Automatic Test Assessment and Verdict Generation by CAN Monitoring

A software architecture to enable automatic in-vehicle testing

MILIVOJ LÄRFARS PERSSON
Automatic Test Assessment and Verdict Generation
by CAN Monitoring

A software architecture to enable automatic in-vehicle testing

by

Milivoj Lärfars Persson
Abstract

In this thesis the author presents an investigation on how to automate and streamline in-vehicle testing by developing a novel automated in-vehicle testing tool. The developed software architecture aims to reduce the amount of manual labor in order to decrease the time necessary to perform the manual in-vehicle tests. This is realized by a software tool with a client-server architecture, where the server monitors the Controller Area Network (CAN) traffic on one of the vehicle’s CAN buses and the clients are test steps to be performed on the System Under Test (SUT). The test steps can be written in the Python programming language and bolted on individually. Through the use of Independent Guarded Assertions (IGAs) multiple tests are allowed to execute continuously and in parallel.

The thesis, written at Scania CV AB, investigates the field of automotive testing by literature study, to identify the State Of The Art (SOTA) in the field; interviews, to capture the Scania CV AB’s needs and requirements; and a case study, to verify and analyze the implemented testing tool. The result of the literature study shows that there already exist many possible solutions, but they focus mostly on virtual environments and software based systems, thus not directly compatible with the in-vehicle environment. From the interviews a number of requirements are identified, both regarding tool functionality and safety related aspects e.g. intrusiveness of in-vehicle tools. Results from the case study shows that the in-vehicle testing environment is more intricate than initially perceived, both in terms of the factors that dictates the time necessary to perform it and technical complexity.

In conclusion, this thesis shows that there is a great potential to improve the manual in-vehicle testing by adding more automated support and tools. It is also evident that the execution time is not the only prospect that can be positively affected. Reproducibility, formal test specifications, and the possibility to produce richer test cases are some of the advantages discovered.
Sammanfattning


Som slutsats visar examensarbetet att det finns en stor potential att förbättra manuella fordonstester genom att tillföra mer automatiserade verktyg. Det är också tydligt att det inte bara är tiden det tar att exekvera test som kan påverkas positivt. Reproducerbarhet, formella testspecificationer och möjligheten att skara mer uttrycksfulla testfall är några fördelar som upptäckts.
Acknowledgment

I would like to express my gratitude to my supervisor at Scania CV AB, Simon Fagerholm, for helping me throughout the thesis project, with matters of both technical and administrative nature. Also, I would like to thank my supervisor Xinhai Zhang for his feedback on the report. Furthermore, I would like to show my appreciation to the participants in the interviews I have conducted and for the understanding and patience of the two drivers helping me with my case study. Without all of the help I have received, this report would look nothing like it does now. My sincere thanks.
## Contents

1 Introduction .................................................. 7
   1.1 Motivation ............................................... 7
   1.2 Background ............................................. 8
      1.2.1 Industrial Aspect .................................. 8
      1.2.2 Test Environments ................................ 9
      1.2.3 Testing Techniques and Test Methods ............. 13
   1.3 Terminology ............................................ 14
   1.4 Use Case ................................................. 17

2 Research Design ........................................... 19
   2.1 Research Goal .......................................... 19
   2.2 Research Question ..................................... 19
   2.3 Scope and Assumptions .................................. 20
      2.3.1 Limitations and Internal Assumptions .......... 20
      2.3.2 Included Scope .................................... 21
      2.3.3 External Assumptions .............................. 21
   2.4 Research Methods ...................................... 22
      2.4.1 Literature Study .................................. 22
      2.4.2 Interviews .......................................... 22
      2.4.3 Case Study with Demonstrator .................... 22

3 Literature Review ........................................ 24
   3.1 Literature Review Approach ......................... 24
   3.2 Results from Literature Review ..................... 25
      3.2.1 Difference Between the Automotive and General Case 25
      3.2.2 Model-based Testing In the Automotive Context 25
      3.2.3 Automotive Test Representation and Language .... 26
      3.2.4 Exotic Algorithms for Vehicle Testing .......... 27
      3.2.5 Monitoring Vehicle State and Data Acquisition 28
      3.2.6 Independent Guarded Assertions .................. 29
      3.2.7 Test Oracles ........................................ 29
   3.3 Discussion on Literature Review Results .......... 30
      3.3.1 Analysis Performed During Automotive Testing 30
      3.3.2 Integration in the Quality Assurance Process .... 30
      3.3.3 Choosing Test Language ............................ 31
      3.3.4 Extensibility of Testing Tools .................... 31
      3.3.5 Issues with Intent Approximation ................. 31
   3.4 Conclusion of Literature Review ................... 31
List of Figures

1.1 V-model at Scania. ................................................................. 9
1.2 SIL testing framework in the general case. .............................. 11
1.3 HIL testing framework in the general case. ............................... 12
1.4 Hierarchy for test suites, test cases, and test steps. ................. 15
1.5 Schematic overview of the terminology used in this thesis. ......... 16

5.1 Proposed architecture for the test software. .......................... 45
5.2 Class diagram of the logical outline of the automated software tool. 46
5.3 Different stages during the utilization of the automated in-vehicle testing tool. ....................................................... 47
5.4 Verdict table from the automated testing tool. ........................ 52
List of Tables

1.1 Comparison table for test bench environment. . . . . . . . . . . . . . . . . . 10
1.2 Comparison table for SIL environment. . . . . . . . . . . . . . . . . . . . 11
1.3 Comparison table for HIL environment. . . . . . . . . . . . . . . . . . . . 12
1.4 Comparison table for in-vehicle environment. . . . . . . . . . . . . . . . . 13

5.1 Test sequences used for the case study. . . . . . . . . . . . . . . . . . . . . 49
5.2 Results from the in-vehicle tests. . . . . . . . . . . . . . . . . . . . . . . . 50
List of Appendix Tables

1  Used sources for the literature review with their relevance score . . . . . 61
Acronyms

**API**  Application Programming Interface.

**CAN**  Controller Area Network.

**ECU**  Electronic Control Unit.

**GUI**  Graphical User Interface.

**HIL**  Hardware-in-the-Loop.

**IGA**  Independent Guarded Assertion.

**SIL**  Software-in-the-Loop.

**SOTA**  State Of The Art.

**SUT**  System Under Test.

**TTCN-3**  Testing and Test Control Notation.
Chapter 1

Introduction

The first chapter of this thesis will cover the background to the problem of and motivation for automating manual in-vehicle testing. This includes the context in which this report resides, use case to illustrate the need, and terminology to define clear and precise terms used throughout this thesis.

1.1 Motivation

According to data from International Organization of Motor Vehicle Manufacturers [1] the number of road vehicles in use in Europe is strictly increasing from as far back as 2005 up until 2014. For 2016, the European Automobile Manufacturers Association [2] concludes that for the period January to, and including, March a total of ca 560 000 heavy trucks were registered as new vehicles in Europe. Comparing to the previous year, this is an expansion of 12.1%.

More vehicles on public roads generate higher demands for safety, emissions reduction, and reliability. With large production quantities of heavy trucks it is reasonable to deduce that the development and quality assurance of new product lines have a tremendous impact in the later stages of the products’ life cycles.

Not only are the newly produced vehicles, in particular heavy trucks, plentiful, but also modular to fit the diverse customer needs. To add on the complexity of vehicle manufacturing, all the released vehicle configurations have to conform to certain international and company specific standards. It is obvious that this results in an enormous testing process to ensure conformance to requirements. Even though more testing activities are being performed in simulated environments, some testing activities require the actual vehicle fully assembled, referenced to as in-vehicle testing in this thesis. When testing a fully assembled vehicle the actions made by the tester, in order to verify that the vehicle conforms to the requirements, is today mostly manual in nature. This leads to tedious work, driving to multiple locations, pulling and pushing levers, and reading instrumentation in order to excite the vehicle in such a way that it produces the desired output.

Given the large vehicle fleet and that the manufacturers must give a certain level of reassurance that their vehicles are without any inhibiting faults, it is compelling to investigate the possibility to further develop the tools used in order to perform the incredibly important testing activities. As mentioned, simulated environments have already been widely accepted at vehicle manufacturers as accelerated tools for testing. Despite the spread of such environments, the manual testing activities still remain for fully assembled vehicles, which will bottleneck the quality assurance process as it is the last testing environment to be executed. Relieving the in-vehicle testing environment of,
at the very least, some of its inconveniences would therefore greatly impact the vehicle manufacturer, and in the long term their costumers, in a positive way.

One way of improving in-vehicle testing is to expedite its execution by automating some of the currently manual tasks. This could lead to an overall shortened development process. Other benefits of investigating renewed tool support for in-vehicle testing is to incorporate recent research findings from the field of automotive testing into the in-vehicle environment. This allows for more modern approaches to use the in-vehicle testing environment as a resource and not regard it as a necessity. Lastly, investigating automated testing tools is an opportunity to scrutinize the application possibilities of novel testing strategies, studied in more detail in Section 3.2.6.

1.2 Background

Not only does the total vehicle fleet increase in Europe, the automotive industry today uses more and more electronics to control and improve the performance and comfort of automotive vehicles. It is described by A. Chong [3, pp. 13–17] that the value of the mere electronics in cars is constantly increasing and making up a larger portion of the total product value than ever before. As a result, more focus is directed towards development of automotive electronics and automated control of vehicles. This increases the demand for cross-domain competences, but also on the entire process of realizing vehicles. In this process quality assurance will thusly receive increased demands for more accurate and extensive testing, whilst at the same time keeping down the cost for doing so.

To ensure high quality of their manufactured vehicles, the commissioning company, Scania CV AB, utilizes several testing environments. One of them is in-vehicle tests, described in more detail under Section 1.2.2, which requires a large amount of manual labor. This is time consuming and it is therefore interesting to explore the possibility to streamline this environment and as a result improve the whole quality assurance chain. The context in which such a solution will exist is introduced in the following sections.

1.2.1 Industrial Aspect

Designers are not the only party that have to incorporate knowledge in electrification and programming into the overall design and working process, but also the vehicle manufacturer’s quality assurance division. This means that higher demands are set on the tests performed not only on hardware, but also on software and the system in its entirety. Therefore embedded systems in vehicles are tested to a greater extent, both in terms of strictness and completeness. This leads to an exhaustive test task [4].

There already exist strategies which tackle the increased complexity of designing and manufacturing in the context of large industrial companies. However, as it will show, these are not all self-sustaining and there is not a great amount of flexibility and adaptability. One approach to deal with increased complexity is by following the V-model in the design process [5]. This makes design, test, and integration activities clear for the organization and enables an iterative workflow by revisiting modules, systems, and even requirements before the product is released.

All of the previously presented background also applies to the commissioning company, Scania CV AB. Scania CV AB is a Swedish heavy truck manufacturer which instantiates its own V-model as shown in Figure 1.1. Here conceptualization creates the demand for a new product or service leading to requirements that governs its finished behavior or function. From these requirements the detailed designs are made at system-level and then down to module- and component-levels if necessary. Continuing the realization process the design is implemented from the bottom up, meaning that the smallest components are realized first, then modules, and lastly the system as a whole. In Figure 1.1 this is
depicted as the semi-transparent arrows following the legs of the V in the V-model, going from conceptualization down to implementation and then back up to integration test. All throughout the implementation in the V-model, verification is done to ensure that the expected behavior or function is attained. Logically, verification is performed in the same order as the implementation; with the smallest components first, modules second, and lastly the systems as a whole. In addition, integration tests are also performed at an even higher level, integrating all the different systems and guaranteeing that their interaction is coherent and correct. This is illustrated in Figure 1.1 as the solid blue double-ended horizontal arrows connecting both legs in the V-model.

Figure 1.1: V-model at Scania.

The group at Scania CV AB, "System Test Transmission", that originally commissioned the thesis are involved in the quality assurance department responsible for the verification and validation of the transmission and driveline at system-level. At the system-level verification within the V-model, tests make certain that the requirements on functions and behavior are fulfilled for their respective system. In the case of heavy truck development and the mechatronic systems within it, this typically implies that software on multiple electronic control units (ECUs) is verified to control the electronics correctly and in turn result in correct actuation of hardware and mechanics. Common examples of questions that are answered by these activities are:

- Does the brake system give a proper response when the brake pedal is pressed?
- Is the feeling when changing gears satisfactory?

1.2.2 Test Environments

In the case of the V-model design process, performing the tests within each phase requires different environments with specific properties to facilitate proper execution resulting in a correct verdict. The most common environments for system-level verification being test bench, Software-in-the-Loop (SIL), Hardware-in-the-Loop (HIL), and in-vehicle test. There also exist additional environments, such as Model-in-the-Loop and Processor-in-the-Loop, but these are not as common as the previously mentioned examples and are
CHAPTER 1. INTRODUCTION

not implemented at the company site where this thesis is conducted. Environments other than test bench, SIL, HIL, and in-vehicle testing are therefore not discussed any further in this thesis because of their limited influence on results and argumentation. Summarizing tables for each environment, Table 1.1–1.4, are included for ease of comparison.

Test Bench

The test bench is the primary test environment for early testing, both in terms of time in the development process and in the V-model. Used at module and unit tests, test benches measure the performance and verifies single modules or components. Usually a test bench setup consist of actual target hardware, often an ECU, together with input and output pairs [6, pp. 11].

The strongest motivational factor for the test bench is the speed of test execution and the fact that there are no, or very little, dependencies on other systems, which allows for early testing. Since this test environment does not rely on other systems, it is incapable of finding faults caused by interfaces and communication between various systems.

Table 1.1: Comparison table for test bench environment.

<table>
<thead>
<tr>
<th>Test bench</th>
<th>Target to be tested</th>
<th>Applied in V-model phase</th>
<th>Advantages</th>
<th>Limitations</th>
<th>Level of automation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test bench</td>
<td>Software and ECU</td>
<td>Early</td>
<td>Speed and few dependencies</td>
<td>ECU hardware in place with software implemented</td>
<td>High</td>
</tr>
</tbody>
</table>

Software-in-the-Loop

When executing tests inside a SIL-rig, developed control algorithm and software is running against a plant model on a PC. Being completely in the software domain, SIL tests does not necessarily have to be performed in real-time. Instead, the tests are executed at a faster rate. Depending on the setup and framework, implemented scripts can access the set- and get-functions controlling the actions performed and monitoring the achieved state respectively. As a consequence, the input to the plant model can be programmatically induced and the output from it is read and evaluated automatically. This is illustrated in Figure 1.2. More advanced framework architectures exist by applying more input to the control algorithm directly and accessing internal states and variables within both the model and the algorithm [7,8].
One benefit of conducting tests inside the SIL environment is that these tests can be conducted early in the design process, before actual hardware and subsystems have been properly realized. Finding deviations early on allows for faster and less costly changes. It also enables a fast and dynamic dispatch of the tests since the service can be locally set up and possibly scripted, leading to an automated process. Despite its benefits, SIL is not without flaws. As the System Under Test (SUT) is not executed in the real-world context, errors connected to final integration and hardware can later emerge; causing unforeseen and unwanted behavior.

<table>
<thead>
<tr>
<th>SIL</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Target to be tested</strong></td>
<td>Software and system</td>
</tr>
<tr>
<td><strong>Applied in V-model phase</strong></td>
<td>Early</td>
</tr>
<tr>
<td><strong>Advantages</strong></td>
<td>Speed, early execution, and low cost</td>
</tr>
<tr>
<td><strong>Limitations</strong></td>
<td>Hardware assumed not to affect software behavior</td>
</tr>
<tr>
<td><strong>Level of automation</strong></td>
<td>High</td>
</tr>
</tbody>
</table>

**Hardware-in-the-Loop**

Following on the idea from the SIL environment, the HIL builds upon it and executes the test on the actual target hardware compared to a simulated execution platform on the PC, which is the case for SIL. In order to facilitate the activities as close to real-world conditions as possible, a plant model calculates how the real system would behave to the provided input from the tester and from the target hardware, which then sends this back to the hardware. In this way the control algorithm gets tricked into believing that it is actually being run inside the real system. This is depicted in Figure 1.3. This environment can also be made more complex with extended capability to configure, stimulate, and read internal states within the various sub-systems inside.
the HIL environment [8,9].

With similar advantages as SIL, the HIL environment executes control algorithms and code on the contemplated hardware and gives access to the faults that arise with such execution. However, the development of the HIL environment itself is expensive and requires that some hardware is already manufactured, which delays its usage to a somewhat later stage in the V-model. Furthermore, it is not as physically portable as the SIL, although this can be countered by adding communication capabilities to its interface that would facilitate off-site testing. Even though real hardware is used, it is still uncertain whether or not all the defects can be found in this environment. Since some subsystems are still simulated there is no guarantee that the same behavior will be seen in the HIL environment as under real-world conditions.

Table 1.3: Comparison table for HIL environment.

<table>
<thead>
<tr>
<th>HIL</th>
<th>Target to be tested</th>
<th>Applied in V-model phase</th>
<th>Advantages</th>
<th>Limitations</th>
<th>Level of automation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Software, ECU, and system</td>
<td>Middle</td>
<td>Real-world representation</td>
<td>Some hardware assumed to be implemented</td>
<td>Medium</td>
</tr>
</tbody>
</table>

**In-vehicle Testing**

Just as the HIL environment relates to SIL by being more similar to the actual real-world implementation, in-vehicle tests relates to the HIL environment in the same manner. This means that all the subsystems are manufactured and assembled in the same way as the final product, that will be shipped to the costumer, and then tested for incorrect behavior. The tester performing the test takes on the role as the end-user and tries to
CHAPTER 1. INTRODUCTION

excite the system in such a way that it reaches a faulty state or to prove that correct functionality is achieved. This is generally accomplished by driving the vehicle through a wide range of different situations on a test track or on public roads.

Being the last test environment used in the V-model, in-vehicle testing provides the most representative use-case for the tester. This is its strongest benefit, but also leads to drawbacks. As actual products are tested, the test itself must be performed in the later stages of the design process, which may delay the discovery of defects that, in turn, lead to higher costs associated with removing them. Tests are also slower to execute as more preparations are necessary and the SUT is not as flexible as the previous environments. It is therefore expensive, even more so than the HIL environment. By nature, in-vehicle testing demands manual labor and is thus associated with the drawbacks connected to manual operation. Time consuming execution and dependencies on the test driver in terms of experience and meticulousness are some of the prominent drawbacks. Despite being subject to some difficulties, manual testing also presents other features that are not necessarily present in other testing environments. The awareness of a human being driving the vehicle means that unexpected faults may be discovered as the driver is capable of using all his or her senses whilst driving and does not only focus on a particular behavior or fault state.

Table 1.4: Comparison table for in-vehicle environment.

<table>
<thead>
<tr>
<th>In-vehicle</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Target to be tested</td>
<td>System</td>
</tr>
<tr>
<td>Applied in V-model phase</td>
<td>Late</td>
</tr>
<tr>
<td>Advantages</td>
<td>Real-world execution and representative use-case</td>
</tr>
<tr>
<td>Limitations</td>
<td>All hardware and software has to be implemented</td>
</tr>
<tr>
<td>Level of automation</td>
<td>Low</td>
</tr>
</tbody>
</table>

1.2.3 Testing Techniques and Test Methods

Questions such as the two mentioned earlier in Section 1.2.1 can of course be answered by the means of different analysis approaches and methods of testing. Therefore there are distinctions to the fundamentally different analysis techniques and test methods currently put into practice. These are the static and dynamic analysis that together with black-box, white-box, glass-box, and clear-box testing approaches covers the testable space. They are described shortly below for reference.

Static Analysis

In static analysis code, models, and documents are reviewed without any execution of the implementation. This means that logical errors and erroneous usage of commands, within the written application code, are found before runtime [10]. P. Emanuelsson and U. Nilsson [11] also points out that static analysis can catch errors associated with the programming language itself such as resource leaks and in some cases security related vulnerabilities.

Dynamic Analysis

Contrary to its static counterpart, dynamic analysis involves the actual execution of the object to be analyzed. Hence, dynamic analysis can derive properties for a program that
CHAPTER 1. INTRODUCTION

hold for one or more executions [12].

Black-box Testing
As its name suggests, black-box testing regards the SUT as a black box with unknown content. All that is known is the input that is passed to it and the requirements that applies for the SUT together with the expected output for each execution [13, pp. 288–289].

White-box Testing
White-box testing is the opposite of black-box testing in the sense that the source code within the SUT is accessible resulting in a more transparent test design. The tester who constructs the test can thus perform the testing procedure more precisely, e.g. by controlling code coverage and boundary conditions [10]. Despite it being the opposite of black-box testing there are still possible variants with different levels of access to white-box testing. Hence, the following two methods are distinguished.

Glass-box Testing
By extending the definition of the previous white-box testing method, the glass-box specify that code is as readable as in the case for white-box testing, but the software cannot be changed. An everyday example is the case with third party software that do supply their source code, but by tampering with it any support-license is rendered useless [13, pp. 46].

Clear-box Testing
Being the most flexible of the mentioned testing methods in this chapter, clear-box testing enables both access in terms of overview and in terms of editability. Here, tampering with the original software is not restrained in any way [13, pp. 46].

1.3 Terminology
Throughout the thesis, language specific to the field of testing will be used in order to precisely express ideas, arguments, statements, etc. Because the existing terminology does not address all appropriate facets, terms specific for the automotive test domain and this thesis in particular are defined below. This section will also put these terms into their context.

One of the most central concept of this thesis is test. A test in the context of this thesis, and also in the literature [14], is to be considered as a non-atomic concept of activities to check how and if a system conforms to specified requirements. Because tests are generally non-atomic, they are in this thesis defined as test steps, test cases, and test suites. The hierarchical relationship between these three definitions is depicted in Figure 1.4. The test step is defined as the actions and assessment criteria to verify one atomic behavioral requirement. Test steps has thus a pre-condition and a post-condition. Test cases are composed of test steps to cover the demanded test activity of one system function. The test suite is the complete test activity ordered by the system owner of the SUT including one or more test cases.
Another central concept is testing. There is no exact definition of the term, but instead a general notion of what it means. To avoid ambiguity, testing is defined, in this thesis, as the process of:

1. taking input data and stimuli together with an expected outcome,
2. executing the test case,
3. gathering the resulting data from the execution,
4. comparing the outcome with the expected outcome,
5. deciding on a verdict for each test step,
6. and finally delivering this verdict with necessary information for further analysis and documentation,

where stimuli to the SUT are the actions to be performed on the system in order to excite it and input data is other information related to the test, e.g. demands on the situation in which the test must be performed, prerequisites, etc. This process is illustrated in Figure 1.5 as the grayed out zone connected to the thought bubble ”Testing”.

The act of comparing outcome with the expected outcome and producing a verdict is defined as assessment or assessing the test, shown with the thought bubble ”Assessment” in Figure 1.5. After verdict and additional information have been passed on from the testing phase it will, if necessary, be further analyzed then documented, assembled, and traced to the initial requirement(s). This is what decides whether or not a function or system is considered to be functional or dysfunctional. Figure 1.5 depicts the testing definition in the context of the implemented V-model at the commissioning company as described above.
Additional terms present in Figure 1.5 corresponds to the areas which are not part of the scope of this thesis, see Section 2.3. Starting with the requirements, they are defined as compulsory, necessary, or wanted functions or behavior of the SUT.

Test design is the design of the test itself, which decides upon the correct environment to perform the test in, e.g. SIL, HIL, or inside a real vehicle on the road, but also additional aspects such as the necessary test cases and steps needed to fulfill a requirement. Put into general terms the test design answers questions such as why, where, and how.

Resulting from the test design are the input and stimuli to the SUT together with the expected outcome when executing the test. Inputs and stimuli state what the necessary actions are in order to excite the SUT in such a way that the correct function or behavior can be verified. In addition, they also describe in what context these actions are to be taken.

After the testing process, a verdict is passed on and additional information is added if necessary. The verdict itself is the result of comparing how the SUT reacted to the inputs and stimuli compared to what was the expected behavior. Such a result could in some cases be a "pass" or "fail" together with additional information regarding when
this result was achieved and why.

In order to make use of the verdict the test’s data is documented and traced back to the initial requirement for future reference. If the testing phase confirms that a function or behavior is satisfactory according to the expected outcome the function is said to be verified. If this is not the case, changes are made to either the SUT or to the test itself.

The purpose of the test and requirements management is to keep track of all the complete and ongoing tests, their verdicts, and upcoming quality assurance events. This is depicted as the double ended arrow in Figure 1.5 connecting requirements to test documentation and traceability to requirements. A concrete example of this is to ensure that all mandatory tests are performed before the product is cleared for production or sent to customer.

1.4 Use Case

Given all the different available test environments and possible methods, the group "System Test Transmission" has chosen to focus on the HIL and in-vehicle environments performing black-box testing in most cases. The reasoning for this is that the HIL environment provides enough flexibility and an interface to which it is possible to automate testing. This speeds up the quality assurance process and at the same time allowing for accurate scenarios that can be tested. Even though the HIL environment serves multiple purposes it still needs to be complemented with in-vehicle tests in order to provide an environment to verify the drivers feeling for the vehicle and handling. In addition it also provides an important step to minimize the risk that no system is released with faults discovered by driving the truck under intended conditions. Yet another immensely important factor for conducting the manual in-vehicle testing is that it is seen as a necessity because of its capability to verify safety critical requirements.

Because of the necessity of in-vehicle tests, which are associated with some unwanted problems as stated previously, issues connected to manual testing are forced into the testing process. A concrete example of how one such predicament manifests itself in in-vehicle testing is exemplified by the following use case:

There are regulations and legislations regarding heavy trucks’ behavior. For the manufacturers, this translates into multiple requirements. One of these requirements regulate what the highest permissible speed is. It states that a truck shall not reach speeds greater than 90 km/h. Transforming this into a testable statement would result in a test to verify that no torque is applied if the truck’s speed is greater than 90 km/h. The test driver, who is responsible to verify that the current software conforms with this requirements, has obviously more than this requirement to test and will therefore have to follow several test steps in order to ensure that all the designated test steps in the test case are covered. Whilst driving, the driver might reach a speed equal to or greater than 90 km/h when performing another test step, unrelated to the requirement mentioned earlier. Later, the driver notices that the next test step to execute is associated with the 90 km/h-limit requirement. Due to the difficulty for a driver to monitor all the conditions for all test steps at the same time he or she will likely be forced to accelerate the truck to 90 km/h again in order to be confident that the test step has been executed and that the correct behavior is verified. This repetition causes extra expenses in terms of time and resources.

It is evident from the use case that the current employed method for conducting tests is the black-box method. As these tests are performed at system-level it is difficult to dive into the implementation details governing the SUT’s behavior. This is because the intricate relationship between subsystems, modules, and components causes the complexity of any white-box testing method, or derivative thereof, to explode.
Tedious re-testing of single test steps increases the total time needed to go through a whole test case or test suite. These issues are handled in environments such as HIL by automation, where testing is less expensive compared to when it is performed in an in-vehicle environment. One solution would therefore be to transfer some of the features that makes environments such as HIL less expensive to the in-vehicle testing environment. By designing and implementing a novel testing architecture and automated tool for in-vehicle testing some of the burdens connected to the manual execution could be removed or made less inconvenient.

The tool will therefore be able to perform test steps automatically, similarly to SIL and HIL, and thus decrease the effort to conduct the test case. Applied to the previously mentioned use case, such an automated in-vehicle testing tool will identify each testable state and ensure that tests associated with it are executed, assessed, and given a verdict. This way the driver will be informed that he or she already performed the test in an earlier stage and can proceed to the next test step instead.
Chapter 2

Research Design

This chapter covers the goal set out to be investigated and the research questions that comes with this investigation. It also contains the scope and limitations used. Lastly the methods that have been applied, in order to answer the research questions, are discussed.

2.1 Research Goal

The goal of this thesis is to investigate how automatic assessment and verdict generation by the means of monitoring of the CAN-bus can improve the current test situation for manual in-vehicle tests that are performed at a truck manufacturer today. The manual in-vehicle tests verify and validate the software at a system-level. The main reason for performing this investigation is that manual tests in vehicles are associated with several drawbacks, e.g. time consuming execution and dependence on test driver experience. Even so, they are still necessary from a safety and validation point of view as described in Section 1.2. This means that even though there are negative aspects of these types of tests, they are still essential and it is therefore desirable to study the possibility to reduce, or remove, some of the drawbacks.

The use case in Section 1.4 presents a number of seemingly unavoidable issues. However, if this test were to be automated instead the driver would be able to omit accelerating up to 90 km/h again because the automated oracle has already verified the correct, or incorrect, behavior the first time the truck reached a speed greater than 90 km/h. This sort of automated functionality has thus the potential to decrease the required execution time of each test step resulting in a faster overall test procedure. In turn, this gives rise to numerous potential benefits such as a cost effective development process, verification procedure of higher quality, and more consistent verdicts.

This is explored by implementing a novel automated test tool realized as a software program with an architecture designed to counter the drawbacks previously mentioned. The implementation was tested in industrial conditions and then analyzed with respect to its performance and utility.

2.2 Research Question

The formal query, which will produce an answer such that the goal of the thesis is reached, can be formulated as

"How to automate oracle assessment for manual tests in trucks during runtime to decrease the total testing time spent by the test driver, executing the test inside the truck?"
As to break this general and broad question down into manageable investigation fields it is partitioned into different components. There are four general components within the research question: the oracle, inputs and outputs to the test tool, software architecture, and ethical considerations.

The first aspect regarding the assessing oracle, can be addressed by answering the questions:

Q1 What are the State Of The Art (SOTA) methods to automate the oracle assessment process?
Q2 What are the requirements for automatic test oracles from Scania?
Q3 Do these SOTA methods fulfill all the requirements, regarding automatic test oracles, from Scania? If they do not, what are the gaps between the already existing solutions and Scania’s requirements?

The second element to consider is the inputs and outputs of the implemented automated testing tool, which will work as a demonstrator and proof of concept for the contemplated solution proposed by this paper. The inputs and outputs of the automated in-vehicle testing tool will be defined by answering:

Q4 What inputs and outputs, to an automated in-vehicle testing tool, are required in order to facilitate automated testing in an in-vehicle environment?

As for the third facet, concerning architectural design which fulfills the established requirements, it will be probed by answering:

Q5 To bridge any possible gaps between current solution and unresolved requirements, what software architecture satisfies all of these requirements while still enabling the necessary inputs and outputs according to question Q4?

Lastly the ethical considerations to analyze are:

Q6 What are the hazards with automated testing compared to manual testing?
Q7 What, if any, are the preventive measures that can be taken to counter these hazards?

2.3 Scope and Assumptions

In order to fully answer questions Q1 to Q7 in Section 2.2 the scope of the research is to be narrowed down to a more specific task. Therefore, this thesis is to investigate a novel architectural approach to an automated testing tool for embedded systems within a vehicle.

2.3.1 Limitations and Internal Assumptions

The scope of the research is thus limited to the testing phase, as defined in Section 1.3 with the grayed out box, in the V-model, as described in Section 1.2. This means that activities before, after, and surrounding the testing phase are considered out of scope. Examples of these activities are creation of requirements, test design, format and structure of the inputs, stimuli, and expected outcome of the test. Activities of the phases that are considered out of scope are:

- ensure sufficient test coverage,
• model-based testing or other methods and approaches to generate test cases,
• specification of the test steps,
• defining and specifying file formats and types.

Instead, following assumptions exist for the four bullets above:
• test coverage does not impact the investigation of an automated test tool,
• test cases are generated to such an extent that they can be written in a scripting language,
• test steps are formally defined beforehand,
• any format used as input or output is adapted at the company site.

Effectively, this means that the activities preceding the testing phase defined in Section 1.3 are assumed to be already performed. Their deliverables are also presumed to be delivered. The reason for this is that the intention is not to evaluate how an automated testing tool for manual in-vehicle tests is to be adopted at a vehicle manufacturer, but instead to investigate the potential benefits when using such a tool.

Moreover, the format and structure of the verdict and additional information from the testing phase are also disregarded together with the test documentation and how it is traced back to the requirements. The activities that define these phases, which are also out of scope, are:

• summarize test verdicts for deciding on requirement conformance,
• save, document and manage the outcome of the test execution,
• present verdict or documented outcome to increase transparency.

In addition, the only test method that is to be automated by any solution provided by this thesis is the black-box testing method, as described in Section 1.2.3. The reasoning for this limitation is that this is the primary method in use at system-level testing, as stated in Section 1.4, at the commissioning company and will thus cause the greatest impact when presented with an alternative solution for execution.

Lastly, the management of test suites, cases, and steps is not investigated as this does not answer any of the research questions stated in 2.2.

2.3.2 Included Scope

This thesis will, however, cover how an architecture for a software tool is to be designed to facilitate automation of manual in-vehicle testing. This design will give an answer to how to automatically and continuously execute test steps and test cases in an in-vehicle test environment. Furthermore, it will exemplify how to produce verdicts automatically for manual in-vehicle testing. Finally, the thesis will scrutinize possible time savings when performing in-vehicle tests automatically.

2.3.3 External Assumptions

It is not only the tests and the software tool itself that has assumed features. The SUT is presumed to have a multiple testable requirements to some extent. This means that the functions or features that these requirements govern should exist on a distributed ECU network. Effectively, this also implies that the information, that has to be assessed in order to determine a verdict, is observable on one single CAN-bus. In practice the SUT used for this thesis is the vehicle itself, a heavy truck manufactured by Scania CV AB, and more precisely the driveline system within the vehicle.
2.4 Research Methods

In order to reach a sound conclusion and answer the stated research question and subquestions, Q1 to Q7, three different data collection and analysis methods are applied. The three methods being: literature review, interviews, and case study. Each of the methods answer one or more of the research questions. Below is the motivation for each method and its connection to the research questions.

2.4.1 Literature Study

The main purpose of the literature study is to investigate the current SOTA in the field of test automation and more specifically automatic oracles for embedded systems. This method seeks to answer questions Q1 and Q4. For the case with Q1, both the best practice within the research field and the automotive industry will be investigated in order to shed light on the different solutions and considerations that exist within both fields; academia and industry. Regarding question Q4 the literature will be used to find the different key aspects that researchers have found essential for enabling automation of tests. This will increase the chance of a useful result and minimize the risk of concluding that the designed implementation did not take all essential considerations into account.

2.4.2 Interviews

By conducting semi-structured interviews with employees at Scania the thesis can be put into a applicable context, resulting in a more useful and representable results. The reason is that without knowledge of the situation in which the solution will be presented and verified in it is likely that the conclusion falls short due to bad interfaceability or poorly established workflow. Hence, the questions to be answered by the interviews are Q2, Q4, Q6, and Q7. The last two questions are especially important to answer since the experience of the interviewees is of great value with regards to safety since these types of problems are often closely related to practical issues and habits.

The interviews are all conducted at the company site. For each of the interviews, time is allocated and the interview itself is conducted in a private environment, e.g. meeting room, to separate the interview from the regular working environment [15]. The interviewees are all employees at Scania engaged directly with the testing process at the company, but have different roles and tasks. In addition, all the interviewees are part of or directly connected to the same group that commissioned the thesis.

2.4.3 Case Study with Demonstrator

In order to verify and analyze the outcome of the thesis and its implementation a case study is performed comparing the demonstrator against a typical case. The demonstrator is an automated in-vehicle testing tool that performs assessment and generates verdicts automatically for heavy trucks. This means that executions of a test case are analyzed with and without the use of an automated testing tool. The objective of performing the case study is to answer the major research question stated in 2.2. However, it is expected that with proper documentation and monitoring it will be possible to not only give an answer to the time related aspect, but also other influential elements. These can be related to verifying answers to questions Q3, Q5, Q6, and Q7. Even though these are the predetermined fields to be investigated, it is possible that other aspects, consideration, etc. emerge because of the real-world execution.

The outline of the case study will be to run a simulated test case consisting of automated test steps created beforehand. The test case is simulated in the sense that its main purpose is not to verify or validate the vehicle’s functionality, robustness,
 CHAPTER 2. RESEARCH DESIGN

conformance to requirements, etc., but instead to represent a set of test steps that constitute the selected base test steps that are possible to perform with the test tool. An analogy to this is that these test steps are to be considered as ”base vectors” that together span the test space that is reachable with the automated testing tool.

This fabricated test case will also be doped with test steps that are not automated; for instance how the driver experienced gear changes. This injection is made to make sure that the driver has some level of manual labor. There are several reasons for forcing non-automated driver interventions, some of them being:

- It is not desirable to only see how the automated test steps are performing in isolation, but more interesting to see the interaction between the driver and the tool together with what aid he or she is given.

- There will most likely not exist a test case that exclusively constitutes of automated test steps at Scania and this is in all likelihood true for most automotive manufacturers performing in-vehicle tests in general.

- The time measurement and performance comparison, between test activities using and not using an automated testing tool, is not only dependent on the test steps that are automated, but on the test case as a whole (test step execution sequence, ease of execution, etc.).

While executing the simulated test case, the driver will be monitored by camera, screen capture (of the computer screen used while testing), and the author himself. This will be true for both the case of test execution with and without any test automation tool.
Chapter 3

Literature Review

This chapter explains how the literature review was conducted and presents the current best practice of test automation and the constraints caused by the automated test process. Testing is not a unique practice for the automotive industry, but is instead performed in a wide variety of different fields. It is especially popular in software development and the research field of software testing is not a recent undertaking, but has instead grown large with different specializations and focus on different methods, one which is test automation. Because the automotive industry in general, and truck manufacturers in particular, share some of the quality assurance problems with the software field it is of interest to determine if the progress made in the adjacent field is transferable.

3.1 Literature Review Approach

For the literature review conducted for this thesis multiple sources of information is used. Databases are used in order to find and sort among the available published information, but in some cases books are instead accessed through means of referral. The databases used are SAE Digital Library, IEEE Xplore, ACM Digital Library, and DiVa.

The strategy for extracting relevant articles, papers, etc. is to initially search each and every database with the use of broad and general search strings. In order to narrow the amount of hits generated from the search, more precise terms are added to specify the relevant field of interest. Some of the search terms being:

- automat*,
- real-time,
- test*,
- automotive,
- oracle,
- CAN,
- monitor*,
- acquisition.

Together these terms are combined with logical separators and form search strings. One example of such a string out of many used is:
"automat* AND (test OR verification) AND (analy* OR assess*) AND (ecu OR 'electronic control unit')"

where the asterisk is a wildcard character.

The papers used are in average published around 2008, with the oldest publication dating from 1982 to the earliest from 2015. The more recent publication focus mostly on the technical implementation while the earlier sources cover more of the qualitative data collection methods used and some fundamental testing theory.

Whenever a relevant paper is found it is cataloged in the thesis author’s own record of useful references. This cataloguing includes meta information such as author, title, keywords, etc. but also a rating from 0 to 10. The rating represents the usefulness of that particular source with 10 being most useful and 0 being not useful at all. For all the more relevant articles, rated 7 and up, their respective list of reference is studied in order to discover new relevant sources of information. As more articles are collected this way new keywords and search terms emerge and are later used in an iterative search on the databases used.

3.2 Results from Literature Review

Below follows the literature findings regarding the peculiarities, specifications, and thoughts on automation of testing. These form the SOTA in the field of test automation for the automotive application.

3.2.1 Difference Between the Automotive and General Case

In the automotive industry the process of testing is continuously refined to increase efficiency. Therefore a chain of tools is commonly used to facilitate a quicker process. This toolchain is described by A. Bansal et al. [16] with three layers: test management, test implementation, test execution. The test management corresponds to the requirements and test documentation and traceability to requirements phase in Figure 1.5 shown in Section 1.3. Test implementation and execution is thus defined as test design and testing respectively.

It is not only the process that is different for the automotive industry compared to the general development and quality assurance case. There are other technical factors that play a large role in how testing must be performed to ensure its conformance to requirements. Automotive systems rely on real-time and continuous signals and can thus not be handled by conventional testing techniques in most cases [8]. The traditional approach to automatic testing in general is to run test scripts in sequence on machines without real-time capabilities and perform analysis and assessment without an accurate timing environment [8]. Because of this A. Bansal et al. [16] suggested that an ideal test tool should provide real-time and continuous testing capabilities so that the time for analysis after execution is reduced. As a consequence, running such a tool in any PC environment will be difficult as it does not fulfill the real-time requirement and can exhibit latency issues [16].

3.2.2 Model-based Testing In the Automotive Context

One way to implement the verification process and increase the development efficiency is to utilize model-based testing. Because of this and due to its recent popularity in the literature for development processes it is interesting to scrutinize the possibility of merging a solution for automated in-vehicle testing with the model-based testing setting. Although a full incorporation might not fall within the scope of this thesis, some benefits
CHAPTER 3. LITERATURE REVIEW

and desirable features might still be extracted from model-based testing and injected into the mostly manual in-vehicle testing environment.

With model-based testing functional models are created throughout the development process and constantly provide a source for test case generation for the department responsible for quality assurance. This leads to a number of different benefits such as test automation and reuse [8,17]. In their paper, N. Gautam and O. P. Yadav [17] explains that the test reuse exist in three dimensions: along the V-model, through time, and across product lines. The reasoning is that if a model is the base for the development of the product through the whole process chain the same model can be used to produce automated test cases during the whole development period, even as the developers delve deeper into the technical design. By being able to record test cases and comparing previous outcome to current, regression testing is speed up which is the benefit in the dimension of time. Lastly, test vectors and results from different product lines with the same system functionality, requirements, or modules share a common model resulting in decreased amount of product-specific test cases that are forced upon quality assurance personnel [17].

In order to fully take advantage of the benefits with model-based testing or inject some of its benefits into a process not governed by models, the testing tool itself must be compatible with different levels within the development process, such as the V-model. This is true for the first dimension in the reuse argument stated by G. Naveen and Y. O. Prakash [17] as a tool unable to interface with other levels within the process chain will limit its reuse in that dimension. Another use of cross-level applicability is to be able to perform activities earlier in the development chain. By doing so the cost of discovering an error is lesser than what it would have been if found in a later stage. If this is facilitated by keeping the test case constant throughout implementation and integration there is no additional cost for earlier testing [8]. By incorporating or enabling different phases of the verification process the testing becomes streamlined and also more efficient [16].

3.2.3 Automotive Test Representation and Language

When specifying a test case from requirements distinct steps are taken before the test case can be executed and assessed. This is because the requirements are not always written in a form that is formal or as an obvious test case to begin with. This means that they have to be refined in the test design phase in the verification process, as defined in Section 1.3. Refining requirements include specifying prerequisites and surrounding circumstances, but also quantifying behavior into unambiguous statements. After being refined the requirements are in a test representation form, suitable to be used as a blueprint to write the different test steps with.

At this point all the different technicalities are specified together with the necessary actions to be performed in order to excite the SUT in such a way that it is put in a testable state. This is done by a test language that can either be a script in a scripting or programming language or a test specification with a number of manual actions for a driver to perform inside the vehicle whilst driving it.

Both test representation and test language, mentioned above, have been subjects to research in the literature. The findings within both fields are stated in the following two subsections.

Test representation

Any test to be written for a particular application or tool must be expressed with a syntax that enables all the possible scenarios to be tested. In addition, other features, such as requirements traceability and grouping of connected test steps, are also desired. The same types of requirements exist for verification and validation processes in fields other
CHAPTER 3. LITERATURE REVIEW

than the automotive, one being telecommunication. The telecommunication industry have therefore established the Testing and Test Control Notation (TTCN-3) which is a test-specification standard that provides these features whilst at the same time being technology independent. Despite its power to support the previously mentioned features it is limited in its way to describe real-time constraints and continuous signals [18]. Consequently, there have been attempts to expand the TTCN-3 standard to incorporate the additional demands that are necessary when specifying behavior for embedded systems. One attempt being TTCN-3-embedded that aims to formalize requirements between suppliers and car manufacturers [19].

Test Languages

Currently when automating test execution in industry there are two major methods to create a formal description of the test to be run in order to make sure that it is understood by a computer. They are graphical and script based descriptions [16].

Graphical designs have been proven to be able to take the form of state flow diagrams, or more precisely timed automata [20]. With a timed automaton it is possible to express the real-time behavior of automotive systems and requirements, which is difficult or impossible in the case of time independent state machines [20]. Another form that tests can take with a graphical nature is models representing the functionality and behavior of the systems. These models can later be used to auto-generate the test cases [8,17].

The alternative to graphical representation is scripting languages [16]. They present a more conventional way of describing tests with a programming language. The basic interaction towards the test environment is the use of Application Programming Interface (API), which is currently used at Scania for accessing get- and set-functionalities in the HIL environment [21,22].

Both methods have their own benefits and drawbacks. E. Bringmann and A. Kramer [8] states that conventional scripting languages have a poor ability to express the continuously changing signals and complex relationships within automotive systems. However, the alternative to use graphical representation by the means of timed automata also presents difficulties when test cases or test steps become more complex. The transitions and states necessary to capture correct events and states rapidly become extensive [20]. In the case of model-based testing techniques it is necessary to stipulate models early on in the development process and continuously develop these as the product is realized throughout the development process.

Instead, J. Lindgren [23] states that even though it is limited by being a conventional method of expressing tests, a script language, such as Python, is flexible and easy to use when designing tests and configurations to be automatically executed. In addition, it is not as easy as to say that script based programming languages lack expressibility. Rather, a number of different aspects have to be considered. Four of these factors that affect the choice of language, but also any type of test representation, are readability, write-ability, reliability, and cost [22]. Furthermore, it has also been noted by D. Yang [22] that higher level languages like Python do not necessarily lead to shortcomings in terms of performance when compared with other non-script languages such as C, C++ and Java. In fact, the Python written code was instead around half the length of the other alternatives without major performance losses [22].

3.2.4 Exotic Algorithms for Vehicle Testing

With more computational capability and an increased possibility to extract data from embedded systems comes the means to utilize more exotic algorithms and methods to perform analysis on the SUT. These are mainly learning or training based algorithms together with more advanced mathematical tools that can automate analysis. It is
CHAPTER 3. LITERATURE REVIEW

because of this interesting to investigate the methods that have been proven effective in automotive testing as it is desirable to facilitate these very algorithms and methods, but not necessarily implement them, in an automated in-vehicle testing tool.

One of these exotic methods is the use of machine learning algorithms for classifying or cluster objects [24]. Another algorithm is neural networks. The genetic algorithms are trained against data and can later classify or make decisions based on the trained network given proper data. This is suitable for large data sets with complex internal relationships, e.g. determining if a vehicle demonstrate the correct drive-ability behavior given a large number of sensor data [25]. A third possibility is to update the test itself and its test vector with the use of evolutionary algorithms. In this way the tests are continuously and automatically being updated to increase performance according to the specified fitness evaluation function [26].

In addition to the more exotic methods that can be applied when testing automotive systems, statistical analysis can also be performed on the continuous signals in order to ease or remove the manual signal analysis. One such proposal is presented by T. Schmidt et al. [18] where output signals are compared with a reference signal and a difference vector is formed stating during what time intervals the output deviated from the expected output. Here, it is also possible to use a certain amount of slack for the occasions where the conformance is not strictly necessary. Another take on using statistical analysis is made by M. Conrad et al. [27], who also use signal comparison and slack. M. Conrad et al. [27] does not stop at only performing signal comparison, but also make use of signal processing techniques to determine a more representative analysis of the behavior of continuous signals.

3.2.5 Monitoring Vehicle State and Data Acquisition

Before any test may be initiated, the SUT must firstly be monitored and secondly data from it must be extracted. It is no novel task to extract data from vehicles through the use of on-board informational networks and buses. This has already been addressed by advanced vehicle diagnostic systems. Nonetheless, a testing oriented solution have other requirements to uphold. Some of the demands on such a solution as stated by K. V. Gorder et al. [28] are:

- quick and easy software and hardware setup,
- flexible,
- modular design.

Monitoring the current state of a vehicle is easy today, as its subsystems are distributed and connected through communication buses, usually CAN [29]. As the subsystems pass information amongst themselves, measured external states, e.g. vehicle speed, and internal states, e.g. engine mode, become available on the bus. This means that it is possible to monitor and obtain a partial, but valuable, representation of the current state of the vehicle by monitoring the communication bus [30].

Discussed by A. Kane et al. [30], issues that arise with such a system are intent approximation, representation language, system mapping, and observability. The first issue is regarding how a test should be designed in the first place given that the SUT acts as a black-box and the system as a whole has more complex functionality than that which is tested. It is in these cases difficult to predict the expected behavior, at least without a system model, given the test input as other systems might interfere with the same variables that are monitored in the test. Problems with the representation language is concerning how to express a test so that it will capture the correct behavior without being too expressive and complicated. As mentioned in Section 3.2.3 there
are a number of possible choices where script languages and timed automata are two possibilities. The third issue deals with how to map the current system state to the monitoring implementation. This includes several sub-problems such as sampling difficulties, continuous to discrete conversion, modeled behavior for when utilizing a HIL test rig for example. Lastly, problems regarding observability arises when the necessary data to perform a specific test case is not available on the observed communication bus [30].

3.2.6 Independent Guarded Assertions

The current practice in automated testing in the automotive industry is by sequential execution of test scripts. This means that some stimuli is provided to the SUT and then followed by a number of assertions and ending with stimuli to wrap up the test if necessary. This can also be viewed as three different phases within one test case or even test step, depending on its extensiveness, named: setup, testing, and tear down phase [21]. By separating the stimuli from the assertion it is instead possible to better represent the real-world situation, reduce the time to perform the test execution, and improve the defect detection [21].

T. Gustafsson et al. [21] calls this approach IGA. This is based on the fact that every test step can be viewed as a guard, indicating in what state the SUT must enter in order to be tested, and an assertion. When the guard is fulfilled comparison and analysis is performed on input data in order to produce a verdict. The actual independence is a result of two features. The first being that test steps are independent of the SUT since they do not influence or change it. The second being that test steps are independent of each other and does not have to wait for other test steps to finish or produce a result in order to generate their own verdict. This means that test steps must exclusively be confined with read-only actions on the SUT. This further lead to the fact that test steps can be run in parallel. Another consequence is that test steps with similar or identical setup and tear-down phases will run simultaneously and thus saving time by not having to excite the SUT all over again [21]. In addition, it is shown that the same approach is valid for recorded data which enables analysis after the test case have been executed, if data was recorded during the test. Continuing on these findings, timed automata have been used to represent these IGAs [20].

3.2.7 Test Oracles

Most common in the software testing context, test oracles is not a novel nor poorly researched field of study. Instead it has been probed and analyzed by many publications, as stated by E. J. Weyuker [31]. The term test oracle is defined as a mechanism for generating the expected result necessary to perform the test assessment [32, 33]. Another definition is that an oracle is the mechanism that determines the correctness of the input and output data from the SUT, which implies some expected result for the assertion [31]. In order to produce an expected output many different strategies exist including manual verification, earlier versions of the SUT, and separate implementation [32, 33].

Even with the numerous amounts of different oracle strategies it is not certain that the SUT is testable by these methods alone or without modifications to them [31]. Therefore careful design of said oracle must give an accurate response to the provided stimuli. This conundrum is referred to as the oracle problem [30].

A system is considered non-testable if it can fall into to any of the three following classifications [31]:

- the SUT itself is designed to determine the correct answer,
• the output of the SUT is too plentiful to check,
• or there is a misconception as to what the correct output is.

To deal with non-testable systems there are some suggested solutions to test some parts of the system or partly test it. To tackle the first two cases listed above simplification of the data is often used. By testing on simple cases and extrapolating, correctness of the systems as a whole can be determined with reasonable certainty while still retaining a justifiable cost in terms of time and effort [31].

Another approach is to recognize the oracle in use as partial. One way of delimiting the oracle’s scope is to limit it with two boundaries. The first boundary being that the oracle does not cover all of the properties that exist in the SUT. The second being that the oracle give approximates bounds which the SUT must fall within instead of specifying exact behavior. Because of this it can be considered impractical to define complete and formal specification of the behavior for a complex system, such as a modern vehicle [30].

Further justification for simplification being a useful solution to an oracle problem is that a larger number of variables monitored does not necessarily mean a significant increase in performance, in terms of fault detection rate, for the oracle [34]. Other related investigations to the relationship between performance and complexity of oracles also points towards the case that over-complex oracles only have a slight increase in their fault finding rate compared to less complex oracles [35].

3.3 Discussion on Literature Review Results

In this section the results from the literature review are presented in relation to the respective research questions. The findings are presented thematically with each theme in its own section.

3.3.1 Analysis Performed During Automotive Testing

When performing testing activities for the automotive application, some sort of analysis will at some point be conducted in order to conclude on a verdict. The current best practice requires this analysis to be able to handle real-time aspects and at the same time be continuous in its execution. By continuously doing analysis the testing becomes more effective as less analysis has to be performed afterwards with recorded data. This is related to research question Q1 by implying that continuous real-time analysis is used and needed in the oracle assessment. This also leads to demands on the inputs to an automated in-vehicle testing tool. Hence, continuous real-time requirements are added to the inputs and outputs from research question Q4.

As stated in Section 3.2.5, the most essential analysis does not require a total system awareness in order to conclude on a verdict. Instead, partial information is sufficient. This limits the amount of necessary inputs to an automated in-vehicle testing tool to observing the CAN traffic in order to perform its assessment. Thusly, adding to the research question Q4.

3.3.2 Integration in the Quality Assurance Process

In order for new testing tools to be useful they have to be proven integrable in the quality assurance process. The quality assurance process is in constant change with one future outcome being the adoption of a model-based mind-set. By both facilitating such a context and trying to duplicate some of its benefits, an automated in-vehicle testing tool will ensure that the different levels of reuse, mentioned in Section 3.2.2, are fulfilled.
CHAPTER 3. LITERATURE REVIEW

This connects to the current SOTA methods for automation of oracle assessment and research question Q1.

3.3.3 Choosing Test Language

Script based development and representation for test cases seems to be the most general approach currently applied. The best practice also suggests state machine inspired representations, but ultimately this can be rewritten as code or scripts if the latter two are properly defined. For research question Q4 this means that test steps from test developers are handled by the automated in-vehicle testing tool as executable scripts.

3.3.4 Extensibility of Testing Tools

As shown in Section 3.2.4 many different techniques exist for conducting completely new types of analysis of vehicle data. It is therefore also feasible to assume that similarly novel techniques will emerge with time as the vehicles themselves grow ever more complex. Logically, any tool implemented must therefore also be able to allow for new techniques that may not be implemented today. By facilitating extensibility it should therefore be possible to create today’s test steps, but also test steps in the future for functionality that does not currently exist. This would create a future-proof design, increasing the likelihood of it being adopted at the company site. Studying the current research it is evident that writing executable code is the de facto method for implementing new analysis techniques. This further stresses the importance of executable test steps, in the form of scripts, as previously stated in Section 3.3.3.

3.3.5 Issues with Intent Approximation

For strictly observing systems the problem of intent approximation become prominent. This is especially true for an in-vehicle environment as the driver will be in full control of the vehicle and it will therefore be difficult to read his or her intention. However, using IGAs would remove this difficulty as stimuli and assessment are separated. The driver’s intent is after partitioning not of interest anymore as guard and assertion will instead always try to resolve whether or not the SUT is in a testable state and if so, what the verdict should be. This translates well into the in-vehicle test environment and is a concrete example, in the context of research question Q1, of an implemented SOTA method to automate oracle assessment.

3.4 Conclusion of Literature Review

The solutions that are presented in the literature mainly focuses on virtual environments, such as HIL, and how these can be made more efficient. The in-vehicle test environment is in this sense neglected even though it will most likely remain crucial in the quality assurance process.

Connected to the literature review are research questions Q1 and Q4. In the case of Q1, a summarizing answer is that models from the model-based development would generate test cases in the form of an executable scripts. Scripts can then be automatically executed and assessed. Following the scope from Section 2.3, only the execution and assessment is relevant for Q1. In order to increase the fault detection rate, IGAs are used to expand the test window from being a narrow sample, after predefined stimuli, to cover the full time span. The analysis performed on the collected data can be based on anything from simple logic satisfaction to more complex techniques such as neural networks or signal processing. This is expedited by accessing the CAN-bus of the vehicle.
Concerning research question Q4, it is clear that the partial information sent over the CAN-bus is satisfactory. When extracting information from the CAN-bus it is still important to make sure that signals have comparable real-time features, e.g. timestamps.
Chapter 4

Interviews

In this chapter the interviews performed to capture the actual situation and demand at the commissioning company is presented. Together with the result, analysis is performed and information connected to the research goal is extracted.

4.1 Interview Design

The choice to use interviews as a data collection method is mainly based on the fact that the research question stated in Section 2.2 demands a probing data acquisition technique to find an approach to the usual process of testing. Another purpose of using interviews as a data collection method is to profit from the existing knowledge and expertise of practitioners within the field of testing. In order to fully take advantage of the interviewee’s proficiency the interview is of an semi-structured type. This way the input and reactions from the interviewees are captured while still ensuring that the relevant topics are probed. This leads to the possibility of cross referencing and comparing answers given as mentioned by D. Cohen and B. Crabtree [36] for greater understanding of the true need and expectations existing at the commissioning company.

As the sample, from which the interviews are performed on, is small it is necessary to extract information of high quality instead of larger quantity, which is according to J. Laforest [37, pp. 1] a feature of the semi-structured interview. This is further supported by the fact that there is no current solution to the research question in Section 2.2 that can be analyzed by the respondents.

Used as key informants, as suggested by J. Laforest [37, pp. 2], the employees from the commissioning company pose as the sole interviewees for this data collection. At the company site all of the interview participants belong to the same work group, save for one participant who have a managing and logistics role, connected to that very same group, in the quality assurance division for software in the driveline ECUs. The rest of the group are involved in the actual verification and validation process with activities such as test design, testing, and documentation. A total of six interviews are held ranging from half-hour to an hour long. It is reasonable to argue that an increased number of participants would generate a more representative result, deepening the understanding of the problem, without necessarily causing data saturation [38, pp. 239]. However the number of employees connected to the probed field is limited and as the results will show, many of the answers and responses did overlap with this set of interviewees which lead to the conclusion that the set of respondents is sizable enough. The interviews are held at the company site in meeting rooms generally close, but still separate, to the normal working environment, as suggested by L. S. Whiting [15]. For all interviews an interview plan is used to ensure sufficient coverage. During the interviews notes and
audio recordings are made.

### 4.2 Data Analysis

As a first step in the data analysis all recordings are transcribed into plain text. In order to sort and categorize the answers given during the interviews thematic coding is then applied to the transcripts resulting in a set of themes that encapsulates all the answers. The emerging themes, and with some sub-themes, are:

- relation to testing/activities or duties,
  - test developer,
  - test engineer,
  - test manager,
  - test leader,

- problems or obstacles that exist today,

- test execution/analysis/program support existing today,
  - tools,
  - analysis methods,

- potential benefits or gains with automated in-vehicle tests,
  - resource related,
  - performance related,
  - safety related,

- expectations,
  - development and maintenance,
  - functionality,

- test cases that can be automated,
  - simple,
  - complex,

For clarification purposes the different roles, listed as sub-themes, can be described with the following explanations. The *test developer* takes mainly on the role as test designer choosing in what environment certain tests should be performed in, but also software developer writing test scripts for HIL and system analyst. *Test engineers* performs the in-vehicle tests using both knowledge and experience to deduce verdicts for the behavior of the SUT, but also carries out analysis when deviations are discovered. As *test manager* the role becomes supportive, but also demands a comprehensive view of both tests performed, systems that are tested, and the process in which these activities exist. By coordination within and outside the interviewed group, the *test leader* assumes logistic responsibility including planning, prediction of future events, and summarizing the results in order to verify or validate the SUT.

With these emerging themes it is possible to both connect the interviewees connections to the testing process with the answer they provide. In this way true relation to the need for a solution can be realized together with what such a solution proposal should include and to whom it should be provided to.
CHAPTER 4. INTERVIEWS

4.3 Results from Interviews

The findings from the interviews are many. Some are directly relevant for the research goal of this thesis and some are more vaguely related to the stated goal. By going through each of the themes the respective discoveries for that theme will be presented and analyzed below.

4.3.1 Problems Existing Today

Going through the many answers given, it is clear that there does not exist a sole solution for all problems mentioned. The problems vary from being process oriented to specifics when conducting certain test steps. With issues connected to the implemented process it is mentioned that handover and responsibility in between groups makes testing inadequate and not connected to the development as desired.

Flexibility is another reoccurring issue brought up. Today tests are ruled by static documentation, leaving little adaptability for what is possible to do whilst performing in-vehicle tests, but also in what manner tests should be carried out. This can also lead to conflicts between test steps that becomes contradictory to one another. This is further amplified as more modular designs are to be tested and leads to conditional test specifications in order to capture the different expected behavior depending on the current configuration.

Issues regarding uncertainty are also present. This means that it is not always clear as to what exactly have been tested against what, which in turn lead to duplicate testing activities only to make sure a set of functionalities have been tested at least once. In long term this means that the test coverage is unknown in some cases.

Some technical factors also affect the process negatively. As an automotive system relies on continuous and real-time signals it becomes hard for a human test driver to both pay attention to the surrounding environment and the current and previous vehicle states. Some state changes are even so fast a human cannot monitor them without the help from digital measurement and diagnostics tools. Other behavior are slow as to the point that it might be difficult for the driver to continuously monitor the current state throughout the test.

The greatest concern is, however, connected to resource limitations. This is reflected in the notion that there is not enough time to perform as many analysis activities as each individual would like to. These marginalized analysis activities include both analysis during the test execution and after the test have been performed with logged data. In addition, the amount of testable hardware is also limited, which further decrease the possibility to perform all necessary testing tasks within a given time frame.

4.3.2 Existing Support

There is a myriad of different tools used whilst testing. They exist in the hardware and software domain as well as across them both. Generally computer based tools are used and their purpose is mainly to capture, present, and log data for post-analysis. Despite the amount of existing support there is no real synergy amongst the tools and some overlap in their abilities to aid analysis and verdict generation.

The methods used to find defects can roughly be divided into formal and informal methods. Formal methods being clear instructions on how to operate or stimulate the SUT in order to reach a verdict. Informal methods are instead based on the intuition and experience of the driver when performing in-vehicle tests. This will range from making sure the feeling of driving the vehicle is satisfactory to discovering erroneous behavior that had not been anticipated before commencing the test.
4.3.3 Benefits with Automated In-vehicle Tests

As resources are among the most pressing issues the envisioned benefits of automating manual in-vehicle tests touched upon these issues as well. For the resource related benefits the idea is mostly that automation will lead to a shortened time in the actual vehicle relieving resources in terms of manpower and hardware available for other quality assurance activities. This is for example predicted with a decreased repetition of activities which is currently an issue as stated in Section 4.3.1.

Performance related advantages are conceived as the ability to catch behavior that occur quickly or over very long time. This leads to a richer and complete analysis taking into account more of the available data than what is currently done. It is also believed that it will become clearer what tests are actually being performed as an automated execution will have a formal declaration of when tests are, or are not, executed and what, if any, verdicts was produced as a result of the tests. Additionally, the driver is thought to be able to become more creative in his or hers exploration of the SUT as a result of both having more time at hand as stated in the previous paragraph and having number of states covered automatically which frees his or hers attention to other unforeseen defects otherwise undiscovered.

Lastly safety related benefits is brought up. The reasoning is that if the driver is given more attention to operate the vehicle, letting a tool automatically assert tests, then he or she will decrease the risk of entering a hazardous situation. Especially evident would this advantage be if the driver is not forced to investigate and analyze the vehicle state, whilst in traffic and on public roads, by the means of looking at a computer screen inside the cab, but instead focus on the road and traffic condition.

4.3.4 Expectations

The expectations connected to the development and maintenance of an automated testing tool is that it should be easily integrated into the already existing set of tools used. This means that it should use as much free and non-proprietary software as possible. It also means that the source code in which the automated in-vehicle testing tool is developed and written in should be a language which is well established. In the case of the commissioning group, System Test Transmission, this implies using Python as this is the language that they are most familiar with.

As for desired functionality of any automated testing tool the requests range from general behavior to technical details. The areas concerned are the presentation of information during and after test execution, the necessary information to perform analysis, and interaction with the driver. A clear and visual interface to the driver is seen as a natural presentation format. This includes a non-intrusive design and continuous updates on the test progress. The information necessary to perform the test is said to be the variables or signals that produces the generated verdict together with other metadata such as location, the responsible development team for the SUT, and data from before and after the verdict was generated to put it into context. When it comes to the interaction with the driver, it is requested that suggestions are to be made on the stimuli that the driver needs to provide in order to put the vehicle in a testable state. In addition, information regarding the tests that have and have not yet been tested should also be provided in order to give the driver a sense of the progress he or she is making. This is envisioned as a dialog between the driver and the tool, but what medium this dialog should be conducted over is not clear. Extending on the thought of automation some suggestions emerged regarding automatic test documentation generation and the possibility to log data for more advanced and in-depth analysis after the test have been finished in-vehicle.
4.3.5 Test Cases to Automate

As different types of systems are tested, there exist various test steps to verify them. Some of them are considered simple and include checking if certain variables have been assigned correct values given a particular vehicle state. These types of tests are today considered tedious and time consuming, but they are still of course carried out.

Other types are considered complex and can include more or less difficult calculations or estimations. With some tests the signal to be analyzed needs first to be conditioned in order to be of value and in some cases, i.e. with accelerometer data, this is unfeasible to perform in-vehicle when already having limited resources. In those cases the test is instead moved to another test environment to free resources, but might instead mean that the quality of the test itself is decreased as the supplementary environment might not represent the real-world implementation.

4.4 Discussion on Interview Results

It is easy to identify that the problems present exist in many phases within the testing process. The existing problems are also divergent. Some obstacles regarding the implemented process are for example very different from the issues concerning the tool support for in-vehicle tests. It therefore becomes obvious that it is not feasible for this thesis to cover and solve all the mentioned problems. Instead, it should stick to the stated scope in Section 2.3. The process oriented dilemmas are therefore outside the delimitation of this thesis. This is also true for how the test steps are to be designed, handled, and updated according to Figure 1.5. Some expectations from the interviewees are also defined as out of scope. These are the expectations regarding documentation generation and the possibility to facilitate offline and more in-depth analysis after the test has ended.

In order to answer research question Q2, following requirements on an automated in-vehicle testing tool are identified from the interview results above:

- R1 Familiar programming language and environment for writing test steps.
- R2 Continuously updates the current state of the vehicle as the test is progressing.
- R3 Communicate the current testing progress made to the driver.
- R4 Enable visual interaction with the driver.
- R5 All execution is to be performed inside the vehicle (in-vehicle environment).
- R6 Test steps shall be able to be bolted on to the test tool.

From the interviews research question Q4 can be complemented with the following requirements:

- R7 It shall be possible to access the CAN signals that are responsible for the generated verdict.
- R8 The solution shall be extensible in the meaning that it should be possible to provide additional meta-information, as mentioned in 4.3.4, alongside the verdict, inputs and outputs, stated in specification R7 above.

The signals referred to in requirement R7 are the signals within the CAN messages sent over the CAN-bus. Since multiple signals can exist within a single message any automated testing tool must be able to extract all relevant signals from each message. Since no other information is considered necessary in order to perform the currently
employed test steps an automated in-vehicle testing tool will be limited to the observed signals over the CAN-bus.

Moving on to research question Q6, the most prominent hazard is connected to situations when the test driver have to focus on the computer screen whilst driving. This endangers the driver himself, but also the surrounding traffic, people, or objects. This means that a tool requiring the drivers attention will always have the potential to become a hazardous element in his or hers driving environment. With the automated testing tool it is believed that the attention a driver has to put aside for conducting the ongoing test step can be decreased from today’s amount by automatically providing the needed information. In this way driver interaction is decreased and should result in more available attention for the driver to perform other tasks with, i.e. observe and act on the current road and traffic conditions. This answers research question Q7.

4.5 Conclusion of Interviews

From the interviews, eight requirements, R1–R8, are formed in order to answer research questions Q2 and Q4. In addition to the requirements a potential hazard with the in-vehicle testing environment is elucidated. Whenever a tool demands the drivers attention it poses as a safety risk, since this can ultimately endanger both the driver and surrounding people or objects. This corresponds to research question Q6. If the tool would display necessary information without forcing the driver to perform any additional action, this hazard could instead be decreased. This would be the preventive measure to be investigated in research question Q7.
Chapter 5

Case Study with Demonstrator

This chapter describes how the practical work is implemented. As for a typical product development or engineering design project it describes the stages carried out in the development process. The designed demonstrator is an automated in-vehicle testing tool allowing test developers to write the test steps as Python scripts. The automated in-vehicle testing tool then executes these scripts continuously and in parallel when connected to a running vehicle's CAN-bus. Results from the case study utilizing the developed demonstrator are presented towards the end of this chapter.

5.1 Difference Analysis Between Literature and Requirements

The first requirement identified from the interviews, R1, states demands regarding the technical implementation. Knowing that the current practice of scripting tests for the HIL environment is using Python, it is reasonable to conclude that the implementation should also be scriptable with the same language in order to flatten out the learning curve. Python is also discussed in the literature with both benefits and drawbacks being presented in Section 3.2.3. However, it is stated that it is not feasible to give each possible language a simple score as a more complex analysis must be performed. Because of this it is determined that Python is a sound choice as a programming language for the implementation as a whole due to maintainability and acceptability reasons. Python is also chosen as the scripting language for the actual test steps to be performed inside the vehicle because of readability, writability, and cost reasons. The readability is considered high for Python as a scripting language since it is a language without many unnecessary symbols, but also because the commissioning company has a lot of experience working with it. The same applies for the writability reason. Since Python is not a proprietary language it supplies the source-code free-of-charge which is highly beneficial for a bigger company as it is not necessary in this case to attain licenses. Further benefits with Python as a base for the implementation is that there is a multitude of packets and extensions, also free-of-charge, that aid both development and maintenance of the implementation. This is compared to other alternatives such as Matlab that has packets and extensions as well, but in many cases these are not free-of-charge.

Continuing with the second requirement, R2, from Section 4.4 it is imperative to facilitate continuous updates on the test progress to the driver. This said, it is not necessary to conduct all the tests in real-time, only to be able to test real-time behavior. This has already been solved by environment such as the case with HIL. This type of solution also already exist at the commissioning company with the use of client-
server architecture for their communication and control over the HIL environments. Nonetheless, this existing solution relies upon continuous polling in order to observe the desired behavior which can in unfortunate scenarios lead to data being missed or timing issues.

Both requirement R3 and R4 suggest that a graphical user interface is to be possible to present the current results. This is, by using a language such as Python, not an issue as long as the software architecture is made modular and well defined.

As for requirement R5 it is reasonable to conclude that the solution will be implemented on a PC as they can easily be brought to the in-vehicle environment. This as also been the solution of similar projects and has proven to be a solid design [28].

Requirement R6 calls for a modular architecture allowing for different use cases. This means that it is expected that testing will be performed with different test steps and this should be reflected in the sense that different scripts shall be able to be added or removed from the testing software as needed. One solution to this is the same as mentioned previously when using a client-server architecture. This way test steps can be clients connected to a server that holds an arbitrary amount of test-clients.

The last two requirements, R7 and R8, touch upon the ability to attach more information regarding what is tested and the signals used in that test. They also state that there should be support for adding additional information, other than which is used today, to the output.

5.2 Ethical Considerations

When implementing a software tool to support personnel in the quality assurance process it is important to consider the impact that is made by the new tool. In this case the importance becomes even more evident as it is to be used in the cab whilst driving. The two major considerations are fault injection and intrusiveness.

Fault injection can occur if the testing tool affects the system in such a way that a fault is injected that would otherwise not appear. In the case of an in-vehicle environment this is to be viewed as a serious risk as injected faults may lead to dangerous failures for both the driver and people in the vehicle’s vicinity. This is to be tackled by only allowing the tool to read from the CAN bus of the vehicle and perform its test on an independent PC. In this way the act of monitoring the CAN signal does not inject any faults and if the tool itself malfunctions it will not affect the SUT as it was not be aware of the tool in the first place.

The level of intrusiveness also affect the safety of a in-vehicle testing tool. If the tool requires too much of the drivers attention he or she may not be able to detect the surrounding environment and therefore put him- or herself in danger, as well as anybody being nearby. To counter this care has to be taken in three aspects:

- what is shown to the driver,
- what the driver has to do,
- what the driver could do.

If too much information is shown or the information shown is hard to read this will force the driver to use more attention in order to make use of the information. The same applies for the actions that the driver is forced to take. Actions that are to be done during setup and tear-down are allowed to be more intrusive as they can be performed before going out on roads etc. whilst actions during the actual driving have to be minimalistic in order to not interfere with the driving. The third aspect deals with the fact that a tool that allows for many different operations may cause the driver to steer his or her
attention to the tool exploring the information and functions available. This might be a result of his or her preference or because whoever designed the test for the tool is aware of the ability to extract more from the tool and hence may write the test, without understanding the consequences, in such a way that this will become mandatory for the driver. The idea is therefore to develop a minimalistic interface only displaying the most crucial information without the possibility of interacting with it, with the sole exception being to turn the tool off.

5.3 Additional Considerations

Some considerations exist in the literature that cannot be tied to any of the requirements that the respondents from the interviews mentioned. One such consideration is that not all signals have to be monitored in order to perform a test. In fact, as presented in Section 3.2.7, an increasing amount of monitored states and signals does not correspond to an equally increasing fault detection. This means that it is possible to partly monitor a system and still find faults effectively without too much effort. What is, however, needed is to observe relevant input and output to the system.

Another factor is that the automated testing tool should be able to interface with other phases within the quality assurance process, such as the V-model, in order to fully take advantage of a flexible and effective test environment. In practice this means that there should exist an interface that is compatible with the SIL and HIL environments as these are the existing system-level test environment in place at the commissioning company. As both today are compatible with Python scripting languages in one way or another, this becomes an additional reason for utilizing the Python scripting language for test step creation.

Taking into consideration that the implementation is to run inside a vehicle, in-vehicle test environment, it is also reasonable to imagine that all the inputs to the SUT are performed by the driver. This means that whatever test step is running it will not be able to provide stimuli as it sees fit, but must instead comply with the actions carried out by the driver. This leads to a dilemma regarding how to discover when the driver is trying to execute a test and asserting the result. The *intent* and *result* problem have already been probed and one solution is to split both problems into separate steps by the means of IGAs. With IGAs the conditions for a particular test is instead always checked and will therefore not be overlooked because of misinterpretation of the driver actions. As a result, the *intent* problem is with this solution solved. As for the *result* problem, it can analogously be addressed by IGAs since when the intent is clear the result is a mere data retrieval and analysis task. The research performed on IGAs does however not fully investigate the possibility to assert real-time requirements with this solution, which is a must for any automotive testing tool.

The read-only aspect of monitoring testing tools have also been discussed in the literature, but is not a clear subject brought up in the interviews. Read-only does not only imply safety related benefits, which are discussed in greater detail in Section 5.2, but also enables parallelism. This is because truly parallel tests must not be dependent on each other, otherwise there might be downtime when one test step is waiting for another to finish or come to a conclusion. Even if the test steps are not directly dependent of each other, stimulating the SUT infer another type of dependency. If a test step changes the state of the SUT then other test steps are dependent on the first test step to change the SUT back to its original state or they are instead forced to provide their own stimuli to force the SUT into their own testable state, which in turn result in the same dilemma. This can be seen as a third independence level adding to the two mentioned earlier in Section 3.2.6.
5.4 The Demonstrator

The designed demonstrator is an automated in-vehicle testing tool realized as a software program to be run on a PC, preferably a laptop. How to use it and what groups of people are affected by its designed is described in the following subsections.

5.4.1 Usage

With the contemplated use at system-level testing and verification of embedded and distributed vehicle systems, the tool will in the scope of this thesis be used by the group "System Test Transmission" at Scania CV AB. In their quality assurance assignments, "System Test Transmission" performs today manual in-vehicle testing activities to verify that their products conforms to requirements.

When using the developed automated in-vehicle testing tool a limited number of steps are necessary to perform. The first step is to produce implemented test clients that are to be executed during the testing of the SUT. This involves using the test steps developed for in-vehicle testing and writing them down as Python scripts using the general syntax demonstrated later in this section. This is normally done by the test developers.

After the test developer has written the test steps, as Python scripts, the test engineer can include the desired test steps to the automated in-vehicle testing tool. This means that each test step that has to be executed and that have been written as a script will be assessed automatically and continuously whilst the test engineer drives the vehicle around the test track or on public roads. As for the test steps that are not applicable for the test suite to be performed, these are simply not included for the next executions and will therefore not perform their unnecessary assessment.

At this point the test engineer brings the PC, laptop, with the automated in-vehicle testing tool out to the vehicle to be tested. By starting the vehicle, connecting the PC to the vehicle’s CAN-bus, starting the third party CAN interface tool, and launching the automated in-vehicle testing tool itself the test engineer can begin testing the vehicle. As the automated in-vehicle testing tool will monitor all, from the test steps, required CAN signals continuously, the driver can focus on handling the vehicle and follow the test specification document. When finding him- or herself in an appropriate situation, the test engineer can examine the verdicts generated from the automated in-vehicle testing tool in order to determine if a test step already been executed.

5.4.2 Stakeholders

There are mainly two different stakeholders for the automated testing tool: test developers and test engineers. However, other less obvious stakeholders exist. These are the tool owner and maintainer, which does not necessarily have to be two separate entities.

Test developer

The test developer is, as in the case of tests performed in the HIL environment, responsible for writing and maintaining the test steps and their scripts. Therefore, the test developer will interact with the API and be affected by its capabilities and limitations. Since the entire implementation is realized by Python, the test developers at Scania CV AB are capable of writing the correct syntax and algorithms with the language as they are already familiar with it. This is considered as the primary consideration.

Second to consider is the access to the messages sent on the CAN-bus. Today all signals have been given strict definitions including, but not exclusively, name, length, range, unit, etc. It is therefore important to have the test developer focus on the algorithm of the test step and not intricate details such as what CAN message ID is for a certain
signal to be monitored. This is accomplished by the use of CAN database files which allows the test developer to use familiar names and attributes for signals without knowing the exact implementation of these.

To show how test steps are written to become test clients and scripts the following pseudo code is provided showing how a test step defines its monitored signals and returns them to the requesting server. This is performed in the setup phase.

```python
monitored_signals = [ENGINE_SENSOR->ENGINE_TORQUE, ...]
guard = SPEED_SENSOR->VEHICLE_SPEED > 90.0
if not initialized:
    do initialization if necessary...
return (monitored_signals, guard)
```

This pseudo code is inspired by the Python syntax used in the actual implementation and its content represent the test step from the use case in Section 1.4. In the pseudo code from above initialization is mentioned. This is because the test developer has the possibility to perform initialization actions before the server start updating its subscribers. Possible implementations that requires initialization is to define default values or set the number of occurrences a specific event may occur to satisfy the test step. During the update loop phase the test client is instead updated on the most recent signal update and make use of the IGAs by writing the assessment of the test in the following pseudo code.

```python
if guard is True:
    if ENGINE_SENSOR->ENGINE_TORQUE > 0:
        result = FAILED
    else:
        result = NOT_FAILED
else:
    result = NOT_TESTED
return result
```

As for the crucial real-time analysis properties they are not visible through any real-time behavior of the software tool itself as its not running in real-time. Instead the real-time analysis capabilities are supported by supplying each signal with an absolute timestamp. This allows for comparison between timestamps for one signal to another with the same resolution as the CAN messages are sent on the CAN-bus. This means that events transpiring faster than the allowed transfer speed on CAN or events local to one ECU on the CAN-bus cannot be monitored as their updates are too frequent or not observable. An examples of this is injection and ignition of fuel into the engine cylinders. In addition, continuous signals that are transmitted over CAN will become sampled, both by its designated sensor estimates its digital value from the real-world and the ECU transmitting the sensor value out on the CAN-bus.

**Test Engineer**

For the test engineer the consideration mostly touch upon the utility of the software tool. Some considerations regarding this has already been addressed in Section 5.2 for example. However, it is not desired by the test engineers to be involved with sophisticated details in how the actual scripts or the software tool itself is written, but instead to seamlessly use the result of it. Therefore a Graphical User Interface (GUI) is developed with no real control over the representation other than means to close the tool when necessary. Launching the automated testing tool itself is simply done by adding the desired test steps written as Python scripts to the list of test steps to be executed and launch the server Python program.
Owner and maintainer

Currently it is not clear as to who would be the actual owner of the automated testing tool. This means that it is difficult to analyze how the tool itself would affect this stakeholder. Despite this uncertainty, there are still some conclusions to be made. This is because some of the reason used for the stakeholder test developer can also be applied in this case. As Python seems to be the de facto language for most quality assurance employees, mostly due to its expressiveness, non-proprietary license, and simplicity, it is reasonable that this is another support for using the language for the back end of the software tool.

5.4.3 Architecture

A client-server architecture is adopted for the fundamental structure of the devised tool. In this way test steps can be bolted on as clients to a server managing the information flow from the truck. The server itself will be monitoring the a CAN bus and save its signals. Doing so the truck can be viewed as a finite-state machine with various states representing one or multiple signal configurations. The test clients on the other hand can with such a server subscribe to a change of state, representing that the truck has changed its physical state, and ask to be updated whenever this occurs. This way tests are informed and run in parallel without utilizing resources prodigally or having a vehicle state go past unnoticed. This architecture is illustrated in Figure 5.1 showing an example with a brake signal represented in the server as a finite-state machine.
Software Implementation

The implementation is written in Python using third-party programs to fetch CAN signals from the vehicle and broadcasting them to the server who constantly listens to these messages during the update loop phase. Each test client, representing a test step, is run as parallel threads to the server and GUI thread, communicating with each other over non-blocking queues. Because each CAN message has a timestamp it is possible to perform real-time tests by analyzing this timestamp without being forced to actually execute the test step in real-time as frequent updates can be put into the queue and then be removed from the queue whenever the test thread is ready.

Each test step written as a separate target function for a Python thread. This allows each individual test step to save and access previous states of the truck, but also its own previous states. This allows for more complex behavior to be tested other than simple variable comparison, e.g. any of the exotic algorithms mentioned in Section 3.2.4.
Client-server Interaction

To illustrate the logical design of the architecture and the different software objects a class diagram can be used. Such a class diagram is presented in Figure 5.2. Depicted in the middle of Figure 5.2 is the server and the queues that pass data between the different objects. Acting as the main class and program, the server holds all the test clients without knowing their explicit implementation. The only knowledge the server retrieves from each test client is what signals the test client desires to monitor and which of these signals that should cause the test client to be updated. The test clients on the other hand are not directly aware of the server, only that there exist a queue from where they should retrieve updated data. Test clients are, however, indirectly aware of the GUI thread and puts their verdicts on a separate queue for the GUI to present to the driver. Lastly, the CAN Hardware Interface is a third party software application object that enables communication with CAN hardware and feeds the CAN messages to the server when the server’s \texttt{can\_msg\_handler(data)} is called.

![Class diagram of the logical outline of the automated software tool.](image)

The reason for using loose connectivity and one-way communication between the different actors in the diagram from Figure 5.2 is to ensure that there will be no downtime waiting for another actor to respond. This way, the queues feed information from one actor to another asynchronously. Using non-blocking queues in this way together with parallel threads, the need for polling is removed entirely which means that the software tool will perform its assessment as fast as its hardware platforms allows.

There are different stages where the automated in-vehicle testing tool is either setting up, in update loop, or in teardown phase. During these stages the server’s different methods are used in order to add monitored signals to its library of signals to record, update the subscribing test clients, etc. These different phases are described in Sections 5.4.3, 5.4.3, and 5.4.3 below. It is not only the server that is affected by the distinct modes of actions seen for the automated in-vehicle testing tool. These stages are also noticeable at the client-side. The clients’ correlation to the three stages are illustrated in Figure 5.3, with each stage separated by a black horizontal dashed line. Figure 5.3 will be referenced to and used for illustrative purposes in each of the stages’ own sections below.
CHAPTER 5. CASE STUDY WITH DEMONSTRATOR

Setup
Starting with the setup phase, the server firstly fetches and initiates the scripted test steps, test clients, to be used in the current testing round. When initiating, each test client is started as a separate thread. From each test client the server asks for its monitored signals and which signal changes the test client is to be updated on. The reason for this is that a test client might monitor many signals, but might only be interested in running its assessment when some of them change. This is shown as the blue arrow going from the test client to the server in Figure 5.3. After the monitored signals are registered, a third party software for communicating with CAN hardware is initialized as a separate process together with the GUI on an independent thread.

Update Loop
During the second, update loop, phase the server starts listening to the CAN messages sent from the vehicle. For every message that the server has a registered monitoring test client it saves the message data together with the timestamp, provided from the third party software API, into its database of monitored signals. If the CAN message has subscribing test clients, these clients are then updated on the change and supplied with the monitored signals they initially requested during the setup phase. In Figure 5.3 this is illustrated with information flowing, as arrows, from the CAN-bus and ultimately to the test client. Once a test client has performed its assessment the verdict is sent to the GUI thread and presented to the driver. If the verdict from the test client is either pass or fail, as opposed to not failed, not passed, inconclusive, or never tested, the test client and its test step is considered to be finished and discontinues its assessment. This is

![Figure 5.3: Different stages during the utilization of the automated in-vehicle testing tool.](image-url)
done in order to relieve computational resources for any remaining threads and processes running on the same PC as the automated in-vehicle testing tool.

Teardown

Teardown phase is reached whenever the driver decides to close the program by shutting down its GUI. When this happens the server ends each test client thread it evoked, shown as the discontinued signal update and white skull in Figure 5.3, before ending its own main loop.

5.4.4 Scenario with Use Case

To illustrate the interaction between the driver, vehicle, server, clients, and third party software the use case from Section 1.4 will be reused. The use case is an actual test step performed when conducting a test case verifying the driveline of the vehicle.

Whilst the vehicle is in motion, the driver presses down the accelerator forcing the vehicle to increase its speed. In this scenario the vehicle will at some point change its speed from $89.0 \text{ km/h}$ to $90.3 \text{ km/h}$ with the driver still pressing down the accelerator. By the engine logic torque is supplied at the instance before the speed is above $90 \text{ km/h}$ and when speeding faster than $90 \text{ km/h}$ the torque output is set to zero, despite the driver pushing down the pedal.

Sending the signals over CAN, the engine ECU and the speed sensor updates all nodes that are currently active on the CAN-bus. One of these nodes is a CAN hardware interface with third party software that relays the signals on the CAN-bus to software applications on a PC. At this point the server will continuously listen for new CAN messages and when it receives a message containing the vehicle speed or the applied engine torque these are saved to a monitored signal database. The server will also push the updated signals to its subscribing test clients. Whenever new CAN messages arrive this cycle starts anew. The server is not at any point aware of the current situation of its clients.

As a client to the server, test clients listen constantly for new signal updates from the server through the communication queue. When the client receives an update containing the vehicle speed and engine torque it passes it through its guard, ensuring that the vehicle is in its testable state. Because the first signal is received with a vehicle speed of $89.0 \text{ km/h}$ the test client considers the vehicle to not be in a testable state and produces a result stating that the test has not been performed. The second time an update is sent from the server, the client determines that the vehicle speed is $90.3 \text{ km/h}$ which fulfills the guard mentioned in the pseudo code in Section 5.4.2. When the guard is fulfilled, the test client proceeds with its test step assessing that the engine torque is in fact zero. This produces a verdict that the test did not fail and the verdict is pushed, in the same manner as updates are pushed to the test client, to the GUI for the driver to inspect.

The driver continues driving until coming to a full stop, taking the opportunity to look at the test steps performed. He or she notices that during the previous episode of driving the vehicle speed up over $90 \text{ km/h}$ and that the test step to evaluate the requirement from Section 1.4 did not fail. Continuing with the rest of the test steps in the test suite, the driver is not longer concerned with accelerating the vehicle up to its maximum allowed speed, but instead focuses on other test steps.
5.5 Data Collection

As described in Section 2.4.3 the case study will put the implemented tool into use in a realistic simulated test environment. This has been done at Scania CV AB’s own test track with two test drivers. One of them, test driver "A" for reference, from the group "System Test Transmission" and the other from the group "System Test Engine", test driver "B" for reference. "A" is a test engineer at "System Test Transmission" and the duties connected to being a test engineer are mainly to performing manual in-vehicle tests, report on detected faults, and update test specifications. When analyzing "A”’s interaction with and without the tool the main purpose is to identify how the regular user behaves in in-vehicle test environment from an tool automation perspective. "B”’s title is test developer and has test scripting, test specification development and some in-vehicle tests as common duties. This will reflect the other aspect of the automated tool as "B” is an intended test-script writer for the tool itself.

Both "A” and "B” performed the same test case four times each, two times with the automated tool and two times without it. Because both participants are pulled aside from the regular assignments as employees it was not possible to perform all four trials in a sequence. Instead, both conducted two test runs and then with less than a work weeks intermission performed the remaining two trials. The order in which the trials where performed can be studied in Table 5.1 with ”Manual” symbolizing a manual test and ”Auto” a test with the automated test tool.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Manual</td>
<td>Auto</td>
</tr>
<tr>
<td>Auto</td>
<td>Manual</td>
<td></td>
</tr>
<tr>
<td>Intermission</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Manual</td>
<td>Manual</td>
<td></td>
</tr>
<tr>
<td>Auto</td>
<td>Auto</td>
<td></td>
</tr>
</tbody>
</table>

The reason for conducting four attempts each is that it can be expected that there are more factors playing a role in how long time it takes to carry out a test case. One of these factors is how familiar the test driver is with the test case. If the driver is unfamiliar with a test case it can be expected to take longer time to complete as less of the necessary activities are known beforehand. By doing four attempts it is expected that it will be possible to see such an impact and be able to take this into account when doing the analysis.

The test specification document, governing what actions and test steps the driver takes during testing, contains 11 automated test steps out of 26 test steps in total. Some of the automated test steps have been split into multiple test scripts for convenience and simplicity, resulting in 18 separate scripts.

5.6 Results from Case Study

By measuring the time taken to conduct the tests from start to finish a result table can be created as shown in Table 5.2.

The figures in Table 5.2 are measured from when the test driver starts testing the truck’s different systems till he or she returned to the original position. Following attempts are then conducted in the same manner, starting from and ending at the original position. The two test drivers did, however, not share the same initial position.
Table 5.2: Results from the in-vehicle tests.

<table>
<thead>
<tr>
<th>Attempt</th>
<th>A/M</th>
<th>Time [min]</th>
<th>A/M</th>
<th>Time [min]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Manual</td>
<td>27</td>
<td>Auto</td>
<td>40</td>
</tr>
<tr>
<td>2</td>
<td>Auto</td>
<td>20</td>
<td>Manual</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>Manual</td>
<td>26</td>
<td>Manual</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>Auto</td>
<td>21</td>
<td>Auto</td>
<td>23</td>
</tr>
</tbody>
</table>

5.7 Discussion on Case Study Results

From Table 5.2 there is no clear relationship between the tool usage and the time taken to conduct the test. The reasons for this are numerous. Firstly it is clear that the first attempt for both "A" and "B" are the attempts that took the longest for them to complete respectively. This indicates that a test case will take longest time to execute if the driver is unfamiliar with the test case. This is in line with what was expected as stated in Section 5.5. This is most evident in the case of driver "B" where the first attempt took at least 67% longer than the later attempts. For participant "A" this change is not as evident as it is only count for 4% at the very least. A change that small can very well be a result of other factors, e.g. other vehicles on the track interfering with the test route.

Looking at the rest of "B"s attempts it is apparent that the same sort of uncertainty exist as it does for the case of "A"s initial attempt. In this case the third attempt of "B" takes 4% longer than the other two leading to the same conclusion as mentioned above. What can be noticed is however the lack of direct and substantial impact from the automatic tool. Since the three last attempts all share the same time, save for the 4% deviation, two possible conclusions exist. One conclusion is that the test tool does little effect on the total time taken, at least compared to other events that can cause time loss. The second conclusion is that its effect is proportionate in size compared to other external events, which in this study skewered the result into showing no obvious correlation between a usage of the automated in-vehicle tool and time taken.

In contrary to the later attempts made by "B", "A" show a decrease of the required time by 24% from attempt three to attempt four. It is possible that this is because of the automated testing tool itself, but it is also coherent with the intermission, which could have affected the time taken to execute the test case. The reasoning for this is that if driver "A" did forget the different test steps included in the test run he or she would have to relearn their corresponding actions and their sequence again, resulting in a longer execution.

In addition to the measured time taken other observations were made as well. They can be divided into different categories related to time, performance, confidence, user interface, safety, complexity and the case study itself. Observations were not only made when performing the actual testing, but also when writing the test scripts as a developer would do in a similar case. These findings are further described in Section 5.7.8.

5.7.1 Time

The time it takes to do a test strongly depends on surrounding circumstances, such as where on the test track you are and what other vehicles are driving nearby. Because some test steps demand that the vehicle is in a certain area or on a particular road on the test track the driver was often forced to drive to a certain location just to be able to test one specific test step even though he or she had already been there before. This is
CHAPTER 5. CASE STUDY WITH DEMONSTRATOR

because it is difficult to remember and perform all the test steps connected to particular situation at once.

5.7.2 Performance

The level of accuracy of the test verdicts that the driver by him- or herself conclude may vary depending on personal preference. This is because when evaluating many of the test steps, even those that are not explicitly dependent on the drivers sensation, the driver often still relies on his or her senses instead of measured signals. This can be exemplified when the driver checks the test step used as a use case in Section 1.4 with the requirement that the engine is not allowed to accelerate the vehicle if the speed is greater than 90 km/h. In this example the driver accelerates the vehicle up to said speed limit and then tries to accelerate by pressing down the accelerator pedal while at the same time trying to feel if the vehicle is accelerating. One problem with such a test execution is that if this is done in a downhill slope the vehicle will accelerate by its own and it may be difficult to distinguish acceleration caused by the engine or the surrounding environment. In cases such as this the automated testing tool was more powerful in its capability to give a more exact depiction of the actual system responses within the vehicle.

Another observation is that the driver could deviate from the intended execution of some test steps. In some cases an alternative execution was performed with equal coverage and resulting in an equivalent verdict. In other cases the driver missed some information in the test step and started performing the test in a faulty manner, only to have to redo the step again correctly. With the automated test tool it becomes apparent when the contemplated actions are omitted as it communicates to the driver that it has not been initiated or completed compared to the case where the driver him- or herself decides if the test is carried out correctly and documents the result. The reason why this occurred during the case study might be because the study is not an actual quality assurance activity in the company process. Therefore, it is plausible that it was not carried out with the same level of precision as it would have otherwise. Another reason, possibly connected with the previous, is that the amount of preparation before the actual testing was not as extensive as in the normal case.

During driver “A”的 second attempt the test program noticeably started to fall behind the events happening and was not able to keep up with the signals it needed to process and assess. This was due to extensive monitoring using multiple monitor softwares connected to different hardware interfaces, e.g. CAN, webcam, etc., on the PC. Once some of the monitoring softwares lessened the burden on the CPU, test scripts started to output their verdict for the driver to view. This sort of issue is to be considered likely to happen when the amount of simultaneously running programs increases. The core problem is that the driver might not realize that this is happening and concluding that test conditions have not been fulfilled, therefore doing the same test steps again only to later realize, when then program catches up, that the test was performed.

Whenever the test driver had finished a number of test steps or when passing parking spaces he or she would pull the vehicle over in order to fill in the test specification document according to the verdict that he or she has assessed. This occurred frequently during all attempts for both drivers and caused each time at least minute long breaks from the actual driving. One driver therefore expressed his or hers impression that decreased interaction with the computer would ease the everyday testing activities because of todays frequent computer interaction.
5.7.3 Confidence

The confidence related observations are connected to the level of trust that the driver has for the tools used. As not all test scripts were functioning correctly, up to half of the test steps could use scripts that contained software bugs later addressed in Section 5.7.6, there was a notion of having to double check both test steps with the faulty scripts and test steps with fully functioning scripts. Unsurprisingly, this made the driver redo some of the test steps to feel confident that they had been properly tested.

5.7.4 User Interface

The interface used consisted of a colored table with the test scripts together with their verdict, as seen in Figure 5.4. One notion observed was that it is not always clear how to utilize the information presented by the graphics. As test steps was considered tested and a verdict was generated colors would change for that particular test and the "Verdict" column would update accordingly, but despite this it was difficult for the driver to map the test script’s result to the test specification document and his or hers actions. This caused the driver to miss that some test cases were already assessed and had been given a verdict.

![Figure 5.4: Verdict table from the automated testing tool.](image)

5.7.5 Safety

As mentioned in Section 5.2 there were some safety related considerations for the case study. One being the risk of fault injection in the case that the tool itself would inject faults into the SUT and therefore cause it to exhibit dangerous behavior. This was not experienced at any time during the case study and it can be argued that the monitoring architecture prevents this and makes it an improbable scenario.

The other mentioned safety related consideration in Section 5.2 is the level of intrusiveness of the automated in-vehicle testing tool. This was noticed whilst performing the case study as the driver sometimes draw their attention to the computer screen when the rows changed colors as a verdict was updated. However, the tool does not require any attention and the drivers instead studied the verdicts in more detail when coming to a full stop, as desired.
5.7.6 Complexity

A somewhat unforeseen problem revealed itself during the course of the case study related to the system complexity. When developing the scripts, many of them had to be redesigned and revisited because signals are system-version dependent. This means that performing test steps on one system might result in error-free scripts, while performing the same test steps on another system might exhibit software bugs and misinterpretation from the automated testing tool.

Version dependencies are not the only complexity issue encountered. The overall system complexity led to non-trivial test steps with multiple signals affecting each other in an unforeseeable manner. Because of this, the stability of the test scripts is a direct result of the script writer’s intrinsic technical knowledge of the SUT. One concrete example of this can be illustrated by the requirement from the use case in Section 1.4. First implemented as a test script with similar logic as the pseudocode from Section 5.4.2, the test step failed the SUT every time. Reviewing the vehicle logs, it was revealed that torque could be generated by the engine while traveling faster than 90 km/h if the clutch was disengaged. This could occur if a gear change is made. After adding additional conditions for the analysis, it was possible to generate an accurate verdict using the updated test script.

5.7.7 Case Study

The longest attempt made by any participant in the case study was 40 min. During this test case, it was discovered that not as many test steps had been triggered by the automated in-vehicle testing tool as expected. The reason for this is that a test case performed during a shorter period of time does not demand many repetitive actions as there are not that many actions to repeat. In order to prove a better correlation between the usage of an automated in-vehicle testing tool and the time needed to perform the test case, a longer test case should be studied. In this way, test steps late in the test specification document might be triggered while performing the earlier test steps and thus have the potential to cut the time necessary to perform the entire test case.

A notion that the participating drivers had while testing the second, third, and fourth time is that the truck should exhibit the same behavior as the previous attempts. This might influence their assessment of the test steps and result in a biased verdict. In addition, this could lead to faster execution as the driver might jump to the same conclusions as those drawn in previous attempts. By utilizing scripted test steps, there is no bias as to previous test runs, and the verdict should produce consistent outputs.

5.7.8 Elimination of Assumptions

With the designed architecture and the use of IGAs, the developer writing the test scripts has to adapt a new way of thinking to not only cover the use case, but instead cover all situations that may trigger the SUT in a way that enables a test step to be executed. The reason for this is that no test step can assume a specific drive pattern before its assessment, which eliminates dangerous vague assumptions regarding the vehicle stimuli. This leads to more formal test specifications, which in turn raises demands for more formal and specific requirements. Furthermore, when writing scripts, the programmer must understand the SUT in layers deeper than the mere input and expected output as other systems and possibly sub-systems may cause interference and produce an output different from the developers’ expectations. This can be a faulty behavior, but might as well be a correct outcome for the given situation.
5.8 Conclusion of Case Study

The case study finds a number of different results and draws interesting conclusions from them. Starting with the main goal, from Section 2.2, of shortening the time for in-vehicle testing, this thesis show that the time taken for in-vehicle testing is not noticeably affected by an automatic in-vehicle testing tool. This is at least true for shorter test cases, ranging from 20 to 40 min. The reason for a low correlation is discussed to be because a short test case does not include many repetitions to begin with. Another reason, not excluding the previous, is that other factors affect the time necessary to perform all test steps in a test case equally or to a greater extent than what an automated in-vehicle testing tool does. One of these factors is the learning curve for the test driver. Another, external, factor could be other road users.

Using scripts to automate test steps is shown to give more accurate and reproducible verdicts. In addition, writing test steps as scripts also forces a formal definition of the requirement that is to be verified.

As a major concern, safety is also a recurring theme discussed. In this case study, no sign of endangerment of either the driver or other road users is found. The identified possible safety related aspects for this case study would be caused by either drawing too much attention from the driver or by injecting faults into the SUT. This also answers research question Q6. To answer research question Q7 the architecture of the demonstrator is strictly limited to a monitoring behavior and is unable to directly interact with the SUT.

Not only does the case study present a number of beneficial findings, but also some issues. A possible issue encountered is lag or latency of the PC running the automated in-vehicle testing tool due to high CPU utilization. Nonetheless, the automated in-vehicle testing tool is capable of catching up to current event as the CPU utilization is decreased. Another problem that emerged is lack of confidence and increased distrust between driver and automated in-vehicle testing tool if the test scripts contain software bugs. This can ultimately cause the driver to doubt all of the automated in-vehicle testing tools verdicts, even those generated from healthy test scripts, as the driver cannot distinguish between scripts containing bugs and scripts that do not.

A rudimentary GUI also cause to some unwanted effects. The drivers experience difficulties mapping generated verdicts to their respective test steps and could therefore not utilize the automatic in-vehicle testing tool to the fullest.

Lastly, when testing embedded and distributed systems, the complexity of the systems themselves is an obstacle. In this case study the monitored signals would change depending on the system variant leading to software bugs. Furthermore, the system complexity of a heavy truck makes it difficult to create stable and robust test scripts that do accurate assessment for all situations.

Research question Q3 is answered in this chapter by a number of methods for streamlining the verification process. However, the methods found mainly focuses on simulated environments and are unable to be applied directly to the in-vehicle environment. Implementing a client-server architecture using continuous, parallel, and scriptable test steps together with IGAs allows for the methods identified in the other environments to be, partly or fully, transferred to the in-vehicle environment. This answers research question Q5.
Chapter 6

Conclusion and Future Work

Setting out to investigate a novel architectural solution for automated in-vehicle tests this thesis finds the associated drawbacks, benefits, obstacles, and possibilities. As a drawback an additional tool inside the cab for the driver to utilize means one more potential source of distraction. The idea and hope is, however, that the total amount of distractions will be lesser than the alternative of not using an automated testing tool. This is based on the concept that automated testing tools could eliminate the need for other manual software tools, if not entirely then partly.

The most prominent benefit stated throughout the thesis is that automation could lead to shorter testing times. Despite the fact that this is not directly evident in the empirical study performed, this feature is believed to be more prominent for larger test cases, or when performing entire test suites. Automation is also connected to more consistent verdicts as the assessment is performed in an identical fashion for all executions. This means that verdicts become comparable, even across the temporal dimension as it can be proved what caused the assessment to generate a certain verdict by studying the tests logic.

Other less obvious features include the possibility to preserve knowledge in executable documentation in the form of scripts compared to long documents or by passing it verbally. This allows for distribution of the knowledge to ensure that it is not locked to a single individual. Another feature is the elimination of assumptions made when designing test cases and test steps. This leads to less guesswork and more formal tests that has the possibility to state demands on more formal requirements that are unambiguously defined.

Current obstacles for adapting the tool in the quality assurance process are mainly focused on the complex behavior of embedded systems in vehicles. One problem that arise as a consequence of this is unstable code. This can further diminish the reliability of the automated in-vehicle testing tool and decrease its credibility.

Since IGAs are well suited for this architecture, as no stimuli can ever be known beforehand, the possibility to migrate HIL test cases to an in-vehicle environment exists. This future possibility increases the tool’s flexibility and enables more holistic test development as scripts from SIL and HIL can be migrated into the real vehicle.

6.1 Threats to Validity

In order to achieve a level of credibility the threats to validity is presented in this section. Being central to both results and conclusion, the case study is a primary threat. It is a validity threat because of the risk of it not being representative as it is, to some extent, fabricated. A number of different countermeasures have been applied in order
to decrease this risk. One being that the test case studied is directly adapted from an existing test case specification with some adjustments made to it. The limitations are added in order to limit the real test case’s scope and extensiveness. This means that even though this very test case does not exist at the company site, all its test steps do exist and are performed in the daily quality assurance work.

Another aspect that could affect the result is that the code used in the case study contained some software faults. This has the potential to prolong the execution time due to confusing or erroneous verdicts. However, it is to be considered a future problem as well as broken test cases and test steps already exist at the commissioning company, e.g. for cases when tests are automated against the HIL environment. Therefore one could argue that it is rather a verifying feature, similar to the normal case, rather than an experimental flaw.

Concerning the data collection only one group is interviewed. This may, as stated in Section 4.1, lead to unsaturated information, but also that some viewpoints, requirements, and needs remain undiscovered. Supporting this limitation is, however, the notion that in large companies, such as Scania CV AB, there will always be contrasting needs, especially between different departments, which cannot all be covered by one single software tool developed during the course of a thesis. Therefore, it is not particularly interesting to keep abreast of all the different opinions. As its scope also declare, in Section 2.3, this thesis limits itself to implementing an architectural solution for an automated in-vehicle testing tool for the in-vehicle environment to be used during system-level testing. As all the participants in the interviews work at system-level testing they are both considered target benefactors and key informants of this thesis, therefore making the decision to use them as interviewees a logical rationalization.

Transferability is an additional aspect that deserves attention. As the investigated problem and developed tool are applicable in virtually any distributed embedded system using CAN-bus communication with real-time demands, the results should be transferable to these cases. Even though the thesis is performed at the truck manufacturer Scania CV AB, it is still feasible to assume that a similar solution and conclusion could be reached for other vehicle manufacturers and developers as well. Despite the similarities no investigation, apart from the literature study that shows similar situations in other fields, is made to support this.

6.2 Future Work

There are four suggested future research fields that would extend and complement the investigation carried out in this thesis. The first field is to conduct larger and longer, in terms of both time and extensiveness, empirical case studies to attain more details regarding the effectiveness and usefulness of the proposed architecture. This should be carried out with the use of more stable software and a more integrated software environment in order to more easily separate the different variables affecting the result.

The second suggestion is to research the possibility to optimize the test route. This would be done in order to cover more test steps in a shorter period of time, but at the same time consider the safety aspect that the route infers on the driver and the immediate surroundings.

A third suggestion for future study would be to investigate the level of generalization of the designed architecture. Such a study should try to answer whether or not it is possible to implement the same architecture to other fields. Contemplated fields of interest are not only other automotive fields, such as cars, but possibly other real-time distributed embedded systems. Any findings related to such a study would bring benefits to all the applicable fields.
The last future research proposal is to scrutinize generation of the IGAs from the initial requirements or from design models. This way IGA-based testing would become a interwoven phase in the model-based process and be able to draw from its benefits.
Bibliography


BIBLIOGRAPHY


Appendix A

As a part of the thesis and one of the data collection methods used to answer some of the research questions from Section 2.2, the studied papers, reports, proceedings, MSc and PhD theses, and books are listed in Table 1 below together with their relevance rating (RR). The different sources as accessed through SAE Digital Library, IEEE Xplore, ACM Digital Library, and DiVa.

Table 1: Used sources for the literature review with their relevance score

<table>
<thead>
<tr>
<th>Title</th>
<th>RR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monitor Based Oracles for Cyber-Physical System Testing</td>
<td>10</td>
</tr>
<tr>
<td>Automotive System Testing by Independent Guarded Assertions</td>
<td>10</td>
</tr>
<tr>
<td>Model-based Testing of Automotive Systems</td>
<td>9</td>
</tr>
<tr>
<td>Better Testing Through Oracle Selection</td>
<td>9</td>
</tr>
<tr>
<td>Offline Analysis of Independent Guarded Assertions in Automotive Integration Testing</td>
<td>9</td>
</tr>
<tr>
<td>Automatic Evaluation of ECU Software Tests</td>
<td>8</td>
</tr>
<tr>
<td>MASTER THESIS - ABSTRACTION LEVELS OF AUTOMATED TEST SCRIPTS</td>
<td>8</td>
</tr>
<tr>
<td>ON TEST DESIGN</td>
<td>8</td>
</tr>
<tr>
<td>On testing non-testable programs</td>
<td>7</td>
</tr>
<tr>
<td>Design and Development of a Computer-Aided In-Vehicle Data Acquisition System for Driving Pattern Analysis</td>
<td>7</td>
</tr>
<tr>
<td>Introducing a Reasonably Complete and Coherent Approach for Model-based Testing</td>
<td>7</td>
</tr>
<tr>
<td>Model Based Development and Auto Testing: A Robust Approach for Reliable Automotive Software Development</td>
<td>7</td>
</tr>
<tr>
<td>A Comparative Study of Industrial Static Analysis Tools</td>
<td>7</td>
</tr>
<tr>
<td>Taming Complexity While Gaining Efficiency: Requirements for the Next Generation of Test Automation Tools</td>
<td>7</td>
</tr>
<tr>
<td>Analysis of requirements for an automated testing and grading assistance system</td>
<td>7</td>
</tr>
<tr>
<td>Design verification and validation in product lifecycle</td>
<td>6</td>
</tr>
<tr>
<td>Effective In-Vehicle Acquisition</td>
<td>6</td>
</tr>
<tr>
<td>A Systematic Approach to Testing Automotive Control Software</td>
<td>6</td>
</tr>
<tr>
<td>Test Specification and Test Methodology for Embedded Systems in Automobiles</td>
<td>6</td>
</tr>
<tr>
<td>Portal of Research Methods and Methodologies for Research Projects and Degree Projects</td>
<td>6</td>
</tr>
<tr>
<td>An Empirical Analysis of Test Oracle Strategies for Model-based Testing</td>
<td>6</td>
</tr>
<tr>
<td>Investigation of a new integration test environment</td>
<td>6</td>
</tr>
<tr>
<td>The Concept of Dynamic Analysis</td>
<td>6</td>
</tr>
<tr>
<td>Evolutionary Safety Testing of Embedded Control Software by Automatically Generating Compact Test Data Sequences</td>
<td>5</td>
</tr>
<tr>
<td>Title</td>
<td>Pages</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>Signal validation in an automotive vehicle using Neural Networks</td>
<td>5</td>
</tr>
<tr>
<td>Driving Asia</td>
<td>5</td>
</tr>
<tr>
<td>Efficient Testing Framework for Simulink Models with MTCD and Automated Test Assessments in the Context of ISO 26262</td>
<td>5</td>
</tr>
<tr>
<td>Automated Verification and Validation Methods for Transmission Control Software</td>
<td>5</td>
</tr>
<tr>
<td>Automatic Test Oracle for Image Processing Applications Using Support Vector Machines</td>
<td>5</td>
</tr>
<tr>
<td>Model-Based Automated Validation Techniques for Automotive Embedded Systems</td>
<td>4</td>
</tr>
<tr>
<td>A Search-Based Approach to Functional Hardware-in-the-Loop Testing</td>
<td>4</td>
</tr>
<tr>
<td>Advanced Automated Onboard Vehicle Diagnostics Testing</td>
<td>4</td>
</tr>
<tr>
<td>A Taxonomy for Test Oracles</td>
<td>3</td>
</tr>
<tr>
<td>GPS-navigering i Scanias telematikenhet</td>
<td>3</td>
</tr>
<tr>
<td>Heuristic Test Oracles: The balance between exhaustive comparison and no comparison at all</td>
<td>3</td>
</tr>
<tr>
<td>The Concept of Dynamic Analysis</td>
<td>3</td>
</tr>
<tr>
<td>Vehicle Driveability Assessment using Neural Networks for Development, Calibration and Quality Tests</td>
<td>3</td>
</tr>
<tr>
<td>Formal Verification for Model-Based Development</td>
<td>3</td>
</tr>
<tr>
<td>SYSTEMATIC DATA-DRIVEN APPROACH TO REAL-TIME FAULT DETECTION AND DIAGNOSIS IN AUTOMOTIVE ENGINES</td>
<td>3</td>
</tr>
<tr>
<td>DESIGN AND IMPLEMENTATION OF VEHICLE FAULT DIAGNOSIS SYSTEM FOR INTELLIGENT TRAVEL</td>
<td>3</td>
</tr>
<tr>
<td>Using thematic analysis in psychology</td>
<td>3</td>
</tr>
<tr>
<td>Knowledge Extraction from Real-World Logged Truck Data</td>
<td>2</td>
</tr>
<tr>
<td>A state-based approach to integration testing based on UML models</td>
<td>1</td>
</tr>
<tr>
<td>Software Tool Qualification According to ISO 26262</td>
<td>1</td>
</tr>
<tr>
<td>A companion to qualitative research</td>
<td>0</td>
</tr>
<tr>
<td>Constraint Programming for Random Testing of a Trading System</td>
<td>0</td>
</tr>
<tr>
<td>A Format for handling automation in Requirements Management</td>
<td>0</td>
</tr>
<tr>
<td>Lessons learnt from using DSLs for automated software testing</td>
<td>0</td>
</tr>
<tr>
<td>An Introduction into the Testing and Test Control Notation (TTCN-3)</td>
<td>0</td>
</tr>
</tbody>
</table>