Die approbierte Originalversion dieser Diplom-/Masterarbeit ist in der Hauptbibliothek der Technischen Universität Wien aufgestellt und zugänglich.
http://www.ub.tuwien.ac.at

The approved original version of this diploma or master thesis is available at the main library of the Vienna University of Technology.
http://www.ub.tuwien.ac.at/eng

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Automated Synthesis of Behavior-Driven Integration Tests for Embedded Systems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Ewald Niederstätter, BSc
Matrikelnummer 0926036

an der Fakultät für Informatik
der Technischen Universität Wien
Betreuung: Univ.Doz. Dipl.-Ing. Dr.techn. Stefan Poledna

Wien, 10. Jänner 2017

Ewald Niederstätter	Stefan Poledna

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Automated Synthesis of Behavior-Driven Integration Tests for Embedded Systems

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Business Informatics

by

Ewald Niederstätter, BSc
Registration Number 0926036

to the Faculty of Informatics

at the Vienna University of Technology

Advisor: Univ.Doz. Dipl.-Ing. Dr.techn. Stefan Poledna

Vienna, 10th January, 2017

Ewald Niederstätter

Stefan Poledna

Erklärung zur Verfassung der Arbeit

Ewald Niederstätter, BSc
Florianigasse 30-32/8, 1080 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 10. Jänner 2017

Ewald Niederstätter

Acknowledgements

First, I would like to sincerely thank my parents Martha and Benedikt Niederstätter for their ongoing support throughout my education. Without their endless trust in me, I would have never had the opportunity to write these lines. Furthermore, I thank my girlfriend Olivia Padalewski for her tireless motivation, encouragement in critical phases and for the reviews of the present work.

Special thanks go to the cooperation partner for the opportunity to immerse into the subject topic and for all resources that have been made available for the implementation of this work. In particular, I would like to thank Alexander Schramek and Christian Hanni for the countless meetings, the feedback and support, which are among other things responsible for the success of my work.

Finally, I would like to thank Univ.Doz. Dipl.-Ing. Dr.techn. Stefan Poledna und Dr. Wilfried Steiner for the possibility of conducting this master thesis, their supervision, continuous support and constructive feedback.

Abstract

Nowadays embedded applications are integrated into simple consumer electronics to highly safety-critical control units. Whereas module tests may be implemented similarly compared to classical software development, the additional hardware component complicates the integration testing process dramatically. In addition to expensive and closed test systems, there are hardly any alternatives and innovative approaches to realize these types of tests.
The underlying work deals with the realization of a combination of integration tests for embedded applications on electronic control units and the test approach of behavior driven development. Thereby, the possibilities of the synthesis and the applicability should be demonstrated. In the course of this, the work comprises the development of an API using the programming language *Python* to provide functions for testing applications on electronic units. The API is tested in combination with an existing test framework for creating behavior driven tests within a prototypical setup. In order to increase the significance of the evaluation results with regards to practical applicability, the realization is tested using an existing and running application project, which is provided by the cooperation partner.
Both the status-quo analysis, the feedback presentation and the conducted expert interviews on the basis of a semi-structured guide indicate the successful implementation of the concept and confirm the applicability in practice. On the one hand, the evaluation reveals factors which state a clear advantage compared to classical test approaches and on the other hand, identifies current limitations and potential challenges.

Keywords
Embedded applications, electronic control units, software development, software testing, integration tests, behavior driven development, behavior driven testing

Kurzfassung

Embedded-Anwendungen werden heutzutage von einfacher Konsumelektronik bis hin zu höchst sicherheitskritischen Steuerungseinheiten eingesetzt. Während Modultests relativ ähnlich zu klassischer Softwareentwicklung realisiert werden können, verkompliziert die zusätzliche Hardwarekomponente das Integrationstesten um ein Vielfaches. Neben sehr teuren und in sich geschlossenen Testsystemen gibt es kaum Alternativen und innovative Ansätze, um diese Art von Tests umzusetzen.
Die vorliegende Arbeit befasst sich mit der Realisierung einer Kombination von Integrationstests für Embedded-Anwendungen auf elektronischen Steuereinheiten und dem Testansatz der verhaltensgetriebenen Softwareentwicklung. Dabei sollen die Möglichkeiten der Synthese und die Anwendbarkeit in der Praxis aufgezeigt werden. Im Zuge dessen umfasst die Arbeit die Entwicklung einer API in der Programmiersprache *Python*, um Funktionen zum Testen von Applikationen auf Steuerungseinheiten bereitzustellen. Die API wird in Kombination mit einem bestehenden Testframework zur Erstellung von verhaltensgetriebenen Testfällen innerhalb eines prototypischen Aufbaus getestet. Um die Aussagekraft in Richtung Praxisanwendung zu erhöhen, wird die Umsetzung an einem bestehenden und in der Praxis laufenden Projekt getestet, welches durch die Kooperation mit einem Partnerunternehmen zur Verfügung steht.
Sowohl die Status-quo Analyse, als auch die Feedback Präsentation und die abschließend zur Evaluierung durchgeführten Experteninterviews auf Basis eines semi-strukturierten Leitfadens zeigen die erfolgreiche Umsetzung des Konzepts und bestätigen die prinzipielle Anwendbarkeit in der Praxis. Die Evaluierung zeigt einerseits Faktoren auf, welche einen klaren Vorteil zu klassischen Testansätzen darstellen, als auch derzeitige Limitierungen und potentielle Herausforderungen.

Schlagwörter
Embedded-Anwendungen, elektronische Steuerungseinheiten, Softwareentwicklung, Softwaretests, Integrationstests, verhaltensgetriebene Softwareentwicklung, verhaltensgetriebene Softwaretests

Contents

Abstract **ix**

Kurzfassung **xi**

Contents **xiii**

1 Introduction **1**
1.1 Aim of the work . . . 3
1.2 Structure of the work . . . 4

2 Terminology **5**
2.1 Embedded systems model . . . 5
2.2 Controller Area Network . . . 8
2.3 Inter-Integrated Circuit . . . 12
2.4 Fundamentals of Testing . . . 15

3 State of the art **21**
3.1 TEmb method . . . 21
3.2 BDD for Circuit Design and Verification . . . 23
3.3 Verification pattern approach . . . 24
3.4 Design and test architecture . . . 25
3.5 Extended Finite State Machines . . . 25
3.6 Vector Informatik GmbH . . . 26

4 Methodology **29**
4.1 Behavior Driven Development . . . 29
4.2 Cooperation partner . . . 31
4.3 Development process . . . 32

5 Implementation **37**
5.1 Architecture . . . 37
5.2 Prototype test setup . . . 45

6 Evaluation **51**

6.1 Status quo analysis & feedback presentation 51
6.2 Expert interviews 52

7 Summary and outlook **59**

List of Figures **61**

List of Tables **62**

List of Listings **63**

Acronyms **65**

Bibliography **67**

A Appendix **71**
A.1 Feature definitions 71
A.2 Step definitions 73
A.3 Test report 75
A.4 Interview guide (german) 76
A.5 Interview guide (english) 78
A.6 Transcription experts interview 1 80
A.7 Transcription expert interview 2 104

CHAPTER 1

Introduction

The application field of embedded software has rapidly increased over the past decades and is present nowadays in almost all areas of life. The industry has to deal with constant change since systems have become larger and more complex. Due to the application in high-reliability and safety-critical fields like the automotive industry, the testing process of such systems is facing severe challenges.

In 1999 the National Aeronautics and Space Administration (NASA) lost two space probes due to software bugs. The first probe missed their orbit by 170 km because the navigation system provided the data in the imperial system and the software developed by NASA assumed the metric system. The second probe crashed because the moving out of the landing legs was interpreted by sensors as contact with the ground and brake rockets were disabled in a height of 40 meters. Both happenings were the result of reduced test phases due to time and cost constraints [1]. In 2013 Toyota was found liable for an accident that killed one woman due to bugs in the electronic throttle system (ETCS) which caused unintended acceleration [2]. The previous examples show only a small piece of defect software with severe consequences.

A study from 2002 delegated by the National Institute of Standards and Technology pointed out that annual costs of inadequate testing and infrastructure amount to $59.5 billion solely in the US [3]. Many software products are tested manually by developers themselves, because they know best how their products should work. This approach is used mostly for cost reasons and can work for relatively small and isolated projects, but can also be very time and cost consuming since tests have to be done every time from scratch. Another significant downside of manual tests is the lack of verification that code changes don't falsify the functionality of the whole system. Additionally, the whole documentation of test design and execution entails a tremendous outlay in terms of synchronization and maintenance. Furthermore, the approach to test embedded software is different to other application fields, since not only the software but also the interaction with the hardware and other real-time requirements have to be considered. Software

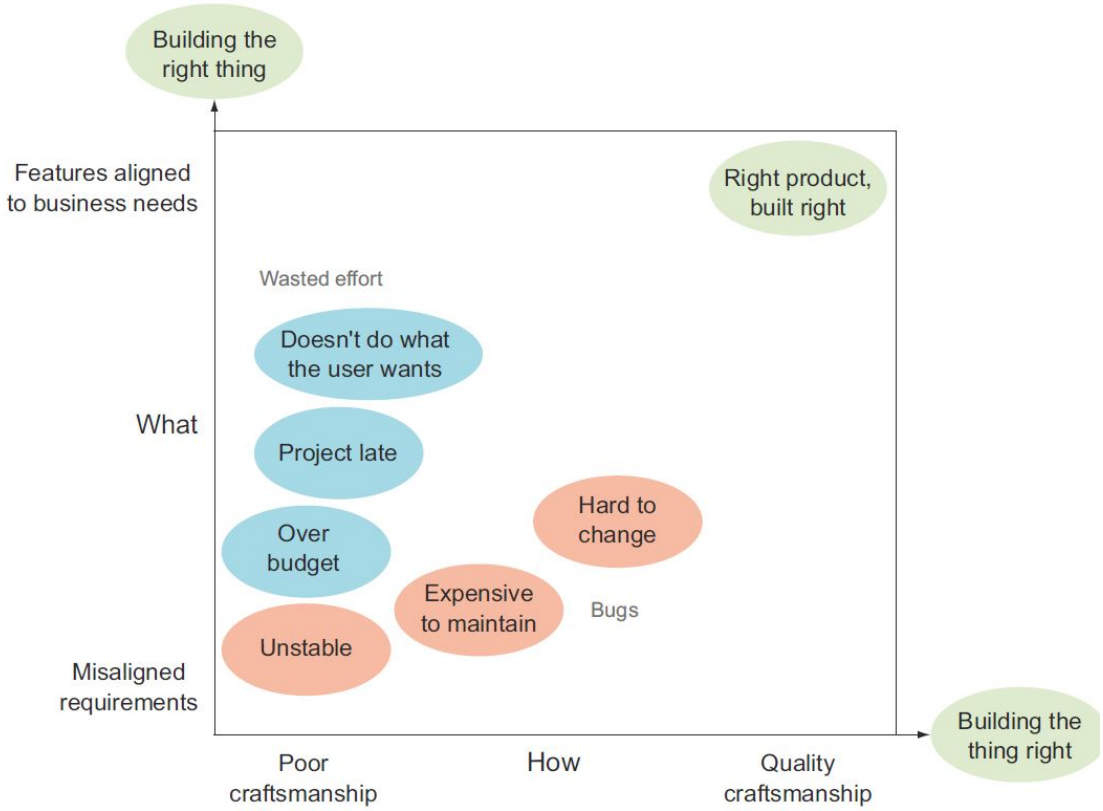

Figure 1.1: Possible issues during software development [5, p. 8]

testing is observed widely as dark art in software development. The exact need, purpose and benefit is not clear, only the necessity as every project does it. The amount and content of test cases or coverage metrics are defined independently from the project because other companies or teams are using them successfully. In many projects, testing is performed because it's a requirement of the customer but the real support during the life-cycle and afterwards for maintenance is completely neglected. This can result in an outcome where the design of the product doesn't fit the testing approach and therefore testing turns out to be very difficult. In theory about 50% of time and budget should be invested into testing, but this rule of thumb is not always taken into account. For example, time or budget restrictions can threaten the completion of projects and in this case it's not uncommon that the necessary savings are achieved by reduced testing activities [4].

The prevalent reasons of failing software projects may be allocated to the following two categories[5, p. 7]:

1. Not building the right software

2. Not building the software right

When not building the right software, the product is realized well, but nobody needs it. On the other hand, when not building the software right, the product comprises a lot of bugs and will be hard to maintain. Both use cases are visualized in figure 1.1. One example showing the stated facts is the development of a payroll system for the Queensland Health Department. The project was initialized in December 2007 with the delivery date of August 2008 and the initial budget of $6 million. The solution was eventually rolled out in 2010 - however, the majority of public servants were overpaid, underpaid or not paid at all due to erroneous code. Subsequently, over 1.000 payroll staff members were required to manually handle 200.000 processes each fortnight. A review of 2012 states, that the project costs increased to $416 million and would cost additional $837 million to apply all required fixes [6].

Software testing is very important not only to detect bugs and defects in a software project, but also to verify that expected requirements are satisfied. Depending on the amount and coverage of tests the risk of safety-critical defects and misconduct can be minimized to a certain degree. However, test concepts and automation approaches are only efficient if an accurate method will be used, otherwise the quality assurance can be very resource intensive. In many cases, traditional test approaches seem very time and cost consuming, as not only each affected test case and related test document have to be changed, but also reviewed and tested again, especially if requirements change frequently. Therefore, more agile alternatives were added to the selection of possible test approaches. However, not every approach can be equally applied in all different areas of software development.

1.1 Aim of the work

The aim of this thesis is to evaluate the use of Behavior Driven Test (BDT) activities within embedded software testing on Electronic Control Unit (ECU)s. As far as the author is aware, this special combination has not been investigated yet. The main outcome is a modular API to support the execution of behavior driven tests for embedded systems.

The proposed strategy of this work leads to the following research questions:

- *How can a behavior driven integration test approach be designed and integrated within embedded application development?*
- *Is the use of behavior driven testing applicable in embedded application development?*

The thesis presents a synthesis as combination between the concepts of behavior driven testing and embedded software testing in an automated way. The evaluation concentrates on system integration tests of embedded software applications on electronic control units.

1.2 Structure of the work

After a general introduction to the problem statement and general motivation for this work in chapter 1, the generic terminology and fundamentals are described in chapter 2. Chapter 3 deals with the current state-of-the-art in the field of embedded software testing and behavior-driven development. Afterwards, used methodologies for the underlying problem are described in chapter 4, together with the planned process to achieve the expected results of this work. Chapter 5 gives an overview of the developed artifact created in the course of this thesis and other technical details on used dependencies. Chapter 6 describes the main findings and results gained from the performed evaluation. Finally, the thesis is concluded within chapter 7, which reflects and summarizes the whole process and obtained results.

CHAPTER 2

Terminology

The following chapter provides fundamental information on relevant terminology. It focuses on the aspects of embedded systems, the *Controller Area Network*, the *Inter-Integrated Circuit* and some general state of the art testing fundamentals.

2.1 Embedded systems model

The expression *embedded system* describes a computer system embedded in some technical context (mechanical or electrical system), usually including hardware. It is either responsible for data and signal processing, or covers control and monitoring functions.

The following list comprises some general descriptions of embedded systems, which additionally describes some characteristics. However, those definitions hold only partially true, as todays embedded systems shifted from the early design principles [7, p. 3 ff]:

- *Embedded systems are more limited in hardware and/or software functionality than a PC.* In terms of hardware limitations (processing performance, power consumption, memory, hardware functionality..) or software limitations (fewer or scaled-down applications, limited OS..).

- *An embedded system is designed to perform a dedicated function.* The design of an embedded system targets primarily one specific function.

- *An embedded system is a computer system with higher quality and reliability requirements than other types of computer systems.* Some target fields of embedded systems focus on high quality and reliability, because very serious problems result from errors or faulty functions (car engine, medical equipment...).

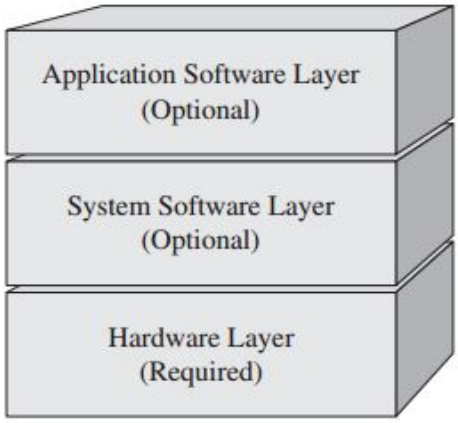

Figure 2.1: Embedded system model [7, p. 10]

Figure 2.1 shows the basic layers of an embedded system model. The *embedded board* contains all physical components and electrical paths to carry the signals between them. It is located in the *hardware layer* of an embedded system. The *system software layer* includes software called device driver. It takes care of the initialization of the underlying hardware and manages access from other layers. The third layer is called *application software layer* and defines the software which will be executed by the embedded system.[8, p. 5]

Figure 2.2 visualizes a generic but more detailed scheme of the hardware layer in an embedded system.

The typical components are [4]:

- **Power supply:** Power supply to run the embedded system, usually from a general source or own supply such as batteries
- **NVM:** Non-volatile memory to store the embedded software (e.g. ROM, flash cards, hard disk)
- **Processing unit:** Target processor for which the embedded software will be compiled
- **RAM:** Random-access memory needed by the processing unit to operate
- **Input/Output:** Layer to handle incoming and outgoing signals
- **D/A and A/D conversion:** Conversion units, as the processing unit requires digital signals.

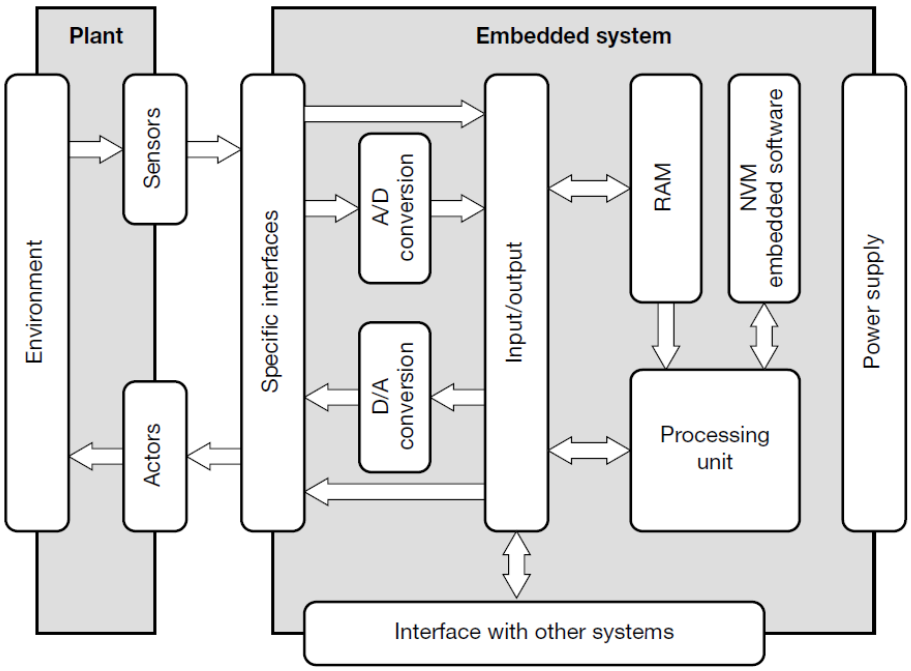

Figure 2.2: Generic scheme of an embedded system [4, p. 5]

The embedded system receives input signals from sensors and reacts with output signals to actors in order to change and manipulate the environment. Today the variety of embedded systems ranges from consumer electronics such as smart watches, music players, microwaves or freezer to traffic light systems, construction machinery or electronic aircraft systems.

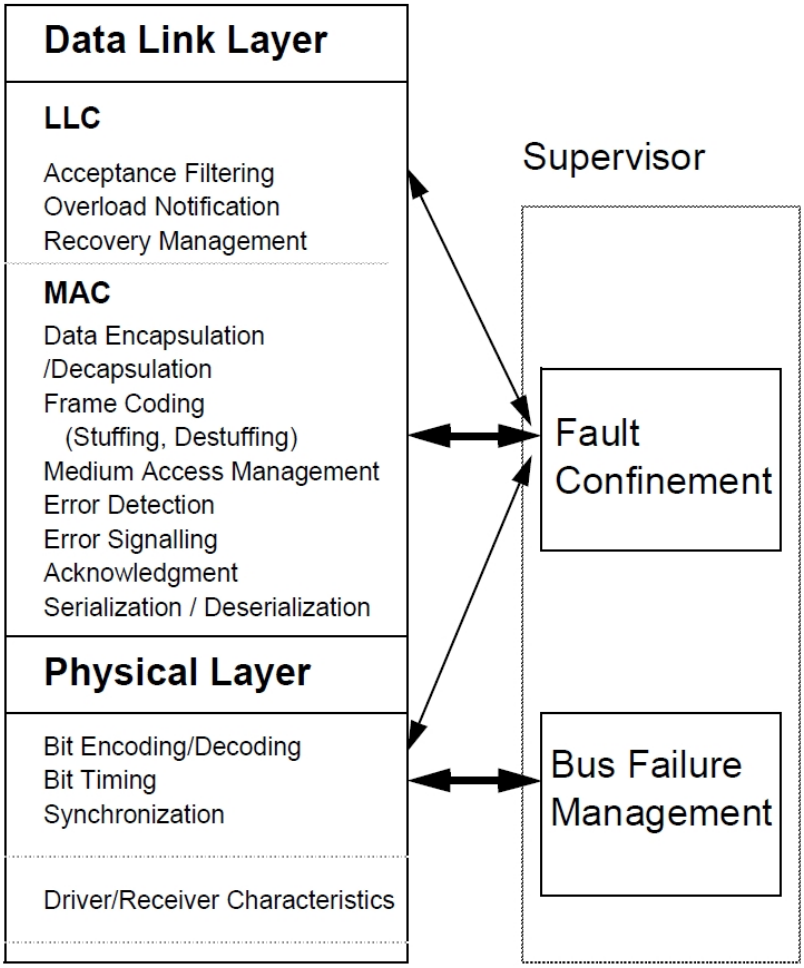

Figure 2.3: CAN layer specification [9, p. 37]

2.2 Controller Area Network

This section describes the Controller Area Network (CAN) standard and gives a basic overview of relevant fundamentals according to the *CAN specification Version 2.0-Part B* [9].

CAN is a serial communications protocol. It is message-based, allows devices to communicate without any host computer and was originally developed for electrical wiring in cars, but is used in many other fields today.

CAN is layered according to the ISO/OSI Model and includes (see figure 2.3):

- the Data Link Layer (including the Logical Link Control (LLC) sublayer and the Medium Access Control (MAC) sublayer

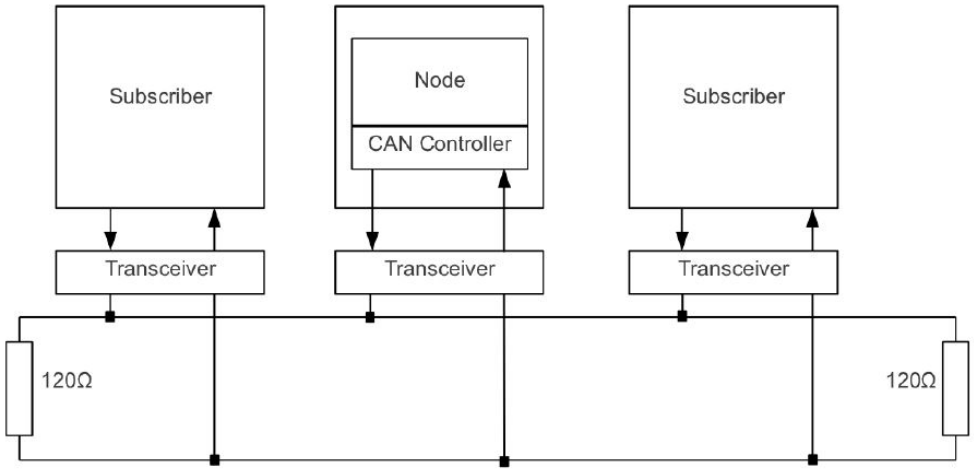

Figure 2.4: CAN layer specification [10, p. 19]

- the Physical Layer

The LLC sublayer is responsible for recovery management, message filtering and overload notification. The MAC sublayer manages message framing, arbitration, error detection and signaling. It acts as the kernel of the protocol and handles both messages received from and messages to be transmitted to the LLC sublayer. The *Physical Layer* takes control of Bit Timing, Bit Encoding and Synchronization. It represents the specification how the signals are physically transmitted.

Figure 2.4 shows a bus example with three subscribed participants. A subscriber consists typically of two parts: the node, which performs actions described by a user and the CAN controller, which enables communication (Physical Layer and Data Link Layer). CAN-High (CANH) and CAN-Low (CANL) are the two wires the CAN transceivers are connected to. They are used to determine the physical state of the bus. If the threshold voltage between CANH and CANL exceeds a defined maximum the state is defined as dominant state (logic '0'). If the threshold voltage doesn't exceed a defined minimum, it is called recessive state (logic '1'). The CAN bus is terminated at each end by a 120 Ω resistor to reduce signal reflection, where some of the signal power gets reflected instead of being carried to the far end of the cable and the subscriber could interpret a wrong bus status (dominant vs. recessive). [10]

2.2.1 Frame types and formats

The following subsection describes the different frame types and formats used in the CAN specification. In order to get a better understanding the data frame is discussed in detail.

The CAN communication is conceptualized using four different frame types:

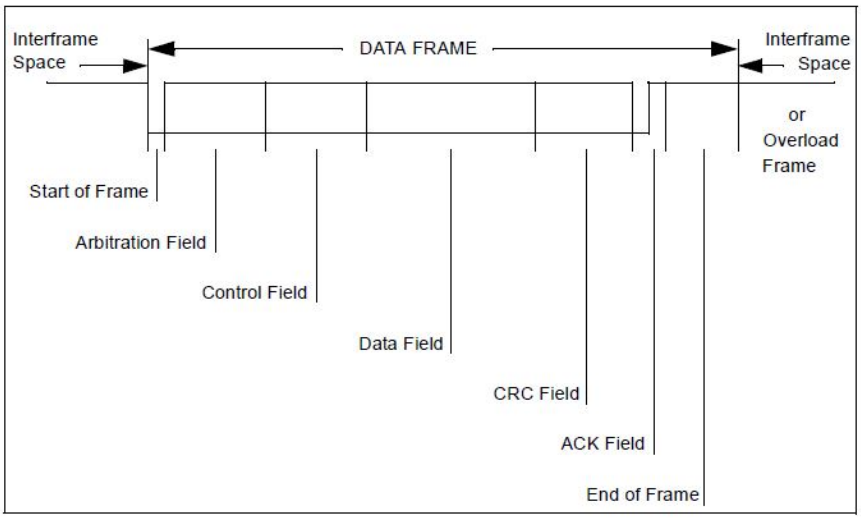

Figure 2.5: CAN data frame structure [9, p. 43]

- **Data frame:** Transfers data from transmitter to receiver
- **Remote frame:** Requests the transmission of a data frame
- **Error frame:** Signalize an error on the bus
- **Overload frame:** Provides delay between two data or remote frames

Furthermore, the CAN specification includes two types of frame formats, which differ in the length of the identifier field. Standard frames are defined with a 11 bit identifier field, whereas Extended frames contain a 29 bit identifier.

The data frame is divided into seven parts as visualized in figure 2.5.

The Start of Frame (SOF) defines the start of a new frame and is represented as a single dominant bit. The Arbitration field needs a differentiation between the two formats. In a Standard frame the Arbitration field includes the 11 bit identifier followed by the RTR bit (Remote Transmission Request). In an Extended frame the arbitration field consists of the 11 bit identifier (Base ID), the SRR bit (Substitute Remote Request), the IDE bit (Identifier Extension Bit), the 18 bit identifier (Extended ID) and the RTR bit (see figure 2.6.

In the Standard frame format the Control field contains the Data Length Code to indicate the number of bytes in the data field, the IDE bit and a reserved bit r0. In comparison, in an Extended format frame the IDE bit is transmitted as second reserved bit r1, as the IDE bit is located in the Arbitration field. The Data field represents the data to be transferred within an 8 byte data frame. The CRC field contains the CRC (Cyclic Redundancy Check) sequence and a CRC delimiter. The sequence is derived by a polynomial division

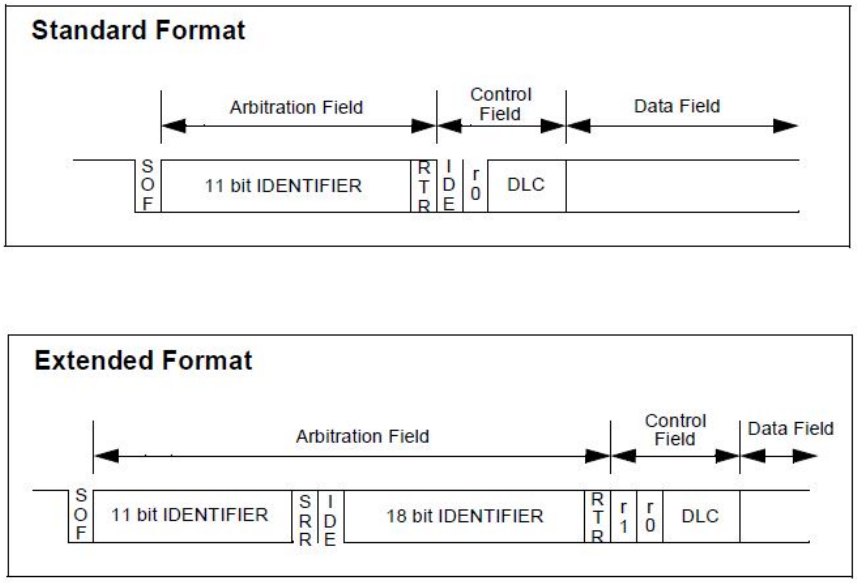

Figure 2.6: Standard vs. extended frame format [9, p. 44]

(coefficients by bit stream of SOF, Arbitration Field, Control Field, Data Field if present and '0' for the 15 lowest coefficients. The CRC delimiter is represented by a single recessive bit. The Ack field consists of the ACK Slot and the ACK Delimiter. The ACK Slot is used to acknowledge a message by a receiver (changing recessive to dominant). The ACK delimiter is always recessive. This means that the ACK Slot is surrounded by two recessive bits. The End of Frame (EOF) consists of seven recessive bits signalizing the end of a frame.

2.2.2 CAN database format

The CAN database (DBC) is a CAN communication description by the company *Vector*[11]. The database contains different properties of the CAN network, the ECU network connected to the bus, CAN messages, frames and signals with all their properties. It is typically used in combination with other CAN tools by *Vector* to visualize the messages of the CAN communication or test different values of signals.

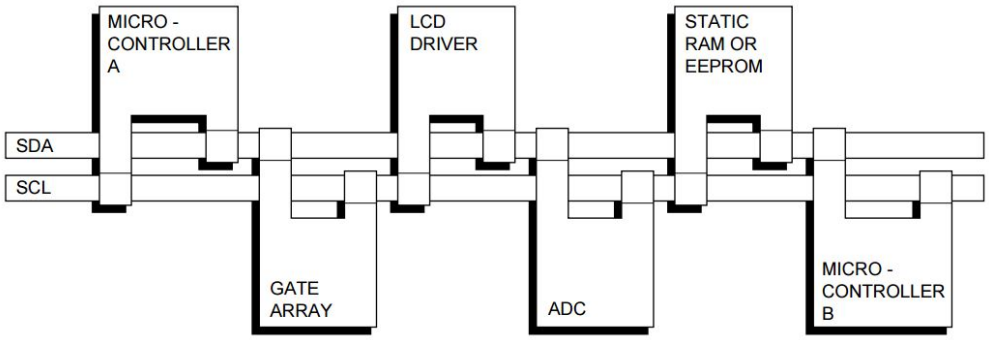

Figure 2.7: Example of an I2C-bus configuration using two microcontrollers [12, p. 7]

2.3 Inter-Integrated Circuit

I^2C (Inter-Integrated Circuit) is a multi-master and multi-slave serial computer bus designed for low-bandwith, short distance on board communication. All master and slave devices are connected through the serial data (SDA) and serial clock (SCL) wire, therefore a unique address is needed for all devices on the bus. Figure 2.7 shows an exemplary bus configuration with two microcontrollers and is used afterwards to describe the implemented protocol.

Due to the fact that *I^2C* works with a master/slave protocol, a master is needed to initiate the communication. The protocol can be described using the following steps:

1. **Start:** Microcontroller A (master device) sends a *START* bit to inform all devices on the bus to listen for their address on the SDA line.
2. **Addressing:** Microcontroller A (master device) sends the address of Microcontroller B (target slave device) and a *READ/WRITE* flag to signalize the communication direction.
3. **Acknowledgment:** Microcontroller B (target slave device) answers with an acknowledgment bit.
4. **Communication:** Depending on the *READ/WRITE* flag, both master and slave can transmit or receive data on the bus. The transmitter sends each time 8 bits of data, afterwards the receiver replies with an acknowledgment bit.
5. **Stop:** After the communication is completed, Microcontroller A (master device) sends a *STOP* bit indicating the communication is done.

Figure 2.8 visualizes the described steps.

If the data transfer was not successful (no ACK flag), the subsequent bit is called *Not Acknowledge* signal. The master can react with a *STOP* signal to close the connection

Figure 2.8: I2C protocol explanation [13]

or a *START* signal to restart a new transfer. The following conditions may lead to the generation of a *NACK* [12, p. 10]:

1. No receiver is present on the bus with the transmitted address, thus there is no device available to respond with an acknowledge.
2. The receiver is unable to receive or transmit because it is performing some real-time function and is not ready to start a communication with the master.
3. During the transfer, the receiver gets data or commands that it does not understand.
4. During the transfer, the receiver cannot receive any additional data bytes.
5. A master-receiver must signal the end of the transfer to the slave transmitter.

The concept behind I^2C tries to solve many interfacing problems for digital control circuits. Some of the benefits are listed below:

- Only two lines required (serial data line and serial clock line)
- Software addressable devices
- Master/slave relationships exist all time
- True multi-master bus (collision detection and arbitration to prevent data corruption)
- Number of bus participants limited only by the bus capacitance (dependent on various conditions)
- Different transfer modes:
  - Standard-mode: Serial, 8-bit oriented, bidirectional data transfers up to 100 kbit/s
  - Fast-mode: Serial, 8-bit oriented, bidirectional data transfers up to 400 kbit/s

- Fast-mode Plus: Serial, 8-bit oriented, bidirectional data transfers up to 1 Mbit/s
- High-speed mode: Serial, 8-bit oriented, bidirectional data transfers up to 3.4 Mbit/s
- Ultra Fast-mode: Serial, 8-bit oriented, unidirectional data transfers up to 5 Mbit/s

2.4 Fundamentals of Testing

The following terminology descriptions are based on the *IEEE Standard Glossary of Software Engineering Terminology* [14], the *ISTQB Certified Tester Foundation Level Syllabus*[15] and Glenford Myers book "The Art Of Software Testing" [16, p. 85 ff].

2.4.1 Verification & Validation

In connection with software quality management, the terms *Verification* and *Validation* may be relevant. It is important to state, that both concepts shall be understood as comprehensive terms, where software testing serves as one possibility to perform software verification and validation.

Verification

The term *Verification* is defined in the *IEEE Standard Glossary of Software Engineering Terminology* as "*the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements.*"[14, p. 85] The term is also related to the question "*Are we building the product right?*" and tries to verify, if the software conforms to its specifications.

Validation

In contrast to *Verification*, the expression *Validation* is understood as *"the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase."*[14, p. 85]. Under this definition, *Validation* addresses the questions "*Are we building the right product?*" and "*Does the software conform to the users' needs?*".

Verification & Validation

Following the previous definitions, *Verification & Validation* are defined as *"the process of determining whether the requirements for a system or component are complete and correct, the products of each development phase fulfill the requirements or conditions imposed by the previous phase, and the final system or component complies with specified requirements."*[14, p. 85]

Figure 2.9 visualizes the concept of *Verification & Validation*.

2.4.2 Necessity of testing

Software applications are present in almost all areas of life. Not correctly working software can result in many problems (e.g. loss of money, time, injuries, death). Faulty software mostly starts with an error of a human being (i.e. mistake), which produces a defect (i.e. bug). If the defect gets executed, it causes a failure. The second failure possibility may

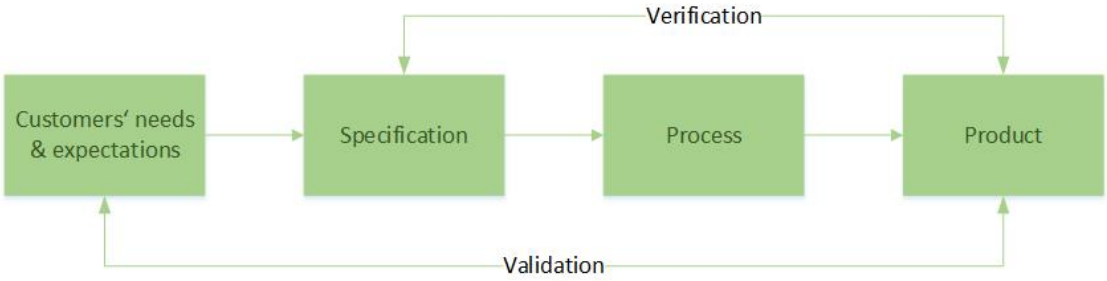

Figure 2.9: Verification & Validation

be caused by environmental conditions, such as magnetism, electronic fields, extreme heat and cold and many more.[15]

Through testing activities it is possible to measure the quality in terms of defects found. When finding defects, the quality can be increased by fixing them.

2.4.3 Testing principles

The following seven testing principles have been established over the past decades.[15, p. 14]

Testing shows presence of defects

Testing can show that defects are present, but cannot prove that there are no defects. Testing reduces the number of undiscovered defects remaining in the software but even if no defects are found, it is not a proof of correctness.

Exhaustive testing is impossible

Testing everything including all combinations of inputs and preconditions is not possible. So, instead of doing the exhaustive testing, risk analysis and priorities should be used to focus on testing efforts.

Early testing

In the software development life cycle testing activities should start as early as possible and should focus on defined objectives.

Defect clustering

A small number of modules contain most of the defects discovered during pre-release testing or show the most operational failures.

Pesticide paradox

If the same kinds of tests are repeated again and again, eventually the same set of test cases will no longer be able to find any new defects. To overcome this "pesticide paradox", it is very important to review the test cases regularly, and to write new and different tests in order to exercise different parts of the software or system to potentially find more defects.

Testing is context depending

Testing is basically context dependent. For example, safety–critical software is tested differently compared to an e-commerce site.

Absence–of–errors fallacy

Finding and fixing defects does not help if the system built is unusable and does not fulfill the users' needs and expectations.

2.4.4 Test levels

As the original *V-Model* includes four development levels, also testing activities are divided into four different levels, namely:

- Component (unit) tests
- Integration tests
- System tests
- Acceptance tests

As there exist a number of variants of the *V-Model* and other software development models, fewer, more or different test levels may be applied depending on the developed product.

Component (unit) tests

The term component test or unit test is defined as "*testing of individual hardware or software units or groups of related units.*" [14, p. 84]

Integration tests

Integration tests can be described as "*testing in which software components, hardware components, or both are combined and tested to evaluate the interaction between them.*" [14, p. 41]

System tests

"*Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements*" [14, p. 78] indicates the practice of system tests.

Acceptance tests

"*Formal testing conducted to determine whether or not a system satisfies its acceptance criteria and to enable the customer to determine whether or not to accept the system.*" [14, p. 2] are the aims of acceptance tests.

2.4.5 Test types

The standard *ISO/IEC/IEEE 29119-4-Test Techniques* [17] provides definitions of test techniques, which can be used during the phases test design and implementation. The suggested techniques are categorized in three broad groups:

- Specification-Based Test Design Techniques
- Structure-Based Test Design Techniques
- Experience-Based Test Design Techniques

The following paragraphs give a general explanation of the categories with representative examples. For a full visualization of the entire categorization please refer to figure 2.10.

Specification-based tests

Specification-based tests, also called black-box or functional tests, refer to test techniques where the internal mechanism or structure of a system gets ignored. The tests focus only on the generated outputs after selecting inputs and execution conditions. [14, p. 34]

Some well-known representatives are:

- Equivalence Partitioning
- Boundary Value Analysis
- State Transition Testing

Structure-based tests

Structure-based, white-box or structural tests are based on an identified structure of the system. The aim is to test the system at source code level to get an error-free environment. [14, p. 74]

Structure-based test techniques include:

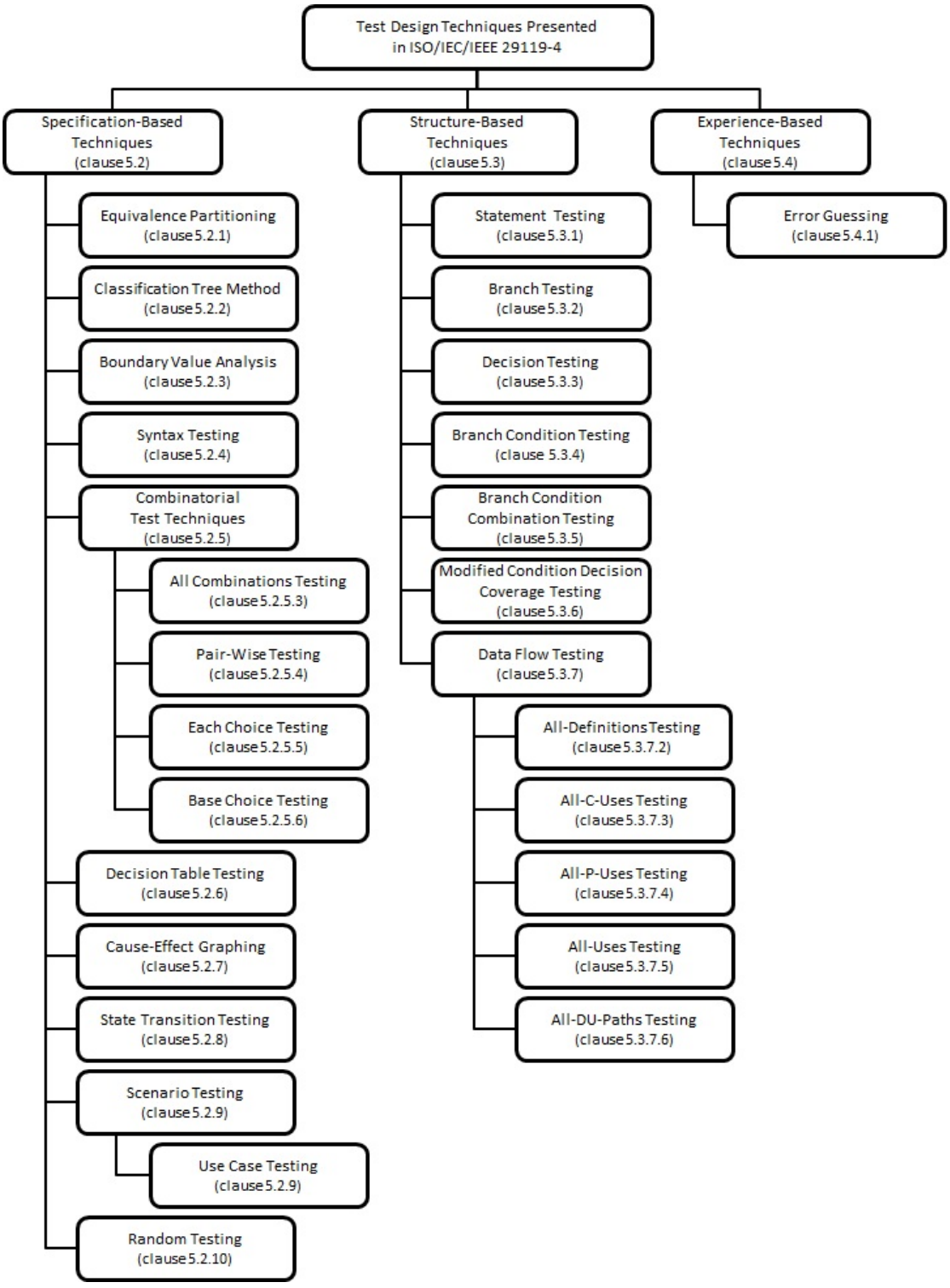

Figure 2.10: ISO/IEC/IEEE 29119-4-Test Techniques [18]

- Statement coverage
- Decision coverage
- Modified condition/decision coverage (MCDC)
- Path coverage

Experience-based tests

Experience-based tests are derived from the experience, skills and intuition of the tester. The most popular technique is *Error Guessing*.

CHAPTER 3

State of the art

Testing embedded software on integration and system level proves to be extremely difficult due to the high number of dependencies. Therefore, such projects require specific test methods and tools, especially when performing safety-critical functions. Currently, the combination of integration testing in the area of embedded applications and BDT isn't represented in the literature. In addition to the fundamental state of the art testing concepts, the following subsections outline related researches in both fields.

3.1 TEmb method

TEmb is introduced by Broekman and Notenboom and describes a method supporting the search of a suitable test approach for a particular embedded system. Figure 3.1 depicts an overview of the methodology.

The idea is that the basis of every approach consists of generic elements related to the four cornerstones of structured testing:

1. **Lifecycle**: Defines the process definition of test activities and shall help to demonstrate the steps which must be done. The lifecycle is divided into the phases preparation, specification, execution, completion and planning & control. Ideally, the activities shall get settled as soon as possible.

2. **Infrastructure**: Includes the test environment, tools and the office environment to perform the designed test activities. The test environment consists of hardware and software (e.g. for different stages of the development), databases (holding test data) , simulation and measurement equipment (equipment to simulate or measure specific signals). Tools shall support different phases described in the lifecycle (e.g. configuration management, requirement analysis, test case and data generator, coverage analyzer, report visualization). Office environment is a critical factor to

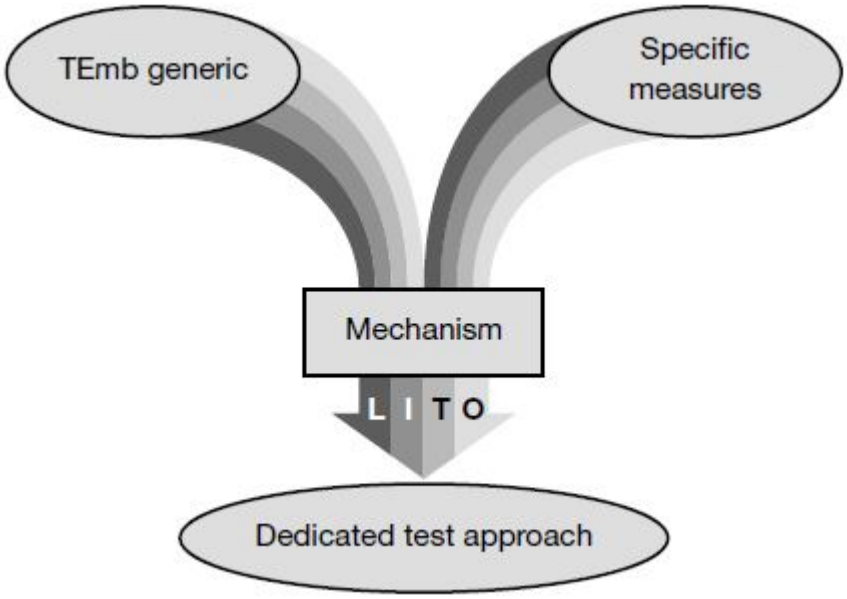

Figure 3.1: TEmb overview [4, p. 8]

increase the quality of the test process (e.g. necessary environment and tools are ready when needed).

3. **Techniques**: Standardizes the performance of test activities by offering working methods to track the progress and evaluate results (e.g. techniques for strategy development, test design, checklists, test automation).

4. **Organization**: Deals with the structure of the team, roles performed by each member as well as required expertise, potential necessary training or additional working staff.

All cornerstones must be considered equally to consider the test approach as structured. For most projects, this generic test process is not concrete enough and the approach has to be enriched with specific measures. This process is called "mechanism for assembling the dedicated test approach" [4, p. 7]. Measures are chosen to cover business risks occurring due to poor product quality and to deal with issues related to specific system characteristics. The relation between measures and characteristics can be represented in the so called LITO-matrix in order to group all identified attributes and to define a matching testing approach.

TEmb is often interpreted as extension to the classical risk-based test strategy. The analysis of potential risks and system characteristics leads to certain solutions, which tackle the respective problems identified for the targeted embedded project. [4, p.10]

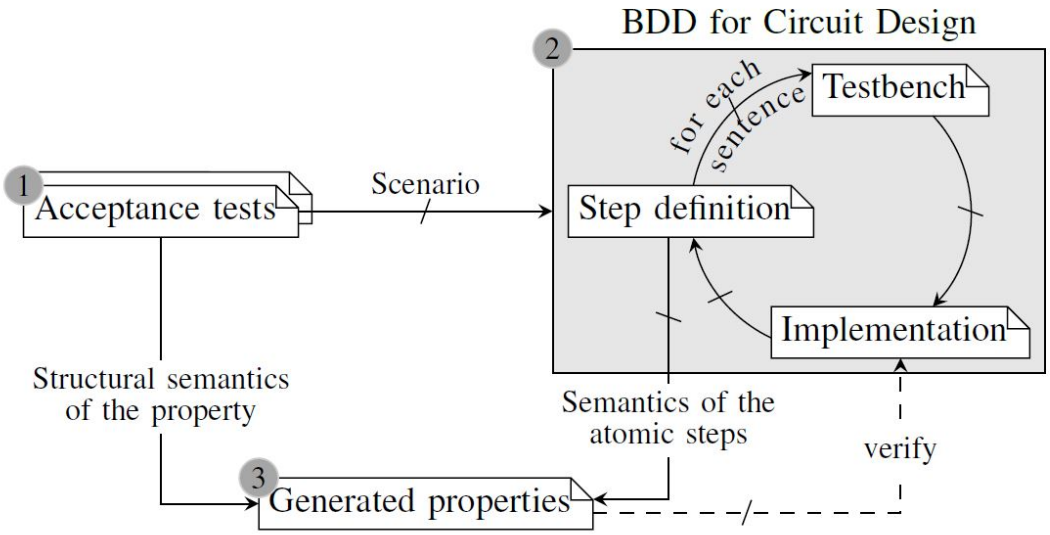

Figure 3.2: Adapted BDD flow [19, p. 3]

3.2 BDD for Circuit Design and Verification

Diepenbeck et al. propose a work flow for circuit design and verification based on Behavior Driven Development (BDD) [19]. First, a customized BDD workflow for the circuit design, using a Hardware Description Language (HDL), is modeled. Second, an automated generalization of the test cases is realized. Thus, properties may be generated to verify the design formally. The realization propose the following advantages [19, p. 1]:

1. Circuit design using a modified BDD flow
2. Formal and improved verification due to automatically generated properties

Figure 3.2 visualizes the realized flow for circuit design. The following enumeration describes the proposed steps by the authors and references to the numbers of figure 3.2.

1. The first step does not differ from the original BDD methodology and aims at describing the hardware features by acceptance tests in natural language using *Gherkin* [20].
2. The gray rectangle represents the actual creation of the circuit design. It comprises the generation of the step definitions, the test bench and the actual implementation. The added circle emphasizes an iterative implementation process. As long as

test cases fail, further development is necessary. This step comprises most of the adoptions, as the circuit design needs the testbench to be executed. The authors outline the separation of actual circuit design and testbench implementation as well as the timing constraints when simulating hardware as major issues and customization requirements.

3. The created acceptance tests are generalized and Property Specification Language (PSL) standardized properties [21] are generated to formally verify the design by standardized algorithms.

The necessary customization is realized within three tasks: testbench implementation, timing resolver and execution of tests. The testbench holds a skeleton in which the test code gets inserted, the module gets instantiated and the signals to be tested get connected. Timing is performed within an internal counter denoting the counter of the cycle. Only relative timing for a step definition is possible, as the step definition may be reused by different scenarios. The execution is performed as follows: First, the module and skeleton is determined for each scenario of the feature definitions using their respective background sections. For each step of the scenario, a testbench is created spanning over the respective amount of steps for failure detection. Afterwards the timing information is added and the respective step code is inserted into the skeleton. Finally, the nested testbench gets compiled and executed.

The proposed work flow of BDD for circuit design has proved to be a successful approach to introduce an agile technique into the target field. Especially the property generation leads to an automated verification possibility against all test patterns. The authors suggest further development in terms of the timing component.

3.3 Verification pattern approach

Wei-Tek Tsai et al. propose the development of test scripts for testing embedded systems based on verification patterns [22]. The approach follows the concept of classifying system scenarios into patterns. For each classified pattern, a test script template is implemented, which comprises the tests of all scenarios belonging to the respective pattern. This process enables the re-usage of the same test scripts for several scenarios and keeps the costs to a minimum. If system requirements change, the already existing test script may be applied for a new scenario belonging to an already in-use pattern.

A critical issue identified is the determination of the number of patterns for a reasonable test coverage for large industrial applications. A considerable number of patterns would increase the costs for identifying the patterns, maintaining the templates and customizing them for the scenarios. Based on the evaluation in different fields, they encountered that only a small number of patterns are necessary even for complex systems including more than 100.000 scenarios [22, p. 69]:

Characteristic	Conventional	Verification pattern
No. of requirement items	99	114
Use of verification framework	No	Yes
LOC (.h)	60.211	6.200
LOC (.cpp)	76.412	10.100
Total LOC	136.623	16.300
LOC/requirement	1.380	143

Table 3.1: Evaluation results verification pattern approach [22, p. 75]

- Safety critical medical device: eight patterns covered 95% of the scenarios
- Ultra-reliable communication processor: four patterns covered 75% of scenarios
- Industrial process-control software: 29 patterns covered 100% of scenarios

Those examples show the potential of the approach and that the technique is not limited to the field of medical devices, for which it was initially targeted. Table 3.1 shows a comparison between the use of verification patterns and a conventional approach. The group using the verification pattern approach reduced the code size to test the system by almost 90% (1.380 to 143 lines of code (LOC) requirement).

3.4 Design and test architecture

Kang et al. present a design and test architecture for embedded software systems. The test part comprises three test steps for the embedded software:

1. **Functional test**: Unit and integration tests to verify the system behavior performs in compliance to the requirements
2. **Host static test**: The software subsystem is tested on the host machine
3. **Target test**: The software is flashed to the target system and integrated within the hardware system.

The proposed method analyses the execution on the target system and is able to extract specific information (e.g. inputs, outputs, conditions) by the usage of a test driver. [23]

3.5 Extended Finite State Machines

Guerrouat and Richter propose the usage of extended finite state machines to test embedded systems and present fault models to automate error detection. The specification

comprises of the behavior description of the environment (sensors and actuators) and the controller itself by state machines. Each function is modeled as one or many state machines and those models may interact with each other via events.
They conclude that the specification based on extended finite state machines enables error diagnostic specifying control and flow. The concept is applied on a simulation, therefore they propose to apply the approach to a real-life system in the field of automotive. [24]

3.6 Vector Informatik GmbH

The Vector Informatik GmbH conceptualizes and develops software and hardware, handling the networking of electronic control systems based on different bus systems [25]. The developed products target especially the automotive world, as their tools include many features tailored specifically to this business field. Nevertheless, the provided solutions get also used in other fields of embedded applications. The following paragraphs include a set of common tools for different testing purposes.

CANoe

CANoe [26] is a powerful software suite for the development, test, simulation and analysis of ECU networks. It supports a variety of different bus standards (e.g. CAN, Local Interconnect Network (LIN), Media Oriented Systems Transport (MOST)) and diagnostic services (e.g. Unified Diagnostic Services (UDS) [27], Universal Measurement and Calibration Protocol (XCP) [28]).

CANoe.DiVa
CANoe.DiVa [26] is an extension of CANoe and realizes the generation of automated tests of implemented diagnostic services. The test cases are generated based on the diagnostic description of an ECU. For this reason, a huge amount of tests can be generated easily and executed within CANoe. The generated report may be imported into CANoe.Diva, which offers a variety of filter options.

vTESTStudio
vTESTStudio [29] enables creating automated tests for ECUs. It includes the following test approaches:

1. Table-based: Defining graphical test tables specifying steps, setup and tear-down functionality.
2. Programming-based: Develop test application using a self-developed programming language called CAPL.

Test cases realized with vTESTStudio can be imported into other tools provided by Vector (e.g. CANoe).

Test hardware
In addition to the software packages, Vector provides a variety of hardware interfaces to physically address the supported technologies (e.g. CAN, LIN, MOST), data loggers, as well as analog and digital measurement hardware. [30]

CHAPTER 4

Methodology

The following chapter describes used methods and approaches to face the problem statement of this thesis. It addresses topics such as the development process, used test methods and the evaluation aspects.

4.1 Behavior Driven Development

BDD combines principles of test driven development, domain driven design and object oriented design. The technique comprises traditional documentation, which can be written and read by all business stakeholders, but has the advantage to be executable by a computer. Therefore, the documentation becomes a dynamic and living part of the test implementation rather than a one time created artifact. This method saves significant resources when keeping code, tests and documentation synchronized. [31, p. 4 ff]

Automated acceptance tests originate in eXtreme Programming. The idea is to write automated tests collaboratively (i.e. developer and stakeholders) to describe the desired behavior of the system. The tests are called acceptance tests, because they should define the system's functionality in order to satisfy the stakeholders' requirements, eventually resulting in an acceptable outcome.

The figures 4.1 and 4.2 compare a traditional development process with the usage of BDD.

When performing a traditional approach, there are many opportunities for information to get lost, ignored or misunderstood. After discussing the desired needs (step 1), all steps are performed independently by the business analyst (step 2), the developer (step 3), the tester (step 4) and the technical writer (step 5). Therefore, there is a high probability, that the software does not exactly meet the stakeholders' requirements and that the documentation is not consistent.

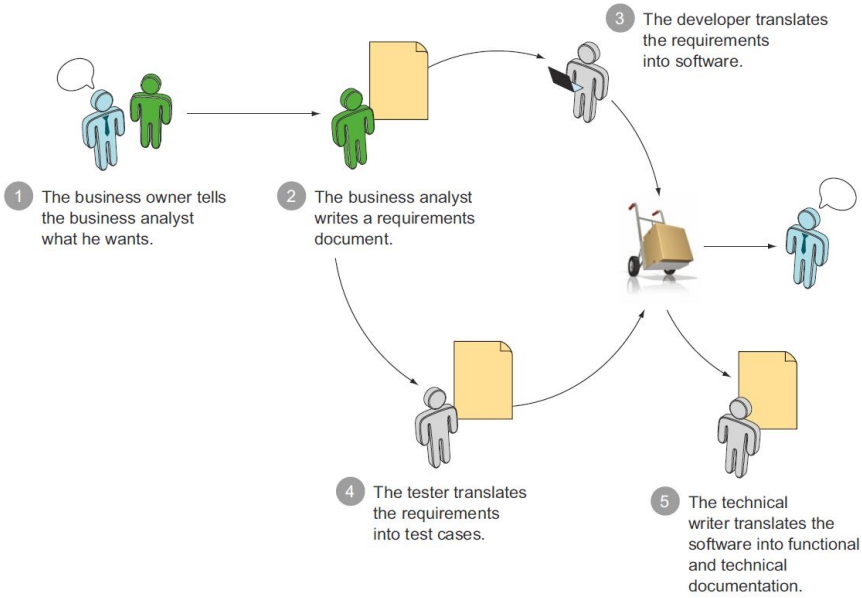

Figure 4.1: Traditional development process [5, p. 5]

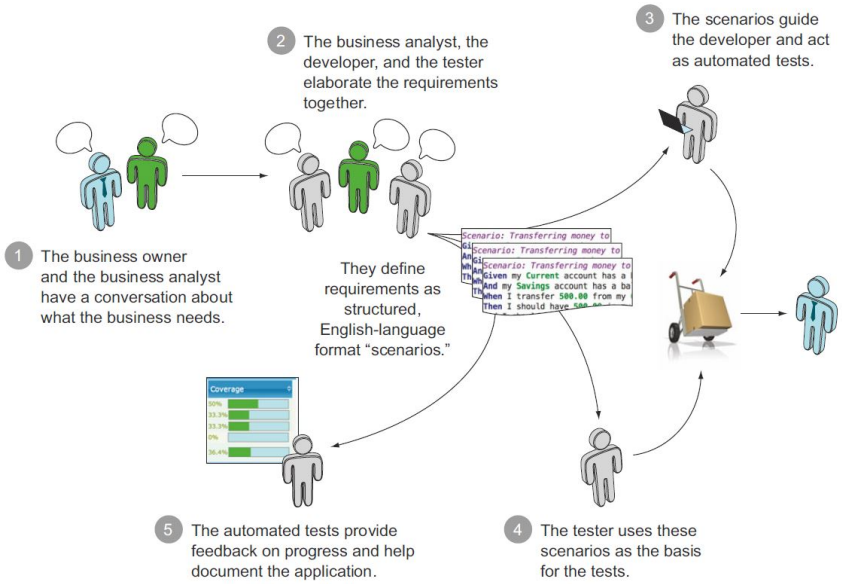

Figure 4.2: BDD development process [5, p. 6]

In comparison to the traditional approach, within BDD the requirements are elaborated together using a common language and all further steps are based on those requirements. The use of examples and conversation reduces the amount of lost or misunderstood information. In order to define BDD tests together, they are written in a language, that business stakeholders can understand. This method includes them into the process and helps understand their requirements. [5, p. 3 ff]
The implementation described in chapter 5 aims at evaluating the combination of BDT and embedded software on an ECU. Due to the limited time frame for the realization of the master thesis, not the entire BDD process, which includes the development of the application, is applied, but an existing software is tested by using BDT. In order to achieve the desired outcome, an API is developed, which provides the functionality to perform integration tests on an embedded application. Test specifications are defined using the language *Gherkin*, which describes the system's behavior by examples in natural language, whereas the respective implementations of the test cases are realized in *Python*.
The proposed methodology shall increase the performance of testing activities for the following reasons:

1. Integration tests of embedded applications are especially complex, as not only software but also hardware is involved. Often, the same test case has to be executed several times and because of requirements changes the entire integration test process has to be repeated. Through the automation of integration tests, recurring work effort and the probability of errors during manual test activities may be reduced.

2. Many projects fail or can not be completed within estimations due to missing or unclear requirements. BDT encourages not only the definition of the system's behavior through examples using the language *Gherkin*, but also the communication between the different stakeholders in a project in order to ensure, that each involved project participant knows exactly what the system should do.

3. Through the usage of *Gherkin*, one part of the test implementation is defined in natural language and therefore directly usable as test documentation, which otherwise has to be created and maintained additionally.

4. As the defined behavior of the system is executable and part of the test implementation (e.g. realized in *Python*, an automated consistency check between test documentation and implementation is ensured. This fact reduces maintenance effort after a requirements change, as affected parts of the test documentation or implementation don't have to be traced manually. Moreover, it is not possible to forget changes in one of both information sources, as the test execution will fail.

4.2 Cooperation partner

This thesis has involved analysis of real world data as well as discussions with industrial experts. For confidentiality reasons the company is referred to as ECU Ltd.

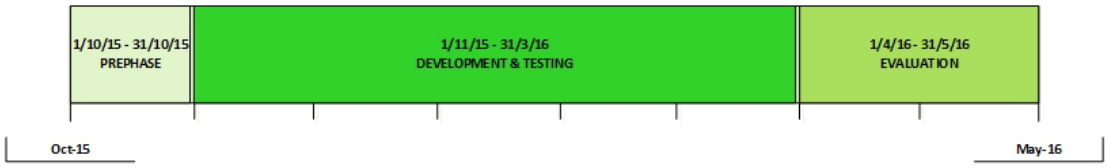

Figure 4.3: Timeline of the development process

The ECU Ltd. operates in the domain of electronic control systems and offers its services as a cooperation partner. Thus, it's possible to access special resources such as a controller unit, a project to test the implementation, different test hardware and also knowledge in the target field. ECU Ltd. provides two technical experts to support the whole process. ECU Ltd., all names and all participants are pseudonyms, which are used throughout this thesis to protect the anonymity of the organization and individuals.

4.3 Development process

The underlying scientific work of this thesis has been executed in three phases (see figure 4.3). The first phase, called "prephase" in the context of the work, deals with the research of literature to gain a better insight knowledge as well as the selection and conceptualization of the infrastructure for the practical part. The second phase, "development & testing" reflects the actual software development process and the prototypical testing of the created artifact. The last phase, "evaluation" applies oneself to the evaluation through expert interviews.

The phases will be described in detail in the following subsections.

4.3.1 Prephase

The first part of the prephase handles literature reviews in order to satisfy necessary requirements and to allow a successful development stage. Therefore, both branches, general embedded development and testing as well as behavior- driven development and testing, have to be addressed. The next step is to evaluate and especially choose concrete software (hardware is mostly predefined due to limited availability) in order to design the future infrastructure. In that case different kinds of BDD tools and programming languages are tested and the one which meets the most requirements is chosen. As the final version of the developed API is tested using an existing customer application of ECU Ltd., a meeting including a small questionnaire is held to better understand the status quo and present issues from daily business in order to address those specifically during the development phase.

4.3.2 Development & Testing

The development of the API follows an incremental and iterative process, which draws to a very considerable extent on the agile software development and project management

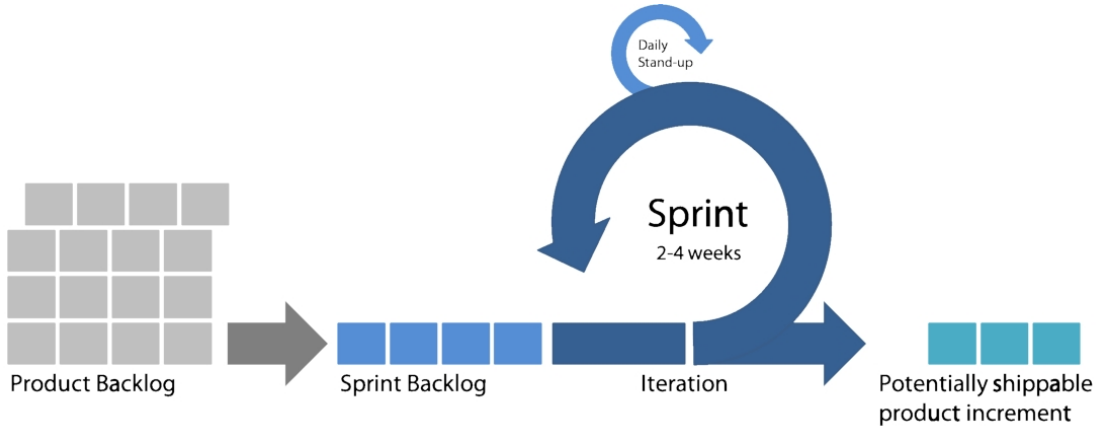

Figure 4.4: Scrum process [32]

framework Scrum (see figure 4.4), which is widely suggested in practice for mid to large product developments involving more than one person.

First, the product backlog is defined, which contains a list of all requirements and features of the desired product. The sprint backlog is derived from the product backlog and holds activities to perform during the next sprint by the team. Therefore, the sprint backlog is updated or redefined for each new sprint. A sprint is a certain time frame (usually two to four weeks) and represents one iteration cycle. During the interval, all tasks defined in the current spring backlog shall be performed. Additionally, a daily stand-up meeting is held to talk about the finished work of the day before, planned activities for the current day and potential obstacles slowing down the progress. This cycle is performed till the end of the development and it results in a shippable product.

The process of this work follows the same pattern but is slightly adapted to the specific circumstances. As the author of this thesis acts as a single developer and tester, the sprint length is set to one week. The sprint meeting with both technical supervisors is performed through a telephone conference (due to distributed locations), which shall help to track the progress, plan the work of the next sprint, discuss issues and get instant feedback on finished activities in order to better plan further tasks. The daily stand-up is not performed due to single developer and location restriction, but the consultation with the supervisors in case of decision support or significant issues is ensured at any time.

After the finished development of the artifact, the functionality and usability is tested through a prototypical test implementation. It shall visualize the usage as well as possible advantages and disadvantages. This realization provides the basis for the evaluation phase. For a detailed hardware setup, environment description and implementation please refer to chapter 5.

4.3.3 Evaluation

To evaluate the implementation, expert interviews are applied. One major reason is the fact, that specific knowledge is required to make significant statements about the developed tool and it would have been very difficult and time consuming to find enough participants in order to get accurate results from a quantitative method. Moreover, the direct feedback of the interviewed experts is used to make improvements, give recommendations and plan further activities in the field. The following evaluation methodology follows the work of Meuser and Nagel [33] [34].

The introduced process to perform expert interviews includes the following steps and activities:

- **Specify research questions and field of interest:** The first step is to identify the research questions for the interviews.
- **Select experts:** According to Meuser and Nagel [33, p. 443], experts have to be chosen respectively to the specified research question, knowledge, position and responsibility.
- **Develop interview guide:** Before asking the first question, the expert should be informed about the goals of the research, scope of the interview and further information about the processing of the information gained from the interview. Moreover, the researcher should not focus directly on the specified research questions, but start with questions regarding the expert's background and experience to create a better environment. Afterwards, research related questions can be asked, where it is also a good idea to ask sub-questions. If the expert agrees, the interview should be at least audio recorded to simply the transcription process and increase the evaluation quality. Finally, the researcher should close the interview formally.
- **Test interview guide:** The interview guide should be tested before the actual interview is performed. It is appropriate to perform the guide test with a person, who has scientific background and / or conducted expert interviews before. This test is used to find redundant or irrelevant questions.
- **Transcription:** To analyze the performed interview, a transcription using the recorded session should be created.
- **Paraphrasing:** Paraphrasing is necessary to perform data aggregation. Not all parts of the interview need to be paraphrased, but only the context relevant parts regarding the intended research output.
- **Headlines:** Headlines are used to group statements in an interview to specific thematic points. This step is easier with the use of interview guides due to the similar structure. [33, p. 458]

- **Thematic comparison:** This step uses the data gained in the previous steps to show common opinions but also deviations and differences among the interviews. The following steps proposed by Meuser and Nagel are integrated in this phase.

The research questions addressed in this master thesis are shown and explained in chapter 1.1. In total, four expert interviews are performed in two sessions, the first one with three experts (due to time restrictions) and the second one with a single expert. All experts have first-hand and in-depth experience in software development and testing in the area of embedded systems and applications. To get accurate and unaffected results from the shared interview session, specific questions are answered independently on paper. The developed interview guide for the expert interviews is available in appendix A.5 and A.4. The interview guide test is performed with a student of the Vienna University of Technology who has performed expert interviews before. The transcriptions of the interviews with headlines are attached in appendix A.6 and A.7.

Furthermore, a final presentation with feedback round is held. The participants consist of employees of ECU Ltd. who are not integrated in the design or development process and employees of ECU Ltd.'s customer which provides the ECU application to test the implemented solution. As the status quo meeting is held before the actual implementation, the final presentation gives an overview on how the concept is implemented, which functions are already available and how it can be positioned in the current test infrastructure.

CHAPTER 5

Implementation

The following chapter deals with concrete design and implementation decisions. It comprises the components of the developed API, additional software and frameworks used for the concept as well as the conceptualization of test setup. The API bears the name *embhave*, which stands for a combination of the terms "embedded" and "behave". In the further course of this work, the name *embhave* is used for the developed artifact.

5.1 Architecture

5.1.1 Deployment & usage

Figure 5.1 depicts an overview of the deployment and the interactions between the different interfaces, software and hardware parts in the test setup for the present master thesis. The diagram comprises two execution strategies, either by the user itself or by a build server. Due to resource restrictions, only the first variant is covered within this master thesis. When executing the test cases through the command line, the test implementation injects functionality of *embhave*, which provides accessibility to the underlying hardware and the ECU. The test report is generated by the test framework and can be imported directly by a build server or visualized by the user.
behave includes a mechanism called environmental control, which may be used to define code to run before or after certain events during the test execution [35]. Within the underlying test setup it is used to initialize the *EmbhaveController* object, which is available in all step definitions afterwards. In the course of the initialization, *embhave* parses the configuration file, which holds all parameters for the usage of the test hardware (e.g. type of CAN adapter, address of I/O boards) and checks if required values are missing. Additionally, DBC and mapping files may be specified inside the configuration file to use them for the CAN communication. This part of the implementation supports the user when creating tests for the CAN communication, as the necessary signals and

frames don't have to be created manually, but already existing information can be reused for testing activities.
In case of missing parameters or errors during initialization, *embhave* terminates the test execution and throws the respective exception. If the initialization is successful, the test runner of *behave* starts to parse the feature definition files and executes the corresponding step definitions. The progress can be tracked on the command line, as each step gets individually printed with its respective outcome (e.g. passed, failed). After all feature definitions are executed, *behave* creates a test report in XML format, which may be imported by a visualization tool or build server.

The following enumeration summarizes inputs, operations and outputs of the realized test setup:

1. **Inputs**:
   a) ***.feature & steps/*.py**: Feature files holding the test specification are defined using *Gherkin* and step definitions implementing the stated steps inside the features are realized in *Python*.
   b) **embhave.cfg**: Configuration file to specify all necessary properties for the test hardware.
   c) ***.dbc**: DBC file specifying the CAN communication (frames and signals transmitted from or to the application) which can be used by *embhave* to send a specified CAN frame or to check a value inside a received frame.
   d) **DBCMapping.cfg & IOMapping.cfg**: Mapping from keywords to specific CAN signals, inputs or outputs, which may be used in the feature definitions to address a specific signal. Thereby, cryptic variable names or addresses in hexadecimal format may be paraphrased and readability gets increased.
2. **Operations**: The high level functions of the *EmbhaveController* are used inside the step definitions to access low level function for the CAN or I/O tests.
3. **Output**: All execution information (outcome of each executed step) gets summarized inside a generated test report in XML format.

As the desired artifact comprises interfaces to both software and hardware components, a suitable environment and architecture has to be chosen. Taking into account various factors, *embhave* is implemented using the programming language *Python* (version 2) due to following reasons:

- **Popularity:** *Python* has been ranked among the most popular coding languages for the last couple of years. Therefore, *Python* guarantees further development in the future within a well established and supported programming language. [36]

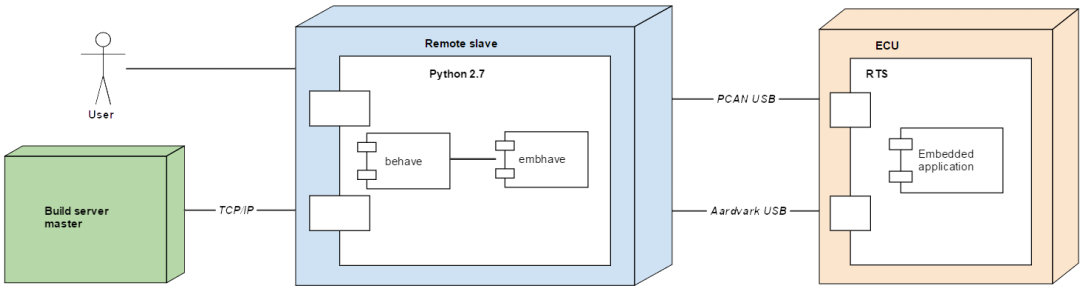

Figure 5.1: Overview of the deployment

- **Available interfaces:** All necessary interfaces to software or hardware are available in *Python*. More details are explained in the following sections.
- **Quality:** The *Python* syntax focuses on readability, re-usability, maintainability, coherence and software quality in general. As the chosen test concept focuses on similar aspects, *Python* can be seen as an obvious choice. [37, 3]
- **Tool set:** *Python* can be seen as a hybrid programming language. It combines features of typical script languages (e.g. Perl) and system development languages (e.g. *C++* or *Java*). This enables a large applicability. [37, 15]
- **License:** *Python* is completely free to use and distribute. It is licensed by the *Python Software Foundation* and the license is compatible with the GNU General Public License (GPL). [38]
- **Portability:** *embhave* is developed and tested on Windows 7, but the portability on a UNIX based operating system is quite probable. The Python standard is written in ANSI C and can be compiled to nearly every platform (e.g. Windows, Unix / Linux, Mac OS, real-time systems, PDAs or cell phones).[37, 14]

embhave comprises different functions to support the generation and execution of integration tests. The main benefit of an API is its independence to the rest of the application chain. This implementation of the test setup is realized with *behave* [39] as test framework, but it can be accessed from every *Python* file.

To combine the intended functionality, two different applications are integrated. The main distinctions are:

- **Command line execution**: The command line contains again two different purposes. First, it is used to visualize basic information (e.g. the current version). Second, in order to reduce implementation effort of the step definitions, a template mechanism is integrated, which is also accessible from the command line. It can be used to generate step files with basic functionality, which have to be

written otherwise repeatedly for each new project. The generator can be executed from the command line with `embhave -generate TESTTYPE OUTPUT`, where TESTTYPE corresponds to the desired template (currently only CAN or IO) and OUTPUT to the output file. The input is derived from a template file inside the tool which can also be extended or reduced by the user.

- **Use as library**: The library functions are the default usage. Methods in the main part, called *EmbhaveController*, ensure accessibility of implemented methods and object structures. To provide the mandatory settings, e.g. for the configuration of test hardware, a configuration file is used. Chapter 5.1.2 describes the structure and functionality of *embhave* in detail.

Compared to manually processed integration tests, the proposed approach improves test activities for several reasons. First, recurring test executions can be performed fast and more efficient, as the effort is much lower after the creation of the test automation is finished. Second, test steps and outcomes have not to be reported manually, as the test reports are generated automatically by the test framework. Additionally, the test report is standardized and importable by several visualization tools or build engines. Thus, different test executions are grouped within one tool and easily comparable. Finally, the proposed test approach enables the execution of the test specification, as it is part of the test implementation. This make an automated consistency check between test implementation and specification possible, as the test execution will not be possible or fail when differences appear. Moreover, the test specification is generated in human readable language, whereby a traditional documentation of the test implementation becomes obsolete. In this approach, the test implementation comprises both specification and actual implementation.

5.1.2 Class diagram

The class diagram is a structural diagram defined by the Unified Modeling Language (UML). It describes the structure of a system by visualizing classes (including attributes and methods) and their relationships. The following list describes the implemented classes briefly, whereas the subsection 5.1.3 gives some inside information about relevant and more complex implementation details and patterns:

- **EmbhaveController:** The class EmbhaveController represents the central controller and is the main entry point of the API to be used in test scripts. It initializes the whole test system by parsing the configuration file and, if available, the CAN processing and mapping file. Furthermore, it holds all relevant data objects to perform test cases and retrieve necessary information.
- **MapperFileStorage:** The storage holds the mapping from keywords to CAN signals or I/Os, which can be used in the feature definition of the test case.

- **CanMatrix:** The CanMatrix imports and represents a DBC file, which can be used to specify CAN frames, which should be sent by the transceiver, or to retrieve values when receiving them. It holds a list of all frames present in the communication specification.
- **Frame:** A frame represents a single CAN message and stores all relevant data such as an identifier, the size and a list of signals.
- **Signal:** A signal is the smallest unit in the CAN communication specification. It typically represents a single value of an input, output or parameter. It holds the startbit inside the frame, the size and other value dependent information such as minimum and maximum.
- **CANController:** To define a generic format and structure for all CAN controller implementations, the class CANController is implemented as an abstract base class. It defines relevant attributes and methods to be compatible with the test execution.
- **PCANController:** The class PCANController represents one implementation for a CAN controller for the PCAN-USB hardware [40]. It contains methods for initializing and releasing the controller and to process the data for receiving and transmitting CAN messages.
- **PCANBasic:** This class holds all low level attributes and functions to ensure the functionality described in the PCANController.
- **IOController:** This class is the equivalent of the CANController for the I/O section in this implementation. It defines relevant attributes and methods for all I/O test hardware implementation.
- **AardvarkController:** The AardvarkController implements functionality to use the Aardvark USB device[41]. It specifies the initialization, reading and writing of I/Os.
- **AardvarkBasic:** Like the PCANBasic class, the AardvarkBasic class contains the low level functions used by the AardvarkController one layer above.
- **IOBoard:** The third realized abstract base class is the IOBoard specifying the structure of an IO-board representation.
- **DOBoard:** The DOBoard class is mainly used to read outputs of the ECU and compare the results with expected values specified in the test case. It holds a list of DO objects used to address a specific output on the board.
- **DO:** This class stores all relevant information of an output, e.g. name, address or default value.
- **DIBoard:** The DIBoard is similar to the DOBoard and is implemented to communicate with the analog and digital inputs of the ECU.

- **DI:** This structure is the equivalent of the DO structure for inputs.

The class diagram with all relevant attributes and methods is visualized in figure 5.2.

5.1.3 Software implementation details

Two major demands for the design and implementation are independence and compatibility. The case, where each test case has to be changed when exchanging the test hardware shall be avoided at all cost. Therefore, the implementation is constructed using a variety of layers. In order to get some insights, the following paragraphs give details about the underlying software implementation. The subsection is structured in three parts: the first part focuses on the hardware implementation among the different layers, the second part gives insights of mapping and parsing components and the last section deals with the EmbhaveController as main entry point.

The classes PCANBasic and AardvarkBasic are provided by the respective company selling the hardware and can be interpreted as hardware abstraction layer (HAL). They ensure accessibility to the hardware by specified functions. The next layers are represented by the PCANController and AardvarkController. The PCANController wraps the low level functions of PCANBasic, ensures correct parameters for hardware initialization and is responsible for the cyclic operation of the CAN transceiver for sending and receiving CAN messages in a fixed time cycle. At the same level, the AardvarkController handles the same functionality for the desired I/O functionality (threading for I/Os not needed). The thread mechanism is realized with the threading module.[42] The basic and relevant implementations for both constructs are declared in the abstract classes CANController and IOController, where the Python module abc [43] (Abstract Base Classes) is used. This decision has the purpose to guide a new implementation of a CAN or I/O controller to ensure, that the methods and objects used in the layer above are still present, as the respective hardware is initialized at run-time by the EmbhaveController dependent on the parsed configuration. The IOBoard is implemented in the same way as abstract base class, whereas the DOBoard and DIBoard represent actual implementations. DI and DO act as structures to define and represent all information of an input or output. The current implementation of *embhave* contains only the implementation of the PCAN USB adapter as CAN controller and the Aardvark USB adapter as I/O controller.

The configuration file is designed to be compatible with the Python module SafeConfigParser [44], which ensures a safe and easy function to parse the configuration file. The mapper file for CAN and I/O definitions is realized as semicolon-separated file and gets parsed and split with Python built-in functions. The DBC file parser is based on an open-source implementation of Eduard Broecker [45] and modified to meet the requirements of the current implementation. It opens the DBC file (plain text file) and decides based on regular expressions which information is relevant and how to proceed.

The EmbhaveController represents the heart of the implementation. First, the configuration file for the test setup gets parsed. Depending on the configuration values,

Figure 5.2: *Embhave* class diagram

the respective hardware objects and mapper are initialized. Afterwards, the low level functions are accessible through the EmbhaveController's high level functions, which may be used in the test implementations. Those functions are wrappers to provide the necessary configuration and to establish the communication with the hardware. The controller handles also the conversion of provided user data to a standardized format, which is passed to the low level functions (e.g. conversion from strings to numbers or from decimal to hexadecimal format). Finally, the release of all initialized objects is realized within a wrapped function.

The following enumeration states environment and software package details used for the implementation of *embhave*, whereas the test framework gets described in more detail in subsection 5.1.4. Thereby, compatibility for usage or development in the future is ensured.

- **Windows 7 64 bit** [46]: Currently the only operating system where *embhave* is tested
- **Python 2.7.11** [47]: Development environment and language
- **Behave 1.2.6** [39]: BDD test framework to define test cases and inject *embhave*
- **PyHamcrest 1.9.0** [48]: Test matcher library to provide assertions
- **Ansicon 1.66** [49]: ANSI escape sequence recognition for the Windows console to enable colored console output of test executions.
- **PCAN Basic API 4.0.3.75** [40]: PCAN-USB Interface used to perform CAN tests
- **Aardvark Basic API 5.15** [41]: Aardvark I^2C Host Adapter to perform I/O tests

5.1.4 Test framework

To evaluate the functionality of *embhave*, an actual prototype implementation of tests is realized. Therefore, an adequate BDD test framework has to be chosen in order to implement test cases. The following frameworks are taken into consideration:

- FitNesse[50]
- Robot Framework[51]
- Behave[39]

FitNesse is an integrated standalone wiki web server. It is used to specify and verify acceptance criteria. A wiki page can be executed as test to verify the test specification

against the actual application. The Robot Framework is another test framework for acceptance tests which uses the concept of keyword-driven testing.

The author chooses Behave as test framework to realize the prototype evaluation visualizing the concept and to build a basis for evaluation. It consists of feature definitions, which represent the textual description of the system's behavior. The test specification is written in a language called Gherkin, which describes the requirements by specific examples to illustrate the behavior in a specific aspect. The second part of the tests consist of step definitions, where each line of the scenario gets a concrete implementation. Behave uses Python to realize those step definitions.[52] A detailed description and visualization of the concept is shown in chapter 5.2.

The following list represents a set of arguments to use Behave as a test framework:

- The test specification is defined using Gherkin, therefore all tests are easily understandable, written in natural language and can be simultaneously as test documentation.
- As the test specifications are written in standard text files and the test implementations are Python files, all relevant test data can be versioned without troubles.
- The tests are executable from the command line and no special tool or installation is required. Additionally, they are also executable by a continuous integration server (e.g. Jenkins).
- Behave produces a unit-test-like test report, which is compatible with all major visualization, test or build environments (e.g. Jenkins).
- The standard Python implementation of tests makes injection of *embhave* functionality very easy. Furthermore, Behave includes a functionality called environmental control, which is used to define code executed before or after specific events during testing. It can be used to initialize the main controller of *embhave* or to reset the test system after a test case.

Additionally to Behave, a separate test matcher is used to improve the specification of what a test should comprise. Therefore, PyHamcrest is chosen, as it is one of the most known and used matchers available in a multitude of programming languages. It includes a wide range of predefined matchers and additionally the possibility to define own matcher objects.

5.2 Prototype test setup

In section 5.1 a basic overview of the deployment is given. This section deals with the evaluation test setup and the implemented prototype tests.
The application, which runs on the controller and serves as the system under test, is

taken from a real customer project of the cooperation partner. The entire customer project deals with a vehicle and includes among others drive control, suspension control and management of different functionality attached to the machine. The controller used for the prototype tests handles one special device mounted to the vehicle and is normally located in a CAN network communicating with five additional controllers and other hardware. To perform integration tests, the controller is isolated from the rest of the machine and added to the presented environment to test the integrated functions developed on the controller.

Figure 5.3 visualizes the applied test setup. The PWM (Pulse Width Modulation) outputs PWM_0, PWM_2 and PWM_3 are not connected to the test system, as it is not possible to perform tests using the available test hardware. Nevertheless, they are connected through load resistances (50 Ω) to ground, as the I/O driver of the controller performs security checks on inputs and outputs and the application will not perform as expected if no load is detected on the specified outputs. The digital outputs DI_5 and DI_6 are connected to the digital output test board, the digital input DI_4 and the analog input AI_0 are connected to the digital input test board. Both I/O test boards are connected to the same I^2C bus and through the Aardvark USB adapter to the remote slave (test computer in the current test setup). Both CAN lines are connected via the PCAN USB adapter to the test computer. The ECU and the test boards are supplied by 12V. The application of the ECU gets flashed by a specific tool provided by the cooperation partner.

To discuss the concept and tests cases, one feature is picked and explained in detail. An extract of more feature and step definitions is accessible in appendix A.1 and A.2. Listing 1 depicts the feature definition "CAN communication between ECUs". It consists of two scenarios, which test the different behavior depending on the state of the CAN communication. The requirement behind this test states, that a CAN controller connected to the same bus as the device under test shall signalize if it's alive or not in order to perform respective actions. The idea behind feature definitions is to describe the whole behavior of the system through features and scenarios and derive all further development and test activities from the feature definitions.

The second part called step definitions of the test implementation is shown in listing 2. It contains one respective function for each statement of the feature definition. This file is used to implement the actual tests in order to interact with the test device. Single steps definitions are reusable and have to be implemented only once.

After executing the test cases, Behave is able to create a test report in different formats. The format used for the prototype implementation of test is XML (Extensible Markup Language), as it is created in the standard unit test output format and therefore compatible with all major visualization or build tools. The test report of the previous example is attached in the appendix A.3.

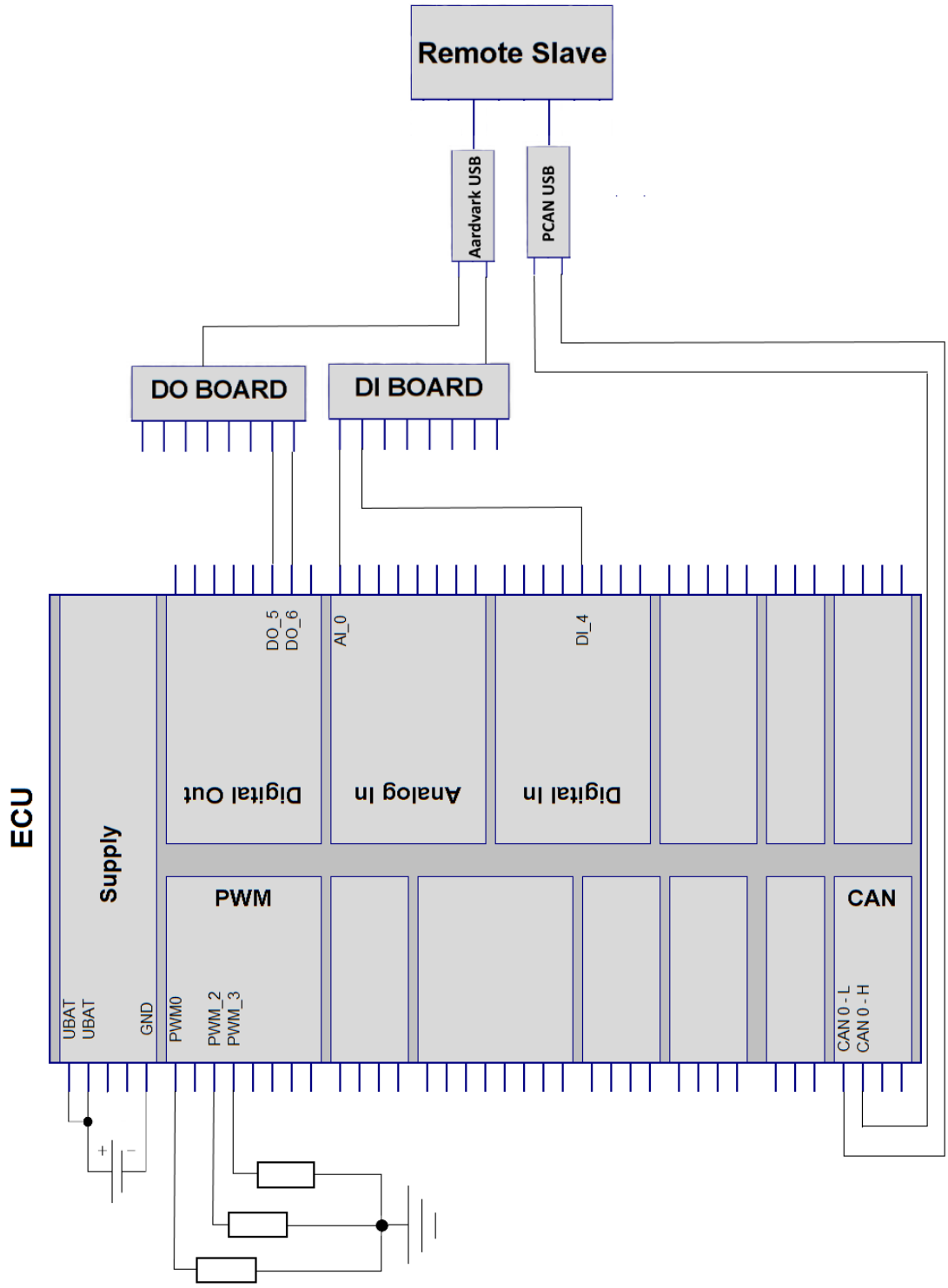

Figure 5.3: ECU I/O and test setup overview

```
# language: en

Feature: CAN communication between ECUs

    Scenario: Normal ECU CAN communication
        Given the ECU application is in operational state "IDLE"
        When the CAN communication between ECUs is "alive"
        Then the CAN signal "ECU not alive" shall be "inactive"

    Scenario: Stop ECU CAN communication
        Given the ECU application is in operational state "IDLE"
        When the CAN communication between ECUs is "interrupted"
        Then the CAN signal "ECU not alive" shall be "active"
```

Listing 1: Behave feature definition

```
from behave import *
from hamcrest import *

from embhave import *

@given(u'the ECU application is in operational state "{state}"')
def step_check_app_state(context, state):
    val = context.emb_controller.get_received_can_value(state, 0)
    if state == "IDLE":
        assert_that(val, is_(equal_to(0)))

@when(u'the CAN communication between ECUs is "{state}"')
def step_when_can_communication_alive(context, state):
    if state == "alive":
        value = 1
    elif state == "interrupted":
        value = 0
    context.emb_controller.set_can_tx(state)


@then(u'the CAN signal "{signal}" shall be "{state}"')
def step_then_signal_is_state(context, signal, state):
    if state == "active":
        value = 1
    elif state == "inactive":
        value = 0
    val = context.emb_controller.get_received_can_value(signal, value)
    assert_that(val, is_(equal_to(value)))
```

Listing 2: Behave step definition

CHAPTER 6

Evaluation

The following chapter deals with the results of the evaluation. There are two phases of evaluation applied within the master thesis: a status quo analysis 6.1 of the actual test habits regarding the application under test together with a final feedback round and semi-structured expert interviews 6.2.

6.1 Status quo analysis & feedback presentation

The status quo analysis is performed during the prephase. 4.3.1 and pursues several objectives, from which a majority are also valid for the final feedback presentation:

- **General information:** As a real customer application is used to test the implementation prototypically, the customer shall be informed.
- **Build awareness:** Testing approaches focusing on additional aspects will most likely enrich the existing test infrastructure. The work shall be presented to build awareness and demonstrate the general need in software testing.
- **Insider knowledge:** Through conversations, the author obtains inside information of real world issues and difficulties. These information shall be beard in mind during development or design decisions.
- **Feedback:** The experience of the participants is used to get feedback and recommendations.

The status-quo meeting starts with a short introduction, information about the author and an explanation about the aim of the thesis and the meeting itself. The concept is presented using a short presentation about the targeted problem, methodology and

expected results. The evaluation is performed using a developed questionnaire. As the status-quo evaluation contains almost 100% of sensible customer data, the results are not included in the master thesis but summarized in the next paragraph.

Integration tests are currently performed three times in two different phases before selling it, including manual, semi-automated and fully automated tests. On average, six employees are involved in a test cycle for approximately six hours. Monthly, three to four errors are detected before roll out and one to two errors after roll out. The participants agree, that the planned concept for the master thesis has a lot of potential. On a scale from one (best) to five (worst), they rated the concept with a score of two regarding usefulness and applicability. The possibility to simulate functional tests for integration or system tests is understood as the highest enrichment. In contrast, the additional complexity and the integration in a current test environment is described as the greatest challenge.

The final feedback presentation is opened with an introduction, some information about the author, the thesis, the presentation and the collaboration between the author, ECU Ltd. and the respective customer. Afterwards, the used methodology BDD, the developed artifact *embhave* and the gained results are presented. The meeting is closed by an open discussion including questions on how to apply the implementation in their specific field.

The concept is perceived as considerably successful, compared with the presented and planned work in the status quo presentation. The fact, that the concept gives new possibilities in functional testing of the application is again affirmed. There are still open questions on how to possibly integrate the concept within the existing test environment. Furthermore, they can imagine additional benefits during the development of the software, but have not yet designed a concrete strategy on how to increase the performance of their testing activities.

6.2 Expert interviews

This section presents the analysis results of the conducted interviews. It concentrates on the main findings and opinions of the different experts. First, the quality criteria used in the thesis are described. Afterwards, a short overview of the experts is given. Finally, the results are grouped into several major topics.

6.2.1 Quality criteria

In order to ensure qualitative research results, certain quality criteria have to be taken into account. The widely known criteria for quantitative research are objectivity, reliability and validity [53, p.80 ff]. As they were criticized for their limitations in qualitative research, the evaluation in this master thesis follows the six criteria proposed by Mayring [54, p. 140 ff]:

Documentation of procedure
Quantitative research deals with standardized techniques and methods, whereas qualitative research is more dependent on the research object. Therefore, a precise description of the research process is mandatory. Chapter 4 gives a detailed description on used methods throughout the different stages of the master thesis.

Argumentative validation of interpretation
It is not possible to prove interpretations. Nonetheless, each interpretation must be enriched with traceable arguments supporting the interpretation.

Rule-based procedure
Qualitative research must be flexible but also systematic. Therefore, the research process has to follow specific rules. The description of the rules and methods applied is available in chapter 4.

Proximity to the subject
Qualitative research aims at investigating the research subject within his/her natural and daily business environment. It is an important factor to establish an open and equal relationship. As the implementation and evaluation takes place within the environment of the cooperation partner, proximity is ensured.

Communicative validation
In order to increase validity and back up findings, results gained from the research may be presented to the participants. In the course of the present master thesis, the final results are demonstrated to the participants of the cooperation partner in the final meeting.

Triangulation
Different data sources, interpretations, theories or methods may be applied in order to gain results from several analysis inputs. As described in section 4.3.3 data from the status quo meeting, expert interviews and final result presentation is taken into consideration.

6.2.2 Expert overview

The underlying master thesis is evaluated using expert interviews as a qualitative research method. Due to the very specific domain, only a limited set of experts is available and willing to participate. In the course of this master thesis four experts are interviewed and their knowledge, opinion and feedback are integrated in the following sections. Due to time and location restrictions, the interviews are conducted within two interview sessions. The first session comprises three experts and one technical supervisor, whereas a single expert is interviewed in the second session. As the supervisor is directly involved in design decision of the developed artifact, his statements are not taken into consideration during the evaluation of the interviews.

Table 6.1 gives an overview of the experts participated in the evaluation process of the master thesis.

Expert	Working period in company	Professional experience and field of work
A1	7 years (including master thesis)	Embedded integration tester
A2	5 years as team-lead, several years before	Team-lead for the past 5 years, several years of software development experience
A3	16 years	Software development and testing experience in different fields of embedded software, as well as certification and safety related topics regarding test management.
A4	11 years	System tester of embedded software and especially drivers

Table 6.1: Expert overview

6.2.3 Results

The following paragraphs deal with the evaluation of the implemented test concept using expert interviews. For this section, three categories have been developed to accurately summarize the main results gained from the interviews and additional data sources. A detailed transcription including paraphrasing can be found in appendix A.6 and A.7.

Evaluation of the concept

All experts identify similar advantages and disadvantages regarding the proposed test concept. The human readable approach of the system requirements as part of the test implementation is considered as a big advantage by all experts. Moreover, the target field of the prototypical implemented test cases (embedded applications on ECUs) is perceived to fit well for the applied test concept, as manual and expensive test activities may be replaced by a flexible and powerful automation.
Expert A4 confirms, that the concept enables the creation of test definitions by non-technicians due to the human readable form of the test specification. This is not the case in traditional test approaches, at least in his field of operation. Regarding this fact, he clearly differentiates *Behave* from other test methods.
Additionally, the automated consistency check between test specification and test implementation is considered by expert as second major advantage next to the readable form by expert A3. He emphasizes, that it is not a check whether the test cases are valid against the actual requirements or not, but consistency between both information sources gets checked and thus, maintenance effort is substantially reduced.

Expert	E & C	General
A1	2	2
A2	2	2
A3	3	3
A4	1-2	1-2

Table 6.2: Expert ratings (1-best | 5-worst)

In parallel to the advantages of the human readable approach, one major problem is identified by the experts in this context: the mightiness of the human language. Expert A3 states, that *"you try to force a formal context to our non-formal language and it is very difficult for us humans, as it is counter-intuitive"*[section A.6, expert A3]. The used test framework *Behave* works with the principle, that all feature definitions created, follow specific rules on how the sentences may be structured and constructed. Those limitations of the human language are perceived as a challenge when defining complex test cases, as the human language is not conclusive enough to express all test cases in detail. The experts agree, that different people may interpret the same sentence of a feature definition differently and create the underlying implementation with different functionality. To prevent this potential source of error, additional functionality for the software editor may be implemented, to support the user during the specification and implementation of the tests. One proposed feature is an auto-completion of feature file sentences dependent on already existing step definitions, as it is challenging to keep the overview when the project grows and the amount of test cases increases. A second recommendation is the support of a global glossary with standard definitions of sentences with predefined step implementations. This feature can support the standardization of test definitions throughout several projects and makes the cooperation of different testers within one project easier.
One critical factor, especially in terms of scalability regarding bigger projects and test requirements, is the time component of tests. As *embhave* is developed and tested in Windows due to different dependencies and restrictions, time critical tests can not be implemented, as the accuracy on measuring values at specific points of time or exact cycle behaviors may not be guaranteed. A1 explains during the interview his experiences, faced issues and confirms, that this factor is present in all fields of embedded software testing and very complex to solve. A first proposed step is the migration to a real-time operating system, where the temporal behavior can be controlled easier.

Table 6.2 depicts the scores assigned by the experts. The row labeled with "E & C" refers to the rating regarding efficiency and costs, whereas the row "General" visualizes the general rating of the proposed test concept. Due to the size of the prototypical implementation, not all critical aspects can be evaluated by the experts, or to put it another way, the experts state, that there may arise problems when applying the

concept in bigger projects and therefore, the emitted ratings relate to the presented implementation and application.

Fusion of test documentation and implementation

The key aspect of the proposed test approach comprises the definition of test specifications in human readable language, which are part of the test implementation, get parsed by the test runner and automatically consistency checked against the test implementation during run-time. Compared to traditional test approaches, this feature is interpreted by all experts as the major advantage and should not be underestimated. Within a project, where requirements change several times, the maintenance effort of changing test documentation, test implementation and keeping both sources consistent may generate not negligible expenses. Expert A2 underlines, that at the current stage no clear disadvantages regarding this fact are identifiable for him, but this does not imply, that there are no issues.

Applicability in practice and outlook

All experts agree on the applicability of the proposed test methodology in practice. Expert A3 highlights this fact by stating *"you proved applicability with your prototypical implementation"*. Additionally, expert A2 clearly confirms, that the concept is applicable in practice from a technical perspective. It has to be mentioned, that the experts also agree on the fact, that the extent of applicability depends on the type of target project (*"The exciting challenge is to find the right type of projects. By now, I developed also several test frameworks, each with its features and especially restrictions. The extent of applicability depends on the project, the requirements and the possibilities."*[section A.6, expert A3]). The prototypical evaluation may not confirm applicability for all kind of projects and in particular sizes of projects, as the scalability has to be evaluated within a separate pilot project.

Summarized, the experts propose the further strategy:

- **Further features**: To counteract possible issues stated by the experts, further features shall be implemented, which support the user of the test concept during the creation of test specifications and implementations. An already mentioned extension would include the auto-completion inside the feature definitions of *Behave* to keep the overview of already implemented functionality within a project. Additionally, a global glossary would improve the cooperation within one project and the standardized test definitions throughout several projects.
- **Pilot project**: The developed solution shall be applied on a further pilot project. This project then has to be of certain size and complexity, comprising different types

of requirements, dependencies and test demands. Additionally, the project should not be "first-of-its-kind", as comparison to existing realizations using traditional test approaches is no longer possible [section A.6, expert A2].

- **Extensions**: As the architecture of *embhave* is constructed very dynamic and flexible, interfaces to other technologies may be added, increasing the supported functionality and possible test coverage of requirements within a certain project (e.g. Ethernet [55], LIN [56], RS232 [57]).

CHAPTER 7

Summary and outlook

The design and implementation of tests for embedded application has proven to be a difficult task. Whereas module tests may be executed similar to standard software, integration tests comprise the additional complexity of hardware and timing dependencies. The alternative to expensive test automation frameworks is the execution of manual integration tests over and over again, increasing the probability of human errors and testing efforts over time. Additionally, the maintenance effort of keeping all sources of a software development project consistent (e.g. software code, software documentation, test implementations, test documentation) may get challenging to control, especially in projects with frequently changing requirements.

The API introduced in this thesis together with the prototypical test setup represent an automated test approach for embedded applications on ECUs using an innovative test methodology. The development was realized together with a cooperation partner using a Scrum-like approach. Thus, the solution was developed incrementally, tackling specific issues faced during each sprint. All decisions of used tools are based on a previous performed analysis and evaluation during the prephase. The implementation is tested within a prototypical test setup using a real-world customer project to increase the significance of the evaluation.

The following two implemented factors, which proved to be successful for a test automation in the field of embedded applications on ECUs, are highlighted. The possibility of creating human readable test specifications, which are simultaneously executable and part of the test implementation were rated as a considerable advantage compared to traditional test approaches according to the interviewed experts in the field. Moreover, they confirmed that the additional software features of the API addressing known issues in current test methodologies like the flexibility on using different test hardware are noticeably improving the status-quo.

Possible limitations may be linked to the scalability of the test approach. During the evaluation, the question on how efficient the test methodology remains when applying it in larger projects arose. As the prototype setup and the resources of the available test hardware are limited, the evaluation on the scalability has not been taken into account within the present work.

The underlying solution of this master thesis answers both research questions addressed in section 1.1. First, the usage of the proposed test framework *Behave* may be integrated in the embedded application development cycle to specify the systems behavior. Together with the developed API *embhave*, the test implementation is able to communicate with the test hardware and to retrieve relevant information for the designed test cases. The design represents a flexible and expandable basis of test software. Second, the applicability is confirmed by several experts using an existing and productive embedded application within a prototypical test setup, for which test cases were designed, implemented and executed successfully.

As there is no other combination of both fields found in the literature, further research and development is necessary to increase the available possibilities and fully demonstrate the power of the proposed test approach. Considering the gained feedback of the performed evaluation, the test strategy shall be applied in a larger pilot project, including more requirements and different types of tests. Thereby, the scalability may be accurately evaluated and further advantages, but also limitations, which have not yet been apparent within the evaluated setting, can be revealed.

List of Figures

1.1 Possible issues during software development 2

2.1 Embedded system model 6
2.2 Generic scheme of an embedded system 7
2.3 CAN layer specification 8
2.4 CAN layer specification 9
2.5 CAN data frame structure 10
2.6 Standard vs. extended frame format 11
2.7 Example of an I2C-bus configuration using two microcontrollers 12
2.8 I2C protocol explanation 13
2.9 Verification & Validation 16
2.10 ISO/IEC/IEEE 29119-4-Test Techniques 19

3.1 TEmb overview 22
3.2 Adapted BDD flow 23

4.1 Traditional development process 30
4.2 BDD development process 30
4.3 Timeline of the development process 32
4.4 Scrum process 33

5.1 Overview of the deployment 39
5.2 *Embhave* class diagram 43
5.3 ECU I/O and test setup overview 47

List of Tables

3.1 Evaluation results verification pattern approach 25

6.1 Expert overview 54
6.2 Expert ratings (1-best | 5-worst) 55

List of Listings

1 Behave feature definition 48
2 Behave step definition 49
3 Feature definition extended 72
4 Step definition extended 74
A.1 Behave test report 75

Acronyms

BDD Behavior Driven Development. 23, 22, 23, 24, 29, 32, 44, 52, 61

BDT Behavior Driven Test. 3, 21, 29, 31

CAN Controller Area Network. 8, 9, 10, 11, 26, 37, 38, 39, 40, 41, 42, 44, 45, 46, 61

DBC CAN database. 11, 37, 38, 40, 42

ECU Electronic Control Unit. 3, 11, 26, 29, 35, 37, 41, 46, 54, 59, 61

GPL GNU General Public License. 39

HDL Hardware Description Language. 22

LIN Local Interconnect Network. 26, 57

LOC lines of code. 25

MOST Media Oriented Systems Transport. 26

NASA National Aeronautics and Space Administration. 1

PSL Property Specification Language. 24

UDS Unified Diagnostic Services. 26

XCP Universal Measurement and Calibration Protocol. 26

Bibliography

[1] Thomas Grechenig, Mario Bernhart, Roland Breiteneder, and Karin Kappel. *Softwaretechnik - Mit Fallbeispielen aus realen Entwicklungsprojekten.* Pearson Deutschland GmbH, München, 1. aufl. edition, 2010.

[2] Toyota's killer firmware: Bad design and its consequences. URL: `http://www.edn.com/design/automotive/4423428/Toyota-s-killer-firmware--Bad-design-and-its-consequences`, Accessed January 6, 2017.

[3] The economic impacts of inadequate infrastructure for software testing. URL: `http://www.edn.com/design/automotive/4423428/Toyota-s-killer-firmware--Bad-design-and-its-consequences`, Accessed January 6, 2017.

[4] Bart Broekman and Edwin Notenboom. *Testing Embedded Software.* Addison-Wesley, 2003.

[5] John Ferguson Smart. *BDD in Action.* Manning Shelter Island, 2015.

[6] Kpmg report. URL: `http://delimiter.com.au/wp-content/uploads/2012/06/KPMG_audit.pdf`, Accessed January 6, 2017.

[7] Tammy Noergaard. *Embedded Systems Architecture. A Comprehensive Guide for Engineers and Programmers.* Newnes, 2013.

[8] Jack Ganssle, Tammy Noergaard, Fred Eady, Lewin Edwards Lewin, David J. Katz, Rick Gentile, Ken Arnold, Kamal Hyder, and Bob Perrin. *Embedded Hardware.* Newnes Know It All. Newnes, 2008.

[9] Can specification version 2.0. URL: `http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can2spec.pdf`, Accessed January 6, 2017.

[10] Walther Operenyi. Experimental framework for controller area network based on a multi-processor-system-on-a-chip. 2012.

[11] Vector candb. URL: https://vector.com/vi_candb_en.html, Accessed January 6, 2017.

[12] I2c-bus specification and user manual. URL: www.nxp.com/documents/user_manual/UM10204.pdf, Accessed January 6, 2017.

[13] Aardvark i2c/spi user manual. URL: http://www.totalphase.com/support/articles/200468316, Accessed January 6, 2017.

[14] Ieee standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84, Dec 1990.

[15] Internation Software Testing Qualifications Board. Certified tester foundation level syllabus. 2011. URL: http://www.istqb.org/downloads/send/2-foundation-level-documents/3-foundation-level-syllabus-2011.html4, Accessed January 6, 2017.

[16] Glenford Myers. *The art of software testing.* John Wiley & Sons, Hoboken, N.J, 2012.

[17] Iso/iec/ieee 29119-4 - software testing - part 4: Test techniques. 2015. URL: http://www.iso.org/iso/catalogue_detail.htm?csnumber=60245, Accessed January 6, 2017.

[18] Iso/iec/ieee 29119-4: Test techniques visualization. URL: http://www.softwaretestingstandard.org/part4.php, Accessed January 6, 2017.

[19] Melanie Diepenbeck, Mathias Soeken, Daniel Große, and Rolf Drechsler. Behavior driven development for circuit design and verification. 2014. URL: http://www.informatik.uni-bremen.de/agra/doc/konf/2014_tap_behaviour_driven_development.pdf, Accessed January 6, 2017.

[20] Gherkin language. URL: https://github.com/cucumber/cucumber/wiki/Gherkin, Accessed January 6, 2017.

[21] Cindy Eisner and Dana Fisman. *A Practical Introduction to PSL (Series on Integrated Circuits and Systems).* Springer, 2006.

[22] Wei-Tek Tsai, Lian Yu, Feng Zhu, and Ray Paul. Rapid embedded system testing using verification patterns. pages 68–75, 2005. IEEE Computer Society 0740-7459/05.

[23] Byeongdo Kang, Young-Jik Kwon, and Roger Y. Lee. A design and test technique for embedded software. 2005. IEEE Computer Society 0-7695-2297-1/05.

[24] A. Guerrouat and H. Richter. A concept for testing and diagnosis of embedded systems based on extended finite states machines. 2004.

[25] Vector gmbh product portfolio. URL: https://vector.com/at_portfolio_en.html, Accessed January 6, 2017.

[26] Canoe. URL: https://vector.com/vi_canoe_en.html, Accessed January 6, 2017.

[27] Uds specification. URL: http://www.iso.org/iso/catalogue_detail.htm?csnumber=55283, Accessed January 6, 2017.

[28] Uds specification. URL: https://www.asam.net/nc/home/standards/standard-detail.html?tx_rbwbmasamstandards_pi1%255BshowUid%255D=519&start=, Accessed January 6, 2017.

[29] vteststudio. URL: https://vector.com/vi_vteststudio_en.html, Accessed January 6, 2017.

[30] Vector test hardware. URL: https://vector.com/vi_hardware_en.html, Accessed January 6, 2017.

[31] Matt Wynne and Aslak Hellesoy. *The Cucumber Book*. The Pragmatic Bookshelf, 2012.

[32] Scrum process. URL: https://www.emaze.com/@AZLZFOCI/Scrum-Workflow, Accessed January 6, 2017.

[33] Michael Meuser and Ulrike Nagel. Experteninterviews - vielfach erprobt, wenig bedacht. ein beitrag zur qualitativen methodendiskussion. *Qualitativempirische Sozialforschung: Konzepte, Methoden, Analysen*, pages 441–471, 1991.

[34] Alexandra Bettina Steiner. *Building a WMS Implementation Framework based on Critical Success Factors of ERP Implementations*.

[35] Behave environmental control. URL: https://pythonhosted.org/behave/tutorial.html#environmental-controls, Accessed January 6, 2017.

[36] Popular coding languages 2016. URL: http://blog.codeeval.com/codeevalblog/2016/2/2/most-popular-coding-languages-of-2016, Accessed January 6, 2017.

[37] Mark Lutz. *Learning Python*. O'Reilly Media Inc., 2009.

[38] Python 2.7 license. URL: https://www.python.org/download/releases/2.7/license/, Accessed January 6, 2017.

[39] Behave documentation. URL: http://pythonhosted.org/behave, Accessed January 6, 2017.

[40] Pcan usb adapter. URL: http://www.peak-system.com/PCAN-USB.199.0.html, Accessed January 6, 2017.

[41] Aardvark i2c adapter. URL: http://www.totalphase.com/products/aardvark-i2cspi/, Accessed January 6, 2017.

[42] Python 2 threading. URL: https://docs.python.org/2/library/threading.html, Accessed January 6, 2017.

[43] Python 2 abstract base classes. URL: https://docs.python.org/2/library/abc.html, Accessed January 6, 2017.

[44] Python 2 configuration parser. URL: https://docs.python.org/2/library/configparser.html, Accessed January 6, 2017.

[45] Canmatrix importer. URL: https://github.com/ebroecker/canmatrix/blob/master/LICENSE, Accessed January 6, 2017.

[46] Windows 7. URL: https://www.microsoft.com/de-de/software-download/windows7, Accessed January 6, 2017.

[47] Python 2.7. URL: https://www.python.org/download/releases/2.7/, Accessed January 6, 2017.

[48] Pyhamcrest documentation. URL: http://pyhamcrest.readthedocs.io/, Accessed January 6, 2017.

[49] Ansicon. URL: http://adoxa.altervista.org/ansicon/, Accessed January 6, 2017.

[50] Fitnesse test framework. URL: https://github.com/ebroecker/canmatrix/blob/master/LICENSE, Accessed January 6, 2017.

[51] Robot framework. URL: https://github.com/ebroecker/canmatrix/blob/master/LICENSE, Accessed January 6, 2017.

[52] Behave philosophy. URL: http://pythonhosted.org/behave/philosophy.html, Accessed January 6, 2017.

[53] Werner Berekoven, Ludwig Eckert, and Peter Ellenrieder. *Marktforschung*. Wiesbaden: Gabler, 2009.

[54] Philipp Mayring. *Einführung in die qualitative Sozialforschung*. Beltz Studium, 2002.

[55] Ieee 802.3 ethernet standards. URL: http://standards.ieee.org/about/get/802/802.3.html, Accessed January 6, 2017.

[56] Lin specification. URL: http://www.cs-group.de/fileadmin/media/Documents/LIN_Specification_Package_2.2A.pdf, Accessed January 6, 2017.

[57] Rs232 fundamentals. URL: https://www.lammertbies.nl/download/dallas-appl-83.pdf, Accessed January 6, 2017.

APPENDIX A

Appendix

A.1 Feature definitions

```
# language: en

Feature: CAN communication between ECUs

        Scenario: Normal ECU CAN communication
                Given the ECU application is in operational state "IDLE"
                When the CAN communication between ECUs is alive
                Then the CAN signal "ECU not alive" shall be "inactive"

        Scenario: Stop ECU CAN communication
                Given the ECU application is in operational state "IDLE"
                When the CAN communication from ECUs stop
                Then the CAN signal "ECU not alive" shall be "active"

Feature: Process Lights

        Scenario: T light left and right out
                Given the ECU application is in operational state "IDLE"
                When the light "R lights" is "off"
                Then the output "T light left" shall be "off"
                And the output "T light right" shall be "off"

        Scenario: Activate T light left and right
                Given the ECU application is in operational state "IDLE"
                When the light "R lights" is "on"
```

```
26                  Then the output "T light left" shall be "on"
27                  And the output "T light right" shall be "on"
28
29  Feature: LM function
30
31          Scenario: Activate LM state
32                  Given the ECU application is not in operational state "W"
33                  And the doors are closed
34                  And no drive approval is given
35                  When receiving the CAN signal "K LM"
36                  Then the ECU application shall change to state "LM"
```

Listing 3: Feature definition extended

A.2 Step definitions

```
from behave import *
from hamcrest import *

from embhave import *
from time import sleep

@given(u'the ECU application is in operational state "{state}"')
def step_check_app_state(context, state):
        val = context.emb_controller.get_received_can_value(state, 0)
        if state == "IDLE":
        assert_that(val, is_(equal_to(0)))

@when(u'the CAN communication between ECUs is alive')
def step_when_can_communication_alive(context):
        context.emb_controller.set_can_tx(1)

@when(u'the CAN communication from ECUs stop')
def step_when_can_communication_stops(context):
        context.emb_controller.set_can_tx(0)

@then(u'the CAN signal "{signal}" shall be "{state}"')
def step_then_signal_is_state(context, signal, state):
        if state == "active":
        value = 1
        elif state == "inactive":
        value = 0
        val = context.emb_controller.get_received_can_value(signal, value)
        assert_that(val, is_(equal_to(value)))

@when(u'the light "{output}" is "{state}"')
def step_when_light_output_value(context, output, state):
        if state == "off":
        value = 0
        elif state == "on":
        value = 1
        context.emb_controller.set_tx_can_signal(output, value)

@then(u'the output "{output}" shall be "{state}"')
def step_get_io_output_value(context, output, state):
        if state == "off":
        value = 0
```

```
42          elif state == "on":
43          value = 1
44          val = context.emb_controller.read_do_output(output)
45          assert_that(str(val), is_(equal_to(value)))
46
47  @given(u'the ECU application is not in operational state "{state}"')
48  def step_check_not_in_operational_state(context, state):
49          val = context.emb_controller.get_received_can_value(state, 0)
50          assert_that(val, is_(equal_to(0)))
51
52  @given(u'the doors are closed')
53  def step_doors_closed(context):
54          context.emb_controller.set_tx_can_signal("doors closed", 1)
55
56  @given(u'no drive approval is given')
57  def step_doors_closed(context):
58          context.emb_controller.set_tx_can_signal("drive approval", 0)
59
60  @when(u'receiving the CAN signal "{signal}"')
61  def step_set_tx_signal(context, signal):
62          context.emb_controller.set_tx_can_signal(signal, 1)
63
64  @then(u'the ECU application shall change to state "{state}"')
65  def step_check_state_change(context, state):
66          val = context.emb_controller.get_received_can_value(state, 1)
67          assert_that(val, is_(equal_to(1)))
```

Listing 4: Step definition extended

A.3 Test report

Listing A.1: Behave test report

```
<?xml version='1.0' encoding='UTF-8'?>
<testsuite errors="0" failures="0" name="thesis_prototype_tests.CAN communication between ECUs" skipped="0" tests="1" time="15.008">
  <testcase classname="thesis_prototype_tests.CAN communication between ECUs" name="Normal ECU CAN communication" status="passed" time="5.0">
    <system-out>
      <![CDATA[
        @scenario.begin
        Scenario: Normal ECU CAN communication
          Given the ECU application is in operational state "IDLE" ... passed in 0.000s
          When the CAN communication between ECUs is "alive" ... passed in 5.000s
          Then the CAN signal "ECU not alive" shall be "inactive" ... passed in 0.000s
        @scenario.end
        ------------------------------------------------------------
      ]]>
    </system-out>
  </testcase>
  <testcase classname="thesis_prototype_tests.CAN communication between ECUs" name="Stop ECU CAN communication" status="passed" time="5.0">
    <system-out>
      <![CDATA[
        @scenario.begin
        Scenario: Stop ECU CAN communication
          Given the ECU application is in operational state "IDLE" ... passed in 0.000s
          When the CAN communication between ECUs is "interrupted" ... passed in 5.000s
          Then the CAN signal "ECU not alive" shall be "active" ... passed in 0.000s
        @scenario.end
        ------------------------------------------------------------
      ]]>
    </system-out>
  </testcase>
</testsuite>
```

A.4 Interview guide (german)

Einführung

- Begrüßung, Danke für Zeit
- Persönliche Vorstellung
- Kurze Vorstellung der Diplomarbeit
- Ziel des Interviews: Evaluierung der Umsetzung auf Sinnhaftigkeit, Anwendbarkeit, Nutzen, Feedback von Experten
- Dauer: 90 – 120 Minuten
- Interview wird Audio - aufgezeichnet (nur um alle Informationen und Aussagen aufzuzeichnen, um sie später beim Transkribieren wieder abrufen zu können)
- Interview wird anonymisiert, transkribiert und in die Masterarbeit aufgenommen

Part 1 – Präsentation & Demo

(Aufzeichnung starten)

- Präsentation zum Problem und erarbeiteten Lösung
- Demo des erarbeiteten Konzepts
- Laufende Diskussion

Part 2 – Experten

Kurze Vorstellung der Experten:

- Name des Experten, Tätigkeitsfeld
- Wie lange arbeiten Sie bereits im Feld?
- Haben Sie Erfahrungen im Bereich *BDD* und/oder *Embedded- Testing*?

Part 3 – Evaluierungsfragen

- Sehen Sie in dieser Umsetzung Vorteile und Nutzen bzw. Nachteile verglichen mit Testansätzen, mit denen Sie arbeiten oder Erfahrung haben?

- Finden Sie die Verschmelzung von Testimplementierung und Dokumentation als vorteilhaft gegenüber traditionellen Testansätzen? Welche Vorteile sehen Sie darin? Erschließen sich daraus auch Nachteile?
- Auf einer Skala von 1 bis 5 (1-beste, 5-schlechteste): Wie würden Sie die vorliegende Lösung im Sinne der Effizienz bewerten und warum?
- Auf einer Skala von 1 bis 5 (1-beste, 5-schlechteste): Wie würden Sie die vorliegende Lösung im Sinne der Kosten bewerten und warum?
- Auf einer Skala von 1 bis 5 (1-beste, 5-schlechteste): Wie würden Sie die vorliegende Lösung im Gesamten bewerten und warum?
- Können Sie sich vorstellen, dass die erarbeitete Umsetzung in der Praxis anwendbar ist? Warum? Warum nicht? Szenarien für sinnvolle Anwendung nennen? Bei welchen Projekten wäre es sinnvoll auf diesen neuartigen Testansatz zu wechseln?

Um die Aussagekraft zu erhöhen und Vorwürfen entgegenzuwirken, dass sich Aussagen von Experten auf andere übertragen, da das Interview gemeinsam geführt wird, werden die Antworten einzelner Fragen von jedem Experten eigenständig auf einem Blatt Papier vermerkt.

Part 4 - Weitere Fragen & Abschluss

- Empfehlungen für zukünftige Arbeiten in diesem Bereich?
- Fragen der Teilnehmer?
- Danksagung an Teilnehmer für Zeit und Feedback

(Aufnahme stoppen)

A.5 Interview guide (english)

Welcome & introduction

- Welcome, thanking for the time
- Personnel introduction
- Short description of master thesis
- Aim of the interview: Evaluation of the implementation regarding reasonableness, applicability, benefit + feedback of expert
- Duration: 90 – 120 minutes
- Interview will be audio recorded (just for the transcription)
- Interview will get anonymised and added to the master thesis

Part 1 – Presentation & demo

(Start recording)

- Presentation of problem statement and elaborated solution
- Live demo of the concept
- Ongoing discussion

Part 2 – Experts

Short introduction of the expert:

- Name and field of work
- How long does the expert work on the field?
- Has the expert experience with BDD or embedded testing?

Part 3 – Evaluation

- Do you see advantages and benefits, disadvantages of the concept compared to known test methods?
- Are there advantages in the fusion of test implementation and documentation compared to traditional test methods? Which advantages? May there be disadvantages as well?

- On a scale of 1 (best) to 5 (worst): How would you rate the solution in terms of efficiency and why?
- On a scale of 1 (best) to 5 (worst): How would you rate the solution in terms of costs and why?
- On a scale of 1 (best) to 5 (worst): How would you rate the solution in general and why?
- Can you image, that the elaborated solution is applied in practice? Why, why not? Scenarios for useful application? Which project may be good to switch to this test concept?

To increase significance and counteract accusations, that the statements of experts may be transferred as the interview takes place together, specific questions get answered using pen and paper.

Part 4 - Additional questions & conclusion

- Recommendations for further work in the area?Empfehlungen für zukünftige Arbeiten in diesem Bereich?
- Questions of experts?Fragen der Teilnehmer?
- Thanksgiving for time and feedback

(Stop recording)

A.6 Transcription experts interview 1

A.6.1 Transcription

Audiofile 1: #0:00

[Einleitung und Motivation des Themas]

[Erklärung BDD und *embhave*]

#7:11

A3: Entschuldigung, Zwischenfrage: DBC- Files, das sind diese CAN... das ist die CAN-Database von Vector?

E: Genau, ganz genau.

[Erklärung Konfiguration und Aufbau]

#12:30

A3: Eine Frage, hast du eigentlich dort nur die Anbindung an den USB und den *Aardvark* oder auch an die ECU auch direkt aus dem Framework heraus?

E: Nein, das Framework ist eigentlich gezielt auf Integrationstests für Applikationen, sprich der Sinn sollte sein ich schicke CAN– Nachrichten oder ich setze Eingänge an der ECU oder Steuergerät und erwarte mir dann folgende Reaktion vom Steuergerät, indem ich wieder einen Pin zurück lese oder eine Message die von der ECU dann kommt einlese und schaue ob der Wert so gesetzt wurde wie ich es mir erwarte nach diesem Testfall.

A1: Das heißt du hast keine Möglichkeit die Software auf der ECU selber zu steuern, nicht zu steuern sondern zu...

A3: ...auch zu "monitoren"...

A1: ... zu konfigurieren zum Beispiel?

E: Nein.

S1: In der momentanen Version ist das nicht im Scope, zukünftig ist die Idee eine Anbindung auch an zum Beispiel unsere DLL zu haben für den Downloader, dass du im Testfall schreiben kannst die Version soll drinnen sein.

E: Genau.

A1: OK.

S1: Zukunftsmusik wäre im Testfall zu definieren wie die Applikation ausschaut und dann was zu übersetzen das dann rüber-geflashed wird. Das ist das Ende der Fahnenstange, dort ist es noch recht nebelig.

A3: Ist die Frage ob ich das beim Integrationstest dann noch darf...

S1: Eh nicht, nein nein, da geht es dann um Modultests, nicht nur um Integrationstests.

[Erklärung Konzeption]

#14:52

A3: Für Modultests bräuchtest du aber dann natürlich ein Interface auch zu... um interne Zustände von der ECU zu kommen?

E: Genau, ja. Das ist jetzt nicht unbedingt... wie schon gesagt der primäre Scope, aber es wäre durchaus möglich dann dahingehend zu erweitern.

#15.20

[Erklärung Prototyp-Testfälle]

#16:40

A3: Das heißt die sind jetzt implementiert aber mit Python – Decorator?

E: Bitte?

A3: Das ist ja implementiert mit Python – Decorator?

E: Genau, diese Decorator kommen von *Behave*, das ist das Testframework, das ich verwende um die Tests durchzuführen. Ich könnte noch das... Environment- File zeigen.

[Zeigt Environment- File]

[Erklärung Prototyp-Tests und LIVE Demo]

#20:30

A1: Das ist jetzt nur über den CAN Bus, oder?

E: Das waren jetzt... das ist CAN und I/O.

A1: OK, du sagst ihm er soll die einschalten und per I/O liest du zurück ob er wirklich ein ist?

E: Genau, das kommt eben je nach Implementierung an wie wird was überprüft und welche Möglichkeiten habe ich zu testen was passiert ist. In dem Fall ist es... dieser Usecase ist wenn an der Maschine die Rücklichter eingeschaltet werden dann sollen die Lichter an der WW[Teil der Maschine] auch eingeschaltet werden und die Information, ob die Rücklichter eingeschaltet wurden kommt eben vom CAN und die Lichter an der WW[Teil der Maschine] sind digitale Ausgänge, die ich dann hier zurück lesen kann.

A2: Initialisiert er vor jedem Test das Testsystem zum gleichen Zustand?

E: Genau. Es ist gerade so implementiert, dass ich vom CAN– File... das CAN- File lese ich ein und im CAN- File kann ich Defaultwerte hinterlegen, also welcher Wert ist im Standardfall in diesem Signal drin. Das gleiche mache ich auch mit den I/Os – wenn ich das Mapping- File einlese habe ich die Möglichkeit einen Wert anzugeben der als Defaultwert gilt und nach jedem Test wird das System sei es auf I/O als auch auf CAN Basis geresettet und werden die Defaultwerte wieder eingeschrieben, dass ich

wieder auf der gleichen Basis bin wie vor dem Test. Das Problem, das ich jetzt auch während den Tests herausgefunden habe ist, dass das Resetten – überhaupt im Sinne von Integrationstests – nicht immer sehr einfach ist, das heißt in der finalen Version muss ich da noch ein besseres Konzept finden, weil ich habe zwar kein Problem wieder Defaultwerte reinzuschreiben aber wenn wir uns zum Beispiel den letzten Testfall anschauen: Da geht es darum, dass wenn auf der CAN Tastatur LM drücke, dann wird das auf der ECU realisiert, dass er den Klick bekommt – eine logische 1 – und wartet dann wieder auf die Edge, dass es fällt und dann wird der Klick simuliert. Das heißt wenn ich wie hier – hier hab ich es eh ausgeschrieben – wenn ich eine 1 sende und dann wieder eine 0 sende, das bedeutet für ihn einen Klick, dann wechselt er in den Status. Jetzt müsste ich aber um zu resetten.. reicht es nicht aus nur die 0 zu senden sondern ich müsste wieder eine 1 und eine 0 senden um den Status wieder zu resetten. Das heißt, da bedarf es auf jeden Fall noch einer besseren Reset- und Default-Strategie um zu vermeiden, dass sich der User selbst darum kümmern muss, dass das System eben wieder geresettet wird. Was natürlich auch möglich wäre, was in diesem Aufbau nicht möglich war, ich könnte die Versorgung von der ECU steuern und könnte ein Shutdown und Restart der ECU machen – dann wäre ich auch wieder im selben Status wie davor sozusagen.

A2: Ich denke jetzt zum Beispiel so Dinge wie OS das wird ja kein Signal sein das OS das ist ja irgendein interner Status von der ECU vermutlich oder was ist das Z1 da?

E: Nein, das ist ein Signal, da geht es um den Zustand der Maschine. Also sie haben intern verschiedene States um zu sagen, am Anfang am Startup wenn keine Funktion eingeschalten ist, ist es eben Z1, dann wenn ich eine Funktion einschalten will dann wechselt es in den Status Z2 oder Z3 – je nachdem welche Funktion ich getätigt habe und.. genau, da geht es weniger um den internen Status der ECU sondern mehr um die Applikation und diese Informationen – hier zum Beispiel übermitteln sie die über CAN, weil im richtigen Setup der Maschine ist diese ECU nur eine von fünf Steuergeräten plus Display, das heißt dort werden die Informationen über CAN ausgetauscht um am Display auch zu visualisieren in welchem Status befindet sich die ECU gerade und gibt es einen Fehler oder sonstiges. A2: Also um was es mir eigentlich geht – sind die Testszenarien Reihenfolge unabhängig?

E: Ja, also...

A1: Vorausgesetzt du kannst es resetten dazwischen.

E: Genau, wenn der Resetmechanismus zu 100% funktioniert – ich sag jetzt mal das ist ein Beispiel wo er nicht funktioniert, bei allen anderen funktioniert er eh – wenn der dann so implementiert ist, dass es bei jedem funktioniert, dann kann man die in der Reihenfolge austauschen wie man will. Logisch es kommt auch ein bisschen drauf an wie die Tests geschrieben werden, auf die ich ja dann keinen Einfluss habe, also wenn der User in den Tests Zustände verändert und Zustände so verändert die das Resetten nicht möglich machen, dann ist es natürlich nicht mehr so, dass die Tests ausgeführt werden in der Reihenfolge wie man will aber ja, das ist dann auch ein bisschen Eigenverantwortung.

S1: Das Konzept ist eindeutig so, dass die Testfälle nicht voneinander abhängen sollen

und dass du in dem "Given" herrichten musst was du brauchst und das zieht auch Kreise, dass heißt sinnvollerweise - Stichwort "Design for Testability" – versuchst du die Applikation dann so zu haben, dass du sie eventuell sogar über Mechanismen, die nicht zur User-Funktionalität gehören sondern reine Testmethoden sind, dass du gewisse Ausgangszustände herstellen kannst ohne dich durch eine State-Maschine mit 47 Vorzuständen durch musst.

A1: Genau, das ist ja prinzipiell das Problem glaube ich bei... wenn das Device Under Test stateful ist.

S1: Genau, in dem Augenblick wo du eine State- Maschine hast wäre es echt hilfreich wenn du über das "Given"... und das ist ein wesentlicher Punkt: Das "Given" soll normalerweise auch keine Schritte enthalten die eigentliche Benutzeraktionen sind sondern soll den Ausgangszustand des Testfalls beschreiben und die Implementierung die dahinter liegt soll das idealerweise auch nicht über die Benutzermethoden versuchen zu erreichen, also eben nicht durch 10 Test... 10 Vorzustände glackern, weil du sonst nämlich mit der Wartbarkeit von den Testfällen sofort in des Teufels Küche landest, weil wenn sich irgendwas in der Funktionalität davor ändert, in der State- Maschine an den Bedingungen für die Übergänge, bist du tot, weil du änderst nur noch Testfälle. Wenn du eine Testmethode hast mit der du sagen kannst: "Given ich bin in dem und dem Zustand" und das ruft am Steuergerät was auf das nur in der Debugversion vorhanden ist wo du in den Zustand kommst, dann ist alles fein und das ist die Idee, die hinter dem Konzept steckt.

#28:02

A3: Das heißt für einen neuen Test heißt das neuer *C – Code*?

S1: Nein, neuer Test nur dann wenn du neue... Neuland in der Funktionalität betrittst. Wenn du neue Variationen von schon existierendem Wortschatz brauchst... also im Normalfall brauchst du ein Binding das dir den Wortschatz abdeckt und Testfälle die die Varianten von der Benutzung abdecken und das ist üblicherweise der größere Teil und auch der Wartungsaufwand. Also nicht jeder Testfall sollte neues Binding brauchen, weil wir haben jetzt zum Beispiel Binding das die gesamte Hardware ansprechen kann unter diesen Aliasnamen, unter diesen Mapping- Namen und auch die ganzen CAN Signale und ab jetzt kannst du Testfälle schreiben ohne dass du eine einzige Zeile Python Code schreiben musst.

A1: Vielleicht kommt das ja noch aber was für Möglichkeiten hast du die Zeit miteinzubeziehen?

E: Das ist der zweite Punkt, den ich gerade ansprechen wollte, also die zweite Herausforderung auf die ich gestoßen bin. Der Vorteil sag ich mal an der klassischen Variante wie normalerweise getestet wird, sprich mit zwei Steuergeräten ist, dass sie zwar nicht synchronisiert werden, aber trotzdem in derselben Laufzeit konfiguriert werden können um dann mehr oder weniger sicherstellen zu können, wenn das eine jetzt 50 Zyklen eine Botschaft sendet dann empfängt sie der andere auch relativ genau 50 mal. Das ist bei mir nicht so einfach möglich – ich habe Messungen gemacht wo ich gesehen habe, dass

mein Rechner um CAN Signale zu senden ungefähr alle 50 Millisekunden... nein Blödsinn alle 5 Millisekunden die Messages sendet, das Steuergerät – jetzt dieses Steuergerät ist konfiguriert auf 10 ms, das heißt theoretisch sende ich jede Botschaft zweimal bis er, oder habe ich jede Botschaft zweimal gesendet bis es das Steuergerät einmal empfangen hat. Ich arbeite für das CAN Senden – Empfangen mit einem Timer- Thread und was recht schön ist, das *PCAN* bietet eine Funktionalität mit Windows Events, das heißt ich bekomme eine Art "Call" wenn eine CAN Botschaft empfangen wurde und die jetzt zum Lesen ist, das heißt ich kann, wenn ich weiß dass ich ungefähr 50 ms brauche und das messen kann, dann kann ich die Funktion so gestalten, dass ich auf die restliche Zeit, also nach der Konfiguration des Steuergeräts warte um nicht nochmal alle Botschaften zu senden, aber ganz lösen konnte ich es bis jetzt noch nicht, dass ich einen sicheren Zustand gewährleisten kann, dass es synchron abläuft. Das ist eben eine noch Herausforderung, die in der finalen Version umgesetzt werden soll.

A3: Das ist aber ein generelles Problem, weil weder Windows noch Linux echtzeitfähig sind und daher keine Garantie... ich meine mit dem Problem haben wir ja schon oft genug selbst gekämpft und solange du auf der Windowsseite das Timing steuern musst wirst du es nie 100% hinbekommen.

A1: Eben, es kommt ja total auf das Device Under Test an in welchen Größenordnungen dass du dich hier bewegst, ja, wenn du jetzt sagst so 50, 100ms und so dann sage ich mal ja das kriegst du irgendwie hin und selbst wenn du Linux Realtime patchen musst oder so, dann geht das "easy cheesy" sage ich jetzt mal von der Resolution. Ich meine, meine Frage ist, wir haben Testcases gehabt wo wir auf μs – Basis genau sein müssen. Da geht es um so "safety-reaction-time" und so Sachen und das heißt... du musst dann eben... genau das ist eben dieser Knackpunkt, deswegen haben wir oft das Problem gehabt wir können es nicht mehr automatisieren, weil wir kein Equipment haben, das auf μs genaue Events erzeugt, oder Timestamps oder auch ausgeben lässt. Ich will ja eigentlich wenn ich so Tests mache, I/Os setze, also simple digitale I/Os, dann will ich zu bestimmten Zeitpunkten den Ausgang setzen, μs genau – im Idealfall ns genau sage ich jetzt mal, damit sogar die... damit ich nicht genau die Periode erwische von unserem Runtimesystem, das auf dem Ding läuft und drübersweepen kann, das was ich eh immer mach sozusagen, und dann vergleiche ich dazu wie lange hat es gedauert bis es rauskommt und das muss ich aber 10.000 mal machen oder wie auch immer. Wir helfen uns jetzt zumindest zum Beispiel, dass wir einen Frequenzgenerator haben der asynchron ist, also der... kein periodisches Verhältnis hat zur eigenen Frequenz der CPU und dann vergleichen wir mit dem Oszilloskop indem wir auf unendlich über zeichnen lassen und dann schauen wir wo die Grenzen sind. Ist ein simpler Trick, funktioniert, ist OK, aber genau das ist zum Beispiel ein so ein Knackpunkt wo bis jetzt immer... wo ich bis jetzt immer gescheitert bin da irgendwas zu finden, weil das du hier machst und das gleiche was S1 auch vor hat das ist nur simpel I/O, da kann ich sagen EIN - AUS aber ich habe null Garantien wann das passiert. Wenn ein Testcase fehlschlägt dann muss ich nachschauen warum und dann gibt es eine Chance dass er fehlgeschlagen ist weil das Timing nicht gepasst hat und dann muss man sich in der Theorie halt ob es eh

nicht passieren kann dass ein "false positive" rauskommt nur weil das Timing... sobald ich im Testcase drin stehen habe es muss in 50 μs dies und jenes passieren, oder 50,1, das haben wir auch manchmal, dann hab ich wieder eine Klasse von Testcases die nicht automatisierbar sind.

#34:00

E: Wie ich es gesehen habe oder wie ich es auch traditionell mit zwei Steuergeräten implementiert habe – was sie manchmal machen ist, dass sie die Zeit in Zyklen ausdrücken und schauen dann, OK, das soll 200 Zyklen bleiben und dann beim 201. Zyklus soll die Fehlermeldung gesetzt werden, dann lesen sie 200 mal die Botschaft ein, oder nicht 200 mal sondern bei 10ms 20 mal, lesen die Botschaft 20 mal ein, erwarten dass 20 mal das gleiche drin steht und dann beim 21. mal der Fehler gesetzt wird. Das ist nicht ideal, und eben wenn es auf sehr sicherheits- und timekritische Fälle geht auch nicht zufriedenstellend, aber was ich bis jetzt mitbekommen habe für einen Großteil der Fälle in der Applikationsentwicklung eine zufriedenstellende Lösung.

A1: Es kommt immer darauf an für was du es verwendest. Wenn ich jetzt TDD mache und Regressionstests damit machen will, zum Beispiel, was ich zum Beispiel super finden würde genau so was, dann reicht das für die meisten Fälle. Die CERT wird es nicht schaffen, also ich kann keine Tests machen für CERT, dann müsste es ja kalibriert sein...

S1: Nicht alle...

A1: Genau, natürlich...

S1: Der Scope von der Sache ist so ein Steuergerät mit 5 oder 10 ms Zykluszeit und das reagiert auf die äußere Welt und die Frage ist reagiert es richtig. Wir haben bei unseren Steuergeräten auch PWM Ausgänge wo du rumkonfigurieren kannst und wo sich Flanken dann irgendwie überlappen oder nicht, das kann man damit nicht testen...

A1: Nein, das ist mir voll bewusst.

S1: ...aber die Logik die sich da so im 10ms Rhythmus abspielt, für die als Regressionstest... für die soll das geeignet sein, das ist quasi der gedachte Einsatzzweck für das Tool.

#36:22

E: Das war die Präsentation um was es geht – falls es noch Fragen zur Umsetzung oder zum allgemeinen Konzept gibt wäre jetzt ein guter Zeitpunkt und danach würden wir fortfahren mit meinen Fragen bezüglich des Konzepts und eurer Bewertung dazu.

A1: Ich hätte gleiche eine Frage: Wie machst du das mit den digitalen I/Os? Hängt das auch über den BUS dran oder wie steuerst du das von deinem Maincontroller oder was auch immer du da hast?

E: Genau, also im Maincontroller habe ich einmal einen...

A1: Der Maincontroller ist ein normaler PC.. ist der PC auf dem das drauf läuft... oder Jenkins Slave..

E: Genau, der Maincontroller ist eine Python Applikation sozusagen, der initialisiert mir einen CAN Controller und einen I/O Controller, in meinem Fall eben *PCAN* und *Aardvark* und in den Config- Files...

A1: Egal, du steuerst mit dem Bussystemen deine digitalen I/O.

S1: Also I^2C.

E: Genau, das ist jetzt I^2C, da gebe ich eine Adresse an von den Boards...

A1: Na aber ich versteh schon...

E: ...und habe ein internes Mapping welcher Pin welche Adresse hat und spezifiziere im Config- File.. die Maschine, also das wäre jetzt zum Beispiel der Eingang wo gemessen wird ob die Maschine verankert ist, und den spezifiziere ich dann "Maschine Verankerung" ist dieses Board auf diesem Pin und jedesmal, wenn ich mit dem was tun will sucht sich der Contoller raus, OK, auf welchem Pin willst du jetzt was machen...

S1: Und die Idee war diesen untersten Layer recht schlank zu halten, das heißt wenn es andere Hardware gibt dann tauscht man unten einfach eine Schicht aus und du kannst sie auch mischen, also es darf dann nicht immer nur dieses oder jenes sein sondern dass du die verschiedensten Pfade hast auf die konkrete Hardware über eben in dem Fall den I^2C und über den *PCAN*...

E: Genau, und das war eben auch die Idee vom Maincontroller, dass alle Aufrufe der Tests über den Maincontroller laufen und wenn sich jetzt diese Hardware ändert ich in meinen Tests nicht auch die Aufrufe ändern muss, sondern sich der Maincontroller je nach Konfiguration dann die unterliegenden Schichten ... genau, "auslayered" und da dann weitergearbeitet werden kann.

S1: Also du im Idealfall im Testfall eigentlich gar nicht siehst ob es über den CAN rausgeht oder ob es ein HW Signal ist, sondern du dich auf die Domainsichtweise konzentrieren kannst, das Signal heißt so und wenn das Signal kommt soll das passieren.

E: Gibt es sonst noch Fragen oder...

A2: Ich glaube mal nicht.

E: Vielleicht kommen ja noch welche [schmunzelt].

Audiofile 2: #00:00

A3: [Wir hatten in einem Projekt schon mal eine ähnliche Vorgehensweise] ... da auch in Python geschrieben ist und dort hatten wir eigentlich sozusagen den umgekehrten Ansatz, dass wir Dokumentation und Testimplementierung dadurch konsistent gehalten haben indem wir die Dokumentation aus der Implementierung generiert haben, also genau das Gegenteil wie es du machst. Das haben wir aber dann später aufgegeben, weil wir gesagt haben für die Dokumentation haben wir *Integrity* und die Testimplementierung ist unabhängig davon, das heißt wir haben ja leider schon diesen Bruch, dass wir aktuell diese Möglichkeit gar nicht haben. Wir hätten sie... wir haben sie früher schon gehabt,

haben sie auch genutzt, haben wirklich die PDF Dokumente dann aus den Testscripts generiert, war aber eben genau der umgekehrte Ansatz, also aus der... auch aus dem Grund heraus weil die menschliche Sprache ist nicht eindeutig und nicht starr. Die im Gegensatz dazu die Programmierung und mathematische Sprache ist exakt. Das heißt in dieser Richtung umgekehrt Text zu generieren ist deutlich... relativ einfach. IN der Form... ich muss sagen ich habe diesen Ansatz nicht gekannt, ich würde mir jetzt am Anfang zumindest sehr schwer, ich weiß nicht wie lange es dauert sich an diese Variante zu gewöhnen. Speziell, es wird dann dort schwer wenn es anfängt komplex zu werden. 50% der Tests glaube ich auf so eine Art dass man die damit machen kann, die restlichen 50% werden wahrscheinlich schwierig weil du dort sehr spezifisch in Dinge eingreifen musst und es potentiell sehr komplex werden kann und da tut sich diese textuelle Beschreibung relativ schwer, könnte ich mir vorstellen.

E: Jein, also meines Erachtens, wenn der Testfall in dieser Art und Weise relativ komplex wird, dann würde er es sonst auch werden..

A3: Ja..

E: ... also, die Definition vom Test ist immer gleich komplex. Hier geht es mehr um eine Umschreibung zum Text, weil was ich auch gemacht habe ist, ich habe Integrationstests geschrieben und habe dann im *Integrity* eben geschrieben:

1. Sende die Botschaft "so und so" mit diesen Werten für 50 Zyklen
2. Miss ob das diesen Wert hat
3. Sende es noch ein weiteres Mal
4. Versichere, dass jetzt der Fehler gesendet wurde

A3: Ja, aber die Gefahr besteht dann, dass man für jeden Testcase genau eine neue... eine neue solche Funktionalität benötigt, einen neuen Satz brauchen. Ab einem gewissen Punkt passiert das dann, und das macht das Ganze dann schwierig, weil welche von diesen... weil dann ist es schwierig für den Tester herauszufinden, welche von diesen Sätzen gibt... welche von diesen Sätzen kann ich überhaupt formulieren.

S1: Also den Wortschatz..

A3: Sozusagen den Wortschatz, weil es werden ja diese Sätze, werden ja praktisch in Funktionsnamen übersetzt wenn ich das richtig gesehen habe, das heißt die Sätze müssen auch immer genau dieser Syntax entsprechen. Das ist einerseits für einen Menschen relativ schwierig, weil der Mensch einfach nicht so denkt, weil ich automatisch diese Dinge anders formulieren würde, und ich bin aber gezwungen genau diese Syntax zu verwenden.

A1: Was ja aber nichts Schlechtes ist.

A3: Es ist nicht Schlechtes, ja, wir haben damals den umgekehrten Weg deswegen gemacht um eben genau dieses Problem zu umgehen, weil in der Testcasebeschreibung: jeder

Tester schreibt es anders. Und wir hatten dann genau den Ansatz, dass sich unsere einzelnen Testklassen dann sozusagen ihre eigene Dokumentation selbst geschrieben haben in einer standardisierten Form. Das war unser Ansatz so was auch zu produzieren, nur eben genau den anderen Weg rum.

A2: Es könnte dann ja auch sein, dass durch die Syntax oder genau eigentlich durch die Satzbildung, weil "receiving a CAN signal" ist dann eine Funktion zum Beispiel, mit Parameter?

E: Ja, genau.

A2: ... dass, durch die Satzbildung ich mich irgendwann ziemlich einschränken werde wenn es viel wird, da ich keine Phrasierung finden werde, die dann noch vernünftig lesbar sein wird, sodass die auf eine eigene Funktion geht, weil eigentlich muss ich ja wissen wenn ich den Satz schreibe, dass dahinter eine eigenen Funktion ist.

E: Aber die ist ja noch nicht da...

A2: Die wird generiert?

E: Nein, die muss man machen, aber das macht dann ja im besten Fall der, der es auch schreibt, das heißt er kann sich ja selber überlegen...

A2: Aber man muss wissen, wenn man den Satz formuliert, was man in diesen Satz hineininterpretiert, weil das ist ja das dann was in diese Funktion rein programmiert werden muss. Wenn ich jetzt da 5 habe oder 10 oder vielleicht 20 solche Sätze, dann ist das überschaubar. Wahrscheinlich wenn wir in einem Bereich von 500.000 sind, wird es vermutlich Verwechslungen geben könnte ich mir vorstellen relativ schnell.

E: Verwechslungen von den einzelnen Sätzen jetzt, oder von..?

A2: Nein, dass ein Satz, die Formulierung, möglicherweise in einer kurzen Form doch zu wenig aussagt um zwei sehr ähnliche Testfälle unterscheiden zu können. Das geht auch in die Richtung was A3 sagt mit der Ausdrucksweise in der Prosasprache. Im Prinzip find ich es cool weil man es so schön lesen kann, ich frage mich nur wie es bei richtig großen Projekten ausschaut.

S1: In erster Linie würde verglichen mit dem was wir da sehen die Menge der verfügbaren Signalnamen ansteigen ganz massiv... das ist jetzt halt nur ein Ausschnitt. Der Syntax, der da jetzt mal da ist und sich auf CAN Signale lesen schreiben, auf echte Signale lesen schreiben konzentriert, würde wahrscheinlich wenn man das jetzt für ein nächstes Projekt portiert, nicht extrem ansteigen, weil es geht ja genau darum, dass du beschreibst "diese Signale sollen.. wenn das am Steuergerät anliegt die Reaktionen vom Steuergerät bewirken"..

A2: Schau, du wirst einen Tester vielleicht haben der schreibt: "When receiving the signal S1 with value 1". Dann gibt es vielleicht einen nächsten Testcase der den value noch ändern will, das macht ein anderer Tester der schreibt aber "When received the signal...", dann hast du automatisch im Hintergrund zwei Funktionen.

S1: Das ist eh klar, aber wenn du es dann startest läuft es nicht durch, weil das Binding fehlt und dann muss man sich überlegen ob da jetzt wirklich was fehlt oder ob man nur was anderes formuliert hat was es schon gibt. Also genau genommen ist die eigentlich Theorie die dahinter steckt, man könnte das alles in Python programmieren und du bräuchtest diese Feature- Files nicht. Dann hast du allerdings diesen in dem Fall technologisch geschaffenen Medienbruch zwischen "wie soll es sich verhalten" und "was heißt das jetzt eigentlich für die Pins vom Steuergerät" nicht mehr so scharf mit Feature-File und Binding, sondern dann passiert das was in 9 von 10 Projekten – also ich habe es zumindest bis jetzt so gesehen – dann hast du auf einmal Testfälle wo in das "was soll passieren" immer mehr Implementierungsdetails hinein diffundieren und die Testfälle werden unverständlicher. Das Konzept sorgt einfach nur dafür, dass zwei Dinge getrennt bleiben die nicht zusammen gehören.

A1: Für mich ist es einfach eine extrem hohe Programmiersprache eigentlich, die einfach das Featureset reduziert, wenn es... genau, aber das setzt voraus, dass es möglich ist.

S1: Und ein bisschen Schmerzen erzeugt, jedes Mal wenn du das Featureset oder den Sprachumfang von dem oberen, von diesem Feature- Files erweitern willst, weil dann musst du unten was dazu programmieren musst, und das so quasi Hintergedanken...

A1: ... du willst es vermeiden, wenn es geht – und das finde ich auch an und für sich gut. Also für das, was ich bis jetzt gemacht habe finde ich 80% ja, und 20% sind eben doch zu implementieren, und die machen aber 90% der Arbeit aus.

E: Sicher, ja, aber der Punkt, den ihr auch angesprochen habt kann in vielen Fällen eben auch ein Vorteil sein, denn wenn ich einmal definiere was der Satz "receiving CAN Signal PARAMETER with value PARAMETER" bedeutet, dann kann ich den theoretisch in jedem Szenario wiederverwenden und muss das Binding nur einmal definieren.

A1: Für mich ist das wie eine Funktion. Also ich würde jetzt... also es liest sich schön, aber für mich als Techniker ist es fast... ich würde mich trauen zu wetten es ist fast noch einfacher zu lesen wenn "Funktionsaufruf, erster Parameter, zweiter Parameter".

A3: Das ist auch in meiner Sicht die große Gefahr, weil wir sind nicht gewohnt, dass unsere Sprache exakte Informationen liefert, sondern im Kontext interpretiert wird, und auch mehrere unterschiedlich implementierte Varianten genau dieselbe Aussage liefern. Das ist genau das Problem, was ich hier befürchte, nämlich dass es dann am Schluss... ich mein, wenn man jetzt wie A2 gesagt hat, wenn man 20 Tests hat, ist das sicher etwas was gut maintainbar ist. Wenn ich 500 oder 1000 Tests hab mit 10 verschiedenen Testern werde ich in das Problem laufen, dass "when receiving the CAN signal" dreimal unterschiedlich interpretiert wird, und umgekehrt, dass es drei verschiedene Varianten gibt, die ähnlich klingen oder vielleicht genau das gleiche machen. Diese Gefahr besteht definitiv.

E: Aber besteht die Gefahr nicht bei traditionellen Ansätzen auch, wenn jetzt 3 Tester dasselbe machen und dann umsetzen?

A3: Nein, und zwar deswegen nicht, weil diese Sprache, wir sind gewohnt, dass die eben nicht deterministisch ist, dass die nicht exakt ist, wir sprechen hier nicht von einer formalen Sprache. Du versuchst hier unserer nicht formalen Sprache einen formalen Kontext aufzuzwingen und das ist etwas mit dem wir uns als Menschen sehr schwer tun weil es kontraintuitiv ist. Und deswegen sind zum Beispiel die klassischen Programmiersprachen... da sind wir gewohnt, dass die genau nach einer bestimmten Syntax eben exakt formal spezifiziert sind, und das sind wir konkret als Softwareentwickler gewohnt. Wir wissen genau... und deswegen ist es uns oft auch einfacher... wenn ich mir die Funktion, als Beispiel... ich habe die Spezifikation einer Funktion in Form von Requirements, es ist in den meisten Fällen einfacher ich schaue mir den Sourcecode an um zu verstehen was die Funktion tut, als ich schaue mir diese Requirements an. Weil wir diese formale Sprache intus haben, weil wir in dieser formalen Sprache auf einen Blick sehen, was dort passiert, während die menschliche Sprache hier in der Hinsicht zwar deutlich geschwätziger ist auf der einen Seite, auf der anderen Seite – man braucht deutlich länger um eine halbe Seite zu lesen als wenn ich mir die Funktion im Sourcecode anschaue. Das ist ein Problem, auf das wir immer wieder gestoßen sind.

A1: Ich habe beim V-Projekt gefühlte 30 Requirements gehabt von 500 oder so insgesamt, wo du drei Leute gefragt hast und drei Leute was anderes verstanden haben. Das Problem ist in der Macht der natürlichen Sprache.

E: Ist sicher so, aber...

A2: Also, was man da eigentlich brauchen würde als Tester vermutlicherweise, du brauchst für diese Sätze – Teilsätze – brauchst du in irgendeiner Form eindeutige Repräsentationen, die dann zu einer programmierbaren Funktion führt. Also eigentlich brauchst du für diese Teilsätze eine Beschreibung, die eindeutig ist – irgendwann musst du es programmieren und ins Detail gehen – die beschreibt, "receiving the CAN signal", die für diese Funktion genau beschreibt was diese tun muss. Das ist auch das was A3 sagt, diese Übersetzung hast du normalerweise, indem du direkt den Code anschaust. Also du kommst über die Funktionsrequirements nicht hinweg, weil nicht jeder unter "receiving the CAN signal" das gleiche versteht.

A3: Man findet vielleicht im Laufe des Projekts dann 10 verschiedene Funktionen, die alle im Prinzip CAN- Frames empfangen oder ein Signal aber sich leicht unterscheiden.

S1: Nein, das müsstest du probieren einzudämmen, ich kenne das von früher aus meinem vorherigen Leben, von Projekten wo mehrere Tester so etwas schreiben, da gibt es dann sinnvollerweise ein Glossary wo sozusagen zum Wortschatz beschrieben steht, was die Phrase kann und wenn du einen neuen Testfall schreibst, schaust du in dem Glossary nach ob du was findest was du brauchst.

A2: Ja, das wirst du machen müssen, das brauchst du...

E: Genau, und das ist auch der Sinn wieso es den Template- Generator..

A3: Das Glossary müsste aber in der GUI implementiert sein, dass wenn du drauf fährst, dass du dann schon das Glossary siehst.

S1: Im Prinzip funktioniert das, hast du es kurz irgendwo zum herzeigen? Wenn du anfängst zum Tippen, dann kriegst du eine Auswahlliste, was schon da ist...

E: Ja, genau, das ist hier vom Editor...

A1: Das ist ein grundsätzliches Problem, du musst irgendwann von der natürlichen Sprache in die Requirements – wo das passiert, da ist halt der Zwischenschritt.

A2: Ja, das Autocomplete sagt dir ja nur was es schon gibt, aber es sagt dir nicht was da eigentlich dahinter steht, wie du es rein technisch interpretierst.

E: Aber deswegen ist auch... das ist auch ein Sinn des Template- Generators, der eben Standardfunktionen wie CAN Signal lesen, CAN Signal empfangen, Ausgang lesen, Eingang schreiben... mit dem Generator wird dieses File generiert mit den Aufrufen auf den Controller und schon definiert in einem Satz, der logisch verändert werden kann, wenn es dem User zu wenig oder zu aussagekräftig ist, aber schon mal standardisiert eine Formulierung für diesen Satz definiert und auf den Punkt zurückzukommen, nur damit ich es auch richtig verstehe: deine Sorge (A3) ist, wenn drei verschiedene Tester den Satz "receiving the CAN signal" lesen, dass die Ausformulierung dieser Funktion hier dreimal anders sein würde?

A3: Potentiell, beziehungsweise, dass es vielleicht verschieden Varianten gibt.

A1: Aber wieso soll es verschiedene Varianten geben, du kannst ja nur einmal definieren was das heißt.

A2: Ja, das stimmt schon, das ist schon richtig, du kannst es nur einmal definieren, sonst kriegst du einen Bindingfehler.

A3: Das mit dem Namen kann ich nur einmal definieren... nein, in Python nicht einmal das, da kann ich es theoretisch redefinieren, aber ich kann jetzt genauso dieselbe Funktion unter fünf verschiedenen Namen definieren, weil einfach jeder Tester ein bisschen anders spricht. E: Aber dann würden die Tests ja nicht durchlaufen, wenn die Implementierung dann darunter anders ist...

A1: Aber Moment, wenn ich zehn leicht andere habe, das fallt dir doch auf. Natürlich, wenn ich ein Projekt mit 500 Testern habe, dann muss man sich sowieso was einfallen lassen, aber wenn du zu dritt bist – du liest ja andere...

A2: Zum Beispiel wie du es hier hast, der eine sagt vielleicht "receiving the CAN signal", der nächste sagt "receive the CAN signal" und der dritte sagt "received"...

A1: Ja aber das ist ja das gleiche File, da schaue ich doch zuerst mal nach, ob was da ist, das fällt dir doch beim Review vom Testcase auf...

A2: Das muss ja nicht im selben File stehen, das kann ja vielleicht 300 Seiten weiter hinten stehen.

A1: Ja, aber das...

A2: Ich glaube eben nicht, dass es bei so einfachen Formulierungen bleiben wird in der wirklichen Welt...

A3: Nein, glaube ich eigentlich nicht aus meinen Erfahrungen nach, das funktioniert eben für 50% der Fälle, wo du mit derartig einfachen Konstrukten auskommst, wenn du jetzt schon mal... und du wirst Subvarianten kriegen, du wirst dann haben ein "receiving the CAN signal" mit irgendwas zusätzlich, oder was weiß ich in redundanter Form mit Auswahl, mit irgendwelcher Signalglättung, oder was auch immer, genau dasselbe bei den I/Os. Das heißt du wirst verschiedene Varianten bekommen, und wenn du verschiedene Varianten hast und du musst jetzt eine von denen auswählen, schaust du im Endeffekt zuerst in den Code, weil du nicht weißt, welche ist genau die, die für dich am besten ist. Und dort wo ich auch noch ein großes Problem sehe ist, wie handhabst du Komplexität, wie handhabst du Verschachtelung konkret?

E: Verschachtelung von?

A3: Verschachtelung von Statements. In einer Programmiersprache ist das Standardumfang, in Form von Text gibt es das nicht. Normaler Prosatext hat keine Strukturierung, deswegen sage ich 50% der Tests kann man damit abdecken, aber bei den restlichen 50% läuft man in genau solche Probleme.

E: Jein, aber ich habe hier ja insofern Kontrollstrukturen, indem ich spezielle Schlüsselwörter habe, wie diese Texte aufgebaut sind, das heißt ich kann... wenn ich jetzt einen Status definieren müsste, also wechsle jetzt in... oder ich bin im Status jetzt nicht S1 sondern S2, dann kann ich im Binding definieren – das ist jetzt wieder eine Frage wo das oder ob es auf der ECU, was natürlich das Beste wäre, wenn ich ein Signal schicken könnte in den Status zu wechseln als Debugbotschaft, oder auch wenn ich es im Binding selbst machen muss, kann ich ja die Schritte ausdefinieren, was es bedeutet in den Zustand zu wechseln. Der Fokus der Feature- Files, oder ein Fokus der *BDD* auch verfolgt ist die Kommunikation von den Testcases, also eigentlich vom Behavior. Man könnte mit dem ausformulierten Feature- File, welches in menschenlesbarer Sprache geschrieben ist, zum Kunden gehen und könnte dem zeigen: Das sind die Testfälle, die wir definiert haben, decken die die Funktionalität ab oder gibt es noch Fälle die nicht abgedeckt sind? Wenn man das mit einem *C*- File macht, indem Tests definiert sind, dann werden sich viele Kunden nicht in der Lage sehen das entscheiden zu können. In diesem Fall hat man eben auch eine Möglichkeit Testfälle und –abdeckung zu kommunizieren. Das ist einer der weiteren Punkte auf die diese Art von Tests zielt.

A3: Wie kommunizierst du die Testabdeckung?

S1: Hier noch gar nicht.

A3: Ja, das ist mir nämlich auch aufgefallen [schmunzelt]. Nein, aber wie würde das ausschauen, weil der bisherige Ansatz ist der, dass wir Testabdeckung über die Traceability definieren, wobei die vollständige Testabdeckung kannst du automatisiert gar nicht darstellen. Das musst du immer manuell checken, das heißt diesen Aspekt

kannst du genauso wenig handeln wie wir automatisiert, aber im Endeffekt brauchst du in irgendeiner Form eine Traceability zwischen Test und Requirement.

S1: Also die eigentliche – auch noch nicht wirklich morgen sondern für die Zukunft gedachte Anwendung sind sehr wohl CERT Projekte, keine Kernkraftwerke, aber SIL2 zum Beispiel, und die Idee die dahinter steht ist das, was da oben im Feature steht im Requirementsdokument landet per Import im *Integrity*, das was da unten im Szenario steht in der Testfallspezifikation landet und die Traces sind dadurch definiert, dass es im selben File steht.

A3: Das ist ein interessanter Ansatz, weil der macht im Prinzip das V – Modell von der anderen Seite auf.

S1, E: Genau...

A3: Das heißt, mit diesem Bottom-Up – Ansatz kriegst du dann die Probleme auf der andern Seite vom V...

S1: Naja, nachdem das obere Ende vom V sowieso abstrakt ist und schlussendlich zu abstrakt um falsch zu sein, während aber der Code und die Tests unten ausführbar operativ das Richtige tun müssen...

A3: Was heißt das Richtige? Das Richtige kann ich ja nur durch die Highlevel– Requirements spezifizieren...

S1: Ja, aber da sind wir ja wo du zuerst warst, die Highlevel– Requirements werden von fünf Leuten unterschiedlich interpretiert...

A3: Wenn das der Fall ist, dann habe ich definitiv ein Problem mit meinen Requirements, aber ich kann nicht sagen ich implementiere jetzt irgendwas und aus dem mache ich mir dann meine Highlevel- Requirements. So klingt es jetzt für mich...

A1: Das passt auch in 50, 60, 70% aller Projekte... [lacht]

A3: Du machst teilweise Top-Down, teilweise Bottom-Up...

S1: Die Idee, die dahinter steckt ist, dass du da die einzelnen Verhaltensaspekte vom System sowohl lesbar als auch automatisch ausführbar beschreibst, das heißt das Ende vom Lied ist quasi eine in Prosa geschriebene Dokumentation, wo jede Zeile rot wird wenn sie nicht mehr zutriffst.

#21:35

[Gespräch über unternehmensinterne Details]

#22:38

A2: Wenn ich mir das jetzt hier ansehe im Wesentlichen, diese "Given, And, When, And..." könnte man ja umformulieren auf "Given F1 Parameter XY AND F2..."... letztendlich baut es ja auf so was auf – wie mächtig ist dieser Sprachsatz, weil das schaut ja recht einfach aus: "Given, And, When, And..." gibt es da auch irgendein "Case" oder wie mächtig ist das?

S1: Nein, Kontrollstrukturen gibt es gar keine. Es gibt die Möglichkeit Tabellen dazu zu definieren, um einen logischen Testfall zu schreiben der mit 10 konkreten Wertesätzen durchlaufen soll. Du musst nicht hinschreiben "SIGNAL ist 1", sondern du kannst Platzhalter verwenden, die in einer Tabelle mit konkreten Werten belegt sind.

A3: Die Gefahr, die ich sehe ist, dass wir nichts anderes machen als... wenn ich das jetzt schon in so einer Notation schreibe, wie wenn ich es programmieren würde, dann ist die Frage, wieso schreibe ich es nicht gleich in einer Programmiersprache, die auch wirklich klar eindeutig ist. Für mich klingt das ein bisschen... wenn ich jetzt schon derartige logische Strukturen wie "When, Then, And" definiere, im Prinzip kann ich jede Zeile in einen Funktionsaufruf übersetzen...

E: Das wird es dann ja eh...

A3: Die Frage ist für mich dann, wieso macht dieser Zwischenschritt, der mir zusätzliche Probleme verursacht... warum mach ich den dann, warum verwende ich dann einfach nur sehr sprechende Funktionsnamen und schreibe das nicht gleich direkt als Pythoncode. Dann habe ich genauso einen sprechenden Code.

S1: Weil du das im CERT Dokument verwenden kannst, 1:1 so wie es hier seht...

E: Und es automatisch gegen gechecked wird...

A3: Es gibt keinen Check. Du generierst das eine aus dem anderen, das ist kein Check. Du sicherst zwar, dass es konsistent ist, aber kein Check, du stellst nur die Konsistenz sicher.

E: Ja, Check auf Konsistenz meine ich. Also nicht ob es richtig implementiert ist, das kann ich nicht überprüfen.

A3: Ja.

E: Was S1 gemeint hat, es gibt Tabellen mit Platzhaltern, in welcher man relativ schnell eine größere Anzahl an Tests definieren kann, die zum Beispiel nur mit verschiedenen Parametern aufgerufen werden. Das könnte man dazu verwenden um die Funktion auf verschiedene Eingabeparameter zu überprüfen.

A3: Richtig, du simulierst damit eine FOR-Schleife?

E: Jein, eine FOR-Schleife des Testaufrufs. Wenn diese Tests geparsed werden, wird für jede Zeile dieser Tabelle ein Testcase definiert, und im Hintergrund wäre es dann so wie wenn dieser Testcase viermal kopiert wäre mit den entsprechenden Values.

S1: Aber Kontrollstrukturen hat der Syntax ganz absichtlich keine, weil die Idee, die dahinter steckt ist quasi um eine Abstraktionshöhe über den Sourcecode zu kommen und das gesamte System zu beschreiben durch eine Summe von "wenn ich in dem Zustand bin und ich mache das, dann soll das passieren" und davon ganz viele beschreiben das System abstrakter wie der Sourcecode, sozusagen die unterste noch nicht Sourcecodeebene runtergebrochen von den Requirements. Das ist die Granularität von dem, das geht auf Features vom Gesamtsystem, also nicht auf einzelne Layer wie ein Unittest, sondern

"wenn ich mit dem Gesamtsystem das tu, dann soll das die Reaktion sein" und Behavior-Driven-Design deswegen, weil dort wo es herkommt die Uridee, die ist, dass du zuerst das Verhalten definierst in so einem Syntax, wo es das Binding noch gar nicht gibt und dann gemeinsam Binding und Code implementierst und schaust ob es zusammenpasst.

E: Es ist ja auch ein Test Driven Ansatz..

S1: Also Test Driven nicht auf Unittest- Ebene sondern auch Systemfeature- Ebene.

Audiofile 3:#0:00

[Gespräch über unternehmensinterne Details]

#0:55

E: Das wäre sicher eine notwendige Maßnahme um das wirklich zu validieren. Ich muss logisch auch schauen, dass ich ungefähr im Scope von meinem Diplomarbeitsausmaß bleibe. Was dann wahrscheinlich eh passieren wird ist, dass wir dieses Konzept in B für ein neues Applikationsprojekt ausprobieren werden und dann kann man wirklich sehen, wo die Grenzen sind, inwieweit man das einsetzen kann und für welche Tests. Da müssen wir auch schauen wie wir die genannten Herausforderungen, wie das zeitliche Verhalten, noch in den Griff bekommen und eine richtige Aussage kann man dann nach einer längeren Probephase im Real-Life bewerten. Hier und heute geht es mehr darum, die Stimmung zum allgemeinen Konzept festzuhalten.

A1: Ich glaube für das was die ECU Ltd. macht ist das Konzept sehr "geil", für das was wir bei V1[Projekt] machen bin ich mir nicht sicher, weil die Anforderungen an das Steuerungssystem ein bisschen anders sind. Da haben wir nicht nur I/Os und Safety-I/Os, die ihr auch habt, sondern da kommen die ganzen Integrationstests dazu von den einzelnen Modulen die darauf laufen und die testen wir implizit mit. Damit werden dann die eigentlichen I/O – Tests reduziert auf ein kleines Set von dem Gesamtumfang.

A2: Ja, das stimmt. Auf diesem Signallevel ist nicht der Hauptfokus, es gibt noch viele andere integrative Aspekte.

A1: Im V1[Projekt] – Fall ist es so, dass die eigentliche Steuerungsapplikation nicht wir schreiben sondern der Kunde. Der nimmt sein Projekt und kompiliert das dann. Die machen genauso was, die machen das mit einem 20 Jahre alten Steuerungssystem, welches schon im Einsatz ist einfach umgedreht, also Input und Output verdreht, aber die verlassen sich darauf, dass die Fehlererkennung funktioniert. Die machen nicht mehr die Tests wo sie einzelne Komponenten simulieren. Genau das kann ich mit dem auch wieder machen, ich kann mit dem jetzt das machen was bei V1 – Fall der Kunde macht. Würde vielleicht einen Teil abdecken, aber für uns ist es immer die Frage des Komplexitätsproblems. Es könnte vielleicht einen Teil vereinfachen, aber der Overhead, der dadurch erzeugt wird zu maintainen und es zu verstehen ist dann immer die Frage ob es sich dafür steht. Wir haben sehr wenig automatisiert, weil der Unterschied zwischen den Tests so extrem ist. Das eine sind pure Netzwerktests", das andere in Kombination mit Reviews von Konfigurations- Files und dann noch I/O – Tests, da wäre dann der Automatisierungsaufwand einfach so hoch.

A2: Was auch noch dazu kommt ist, dass wir sehr viele Netzwerktests haben. Da gibt es dann ein ganzes Set aus Boxen mit Switches und Netzwerk, da müsstest du dann bei dem "Given" eine komplette Netzwerkbeschreibung machen und dann bei dem "When"... so ein Szenario würde dann vermutlich sehr aufwendig werden.

S1: Was aber nichts daran ändert, dass es genau die Szenarien gibt und ihr Tests dafür braucht.

A2: Schon richtig, man müsste probieren, wie das dann aussieht.

S1: Das gleiche Problem für P1[Projekt], also Pistenraupen, das ist jetzt ein Steuergerät für einen Teilaspekt, nämlich die WW[Teil der Maschine], welche auf der Maschine montiert ist. Wir sind genauso dran interessiert mit diesem Framework Tests von Systemen abzubilden, wo 3 oder 4 Steuergeräte miteinander verbunden sind und die Kombination dieser Steuergeräte als ein Device Under Test zu betrachten. Auch nicht nur Testfälle zu haben "wenn ich einen Eingang anlege, dann erwarte ich mir bei verbundenem CAN, dass ein Ausgang was macht", sondern ich auf ein Steuergerät Software flashen kann, die dann über das Netz an ein anderes übertragen wird und ich will schauen, ob dieser Flash-Vorgang, eigentlich nicht Teil der eigentlichen Funktionalität, sondern Features die mit dem Boot-Up des gesamten Netzwerks zu tun hat, auch automatisieren können.

#5:49

[Gespräch über anderes Projekt im Unternehmen]

#8:10

E: Somit würde ich jetzt zu den Fragen kommen. Eine kurze Anmerkung dazu: die Fragen zielen auf eure Einschätzung zum Konzept im Fokus des Applikationsdevelopment – also wie schon gesagt ist dieses Konzept nicht gedacht für high-secure- oder Safety-Projekte und würde euch bitten diesen Kontext im Hinterkopf zu bewahren. Ich würde gerne beginnen mit einer kurzen Vorstellrunde mit Name, Tätigkeitsfeld, wie lange ihr dort bereits arbeitet und ob es Erfahrungen im Bereich BDD oder Embedded Testing gibt.

A1: Zu diesem Konzept oder überhaupt?

E: Nein, im Allgemein. Sozusagen, dass ich danach definieren kann, die Menschen, die ich gefragt habe sind dort tätig und es nachvollziehbar ist.

A1: Mein Name ist A1, ich bin seit 5 Jahren Integrations-Tester im Projekt V1, aber das wirst du dann eh rausstreichen [lacht]. Da handelt es sich um ein Distributed Control System, unsere Testabdeckung sieht aus, dass wir Unit-Tests machen, die werden aber bei [Zulieferer] gemacht und wir machen Integrationstests um auch die Software abzudecken.

[Erläutert noch weitere Details zu seiner Tätigkeit im Projekt]

#11:17

A1: Das mache ich jetzt seit 5 Jahren, 6 bzw. 7 Jahren mit der Diplomarbeit.

A2: Mein Name ist A2, ich bin Teamlead für Softwareentwicklung seit 5 Jahren, vorher war ich selbst in der Entwicklung aber nie speziell im Testbereich.

A3: Mein Name ist A3, ich bin Softwareentwickler, seit 16 Jahren unter anderem auch im Testing- Bereich tätig. Ich glaube ich werde nicht genau sagen was ich schon alles gemacht habe, weil dann sind wir morgen noch nicht fertig, jedoch bin ich hier im Haus seit Beginn mit Testkonzepten vor allem für Zertifizierungstests beschäftigt. Ich habe allerdings auch aus der Zeit davor Erfahrungen als Integrationstester und die letzten Jahre bin ich in fast allen unseren Domains tätig gewesen. Deshalb bin ich seit Beginn der Firma in den Testaktivitäten involviert.

A1: Außerdem bist du eine der Safety- Instanzen.

A3: Ja, das ist die letzten Jahre massiv gestiegen.

E: Vielen Dank.

[Kurze Erläuterung zur Vorgehensweise]

E: Sehen Sie in dieser Umsetzung Vorteile und Nutzen verglichen mit Testansätzen, mit denen Sie arbeiten oder Erfahrung haben?

A3: Ich nehme an du möchtest mehr als ja oder nein hören?

E: Ja, vielleicht kurz! [schmunzelt].

A3: OK, fangen wir bei den Vorteilen an. Der Vorteil ist sicher die human-readable Form und die Konsistenz zwischen Implementierung und Dokumentation. Das sehe ich jetzt mal als größten Vorteil, vielleicht fallen mir dann noch andere ein. Bei den Nachteilen habe ich Angst, dass die Skalierbarkeit nicht sehr gut ist, speziell in Richtung Modultests, weil dort die Sonderfälle und Komplexität der Testcases viel größer werden. Das zweite ist, dass es schon einiges an Disziplin braucht, wenn es mehrere Tester bzw. sehr viele Testcases gibt. Hier wäre es gut, wenn es zusätzliche Unterstützung in der GUI gibt, in Form von Helptexten, die die Semantik der Sätze bekäme wäre sicher eine große Hilfe...

E: Sozusagen den Funktionsheader von der entsprechenden Funktion..

A3: Ja, Funktionsheader oder kurze Beschreibung was hinter einem "addressing the object" steht. Ich denke, dass je höher der Testlevel desto eher ist dieser Ansatz durchführbar bzw. ein umso größerer Prozentsatz der Tests sind implementierbar, weil diese im Verhältnis immer simpler werden. Bei High-level / Systemtests kann ich mir vorstellen, dass man 50%-80%, eher 80% der Tests erreichen kann, bei Low-level – Tests vielleicht nur 50% oder noch weniger.

A2: Ich sehe in der Lesbarkeit des Ansatzes den größten Vorteil. Nachteile kann ich noch nicht bewerten, ich habe nur die Befürchtung, dass bei komplexeren Problemen sich möglicherweise die Grenzen auftun werden, das müsste man in einem großen Projekt ausprobieren, auf welche Probleme stößt man und wie löst man das dann. Bei kleinem Projekten wie hier wird es vermutlich keinen Nachteil geben, aber schwierig zu sagen.

A1: Ich glaube es eignet sich sehr gut für Regressionstest und vor allem Steuerungstest wie die Ecu Ltd. macht soweit ich das beurteilen kann. Auch für fertige Anwendungen auf dem Controller, genau die Beispiele die du gebracht hast, funktionieren sicher gut. Meine Bedenken sind, dadurch dass die Timedomäne fehlt und auch die Möglichkeit zum Beschreiben beschränkt ist, was aber Sinn der Sache ist, dass du ein kleineres Subset aller Möglichkeiten aussuchst, aber das nicht ausreicht und man wieder andere Frameworks oder Testmethoden anwenden muss um den Rest abzudecken. Solange dieses Modell nicht 80% aller Testfälle abdeckt, ist jetzt eine Grenze, kann auch 70% oder 90% sein, wird es in der Praxis nicht verwendet werden oder für mich wäre es ein großes Hindernis zu verwenden, weil der Overhead da ist um es zu maintainen und es in alle Prozesse einzufügen.

E: Gleich zur nächsten Frage, die wurde glaube ich eh schon beantwortet, aber trotzdem nochmal: Ist die Verschmelzung von Testimplementierung und Dokumentation vorteilhaft bzw. gibt es auch Nachteile?

A1: Von dieser Sichtweise sehe ich hauptsächlich Vorteile. Der Nachteil wird sein, dass man durch die limitierte Sprache Probleme hat Requirements abzudecken, dass die Sprache nicht mächtig genug ist um das abzudecken, denn genau das ist das Grundproblem zwischen der natürlichen und formaler Sprache und das ist eigentlich eh schon formal. Es versucht die Verbindung herzustellen zwischen natürlicher und formaler Sprache und die Requirements, die ich bis jetzt gesehen habe, wird man jetzt Hausnummer 50%, 60%, 70%, 80% abdecken können und für den Rest muss man sich was anderes einfallen lassen. Dann wird der Maintananceoverhead, bis man alles abdeckt so groß wird, dass der Vorteil verloren geht.

A2: Kurz gesagt ist der Ansatz im Prinzip gut, die Verschmelzung hat aber auch sicher ihre Grenzen.

A3: Verschmelzung von Dokumentation und Testimplementierung definitiv ja, die spannende Frage ist eher welcher Weg dafür der Beste ist. Man muss das Ganze einerseits in die eigene Prozesslandschaft integrieren - gerade ist bei uns eine Trennung zwischen der Dokumentation und der Implementierung, was das Ganze massiv erschwert. Die andere Frage ist, welche die beste Darstellungsforum ist, die beides erreicht. Ist es die wie hier gezeigte human-readable Form, praktisch normaler Prosatext, der aber so eingeschränkt ist, dass er quasi formal ist oder eine Variante, dass man das Ganze in einer formalen Notation darstellt, die für die entsprechenden Fachleute genauso gut lesbar ist, aber den Vorteil hat bereits eine formale und exakte Notation zu haben. Das kann eine wirkliche Sprache sein, das kann Mathematik sein, das kann eine Programmiersprache sein, das hängt vermutlich vom Kontext, vom Anwendungsfeld ab und auch wer sind die Anwender, wer sind die Personen, die diese Definitionen lesen bzw. schreiben müssen. Hier gibt es glaube ich keinen goldenen Weg, keinen Königsweg, der immer der Richtige ist, das muss man von Fall zu Fall entscheiden.

E: Danke. Jetzt hätte ich eine Frage, die am besten am Papier beantwortet wird. Auf einer Skala von 1 bis 5 sollen verschiedene Kriterien bewertet werden zur vorliegenden

Lösung, wobei 1 das beste, 5 das schlechteste und 3 neutral ist.

[Kurze Diskussion zu Fragebögen]

#23:42

E: Die erste Frage ist die vorliegende Lösung im Sinne von Testeffizienz zu bewerten. Noch zum Verständnis, jede Frage richtet sich nicht nur der Implementierung, sondern auch dem ganzen Drumherum wie Wartung oder Dokumentation. Ist die vorliegende Lösung im Sinne der Testeffizienz besser, schlechter oder gleich gut als traditionelle Testansätze in der Domäne bewerten?

[Beantworten am Papier]

E: Man kann gern auch ein Kommentar dazuschreiben falls man es für notwendig hält.

#25:47

E: Nächste Frage bezieht sich auf die Kosten, 1 weniger, 5 mehr Kosten oder 3 gleiche Kosten entstehen?

[Beantworten am Papier]

A1, A3: Am Ende misst sich die Effizienz an den Kosten.

#26:41

E: Jetzt würde ich noch das Konzept im Gesamten bewerten lassen, ist es besser, schlechter oder gleich gut im Kontext und im Vergleich zu klassischen Testansätzen?

A3: Wie würdest du diese Frage in einem Wort ausdrücken?

E: Ich glaube am besten Gesamtansatz oder Gesamtlösung.

[Beantworten am Papier]

#29:02

E: Die letzte Frage zielt auf die Meinung, ob die erarbeitete Lösung in der Praxis anwendbar ist?

A2: Ich sage mal so, ist eindeutig mit Ja zu beantworten, du hast nicht dazu gesagt ob es das billigste sein muss [schmunzelt]. Technisch ist es sicher anwendbar.

E: Ja [grinst], aber die Anwendbarkeit misst man schon auch mit den Kriterien, ob es die Kosten ins Unendliche erhöht oder es keinen Unterschied macht ob man diese Art Tests verwendet oder nicht.

A1: Für mich kommt es ganz klar darauf an welches Projekt es ist..

A3: ...und welche Art von Tests ich damit abdecken will.

A1: Genau, welche Art ist für mich relativ klar, nämlich alle die man so formulieren kann und alles andere nicht. Ob dann die Requirements so geschrieben sind, dass sie ausführbar sind ist dann eine andere Frage.

A3: Das ist nicht nur eine Frage der Requirements sondern auch der Implementierbarkeit.

A1: Natürlich, und auch ein Kundenproblem. Wenn ich jetzt dem Kunden aufdiktiere wie er Requirements schreibt... ich bin ja schon froh wenn er überhaupt welche schreibt.

A3: Wobei wir aber selten gegen die Requirements vom Kunden testen...

A1: Natürlich, wir schreiben unsere eigenen, genau, das stimmt schon.

A3: Von meiner Seite Anwendbarkeit würde ich sagen JA, du hast es ja auch gezeigt, dass es prinzipiell machbar ist. Die spannende Frage ist die Nische zu finden, wo diese Tests funktionieren. Ich habe mittlerweile auch schon einige Frameworks geschrieben, die haben alle ihre Features und vor allem ihre Grenzen, wo sie sinnvoll einsetzbar sind und wo sie keinen Sinn mehr machen, und diese Nische muss man finden. Das hängt vom Projekt, von den Requirements und auch von den Möglichkeiten ab.

[Kurze Diskussion über aktuelle Ansätze und Optimierungspotentiale]

E: Abschließend möchte ich noch fragen, ob ihr Empfehlungen und Erweiterungen seht, in welche Richtung man entwickeln könnte?

A1: Für mich wie ich schon gesagt habe ist die Time-Domain, das muss jetzt nicht unbedingt in Richtung kalibrierte Instrumente gehen, das wirst du wahrscheinlich nicht schaffen. Auch Interfaces erweitern in Richtung Ethernet, aber dann wird es extrem komplex. Aber die wichtigen Anwendungsfälle ergeben sich dann aus der Praxis, die dauert aber sicher über Jahre.

A3: Mein Feedback wäre das Konzept in einem Projekt einzusetzen das nicht zu klein ist und anschließend sich wirklich Zeit nehmen zu evaluieren, nicht nur am Ende sondern auch während der Durchführung. Die Tester sollten den Fokus setzen zu evaluieren, wo Zeit während des Testens verloren geht, wo ist Komplexität und wo sind Schwierigkeiten, weil nur dann hat man den Input um zu wissen wo es hingehen soll. Das Problem hatten wir früher immer wieder, dass es schwierig war Feedback zu bekommen wo die Knackpunkte sind. Denn nur dann weiß ich, an welchen Stellen ich ansetzen muss und dann auch keine Hemmungen zu haben radikale Schnitte zu machen im Sinne von "das hat nicht funktioniert, wir machen es jetzt ganz anders". Aber für das muss man sich dann auch die Zeit nehmen.

A2: Günstig wäre es ein Projekt zu wählen, das nicht "first of its kind" ist, sondern dass man was Ähnliches hat, mit dem man es dann vergleichen kann.

E: Wenn es keine weiteren Fragen gibt bedanke ich mich für eure Zeit und euer Feedback.

A1, A2, A3: Vielen Dank auch an dich.

[Abschließende Worte]

Ende

A.6.2 Paraphrasing expert A1

Profession
A1 works for five years as full time employee (in total seven years) as embedded integration tester. His focus lies on one specific project, where he designs and performs integration tests of the software part.

Advantages of concept
He thinks the concept is very applicable in the field of regression or controller tests, as it is important for ECU Ltd. Furthermore, integration tests for applications on the controller seem to work well.

Disadvantages of concept
As the time domain cannot be taken into account within the current test setup, this missing feature is currently the biggest disadvantage.

Advantages documentation
The fusion of implementation and documentation is perceived as advantage.

Disadvantages documentation
A possible disadvantage could be the limited language set to describe the requirements. Therefore, the maintenance overhead gets bigger, as additional tests have to be designed.

Efficiency & costs
On a scale from one (best) to five (worst), he ranks the concept with a score of two regarding efficiency and costs.

General rating
On a scale from one (best) to five (worst), he ranks the general concept and implementation with a score of two.

Application in practice
He thinks that the concept is applicable in practice for a certain set of projects and for requirements, which can be formulated as executable test documentation.

Optimization
He sees optimization potential in terms of the time domain (time sensitive tests) and to integrate additional interfaces (e.g. Ethernet).

A.6.3 Paraphrasing expert A2

Profession
A2 holds the position of software team-lead for five years now. Prior to that, he worked as embedded software developer.

Advantages of concept
He sees the readability of the test cases as biggest advantage of the whole concept.

Disadvantages of concept
It is difficult to evaluate the disadvantages in this early stage, but there are probably limits when applying the concept in big customer projects with a variety of requirements. He thinks that there are probably no disadvantages when using it in small projects.

Advantages documentation
In general it is a great feature of the concept for him.

Disadvantages documentation
As stated above, the real limits are not clear for him right now, but get visible when using it in bigger projects.

Efficiency & costs
On a scale from one (best) to five (worst), he ranks the concept with a score of two regarding efficiency and costs.

General rating
On a scale from one (best) to five (worst), he ranks the general concept and implementation with a score of two.

Application in practice
Technically definitely applicable for him.

Optimization
He sees optimization potential in terms of the time domain (time sensitive tests) and to integrate additional interfaces (e.g. Ethernet).

A.6.4 Paraphrasing expert A3

Profession
A3 works as software developer and tester for the past 16 years. He was involved in several test conceptualization projects within the company from the beginning and knows the domain very well.

Advantages of concept
Two big advantages of the concept are the human readable form of test cases and the consistency of test documentation and implementation.

Disadvantages of concept
He sees potential disadvantages regarding scalability as the test cases get more complex in bigger projects. Another point he states is the discipline required when multiple tester work in the same project.

Advantages documentation
As stated above definitely an advantage for A3.

Disadvantages documentation
It is hard to define the correct way, as the concept brings the advantage of readability, but probably with the disadvantage, that it is restricted and not all required tests are possible to be defined.

Efficiency & costs
On a scale from one (best) to five (worst), he ranks the concept with a score of three regarding efficiency and costs.

General rating
On a scale from one (best) to five (worst), he ranks the general concept and implementation with a score of three.

Application in practice
Definitely applicable, but it is necessary to find the right type of projects which fit to the concept, as each test concept has special advantages and disadvantages).

Optimization
He advises to apply the concept on a large project, and to take the time to evaluate the process (not only at the end). Afterwards, critical optimization points become visible.

A.7 Transcription expert interview 2

A.7.1 Transcription

Audiofile 1: #0:00

[Einleitung und Motivation des Themas]

[Erklärung BDD]

#04:33

A4: Kann man denn nicht denselben Code immer wieder verwenden, sagen wir einmal?

E: Genau, aber man müsste... natürlich geht das auch, aber man müsste eben, manuell von einem älteren Projekt den Code nehmen... oder meinst du jetzt ganz etwas anderes?

A4: Nein, ich habe mir gedacht, dass man den Code eben nur einmal schreibt und den irgendwo hingibt und jedes Projekt greift dann auf diese Grundbasis darauf zu und verwendet den.

E: Genau, das ist eine Möglichkeit. Das Problem dabei wird wahrscheinlich sein, dass unterschiedliche Projekte auch unterschiedliche Anforderungen haben. Sprich, da kommt eben, auf was ich danach noch eingehen werde, die Zeitrestriktion dazu.

A4: OK.

E: Und zum Beispiel, will nicht jedes Projekt, dass die Messages, die von meinem Tool an das Steuergerät gesendet werden, im selben Zyklus gesendet werden. Prinzipiell ist es jetzt konzipiert, dass man... ich glaube, ich zeige dir das fast besser... oder ich kann es dir auch jetzt schnell zeigen.

[Zeigt kleines Feature File für die Tests]

#07:49

A4: OK, das heißt, dahinter gibt es dann schon eine Library, die von allen verwendet werden kann.

E: Genau. Das ist genau meine Umsetzung. Diese zwei Geschichten hier, das ist...

A4: Das ist nur der Zugriff, wie man zu der Library... eben auf die Library an sich zugreifen kann.

E: Ganz genau.

A4: Noch eine Frage: Mit diesen Files usw... oder mit diesen Steps definiert man wie das Vokabular von den Testcases im Endeffekt dann... ich meine grundsätzlich würde das schon Sinn machen, dass ähnliche Projekte dann auch das gleiche Vokabular verwenden... oder weitgehend gleiche, sagen wir einmal so.

E: Ganz genau. Das ist eben schon bereits das erste Finding, das ich gemacht habe, dass wahrscheinlich nicht mehr im Scope von meinem Diplomarbeit liegt und umgesetzt wird,

aber es wäre auf jeden Fall sinnvoll ein für gewisse Dinge standardisiertes Vokabular zu definieren, um auch ein bisschen die Testcases so zu gestalten, dass so viel wie möglich wieder verwendbar ist. Das auf jeden Fall.

#09:17

A4: Ich meine jetzt auch, wenn... ich weiß nicht, ob das bei ECU Ltd. der Fall ist, aber wenn jetzt z.B. Leute zwischen den Projekten wechseln oder auch wenn Tester jetzt von einem Projekt... Das eine Projekt ist aus und er kommt zum nächsten Projekt. Und wenn dort das Vokabular das gleiche ist oder, sagen wir, zu 90% das gleiche ist, würde es gleiche Sachen geben, dann wäre man ja sehr viel schneller wieder drin.

E: Ganz genau.

A4: Jeder weiß dann, was mit den Worten eben gemeint ist und es meinen dann alle das gleiche damit [lacht]. Dann gehen auch Reviews und irgendwelche Sachen leichter, als wenn jetzt jedes Projekt für sich wieder von Null anfängt [lacht].

E: Auf jeden Fall, ja.

A4: Und ganz ähnliche Worte für unterschiedliche Sachen verwendet. E: Ja, sicher.

#10:10

[Erklärung des Tools und dessen Konfiguration, Erklärung DBC- File]

#14:23

A4: Das heißt, das ist eine Abstraktionsschicht...

E: Genau, das ist eine Abstraktionsschicht zwischen den Low-Level- Ausführungen und den Testbeschreibungen.

A4: OK.

[Erklärt die wichtigsten Klassen des Prototyps]

#15:40

[Erklärt die Motivation für die Struktur]

#16:16

A4: Das heißt, die stellen da jetzt auch ein nach oben hin eine API so zu sagen zur Verfügung, also ein abstraktes Interface.

E: Genau, diese Klassen hier sind abstrakte Interfaces, die Methoden und Attribute definieren, die jeder CAN Controller oder jeder I/O Controller benötigt und die dann gezwungenermaßen auch in einer neuen Implementierung von einem CAN-Controller bereitgestellt werden müssen. Und somit kann ich auch sicherstellen, dass wenn dieser neue Controller hier verwendet wird und kompiliert, dann weiß ich, dass ich eine Baudrate zur Verfügung habe. Weil wenn das nicht der Fall wäre, dann würde ich bereits beim Kompilieren einen Fehler bekommen.

A4: OK.

E: Und so habe ich diese Struktur eigentlich aufgebaut. Was ich noch dazu sagen könnte: das ganze Zeitliche bis jetzt wird auch im Maincontroller umgesetzt. Das ist, einmal ein Timer- Thread. Der ist eben dazu da, dass ich zyklische Nachrichten, die zum Beispiel eben oft bei einem Test benötigt werden, um das Steuergerät in einen Zustand zu bringen in dem es operativ wird. Diese Funktionalität anzubieten und das konfiguriere ich eben auch über das File... welche IDs ich zyklisch an das Steuergerät senden möchte. Das wird dann auch initialisiert und der Timer- Thread wird gestartet.

#18:02

[Erklärt den Stack der finalen Prototypversion]

#19:25

[Erklärt das Setup des Prototyps und welches Steuergerät/ welche Hardware verwendet wurde]

#20:15

A4: Der *Jenkins* ist der normale, so wie man ihn kennt?

E: Ja, ganz, ganz genau. In meinem Setup ist es jetzt ein lokaler *Jenkins*, der bei mir läuft.

A4: Ah OK.

E: Nicht der globale. Genau, wir haben einmal gesagt, wir testen es einmal in einem kleineren Setup und nein, natürlich, wäre es jetzt nicht das Problem, aber... Was man natürlich machen müsste, wenn das einmal ein *Remote-Jenkins* wäre, die Hardware, die ich benötige um die Tests auszuführen, müsste dann auch dort...

A4: Na ja, nicht zwingend.

E: Ja, nicht zwingend, das stimmt. Man könnte es konfigurieren.

A4: Du hast lokal einen PC und du kannst ja dem *Remote-Jenkins* sagen, dass da jetzt ein *Jenkins-Client-Slave*. Das könnte man probieren.

E: Ganz genau, das könnte man auf jeden Fall machen.

A4: Dann hättest du aber auch auf diesem lokalen PC die DLLs drauf und vielleicht Windows installiert statt Linux.

E: Ja, das muss man auch... getestet ist es bis jetzt nur auf Windows. Ich habe zwar darauf geachtet, dass ich so wenig wie möglich plattformspezifisch verwende. Man müsste es natürlich noch testen auf Linux, aber alles von mir in Python programmiert wurde, sei es pfadmäßig, was die größten Probleme meistens macht... das ist eigentlich alles plattformunabhängig gestaltet. Das einzige was wäre, derzeit im Windows gibt es eine relativ schöne Umsetzung für den *PCAN- Adapter*, das sind Windows Events, die eben einen Handle liefern, sobald eine Nachricht ankommt und die einen Interrupt bekommen,

jetzt liegt eine Nachricht an, die ich lesen kann. Was aber natürlich ohne Probleme auch mit einem Timer- Thread umsetzbar wäre.

#22:26

E: Ohne Probleme will ich jetzt nicht sagen, aber...[lacht]

A4: Aber es ist üblich [lacht].

E: Was ich davor schon gesagt habe, die größten Restriktionen jetzt noch ist einerseits auf der I/O-Basis die fehlende Testmöglichkeit... Wie gesagt, ich habe jetzt umgesetzt digitale/ analoge Ein-/Ausgänge. Von der Anbindung her, sobald die Hardware eine Anbindung zulässt, wäre es wahrscheinlich im Tool relativ einfach die Anbindung auch zu schreiben. Es sind mehr oder weniger eigentlich nur Wrapper auf die entsprechende DLL, die eben die Funktionalität bietet. Wir konnten in diesem Zuge jetzt nur einfach I/O-Tests durchführen. Wie gut das dann auch funktionieren würde in komplexeren Aufbauten das müsste man dann mit entsprechender Hardware dann testen. Und das zweite Problem, das tagtäglich im Embedded- Bereich da ist, ist eben die Time-Konstante. Das es vor allem, wenn es von einem PC gesteuert wird, sehr schwierig ist, genaue und kritische Tests zu definieren.

A4: OK, ja, also Echtzeittests.

E: Genau, Echtzeittests. Weil im traditionellen Ansatz wie sie es umsetzen derzeit mit zwei verschiedenen Steuergeräten, hat man zwar auch keine synchronisierte Abstimmung der Geräte, jedoch kann man wenn man die gleiche Zykluszeit konfiguriert, sich ungefähr sicher sein, wenn ich diese Message jetzt 20 mal sende, dann empfängt es das andere Gerät auch 20 Mal.

A4: Ja.

E: Ich habe das jetzt so gelöst, dass ich... Es ist, glaube ich, so ähnlich wie es auch in C durchgeführt wird. Ich habe einen Timer- Thread eben, der die Startzeit misst und die Zeit misst, wie lange der Thread benötigt für einen Durchgang und dann nach Konfiguration die restliche Zeit einfach wartet, um eine halbwegs synchrone Zykluszeit zu erreichen. Aber wie gesagt, es ist natürlich unmöglich genaue Tests zu definieren, das genau nach so viel Zeit irgendetwas passiert oder nicht.

A4: Solche Probleme kennen wir auch [lacht].

E: Genau. Wie gesagt, es erzählen alle, dass das eines der größten Probleme ist, überhaupt eben dann mit Windows und Linux und den verschiedenen Systemen.

#25:20

A4: Wir verwenden zum Teil jetzt Realtime- Linux. Mit Windows habe ich jetzt noch nicht so viel Erfahrung, aber ich kann mir auch vorstellen, dass das ziemlich schwierig ist [lacht]. E: Ganz genau. Das sind eben...

A4: Man ist eben auch generell auf Windows von der Hardware schon weiter weg, als wenn man direkt auf der ECU draufsitzt [lacht].

E: Ganz genau. Eben, das erschwert das Ganze noch einmal immens. Aber wie gesagt...

A4: Aber ich denke für die normale Basisfunktionalität ist das ausreichend. Man kann schon ziemlich viel testen, bevor man dann echt zum Mikrosekunden-Teil übergeht.

E: Ja, genau. Eben, das müsste eben... Da wäre es wahrscheinlich dann wieder notwendig das in einem größeren und genaueren Setup noch einmal nachzustellen.

A4: Ja.

#26:34

[Erklärt die drei Prototyptestfälle]

E: Wie davor schon gesagt, das sind jetzt drei relativ... nicht relativ einfach, aber schlank gehaltene Testfälle. Der erste zielt eigentlich nur darauf aus, wenn der Verkehr von meinem Tool zur ECU stoppt, oder wenn der Verkehr zur ECU stoppt, dann soll die ECU melden, dass die CAN-Kommunikation nicht mehr alive ist. Der zweite soll kontrollieren, ob die Ausgänge der beiden WW[Teil der Maschine]- Lichter auf High geschalten werden, wenn das Signal für die Rücklichter kommt. Und der dritte ist ein „Operational State Change“ der WW[Teil der Maschine] selbst. Da simulieren wir einen Keyboard Klick in der Maschine, um den WW[Teil der Maschine]- Modus von S1 zu LM zu wechseln. Und die Tests sind, wie auch schon gezeigt, in den sogenannten Step-Definitions implementiert und was dazu kommt, ist eben, das Environment- File. Das Environment- File ist jetzt *Behave* spezifisch und wird eben vor allem dazu verwenden um Initialisierungen durchzuführen, die in den Tests benötigt werden. Das ist eben der Ort, wo mein Main Controller initialisiert wird und auch wieder released wird. Und genau, wenn ich diesen Controller hier erstelle, dann kann ich den projektweit in allen Step-Definitions verwenden, wo ich ihn benötige. Und hier haben wir eben noch das Config-File. Ist aufgebaut, um mit Standard- Python- SafeConfigParser kompatibel zu sein. Es werden im Code dann, je nach angegebener Hardware, die notwendigen Parameter überprüft, ob die anwesend sind und dann eben noch auf optionale Parameter, falls es welche gibt. Und mit diesem Config wird gestartet und initialisiert. Und das sind die von mir erwähnten Mapping- Files, wo ein Keyboard auf eine, in dem Fall ist es eben ein Mapping auf ein I/O Board. Und was beim I/O Board noch zusätzlich eingebaut ist, ist die Frage, wie das Resetting funktioniert, weil natürlich auch eine Anforderung an uns war, die Ausführung der Testfälle unabhängig ist. Dass das System in jedem Zustand...

#29:50

A4: Dass man einen definierten Ausgangszustand hat...

E: Genau, dass man einen definierten Ausgangszustand hat, danke. [Erklärt das CAN und Mapping- File]

#30:32

A4: Und alle, die jetzt in den beiden nicht drinnen sind, werden auch mit Null initialisiert?
E: Genau, weil es natürlich sein kann, dass man auch nicht alle Pins verwenden muss oder

eben auch wie bei mir sind jetzt gewisse nur mit Lastwiderständen angeschlossen, aber die mit den Tests nichts zu tun haben, deswegen hat es für mich keinen Sinn gemacht, hier verpflichtend alle Pins anführen zu müssen, wenn ich dann nur die Hälfte oder nicht einmal die Hälfte verwenden muss.

A4: Da wäre dann der Initialaufwand ziemlich groß [lacht].

E: Ganz genau. Im späteren Fall sollte das dann eben der *Jenkins* übernehmen. In unserem Fall mache ich das jetzt über die Konsole.

[Demonstriert die Verwendung der Konsole]

#31:41

[Erzählt von der ursprünglichen Idee eines selbstständigen Tools und erklärt die Gründe für die Entscheidung dagegen, Library statt Tool]

#32:53

[Erklärt die Möglichkeit der Generierung ein *Jenkins* kompatiblen XML Resultats des Reports]

A4: Ja, das machen wir auch [lacht].

E: Ganz genau.

#33:31

[Erklärt die Nutzung für den Enduser: Config- File, Environment- File, Mapping- Files, DBC- File und Definition, Step- Definitions]

[Erklärt, dass auch nur CAN oder I/O Tests definiert werden können]

#35:03

A4: Kann ich beim Test irgendwie sagen von außen, dass ich jetzt nur eine bestimmte Gruppe von Testcases ausführen will oder nicht?

E: Genau, das wird auch eben schon durch *Behave* angeboten. Ich kann zum Beispiel meine Feature- Files in verschiedenen Ordnern hinterlegen und kann dann als Command Parameter den Ordner angeben, dann werden nur diese Testfälle ausgeführt. Oder ich kann, ich bin mir jetzt nicht mehr 100% sicher, wie die Syntax ist, aber ich glaube, ich kann ein Feature- File angeben und mit einem "@" den Namen des Szenarios. Dann wird wirklich nur dieses Szenario ausgeführt. Genau, das kommt aber wie gesagt nicht von mir, sondern das ist bereits Funktionalität der Test Implementierung, die ich da verwendet habe. Eben und ich habe diese verwendet, weil sie meines Erachtens die verbreitetste, oder nicht unbedingt verbreitetste, aber von den Funktionalitäten her ziemlich ausgeklügelte Variante gibt, die... eben es gibt jetzt auch nicht unglaublich viele *Python* Implementierungen für Behaviour Driven.

A4: Ja.

#36:32

[Erzählt über die anfängliche Evaluierung in Richtung Behavior und Entscheidung für Command Line Lösung (*Behave*)]

#38:00

[Beendet die Live Demo]

E: Jetzt wäre die Frage, ob es von dir noch Fragen gibt zur Umsetzung, zum Tool, zur Vorgehensweise? Oder ob dir mehr oder weniger alles schlüssig vorkam? [schmunzelt]

A4: Ja, doch schon schlüssig. Interessant wäre, ob es irgendwie so eine Abschätzung gibt wie umfangreich jetzt ein Feature- File wäre für einen Kompletttest von so einer "ECU irgendwas"? Eben auf welche Größenordnung man da ungefähr kommt? Vom Vokabular her und von der Größe, von dem was der Tester eben dann schreiben muss an Dokumentation und Feature- Files.

E: Das ist eine sehr gute Frage. Beantworten kann ich sie dir jetzt wahrscheinlich nicht. Das wäre dann auch die Sache eines Pilotprojektes, in dem das dann Einsatz findet. Wo man dann wirklich abschätzen und wo man sich dann auch die Zeit nehmen müsste, das zu evaluieren, wie viel Zeit da dann wirklich reingesteckt wurde. Was man im Allgemeinen darüber sagen kann, ist, dass diese Feature- Files, dienen eben wie gesagt... oder sind nicht nur Teil der Testimplementierung, sondern sind auch schon in menschenlesbarer Form. Das heißt, in unserem Prototyp, oder in unserem Stack, wäre dieses File beispielsweise schon ein Output für das VCP zum Beispiel - Verification Case Procedure Protokoll. Auf jeden Fall dort wo alle Testcases... die werden eben bei der, ich kann jetzt nur von ECU Ltd. reden, gerade im *Integrity* definiert und dokumentiert. Und diese hier definierten Texte könnten schon Input für dieses Dokument werden. Und dadurch, dass man einen automatischen Trace...ja, Trace auch und automatische Konsistenzprüfung von der Dokumentation und dem Test selbst hat, ist es eigentlich eine Methodik um diesen Teilgebiet Ressourcen einzusparen. Da spricht jetzt wahrscheinlich meine persönliche Meinung mit rein, da gibt es sicher Menschen, die das anders sehen. Aber ich finde, dass die Formulierung dieser Art Tests nicht unbedingt ein mehr Aufwand, jetzt vom reinen Schreiben her, weil... die Beschreibung der Testfälle müsste sich der Teste auch überlegen, um die eben zu dokumentiert. Und die Testimplementierung braucht er sonst auch. Der strittige Punkt ist es ob jemand argumentiert, dass diese Art von Testimplementierung, sprich das jede Methode nur einen kleinen Teil der Arbeit, des Tests, übernimmt, ob das einen mehr oder weniger Aufwand mit sich bringt. Gewisse Menschen werden sagen, sie tun sich leichter, wenn sie eine Funktion schreiben, die dann diese ganzen Schritte untereinander durchführt und unten das Ergebnis kontrolliert. Andererseits könnte man auch argumentieren, dass eine Zeile „When receiving the CAN signal“ mit diesem Namen...

[Zeigt ein Beispiel].

#42:42

E: Das heißt, ich würde mir in dieser Hinsicht wieder Zeit und Code sparen, weil ich diese Ausformulierungen nur einmal definieren muss.

A4: Und machst keinen Spaghetticode [lacht].

E: Auch die Abhängigkeit von gewissen Code Parts könnte sich dadurch auch vermindern.

A4: Ja. Was ich mich frage ist, wie, wenn man jetzt komplexere... ich weiß ja nicht, wie komplex die Szenarios werden für irgendwelche komplexeren I/O, usw. Wie groß man erwarten kann, dass das wird? Wie viele boolesche Operatoren man dann dazwischen hat, um das zu verschachteln alles. Das wäre für mich eine Frage, die wahrscheinlich auch...

E: Da gibt es zwei Ansätze. Es gibt natürlich die Möglichkeit, wie ich es hier in kürzerer Form gemacht habe. Prinzipiell, die Textformulierungen haben ja auch ein bestimmtes Format und bestimmte Richtlinien – die sind da auch relativ schön eingezeichnet. Und wie ich hier die sogenannten Pre-Conditions definiert habe, könnte man das natürlich ins Unendlich weitermachen und mit ANDs verknüpfen. Das kommt meiner Ansicht nach auf die Art des Projekts an. Im Allgemeinen rät man ein Szenario so klein wie möglich zu halten. Ein Szenario sollte von der funktionalen Test Anzahl so wenig wie möglich testen, weil natürlich wenn man hier ein riesiges Feature Test möchte in einem Zug, das man in einem Test über drei bis vier States wechselt, und Sachen wieder abfragt und wieder neu setzt und schaut was dann passiert, dann ist es natürlich auch klar, dass diese Beschreibung ins Unendliche wachsen könnte. Auch wenn man es klein lässt, dann könnte man sagen, es gibt Szenarien, wo es einfach extrem viele Vorbedingungen braucht, damit der Test durchgeführt werden kann.

A4: Ich möchte zum Beispiel den 6. State von sehr vielen States testen und ich muss da irgendwie erst einmal hinkommen.

E: Dann gibt es zwei verschiedene Ansätze. Der erste ist das Ausformulieren, was nach Philosophie von *BDD* nicht gemacht werden sollte. Und das zweite wäre, die Komplexität von hier raus zu nehmen und in den Hintergrund zu verschachteln, um, was natürlich eine Möglichkeit wäre, ist im Step- File für das spezifische Projekt, das getestet wird, State- Übergänge zu definieren. Das heißt, ich muss im Code nur mehr definieren „Given Ich bin im State 3“...

A4: ... der Code im Hintergrund...

E: ... im Binding ist definiert, was bedeutet „Im State 3“, also wie komme ich in den State 3. Eben die Komplexität von hier herausnehmen. Ist aber meines Erachtens aber auch eine Frage des Test- Scopes, auch was sollen mir die Tests sagen, wenn ich sie durchlese? Will ich diese Details, zum Beispiel dass genau dieses Signal mit diesem Wert kommt im Testfall sehen oder interessiert mich das nicht? Es ist vielleicht auch eine Frage, für wen sind die Tests? Sind die Tests jetzt dem Kunden zu kommunizieren „Schau, wir testen gerade das. Deckt das ab was du dir erwartest an Tests oder nicht?". Oder sind es eben Tests für andere Entwickler oder Deployer oder Tester, für die solche Details auch relevant sind dann, wenn sie einen Testfall durchlesen.

#47:18

A4: Zertifizierungsprojekte sind da wahrscheinlich auch ein guter Use- Case...

E: Ganz genau. Ist auf jeden Fall ein sehr guter Einwand und ist auch ein kritischer Faktor für die Anwendbarkeit dieses Konzepts.

A4: Ja. Ich denke vor allem, wenn man sagt, man nimmt diesen Text der da steht und verwendet ihn nur um dieses Dokument zu machen, das *PTC* jetzt... und dann wird zum Beispiel das *PTC*- Dokument erzeugt für irgendeine Zertifizierung... dann muss das detailliert genug sein damit sich ein Auditor damit auskennt! [lacht] Oder er muss halt davon überzeugt werden, dass im Hintergrund das richtig gemacht wird [lacht].

E: Ja, sicher.

A4: Aber das muss man wahrscheinlich ausprobieren einmal in einem Projekt...

E: Und was man auch dazu sagen muss, dass liegt dann auch immer wieder in der speziellen Vorgangsweise des Users, der das dann verwendet. Weil ich biete ihm sozusagen nur die Mittel, die er verwenden kann, aber wie er sie verwendet, ist ihm dann mehr oder weniger überlassen.

A4: Ja, ja.

E: Und da gibt es wahrscheinlich eben Projekte oder Use- Cases, für die äußere Verschachtlung von Pre- Conditions geeignet ist und dann gibt es vielleicht auch welche, wo man sagt, man beschreibt eine Verschachtlung von 10 Pre- Conditions in einer Ausdrucksweise, die für dieses Projekt passt und verschiebt dieses Binding dann zurück eine Schicht und lässt dort dann definieren, was alles passieren soll. Das kommt dann... eben, wenn ich dann 50 verschiedene Tests zu diesem State habe, um nicht dann 50 Mal hier 10 verschiedene Pre- Conditions zu verschachteln, weil das natürlich auch die Leserlichkeit beeinträchtigt. Dort würde es dann auch wieder Sinn machen, das zu verschieben.

#49:38

A4: Auch zu dem wäre meine nächste Frage. Wenn man jetzt viele Test Cases hat, die im Prinzip gleich sind, bis auf ein/ zwei Werte die man umsetzt. Kann man das auch irgendwie handhaben in einer gescheiten Form?

E: Genau.

A4: Also angenommen ich will auf einem analogen Output, was weiß ich, 20 verschiedene Werte ausprobieren [lacht]...

E: Ich glaube, ich weiß schon... Genau. Was es hier gibt... Zum Vergleich, es gibt Scenarios, das sind im Standard Chargon Testfälle. Und dann gibt es Scenario- Outlines.

[Erklärt den Unterschied und die jeweilige Funktionalität des *Behave*- Parsers]

#51:37

A4: Weil du vorher einmal von... anderes Thema, wegen dem Timing. Wenn man so was definiert, das man sagt, man will es zu einer bestimmten Zeit oder man erwartet sich, dass das in einer bestimmten Zeit kommt.

E: Genau. Es ist prinzipiell vorgesehen, dass man das kann. Ich stelle die Funktionalität zur Verfügung, aber was ich jetzt eben aus meinen Tests bereits gesehen habe und erkannt habe ist, dass diese Tests nicht zu 100 % zuverlässig sind. Ich habe zum Beispiel getestet, wenn ich 49 Zyklen diese Message sende, dann bekomme ich noch keinen Error zurück, aber dann beim 50. Zyklus schon. Und wenn ich das dann bei mir, habe ich das dann eben so hingesetzt, dass es dann für 49 Zyklen sendet, sendet, sendet, immer wieder liest und schaut „OK, kein Error, kein Error, kein Error, kein Error“ und dann beim 50. soll dann der Error da sein. Und das habe ich dann, jetzt mittlerweile schon X-fach durchlaufen lassen, aber zu 70-80% passt es mit dem Timing, dass es das Steuergerät eben auch genau 49 Mal empfängt und dann das 50. Mal... Und dann, aber hin und wieder passiert es, dass er es einmal zu wenig bekommt und dann ist der Error eben nicht gesetzt... Beim letzten Interview habe ich Feedback bekommen, dass bereits schon vor einigen Jahren eine Art Tool geschrieben wurde in Python, auch für Tests und dort auch so ähnliche Konstrukte verwendet wurden, um CAN- Messages zu senden, um CAN- Messages wieder zu empfangen. Und habe auch bereits einen Kontakt erhalten, an wen ich mich da wenden könnte. Und wir würden jetzt eben noch gerne ein Treffen arrangieren, um über die Vorgehensweise zu diskutieren, wie sie da heran gegangen sind und wie vielleicht auch Optimierungen für meine Implementierung bieten könnten.

A4: Aber ich meine, das ist ja dann unter der Oberfläche. Ich meine, das würde jetzt zum Feature- File wahrscheinlich nichts... ändern.

E: Wenn es... Man kann es auf jeden Fall. Wenn es für den Test wichtig ist, auch für die Beschreibung wichtig ist zu sagen, ich habe nach 500 Millisekunden den Fehler, dann kann man das auf jeden Fall hier auch definieren. Das ist ja eigentlich auch das Gute und vorteilhafte an der Methodik, dass man die Sätze im Feature- File nach den Ansprüchen des Projekts und wie die Tests ausschauen soll, formulieren kann. Und auch formulieren kann, welche Schlüsselwörter ich verwende, die dann im Hintergrund zu Parametern werden.

A4: OK.

E: Wir können dann auf jeden Fall auch wenn es während den Fragen eine Frage von dir gibt, dann ist das natürlich auch kein Problem.

Audiofile 2: #0:00

E: Dann würde ich dich bitten... also, wie gesagt, das Interview wird vom Audiofile aus transkribiert, anonymisiert und dann zu Evaluierungszwecken passagenweise in der Arbeit wiederverwendet. Die Transkription wirst du von mir bekommen zum Gegencheck und möchte dich nur kurz bitten, dich für den Ton kurz vorzustellen. Einfach Name, das Tätigkeitsfeld und wie lange du schon im Tätigkeitsfeld tätig bist und ob du Erfahrungen im Bereich *BDD* oder Embedded- Testing hast.

A4: Mein Name ist A4. Ich bin seit 13 Jahren in der ECU Ltd. und seit ungefähr 11 Jahren im Software- Testing, also eigentlich System- Tests in dem Sinn. Erfahrung mit BDD habe ich noch keine. Ich habe es mir kurz einmal angeschaut, aber noch in keinem Projekt verwendet. Embedded- Testing... ein wenig. Ich meine, wir testen Treiber und solche Sachen mit, aber in einer anderen Weise, als das jetzt hier bei den ECUs passiert. Wir haben normalerweise Einschubkarten und PCs und dann kann man das vielleicht nicht ganz vergleichen. Ja.

E: Super, perfekt. Dann würde ich gleich mit der ersten Frage starten. Die wäre, ob du Vorteile im Nutzen dieses Konzepts siehst verglichen mit Testansätzen, die du selber verwendest oder mit denen du Erfahrungen hast?

A4: [überlegt] Also ein Vorteil wäre sicher, der... Reviews vom Feature- File als Dokumentation gleichzeitig. Wenn das vom Prozess her möglich ist, also unter der Voraussetzung, dass das vom Prozess her erlaubt ist, dann ist das sicher ein Vorteil, wenn das konsistent bleibt und weniger Aufwand ist. [Pause] Was eventuell auch ein Vorteil sein kann, dass Leute Tests schreiben können, die nicht gleichzeitig Programmierer sind. Ich meine kommt darauf an, in wie weit sie dann... also diese Feature Funktionen da, wer das dann schreibt [lacht], aber zur Definition von den Testcases selber das könnte jemand machen, der kein Programmierer ist. Das ist bei anderen Vorgangsweisen eigentlich nicht, also bei uns zumindest nicht so.

#03:41

E: Ich schmeiße auch gleich die zweite Frage, da ginge es eben um die Nachteile. Also dieselbe Frage nur im Bezug Nachteile?

A4: [überlegt] Also, es ist schwierig jetzt Nachteile wirklich zu definieren. Ich glaube, man müsste einmal ein komplettes Projekt ausprobieren, um das wirklich beurteilen zu können, was das jetzt heißt, Szenarien zu definieren für eine komplette ECU zum Beispiel. Noch was man sonst noch damit testen will und wie viel Aufwand das dann wirklich ist und wie dann zum Beispiel diese Grammatik von den Test Cases dann wirklich aussieht im Endeffekt. Also wie viel Aufwand das wird und wie gut man sich nachher dann noch auskennt damit. Also wie gut das in dieser Hinsicht dann skaliert oder... ja, das ist sicher ein Ding... auch eine offene Frage wie das dann aussieht, wenn man das in mehreren verschiedenen Projekten verwendet, wie viel man dann re-usen kann davon und wie... oder wie unterschiedlich die dann wären. Das könnte auch ein wichtiger Faktor sein bei längerfristiger Nutzung. Eine offene Frage für mich ist auch, ob sich alle Szenarien auf diese Art gut genug beschreiben lassen und man sich dann wirklich auskennt. Also, wenn es um komplexe Testszenarien geht. Ich meine normalerweise ist das sicher kein Ding oder Thema, da funktioniert es sicher gut. Aber ich könnte mir vorstellen, dass da irgendetwas Komplexeres gibt. Also... Ich meine zum Beispiel, in meinem... anderer Vergleich, aber in meinem Bereich, wüsste ich nicht wie ich solche Szenarien definiere zum Beispiel, aber wie gesagt für die ECU sieht das jetzt brauchbar aus.

#06:30

E: Okay. Haben wir auch schon angeschnitten: Ist die Verschmelzung von der Testimplementierung und der Testdokumentation vorteilhaft gegenüber traditionellen Testansätzen? Aber erschließen sich auch Nachteile daraus? Oder könntest du dir auch Nachteile vorstellen, was diese Verschmelzung mit sich bringt?

A4: Also grundsätzlich bin ich schon ein Fan davon, ich versuche das auch immer wieder auf einem anderen Weg. Die Beschreibung daraus generieren aus dem Testcase zum Beispiel, ist für mich ein Thema. Wenn man das Ganze jetzt unabhängig sehen würde von irgendwelchen Prozessanforderungen, dann sehe ich jetzt, glaube ich, keinen Nachteil. Ich glaube, was ein Problem werden kann, ist eben das... das kommt auf das Einzelne darauf an, das irgendein, das ein Prozess vorschreibt „Du brauchst zuerst die Testcase-Beschreibung“. Also dass das vom Prozess her nicht durchführbar oder erlaubt wäre oder nicht akzeptiert wird von irgendwelchen Projektmanagern oder.... Das könnten Nachteile oder Nachteile... das könnten eben Punkte sein, wo man darüber reden muss, ob man es jetzt zulässt oder nicht... oder zulässig ist oder nicht. Sonst vom... aus Testersicht ist das... sehe ich jetzt keine Nachteile.

#08:25

E: Jetzt würde es darum gehen, die gesamte, komplette Lösung beziehungsweise Umsetzung zu bewerten, anhand verschiedener Kriterien, die auch miteinander teilweise verknüpft sind. Ich definiere einfach einmal die Frage und erläutere dann noch, was ich damit meine. Die Frage lautet: Auf Skala von 1 bis 5, wobei 1 das Beste ist und 5 das Schlechteste, wie würden Sie die vorliegende Lösung im Sinne der Testeffizienz bewerten und warum? Sprich, 1 bedeutet „es ist effizienter", oder „sehr effizienter“, 5 ist „es ist bei weitem nicht so effizient“ und 3 wäre „es ist gleich effizient“.

A4: Mhm.

E: Ich kann aber auch die restlichen zwei auch noch... Also, die Fragen sind immer dieselben, es geht bei der ersten Frage um Effizienz, bei der zweiten um Testkosten und bei der dritten um die Gesamtlösung, also, im Gesamten unter allen Aspekten betrachtet.

A4: [überlegt] Da tue ich mir ehrlich schwer bei diesen Einschätzungen. Testeffizienz... [Pause] Ich tue mir schwer das zu beurteilen jetzt ohne das an einem komplexen Problem gesehen zu haben.

E: Verstehe ich.

A4: Grundsätzlich würde ich sagen, dass das schon eher effizienter ist, als traditionelle Methoden.

E: Wir könnten es auch unter Anführungszeichen beschränken, jetzt nicht auf einfache Testfälle, aber auf den Bezug Applikationstests auf Steuergeräten für die ich das Konzept eigentlich auch erarbeitet habe. Also um den Fokus jetzt nicht auf hoch sensible oder sicherheitskritische Systeme zu testen, sondern eben den Fokus auf diese Anwenderschiene auszulegen.

#12:28

A4: Ja, dann würde ich 1-2 Testeffizienz. Unter der Annahme eben, dass man das wirklich auch als Doku verwenden kann 1:1, also auch Doku zu haben, und dass man so größere Varianten von Tests, die ich irgendwie über Tabellen abbilden kann. Ja.

E: Also die Bewertung würde dann für die anderen zwei auch so gelten?

A4: Was war das zweite noch einmal? [lacht]

E: Das erste ist Testeffizienz, das zweite ist Testkosten und das dritte wäre die Lösung im Gesamten.

A4: Ja, ich meine, die Testkosten hängen ja in diesem Sinne von der Zeit ab, die man braucht. Weil das am Anfang das Teuerste ist. Von dem her gehört das zur Effizienz, proportional, hätte ich jetzt einmal gesagt. [Pause] Das Overall... da wären eben noch die Unsicherheitsfaktoren von den Unbekannten, die ich jetzt nicht beantworten kann. Eben in diese Richtung große Projekte...und so Sachen... Weil vielleicht hängt es auch davon ab, wer das dann wie verwenden wird. Oder macht das ein Entwickler bei uns oder ist das irgendwer, der noch überhaupt keine Ahnung hat von, bei [Zulieferer] oder sonst irgendetwas. Je nachdem kann es noch mehr bringen als bisherige Methoden oder es kann auch ein Problem werden. Also das würde ich jetzt nicht voraussagen wollen. Aber ich denke... ich finde den Ansatz auf jeden Fall sehr interessant und verstehe es auch durchaus.

#15:08

E: Dann kommen wir schon zur letzten schwierigen Frage und das wäre: Ob du dir eben vorstellen könntest, dass diese erarbeitete Umsetzung oder das erarbeitete Konzept in der Praxis Anwendung findet? Wenn ja, aus welchen Gründen? Wenn nein, warum nicht? Und ich sage auch dazu, du hast jetzt auch schon einige Fürs und unter Umständen Widers aufgezählt. Wenn du keine restlichen mehr hast, dann wäre diese Frage schon im Vorhinein so beantwortet. Und ob du dir... aber als zweite Unterfrage eben: Ob es Anwendungen oder Projekte gibt, wo du dir vorstellen könntest, dass das gut passen könnte oder auch nicht so?

A4: [überlegt] Also ich finde in so einem ECU Projekt in so einem Umfang kann man das auf jeden Fall verwenden. Dadurch beantworten sich die meisten Fragen vermutlich [lacht]. Also, für so ein ECU Projekt auf jeden Fall. Darüber hinausgehend, unter Umständen wäre es interessant... sagen von der Hardware her oder generischen Test Hardware sein könnte. Wäre auch wieder anzusehen. Ich weiß jetzt nicht, ob das bei ECUs verwendet werden soll, aber ob damit auch verschiedene Hardware Komponenten getestet werden würden.

E: Genau.

A4: Ich meine da ist derzeit eine *LabView* - Lösung irgendwie angebracht, aber das schließt ja nicht aus, dass man über Python das irgendwie ansteuert. Also da könnte man sich sicher was überlegen. Ja, den genauen Durchblick habe ich noch nicht. Habe ich es noch nicht genau durchdacht oder durchschaut, wie ich es verwenden kann.

E: Also für die Richtung Treiber- Tests.

A4: Netzwerktests, Systemtests. Was wir haben sind eine beliebige Anzahl eigentlich... von 1 bis n PCs mit n Systemkarten drin, mehrere Switches dazwischen und es werden dann Konfigurationstests gestartet, also durchlaufen lassen. Da wird abhängig vom Testcase Testcode generiert, der auch abhängig von der Konfiguration ist. Das ganze wird dann auf die einzelnen Systeme verteilt und gestartet. Dann analysiert man was passiert. Dann noch Switches ein- und ausschalten, solche Sachen. Ja.

E: Und du siehst das Kritische in der Formulierung der Tests oder in ihrer Abhängigkeit von der Zeit, oder?

A4: Nein, nein. Die zeitlichen Komponenten werden in diese Entwicklung von den C Applikationen auf den einzelnen Testgeräten gelöst sozusagen. Das Python selber ist nur der Testmaster. Der generiert die Applikationen und spielt die auf die einzelnen Rechnern runter. Die rennen dann unter Realtime Linux zum Beispiel und machen das Ganze Zeitliche. Python ist dann am Zeitlichen nicht beteiligt. Ich meine, wir haben trotzdem noch Probleme, jetzt nicht so wie wenn ich von einem Python- Master aus auf 10 Rechner in real-time ansteuern müsste. Also das nicht, es ist eher so die Formulierung, wie "mappe" ich das auf das ich schon habe [lacht]. Solche Sachen.

#20:12

E: Genau. Also zu den Evaluierungsfragen wäre das jetzt die letzte gewesen. Zum Abschluss gäbe es noch die Frage für Empfehlungen, für zukünftige Arbeiten in diesem Gebiet beziehungsweise als Weiterführung meiner Umsetzung, ob dir was vorschweben würde?

A4: Naja, ich denke... Also, ich meine der nächste Schritt wäre logischerweise eine Pilotphase, also ich meine ein Pilotprojekt. Wo man eben die ganzen offenen Fragen anschaut. Und der nächste Schritt, ich glaube, danach wäre dann sich andere Einsatzszenarios anzuschauen, wie zum Beispiel diesen Hardwaretest. Oder ich weiß nicht, vielleicht gibt es bei uns noch andere Aufgaben auch für was man das verwenden könnte [Pause]. Ansonsten Empfehlungen... Ich meine, ein Punkt wäre sicher, aber ich weiß nicht, ob man das... das wäre vermutlich eine eigene Arbeit, dass man wenn man sie in mehreren Projekten einsetzt durch irgendwelche Maßnahmen schaut, dass die Grammatik die gleiche bleibt. Ich glaube, das würde sich schon rentieren... auf lange Sicht nämlich.

E: Das ist noch ein Punkt, den ich versuche jetzt abzu... oder versuche eine Art Lösung dafür zu finden. Das kann ich noch als Nachtrag sagen, dass den Editor, den ich hier verwende, der heißt *Atom* – ich weiß nicht, ob du den kennst?

A4: Nein, kenne ich nicht.

E: Der ist von den Entwicklern von *Github* geschrieben und ist... vielleicht sagt dir *Sublime* etwas?

A4: Gehört schon, aber...

E: Auf jeden Fall das Prinzip in diesem Editor ist, dass die Basisfunktionalität sehr klein ist, aber durch sogenannte Packages kann man sich jede Menge an Funktionalität nachinstallieren und diese Packages kann man natürlich auch selbst entwickeln. Das Package, das ich hier verwende, ist eben ein... nennt sich *i18n-gherkin-language* und zielt eben darauf aus, dass... es gibt *Behave* beziehungsweise Behaviour Driven Tests in allen möglichen Sprachen, ich glaube in 80 Sprachen, wo sich natürlich diese Wörter hier ändern und in dem Editor hat man jetzt eben die Möglichkeit, die Sprache auszuwählen und dadurch eben die jeweiligen Schlüsselwörter zu highlighten. Das steuert jetzt in die Richtung „Wie unterstütze ich den User beim Erstellen dieser Files?" und das Plugin hat es zwar schon... oder das Package gab es schon, habe es dann gefunden und habe bemerkt, dass es nicht mehr mit den derzeitigen Standards übereinstimmt und habe dann den Entwickler dieses Packages kontaktiert und mit ihm zusammen das Package in Schwung gebracht. Und so wäre meine Intention und Idee in diese Richtung dieses Problem mit der Grammatik in den Griff zu bekommen, wobei es eine relativ schwierige Frage ist, wie man den User am besten dazu bringt, gewisse Grammatik einzuhalten. Eine erste Idee wäre es zum Beispiel anhand von Schlüsselwörtern die eingegeben werden, gewisse Sätze anzeigen zu lassen, eben standardisierte. Das heißt, wenn ich merke er schreibt „CAN“, was könnte er jetzt wollen, vielleicht will er ein CAN-Signal lesen oder will ein CAN- Signal schreiben und ihm dabei auch zu zeigen: „OK, du könntest jetzt bereits diesen ganzen Satz verwenden“.

#25:24

A4: Dass man da im Prinzip schon die fertigen anzeigt, die schon implementiert sind, oder so eine Vorauswahl?

E: Genau, so etwas in die Richtung. Ist natürlich jetzt noch nicht implementiert und müsste dann wahrscheinlich auch evaluiert werden, wie das angenommen oder eingehalten wird.

A4: Aber man würde sich jedenfalls sehr viel zum Schreiben ersparen. Ich finde das schon sehr sinnvoll.

E: Aber das wäre jetzt eben ein Punkt, der eben zu den Erweiterungen nicht mehr im Scope von der Diplomarbeit, aber für meine Umsetzung im Fokus liegt [Pause]. Wenn es sonst keine Fragen mehr von deiner Seite aus gibt, bedanke ich mich für deine Zeit und hoffe, dass ich die Arbeit noch erfolgreich abschließen kann in naher Zukunft.

A4: [lacht] Ja, also bitte. Aber ich finde es jedenfalls sehr spannend. Bin gespannt etwas von einem Pilotprojekt zu hören, hoffentlich [lacht].

#26:55

E: Ja, werden wir auf jeden Fall schauen. Ich glaube, es ist noch nicht ganz klar, wo das dann stattfinden wird. Anbieten würde sich sicherlich in B etwas, weil die genau mit dieser Art Tests zu tun haben. Aber ich glaube auch, S1 hat schon ein oder zwei Sachen im Kopf, wo er auch Anwendung hier sieht und dann werden wir sehen, wie stark das wirklich nutzbar ist.

A4: Ich denke, das sieht man wirklich erst, wenn man es im realen Betrieb verwendet. Eben diese ganzen zusätzlichen Einflussfaktoren, wie eben Prozesse und das ganze Drumherum. Die Auswirkungen und ja... das ist eben schwer vorher einzuschätzen. Und auch wie die Leute es anwenden... kann man ja nicht wirklich vorher sagen.

E: Sicher.

A4: Bei diversen Sachen schon Probleme gehabt... bei manchen Leuten, bei andere wieder nicht, aber das sieht jeder anders. Das unter die Leute bringen ist eigentlich ein ganz wichtiger Schritt... ein sehr kritischer Schritt. Bei uns muss man wirklich aufpassen.

E: Ja, ganz genau. Alles klar.

A4: Ja, dann noch viel Erfolg.

E: Noch einmal Danke...

A4: Ja, bitte.

[Abschließende Worte]

Ende

A.7.2 Paraphrasing expert A4

Profession
A4 works as system tester at ECU Ltd. for the past eleven years.

Advantages of concept
He sees the usage of feature files as documentation as big advantage. Moreover, the fact, that not only programmers may define test cases, because of the human readable form of the feature files, seems to be an advantage for him.

Disadvantages of concept
It is also difficult for him to define disadvantages by now. He sees potential problems in scalability, re-usability, maintenance effort and complexity of test cases, but those factors have to be evaluated within a bigger pilot project.

Advantages documentation
On the technical perspective are only advantages for him regarding the fusion of test documentation and test implementation.

Disadvantages documentation
No real disadvantages identified by him, one critical point could be restrictions in specific processes (e.g. test documentation and implementation have to be independent).

Efficiency & costs
On a scale from one (best) to five (worst), he ranks the concept with a score of one to two regarding efficiency and costs.

General rating
On a scale from one (best) to five (worst), he ranks the general concept and implementation with a score of one to two.

Application in practice
For the prototypical tested project type (embedded application on ECU) it is definitely applicable. To find other fields, a respective prototype project should be defined and evaluated.

Optimization
He recommends to start a pilot phase to evaluate all open questions and possible restrictions.