulations-Tastkopf. Bezogen auf Abbildung 2 gilt dieser als ECU interne *Calibration Hardware*. Der ETK nutzt meist einen direkten Zugang über die Debug-Schnittstelle oder über den parallelen Daten- und Adressbus des Mikrocontrollers. Damit ist ein echtzeitfähiger Zugriff auf Verstell- und Messgrößen möglich. [ETA18g]

Der ETK ist über eine proprietäre Schnittstelle der ETAS GmbH⁷, auf welcher ein proprietäres Protokoll arbeitet, mit einem Steuergeräte- und Bus-Schnittstellenmodul verbunden.

Es gibt vergleichbare Produkte von anderen Herstellern (z.B. Vector Informatik oder Accurate Technologies Inc.), diese spielen allerdings im Entwicklungsprozess eine untergeordnete Nebenrolle und werden daher nicht weiter behandelt.

Steuergeräte- und Bus-Schnittstellenmodul Das Steuergeräte- und Bus-Schnittstellenmodul wird in Abbildung 2 allgemein als ECU externe *Calibration Hardware* bezeichnet. Ein solches Modul ermöglicht die gleichzeitige zeitsynchrone Erfassung von Messgrößen der ECU, von Fahrzeugbussen und anderen zugeschalteten Messmodulen. Über eine Ethernet Schnittstelle, die das XCP-Protokoll nutzt, wird eine Verbindung mit dem Host-Rechner hergestellt. [ETA18f]

Da in der Toolkette mit ETK die ECU den Datenverkehr nicht selbst bearbeiten muss ist die zweite Variante deutlich schneller, obwohl beide Varianten aus Sicht des Host-Rechners dasselbe Protokoll nutzen.

⁶Daten die nicht zu den Nutzdaten gehören

⁷ETAS GmbH, Stuttgart, Deutschland

2.2.4 Calibration Software

Die Toolkette wird mit der *Calibration Software* komplettiert.

INCA Das INCA Softwarepaket von ETAS wird als Werkzeug für Messung, Steuergeräte-Applikation und Diagnose genutzt. Es stellt bei beiden Zugriffsvarianten das User Interface (UI) bereit und ist das Ende der Toolkette. (siehe auch Kapitel 2.7)

2.2.5 Weitere Varianten

Der Vollständigkeit halber sind hier noch einige weitere Abwandlungen der angesprochenen Toolkettenvarianten erwähnt.

Bei der Nutzung von speziellen ETKs (XETKs) wird die ECU externe *Calibration Hardware* nicht benötigt. Diese ETKs stellen direkt eine Ethernetschnittstelle zur Verfügung, welche das XCP-Protokoll implementiert. Dennoch kann auch diese ETK-Variante an ECU externe *Calibration Hardware* angeschlossen werden. Das ermöglicht gleichzeitige zeitsynchrone Erfassung von Messgrößen der ECU, von Fahrzeugbussen und anderen zugeschalteten Messmodulen. Wenn der ECU-Zugriff direkt mit XCP über CAN oder Ethernet realisiert ist gilt selbiges.

2.2.6 Flashen und Speicherseitenverwaltung

Neben dem Zugriff auf Verstell- und Messgrößen ermöglicht eine Applikationstoolkette das Beschreiben des Flash-Speichers einer ECU. Ein solcher Flashvorgang (kurz Flashen) wird benötigt, wenn ein neuer Programmstand (eine neue Firmware) oder ein Datensatz aufgespielt werden soll.

Ein Datensatz, der den kompletten Umfang an Verstellgrößen des zugehörigen Programmstandes beinhaltet, wird auch als Datenstand bezeichnet. Solche Datenstände liegen normal im ECU-Flash, können aber auch als Sicherung auf einem Rechner gespeichert werden. Liegt ein Datensatz im Flash der ECU ist er nur durch einen Flashvorgang veränderbar. Damit ein Applikateur trotzdem schnell einzelne Verstellgrößen anpassen kann, werden diese in den RAM des ED kopiert. Der ECU kann vorgegeben werden, ob sie für ihr aktuell laufendes Programm die Größen aus dem RAM (Arbeitsseite - AS) oder aus dem Flash (Referenzseite - RS) nutzen soll. Wurden Verstellgrößen im RAM verändert, können die Änderungen als Datenstand auf dem Rechner gesichert werden. Das ist dann nötig, wenn die Anpassungen noch nicht in den Flashspeicher geschrieben werden sollen, die Fahrzeugversorgung aber abgeschaltet wird, wobei der flüchtige RAM die Werte verlieren würde. Verstellgrößen in einem Datenstand auf einem Rechner können „offline“ bearbeitet werden. Ein Datenstand kann separat von einem Programmstand in den Flash der ECU oder den ED-RAM geladen werden. Wird ein Datenstand direkt aus dem RAM des ED in den Flash-Speicher der ECU gespielt, spricht man von einem Kopieren von AS nach RS. Um Referenzwerte wiederherzustellen ist ein Kopieren von RS nach AS ebenso möglich. (vgl. Abbildung 7)

Für den Flashvorgang ist eine sogenannte ProF-Konfiguration erforderlich. Die ProF-Konfiguration dient als Eingang für die Programm Flash Toolkomponente (ProF). Die Konfiguration definiert die Speicherbereiche für Code- und Datensegmente, abhängig vom im Steuergerät verbauten Mikrocontrollertyp.

2.3 Test und Freigabe

Jede Variante der Applikationstoolkette, die in Kapitel 2.2 erwähnt wurde, kann in ihren Hardware- und Softwarekomponenten variieren. Um sicherstellen zu können, dass die Toolkette in sich konsistent ist und keine Fehler aufweist, muss die Kombination, die im jeweiligen Projekt genutzt wird, verifiziert und freigegeben sein. Die Verifikation der beteiligten Komponenten erfolgt durch unterschiedliche Tests.

2.3.1 Terminologie des Testens

Ganz allgemein versteht man unter dem Begriff *Testen* Folgendes [SL12]: „Unter dem Testen von Software wird jede (im Allgemeinen stichprobenartige) Ausführung eines Testobjekts verstanden, die der Überprüfung des Testobjektes dient. Die Randbedingungen für die Ausführung des Tests müssen festgelegt sein. Ein Vergleich zwischen Soll- und Istverhalten des Testobjekts dient zur Bestimmung, ob das Testobjekt die geforderten Eigenschaften erfüllt.“

Das Ergebnis eines Testes gibt Feedback zur Erkennung von Abweichungen, zur Performance und ggf. zur Identifizierung von potentiellen qualitativen Verbesserungen für die geplante Anwendung. [IEE08]

Entsprechend des ISTQB (International Software Testing Qualifications Board) Glossars [IST16] lassen sich weitere Begriffe festlegen. Da dieses Glossar in englische Sprache verfasst ist, kann es durch die Übersetzung zu Interpretationsspielräumen kommen. Zudem wurden einige Begriffe präzisiert, um sich besser für die Gegebenheiten dieser Arbeit zu eignen.

Die ISTQB entnimmt den Begriff *Test* direkt der IEEE 829 und beschreibt damit eine Sammlung von einem oder mehreren *Testfällen*. Demnach definiert ein *Testfall* (IEEE 610) eine Sammlung von *a*) Eingangswerten, *b*) Ausführungsvorbedingungen, *c*) erwartetem Ergebnis und *d*) Ausführungsnachbedingungen, für ein bestimmtes *Testobjekt* (ISTQB) oder eine *Testbedingung* (ISTQB). Das *Testobjekt* oder *System Under Test (SUT)* (ISTQB) ist die Komponente oder das System, welches getestet werden soll. Je nach Strukturierung können die Begriffe gleichbedeutend sein oder ein SUT kann eine Sammlung von *Testobjekten* sein. Ein *Testobjekt* besteht meist aus mehreren *Testitems* (ISTQB), welche einzelne individuelle Elemente darstellen, die getestet werden sollen. Eine *Testbedingung* ist ein Teil oder eine bestimmte Funktionalität einer Komponente oder eines Systems, welche/r anhand von einem oder mehreren *Testfällen* verifiziert werden soll.

Unter einer *Testsuite* (ISTQB) versteht man die Strukturierung einer Sammlung von *Testfällen*. Eine solche Strukturierung zeichnet sich meist dadurch aus, dass die Ausführungsnachbedingung eines *Testfalles* die Ausführungsvorbedingung des nächsten *Testfalles* ist, sodass sich ein Ablauf ergibt. In dieser Arbeit wird deshalb der Begriff *Testablauf* anstatt *Testsuite* genutzt. *Testsuite* hingegen soll den *Testablauf* in Kombination mit der *Testumgebung* (IEEE 610) beschreiben.

Die *Testumgebung* beschreibt die Umgebung, welche die benutzte Hardware, Instrumente, Software Tools und andere unterstützende Elemente beinhaltet. Ein *Testskript* (ISTQB) wird meist als Bezeichnung für eine automatisierte Sequenz von Schritten zur Ausführung eines *Testfalles* bezeichnet. Die *Testspezifikation* (auch Testfallbibliothek) beschreibt ein Dokument welches *Testfälle* und ihre Ausführungsvorbedingungen, Ausführungsnachbedingungen, *Testbedingungen* und teilweise *Testskripte* beschreibt.

Abschließend folgt ein *Testreport* (ISTQB) oder *Testprotokoll* (ISTQB) welche analysierte Daten und Ergebnisse der durchgeführten *Testfälle* beinhaltet.

Testdesign (ISTQB) beschreibt den Prozess, bei dem allgemeine *Testziele* (ISTQB) in greifbare *Testbedingungen* und *Testfälle* überführt werden. Ein *Testziel* (ISTQB) ist die Ursache, der Grund oder der Anlass einen Test auszulegen und auszuführen.

2.3.2 Freigabe

Abbildung 3 visualisiert das Vorgehen, dass für die Freigabe einer neuen oder geänderten Toolkette notwendig ist. Einzelne Komponenten wie z.B. ein Steuergeräte- und Bus-Schnittstellenmodul werden zuerst einzeln vom jeweiligen Hersteller getestet und müssen deshalb nicht noch einmal separat geprüft werden.

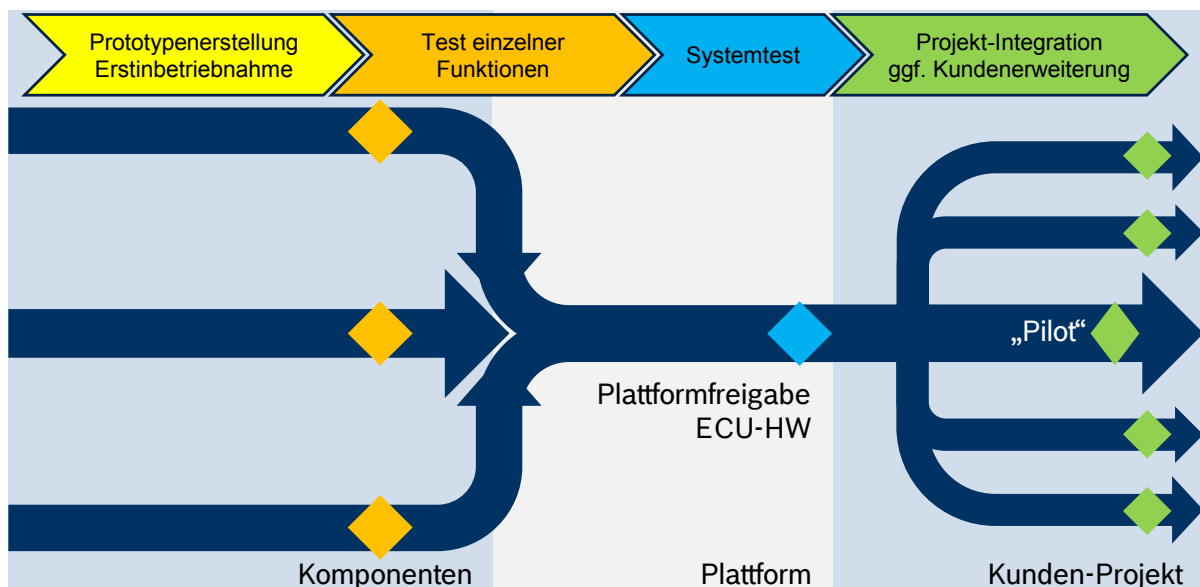


Abbildung 3: Test und Freigabe der Toolkette (*Quelle: Intranet Bosch*)

Auf den Test und die Freigabe einzelner Komponenten folgt der Test des Gesamtsystems, der Plattform. Hierbei wird die ganze Toolkette anhand eine Freigabecheckliste mit unterschiedlichen Testfällen geprüft. Die erste Freigabe einer neuen Plattform ist meist Teil eines Pilotprojektes. Eine so freigegebene Plattform kann, nach Abschluss des Pilotprojektes, auch in anderen Projekten genutzt werden.

2.4 Testvorgehen

Nachdem in Kapitel 2.3 bereits auf die grundlegende Terminologie des Testens und das Testen als Verifikationsmethode eingegangen wurde, geht dieses Kapitel nun auf Vorgehensweisen für Testen ein.

Für einen optimalen Test sollte es das Ziel sein eine möglichst geringe aber ausreichende Anzahl an zu erstellenden Tests zu definieren, um die maximale Anzahl Fehler zu identifizieren. Diese optimale Anzahl an Tests ist damit dennoch stichprobenartig. Diese stichprobenartige Charakteristik von Tests und die damit von der Testvollständigkeit abhängige Qualität der Testaussage bedeutet, dass das Vorgehen bei der Erstellung von Tests (Testdesign) einen maßgeblichen Einfluss auf den Erfolg des Tests hat. [Nö12]

Es geht hierbei um das sogenannte *Test Orakel Problem*. In einem Test wird auf jede Eingabe eine Ausgabe des SUT erwartet. Im Zuge des Testes muss dann die Ist-Ausgabe mit einer Soll-Ausgabe verglichen werden, um feststellen zu können, ob das System ordnungsgemäß funktioniert. Für diesen Vergleich muss bekannt sein, welche Ausgabe bei welcher Eingabe zu erwarten ist, wenn das SUT korrekt arbeitet. Das *Test Orakel* gibt diese erwarteten Ausgaben vor. Leiten sich die korrekten Ausgaben aus z.B. Spezifikationsdokumenten für das SUT ab, stellen diese das *Test Orakel* dar. Es liegt allerdings in der Natur solcher Spezifikationen, dass sie teilweise ungenau oder unvollständig sind. Daraus folgt, dass auch der Test auf Basis der erwarteten Ausgabe entsprechend des *Test Orakels* unvollständig ist (vgl. Blackbox-Test im folgenden Unterkapitel). [BHM⁺15, How78]

2.4.1 Testverfahren

Grundsätzlich lassen sich Testverfahren in drei Klassen aufteilen, welche für die Erstellung von Tests hinzugezogen werden können [Nö12]:

- Funktionale Tests,
- Nicht-Funktionale Tests und
- Diversifizierende Tests.

Synonyme für *Funktionale Tests* und *Nicht-Funktionale Tests* sind Blackbox- und Whitebox-Tests. [SW02]

Bei einem Whitebox-Test (oder *Nicht-Funktionalen Test*) wird die innere Struktur des Testobjektes (z.B. auf Robustheit) geprüft, indem die Auswahl der Tests von der Implementierung abgeleitet wird. Dazu muss die interne Struktur des Testobjektes dem Tester in Gänze bekannt sein.

Bei einem Blackbox-Test (oder *Funktionalen Test*) werden die Tests lediglich entsprechend der Anforderung an das Testobjekt und anhand der beobachtbaren Ein- und Ausgabe formuliert, ohne die Struktur der Implementierung zu beachten bzw. kennen zu müssen. Ist die Struktur eines zu testenden Gesamtsystems bekannt, nicht aber die interne Struktur einzelner Objekte des Gesamtsystems bietet sich der Mittelweg zwischen Blackbox- und Whitebox-Test an, der Greybox-Test (vgl. Abbildung 4). [Bei95, SL12, BS08]

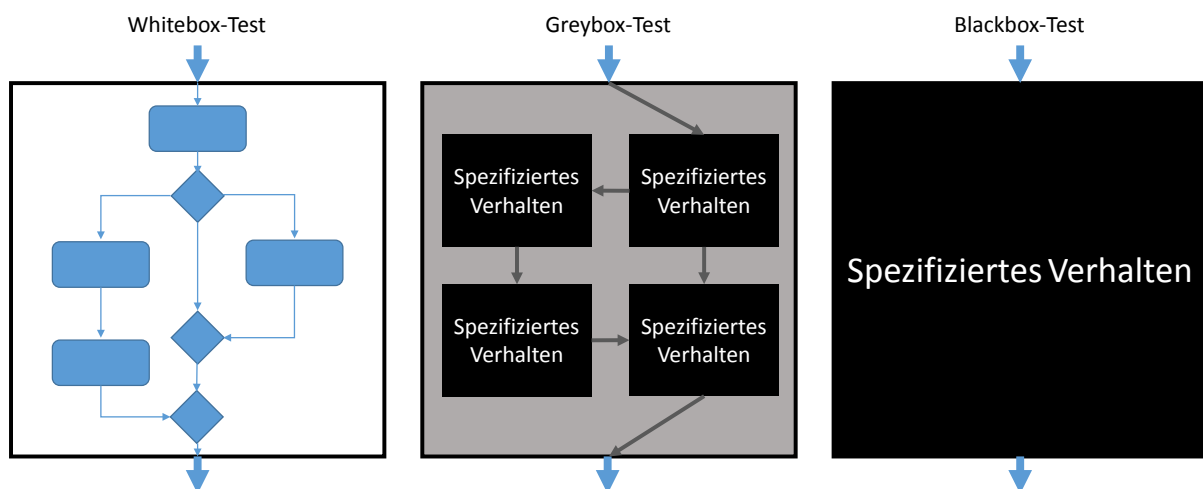


Abbildung 4: White-, Grey-, Blackbox-Test

Der Vorteil von Greybox- oder Blackbox-Tests ist, dass Testfälle direkt aus den Spezifikationsdokumenten abgeleitet werden können und bei Testerfolg direkt den geforderten Funktionsumfang bestätigen. Diese Abhängigkeit bedeutet aber auch, dass die Qualität des Tests nur so hoch sein kann wie die Qualität der Referenz, die die Spezifikationsdokumente zur Verfügung stellen.

Diversifizierende Tests verzichten verglichen mit *Funktionalen Tests* und *Nicht Funktionalen Tests* ganz auf den Vergleich von IST- und SOLL-Verhalten, egal ob diese durch die Implementierung oder die Spezifikation gegeben sind. Bei einem *Diversifizierende Tests* werden stattdessen verschiedene Versionen des Testobjektes miteinander verglichen. Ein solcher *Diversifizierende Test* ist der Regressionstest.

Ein Regressionstest kann dann durchgeführt werden, wenn ein System bereits in Gänze getestet wurde. Gibt es Änderungen in dem bereits getesteten System kann eine minimale Menge Testfälle ermittelt werden, die von der Änderung betroffen sind. So wird in einem dynamischen Entwicklungsprozess verhindert, dass für jede Änderung der gesamte Test wiederholt werden muss. Zeit und Kosten werden gespart, ohne Abstriche bei der Qualität hinnehmen zu müssen.

2.4.2 IEEE 829 [IEE08]

Der IEEE 829 „*Standard for Software and System Test Documentation*“ Standard beschreibt in unterschiedlichen Ebenen Vorgänge, Strukturen und Dokumentationen für Testaktivitäten im Bereich von Software und Systemen. Er:

- etabliert ein Grundgerüst für Testprozesse, -aktivitäten und -aufgaben während des gesamten Lebenszyklus einer Software,
- definiert Testaufgaben und deren benötigte Ein- und Ausgaben,
- legt die minimalen Testaufgaben basierend auf einer Einordnung eines Systems in *Integrity level*⁸ fest
- und schlägt in einem Satz aus acht Basis-Dokumenten eine Grundlage für die Testdokumentation vor.

Der IEEE 829 Standard ist Teil des Lehrplans von sowohl ISTQB Qualifizierungen sowie von ASQF Qualifizierungen (Arbeitskreis Software-Qualität und -Fortbildung).

2.5 V-Modell

Das V-Modell wurde ursprünglich von Boehm [Boe84] als symmetrisches Vorgehensmodell eingeführt und baut auf dem Wasserfallmodell [Boe81] auf. Neben vielen anderen Industriebereichen wird dieses inzwischen auch in der Entwicklung von Hardware und Software in der Automobilindustrie flächendeckend als Standardmodell verwendet [Nö12]. Das analytisch planende Vorgehen des Wasserfallmodells wird hierbei übernommen und um korrespondierende Testschritte erweitert. Jedem Schritt der analytischen Phase im absteigenden Ast (Wasserfall) steht ein entsprechender Schritt in der synthetischen Phase im aufsteigenden Ast gegenüber (vgl. Abbildung 5).

⁸„An integrity level is an indication of the relative importance of software (or of a software characteristic, component, or test level) to its stakeholders, [...]“ [IEE08]

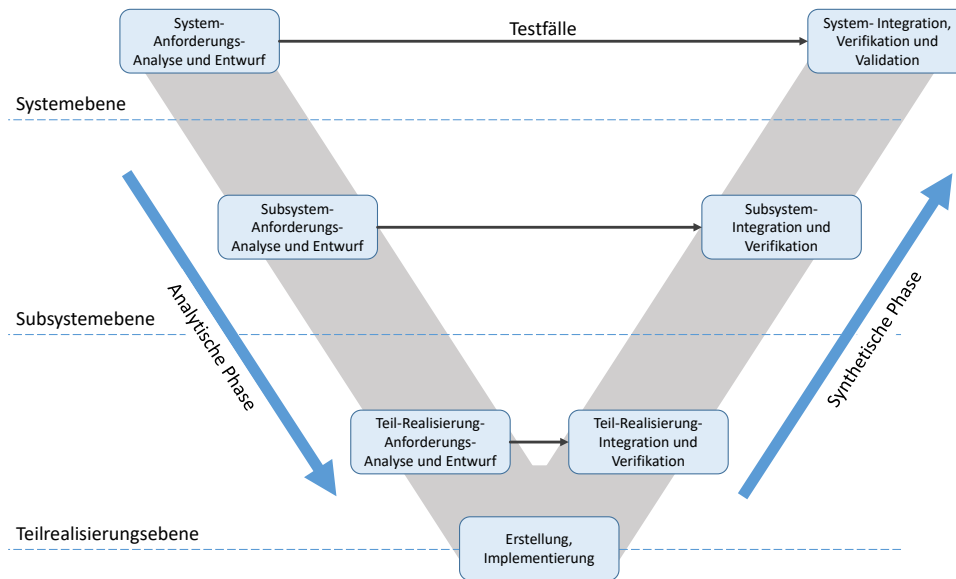


Abbildung 5: V-Modell

Somit ergibt sich ein im absteigenden Ast bei der Spezifikation ein Vorgehen, bei welchem, in immer weniger abstrahierenden Ebenen, das Gesamtsystem erfasst wird. Das so geplante System wird umgesetzt und dann in zunehmend abstrahierenden Ebenen korrespondierend zu dem absteigenden Ast im aufsteigenden Ast integriert, verifiziert und getestet.

2.6 ISO 26262

Die ISO 26262 „Road vehicles - Functional safety“ adaptiert die IEC 61508 und passt diese an die speziellen Gegebenheiten der Anwendung in elektrischen und elektronischen (E/E) Systemen in Kraftfahrzeugen an. Der Standard gewährleistet, bei der weiterhin immer größer werdenden technologischer Komplexität, funktionale Sicherheit in E/E-Systemen in Kraftfahrzeugen. Hierzu gibt die Norm ein Prozessrahmenwerk und Vorgehensmodell, zusammen mit den nötigen Aktivitäten, Arbeitsprodukten und Methoden vor. [ISO11]

Die Abbildung 6 zeigt die Struktur aller zehn Bände aus denen sich die ISO 26262 zusammensetzt, wovon sich die Bände 3 bis 7 auf das V-Modell und dessen geforderte Aktivitäten und Methoden beziehen. Dieses V-Modell erweitert das in 2.5 besprochene V-Modell. Beginnend in einer Konzeptphase (Band 3) wird dann in die Produktentwicklung auf Systemlevel übergegangen (Band 4). Weiter im Detail folgen einzelne Entwicklungsprozesse für Hardware (Band 5) und Software (Band 6), welche in ihrem Vorgehen untergeordneten V-Modellen folgen. Nach der Entwicklung von Software und Hardware wird wieder auf Systemebene abstrahiert, wo Verifikations- und Testvorgehen beschrieben sind (ebenfalls Band 4). Das V-Modell wird mit dem Schritt der Produktion und des Betriebes abgeschlossen (Band 7). Die anderen Bände (1, 2, 8, 9, 10) stellen unterstützende Informationen zur Verfügung wie ein Glossar mit Begriffen (Band 1) oder Vorgänge zur Analyse und Einordnung nach Automotive Safety Integrity Level (ASIL) (Band 9).

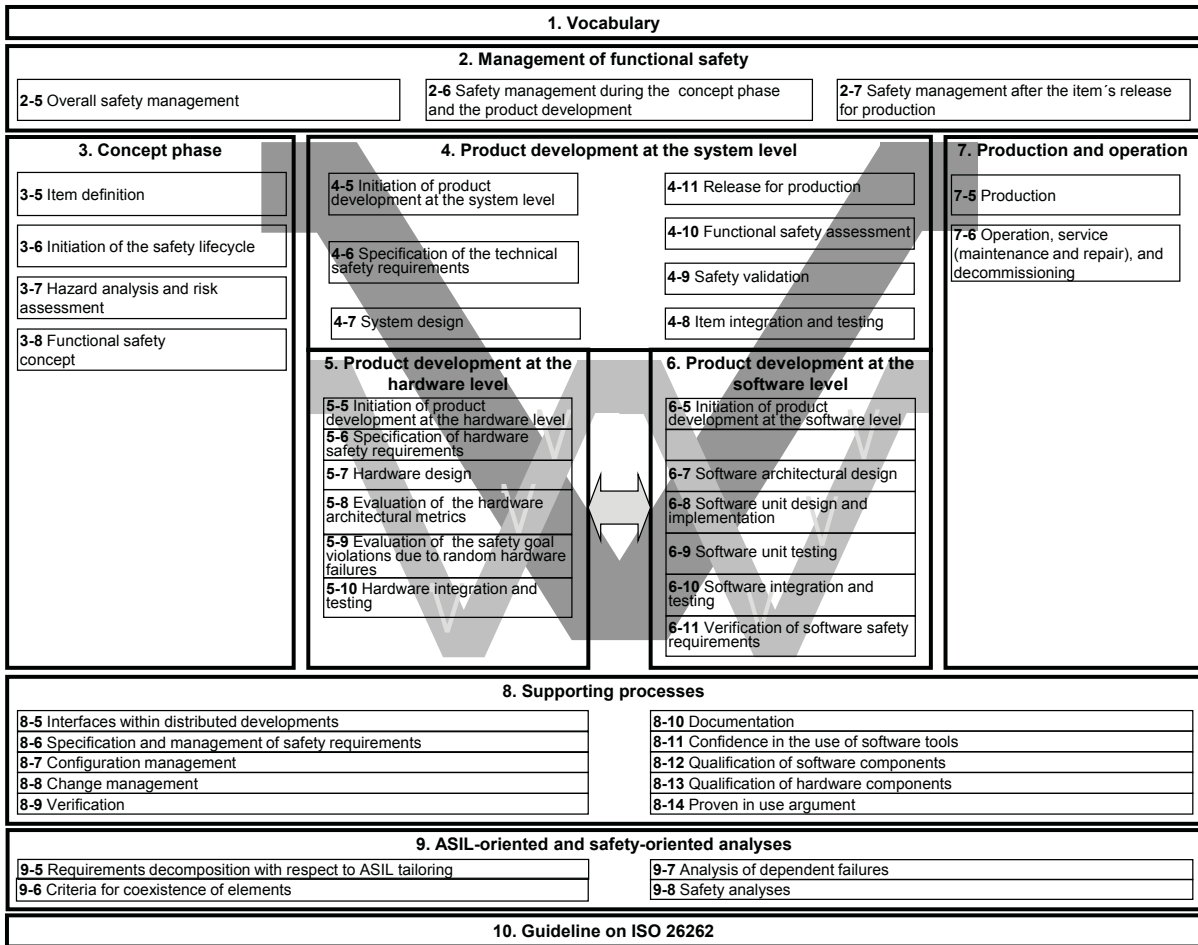


Abbildung 6: Überblick ISO 26262

2.7 INCA

„INCA-Softwareprodukte werden bei der Entwicklung und beim Test von Steuergeräten sowie bei der Validierung und Kalibrierung von elektronisch gesteuerten Systemen im Fahrzeug, am Prüfstand oder in einer virtuellen Umgebung am PC eingesetzt.“ [ETA18e]

INCA ist das Softwaretool der ETAS GmbH, welches den, für den Applikateur nutzbaren, Kalibrierzugriff sicherstellt. Über die in Kapitel 2.2 angesprochenen Zugriffsvarianten ermöglicht es INCA dem Applikateur in sogenannten *Experimenten* mit den verschiedenen Größen der ECU zu arbeiten.

In einem *Experiment* kann sich der Applikateur Verstellkenngößen, -linien und -felder anzeigen lassen und diese verändern, während er sich Messgrößen anzeigen lässt oder diese aufzeichnet. Ein *Experiment* ist Teil eines *Workspaces*, zudem die *Hardwarekonfiguration* mit dem *Projekt* gehört. Das *Projekt* beinhaltet die in Kapitel 2.8 genauer angesprochenen Softwareartefakte für die aktuelle Software, die auf der ECU läuft. Im *Workspace* kommen dann *Projekt*, *Hardwarekonfiguration* und *Experiment* zusammen und nachdem dort die verwendete Hardware (ECU intern und extern) ausgewählt wurde, kann das *Experiment* gestartet werden.

Zudem bietet INCA Funktionalitäten zum Flashen der ECU und zur Speicherseitenverwaltung. Abbildung 7 zeigt die Möglichkeiten der Speicherseitenverwaltung. INCA gewährleistet, dass die Speicherseiten auf dem Rechner und der ECU immer auf demselben Stand sind. Um die Stände der Speicherseiten vergleichen zu können, berechnet

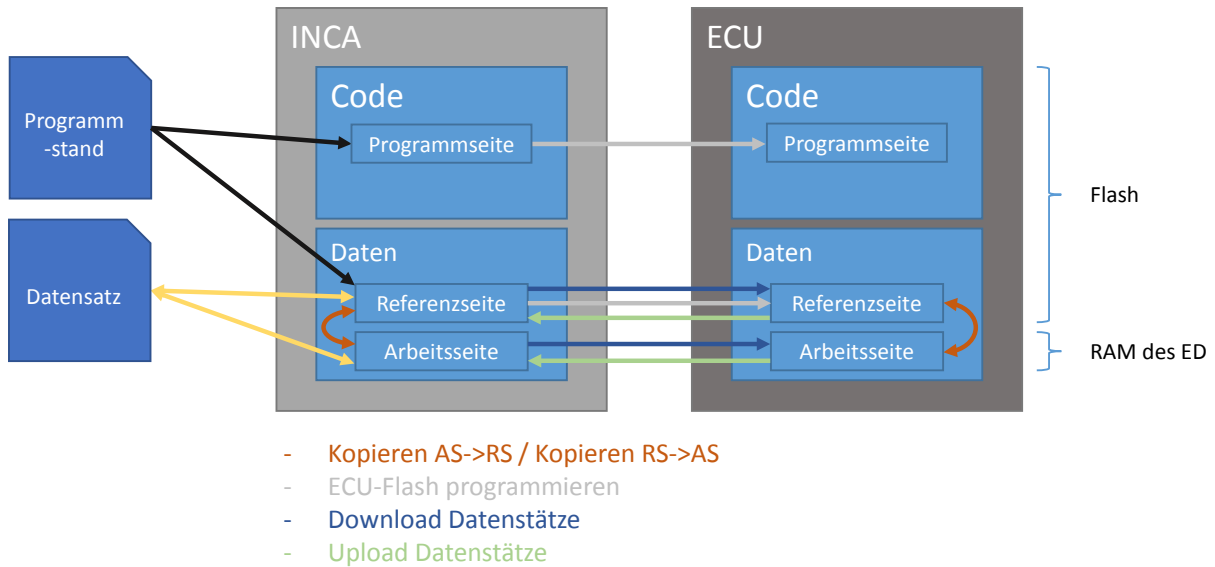


Abbildung 7: Speicherseitenverwaltung in INCA

INCA für alle sechs Speicherseiten (je drei für INCA und ECU: Programmseite, Referenzseite, Arbeitsseite) Prüfsummen⁹. Durch Vergleichen dieser Prüfsummen ist es möglich, Unstimmigkeit zwischen dem Stand in INCA und der ECU zu erkennen. Läuft bspw. auf der ECU ein anderer Programmstand als der, den der Nutzer in INCA geladen hat, ist mindestens die Prüfsumme für die Programmseite zwischen INCA und ECU unterschiedlich. Außerdem lassen die Prüfsummen Rückschlüsse darüber zu, ob an den Werten der Verstellgrößen der Arbeitsseite Veränderungen gegenüber der „originalen“ Werte vorgenommen wurden, wie sie auf der Referenzseite vorhanden sind.

2.8 ECU-Test

„ECU-Test ist eine Testautomatisierungs-Software für die Validierung eingebetteter Systeme im Automotive-Umfeld.“ [Tra18]

Abbildung 8 zeigt einen Screenshot als Beispiel für die Benutzeroberfläche.

Eine Testsuite, in ECU-Test als Projekt bezeichnet, besteht aus einer Testkonfiguration, einer Testbenchkonfiguration und den einzelnen Testfällen, welche in sogenannten Packages definiert werden.

Die Testkonfiguration enthält diverse Einstellungen und Metadaten. Dazu gehören u.a. der Windows-Benutzer, der den Test durchgeführt hat, und die Option, dass nach einem fehlgeschlagenen Testfall der Test abgebrochen oder fortgeführt werden soll. Ebenfalls Teil der Testkonfiguration sind Referenzen auf die Softwareartefakte des aktuellen Softwarestands sowie globale Definitionen von Konstanten, wie z.B. von Arbeitsplatzpfaden. Zu den Softwareartefakten gehören:

- Eine *.hex-Datei, welche das eigentliche kompilierte Programm, das auf der ECU läuft, beinhaltet.
- Eine *.a2l-Datei (auch ASAP2 genannt), wobei es sich um eine Beschreibungsdatei handelt, die in menschenlesbarem Text (ASCII-codiert) alle steuergeräteinternen

⁹die Prüfsummen werden über eine Hashfunktion gebildet

Größen per symbolischen Namen für den Zugriff bekannt macht. Sie beinhaltet sowohl die Ablagestruktur der Größen sowie deren Speicheradressen, Datentypen, Umrechnungsvorschriften und physikalische Einheiten.

- Eine *.elf-Datei welche nötig ist, falls der Testaufbau einen Debugger beinhaltet. Das Executable and Linking Format beinhaltet Informationen im sogenannten DWARF Format. Diese ermöglichen es dem Debugger Verbindungen zwischen originalem Hochsprachenquelltext und Maschinencode herzustellen. Somit sind für den Debugger z.B. die Speicherbereiche von Variablen erreichbar oder Breakpoints im C-Code können mit dem Maschinencode verknüpft werden.
- Eine *.db3-Datei (local Workspace), welche eine SQLite¹⁰-Datenbank beinhaltet. Diese beschreibt alle im Softwarestand enthaltenen Funktionsbausteine und Quelldateien sowie deren Versionen und Revisionen.

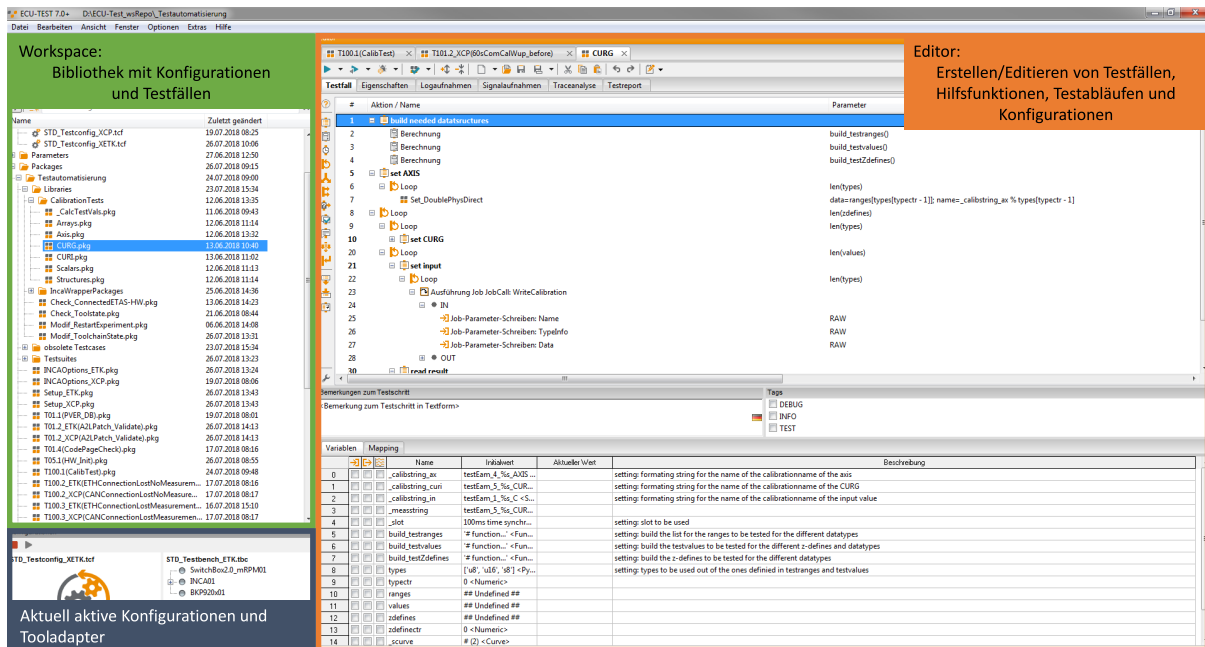


Abbildung 8: ECU-Test

Innerhalb der Testbenchkonfiguration werden die benutzten Tools definiert. Im Falle dieser Testautomatisierung wären das bspw. INCA, die Switchbox, das Netzteil und der Applikationszugriff über externe Calibration-HW via ETK (Toolkette).

Packages beinhalten die Abläufe, Testskripte, der einzelnen Testfälle, die in einer grafischen Programmiersprache definiert werden. Hierbei lassen sich Packages innerhalb von anderen Packages als Funktionen mit Argumenten und Rückgabewerten aufrufen, wodurch sich der Ablauf besser strukturieren lässt.

ECU-Test kann zudem an vielen Stellen durch Python-Skripte erweitert werden. So können z.B. neue/zusätzliche Tools und deren Funktionen durch das Implementieren einer abgeleiteten Klasse der Basisklasse „Tooladapter“ ergänzt werden. Kapitel 5.1 geht auf diesen Vorgang genauer ein.

¹⁰SQLite ist ein dateibasiertes Datenbanksysteme, das sich direkt in Programme integrieren lässt, ohne zusätzlichen SQL-Server

2.9 Python

„Python ist eine portable, interpretative, objektorientierte Programmiersprache.“ [Wei13] Die interpretative Eigenschaft der Sprache ermöglicht die Nutzung der Python-Programme (Skripte) auf verschiedenen Systemplattformen (Unix, Windows, Mac OS). Der starke Abstraktionsgrad der Sprache ermöglicht sehr kompakte Programmtexte, die leicht zu erfassen sind. Sowohl objektorientiertes als auch funktionales oder imperatives Programmieren werden unterstützt. In sogenannten Modulen (vergleichbar mit Bibliotheken in C/C++, C# oder Java) können Teile von Python-Programmen aufgeteilt und wiederverwendet werden. Durch eine Vielzahl von Standardmodulen und Erweiterungsmodulen ist Python sehr flexibel und vielseitig einsetzbar. [Pyt18]

Es gibt aktuell zwei unterschiedliche Sprachvarianten von Python: Python 2 und Python 3. Mit dem 2007 erschienenen Python 3 entschied man sich einige grundlegenden Designschwächen von Python zu beseitigen. In dieser Arbeit wird hauptsächlich, aufgrund der Kompatibilität mit ECU-Test, Python 2.7 genutzt.

2.10 Testautomatisierung

Um die Plattformfreigabe zu erleichtern wurden bereits vor dieser Arbeit einige Mechanismen, Hardware und Software, zur Testautomatisierung durch die PS-EC/EBT3 umgesetzt bzw. implementiert.

Der Testaufbau mit einer Applikationstoolkette außerhalb eines Testfahrzeuges benötigt eine 12V Versorgung. Eine sogenannte „Switchbox“¹¹ ermöglicht einer Automatisierung, über USB verbunden, diverse Steuersignale einer ECU zu stimulieren (Versorgungsspannung an/aus, Zündung an/aus, aktuelle Motordrehzahl, ...).

Durch diese Hardwareelemente und die COM-API, die INCA zur Verfügung stellt, wurden bereits einzelne Testfälle provisorisch umgesetzt, die sich manuell nicht durchführen lassen. In Kapitel 3.2 wird genauer auf diese bereits vorhandenen Lösungen eingegangen und geprüft inwiefern sie weiter verwendet werden sollen oder können.

¹¹im Wesentlichen ein Arduino-Mikrocontroller mit diversen Schaltern, Relais und einem Drehencoder für die Drehzahlwahl

2.10.1 Testaufbau

Zur Entwicklung und Erprobung der Testautomatisierung ist ein Hardware- und Software-Aufbau nötig. Vorerst wird sich hierbei auf einen fixen Aufbau beschränkt. Dieser Aufbau beinhaltet ein Gleichspannungsnetzteil, eine Switchbox, ein Steuergerät mit ED, ein ETK, ein Steuergeräte- und Bus-Schnittstellenmodul sowie den Prüfrechner (vgl. Abbildung 9).

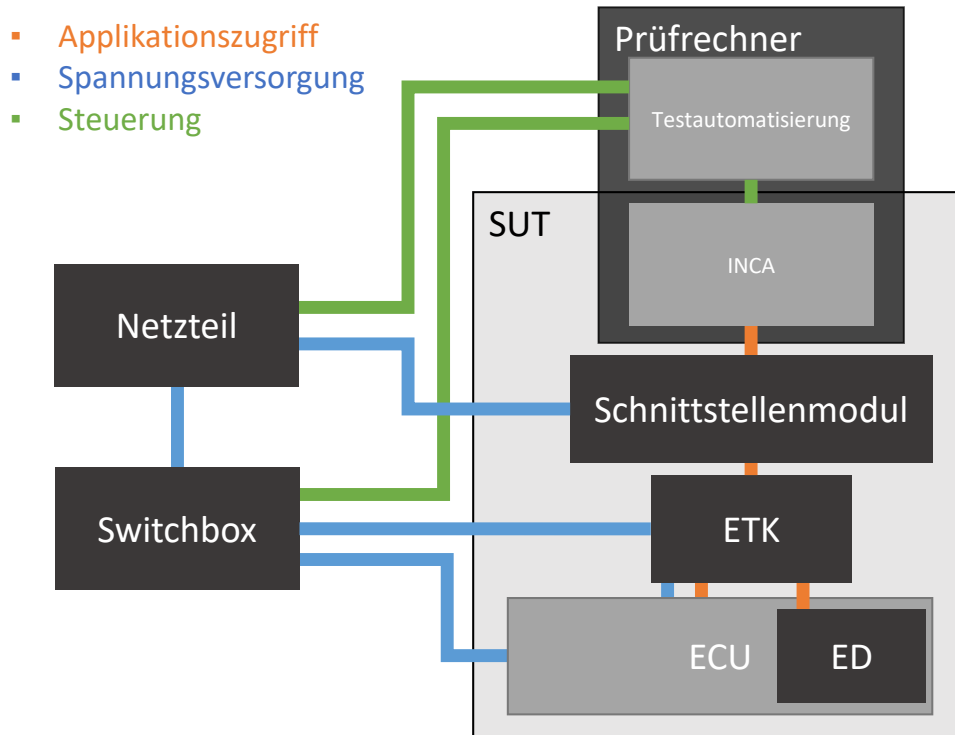


Abbildung 9: Testaufbau

Netzteil Das verwendete Netzteil ist ein BK Precision 9205 Multi-Range Programmable DC-Power Supply [BK 18]. Ein Adapter verbindet die RS232-Schnittstelle des Netzteils per USB mit dem Testrechner. Dies ermöglicht eine Fernsteuerung des Netzteils über serielle Kommunikation. Um das Netzteil zu steuern werden über diese Verbindung ASCII-codierte Anweisungen gesendet, welche durch die IEEE 488.2 genormt sind.

Switchbox Die Switchbox (siehe Abbildung 10) bietet die Möglichkeit diverse Steuersignale der ECU zu stimulieren. Hierzu wird die Switchbox per USB mit dem Testrechner verbunden und kann über serielle Schnittstelle (COM) angesprochen werden. Die Steueranweisungen sind proprietäre ASCII-codierte Strings. Relevante Steuersignale, die die Switchbox steuern kann sind:

- T15 = an/aus: Die Klemme bzw. das Terminal, welche/s im Serienfahrzeug mit der Zündung verbunden ist.
- T30 = an/aus: Die Klemme bzw. das Terminal, welche/s im Serienfahrzeug die Spannungsversorgung von u.a. der ECU an- und abschaltet.

- Drehzahl = 0...8000: Die Drehzahl, welche der ECU von der Kurbelwelle und der Nockenwelle zurückgemeldet wird.
- HR = an/aus: Das Hauptrelais (in Abbildung 10 als „Aux. Relay“ bezeichnet), welches normalerweise von der ECU selbst gesteuert wird, von der Switchbox aber überschrieben oder abgefragt werden kann. Über das Hauptrelais schaltet die ECU bei Fahrzeugstart (T15 und T30 an) die Bordelektronik an.

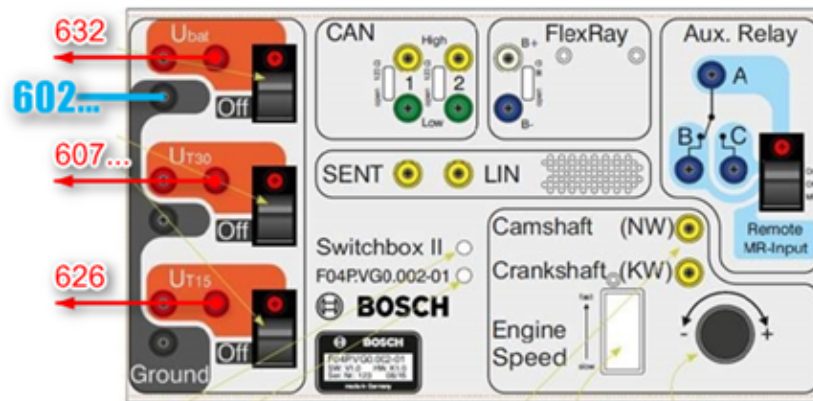


Abbildung 10: Switchbox (Quelle: Intranet Bosch)

Toolkette Die in diesem Testaufbau verwendete Toolkette beinhaltet ein ETK-System für den Applikationszugriff, wie es in Kapitel 2.2 erläutert wurde. Der verwendete ETK ist ein ETK-S21.1 [ETA18d] und das Schnittstellenmodul ein ES595 [ETA18c]. Zudem enthalten sind der Testrechner und das Steuergerät mit ED.

Für den XCP Testaufbau muss in der Toolkette lediglich das Schnittstellenmodul gewechselt und das ETK von Schnittstellenmodul und ECU getrennt werden. Als Schnittstellenmodul für XCP wird das ES523 [ETA18b] genutzt, da es, anders als das ES595, neben traditionellem CAN auch CAN FD unterstützt.

3 Konzeption

Dieses Kapitel beinhaltet die Use-Case-Analyse (Kapitel 3.1) und Ist-Stand-Analyse (Kapitel 3.2). Sie stellen die Grundlage des weiteren Vorgehens dar und lassen sich im V-Modell als Teil der „System- Anforderungs- Analyse und Entwurf“ einordnen.

3.1 Use-Case-Analyse

Ein Use-Case bezieht sich auf einen Anwendungsfall, in welchem ein Test durchgeführt werden soll. Diese Anwendungsfälle dienen als Einstieg des Testdesigns, da sie den Anlass für die Testerstellung angeben.

Die Applikationstoolkette kann, wie in Kapitel 2.2 erläutert, in unterschiedlichen Varianten auftreten. Da sich diese Varianten in zwei Hauptgruppen einteilen lassen, stellen diese die ersten zwei Use-Cases dar: der Applikationszugriff über einen ETK und der Zugriff über XCP.

Des Weiteren ist das Flashen ein wichtiger Vorgang, der von der Applikationstoolkette abhängig ist. Daher wird dieser als dritter Use-Case aufgenommen.

3.1.1 Use-Case 1: Freigabe ETK-System

Der erste Use-Case ist die Freigabe eines ETK-Systems für die Applikation. Diese Freigabe erfolgte bis jetzt entsprechend einer Freigabecheckliste in Excel. Diese Freigabecheckliste fordert von dem System im Wesentlichen, dass es:

- in INCA automatisch erkannt wird,
- Flashen ermöglicht,
- CalibrationWakeup¹² (CalWup) unterstützt,
- Speicherseitenverwaltung zulässt,
- Verstellen und Messen ermöglicht,
- definierte Resetverhalten aufweist und
- Dauerlaufmessungen fehlerfrei aufzeichnen kann.

3.1.2 Use-Case 2: Freigabe XCP-System

Die Freigabe eines XCP-Systems läuft ähnlich der eines ETK-Systems ab. Neben dem Hardwareaufbau der Toolkette unterscheidet sich die Freigabe eines XCP-Systems nur durch einzelne Anforderungen, die ebenso in einer Freigabecheckliste dargestellt sind. Flashen und Dauermessungen sind in der Freigabecheckliste für XCP-Systeme nicht gefragt. Die XCP-System-Freigabe setzt dafür ein definiertes Verhalten bei Leitungsunterbrechung voraus, sowie Datenerhalt, falls ein dauerversorgtes ED verbaut ist.

¹²das Steuergerät wird „geweckt“, ohne dass die Zündung des Fahrzeuges aktiv sein muss (T30 an, T15 aus)

3.1.3 Use-Case 3: Flashvorgang

Um eine ECU zu Flashen ist eine Speicherbereichszuweisung erforderlich. Die Zuweisung erfolgt anhand von Informationen aus der teilweise automatisiert generierten ProF-Konfiguration. Daher soll diese Konfiguration in Zukunft ebenfalls einem automatisierten Test unterzogen werden.

3.1.4 Zusammenfassung und Bewertung

Bereits die Erfassung und Analyse der vorgegebenen drei Use-Cases lässt eine Optimierung der Struktur der Freigabeprozesse zu.

Ein reines XCP-System ermöglicht kein Flashen. Der für die Freigabe zu prüfende Punkt des Flashvorgangs in ETK-Systemen lässt sich somit nicht auf Use-Case 2 übertragen. Das heißt, dass der dritte Use-Case, in Bezug auf die geplante Automatisierung, immer ein ETK-System nutzen muss. Use-Case 3 ordnet sich also Use-Case 1 unter. Bei jeder Freigabe eines ETK-Systems ist der Test des Flashvorgangs, und die damit verbundene Überprüfung der ProF-Konfiguration, bereits gegeben. Falls sich in einem ETK-System lediglich die ProF-Konfiguration ändert, kann ein stark vereinfachter Test lediglich die davon abhängigen Testfälle erneut prüfen (Regressionstest).

Dieser Zusammenhang lässt sich auch innerhalb eines Use-Case Diagramms darstellen. Abbildung 11 zeigt dieses. Um die ProF-Konfiguration freigeben zu können ist eine Freigabe des entsprechenden ETK-Systems nötig. Umgekehrt gehört der Test der ProF-Konfiguration zur Freigabe des ETK-Systems. Die Freigabe eines ETK-Systems erweitert die Freigabe des XCP-Systems, da auf ein ED praktisch immer via XCP zugegriffen werden kann, während der Zugriff über einen ETK optional hinzukommt. Für den Kunden ist am Ende die Freigabe des XCP- und des ETK-Systems relevant, wobei die Freigabe der ProF-Konfiguration implizit enthalten ist. Der Tester testet jedoch teilweise alle drei Use-Cases getrennt.

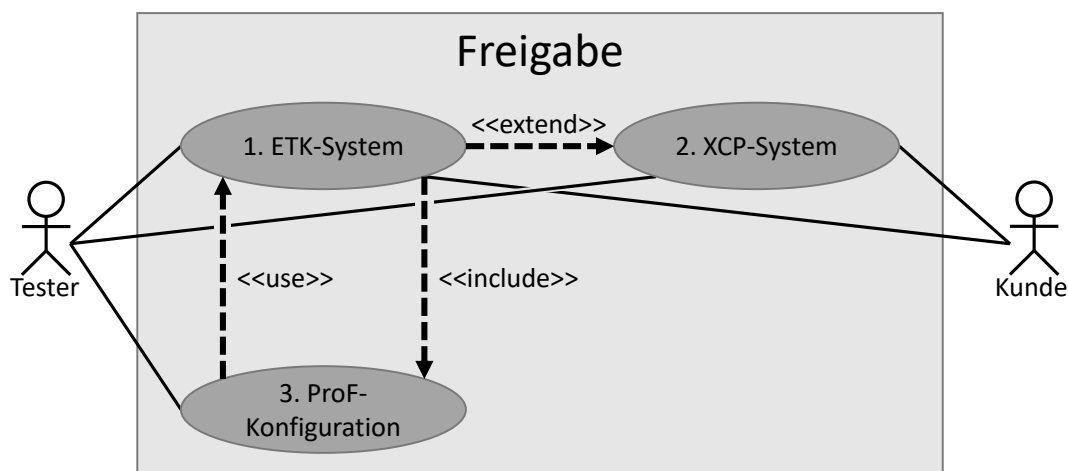


Abbildung 11: Use-Case Diagramm

Weiter können anderen Unterschiede zwischen der Freigabe eines ETK- und XCP-Systems auf Harmonisierbarkeit untersucht werden. So zeigen sich im aktuellen Stand keine Argumente, die dagegen sprechen Tests bezüglich Dauermessungen, Datenerhalt und Leitungsunterbrechung in Use-Case 1 und Use-Case 2 durchzuführen. Da die meisten Anforderungen der Use-Cases sich aus den Testfalllisten der Freigabechecklisten ableiten, die

wiederum zum Großteil allein auf Erfahrungsbasis entwickelt wurden, ist ein Austausch von Anforderungen zwischen den beiden Use-Cases eine sinnvolle Erweiterung der Tests.

Aus den genannten drei Use-Cases ergeben sich zwei Systeme die getestet werden sollen. Fortan wird also nur noch von den ersten beiden Use-Cases gesprochen und der Use-Case Flashvorgang als Teil des Ersten betrachtet.

3.2 Ist-Stand-Analyse

Die Ist-Stand-Analyse beschäftigt sich mit bereits bestehenden Systemen zur Freigabe, manuell oder automatisiert. Es wird auf die unterschiedlichen, bereits bestehenden Tools eingegangen, die für die Freigabe und das Testen genutzt werden, auf Ablaufdefinitionen in den Freigabechecklisten und auf die bestehenden Dokumentationssysteme.

3.2.1 Toolsysteme

Für die Freigabe von ETK-Systemen und XCP-Systemen sind zwei Personen zuständig. Dies hat dazu geführt, dass zwei Tools entstanden sind, die bei den Tests für die Freigabe mit verwendet werden.

ETK-Tester Die erste so entstandene Lösung ist der sogenannte *ETK-Tester*. Ein Tool welches mit C#¹³ unter Verwendung von WPF¹⁴ entwickelt wurde. Abbildung 12 zeigt einen Screenshot des Tools.

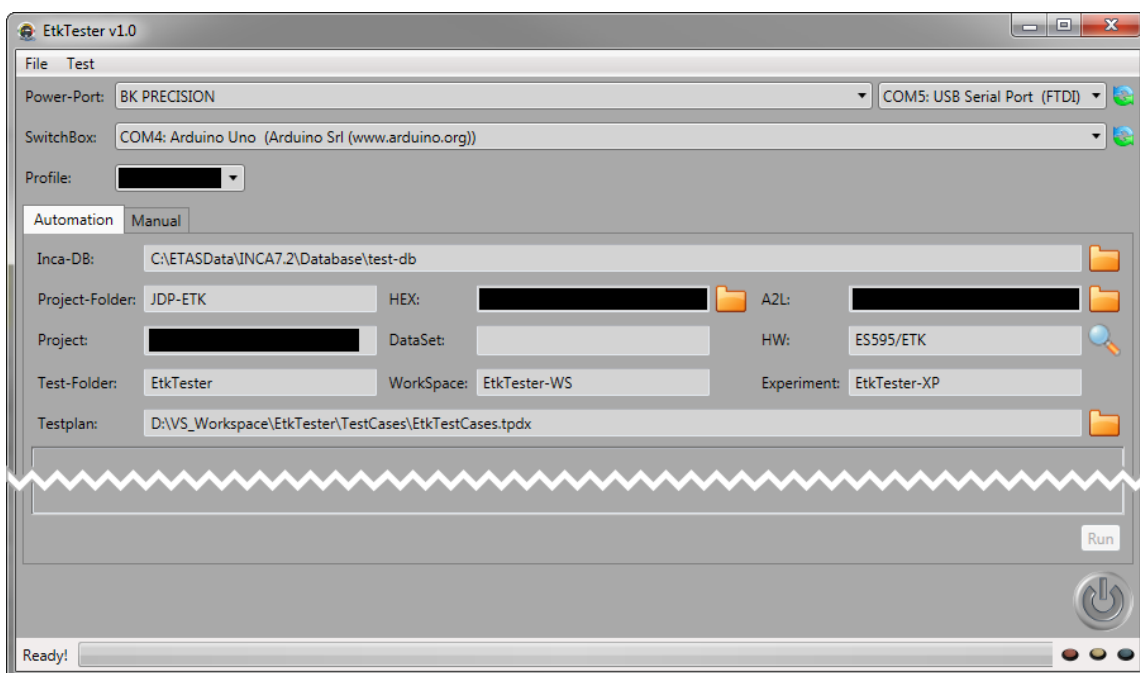


Abbildung 12: ETK-Tester

¹³C# ist eine von Microsoft entwickelte objektorientierte Programmiersprache, welche Teil des .NET-Frameworks ist, und sich stark an C++ orientiert

¹⁴Windows Presentation Foundation (WPF) ist ein Framework zur Entwicklung von grafischen Benutzeroberflächen

Das Tool bietet die Möglichkeit sich mit der Switchbox und einem Netzteil zu verbinden sowie die INCA-API zu nutzen. In einem Testplan können im XML¹⁵-Format Testfälle angegeben werden, die geprüft werden sollen. Die Testskripte dieser Testfälle müssen vorher ebenso in XML-Dateien definiert werden. Während das Tool die Switchbox und das Netzteil in fast ihrem ganzen Funktionsumfang steuern kann, ist das bei der INCA-API nicht der Fall. Viele Testfälle lassen sich aufgrund dieser Beschränkung nicht durch das Tool umsetzen. Es bietet hauptsächlich die Funktion das Verstellen zu prüfen. Dabei wird automatisiert eine Vielzahl von Verstellgrößen unterschiedlicher Datentypen beschrieben und aus Referenzmessgrößen zurückgelesen. Somit kann ein erfolgreiches Verstellen verifiziert werden.

VBA-Teilautomatisierung Die Freigabe von XCP-Systemen wird durch eine VBA-Teilautomatisierung¹⁶ unterstützt. Sie implementiert ebenfalls die INCA-API und den Zugriff auf die Switchbox sowie die Möglichkeit *.a2l-Dateien zu durchsuchen und abzuändern. Des Weiteren wird in der Teilautomatisierung die SQLite-Datenbank durchsucht, die Informationen zum verwendeten Programmstand beinhaltet. Die Teilautomatisierung deckt allerdings nicht alle Testfälle ab, die die momentane Freigabecheckliste für XCP-Systeme beinhaltet. Sie weist zudem aufgrund ihres ungleichmäßigen Wachstums eine unübersichtliche Struktur auf. Viele Testfälle sind noch nicht umgesetzt bzw. erfordern Zugriff auf das Netzteil, den die Implementierung nicht hat.

ECU-Test Wie bereits in Kapitel 2.8 beschrieben, bietet ECU-Test ein Framework für Tests an ECUs, welches sich in anderen Abteilungen der Robert Bosch GmbH bereits bewährt hat. ECU-Test wird dort allerdings für HIL- und SIL-Tests genutzt, während im Falle dieser Arbeit eher von einem Hardware Integration Test (HIT) gesprochen werden muss. Durch die Erweiterbarkeit von ECU-Test in Python ist die Möglichkeit gegeben fehlende Funktionen zu ergänzen, die für diese Arbeit nötig sind. Ein Qualifizierungs-Prozess für ISO 26262 ist zudem durch ein Qualifizierungs-Kit möglich. Somit kann eine Testautomatisierung mit festgelegten und dokumentierten Anwendungsfällen entsprechend ISO 26262 qualifiziert werden.

3.2.2 Ablaufdefinition

Durch die Analyse der bereits bestehenden Tools hat sich gezeigt, dass für eine Automatisierung drei Arten der Ablaufdefinition möglich wären.

Hardcoded Die VBA-Teilautomatisierung realisiert einen Testablauf, der so direkt im VBA-Quellcode hinterlegt ist. Eine gewisse Struktur ist dadurch realisiert, dass einzelne Testfälle in Funktionen implementiert sind. Um also nur bestimmte Testfälle auszuführen, müssen deren Funktionsaufrufe im Quellcode deaktiviert werden. Wenn der gesamte Testablauf so manipuliert wird kann es allerdings passieren, dass gewisse Vorbedingungen für Testfälle nicht mehr gegeben sind, die vorher durch den vorangehenden Testfall erreicht wurden. Ein solcher „hart“ codierter Ablauf ist zudem schwer zu erweitern, wenn sich zusätzliche Testfälle in den bestehenden Ablauf eingliedern sollen.

¹⁵Extensible Markup Language (XML) ist eine Auszeichnungssprache, die so ausgelegt wurde, dass sie menschen- und maschinenlesbar ist

¹⁶Visual Basic for Applications (VBA) ist eine Skriptsprache von Microsoft, welche in die meisten Microsoft Office-Produkte integriert ist, um diese erweitern zu können

Dateibasiert Das *ETK-Tester* Tool nutzt eine dateibasierte Definition des Ablaufes. Hierbei werden sowohl die Testskripte der einzelnen Testfälle, als auch der Testablauf, in XML-Dateien definiert. Es ist also möglich ohne Programmierkenntnisse Testfälle und Testabläufe zu editieren. Zudem können mehrere Abläufe durch mehrere Ablaufdateien hinterlegt werden. Ein sicheres Umgehen mit Vor- und Nachbedingungen einzelner Testfälle ist nicht gegeben. Es liegt in der Verantwortung des Testfall-Autors dafür zu sorgen, dass der Testfall keine Vorbedingung voraussetzt, die durch andere Testfälle erzeugt werden oder in der Verantwortung des Testablauf-Autors, die Testfälle nur so anzuordnen, dass die Vorbedingungen jedes Testfalles durch den/die Vorhergehenden gegeben sind. Die Testskripte neuer Testfälle können als neue Datei angelegt werden und in der Testablauf-Datei als Teil des Ablaufes integriert werden. Da eine grafische Benutzeroberfläche fehlt, erschwert das die Pflege und Erstellung der einzelnen Testfälle.

Toolgestützt In ECU-Test werden einzelne Testfälle in sogenannten Packages (Testskripte laut ISTQB) realisiert (vgl. Kapitel 2.8). Einzelne Packages können auch durch andere Packages aufgerufen werden. Sogenannte Projekte definieren Testsuiten um den Testablauf und die Testumgebung festzulegen. Diese Strukturierung ermöglicht es, dass aus einer gegebenen Testfallbibliothek (eine Bibliothek mit Packages) und Bibliothek für Testbenchkonfigurationen und Testkonfigurationen die benötigte Testsuite zusammengestellt wird. Diese Strukturierung ermöglicht es eine Vielzahl von Tests bzw. Testsuiten anzulegen, sofern alle benötigten Testfälle und Konfigurationen in den entsprechenden Bibliotheken erstellt und angelegt wurden. Um definiertes Verhalten bei bestimmten Vor- und Nachbedingungen zu erreichen ist eine Strukturierung in „Preconditions“ und „Postconditions“ innerhalb von Packages möglich. Damit wird erreicht, dass jeder Testfall seine eigenen Vorbedingungen selbst herstellt oder der Testablauf-Autor darauf hingewiesen wird, dass der von ihm definierte Ablauf so ungültig ist. Designfehler werden damit vermieden. Durch die Erweiterung der Bibliotheken für Packages und Konfigurationen ist es möglich, neue Testfälle und Testumgebungen einzupflegen.

3.2.3 Dokumentationssystem

Das Dokumentationssystem bezieht sich sowohl auf die Dokumentation von Tests, Testfällen und Testsuiten, als auch auf die Dokumentation der Ergebnisse eines Testes, den Testreports.

Momentan wird für die Freigabe von ETK-Systemen und XCP-Systemen ein Excel-Dokument genutzt, die „Freigabecheckliste“ (vgl. Abbildung 13). In dieser sind Metadaten zu verwendeter Hardware, Software, Tester, Ort, Datum, Programmstand usw. enthalten sowie die einzelnen Testobjekte und deren Testitems mit den entsprechenden Testfällen gelistet. Somit dient sie sowohl als Dokumentation für den Test selbst als auch für das Ergebnis, den Report. Diese Liste wird abgearbeitet und die einzelnen Testfälle als i.O. oder fehlerhaft markiert. Ein Teil dieser Fälle wird im Falle der XCP-Freigabecheckliste durch die hinterlegten VBA-Skripte geprüft und die entsprechenden Markierungen gesetzt. Nicht automatisierte Testfälle müssen manuell getestet und markiert werden. Das gilt auch für die Freigabe des ETK-Systems, wobei hier keine VBA-Skripte genutzt werden sondern einige Testfälle durch das *ETK-Tester* Tool getestet werden können, deren Ergebnisse aber von Hand in das Excel-Dokument eingetragen werden müssen.

12	Speicherseitenverwaltung bei KI.15 aus	
12.1	Werden Checksummen als gleich erkannt	
12.2	Läuft Programm	
12.3	Download von Datensatz nach AS+RS erfolgreich	
12.4	Kopieren Daten AS nach RS	
12.5	Kopieren Daten RS nach AS	
12.6	Upload durchführen (ECU -> AS, RS) ☐ Alle Seiten o.k.?	
12.7	Datenerhalt nach T30 ON/OFF/ON	

Abbildung 13: Ausschnitt aus der Freigabecheckliste für ETK-Systeme

Viele, der in den Excel-Dokumenten genannten Testfälle, sind dort zudem nur kurz, lückenhaft oder unvollständig definiert. Eine andere Dokumentation für die Testfälle gibt es derzeit nicht.

3.2.4 Zusammenfassung und Bewertung

Die Testautomatisierung durch VBA-Skripte für den Use-Case „Freigabe XCP-System“ ist durch den im Quellcode hinterlegten Ablauf zu starr, um sie sinnvoll fortzuführen und für den Use-Case „Freigabe ETK-System“ anzupassen. Allerdings beinhalten die VBA-Skripte bereits einige komplexere Testfälle, die hauptsächlich die INCA-API nutzen. Diese können später als Vorlagen und Beispiele dienen.

Das *ETK-Tester* Tool ist im Prinzip ein bereits abgeschlossenes/fertiges Tool, welches nur durch größere Änderungen so angepasst werden könnte, dass es die noch fehlenden Testfälle abdecken kann. Wie auch die VBA-Skripte können Teile des Quellcodes des *ETK-Tester* Tools ggf. später in einem neuen Tools wiederverwendet werden bzw. als Vorlagen und Beispiele dienen.

Ein Framework wie ECU-Test stellt eine sinnvolle Basis für die Testautomatisierung dar. Gegenüber einem selbst entwickelten Tool ist die Robustheit und Funktionalität eines solchen Frameworks schon über Jahre hinweg erwiesen worden. Das bestätigt u.a. auch das Qualifizierungs-Kit für ISO 26262, welches für ECU-Test zur Verfügung steht. Ein weiterer Vorteil ist der reduzierte Zeitbedarf für die Umsetzung der Testautomatisierung. Die Entwicklung eines eigenen Framework-Tools würde den Großteil der Zeit dieser Arbeit beanspruchen. ECU-Test als Framework zu nutzen erlaubt es mehr Zeit mit der detaillierten Aufarbeitung der Testfälle zu verbringen, sowie mehr Testfälle umzusetzen.

Die toolgestützte Ablaufdefinition, die ECU-Test ermöglicht, bietet alle Vorteile der datenbasierten Definition gegenüber der „Hardcoded“ Vorgehensweise. Sowohl die Abläufe einzelner Testfälle, die Testskripte, als auch die Testabläufe lassen sich ohne Programmierkenntnisse schnell festlegen und ändern. Ein Einarbeiten in die INCA-API oder die

serielle Schnittstelle für Netzteil oder Switchbox ist für den Autor von Testskripten und Testabläufen nicht nötig. Der zusätzliche Vorteil der Ablaufdefinitionen in ECU-Test gegenüber den Dateibasierten ist, dass der Autor der Testskripte und Testabläufe die Syntax und die Semantik nicht kennen muss, nach denen dateibasierte Ablaufdefinitionen aufgebaut wären.

Als Anforderung für die Testautomatisierung, liegen lediglich die in Excel formulierten Freigabechecklisten vor. Diese wiederum basieren auf Systemanforderungen und Erfahrungen. Die Dokumentation und die Herleitung der Testfälle aus diesen Systemanforderungen und Erfahrungen ist nicht durchgängig bzw. zum Großteil gar nicht vorhanden. Eine neue ausführlichere Dokumentation für Testfälle soll umgesetzt werden. Hierzu bietet es sich an die Testfälle allgemein entsprechend IEEE 829 (ggf. angepasst) zu dokumentieren und die genauen Abläufe einzelner Testfälle, Testskripte, in ihrer Implementierung als ECU-Test-Package genau zu kommentieren.

Wird in ECU-Test ein Testablauf gestartet wird am Ende automatisch einen Report generiert. Dieser kann in Form einer simplen Checkliste als z.B. HTML-Datei ausgegeben werden. Zudem wird ein ausführlicher Report in einem proprietären Format abgelegt. Der ausführliche Report hält genaue Informationen zu jedem einzelnen Testschritt, der einzelnen Testskripte, bereit. Um das proprietäre Format zu lesen ist das entsprechende Tool notwendig.

Mit der Wahl von ECU-Test als Framework für die Testautomatisierung wird also auch die Dokumentationsstruktur der Tests für die zwei angedachten Use-Cases überarbeitet. Es gibt eine Dokumentation für die einzelnen Testfälle und eine ausführliche Dokumentation der einzelnen Testskripte. Es wird ein Report generiert, der sich als Dokument für die Weitergabe eignet, um die erfolgreiche Freigabe eines XCP-/ETK-Systems nachzuweisen. Zudem wird ein Report generiert, der dem Tester ausführliche Informationen bietet, um ggf. bei der Fehlersuche zu helfen. Ein Nachteil dieser Dokumentationsstruktur ist allerdings, dass bei der Änderung oder Neuerstellung eines Testfalls sowohl die Dokumentation nach IEEE 829 angepasst werden muss, als auch die Dokumentation des entsprechenden Testskriptes in ECU-Test.

Des Weiteren ist aus dem momentanen Stand der Arbeit heraus noch nicht mit abschließender Sicherheit zu sagen, inwiefern sich ECU-Test für HIT eignet. Viele Testfälle ähneln solchen aus HIL oder SIL-Tests, wie sie normal mit ECU-Test realisiert werden. Einige Testfälle erfordern aber bestimmte Testschritte, bei welchen erst untersucht werden muss, inwiefern ECU-Test sie unterstützt bzw. sich dahingehend erweitern lässt.

Die Testautomatisierung für die Freigabe von Toolketten mit XCP- und ETK-Systemen wird als Teil dieser Arbeit deshalb in ECU-Test umgesetzt und die Testfälle entsprechend IEEE 829 dokumentiert.

3.3 Ergebnis

In dieser Arbeit sind zwei Use-Cases für die Testautomatisierung zu betrachten: die Freigabe von ETK-Systemen und die von XCP-Systemen. ECU-Test wird als Software-Framework genutzt, um die Automatisierung umzusetzen und bietet gleichzeitig die Möglichkeit einzelne Testfälle entsprechend zu dokumentieren. Zusätzlich wird eine abstraktere Dokumentation einzelner Testfälle nach IEEE 829 umgesetzt.

Als Quelle für die Testfälle werden die bereits vorhandenen Freigabechecklisten als Anforderungsdokument betrachtet. Die sich so ergebenden Testfälle sollten aber möglichst einzeln betrachtet werden, um sie systematisch zu erweitern oder zu harmonisieren.

Die resultierenden Testfälle werden in einer Testfallbibliothek in ECU-Test umgesetzt. Reports für Ergebnisse des Tests (Testreports) als auch über den Test selbst (Testskripte) sind ebenfalls mit der Hilfe von ECU-Test umzusetzen.

4 Testfallbeschreibung

Dieses Kapitel ordnet sich im V-Modell als „Subsystem- Anforderungs- Analyse und Entwurf“ ein. Testfälle werden erfasst, analysiert und dokumentiert entsprechend der in Kapitel 3 ermittelten Konzepte bzw. basierend auf dem dort aufgearbeiteten Stand.

4.1 Anwendung IEEE 829

Die IEEE 829 beschreibt wie ein Test in unterschiedlichen Ebenen (Levels) dokumentiert werden sollte, die optional in ihrer Nutzung von einer Bewertung des *Integrity Level* des zu testenden Systems abhängig gemacht werden können. Neben Ebenen für die Dokumentation von Testfällen gehören hierzu Ebenen für organisatorische Informationen wie testender Mitarbeiter, Kosten oder eingeplante Zeit. Ebenso gibt es Ebenen für die Testreports und -logs nach der Testausführung. Da solche organisatorischen Informationen bei den zwei angesprochenen Use-Cases oft im Voraus nicht spezifisch festgelegt sind, fallen die entsprechenden Ebenen der IEEE 829 hier aus der Betrachtung. Testreports und Testlogs werden durch ECU-Test bereitgestellt und sind Teil einer späteren Betrachtung (siehe Kapitel 5.2.2). Die Ausarbeitung des *Integrity Level* ist nicht Bestandteil dieser Arbeit, da der zu erwartende Mehraufwand den zur Verfügung gestellten deutlich überschreiten würde.

Relevant für die Testfallbeschreibung sind demnach die Ebenen *Level Test Design* und *Level Test Case*. Basierend auf diesen Ebenen wurde eine Dokumentation für einzelne Testfälle definiert, die sich hauptsächlich an *Level Test Case* orientiert. Die Tabelle 1 beschreibt den Zusammenhang von *Level Test Case* und der definierten Struktur zur Dokumentation eines einzelnen Testfalles.

IEEE 829 - Level Test Case	angepasste IEE 829
Test case identifier	Kennzeichnung
-	Testfalltyp
Objective	Testfallzweck
-	Testfallquelle
-	Testfallanforderung
-	Testobjekt
Special procedural requirements	Testfallvorzustände und Testfallnachzustände
Intercase dependencies	Vorgängertestfälle und Nachfolgetestfälle
Environmental needs	Testumgebung
Inputs	Testfallargumente
Outcomes(s)	Testfallergebnisse
-	Testfallstatus

Tabelle 1: Testfalldokumentation; angepasste IEEE 829

Kennzeichnung (nach IEEE 829 - Level Test Case) Eine eindeutige Kennzeichnung für jeden Testfall. Für diese Kennzeichnung wurde eine fortlaufende Nummerierung gewählt. Zudem beinhaltet sie die Information, ob sich dieser Testfall zwischen XCP-Systemen und ETK-Systemen harmonisieren ließ (d.h. derselbe Testfall kann in beiden Use-Cases genutzt werden) oder nicht.

Testfalltyp Der *Testfalltyp* erweitert die IEEE 829 und dient zur Dokumentation der Art des Testfalles; kann der Testfall automatisiert werden, ist ein manueller Eingriff des Testers nötig oder wird eine automatisierte Umsetzung des Testfalles ausgeschlossen. Entsprechend dieser Einordnung wird das Feld mit den Schlüsselworten: automatisiert, manuell oder mit Nutzereingabe gefüllt.

Testfallzweck (nach IEEE 829 - Level Test Case) Der *Testfallzweck* beschreibt das Ziel des Testfalles bzw. warum dieser Testfall durchgeführt wird und was er beweisen soll.

Testfallquelle Die *Testfallquelle* erweitert die IEEE 829 und soll dokumentieren, woher der Testfall kommt bzw. durch was er gefordert wurde. Da die zu testenden Anforderungen momentan nur durch die Freigabechecklisten und mündlich definiert werden, kann in diesem Punkt zukünftig genauer nachvollziehbar hinterlegt werden, woher ein bestimmter Testfall kam, bzw. auf welche Anregung hin er entstand.

Testfallanforderung (nach IEEE 829 - Level Test Design) Dieser Dokumentationspunkt ergibt sich laut IEEE 829 aus der übergeordneten Ebene *Level Test Design* wird hier aber direkt in die Testfalldokumentation integriert. Die *Testfallanforderung* beschreibt, welche Anforderung an das System den Testfall nötig macht. Mehrere Testfälle können sich aus derselben Anforderung entwickeln, unterscheiden sich dann aber in einer genaueren Beschreibung des *Testfallzweckes*.

Testobjekt (nach IEEE 829 - Level Test Design) Dieser Dokumentationspunkt ergibt sich laut IEEE 829 aus der übergeordneten Ebene *Level Test Design* wird hier aber direkt in die Testfalldokumentation integriert. Das Testobjekt beschreibt ein bestimmtes System oder eine Komponente die bei diesen Testfall getestet wird. (vgl. ISTQB Definition 2.3.1)

Testfallvorzustände und Testfallnachzustände (nach IEEE 829 - Level Test Case) Die *Testfallvorzustände* beschreiben, in welchem Zustand sich das zu testende System (SUT) oder Objekt befinden muss, damit der Testfall ausgeführt werden kann. Die *Testfallnachzustände* beschreiben den Zustand des SUT oder Objektes nachdem Testfall.

Vorgängertestfälle und Nachfolgetestfälle (nach IEEE 829 - Level Test Case) Die *Vorgängertestfälle* und die *Nachfolgetestfälle* ergeben sich aus den *Testfallvorzuständen* und *Testfallnachzuständen*. Der Umgang mit den sich so ergebenden Abhängigkeiten und ob diese vermieden werden sollten sind Teil der Architekturüberlegung bei der Erstellung der Testfallbibliothek. (siehe Kapitel 5.2.3)

Testumgebung (nach IEEE 829 - Level Test Case) Die Dokumentation der *Testumgebung* definiert alle für den Testfall nötigen Komponenten des SUT und die zusätzlich nötige Hardware und Software.

Testfallargumente (nach IEEE 829 - Level Test Case) *Testfallargumente* beinhalten die konkreten Eingangstimulationen des SUT, welche dieser Testfall ausführen soll.

Testfallergebnisse (nach IEEE 829 - Level Test Case) Die *Testfallergebnisse* beinhalten die erwarteten Ausgaben/Rückmeldungen bzw. das erwartete Verhalten des SUT basierend auf den *Testfallargumenten*.

Testfallstatus Der *Testfallstatus* erweitert die IEEE 829 und dient als Dokumentation für die Umsetzung der Testfallbibliothek. Unter diesem Punkt kann festgehalten werden, ob ein Testfall ermittelt, genau spezifiziert oder bereits umgesetzt und getestet ist.

Die Tabellen 2 und 3 zeigen zwei Beispiele, wie Testfälle nun entsprechend der so vorgegebenen Testfalldokumentation festgehalten werden.

Kennzeichnung	T101.2_XCP
Testfalltyp	automatisiert
Testfallzweck	ECU darf nicht zu früh ausgehen
Testfallquelle	„alte“ Freigabecheckliste
Testfallanforderung	nach 60s ohne Kommunikation im CalibrationWakeup muss die ECU abschalten
Testfallobjekt	CalibrationWakeup
Testfallvorzustände	ECU im CalibrationWakeup und Kalibrierzugriff über INCA
Testfallnachzustände	ECU im CalibrationWakeup und Kalibrierzugriff über INCA
Vorgängertestfälle	T101.1_XCP
Nachfolgetestfälle	-
Testumgebung	Testumgebung mit ECU, externe Cal.-HW, INCA, Switchbox, Netzteil
Testfallargumente	Experiment schließen und 55s abwarten
Testfallergebnisse	Kalibrierzugriff in INCA, CalibrationWakeup-TaskCounter > 0, ECU an
Testfallstatus	spezifiziert

Tabelle 2: Beispiel Testfall 1 in der Testfalldokumentation

Kennzeichnung	T101.3_XCP
Testfalltyp	automatisiert
Testfallzweck	ECU darf nicht zu spät ausgehen
Testfallquelle	„alte“ Freigabecheckliste
Testfallanforderung	nach 60s ohne Kommunikation im CalibrationWakeup muss die ECU abschalten
Testfallobjekt	CalibrationWakeup
Testfallvorzustände	ECU im CalibrationWakeup und Kalibrierzugriff über INCA
Testfallnachzustände	ECU aus, INCA getrennt
Vorgängertestfälle	T101.1_XCP oder T101.2_XCP
Nachfolgetestfälle	-
Testumgebung	Testumgebung mit ECU, externe Cal.-HW, INCA, Switchbox, Netzteil
Testfallargumente	Experiment schließen und 65s abwarten
Testfallergebnisse	keine Verbindung zur ECU in INCA möglich, ECU aus
Testfallstatus	spezifiziert

Tabelle 3: Beispiel Testfall 2 in der Testfalldokumentation

Beide Testfälle sind für dieselbe Testfallanforderung des gleichen Testobjektes. Sie prüfen die Anforderung auf zwei unterschiedliche Weisen. Für den einen Testfall ist der Zweck festzustellen, ob die ECU im CalWup ohne Kommunikation nicht schon zu früh abschaltet und der andere prüft, ob sie nicht zu spät oder gar nicht abschaltet. Die Testfälle sind XCP spezifisch, da eine solche Abschaltung in ETK-Systemen nicht stattfindet. In ETK-Systemen besteht eine dauerhafte Kommunikation zur ECU, auch wenn der Nutzer kein INCA-Experiment geöffnet hat.

Während dieser Arbeit hat sich gezeigt, dass diese Art der Dokumentation für einen Überblick über einen Test (bzw. die ganze Testfallbibliothek) oder das Nachvollziehen eines Testes ausreichend ist. Für die Implementierung eines automatisierten Testes der einzelnen Testfällen sind die Definitionen in *Testfallargumente* und *Testfallergebnis* allerdings oft unzureichend. Da der genaue automatisierte Ablauf einzelner Testfälle (Testskripte) aber erst im Laufe der Arbeit entsteht und eine doppelte Dokumentation von gleichem Informationsgehalt vermieden werden soll, wird die Implementierung des Testskriptes in ECU-Test so ausführlich dokumentiert und übersichtlich genug gestaltet, dass keine zusätzliche Dokumentation nötig ist.

Dieses Vorgehen entspricht dem V-Modell, da eine genaue Definition von Testskripten für Testfälle in die Teilrealisierungsebene fällt und nicht in die Subsystemebene, wie die Testfallbeschreibungsdefinitionen aus diesem Kapitel.

4.2 Analyse Freigabechecklisten

Basierend auf der nun festgelegten Dokumentation für Testfälle wurde begonnen, einzelne Testfälle aus den gegebenen Freigabechecklisten zu extrahieren. Bereits hierbei haben sich starke Ähnlichkeiten zwischen den Testfällen für die zwei Use-Cases ergeben (vgl. Use-Case Analyse Kapitel 3.1). Unter Hinzuziehen der jeweiligen Experten für die zwei Zugriffsvarianten wurde eine erste Harmonisierung zwischen den Testfällen vorgenommen.

Bei der Analyse der Freigabechecklisten hat sich zudem gezeigt, dass diverse Testfälle für beide Systeme sinnvoll wären, aber bisher nur in einem der beiden Use-Cases genutzt wurden. Dementsprechend wurden die so identifizierten „neuen“ Testfälle für beide Use-Cases angedacht und in der Testfalldokumentation als harmonisierter Testfall aufgenommen.

Ein spezielleres Beispiel für einen solchen Testfall ist der Test des Verstellzugriffes. Programmstände, die für die Freigabe der Toolkette genutzt werden, werden um zusätzliche Testfunktionen erweitert. Diese Testfunktionen beinhalten Variablen von allen möglichen Datentypen, die im Umfeld der ECU-Software Entwicklung und der ECU-Funktionsentwicklung genutzt werden. Der Begriff Datentyp bezieht sich hierbei sowohl auf elementare Datentypen wie *Unsigned Integer*¹⁷ 32bit oder *Real*¹⁸ 32bit, als auch auf zusammengesetzte Datentypen wie *Arrays* oder *Structures*. Diese Variablen sind je einmal mit einer Verstellgröße als auch mit einer Messgröße verknüpft. Im Testfall für den Verstellzugriff aus der Freigabecheckliste für XCP-Systeme wurden nur stichprobenartig einige dieser Verstellgrößen verstellt und die entsprechenden Messgrößen überprüft. Im Testfall für den Verstellzugriff bei ETK-Systemen wurden hingegen alle getestet (mit Hilfe des *ETK-Testers*). Diese Ergänzung wurde in einen harmonisierten Testfall für beide Use-Cases übernommen.

Weitere Beispiele für Testfälle, die aus einer Freigabecheckliste stammen und für den jeweils anderen Use-Case genutzt werden können, sind Testfälle zu Testobjekten die bereits in Kapitel 3.1.4 aufgefallen sind, wie die Untersuchung von Verhalten bei Leitungsunterbrechungen.

Beide untersuchten Freigabechecklisten haben außerdem einige Einträge zu Metadaten. Dazu gehören neben Nennung von Tester, Ort und Datum ebenso Daten zu dem SUT,

¹⁷vorzeichenlose ganze Zahlen

¹⁸Gleitkommazahlen

der verwendeten Softwaretools und ECU-Software. Konkret sind das bspw. die Version von INCA, der getestete ETK-Typ, die Transportlayer-Instanz bei XCP-Systemen, der ECU-Typ, der Programmstand auf der ECU, die Versionsnummer der Testfunktionen im verwendeten Programmstand etc. Im Zuge der Testautomatisierung sollen diese Metadaten auch automatisiert erfasst werden (soweit möglich) und Teil der Testsuite sein. Da einige dieser Metadaten automatisiert auf Plausibilität geprüft werden können, werden diese Tests ebenfalls als Testfälle in die Dokumentation aufgenommen.

4.3 Erfassung weiterer Testfälle

Die ISO 26262 beschreibt in ihrem 4. Band in Kapitel 8 „Item integration and testing“ (vgl. Abbildung 6). Dieses Kapitel der ISO 26262 nennt Methoden zur Ableitung von Testfällen basierend auf der Bewertung des Systems entsprechend ASIL. Aufgrund der fehlenden Grundlagen für eine solche Bewertung (vgl. Kapitel 4.1) werden die Methoden ohne eine solche Einordnung betrachtet. Dennoch sei hier die Legende erwähnt, die relevant wird, falls eine Einordnung nach ASIL vorgenommen wird.

++	starke Empfehlung (highly recommended)
+	Empfehlung (recommended)
o	keine Empfehlung (no recommendation)

Die Tabelle 4 zeigt die entsprechenden Methoden die so direkt der Norm entnommen wurde.

Methods		ASIL			
		A	B	C	D
1a	Analysis of requirements	++	++	++	++
1b	Analysis of external and internal interfaces	+	++	++	++
1c	Generation and analysis of equivalence classes for hardware-software integration	+	+	++	++
1d	Analysis of boundary values	+	+	++	++
1e	Error guessing based on knowledge or experience	+	+	++	++
1f	Analysis of functional dependencies	+	+	++	++
1g	Analysis of common limit conditions, sequences, and sources of dependent failures	+	+	++	++
1h	Analysis of environmental conditions and operational use cases	+	++	++	++
1i	Analysis of field experience	+	++	++	++

Tabelle 4: Methods for deriving test cases for integration testing [ISO11]

Da die meisten, sich durch diese Methoden ergebenden, Testfälle bereits aus den Freigabechecklisten bekannt sind, wird nachfolgend zum Großteil beispielhaft darauf eingegangen, welche Testfälle das Produkt welcher Methode gewesen wären.

Analyse der Anforderungen (1a) Wie bereits in Kapitel 4.2 beschrieben, ergeben sich die meisten Testfälle direkt aus den Freigabechecklisten, die gleichzeitig die einzige offiziell vorhandene Definition von Anforderungen darstellen. Darunter auch Testfälle, die im Prinzip aus der Anforderung an das eine System in die Anforderung des anderen Systems übernommen werden können und somit auch dort zu Testfällen werden.

Analyse von externen und internen Schnittstellen (1b) Im Prinzip handelt es sich bei der Freigabe der Toolkette um einen Integrationstest der eine Applikationsschnittstelle

prüft. Dementsprechend prüfen die meisten Testfälle, die bereits aus den Freigabechecklisten bekannt sind, die Schnittstellen zwischen den unterschiedlichen Hardware- und Softwareteilen der Toolkette. Hierzu gehören u.a. die Schnittstelle zwischen ECU und ED-RAM untereinander und zum ETK, die Schnittstelle von ETK zum Schnittstellenmodul und die Schnittstelle vom Schnittstellenmodul zum Testrechner. Diese unterschiedlichen Schnittstellen werden allerdings fast ausschließlich als eine Schnittstelle angesehen, die Messen und Verstellen von Werten der ECU am Testrechner ermöglicht. Im Prinzip handelt es sich also um einen Blackbox-Test. Um die internen Schnittstellen zu testen, entsprechend dem Testverfahren bei Whitebox-Tests, wären Informationen und Zugriff über/auf diese größtenteils proprietäre Schnittstellen nötig. Eine Analyse der Schnittstellen ergibt also keine weiteren Testfälle. Es ist allerdings möglich, den bereits erfassten Testfall des „Verhaltens bei Leitungsunterbrechung“ als Produkt einer solchen Schnittstellenanalyse zu sehen.

Generierung und Analyse von äquivalenten Klassen für Hardware-Software Integration (1c) Diese Methode ist irrelevant, da es sich nicht um einen Test für Hardware-Software Integration handelt.

Analyse von Grenzwerten (1d) Ein Test auf Grenzwertverhalten ist in den Testfällen, in welchen mit Verstellgrößen gearbeitet wird, möglich. Der Testfall für den Verstellzugriff sollte daher nicht nur testen, ob die jeweiligen Messgrößen auf Veränderung der verknüpften Verstellgrößen reagieren, sondern es sollten auch dem Datentyp entsprechende Grenzwerte geprüft werden. INCA blockiert zwar ein Verstellen mit Werten größer oder kleiner dem zulässigen Bereich des Datentyps, aber es sollte bspw. ein *Signed Integer* mit dem Grenzwert -1 getestet werden. Durch gleichzeitiges Verstellen mehrerer Variablen und darauffolgendes Messen mehrerer Variablen, kann bei einem solchen Testfall festgestellt werden, ob der Speicherzugriff, den der ETK realisiert, keine Adressbereiche verletzt und damit ggf. andere Variablen fälschlicherweise überschreibt.

Analyse funktionaler Abhängigkeiten (1f) Testfälle zur funktionalen Abhängigkeit ergeben sich ebenfalls aus den bereits angedachten Anforderungen aus den Freigabechecklisten. Die definierte Dokumentationsstruktur für Testfälle listet spezifisch die Abhängigkeit von bestimmten Testfällen zu Anderen im Bezug auf Vor- und Nachbedingung. Funktionale Abhängigkeiten können sich aber auch ergeben, indem bestimmte Testfälle nur erfolgreich abgeschlossen werden können, wenn andere bereits erfolgreich abgeschlossen wurden. Ist z.B. der Testfall der „automatischen Hardwareinitialisierung in INCA“ fehlerhaft, kann keine Verbindung zur ECU hergestellt werden. Der Großteil der anderen Testfälle erfordert eine solche Verbindung und würde somit auch fehlschlagen. Im Zuge der Architekturauslegung für eine Testsuite (Kapitel 5.2) wird der Umgang mit solchen Abhängigkeiten geklärt.

Analyse gemeinsamer Grenzbedingungen, Abläufe und Quellen zusammenhängender Fehler (1g) Die Analyse von Quellen zusammenhängender Fehler spiegelt sich bereits in der Analyse der funktionalen Abhängigkeiten wieder. Es gibt einige Testfälle, die basierend auf den selben Fehlern fehlschlagen können. Testfälle, die sich basierend auf der Analyse gemeinsamen Grenzbedingungen und/oder Abläufen ergeben, finden sich in diesen Use-Cases keine.

Analyse von Umgebungsbedingungen und bedienungsbedingter Use-Cases (1h) Die Analyse von Umgebungsbedingungen und bedienungsbedingter Use-Cases führt zur Bestätigung weiterer Testfälle, welche bereits in den Freigabechecklisten angedacht wurden. Hierzu gehören Testfälle zum definierten Resetverhalten bei Spannungseinbrüchen der Versorgungsspannung oder bei Leitungsunterbrechung sowie Dauermessungen bei Grenztemperaturen. Letztere werden in der Testfalldokumentation angedacht, aber nicht in der Testautomatisierung umgesetzt, da die Automatisierung für die Anwendung an einem Laborplatz ausgelegt werden soll und nicht für eine Anwendung in Kälte-/Wärmekammern.

Fehlerabschätzung und Analyse von Felderfahrung (1e und 1i) Beide dieser Methoden setzen fortgeschrittene Erfahrung mit den Toolketten voraus. Aus den Methoden ergeben sich für die Testautomatisierung also nur die Testfälle, die bereits durch Erfahrungswerte der jeweiligen Experten Teil der Freigabechecklisten wurden.

4.4 Testfallsammlung

Bereits in Kapitel 3.1 haben sich diverse Testobjekte abgezeichnet. Dort wurde besprochen, wie sich Testfälle zu diesen Testobjekten ergeben haben. Es wurde eine Dokumentationsstruktur definiert, in welcher die ermittelten Testfälle abgelegt wurden.

Abbildung 14 dient, um abschließend ein durchgängiges Beispiel für das Vorgehen zu zeigen.

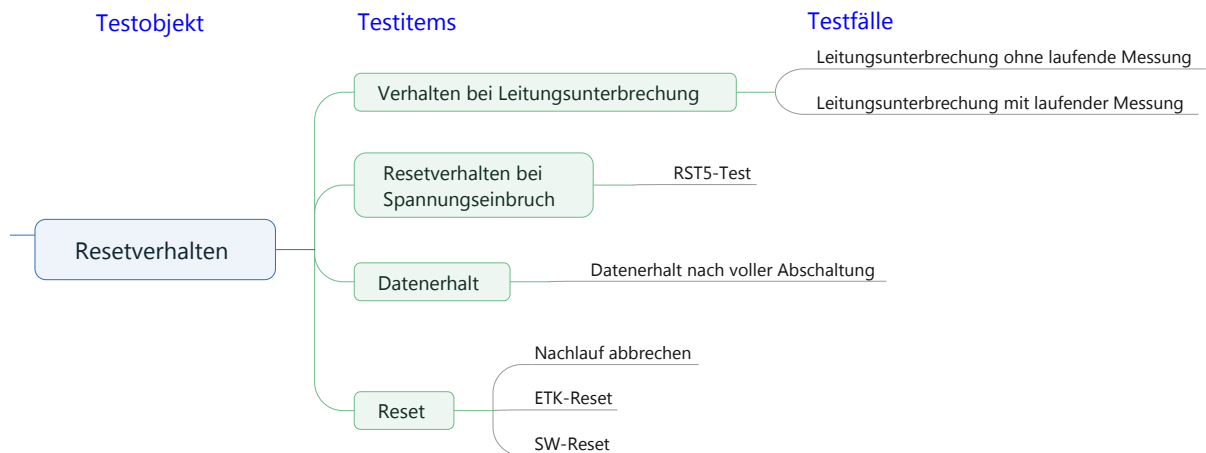


Abbildung 14: Struktur Testfallerfassung

Die Use-Case-Analyse aus Kapitel 3.1 hat diverse Testobjekte dargelegt, die für die Freigabe der zwei Systeme getestet werden müssen. So stellen z.B. die Freigabechecklisten gewisse Anforderungen an das *Resetverhalten* der Systeme. Basierend auf der Analyse entsprechen der Methoden aus Kapitel 8 der ISO 26262-4 ergaben sich dann aus der Freigabecheckliste diverse Testfälle zu den ermittelten Testobjekten. Diese Testfälle (in der Abbildung die Zweige auf der rechten Seite) behandeln oft dasselbe Testitem (in der Abbildung mittig). Das Testitem stellt also eine zusätzliche Strukturebene für die Dokumentation der Testfälle dar. Für das Resetverhalten ist das bspw. das Testitem *Leitungsunterbrechung* das in zwei Testfällen getestet wird; einmal *mit laufender Messung* und einmal *ohne laufende Messung*. Die erfassten Testfälle wurden dann entsprechend der angepassten IEEE 829 dokumentiert. Tabelle 5 zeigt den entsprechenden Eintrag der

Dokumentation passend zu dem Testfall für *Leistungsunterbrechung bei laufender Messung*. Die Begriffe *Testobjekt* und *Testitem* werden in der Dokumentation als *Testobjekt* zusammengefasst und die jeweils spezifischere Unterordnung genannt.

Kennzeichnung	T100.2
Testfalltyp	mit Nutzereingabe
Testfallzweck	wenn eine Messung läuft und eine Leistungsunterbrechung stattfindet, muss bei Wiederherstellung der Verbindung automatisch die Messung neu gestartet werden
Testfallquelle	„alte“ Freigabecheckliste
Testfallanforderung	bei Leistungsunterbrechung muss nach Wiederherstellung der Verbindung in den Zustand vor der Unterbrechung zurückgekehrt werden
Testfallobjekt	Leistungsunterbrechung
Testfallvorzustände	ECU Kalibrierzugriff über INCA, Messung läuft
Testfallnachzustände	ECU Kalibrierzugriff über INCA, Messung läuft
Vorgängertestfälle	-
Nachfolgetestfälle	-
Testumgebung	Testumgebung mit ECU, (ETK), externe Cal.-HW, INCA, Switchbox, Netzteil
Testfallargumente	Verbindung physikalisch trennen und wieder verbinden (händisch)
Testfallergebnisse	Kalibrierzugriff wird automatisch wiederhergestellt und die Messung neu gestartet
Testfallstatus	getestet

Tabelle 5: Beispiel Testfall 3 in der Testfalldokumentation

Im Anhang findet sich ein Screenshot der vollständigen Testfalldokumentation (Abbildung [A.8](#) und [A.9](#)).

5 Architektur

Mit Hilfe der nun bestehenden Testfallsammlung (Kapitel 4) und den Informationen aus der Konzeption (Kapitel 3) wird in diesem Kapitel die genaue Architektur diskutiert, auf deren Basis der Test in ECU-Test umgesetzt wird. Das Kapitel ist somit im V-Modell als „Teil-Realisierung- Anforderungs- Analyse und Entwurf“ einzuordnen, teilweise sogar bereits als „Erstellung, Implementierung“.

5.1 Treiber und Schnittstellen

Mit Hilfe der angelegten Testfalldokumentation ist es möglich abzuschätzen, welche Funktionalitäten in ECU-Test gegeben sein müssen, um die jeweiligen Testskripte für die einzelnen Testfälle umsetzen zu können. Es wurde eine Liste mit Funktionen angelegt, die nötig sind um:

- die Switchbox zu steuern,
- das Netzteil zu steuern,
- INCA zu steuern,
- SQLite-Datenbanken zu durchsuchen,
- diverse Metadaten aus der Windows Registry zu beziehen und
- *.a2l-Dateien zu durchsuchen und zu ändern, ein A2L-Parser.

Im Anhang ist ein Screenshot hinterlegt, der die Sammlung dieser Funktionalitäten aufzeigt (Abbildung A.7).

Das folgende Kapitel behandelt die, sich durch diese Anforderungen ergebenden, Schnittstellen bzw. Erweiterungen, die für ECU-Test umgesetzt wurden.

5.1.1 Netzteil

ECU-Test bietet eine Art der Erweiterbarkeit in sogenannten Tooladaptern. Ein Tooladapter wird als Python-Quelldatei angelegt und beinhaltet eine Ableitung der Klasse *Tooladapter*. In einem Test wird dann definiert, welche hinterlegten Tooladapter Teil der Testbenchkonfiguration sind und mit welchen Einstellungen sie genutzt werden sollen. Bei der Testausführung wird dann eine Verbindung mit dem jeweiligen Tooladapter hergestellt, um die von ihm zur Verfügung gestellten *Tool Jobs* nutzen zu können. Dieses Verbinden (und später Trennen) sowie die Möglichkeit Einstellungen zu hinterlegen sind die besonderen Eigenschaften eines solchen Tooladapters. Zur Umsetzung einer Erweiterung für die Steuerung eines Netzteils bietet sich ein solcher Tooladapter an, weil durch ihn das Verbinden und Trennen der COM-Schnittstelle mit dem Testablauf automatisch gekoppelt werden kann. Ebenso bietet er die Möglichkeit einfache Einstellungen für den Nutzer bereitzustellen, um z.B. den COM-Port zu wählen oder definieren zu lassen, ob während des Testablaufes die manuelle Bedienung des Netzteils gesperrt sein soll.

Abbildung 15 zeigt einen Screenshot von ECU-Test, der den umgesetzten Tooladapter für das Netzteil zeigt (das verwendete Netzteil ist vom Typ BK Precision 9205). Die rechte Seite des Ausschnittes zeigt die Testbenchkonfiguration, in welcher der entsprechende

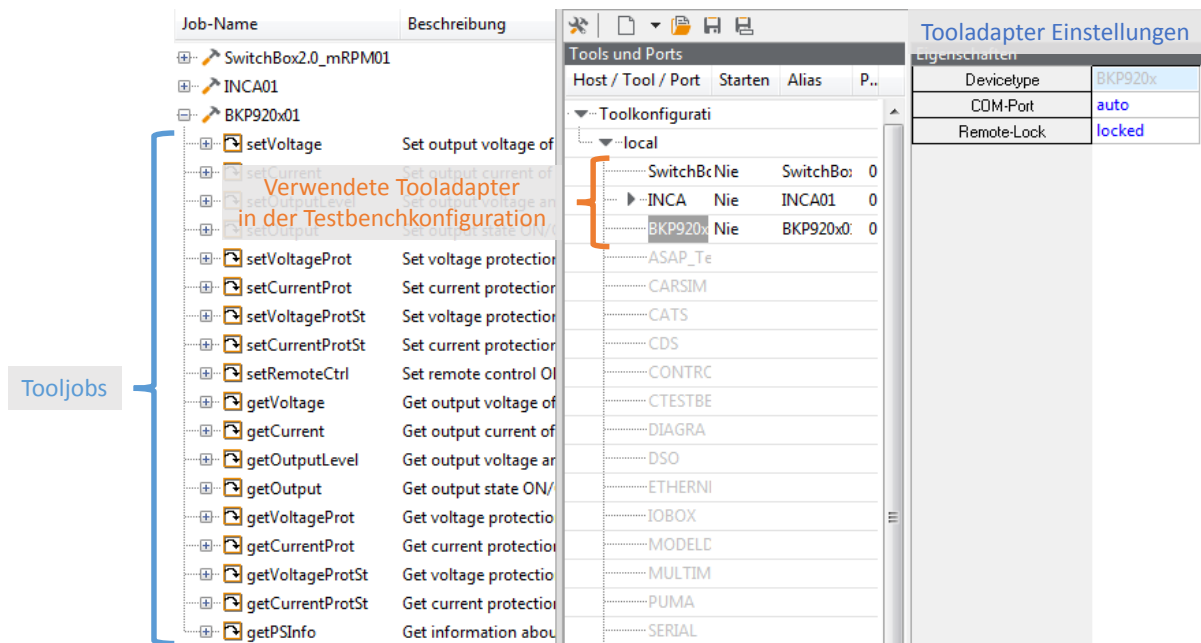


Abbildung 15: Beispiel für einen Tooladapter in ECU-Test

Tooladapter namens „BKP920x“ aktiviert und mit den entsprechenden Einstellungen versehen wurde. Links im Ausschnitt sind die *Tool Jobs* zu sehen, die der Tooladapter zur Verfügung stellt.

Zuerst wurden also die oben aufgeführten Einstellungen angelegt. Eine zur Wahl des zu verwendenden COM-Ports und eine zur Wahl, ob die manuelle Bedienung während des Testablaufs gesperrt sein soll. Die Wahl des COM-Ports bietet auch die Möglichkeit „auto“ wobei in der Registry nach verbundenen COM-Geräten gesucht wird. In den dort gelisteten Geräten hat das Netzteil einen eindeutigen Namen, der sich lediglich durch Abweichungen in einer angehängten Nummer unterscheiden kann (Bsp. „VCP0“, „VCP3“). Somit kann der zu verwendende COM-Port über die Registryeinträge automatisch ermittelt werden.

Der zweite Teil der Implementierung sind die Methoden *OnConfigure()* und *OnUnconfigure()*. Diese werden automatisch von ECU-Test aufgerufen, sobald der Tooladapter gestartet oder gestoppt werden soll. Sie beinhalten den Verbindungsaufbau bzw. -trennung zu dem Netzteil mit Hilfe des Pythonmoduls „pySerial“ [Lie18]. Welcher Port hierbei verwendet werden soll wird der entsprechenden Einstellung entnommen bzw. aus der Registry ausgelesen. Es folgt das Senden von Befehlen an das Netzteil, um es auf Remotesteuerung zu setzen und ggf. für manuelles Bedienen zu sperren bzw. bei *OnUnconfigure()* diese Sperrung wieder aufzuheben. Die Ableitung der Tooladapter-Klasse muss ebenfalls eine Methode *IsBroken()* definieren. Diese wird vor jedem Testschritt aufgerufen, um zu prüfen, ob der Tooladapter noch funktionsfähig ist. Im Falle des Netzteils wurde diese Methode so umgesetzt, dass sie in der Registry nachdem zuvor bei *OnConfigure()* gemerkten Namen des Netzteils sucht, um festzustellen, ob die Verbindung vielleicht physikalisch getrennt wurde. Schlägt die Überprüfung durch die Methode *IsBroken()* fehl, so wird der aktuelle Testablauf in ECU-Test mit einer entsprechenden Fehlermeldung beendet.

Der Hauptteil der Implementierung eines Tooladapters sind die *Tool Jobs*. Diese definieren die Funktionen eines Tooladapters, die später in der grafischen Testskriptimplementierung genutzt werden können. Im Falle des Netzteils handelt es sich hierbei um

Jobs für das Setzen von Ausgangsspannung und Ausgangsstrom sowie das Aktivieren und Deaktivieren des Ausgangs. Der Vollständigkeit halber wurden auch andere Funktionen umgesetzt, die in dieser Arbeit allerdings keine Anwendung finden. Diese zusätzlichen Jobs sollen es ermöglichen, dass der Tooladapter in Zukunft in anderen Projekten genutzt werden kann, ohne dass er erweitert werden muss. Dazu gehören z.B. Jobs zum Setzen von Strom- und Spannungswarngrenzen, für den Überspannungsschutz aber auch Jobs zum Auslesen von aktuell gesetzten Strom- und Spannungspegeln. All diese Jobs nutzen das, bei der Verbindung angelegte, „pySerial“-Objekt, um die jeweiligen seriellen Befehle an das Netzteil zu senden und die Antwort auszulesen. Diese Befehle entsprechen, wie in Kapitel 2.10.1 angesprochen, durch die IEEE 488.2 genormter Anweisungen. Das ermöglicht es den Tooladapter für eine Vielzahl von Netzteilen zu nutzen, deren Befehle ebenfalls dieser Norm entsprechen.

5.1.2 Switchbox

Da die Switchbox, ebenso wie das Netzteil, über ASCII-codierte serielle Befehle an der COM-Schnittstelle gesteuert wird, wurde sie ebenfalls als Tooladapter auf Basis des „pySerial“-Pythonmoduls umgesetzt. Für diesen Tooladapter war bereits Quellcode aus einer anderen Abteilung vorhanden. Für die Zwecke der Testautomatisierung musste dieser allerdings um ein paar Funktionen erweitert werden. Es gab z.B. noch keinen Job zum Setzen der Drehzahlstimulierung. Ebenso war kein Job vorhanden, der auf den Switchboxbefehl zur Abfrage des Status des Hauptrelais zugreift. Da das Hauptrelais von der ECU gesteuert wird und ein Indiz dafür ist, ob der Nachlauf¹⁹ (PostDrive) abgeschlossen wurde oder nicht, ist die Abfrage des Hauptrelaisstatus eine wichtige Funktion für einige der Testfälle.

Die Implementierung dieser zusätzlichen Jobs hat dann eine Fehlfunktion in der Abfrage des Hauptrelaisstatus auf Seiten der Switchbox-Firmware offenbart. Wie Abbildung 10 auf Seite 18 zeigt, hat der Schalter für das Hauptrelais (HR) drei Positionen. „On“ und „Off“ um manuell ein ein- oder ausgeschaltetes HR zu simulieren und „MR“. In der Stellung „MR“ wird das HR durch eine Steuerleitung von der ECU geschaltet. Während der Testautomatisierung wird das HR lediglich in der „MR“ Stellung genutzt. Der Abfragebefehl der Switchbox meldet aber nur den aktuellen Status zurück, wenn das HR auf „On“ oder „Off“ steht. Durch eine Änderung in der Firmware ließ sich dieses Problem beheben. Dazu wurde der Rückmeldewert der Statusabfrage direkt auf den Pegel des Pins bezogen wurde, der das verbaute Relais steuert, anstatt auf den Pegel des Pins, welcher mit dem physikalischen Schalter verbunden ist. Um später Fehlfunktionen zu vermeiden, falls die falsche Firmwareversion auf der im Testaufbau benutzten Switchbox aufgespielt ist, wird in der *OnConfigure()* Methode, über einen entsprechenden Befehl an die Switchbox, die aktuelle Firmware abgefragt und mit einem Soll-Wert verglichen. Die Soll-Wert-Abfrage lässt hierbei Spielraum für neue Firmwareversionen, die auf der Angepassten aufbauen.

Der erweiterte Tooladapter kann nun:

- T15 an- und ausschalten,
- T30 an- und ausschalten,

¹⁹nachdem die Zündung (T15) ausgeschaltet wurde, geht das Fahrzeug (im Testaufbau nur das Hauptrelais des mit der Switchbox „simulierten“ Fahrzeugs) nicht sofort aus, sondern geht in den sogenannten PostDrive über

- HR an- und ausschalten,
- die Drehzahl setzen,
- die Sperrung der manuellen Schalter durch die Software zurücksetzen und
- Version der Switchbox sowie den aktuellen Status des HR abfragen.

5.1.3 INCA

ECU-Test bietet bereits in seinem Standardumfang einen integrierten Tooladapter für INCA. Dieser bietet Zugriff auf Mess- und Verstellgrößen der ECU über INCA, basierend auf der eingelesenen *.a2l und *.hex-Dateien. Er beinhaltet auch einige *Tool Jobs*, wie die der benutzerdefinierte Tooladapter. Die wichtigsten dieser *Tool Jobs* sind:

- initialisiere Hardware,
- prüfe die Verbindung der aktuell gewählten Zugriffsvariante (hier ETK oder XCP),
- lese eine Messgröße,
- schreibe eine Verstellgröße,
- starte oder stoppe eine Messung oder eine Aufnahme,
- flashe ECU mit *.hex-Datei,
- wechsele auf die AS oder auf die RS,
- lade aktuellen Datensatz hoch/runter und
- kopiere RS auf AS.

Der Vorteil des Zugriffes auf Mess- oder Verstellgrößen über die entsprechenden *Tool Jobs* ist, dass die Bezeichner für die jeweiligen Größen als Argument übergeben werden können. Wird jedoch der integrierte Zugriff, basierend auf *.hex und *.a2l genutzt, so werden die Größen aus einer fixen Liste gewählt. Im Testskript kann der Bezeichner dann nicht mehr geändert werden und es ist nicht möglich, wie mit dem *Tool Jobs*, einen variablen Bezeichner zu übergeben.

Die so zur Verfügung stehenden *Tool Jobs* decken allerdings nicht alle API-Funktionen von INCA ab, die für die Testautomatisierung nötig sind. Da es sich um einen in ECU-Test integrierten Tooladapter handelt, ist eine einfache Erweiterung, wie bei dem Tooladapter für die Switchbox, nicht möglich.

Eine Möglichkeit um die zusätzlichen Funktionen in ECU-Test zugänglich zu machen, ist einen neuen Tooladapter zu schreiben, der entweder parallel zu dem bereit vorhandenen arbeitet oder ihn ganz ersetzt. Da der bereits vorhandene Tooladapter aber durch die Verknüpfung mit *.a2l und *.hex-Dateien und den Zugriff auf Mess- und Verstellgrößen relativ tief in ECU-Test integriert ist, wäre ein Ersetzen durch einen neuen Tooladapter, welcher alle alten und neue Funktionen implementiert, äußerst zeitaufwendig. Ein parallel funktionierender Tooladapter stellt sich ebenfalls als komplizierte Lösung dar, da sich beide unabhängig voneinander mit der INCA-API verbinden müssten ohne auf die dort geöffneten Handles des jeweils Anderen zugreifen zu können.

Die Lösung für dieses Problem stellt die Erweiterung von ECU-Test mit simple Python-Funktionen dar. In einer Quellcodedatei werden diese Funktionen hinterlegt und in ECU-Test geladen, wo sie dann entsprechend der Form

```
user.<Ordner>.<Quelldateiname>.<Funktionsname>(<Argumente>)
```

aufgerufen werden können. Diese Funktionen werden so ausgelegt, dass sie eine zweite Verbindung zur API öffnen (es wird das Pythonmodul „pywin32“ [H+18] genutzt um auf COM-Objekte zuzugreifen) und voraussetzen, dass INCA über den Tooladapter bereits in einen bestimmten Zustand gebracht wurde. Dann können innerhalb der API die Handles der, durch den Tooladapter geöffneten, INCA-Elemente abgefragt und die entsprechenden Funktionen aufgerufen werden. Am Ende der Funktionen ist von einem expliziten Schließen der API-Verbindung abzusehen, da sonst auch die Verbindung des Tooladapters zur API getrennt werden würde. Dieses Vorgehen stellt einen funktionierenden Workaround dar, der in Zukunft einfach ersetzt werden kann, sobald die Funktionen von TraceTronic in den Standardumfang des Tooladapters eingepflegt wurden. Die zusätzlich benötigten Funktionen wurden aus diesem Grund, in Kooperation mit einer anderen Abteilung die ebenfalls ECU-Test nutzt, gesammelt und notiert. Diese „Wunschliste“ wird an TraceTronic übergeben und beinhaltet u.a. die im Workaround umgesetzten Funktionen um:

- Benutzeroptionen zu manipulieren,
- Optionen der Hardwarekonfiguration zu manipulieren,
- eine Neuberechnung der Prüfsummen für die Speicherseiten anzustoßen,
- die berechneten Prüfsummen ausgeben zu lassen bzw. zu vergleichen,
- die API des INCA-Tools „IP-Manager“ zu nutzen, um verbundene Schnittstellenmodule zu erkennen und
- Inhalte der aktuellen veränderten Arbeitsseite auf die Referenzseite zu Flashen (Kopieren AS auf RS vgl. Abbildung 7).

Da es sich bei diesen Funktionen zum Großteil um Funktionen handelt, welche ausschließlich für HITs benötigt und bisher bei SIL- und HIL-Anwendungen (die eigentlichen Anwendungsbereiche für ECU-Test) nicht vermisst wurden, ist abzuwarten, inwiefern solche Erweiterungen von TraceTronic umgesetzt werden.

5.1.4 SQLite

Einige Testfälle erfordern die Überprüfung einer Datenbank auf Vorhandensein bestimmte Einträge. So wird bspw. in einen Softwarestand ein bestimmter Baustein mit Testfunktionen und -größen integriert, welche im Test genutzt werden. In einer SQLite-Datenbank namens *Workspace* sind u.a. alle Bausteine, und deren Versionen, sowie Quelldateinamen hinterlegt, die in dem Softwarestand genutzt wurden. Um die nötigen Einträge zu überprüfen werden, wie für die INCA-Erweiterung, simple Python-Funktionen genutzt, die mit Hilfe des Pythonmoduls „sqlite3“ (im Standardumfang von Python) auf den Workspace zugreifen.

5.1.5 Registry

Zur Erfassung von diversen Metadaten wie Benutzer, Rechner-Name und Programmversionen ist eine Erweiterung für ECU-Test nötig, welche diese Daten aus der Windows Registry abrufen kann. Diese Erweiterung wird wieder über Python-Funktionen realisiert, die in ECU-Test geladen werden. Es wird das „_winreg“ Modul aus dem Standardumfang von Python 2.7 genutzt, um auf die nötigen Registryschlüssel zuzugreifen.

5.1.6 A2L-Parser

Eine A2L-Datei ist eine ASCII-codierte, menschenlesbare Beschreibungsdatei. Sie beinhaltet diverse Informationen über die ECU und den darauf laufenden Softwarestand für Tools wie INCA. Dazu gehören u.a. symbolische Namen für steuergeräteinterne Größen, sowie deren physikalische Einheiten, auf die über die Toolkette zugegriffen werden kann. Die A2L hält aber auch Einstellungen für die Zugriffsvarianten bereit. Dazu gehören z.B. die maximale CAN-Buslast, die XCP erzeugen darf oder die maximale Anzahl an Messgrößen pro Messraster, die INCA bei Zugriff über ETK zulassen darf.

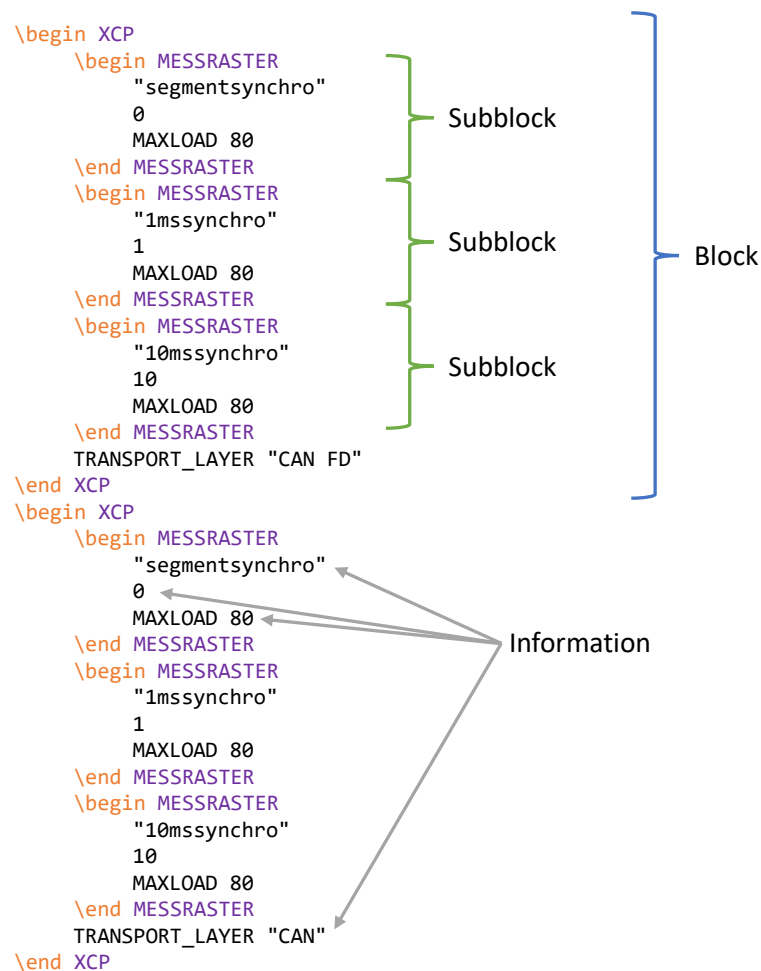


Abbildung 16: Prinzipieller Aufbau einer A2L

Sowohl im Falle von ETK-Systemen als auch bei XCP-Systemen müssen vor der Ausführung der meisten Testfälle einige Einstellungen innerhalb der A2L verifiziert und angepasst werden. Ein A2L-Parser in Python soll ECU-Test dazu befähigen eine A2L in eine

Datenstruktur einzulesen und in dieser nach den benötigten Informationen zu suchen. Dabei sollen den Informationen auch Zeilennummern hinterlegt werden, sodass basierend auf diesen die A2L automatisiert gepatcht werden kann. Abbildung 16 zeigt den prinzipiellen Aufbau einer A2L. Der Parser wurde so ausgelegt, dass ihm zu Beginn der Dateipfad der A2L und ein Blockpfad angegeben werden muss. Der Blockpfad beschreibt wo die gesuchte Information in der A2L zu finden ist.

In Bezug auf das gegebene Beispiel in Abbildung 16 könnte als Blockpfad 1. *XCP* oder 2. *XCP/MESSRASTER* gegeben sein. Im ersten Fall würde die Datenstruktur zwei Blöcke beinhalten, welche jeweils aus einer Information und drei Subblöcken bestehen. Die Subblöcke wiederum beinhalten je drei Informationen und keine weiteren Subblöcke. Da in einer A2L Blöcke aber oft noch tiefer als in diesem Beispiel geschachtelt sind, wurde der Parser so aufgebaut, dass durch einen rekursiven Aufruf auch Subblöcke von Subblöcken sowie deren Subblöcke usw. erfasst werden. Wurde also einmal der oberste, für den Nutzer interessante, Pfad angegeben, so kann danach in der Datenstruktur über die Subblöcke zu immer tieferen Pfaden gelangt werden. Im Bezug auf das Beispiel ist das insofern relevant, da ein Parsen entsprechend Pfad 2 zwei Blöcke zurückgeben würde, in denen man die Information *“segmentsynchro“* finden kann. Beide Blöcken sähen völlig identisch aus. Wird allerdings mit Pfad 1. eingelesen, so kann in den zwei zurückgegebenen Blöcken der ausgesucht werden, der in der Information *TRANSPORTLAYER* z.B. den Eintrag *“CAN FD“* beinhaltet. Danach können die Subblöcke dieses Blockes auf den Subblock mit der Information *“segmentsynchro“* durchsucht und dann geprüft werden, ob der Subblock die Information *0* beinhaltet.

Es fällt auf, dass die Informationen innerhalb eines Blockes teilweise mit Bezeichnungen (*MAXLOAD* oder *TRANSPORTLAYER*) hinterlegt sind, in manchen Fällen aber nur Werte eingetragen sind (1. und 2. Information in den *MESSRASTER*-Blöcken). Bei den Informationen ohne Bezeichner ist lediglich die Position innerhalb des Blockes eine Aussage darüber, worauf sich die Information bezieht. Die Methode um Informationen aus einem Block ausgeben zu lassen nimmt als Argument deshalb sowohl einen numerischen Index als auch einen Bezeichner-String. Zudem wird optional von dieser Methode auch die absolute Zeilennummer innerhalb der A2L-Datei ausgegeben, in welcher die gesuchte Information gefunden wurde, um damit die A2L anpassen bzw. patchen zu können.

Der Parser und der Patcher ermöglichen es somit für den aktuellen Transportlayer *CAN FD* im Messraster *segmentsynchro* als maximale Buslast (*MAXLOAD*) statt 80% für Testzwecke 100% zuzulassen, indem dieser Wert in der entsprechenden Zeile ersetzt/gepatcht wird.

5.2 Testsuite

Dieses Kapitel beschäftigt sich mit der Architektur der obersten Abstraktionsebene der Testautomatisierung. In ECU-Test ist das offizielle Äquivalent für eine Testsuite ein *Projekt*. Laut ISTQB ist eine Testsuite lediglich ein Ablauf von Testfällen (vgl. Kapitel 2.3.1). In ECU-Test beinhaltet ein *Projekt* aber neben einem Ablauf von Testfällen, *Packages*, eine *Testkonfiguration* und eine *Testbenchkonfiguration*. Diese definieren Einstellungen für den Test bzw. Informationen zum aktuellen SUT. Das aktuelle SUT ist hierbei immer eine Applikationstoolkette, entweder mit XCP- oder ETK-Zugriff, aber mit variierender Hard- und Software. Der Test wird so ausgelegt, dass die Testsuite (hier also ein Ablauf von Testfällen, eine Testkonfiguration und eine Testbenchkonfiguration) das einzige Element ist, welches vom Tester, basierend auf den unterschiedlichen Toolketten, in der Testautomatisierung angepasst werden muss.

5.2.1 Testkonfiguration und Testbenchkonfiguration

Die Testkonfiguration beinhaltet diverse Test und SUT (Toolketten) spezifische Parameter sowie Informationen für die Erstellung des Testreports. Sie stellt den Hauptanlaufpunkt für die Anpassung des Testes dar. In der Testbenchkonfiguration sind ebenso Test und SUT spezifische Einstellungen vorhanden, die ggf. angepasst werden müssen.

Kapitel 5.1 hat bereits die drei nötigen Tooladapter beschrieben, die in der Testbenchkonfiguration genutzt werden: INCA, die Switchbox und das Netzteil. Welche Tooladapter für unterschiedliche Tests/SUTs genutzt werden unterscheidet sich im Falle dieser Testautomatisierung nicht. Unterschiede gibt es nur bei den Einstellungen des Tooladapters für INCA. Sie beinhalten Informationen zu den nötigen INCA-Pfaden, in denen der zu nutzende INCA-Workspace abgelegt werden soll (der ggf. von ECU-Test zusammen mit Experiment und Projekt automatisch angelegt wird). Eine weitere Einstellung beinhaltet den Gerätenamen, der die zu nutzende Zugriffsvariante beschreibt. ECU-Test bietet die Möglichkeit über globale Konstanten Einstellungen und Parameter der Testbenchkonfiguration in die Konstantendefinition der Testkonfiguration auszulagern. Diese Möglichkeit wurde für die Testautomatisierung wahrgenommen, um zu erreichen, dass jedes Mal wenn eine neue Toolkette getestet werden soll, der Tester nur die Testkonfiguration und das Projekt anpassen muss.

Soll eine neue Toolkette getestet werden ist also die Testkonfiguration der erste Anlaufpunkt um Dateipfade zu

- dem zu verwendenden INCA-Workspace (vgl. Kapitel 2.7),
- den Softwareartefakten *.a2l und *.hex,
- der Workspace SQLite-Datenbank (*.db3) und
- der zu verwendenden ProF-Konfiguration bei Flash-Testfällen

anzupassen. Des Weiteren wird in der Testkonfiguration

- die Anzahl Zylinder des Motors, für den der aktuelle Programmstand gedacht ist, angegeben,
- der Systemtyp (XCP oder ETK, für die Unterscheidung innerhalb von ECU-Test) festgelegt und

- der Gerätename bzw. bei XCP zusätzlich der Transportlayer (z.B. „ETKC:1“ oder „XCP:1, CAN FD Channel“, für die Zuweisung in INCA) angegeben.

Basierend auf den so vereinheitlichten Daten aus der Testkonfiguration kann der Testablauf dann entweder den bereits vorhandenen INCA-Workspace laden und mit den Angaben aus diesen Daten vergleichen (z.B. ob das INCA-Projekt dieselben Softwareartefakte nennt die Testkonfiguration) oder den Workspace neu anlegen. Einige der Daten, wie die Anzahl der Zylinder oder der Ablageort der Workspace-Datenbank, sind als Informationen für bestimmte Testfälle nötig und können nicht automatisiert ermittelt werden.

5.2.2 Testreport

Obwohl der Testreport laut ISTQB ebenfalls nicht Teil der Testsuite ist, wird er hier, in Anlehnung an die Struktur von ECU-Test, als solcher angenommen.

Es werden drei Arten des Testreports angedacht:

- ein Testreport für interne Zwecke mit ausführlichen Informationen zur Fehlersuche,
- ein Testreport für externe Zwecke, welcher einzelne Testfälle und deren Testschritte bewertet, nicht aber deren genaue Struktur hinterlegt und
- ein Testreport, der den gesamten Testablauf basierend auf den Testschritten einzelner Testskripte für alle Testfälle verbal und unbewertet beschreibt, nach ISTQB eine Testspezifikation.

Interner Testreport Der Testreport für interne Zwecke wird automatisch von ECU-Test in einem proprietären Format nach Testausführung generiert. In einem entsprechenden, mit ECU-Test mitgelieferten „Viewer“ kann der Inhalt angezeigt werden. Er beinhaltet Informationen bis in die niedrigste Ebene. Dazu gehören Informationen über Berechnungen, Job-Aufrufe, Funktions-Aufrufe, Zugriff auf Mess- oder Verstellgrößen, untergeordnete Package-Aufrufe und deren genauer Inhalt sowie die jeweiligen Argumente und Rückgabewerte bzw. Bewertungen all dieser Elemente. Ebenfalls beinhaltet er Graphen aufgezeichneter Messgrößen. ECU-Test bietet hier fünf Bewertungsergebnisse. Tabelle 6 beschreibt, wie diese im Falle dieser Testautomatisierung genutzt werden.

Name	Bedeutung
NONE	Dieser Testschritt ist ein unbewerteter Testschritt.
SUCCESS	Dieser Testschritt wurde erfolgreich und wie erwartet durchgeführt, das SUT hat wie gewünscht reagiert.
INCONCLUSIVE	Dieser Testschritt konnte nicht eindeutig bewertet werden bzw. sein Ergebnis konnte weder dem positiven noch dem negativen Erwartungswert zugeordnet werden.
FAILED	Dieser Testschritt ist fehlgeschlagen, das SUT hat nicht wie gewünscht reagiert.
ERROR	Dieser Testschritt ist fehlerhaft. Ein Fehler in der Testautomatisierung ist aufgetreten, entweder durch einen Fehler des Testers oder durch einen Fehler in der Implementierung ausgelöst.

Tabelle 6: Bewertungsergebnisse und ihre Bedeutung in der Testautomatisierung

In der Umsetzung soll darauf geachtet werden, dass das Bewertungsergebnis *INCONCLUSIVE* nicht vorkommen kann. Ebenso wie das Ergebnis *ERROR*. *ERROR* wird allerdings genutzt, um den Tester darauf hinzuweisen, dass er eventuell bei der Anpassung

der Testsuite einen Fehler gemacht hat. Das könnte z.B. passieren, wenn ein bereits angelegter INCA-Workspace genutzt wird, der andere Softwareartefakte beinhaltet als die Testkonfiguration. Ein weiteres Beispiel wäre ein Projekt mit einem Testfall, der nur für ETK-Systeme gedacht ist, aber in einem SUT mit Zugriff über XCP genutzt wird.

Ein weiterer Teil des Testreports ist die Auflistung von Metadaten. Hierbei gibt es die, die vom Tester manuell in der Testkonfiguration genannt wurden und die, die automatisch erfasst wurden. Zweitere setzen voraus, dass in der Testkonfiguration ein Python-Skript definiert wurde, welches diese erfasst. Das Skript ist so aufzubauen, dass es eine Funktion `GetReportData()` beinhaltet, die ein Dictionary zurückgibt. Der Dictionary-Key ist hierbei der statische Bezeichner für die Information und das Dictionary-Value die ermittelte Information. Innerhalb dieser Funktion können auch Python-Funktionen genutzt werden, die in ECU-Test geladen wurden (z.B. die aus Kapitel 5.1).

Innerhalb des „Viewer“ ist es zudem möglich, Einträge einzelner Testschritte oder auch ganzer Testfälle, zu kommentieren und die Bewertung zu ändern. Bei Änderungen der Bewertung wird auch die Historie mit Datum und Editor gespeichert.

Ein Beispiel für einen internen Testreport ist im Anhang durch drei Screenshots hinterlegt. Dort ist die Gesamtübersicht gezeigt (Abbildung A.1), die Darstellung der Metadaten als „Benutzerdefinierte Informationen“ (Abbildung A.2) und das genaue Testskript eines Testfalles (Abbildung A.3).

Externer Testreport Die Generierung des externen Reports kann automatisch am Ende des Testablaufes ausgeführt werden. Es ist auch möglich den externe Report aus dem „Viewer“ heraus mit dem Internen als Basis zu generieren. Innerhalb der Testkonfiguration steht eine Einstellung zur Verfügung, um zu wählen, ob automatisch ein externer Report erstellt werden soll oder nicht. Der Vorteil bei einer Generierung aus dem internen Report ist, dass ggf. manuell geänderte Bewertungen und Kommentare ihren Weg in den externen Report finden. Somit können z.B. *FAILED*-Testfälle mit einem Kommentar versehen werden, welcher argumentiert, weshalb dieser Fehlschlag zustande kam und warum er irrelevant für die Freigabe ist.

#	Aktion / Name
1	Precondition
12	testvariable read - original value
28	testvariable write - new_value = original value / 2
29	If (-2 < old_value < 2)
30	Then
31	Berechnung
32	Else
34	Ausführung Job JobCall: WriteCalibration
41	restart experiment
51	datapages INCA vs ECU should be different now
55	Download Workbase
69	datapages INCA vs ECU should be equal again
73	Postcondition

(a) Testskript im ECU-Test Package

Testfall	
#	Aktion
1	Precondition
12	testvariable read - original value
28	testvariable write - new_value = original value / 2
41	restart experiment
51	datapages INCA vs ECU should be different now
55	Download Workbase
69	datapages INCA vs ECU should be equal again
73	Postcondition

(b) Testskript im externen Report

Abbildung 17: Vergleich der Darstellung eines Testskriptes: Package und externer Report

Beide Varianten greifen auf eine Ausgabeformatdefinition innerhalb der Testkonfiguration zurück. In dieser soll festgelegt werden, dass der externe Report als HTML/JavaScript-Seite ausgegeben wird, sodass er auch ohne „Viewer“ betrachtet werden kann. Die Informationen innerhalb des Reports adaptieren die meisten Informationen des Internen, werden aber bezüglich der Details einzelner Testschritte beschränkt. Um dennoch lesbar den Ablauf wiederzugeben, soll bei der Umsetzung der Testfälle darauf geachtet werden, dass die genauen Aufrufe von Jobs, Funktionen, etc. immer Kommentarblöcken untergeordnet sind.

Abbildung 17 sollen diesen Zusammenhang darstellen. Links (17(a)) ist der Testfall in ECU-Test so dargestellt, dass einzelne Testschritte, des Testskriptes, in vollen Details gezeigt werden. Rechts (17(b)) ist der Testfall im Testreport demonstriert, der nur noch die oberste Ebene, ohne genauere Details zu den Testschritten, beinhaltet. Alle darunterliegenden Schritte werden ausgeblendet. Die Bewertung der untergeordneten Testschritte wird zusammengefasst angezeigt, wobei immer die schlechteste Bewertung die Gesamtbewertung ergibt. Grün steht hier für *SUCCESS* und Grau für *NONE*.

Testspezifikation Der dritte Typ Testreport der angedacht wurde ist die Testspezifikation. Sie wird aus einem Projekt, das einen Ablauf von Packages beinhaltet, heraus generiert und als HTML/JavaScript-Seite ausgegeben. Damit ähnelt sie dem externen Report, nur ohne Bewertung und Metadaten.

Im Anhang ist ein Screenshot für eine beispielhafte HTML-Testspezifikation mit geöffnetem Testskript zu finden (Abbildung A.4).

5.2.3 Testablauf

Laut ISTQB ist Testablauf ein anderer Begriff für Testsuite. Da Testsuite im Zusammenhang mit ECU-Test neben dem Ablauf zusätzlich diverse Einstellungen und Parameter beinhaltet, wird hier weiter der Begriff Testablauf genutzt, um die Abfolge mehrerer Testskripte zu beschreiben.

Wie in Kapitel 3 bereits diskutiert, sollte die Architektur des Testablaufes es ermöglichen, einzelne Testfälle möglichst unabhängig voneinander ausführen zu können. Damit wäre die Testautomatisierung für Regressionstests geeignet, bei denen nur eine Untermenge aller, für das System relevante, Testfälle getestet wird. Falls ein Testfall Fehler aufzeigt kann er so nach der Problembehebung schnell wieder ausgeführt werden um die Problembehebung zu verifizieren. Da einige Testfälle bereits in den vorhandenen Lösungen über zehn Minuten zur Ausführung benötigen, wäre die Möglichkeit solcher Regressionstests für iterative Fehlersuche/Problembehebung äußerst vorteilhaft. Ein weiterer Vorteil solcher unabhängiger Testfälle ist es, dass der Tester bei der Zusammenstellung eines Ablaufes weniger Fehler machen kann.

Um also zu erreichen, dass die Testfälle unabhängig voneinander funktionieren und möglichst nicht von den Testfallnachbedingungen anderer Testfälle abhängig sind, wurden die Testfallvor- und Testfallnachzustände genauer untersucht. Dabei hat sich gezeigt, dass die meisten Testfälle ihre Vorbedingung auf einen Vorzustand beziehen, welcher einen Gesamtzustand der Toolkette beschreibt. Genauer sind dabei folgende Zustände aufgefallen:

- „Alloff“ - T30 und T15 sind aus, die ECU ist aus, das Schnittstellenmodul ist versorgt und ist vom Prüfrechner erkannt, da die ECU aus ist besteht aber kein Zugriff in INCA
- „ECUOff“ - T30 ist an, T15 ist aus, die ECU ist ebenfalls aus, das Schnittstellenmodul ist versorgt und ist vom Prüfrechner erkannt, da die ECU aus ist besteht aber kein Zugriff in INCA
- „CalWup“ - T30 is an, T15 ist aus, die ECU wurde per CalWup geweckt und es besteht Zugriff auf die ECU in INCA
- „Calib“ - T30 und T15 sind an, die ECU ist an, es besteht Zugriff auf die ECU in INCA
- „PostDrive“ - T30 is an, T15 ist aus, die ECU ist noch an und befindet sich im PostDrive, es besteht Zugriff auf die ECU in INCA

Die erste Auffälligkeit besteht darin, dass in jedem der fünf Zustände das Schnittstellenmodul versorgt und vom Prüfrechner erkannt sein muss. *CalWup*, *Calib* und *PostDrive* erwähnen das zwar nicht explizit, es ist aber implizit durch den bestehenden Zugriff von INCA auf die ECU gegeben. Daraus folgt, dass lediglich zu Beginn des Testablaufes einmalig die Versorgung für das Schnittstellenmodul aktiviert werden muss. Daraufhin kann mit der in Kapitel 5.1 angesprochenen Funktion zur Abfrage der verbundenen Schnittstellenmodule via „IP-Manager“geprüft werden, ob das Schnittstellenmodul verbunden ist. Ein explizites „Setup“-Package welches in jedem Testablauf an erster Stelle stehen muss kann diesen Zustand herstellen.

Der Unterschied zwischen *Alloff* und *ECUOff* ist nur für den Systemtyp ETK relevant, da das ETK durch das Schnittstellenmodul versorgt wird, sobald T30 an ist. Für den Systemtyp sind diese beiden Zustände als funktional gleichwertig zu betrachten.

Bei der Auflistung der Zustände fällt zudem auf, dass einige nur erreichbar sind, indem vorher andere erreicht wurden. So ist z.B. für ETK-Systeme *CalWup* nur erreichbar, wenn vorher *ECUOff* als Zustand hergestellt wurde. Eine Modellierung dieser Zustände als Zustandsmaschine ist naheliegend. Abbildung 18 zeigt das Ergebnis dieser Modellierung für ein ETK-System.

Die Darstellung zeigt hierbei alle fünf angesprochenen Zustände sowie den Zustand, in dem sich die Toolkette zu Beginn des Ablaufes befindet. An den Übergängen sind die Aktionen dargestellt, die für die jeweiligen Transition nötig sind. „HW-Init“ beschreibt hierbei den Vorgang der Hardwareinitialisierung in INCA, der über die API oder bei geöffnetem Experiment manuell ausgeführt werden kann. „Warten“ soll darstellen, dass der Übergang automatisch stattfindet, sobald eine gewisse Zeit²⁰ vergangen ist.

Der Übergang zu Beginn von Start nach *Alloff* wird durch das bereits angesprochene Setup-Package realisiert. Von dort an ist der aktuelle Zustand abhängig vom vorangegangenen Testfall und der gewünschte Zustand ist abhängig vom aktuellen Testfall. Daraus folgt, dass in der Vorbedingung jedes Testfalles eine Prüfung des aktuellen Toolkettenzustandes stattfinden muss, um dann, entsprechend der Modellierung, die richtigen Aktionen ausführen zu können, die den gewünschten Toolkettenzustand erzeugen.

²⁰Diese Zeit variiert im Serienfahrzeug in Abhängigkeit diverser Temperaturwerte und einiger anderen Variablen. Die entsprechenden Größen sind im Testaufbau allerdings nicht definiert, weshalb die Zeit nicht genau vorhergesagt werden kann. Um ein Abschalten dennoch schnell erkennen zu können hilft die in Kapitel 5.1.3 angesprochene Funktion zur Abfrage des HR.

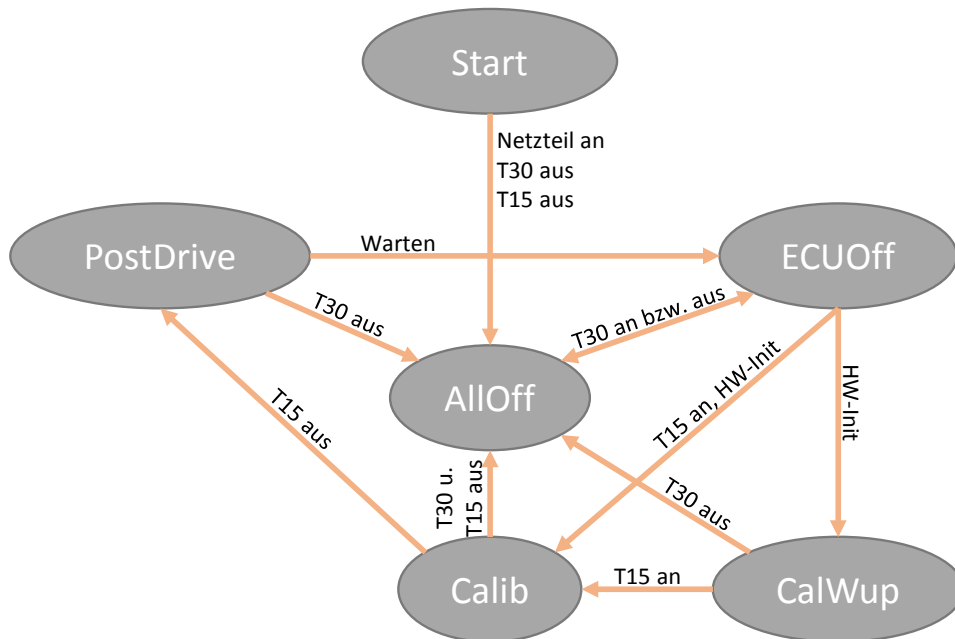


Abbildung 18: Zustandsmaschine für ETK-Systeme

Im Graph sind nicht alle Übergänge eingezeichnet, die denkbar sind, sondern ausschließlich die, die auch genutzt werden sollen. In diversen Zuständen wäre es z.B. möglich T15 anzuschalten während T30 noch an ist. Eine solche Aktion würde allerdings keine Reaktion erzeugen, könnte ggf. aber bei darauffolgenden Übergängen für Fehler sorgen. Um solche Fehler zu vermeiden sollte, nach der Feststellung des aktuellen Zustands, die Relais T30 und T15 zur Absicherung in den erwarteten Schaltzustand geschaltet werden auch wenn sie normal bereits in diesem Schaltzustand seien sollten. Diese Absicherung ist vor allem bei XCP-Systemen nötig, bei denen die Zustände *ECUOff* und *AllOff* zusammengefasst werden.

Abschließend lässt sich also sagen, dass die Problematik der Vorbedingungen durch ein Package gelöst werden kann, welches am Anfang jedes Testfalles steht. Dieses Package prüft über den „IP-Manager“, ob ein Schnittstellenmodul verbunden ist und ermittelt dann den aktuellen Zustand der Toolkette. Dem Package wird vom Testfall ein Argument „Wunschzustand“ übergeben, der dann entsprechend der Modellierung als Zustandsmaschine hergestellt wird. Innerhalb des Packages wird eine Fallunterscheidung, basierend auf einer globalen Konstante, getroffen, ob ein ETK- oder ein XCP-System vorliegt. Diese Fallunterscheidung ermöglicht es zusätzliche Testfälle zwischen beiden Use-Cases zu harmonisieren, die sich lediglich in ihrem Testskript beim Erreichen eines bestimmten Zustandes unterscheiden.

Ein Beispielablauf soll den so erzielten Umgang mit Vor- und Nachbedingungen verdeutlichen:

- Testfall 1 wird im Zustand *Calib* verlassen,
- Testfall 2 ruft dann in der Vorbedingung das Package mit dem Wunschzustand *CalWup* auf.

Das Package stellt nun in Testfall 2 zuerst fest, dass der aktuelle Zustand *Calib* ist und wählt entsprechend der modellierten Zustandsmaschine den kürzesten Weg, um in

den Zustand *CalWup* zu kommen. Hierzu wurde innerhalb von Switch-Case Anweisungen für jede Ausgangszustands-Zielzustands-Kombination der kürzeste Weg entsprechend der Zustandsmaschine hinterlegt. Somit ist egal, mit welchem Toolkettenzustand ein Testfall verlassen wird, da jeder Testfall in seiner Vorbedingung durch das Package den für ihn nötigen Zustand herstellt.

Neben der Problematik der Vor- und Nachbedingungen bezüglich des Toolkettenzustand gibt es aber auch relevante Vor- und Nachbedingungen, die sich auf Unterschiede zwischen Daten auf Referenz- und Arbeitsseite beziehen. Wenn in einem Testfall Verstellgrößen verändert wurden, gibt es eine Differenz zwischen AS und RS. Diese Differenz blockiert diverse INCA-Funktionalitäten. Also muss zusätzlich in der Nachbedingung von Testfällen, welche Verstellgrößen verändern, die AS zurückgesetzt werden bzw. die Inhalte der RS auf die AS kopiert werden.

6 Umsetzung

Im V-Modell befindet sich dieses Kapitel im Abschnitt „Erstellung, Implementierung“, „Teil-Realisierung- Integration und Verifikation“ und „Subsystem- Integration und Verifikation“. Die eigentliche Implementierung behandelt die Umsetzung der Testfälle aus der Testfallsammlung (Kapitel 4) mit Hilfe der bereits umgesetzten und der nativ in ECU-Test vorhandenen Funktionen (Kapitel 5.1). Damit ist eine Integration und Verifikation dieser Subsysteme bereits in diesem Kapitel gegeben. Tests der Umsetzung der Testfälle an Versuchsaufbauten (vgl. Kapitel 2.10.1) für ETK-Systeme und XCP-Systeme sind angedacht. Somit ist auch die Umsetzung der Ablaufarchitektur (Kapitel 5.2.3) für ganze Use-Cases (Kapitel 3.1) in diesem Kapitel erwähnt, also eine Integration und Verifikation von Subsystemen.

6.1 ETK-System

Da die bereits vorhandenen Lösungen (Kapitel 3.2) für XCP-Systeme umfangreicher ausfallen, soll mit der Umsetzung für ETK-Systeme begonnen werden. Das verwendete ETK-System ist ein bereits freigegebenes und damit geeignet alle Testfälle zu prüfen.

6.1.1 Umsetzung: Konfiguration und Setup

Um in ECU-Test Packages ausführen zu können sind eine Testkonfiguration und eine Testbenchkonfiguration nötig. Zuerst wurde die Testkonfiguration angelegt. Aufgrund der Architekturüberlegungen in Kapitel 5.2.1 war die Umsetzung dieser problemlos und direkt möglich. Die nötigen globalen Konstanten (siehe Abbildung 19) wurden angelegt und die Informationen eingepflegt. Felder für Metadaten im Testreport wurden entsprechend der Vorlage der ETK-Freigabecheckliste hinzugefügt. Ein Python-Skript wurde in der Konfiguration hinterlegt, welches alle dynamisch erfassbaren Metadaten für den Report erzeugt (Tester, Computernamen und -hersteller, Uhrzeit und Datum, Betriebssystem, verwendete Pfade, ECU-Test und INCA(-API) Versionen). Ein Ausgabeformat für die vereinfachte HTML-Darstellung (Kapitel 5.2.2) wurde in der Konfiguration definiert.

Globale Konstanten		
Bezeichner	Wert	Beschreibung
G_DeviceName	'ETK:1'	INCA DeviceName of the device to be used
G_INCADB	'test-db'	INCA-DB name; the name of the folder in the G_INCARoot directory which contains the INCA-DB to be used
G_INCAExperiment	'ECU-Test_Exp'	INCA-Experiment to be used
G_INCAFolder	'ECU-Test_Aut'	Mainfolder of the INCA-DB which contains/should contain the Experiment, the Workspace and the Project
G_INCARoot	'C:\NETASData\INCA7.2\Database'	Windowspath to the INCA-DB files
G_INCAWorkspace	'ECU-Test_Ws'	INCA-Workspace to be used
G_ProFPath	'C:\NETASDATA\ [REDACTED]'	Path to the ProF-Config that should be used to flash
G_PVERPath	'D:\Programmstaende\ [REDACTED]'	Path of the PVER to be used (needed to find the workunit.lws.cc.db3)
G_SwitchboxRPMType	'7'	A char describing the RPM-type (Project) the Switchbox should use to set the RPM [REDACTED]
G_SynchroTicks	12	Number of cylinders times the events per synchro in the project to be tested (Cylinder * Events_per_Synchro)
SYSTEMTYPE	'ETK'	System Under Test: ETK, XCP

Abbildung 19: Konfiguration über globale Konstanten

Das Anlegen einer Testbenchkonfiguration fiel, dank der Architekturüberlegungen, ebenso direkt aus. Der Ansatz der Auslagerung von Einstellungen der Testbenchkonfiguration in globale Konstanten der Testkonfiguration wurde für alle Einstellungen verfolgt, die für unterschiedliche SUTs angepasst werden müssten.

Alle drei so erstellte Dateien (Testkonfiguration, Testbenchkonfiguration und Reporterstellungsskript) wurden als ETK-spezifisch gespeichert.

Vor der Umsetzung der einzelnen Testfälle wurden diverse Subpackages angelegt. So wurde für jede der erweiterten INCA-Funktionen (Kapitel 5.1.3) ein Package umgesetzt, das als Wrapper dient. Die Nutzung von Packages als Wrapper für den Aufruf benutzerdefinierter Pythonfunktionen verbessert die Lesbarkeit von Testfällen, die diese Funktionen benötigen. Des Weiteren fungieren die Wrapper-Packages als zusätzliche Abstraktionsebene, falls sich eine Pythonfunktion ändert. Durch eine geschickte Anpassung der Wrapper-Packages kann vermieden werden, dass jeder Testfall, der die Funktion nutzt, angeglichen werden muss.

ECU-Test funktioniert in Kombination mit INCA so, dass am Anfang des Testes das entsprechende INCA-Experiment geöffnet und erst zum Ende des Testes wieder geschlossen wird (normale Vorgehensweise in SIL- und HIL-Anwendungen). Für einige Testfälle dieser Testautomatisierung muss das Experiment allerdings während eines Ablaufes zwangsweise geschlossen werden, um einen Absturz zu simulieren. Um das zu erreichen muss der INCA-Tooladapter in der ECU-Test Testbenchkonfiguration gestoppt und neugestartet werden. Dieser Neustart kann normalerweise nur auf Projektebene automatisiert angestoßen werden (also zwischen einzelnen Testfällen) und nicht innerhalb eines Packages (Testfall). Ein solches Vorgehen würde einen sehr fehleranfälligen Ablauf darstellen. Nach einiger Recherche hat sich eine Funktionalität offenbart, die über die ECU-Test interne Python-API eine Methode zur Verfügung stellt, die einzelne Tooladapter starten/stoppen/neu starten kann. Da sich diese Funktionalität auch für eine bessere Strukturierung des Setup-Packages anbietet, wurde der jeweilige API-Aufruf für die einzelnen Tooladapter ebenfalls in, als Wrapper fungierende, Subpackages geschrieben. Die Recherche in der Dokumentation, der internen API, hat zudem eine Methode gezeigt, mit der der aktuelle Zustand eines Tooladapters abgefragt werden kann. Es wurde ein Subpackage angelegt, welches mit der Hilfe der Methode abfragen kann, ob der Tester vielleicht das INCA-Experiment geschlossen hat, obwohl ECU-Test noch davon ausgeht, dass es Zugriff darauf hat.

In Kapitel 5.2.3 wurde besprochen, wie der gesamte Toolkettenzustand durch ein Package zur Manipulation der Zustandsmaschine umgesetzt werden kann. Das Ergebnis dessen Umsetzung gehört ebenfalls in die Bibliothek der Subpackages und nutzt neben der absichernden „IP-Manager“-Abfrage auch die Abfrage für den Status des INCA-Tooladapters.

Ebenfalls entsprechend der Architekturüberlegungen wurde ein Setup-Package angelegt, welches die nötigen Tooladapter startet und die Toolkette in den *AllOff*-Zustand bringt. Ein zweites Package wurde realisiert, welches dem Setup-Package im Ablauf immer folgen sollte. Dieses Package setzt diverse geforderte INCA-Optionen.

Ein Projekt wurde erstellt, welches später den Beispielablauf für die Freigabe eines ETK-Systems beinhalten soll. Im ersten Schritt stellt dieses Projekt allerdings nur einen Zustand der Testautomatisierung her, aus dem heraus einzelne Testfälle nach ihrer Implementierung getestet werden können. Dazu werden vom Projekt die erstellte ETK Testkonfiguration und Testbenchkonfiguration geladen und das Setup- und Options-Package ausgeführt.

Abbildung A.5 im Anhang zeigt dieses Projekt. Dem Options-Package folgen in diesem Beispielscreenshot bereits ein Großteil der Testfall-Packages.

6.1.2 Umsetzung: Testfälle

Mit der nun geschaffenen Grundlage konnten die Testfälle sehr direkt, entsprechend der Anforderungen an sie aus der Testfallsammlung, umgesetzt werden. Genauere Informationen über den Ablauf innerhalb eines Testskriptes wurden gleichermaßen aus der Testfallsammlung, den bereits bestehenden Lösungen und dem ständigen Dialog mit den Experten gewonnen. Da viele der Abläufe tiefgreifendere Kenntnisse der Toolkette erfordern, als sie ohne Weiteres innerhalb dieser Arbeit aufgearbeitet werden könnten, hat die Ermittlung der Informationen für die Abläufe einen Großteil der Zeit der Umsetzung beansprucht. Folgend soll daher nur beispielhaft auf die genaue Umsetzung ausgewählter Testfälle eingegangen werden.

T100.1(CalibTest) Dieser Testfall dient dazu, den Zugriff auf Verstellgrößen zu testen. Im Vorfeld zum Test für die Freigabe werden hierzu Testgrößen in den Programmstand implementiert. Diese bestehen je aus einer Verstell- und einer Messgröße, welche direkt verknüpft sind. Somit kann das Verstellen einer Verstellgröße direkt durch das Überprüfen der zugehörigen Messgröße verifiziert werden. Es gibt Testgrößen für alle Datentypen (außer *double*), die der auf der ECU verwendete 32bit-Mikrocontroller unterstützt. Der *double*-Datentype (bzw. float64) wird ausgeschlossen, da er nicht von CAN übertragen werden kann und deshalb auch für die Nutzung in ECU-Programmen, durch die entsprechenden Coding Guidelines, verboten ist. Konkret bleiben die Datentypen *ubyte*, *uword*, *ulong*, *sbyte*, *sword*, *slong* und *float32*. Neben Testgrößen mit diesen Datentypen als simple Skalare gibt es auch Strukturen, Arrays, Kennlinien und Kennfelder (ein- und zweidimensionale Lookup-Tabellen). Als Beispiel für die Umsetzung wird hier auf das Testen der Arrays eingegangen, zum gesamten Testfall gehören aber auch Tests für die anderen Verbunddatentypen und für die Skalare. Die Testgrößen-Arrays haben immer eine Dimension von $n = 2$.

Zuerst wird in der Vorbedingung der Toolkettezustand *Calib* hergestellt. Dann wird die Messung gestartet, sodass Messgrößen ausgelesen werden können. Ein Wechseln des Zugriffes auf die AS lässt Verstellen zu. Nun werden die Werte festgelegt, mit welchen die einzelnen Datentypen getestet werden sollen. Im Bezug auf Kapitel 4.3 werden hierbei vor allem Werte geprüft, die Grenzwerte für den jeweiligen Datentyp darstellen. Für die *unsigned* Datentypen werden also von der Anzahl der Bits (*bits*) abhängig die vier Werte

$[0, 2^{\text{bits}-1}, 2^{\text{bits}/5*2}, 2^{\text{bits}/5*3}]$

geprüft, wobei die letzteren Zwei gerundet werden. Sie stellen Zwischenwerte neben den Grenzwerten dar. Im Falle der *signed* Datentypen werden die vier Werte

$[0, -1, 2^{\text{bits}/2-1}, -2^{\text{bits}/2}]$

genutzt. Sie stellen einmal die dezimalen und einmal die binären Grenzwerte des Datentyps dar. Für den *float32* Datentyp werden die Werte

$[0.0, 1000, -3.031649\text{E-}13, 1\text{e-}37]$

geprüft. Der letzte Wert für den *float32* Datentyp stellt den Grenzwert dar, bei welchem eine Überprüfung mit einer Toleranz von zehn Nachkommastellen noch positiv ausfällt. Die drei hier angedeuteten Listen ergeben für alle Datentypen sieben Listen. Diese sieben Listen werden in einem Dictionary (*testvalues*) gespeichert, wobei der jeweilige Datentyp der Key ist und die Liste das Value.

Ebenfalls entsprechend Kapitel 4.3 werden auch immer erst alle Verstellgrößen der Testgrößen verstellt und dann die Messgrößen überprüft. Da die Verstellgrößen im Speicher des Mikrocontrollers als direkt benachbart angelegt wurden, kann so sichergestellt werden, dass der Verstellzugriff nicht falsche Speicherbereiche beschreibt.

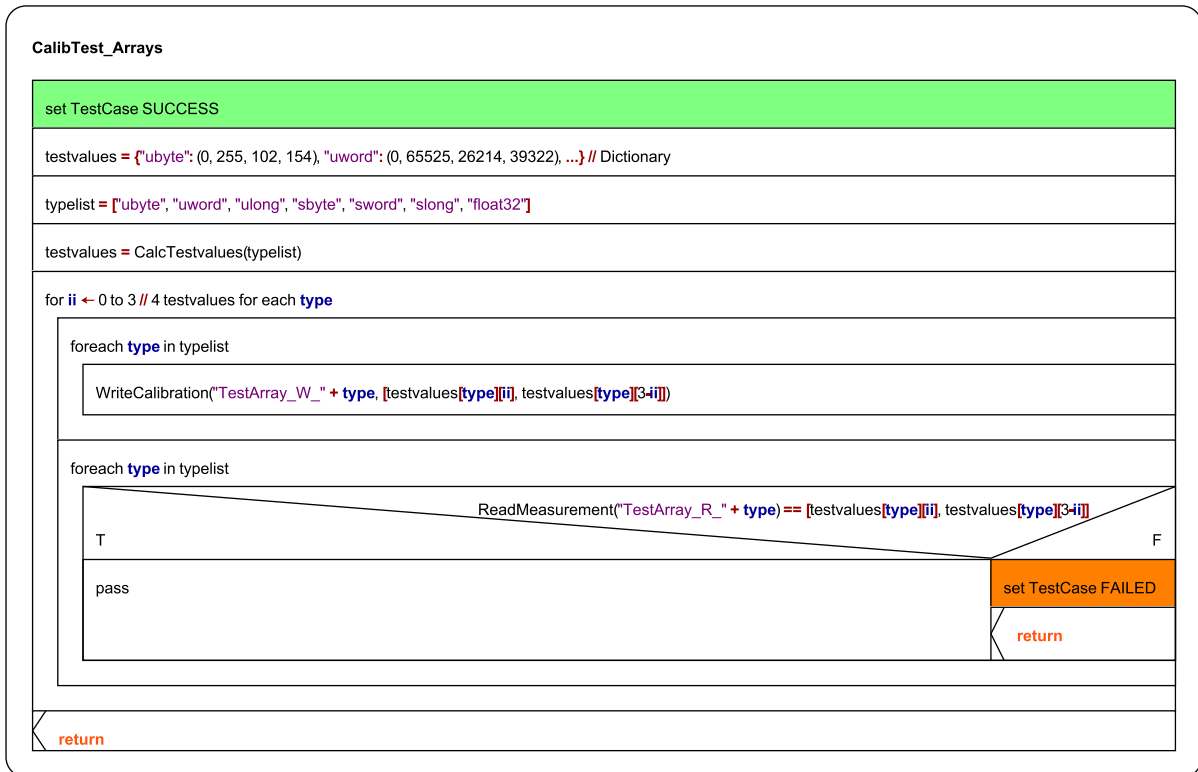


Abbildung 20: Struktogramm für das Testskript *Arrays* als Teil von T100.1(CalibTest)

Es ergibt sich ein Testskript, dessen Ablauf sich gut als Struktogramm darstellen lässt (siehe Abbildung 20).

`CalcTestvalues()` stellt im Struktogramm die Funktion dar, die basierend auf der Liste der gegebenen Datentypen in `typelist`, besagte zu testende Werte berechnet und in ein Dictionary speichert. `ReadMeasurement()` und `WriteCalibration()` sind die Funktionen, die den Zugriff auf die ECU Mess- und Verstellgrößen herstellen. Das erste Argument ist ein String, der den Bezeichner für die entsprechende Größe beinhaltet. Dieser String wird dynamisch aus mehreren Substrings zusammengesetzt. Falls sich die Testsoftware in Zukunft ändert, sind diese Substrings über entsprechende Einstellungen im Package-Header²¹ anpassbar. Das zweite Argument, im Falle von `WriteCalibration()`, beschreibt die zu schreibenden Werte für das Array. Wird im zweiten `foreach`-Loop eine Unstimmigkeit erkannt, wird das Ergebnis des Testfalles auf `FAILED` gesetzt, der Testfall aber weiter abgearbeitet. So kann später im internen Testreport nachvollzogen werden, wo genau Fehler auftraten.

Ist der Testfall abgeschlossen, muss die Messung wieder gestoppt werden und durch ein Kopieren von RS nach AS alle Verstellungen rückgängig gemacht werden.

²¹in jedem Package können Konstanten mit Beschreibungen hinterlegt werden und als Einstellparameter für den Testfall markiert werden

T107.1(Synchroperiode) Um Messgrößen aus der ECU auszulesen und aufzuzeichnen bzw. in INCA anzuzeigen werden Messraster genutzt. So ist es in ETK-Systemen bspw. möglich Messgrößen mit einer Abtastrate von $1ms$, $5ms$, $10ms$, $20ms$, $100ms$ und $1000ms$ aufzuzeichnen/abzufragen. Neben diesen zeitsynchronen Messrastern gibt es auch Messraster, welche von der aktuellen Motordrehzahl abhängig sind, sogenannte segmentsynchrone Messraster.

Im Testfall „Synchroperiode“ wird sichergestellt, dass sich diese segmentsynchronen Messraster entsprechend der Drehzahl anpassen. Es gibt in ETK-Systemen zwei solche Messraster, welche konfigurierbar phasenverschoben Werte liefern.

Hierzu wird erst in der Vorbedingung die Toolkette in den Zustand *Calib* gebracht. Per entsprechendem Switchbox-Tooljob wird dann die Drehzahl auf $0min^{-1}$ gesetzt. Es wird eine Aufzeichnung in INCA gestartet, welche „Testcounter“ (die Testcounter inkrementieren sich um 1 für jeden Tick des verknüpften Messrasters), für beide Messraster, aufzeichnet und dann nach immer $5s$ die Drehzahl auf je $1000min^{-1}$, $3000min^{-1}$ und $5000min^{-1}$ anhebt. Diese drei Drehzahlstufen sind anpassbar, sollten aber für die meisten Programmstände innerhalb der theoretisch möglichen Drehzahl liegen. In der Nachbedingung des Testskriptes wird die Aufzeichnung dann wieder gestoppt und die Drehzahl zurück auf $0min^{-1}$ gesetzt.

Es folgt die Ausführung einer sogenannten *Traceanalyse*. Die Messung wird für die Traceanalyse zuerst wieder in vier Abschnitte getrennt ($0min^{-1}$, $1000min^{-1}$, $3000min^{-1}$ und $5000min^{-1}$). Dann werden die einzelnen Abschnitte der Aufzeichnung durch ein Pythonskript überprüft. Die Überprüfung bewertet, ob die Testcounter ohne Sprünge immer um $+1$ inkrementieren und innerhalb der je $5s$ -Abschnitte ein Min-Max-Delta aufweisen, welches der Gleichung 1 entspricht. Im Prinzip werden die Counter für jeden Arbeitstakt einmal inkrementiert. Bei einem Viertaktmotor mit vier Zylindern sind das zwei Arbeitstakte je Umdrehung und somit auch zwei Inkremente. Bei einer Drehzahl von $0min^{-1}$ arbeiten die segmentsynchronen Messraster nicht, dementsprechend stehen auch die verknüpften Testcounter.

$$\Delta = \text{Drehzahl} \cdot \text{Abschnittsdauer} \cdot \text{AnzahlZylinder}/2 \cdot \text{Events_pro_Synchro} \quad (1)$$

Da die Testcounter teilweise leicht von diesem Delta abweichen, wird bei der Bewertung eine relative Toleranz von $\pm 10\%$ zugelassen bzw. im Bereich von $0min^{-1}$ eine absolute Toleranz von ± 10 . Die für die Berechnung nötige Information über die Anzahl der Zylinder des Programmstandes ist Teil der globalen Konstanten aus der Testkonfiguration (siehe Kapitel 5.2.1).

Ein Überlauf der Testcounter kann ausgeschlossen werden, da die Counter vom Datentyp *unsigned long* (4 Byte) sind. Die Testcounter beginnen in diesem Testschritt bei 0, da es keinen anderen Testschritt gibt, welcher die Drehzahl über $0min^{-1}$ setzt. Daraus folgt, dass selbst wenn in allen drei Bereichen über $0min^{-1}$ die Drehzahl $8000min^{-1}$ ²² betragen würde, die Counter entsprechend Gleichung 1 während dieser $15s$ nur den Wert 2000 pro Zylinder erreichen (in Dieselmotoren ist der Wert *Events_pro_Synchro* = 2, in Benzinmotoren 1). Der Datentyp *unsigned long* hat sein Maximum bei ca. $4 \cdot 10^9$.

²²die maximale Drehzahl, welche über die Switchbox gesetzt werden kann, unabhängig vom Programmstand

T104.1(WorkbaseDownload) Der Testfall für die Überprüfung der Funktionalität zum Herunterladen der Arbeitsseite (WorkbaseDownload) fällt, verglichen mit den zwei zuvor erwähnten, simpel aus.

In der Vorbedingung dieses Testfalls wird der Toolkettenzustand *Calib* hergestellt. Nachdem explizit überprüft wurde, ob die Prüfsummen von AS und RS dieselben sind, sowie die Prüfsummen der Seiten auf der ECU dieselben sein müssen wie die in INCA, kann die Messung gestartet werden und auf die Arbeitsseite als aktive Seite gewechselt werden.

Nach der erfolgreichen Ausführung der Nachbedingung wird eine skalare Messgröße der Testgrößen, sword, die auch für T100.1 verwendet wurden, ausgelesen. Erfüllt der ausgelesene Wert die Bedingung $-2 < Wert < 2$, so wird die entsprechende Verstellgröße auf $Wert + 10$ gesetzt. Ist die Bedingung nicht erfüllt, so wird die Verstellgröße auf die abgerundete Hälfte des originalen Wertes verstellt. Somit ist sichergestellt, dass nun die Verstellgröße einen anderen Wert beinhaltet als zuvor, unabhängig davon, wie der alte Wert war.

Der nächste Schritt im Testskript ist das Neustarten des INCA-Experiments. Damit wird eine Unstimmigkeit zwischen den AS in INCA und auf der ECU erzeugt. Diese kommt dadurch zustande, dass INCA beim Experimentneustart ohne explizites Speichern der aktuellen AS als Datensatz die Änderungen durch Verstellen vergisst. Die ECU hingegen läuft während des Neustarts „normal“ weiter und behält die Änderung an der Verstellgröße. Die Prüfsumme der AS in INCA ist eine andere, als die Prüfsumme der AS auf der ECU.

Nun wird die Funktion zum Herunterladen der AS und RS von INCA in die ECU angestoßen (vgl. Kapitel 2.7). Nachdem die Prüfsummen über die entsprechende Funktion neu berechnet wurden, müssen sie nach erfolgreichem Herunterladen wieder dieselben sein.

In der Nachbedingung wird die Messung gestoppt. Ein explizites Kopieren der RS auf die AS zum Herstellen der Ausgangswerte ist nicht nötig, wenn der Testfall erfolgreich durchgeführt wurde. Um allerdings auch im Fehlerfall mögliche Veränderungen an der AS zurückzusetzen wird ein Kopieren dennoch durchgeführt.

T107.2 - T107.6 In der Testfallsammlung sind einige Testfälle zu Dauermessungen und Belastungsmessungen angedacht. Die Dauermessungen werden, wie bereits in Kapitel 4.3 erwähnt, nicht betrachtet. Belastungstests werden in dieser Testautomatisierung auch nicht umgesetzt. Der Grund dafür liegt hauptsächlich darin, dass sie oft völlig separat vom eigentlichen Freigabeprozess mit eigenen Hilfstools durchgeführt werden. Zudem lassen sich die dafür nötigen Abläufe nur schwer ganz automatisiert umsetzen. Ein Belastungstest überprüft bspw. die Prozessorlast bei normaler Programmausführung ohne jeglichen Kalibrierzugriff mit der Prozessorlast bei Zugriff mit maximaler Messdatenrate. Zur Umsetzung dieses Testfalles wäre Debuggerzugriff nötig, um die Prozessorlast auslesen zu können, wenn kein Kalibrierzugriff besteht. Um dann die Last bei maximaler Messdatenrate zu ermitteln müsste erst die maximale Messdatenrate erfasst werden. Dazu würde ein INCA-Experiment iterativ so lange mit weiteren Messgrößen gefüllt werden, bis die Kommunikation Aussetzer feststellen lässt. Für einen solchen Test sind genug Messgrößen nötig, die zusätzlich (künstlich) in der *.a2l-Datei angelegt werden müssen. In Summe sind das viele zusätzlich nötige Funktionalitäten, die innerhalb der Testautomatisierung und in der Toolkette vorhanden sein müssten. Da solche Tests meist bereits

anderweitig durchgeführt werden, wären die entsprechenden Erweiterungen in der Testautomatisierung den Aufwand nicht wert.

6.1.3 Validierung der Testfälle

Auf Basis des nun fertigen Ablaufs für den Use-Case ETK-System konnte ein vollständiger Testablauf gestartet werden. Die Ergebnisse der einzelnen Testfälle im Report decken sich dabei mit den Ergebnissen aus der originalen Freigabe. Unterschiedliche Ablaufreihenfolgen wurden getestet ohne Abweichungen festzustellen. Einzelne Testfälle wurden, soweit möglich, an negativ Beispielen ausgeführt, um die Fehlererkennung zu validieren.

6.2 XCP-System

Nachdem alle Testfälle für den Use-Case ETK-Systeme umgesetzt wurden, ist nun zu betrachten, inwiefern sich diese wie geplant auch für XCP-Systeme nutzen lassen. Diese Betrachtung kann im V-Modell als „Subsystem- Integration und Verifikation“ angesehen werden, da sie sich vor allem damit beschäftigt, inwiefern die Harmonisierung von ETK- und XCP-Testfällen aus Kapitel 4 gelungen ist. Kapitel 4 lag im V-Modell auf der gleichen Ebene im analytischen (fallenden) Ast.

6.2.1 Anpassung der Zustandsmaschine

Wie in Kapitel 5.2.3 erwähnt, muss für eine Übertragung der Testfälle für das ETK-System zum XCP-System das Setup-Package angepasst werden, da die modellierte Zustandsmaschine kleine Unterschiede aufweist. Ein bereits erwähnter Unterschied ist die Unterscheidung zwischen den Zuständen *AllOff* und *ECUOff*. Der Übergang aus diesem zusammengeführten Zustand in den *CalWup* kann in XCP-Systemen nicht durch einen HW-Initialisierung in INCA ausgelöst werden. Um in den CalibrationWakeup zu kommen muss eine Signalleitung auf 5V gezogen werden. Dafür wird im Testaufbau für XCP-Systeme das Relais der Switchbox genutzt, welches normalerweise das Hauptrelais simuliert.

Im Setup-Package wird also eine zweite Variante zur Ansteuerung der Zustandsmaschine implementiert. Durch die globale Konstante für den Systemtyp wird dann innerhalb des Setup-Packages entschieden, welche Variante genutzt werden soll. Somit sollten die meisten Testfälle, die ja alle das Setup-Packages nutzen um den Toolkettenzustand zu manipulieren, ohne weitere Änderungen für XCP-Systeme nutzbar sein.

6.2.2 Anpassung im Versuchsaufbau in HW und SW

Zur Überprüfung wurde der HW-Versuchsaufbau geändert, wobei

- die Toolkette mit einem anderen Schnittstellenmodul ausgestattet wurde (siehe Kapitel 2.10.1),
- die Switchbox so angebunden wurde, dass die HR-Simulation ignoriert²³ und das Relais für den CalWup genutzt wurde und
- eine andere ECU angeschlossen wurde, für die ein funktionierender Programmstand zur Verfügung stand, welche XCP unterstützt.

²³während in ETK-Systemen die Rückmeldung des Relais wichtig ist, damit die ECU nicht gleich wieder ausgeht, ist diese Rückmeldung in XCP-Systemen irrelevant

Wie auch schon für den Use-Case ETK-System wurde eine Testkonfiguration und eine Testbenchkonfiguration angelegt, in welcher die entsprechend angepassten Dateipfade und Informationen zum Programmstand angepasst wurden. Die Konfiguration für XCP-Systeme muss zusätzlich die Information für den zu verwendenden Transportlayer beinhalten (vgl. Kapitel 5.2.1). Als Skript zur automatisierten Erfassung von Metadaten wird dasselbe wie für ETK-Systeme verwendet. Die Ausgabeformatdefinition wird ebenfalls direkt übernommen. Wie auch schon im Use-Case ETK-System wurden die so angelegten Testkonfigurations- und Testbenchkonfigurations-Dateien als XCP-spezifisch gespeichert. Es wurde je eine XCP-Version des Setup- und Options-Packages angelegt, in denen kleine Änderungen gegenüber den ETK-Versionen vorgenommen wurden.

Aus diesen neuen Konfigurationen und dem Setup- und Options-Package entstand ein neues Projekt in ECU-Test. Diesem wurden dann die harmonisierten Testfälle hinzugefügt. Der entstandene Ablauf ist im Anhang A.6 gezeigt. Dort kann er auch mit dem Ablauf für ETK-Systeme verglichen werden (Abbildung A.5).

6.2.3 Validierung der Funktion der harmonisierten Testfälle

Ein Ausführen des neuen XCP-Testablaufes hat bestätigt, dass die Harmonisierung der Testfälle und die Kapselung der Vorbedingungen in ein Setup-Package, eine Testfallbibliothek implementieren ließen, die systemunabhängig funktioniert. Fast alle Testfälle wurden ohne *ERROR* durchgeführt. Probleme gab es nur bei den Testfällen, die Messraster nutzen, welche XCP nicht unterstützt. So unterstützt XCP nur einen Teil der Messraster, die mit einem ETK möglich sind, da ein ETK deutlich höhere Datenraten ermöglicht. Durch die globale Konstante für den Systemtyp konnte das Problem aber schnell behoben werden indem, basierend auf der Konstante, die betroffenen Messgrößen einfach in ein unter XCP vorhandenes Messraster gelegt wurden. Im Falle des Testfalles für die Synchronperiode (T107.1) wurde einfach die Überprüfung des zweiten segmentsynchronen Messrasters, in Abhängigkeit von der Konstante, ausgeblendet, da XCP nur eines unterstützt.

6.2.4 Umsetzung: XCP-spezifische Testfälle

Abschließend wurden die nicht harmonisierbaren Testfälle für XCP umgesetzt. Hierbei handelt es sich vor allem um Testfälle, wie sie bereits am Ende von Kapitel 4.1 erwähnt wurden. Die XCP-Kommunikation besteht nur, wenn in INCA ein Experiment geöffnet ist. Ist es das nicht und die XCP-Kommunikation wird dementsprechend ausgesetzt, so geht eine ECU, die sich im CalWup befindet, nach 60s aus. Das wird in drei Testfällen getestet:

- Erst wird der CalWup ausgelöst, dann wird für 55s ohne offenes Experiment gewartet. Wird nach dieser Zeit das Experiment wieder geöffnet, muss die ECU noch an sein und INCA sich verbinden lassen. (T101.2_XCP)
- Erst wird der CalWup ausgelöst, dann wird für 65s ohne offenes Experiment gewartet. Wird nach dieser Zeit das Experiment wieder geöffnet, muss die ECU aus sein und INCA darf sich nicht verbinden lassen. (T101.3_XCP)
- Erst wird der CalWup ausgelöst, dann wird ein leeres Experiment (keine Mess- oder Verstellgrößen in der Ansicht) geöffnet. Die ECU darf nach 65s nicht ausgehen, die Verbindung muss bestehen bleiben. (T101.4_XCP)

Da in ETK-Systemen auch bei beschlossenen Experiment eine dauerhafte Kommunikation zwischen Schnittstellenmodul zum ETK und von dort zu ECU besteht, sind diese drei Testfälle für ETK-Systeme irrelevant.

6.3 Dokumentation

Um die Umsetzung der Testautomatisierung für ETK- und XCP-Systeme abzuschließen, wurde die Dokumentation angepasst. Es galt zu verhindern, dass bei Änderungen an Testfällen mehrere Dokumente gepflegt werden müssen. Das wurde erreicht, indem die Beschreibung der Testfälle aus der Testfallsammlung (angepasste IEEE 829, Kapitel 4) in die Dokumentation für einzelne Packages in ECU Test übernommen wurde. Diese Anpassung sorgt auch dafür, dass im Testreport genaue Informationen zu Zweck und Anforderung eines bestimmten Testfalles erwähnt werden.

Weiter wurde eine Anleitung für die Testautomatisierung erstellt und in einer Präsentation den zwei Experten, die sie in Zukunft anwenden sollen, erläutert.

Eine spezielle Dokumentation wurde für das Setup-Packages angelegt. In dieser wird beschrieben, wie Systeme als Zustandsmaschinen zu modellieren sind und wie dann, basierend darauf, eine Ansteuerung in ECU-Test umgesetzt werden kann. Damit ist die Möglichkeit gegeben in Zukunft möglichst einfach neue Systeme in die Automatisierung aufzunehmen wie z.B. XETK oder FETK-Systeme.

Da ECU-Test Anfang 2019 von Python 2 auf Python 3 umgestellt werden soll, wurden vorsorglich (nach bestem Kenntnisstand) die Stellen des Quellcodes, innerhalb von ECU-Test und in Erweiterungen, dokumentiert und markiert, die für Python 3 angepasst werden müssen.

7 Zusammenfassung und Ausblick

Dieses Kapitel stellt den Abschluss der Arbeit dar. Im V-Modell ist es somit als „System-Integration, Verifikation und Validation“ einzuordnen. Es bespricht das Ergebnis der Arbeit in Bezug auf die Situation vor der Arbeit (Kapitel 7.1) und gibt einen Ausblick auf Punkte, die in Zukunft weiter verfolgt/bearbeitet werden könnten (Kapitel 7.2).

7.1 Ergebnis

Entsprechend der Aufgabenstellung (Kapitel 1.3) wurden die ursprünglich drei Use-Cases erfolgreich untersucht. Hierbei wurde festgestellt, dass der Use-Case der Freigabe von ProF-Konfigurationen dem Use-Case der Freigabe von ETK-Systemen untergeordnet bzw. als Teil dessen betrachtet wird.

Es wurde der Ist-Stand des Freigabevorgangs für die zwei Use-Cases ermittelt (Kapitel 3). Dazu wurden nicht nur bereits vorhandene Lösungen gesichtet, die die Freigabe durch Automatisierung unterstützen, sondern auch die einzelnen Testobjekte und Testfälle, die sich aus diesen ergeben, genauer betrachtet. Die bereits bestehenden Freigabechecklisten für die Use-Cases ETK-System und XCP-System wurden als Grundlage für eine detailliertere Dokumentation von Testfällen genutzt.

Diese neue Dokumentation, die Testfallsammlung, wurde in ihrer Struktur und in ihrem Inhalt an die IEEE 829 angelehnt (Kapitel 4). Somit bietet sie eine durchgängige Informationsquelle, die für beide Use-Cases einheitlich aufgebaut ist. Neben den Testfallargumenten und -ergebnissen, wie sie in den Checklisten genannt wurden, sind in ihr zusätzlich Testfallquellen, formale Testfallanforderungen, erforderliche Vor- und Nachbedingungen der jeweiligen Testfälle u.v.m. festgehalten. Im Zuge der Vereinheitlichung der Dokumentation beider Use-Cases wurden auch die Testfälle auf Harmonisierbarkeit untersucht. So konnte bereits vor der Umsetzung gezeigt werden, dass viele Testfälle nicht in unterschiedlichen Varianten für die Use-Cases angedacht werden müssen, sondern die richtige Architektur der Testautomatisierung ein Wiederverwerten von bestimmten Testfällen erlauben müsste.

Auf Basis dieser Vorbereitung konnte ein Framework für die Umsetzung der Testautomatisierung gewählt werden: ECU-Test. Es wurde betrachtet, welche Funktionen das Framework bereits besitzt und welche erweitert werden müssen, um alle Testfälle aus der Testfallsammlung umsetzen zu können (Kapitel 5). Hierbei wurde auch die allgemeine Architektur aufgearbeitet, in der die Testautomatisierung realisiert werden soll (vgl. Abbildung 21).

In der Abbildung ist die Testfallsammlung als Testfallbibliothek dargestellt. Zusammen mit den bereits vorhandenen und erweiterten Funktionen von ECU-Test (Funktionen und Tooljobs) wurde aus der Testfallbibliothek eine Testskriptbibliothek (Kapitel 6). Diese Bibliothek beinhaltet automatisierte Testskripte für jeden Testfall der Sammlung. Die Testskriptbibliothek wurde hierbei so implementiert, dass sie entsprechend der angedachten Harmonisierung zwischen beiden Use-Cases möglichst allgemeingültig funktioniert. Bis auf einzelne spezifische Fälle ist das für den Großteil der Testfälle gelungen.

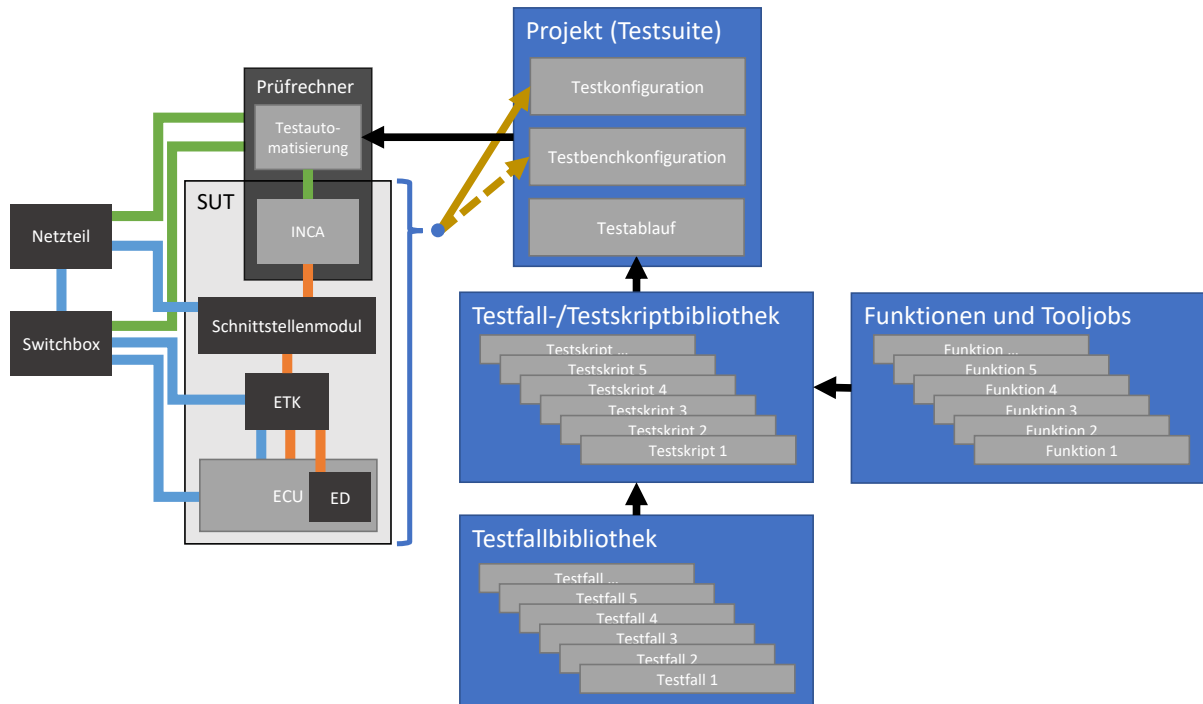


Abbildung 21: Architektur

Der Tester kann dann ein Projekt zusammensetzen, das einen Testablauf aus einer beliebigen Untermenge der Testskripte realisiert. Das Projekt stellt hierbei zudem die Abstraktionsebene dar, in der der Tester Informationen zum verwendeten Testaufbau hinterlegen kann. Er muss hier das verwendete Netzteil festlegen, so wie Informationen zum SUT: Handelt es sich um ein ETK- oder ein XCP-System? Welcher Programmstand soll verwendet werden? Wo liegt die INCA-Datenbank auf dem Prüfrechner? etc.

Weitere beinhaltet das Projekt die Wahl eines oder mehrerer Ausgabeformaten für die zu generierenden Testreports. Es können eigene Formate angelegt werden oder Vorlagen genutzt werden, die im Zuge dieser Arbeit angelegt wurden.

Die Arbeit wurde für die weitere Benutzung dokumentiert. Diverse Dokumente beschreiben:

- die Bedienung der Testautomatisierung in einer Anleitung,
- welche Funktionen in ECU-Test erweitert wurden,
- die Möglichkeiten, die genutzt wurden, um ECU-Test zu erweitern,
- Bugs und Fehler die während der Arbeit im Bezug auf diverse Tools aufgefallen sind,
- Anpassungen die an der Switchbox vorgenommen wurden,
- das Vorgehen, um weitere System-Typen anzulegen, die die selben Testskripte nutzen sollen und
- Ideen, die während der Arbeit aufgekommen sind, um die Testautomatisierung in zukünftigen Projekten weiter auszubauen.

Die Testautomatisierung wurde also erfolgreich umgesetzt und so dokumentiert hinterlegt, dass sie genutzt, gepflegt und ggf. erweitert werden kann. Sie ersetzt und erweitert die bereits bestehenden Lösungen zur Testautomatisierung in einer vereinheitlichten Form für beide Use-Cases.

7.2 Ausblick

Während der Arbeit an der Testautomatisierung sind diverse Punkte für zukünftige Erweiterungen und Verbesserungen aufgefallen, die hier zusammengefasst sind. Dieser Abschnitt soll aber auch Vorgehen der Arbeit bewerten und inwiefern sie sich für zukünftige Weiterentwicklungen eignen.

ECU-Test ECU-Test als Framework für die Testautomatisierung hat einige Teile der Umsetzung vereinfacht und unterstützt. Die Verwendung von ECU-Test außerhalb seines eigentlichen Anwendungsbereiches (HIT statt SIL/HIL) hat sich jedoch immer wieder gezeigt. In Zukunft können im Dialog Erweiterungen und Fehlerbehebungen für ECU-Test angestrebt werden, die diese Arbeit aufgedeckt hat. Damit wäre es auch möglich die Komplexität diverser Testfälle zu verringern und sie so robuster zu gestalten.

Harmonisierung Use-Cases Die zuvor getrennt betrachteten Use-Cases konnten im Zuge dieser Arbeit gut zusammengeführt werden. Im Testablauf für ETK-Systeme sind lediglich 5 aus 23 Testfällen ETK spezifisch, im Testablauf für XCP-Systeme 6 aus 23. Die Testfallbibliothek beinhaltet 17 allgemeingültige Testfälle aus insgesamt 28.

Diese Harmonisierung wurde hauptsächlich durch die Trennung beim Erreichen der Vorzustände erreicht. Wenn die Testautomatisierung in Zukunft für andere Use-Cases/-Systeme angepasst werden soll, verspricht dieses Vorgehen, dass auch dort die allgemeingültigen Testfälle ohne umfangreiche Anpassungen genutzt werden können.

Switchbox und „Multi-Test-Matrix“ Einige Testfälle benötigen die Interaktion des Testers mit dem Testaufbau, wenn z.B. Verhalten bei Leitungsunterbrechung getestet werden soll. In Zukunft könnte die verwendete Switchbox um zusätzliche Relais erweitert werden, um die Leitungen automatisch trennen und wieder verbinden zu können. Es kamen auch Überlegungen auf, die Switchbox zu einer Art „Multi-Test-Matrix“ zu erweitern. Eine solche Matrix könnte es ermöglichen mehrere ECUs, Schnittstellen-Module und ETKs automatisch gleichzeitig bzw. nacheinander zu testen. Um das zu erreichen müsste diese Matrix zwischen unterschiedlichen CAN-Leitungen, Ethernetanschlüssen und einigen Bananensteckeranschlüssen ferngesteuert umschalten können.

Relevanz ISO 26262 ECU-Test ermöglicht die Qualifizierung der Testautomatisierung für festgelegte und dokumentierte Anwendungsfälle. Hierbei wird sich auf die Toolklassifizierung der Norm bezogen. Diese fordert eine Bewertung des *Tool impact* (TI), der Tool Error Detection (TD) und dem daraus folgenden *Tool Confidence Level* (TCL) (vgl. Abbildung 22).

Eine entsprechende Betrachtung der Testautomatisierung würde vermutlich zu einer Einordnung TI1 und damit TCL1 führen, da sie lediglich an Laborplätzen genutzt wird (Use-Cases Test von ETK- und XCP-Systemen). Ein Fehler in der Automatisierung, der bspw. eine falsche Größe über INCA in der ECU beschreibt und damit den Beschleunigungswunsch des Fahrers unerwartet auf 100% setzt, hätte keinerlei schädigende reale

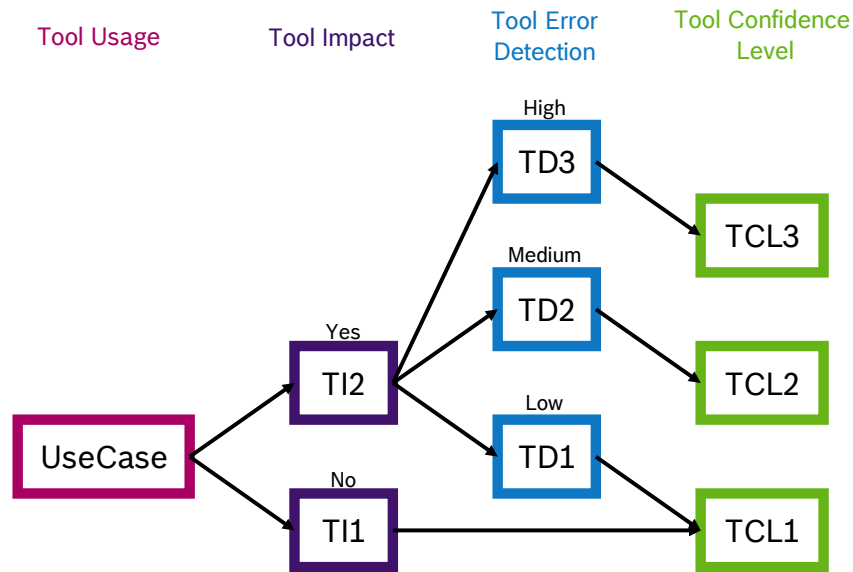


Abbildung 22: Toolklassifizierung

Folgen. Es würden keine Personen verletzt oder geschädigt. Somit ist auch keine Qualifizierung der Testautomatisierung nötig, da diese nur für Tools mit TCL2 und TCL3 gefordert ist. Aus momentaner Sicht ist selbst eine Klassifizierung der Testautomatisierung nicht nötig, da sie lediglich unterstützend in einem Freigabeprozess genutzt wird, der selbst nicht Teil eines, von der ISO 26262, abgedeckten Entwicklungsschrittes ist.

Rückmeldung aus Anwendung Im Zuge dieser Arbeit wurde die Testautomatisierung an drei freigegebenen Toolkettenvarianten erprobt. Weitere mögliche Verbesserungen und Anpassungen werden sich erst zeigen, wenn sie beim Test weiterer Toolkettenvarianten genutzt wurde, die noch nicht freigegeben sind.

Die Dokumentation, die umgesetzten Abstraktionsebenen und Einstellungen (Wrapper für eigene Funktionen, Setup-Package, Einstellungen innerhalb von Packages) müssen zeigen, inwiefern sie eine Anpassung bei zukünftigen Änderungen an der INCA-API, ECU-Test und der Testsoftware im Softwarestand für Tests, die nicht auszuschließen bzw. geplant sind, möglichst einfach gestalten.

Literatur

- [ASA17] ASAM e. V. ASAM MCD-1 XCP. <https://www.asam.net/standards/detail/mcd-1-xcp/>, 2017. Abgerufen am 24. April, 2018.
- [Bei95] Boris Beizer. *Black box testing : techniques for functional testing of software and systems*. Wiley, New York [u.a.], 1995.
- [BHM⁺15] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, und Shin Yoo. The oracle problem in software testing: A survey. *IEEE Transactions on Software Engineering*, 41(5):507–525, 2015.
- [BK 18] BK Precision. BK Precision 9205 Multi-Range Programmable DC-Power Supply. <https://www.bkprecision.com/products/power-supplies/9205-600w-multi-range-60v-25a-dc-power-supply.html>, 2018. Version 3.1, Abgerufen am 18. Juni, 2018.
- [Boe81] Barry Boehm. *Software engineering economics*. Eglewood Cliffs, 1981.
- [Boe84] Barry Boehm. *Verifying and Validating Software Requirements Specifications*. IEEE Software, 1984.
- [BS08] Salim Baharom und Zarina Shukur. Module documentation based testing using Grey-Box approach. <https://ieeexplore.ieee.org/document/4631651/>, 2008. Abgerufen am 17. Mai, 2018.
- [Doc13] Docupedia Calibration System. Calibration System Messperformance. Intranet Robert Bosch GmbH, 2013. Abgerufen am 23. April, 2018.
- [ETA18a] ETAS GmbH. Calibrating Automotive Electronics. Homepage ETAS GmbH: https://www.etas.com/en/products/solutions_calibrating_automotive_electronics.php, 2018. Abgerufen am 24. April, 2018.
- [ETA18b] ETAS GmbH. ES523. https://www.etas.com/de/products/es523_can_fd_interface_module.php, 2018. Abgerufen am 18. Juni, 2018.
- [ETA18c] ETAS GmbH. ES595. <https://www.etas.com/de/products/es595.php>, 2018. Abgerufen am 18. Juni, 2018.
- [ETA18d] ETAS GmbH. ETK-S21.1. <https://www.etas.com/de/products/etks21.php>, 2018. Abgerufen am 18. Juni, 2018.
- [ETA18e] ETAS GmbH. INCA. https://www.etas.com/de/products/inca_software_products.php, 2018. Abgerufen am 18. Juni, 2018.
- [ETA18f] ETAS GmbH. Steuergeräte- und Bus-Schnittstellenmodul. https://www.etas.com/de/products/es891_es892_fetk_ecu_bus_interface_modules.php, 2018. Abgerufen am 25. April, 2018.
- [ETA18g] ETAS GmbH. Universelle Steuergeräte-Schnittstellen. https://www.etas.com/de/products/etk_fetk_xetk_ecu_interfaces.php, 2018. Abgerufen am 25. April, 2018.

- [Gra18] Claus Graf. CalToolDevelopment: Methods and Tools. Intranet Robert Bosch GmbH, 2018. Abgerufen am 05. Juni, 2018.
- [H⁺18] Mark Hammond et al. pywin32 223. <https://pypi.org/project/pywin32/>, 2018. Abgerufen am 10. Juni, 2018.
- [How78] W.E. Howden. Theoretical and empirical studies of program testing. *IEEE Transactions on Software Engineering*, SE-4(4):293–298, 1978.
- [IEE08] IEEE. IEEE 829-2008 - Standard for Software and System Test Documentation. <https://ieeexplore.ieee.org/servlet/opac?punumber=4578271>, 2008. Abgerufen am 03. Mai, 2018.
- [ISO11] ISO. Iso 26262-4, 2011.
- [IST16] ISTQB. Standard Glossary of Terms used in Software Testing. <https://www.istqb.org/component/jdownloads/send/20-istqb-glossary/186-glossary-all-terms.html>, 2016. Version 3.1, Abgerufen am 17. Mai, 2018.
- [Lie18] Chris Liechti. pyserial3.4. <https://pypi.org/project/pyserial/>, 2018. Abgerufen am 10. Juni, 2018.
- [Nö12] Ralf Nörenberg. *Effizienter Regressionstest von E/E-Systemen nach ISO 26262*. Steinbuch series on advances in information technology ; 3. Dissertation, KIT Scientific Publishing, Karlsruhe, 2012.
- [Pyt18] Python Software Foundation. Python. <https://www.python.org/>, 2018. Abgerufen am 03. Mai, 2018.
- [PZ16] Andreas Patzer und Rainer Zaiser. *XCP - The Standard Protocol for ECU Development*. Vector Informatik GmbH, V3.0, 2016.
- [RD14] Konrad Reif und Karl-Heinz Dietsche. *Kraftfahrtechnisches Taschenbuch*. Robert Bosch GmbH, Karlsruhe, 28. überarb. und erweiterte Aufl., 2014.
- [SL12] Andreas Spiller und Tilo Linz. *Basiswissen Softwaretest - Aus- und Weiterbildung zum Certified Tester (Foundation Level nach ISTQB-Standard)*. Dpunkt-Verl., Bremen, 5. überarb. und aktualisierte Aufl., 2012.
- [SW02] Harry M. Sneed und Mario Winter. *Testen objektorientierter Software: das Praxishandbuch für den Test objektorientierter Client-Server-Systeme*. Hanser, 2002.
- [Tra18] TraceTronic GmbH. ECU-TEST. <https://www.tracetronic.de/produkte/ecu-test/>, 2018. Abgerufen am 02. Mai, 2018.
- [Vec18] Vector Informatik GmbH. Steuergeräte-Kalibrierung. https://vector.com/vi_ecu_measurement_de.html, 2018. Abgerufen am 25. April, 2018.
- [Wei13] Michael Weigend. *Python GE-PACKT : [schneller Zugriff auf Module, Klassen und Funktionen ; tkinter, Datenbanken, OOP und Internetprogrammierung; für die Versionen Python 3.3 und 2.7]*. Die GE-PACKTE Referenz. mitp, Heidelberg, 5. überarb. Aufl., 2013.

Abbildungsverzeichnis

1	ECU-Funktionen und ihre Größen	4
2	Schematische Darstellung der Toolkette	4
3	Test und Freigabe der Toolkette	9
4	White-, Grey-, Blackbox-Test	10
5	V-Modell	12
6	Überblick ISO 26262	13
7	Speicherseitenverwaltung in INCA	14
8	ECU-Test	15
9	Testaufbau	17
10	Switchbox	18
11	Use-Case Diagramm	20
12	ETK-Tester	21
13	Ausschnitt aus der Freigabecheckliste für ETK-Systeme	24
14	Struktur Testfallerfassung	33
15	Beispiel für einen Tooladapter in ECU-Test	36
16	Prinzipieller Aufbau einer A2L	40
17	Vergleich der Darstellung eines Testskriptes: Package und externer Report	44
18	Zustandsmaschine für ETK-Systeme	47
19	Konfiguration über globale Konstanten	49
20	Struktogramm für das Testskript <i>Arrays</i> als Teil von T100.1(CalibTest)	52
21	Architektur	59
22	Toolklassifizierung	61

Tabellenverzeichnis

1	Testfalldokumentation; angepasste IEEE 829	27
2	Beispiel Testfall 1 in der Testfalldokumentation	29
3	Beispiel Testfall 2 in der Testfalldokumentation	29
4	Methods for deriving test cases for integration testing	31
5	Beispiel Testfall 3 in der Testfalldokumentation	34
6	Bewertungsergebnisse und ihre Bedeutung in der Testautomatisierung	43

Abkürzungsverzeichnis

API	Application Programming Interface
ASAP	Working Party on Standardization of Applications
ASIL	Automotive Safety Integrity Level
ASQF	Arbeitskreis Software-Qualität und -Fortbildung
CalWup	CalibrationWakeup
CAN FD	Controller Area Network Flexible Data-Rate
CAN	Controller Area Network
CCP	CAN Calibration Protocol
CTO	Command Transfer Object
DAP	Debug Access Port
DTO	Data Transfer Object
DWARF	Debugging Data Format
E/E	Elektrisch und Elektronisch
ECU	Engine Control Unit
ED	Emulation Device
ELF	Executable and Linkable Format
ETAS	ETAS GmbH, Stuttgart, Deutschland
ETK	Emulations Tastkopf
HIL	Hardware In the Loop
HIT	Hardware Integration Test
HR	Hauptrelais
HW	Hardware
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
INCA	Applikationstoolsoftware der ETAS GmbH
ISO	International Organisation for Standardization
ISTQB	International Software Testing Qualification Board
JTAG	Joint Test Action Group

PD	Production Device
ProF	Programm Flash Toolkomponente
RAM	Random Access Memory
SIL	Software In the Loop
SW	Software
T15	Anschlussklemme der ECU, welche der Zündung (an/aus) entspricht
T30	Anschlussklemme der ECU, welche der Dauerversorgung (an/aus) entspricht
UI	User Interface
USB	Universal Serial Bus
VBA	Visual Basic for Applications
WPF	Windows Presentation Foundation
XCP	Universal Measurement and Calibration Protocol
XML	Extensible Markup Language

Anhang



Abbildung A.1: Beispiel interner Testreport 1

Echtheitsnachweis des Testreports erfolgreich durchgeführt. Testreport wurde nicht manipuliert.

Projekt
 Konfigurationswech
 PatchA2J_and_Resta
 Setup_ETK
 INCAOptions_ETK
 T01.1(PVER_D0)
 T05.1(HW_Jnt)
 T100.2_ETK(ETHCon
 T100.3_ETK(ETHCon
 T101.4(CalibPageChe
 T100.1(Calib7 test)
T104.3(RP0WP)
 T101.1(CalibWupStart
 T104.7(RP0WP_T15
 T102.1(FirstCycleMe
 T107.1(Synchroperc
 T105.4_ETK(ETKrese
 T105.3(postDiveInk
 T107.3(postDiveMe
 T104.1(WorbaseD0
 T104.2(WorbaseUp
 T104.5(WorbaseD0
 T104.6(WorbaseUp
 T105.1(RS75)
 T106.1(0aa4Rerentio
 T103.1_ETK(Fishing
 T103.2_ETK(Fishing

Ausführungsmodus: Automatisch
Ausführungsdauer: 00:00:0.97
Bemerkung: Lauf #1
Beschreibung:
 Testfalltyp: Automatisch
 Testfallid: Keine Referenzseite nach Abtestsuite prüfen
 Testfallquelle: alle Freigabecheckliste
 Testfallanforderung: Speicherseite muss von der Referenzseite in die Abtestsuite kopierbar sein
 Testvorgang: Freigabe Toolkette
 Testobjekt: Speicherseitenverwaltung
 Testumgebung: externe HW verbunden, einmalig initialisiert, Calib
 Testfallzustände: T05.1(HW_Jnt)
 Vorgangstestfälle: -
 Nachfolgetestfälle: -
 Testumgebung: Testkonfig, Testbenchkonfig, Toolkette
 Testfallargumente: eine Vstellgröße vorstellen, Speicherseitenverwaltung Referenzseite nach Abtestsuite kopieren

Dateiname: D:\ECU-Test_w\Repo_\Testautomatisierung\Packages\Testautomatisierung\T104-3(RP0WP).pkg
Testskriptid: T104.3
Profrechner: SE-Z0R8Q
Version: 1.0

Benutzerdefinierte Informationen

Customer	Bosch
ECU-No./Type	[REDACTED]
ETK	ETK-S21_1_A
ETK-S SerialNo.	SN0140466
ETK-Tools	V4.1.9
HSP	V11.8.0
Location	[REDACTED]
ext. Tool-Hardware	ESS95.1
Tester	TWGSZI
Computername	SI-Z0R8Q / HP ZBook 15 G3
OperatingSystem	Windows 7 Enterprise
PVER-Path	[REDACTED]
Inca Version	Inca V7.2.8 Build 1331
Inca-API Version	Inca-API V7.2
Directory of *.azl	[REDACTED]
PVER - Name/Variant/Revision	[REDACTED]
BC-CompPtc - Variant/Revision	300.6.0.0
FC-ARB/XCP - Variant/Revision	1.22.0.0
AGI/AGI - Variant/Revision	5.8.0.4
BX-Test - Variant/Revision	1.0.0.0

Abbildung A.2: Beispiel interner Testreport 2

Suchen

Echtheit/nachweis des Testreports erfolgreich durchgeführt. Testreport wurde nicht manipuliert.


#	Aktion/Name	Wert	Bemerkung
1	Precondition		
2	Modif_ToolchainState		
3	Job-Ausführung: SwitchToWorkingPage		
7	Modif_RecalculateChecksums		
8	Check_ChecksumsRPhysWP		
9	Berechnung	rpwsp -> True	
10	testvariable WP read - original value		
11	Job-Ausführung: ReadCalibration		
17	testvariable WP write - new value = original value / 2	False	
18	if (2 < old_value < 2)		
21	Else	old_value / 2 -> <64.0 -> new_value	
22	Berechnung		
23	Job-Ausführung: WriteCalibration		
30	RP and WP should be different now		
31	Modif_RecalculateChecksums		
32	Berechnung		
33	Check_ChecksumsRPhysWP		
34	Berechnung		
35	copy RP to WP		
36	Job-Ausführung: StopMeasurement		
40	Job-Ausführung: Stop		
44	Job-Ausführung: CopyReferencePageToWorkingPage		
48	RP and WP should be equal again		
49	Modif_RecalculateChecksums		
50	Check_ChecksumsRPhysWP		
51	Berechnung	rpwsp -> True	
52	Postcondition		
53	Job-Ausführung: StopMeasurement		
57	Job-Ausführung: Stop		
61	Job-Ausführung: CopyReferencePageToWorkingPage		
65	Modif_RecalculateChecksums		

if something went wrong we need t

Bewertung	Zeit [s]	Aktion/Name	Wert	Bemerkung
	0.001	Precondition		

Parameter
Bewertung der enthaltenen Testschritte überschreiben (NONE/ERROR) True

Abbildung A.3: Beispiel interner Testreport 3



Testspezifikation:
T104.7(RPtoWP, T15 off)

ID:	2Q8RR4
Name:	T104.7(RPtoWP, T15 off)
Designer:	TWG25I
Status:	Ready
Version:	1.0
Beschreibung:	<p>Testfalltyp: automatisiert</p> <p>Testfallzweck: Kopieren Referenzseite nach Arbeitsseite bei T15 aus prüfen</p> <p>Testfallquelle: alte Freigabecheckliste</p> <p>Testfallanforderung: Speicherseite muss von der Referenzseite in die Arbeitsseite im CalWup kopierbar sein</p> <p>Testvorgang: Freigabe Toolkette</p> <p>Testobjekt: Speicherseitenverwaltung</p> <p>Testfallvorzustände: externe HW verbunden, einmalig initialisiert; CalWup</p> <p>Testfallnachzustände: -</p> <p>Vorgangertestfälle: T05.1(HW_Init)</p> <p>Nachfolgetestfälle: -</p> <p>Testumgebung: Testkonfig, Testbenchkonfig, Toolkette</p> <p>Testfallargumente: eine Verstellgröße vorstellen, Speicherseitenverwaltung Referenzseite nach Arbeitsseite kopieren</p> <p>Testfallergebnisse: Prüfsummen gleich: JA/NEIN</p>
Ablauf	<pre>testvariable WP read - original value testvariable WP write - new value = original value / 2 RP and WP should be different now copy RP to WP RP and WP should be equal again</pre>

TestSpec powered by [iQuery 1.9.0](#)

generiert am 26.07.2018 14:33 mit ECU-TEST: 7.0.1.72040

Abbildung A.4: Beispiel Testspezifikation

Komponente	Eigenschaften
Projekt	
Konfigurationswechsel	STD_Testconfig_ETK.tcf; STD_Testbench_ETK.tbc
PatchA2I_and_Restart_Config	
T01.2_ETK(A2LPatch_Validate)	T01.2_ETK(A2LPatch_Validate).pkg
Konfigurationswechsel	STD_Testconfig_ETK.tcf; STD_Testbench_ETK.tbc
Setup_ETK	Setup_ETK.pkg
INCAOptions_ETK	INCAOptions_ETK.pkg
T01.1(PVER_DB)	T01.1(PVER_DB).pkg
T05.1(HW_Init)	T05.1(HW_Init).pkg
T100.2_ETK(ETHConnectionLostNoMeasur	T100.2_ETK(ETHConnectionLostNoMeasurement).pkg
T100.3_ETK(ETHConnectionLostMeasurement	T100.3_ETK(ETHConnectionLostMeasurement).pkg
T01.4(CodePageCheck)	T01.4(CodePageCheck).pkg
T100.1(CalibTest)	T100.1(CalibTest).pkg
T104.3(RPtoWP)	T104.3(RPtoWP).pkg
T101.1(CalWupStart)	T101.1(CalWupStart).pkg
T104.7(RPtoWP, T15 off)	T104.7(RPtoWP, T15 off).pkg
T102.1(FirstCycleMeasurement)	T102.1(FirstCycleMeasurement).pkg
T107.1(Synchroperiode)	T107.1(Synchroperiode).pkg
T105.4_ETK(ETKReset)	T105.4_ETK(ETKReset).pkg
T105.3(PostDriveInterrupt)	T105.3(PostDriveInterrupt).pkg
T107.3(PostDriveMeasurement)	T107.3(PostDriveMeasurement).pkg
T104.1(WorkbaseDownload)	T104.1(WorkbaseDownload).pkg
T104.2(WorkbaseUpload)	T104.2(WorkbaseUpload).pkg
T104.5(WorkbaseDownload, T15 off)	T104.5(WorkbaseDownload, T15 off).pkg
T104.6(WorkbaseUpload, T15 off)	T104.6(WorkbaseUpload, T15 off).pkg
T105.1(RST5)	T105.1(RST5).pkg
T106.1(DataRetention)	T106.1(DataRetention).pkg
T103.1_ETK(Flashing)	T103.1_ETK(Flashing).pkg
T103.2_ETK(Flashing, T15 off)	T103.2_ETK(Flashing, T15 off).pkg

Abbildung A.5: Ablauf und Testfälle für Use-Case ETK

Komponente	Eigenschaften
Projekt	
Konfigurationswechsel	STD_Testconfig_XCP.tcf; STD_Testbench_XCP.tbc
PatchA2I_and_Restart_Config	
T01.2_XCP(A2LPatch_Validate)	T01.2_XCP(A2LPatch_Validate).pkg
Konfigurationswechsel	STD_Testconfig_XCP.tcf; STD_Testbench_XCP.tbc
Setup_XCP	Setup_XCP.pkg
INCAOptions_XCP	INCAOptions_XCP.pkg
T01.1(PVER_DB)	T01.1(PVER_DB).pkg
T05.1(HW_Init)	T05.1(HW_Init).pkg
T100.2_XCP(CANConnectionLostNoMeasuren	T100.2_XCP(CANConnectionLostNoMeasurement).pkg
T100.3_XCP(CANConnectionLostMeasuremen	T100.3_XCP(CANConnectionLostMeasurement).pkg
T01.4(CodePageCheck)	T01.4(CodePageCheck).pkg
T100.1(CalibTest)	T100.1(CalibTest).pkg
T104.3(RPtoWP)	T104.3(RPtoWP).pkg
T101.1(CalWupStart)	T101.1(CalWupStart).pkg
T101.2_XCP(60sComCalWup_before)	T101.2_XCP(60sComCalWup_before).pkg
T101.3_XCP(60sComCalWup_after)	T101.3_XCP(60sComCalWup_after).pkg
T101.4_XCP(60sComCalWup_inExp)	T101.4_XCP(60sComCalWup_inExp).pkg
T101.5_XCP(CalWupMeasurement)	T101.5_XCP(CalWupMeasurement).pkg
T104.7(RPtoWP, T15 off)	T104.7(RPtoWP, T15 off).pkg
T102.1(FirstCycleMeasurement)	T102.1(FirstCycleMeasurement).pkg
T107.1(Synchroperiode)	T107.1(Synchroperiode).pkg
T105.3(PostDriveInterrupt)	T105.3(PostDriveInterrupt).pkg
T107.3(PostDriveMeasurement)	T107.3(PostDriveMeasurement).pkg
T104.1(WorkbaseDownload)	T104.1(WorkbaseDownload).pkg
T104.2(WorkbaseUpload)	T104.2(WorkbaseUpload).pkg
T104.5(WorkbaseDownload, T15 off)	T104.5(WorkbaseDownload, T15 off).pkg
T104.6(WorkbaseUpload, T15 off)	T104.6(WorkbaseUpload, T15 off).pkg
T105.1(RST5)	T105.1(RST5).pkg
T106.1(DataRetention)	T106.1(DataRetention).pkg

Abbildung A.6: Ablauf und Testfälle für Use-Case XCP

Testfallsammlung	
1	
2	Schlüsselwörter Testfalltyp: automatisiert, manuell, mit Nutzereingabe
3	Schlüsselwörter Testfallstatus: ermittelt, spezifiziert, getestet
4	Kennzeichnung T01.1
5	Testfalltyp automatisiert
6	Testfallzweck es muss überprüft werden ob der Programmstand lauffähig und angepasst für den Testablauf ist
7	Testfallquelle andere Testfälle
8	Testfallanforderung in Workspce (*.db3) prüfen ob Test-BC/FC registriert sind, ob es "modified" Einträge gibt und in *.xml prüfen ob Scheduling-Einträge vorhanden sind
9	Testvorgang Freigabe Toolkette
10	Testobjekt Programmstand
11	Testfallvorzustände alle Inputdateien wurden angegeben
12	Testfallnachzustände -
13	Vorgängertestfälle
14	Nachfolgetestfälle
15	Testumgebung Testumgebung mit eingelesenem angepassten Programmstand
16	Testfallargumente Name des Test-BC/FC und Namen der benötigten Messraster
17	Testfallergebnisse Test-BC und FC(s) in Build vorhanden und in Scheduling gelistet: JA/NEIN
18	Testfallstatus getestet
19	
20	Kennzeichnung T01.2
21	Testfalltyp automatisiert
22	Testfallzweck in der *.a2l müssen für die Freigabe diverse Anpassungen vorhanden werden
23	Testfallquelle andere Testfälle
24	Testfallanforderung die relevanten Anpassungen in der *.a2l müssen vorhanden sein
25	Testvorgang Freigabe Toolkette
26	Testobjekt Programmstand
27	Testfallvorzustände alle Inputdateien wurden angegeben
28	Testfallnachzustände -
29	Vorgängertestfälle T01
30	Nachfolgetestfälle
31	Testumgebung Testumgebung mit eingelesenem angepassten Programmstand
32	Testfallargumente Inhalt *.a2l
33	Testfallergebnisse Anpassungen vorhanden: JA/NEIN
34	Testfallstatus getestet
35	
36	Kennzeichnung T01.3
37	Testfalltyp automatisiert
38	Testfallzweck in der *.a2l müssen gewisse Anpassungen vorgenommen werden
39	Testfallquelle Anforderungen für Laufzeitmessung
40	Testfallanforderung die relevanten Anpassungen in der *.a2l müssen gepatcht werden
41	Testvorgang Freigabe Toolkette
42	Testobjekt Programmstand
43	Testfallvorzustände alle Inputdateien wurden angegeben
44	Testfallnachzustände *.a2l ist für die Lauzeitmessung gepatcht
45	Vorgängertestfälle
46	Nachfolgetestfälle
47	Testumgebung Testumgebung mit eingelesenem angepassten Programmstand
48	Testfallargumente Inhalt *.a2l
49	Testfallergebnisse Anpassung erfolgreich: JA/NEIN
50	Testfallstatus getestet
51	
52	Kennzeichnung T02.1
67	
68	Kennzeichnung T03.1_ETK
83	
84	Kennzeichnung T03.1_XCP
99	
100	Kennzeichnung T04.1_ETK
115	
116	Kennzeichnung T05.1_ETK
131	
132	Kennzeichnung T06.1_XCP
147	
148	Kennzeichnung T01.4
163	
164	Kennzeichnung T100.1
179	
180	Kennzeichnung T100.2_ETK
195	
196	Kennzeichnung T100.2_XCP
211	
212	Kennzeichnung T100.3_ETK
227	
228	Kennzeichnung T100.3_XCP
243	
244	Kennzeichnung T101.1_ETK

Abbildung A.8: Testfallsammlung

+	260	Kennzeichnung	T101.1_XCP
	275		
	276	Kennzeichnung	T101.2_XCP
+	291		
	292	Kennzeichnung	T101.3_XCP
+	307		
	308	Kennzeichnung	T101.4_XCP
+	323		
	324	Kennzeichnung	T101.5_XCP
+	339		
	340	Kennzeichnung	T102.1
+	355		
	356	Kennzeichnung	T103.1_ETK
+	371		
	372	Kennzeichnung	T103.2_ETK
+	387		
	388	Kennzeichnung	T104.1
+	403		
	404	Kennzeichnung	T104.2
+	419		
	420	Kennzeichnung	T104.3
+	435		
	436	Kennzeichnung	T104.4
+	451		
	452	Kennzeichnung	T104.5
+	467		
	468	Kennzeichnung	T104.6
+	483		
	484	Kennzeichnung	T104.7
+	499		
	500	Kennzeichnung	T104.8
+	515		
	516	Kennzeichnung	T105.1
+	531		
	532	Kennzeichnung	T105.2
+	547		
	548	Kennzeichnung	obs T105.3
+	563		
	564	Kennzeichnung	T105.4_ETK
+	579		
	580	Kennzeichnung	T106.1
+	595		
	596	Kennzeichnung	T107.1
+	611		
	612	Kennzeichnung	T107.2_ETK
+	627		
	628	Kennzeichnung	T107.2_XCP
+	643		
	644	Kennzeichnung	T107.3
+	659		
	660	Kennzeichnung	T107.4
+	675		
	676	Kennzeichnung	T107.5
+	691		
	692	Kennzeichnung	T107.6
+	707		

Abbildung A.9: Testfallsammlung