Optimized Testing Process in Vehicles Using an Augmented Data Logger

Karsten Hünlich
Steinbeis Interagierende Systeme GmbH
Esslingen, Germany
karsten.huenlich@steinbeis-ias.de

Ulrich Bröckl
University of Applied Sciences Karlsruhe
Karlsruhe, Germany
ulrich.broeckl@hs-karlsruhe.de

Daniel Ulmer
Steinbeis Interagierende Systeme GmbH
Esslingen, Germany
daniel.ulmer@steinbeis-ias.de

Steffen Wittel
Steinbeis Interagierende Systeme GmbH
Esslingen, Germany
steffen.wittel@steinbeis-ias.de

***Abstract*—The growing amount of electronic components in vehicles requires an increasing communication load between these components and hence an increasing load on the vehicles communication buses. Both aspects entail an increasing workload for the test engineer developing and executing test cases to verify the required system behaviour in the vehicle. This article considers a way to automate and reduce the workload for in-vehicle testing by augmenting the functionality of current data loggers. The idea is to use the data logger for supporting the testing process for test drivers. The introduced implementation shows a way to verify the test cases' execution on the fly in order to avoid finding erroneously executed test cases at a later point in time. Additionally, the presented implementation seamlessly includes the test environment for in–vehicle testing into the tool chain, which is already used on lower integration levels. This allows the test engineer to reuse test cases from the lower integration levels in vehicle tests and to compare the results from test runs on different integration levels. The paper describes two stages of the development process of the augmented datalogger and includes the first feedback collected in a case study with a prototypical implementation.**

Keywords – automotie, data logger, intelligent data logger, test case development, test case monitoring

I. Introduction

This paper offers a closer look at the augmented datalogger and the associated process of in-vehicle testing as they were shown in [1].

Many different data loggers are used in the automotive industry. Primarily, they are designed to record the communication between Electronic Control Units (ECUs) [2]. In more advanced systems, the data content of the Random Access Memory (RAM) of the ECUs is additionally recorded [3]. These data loggers become more and more important to the test engineers because the number of the networked ECUs and hence the testing efforts in a vehicle is continuously increasing. From each requirement on vehicle level, the test engineers have to derive test cases to ensure that the ECUs in a vehicle are performing the correct action within correct time constraints. To check this in an in-vehicle test it is necessary to record the bus traffic and the data content of the ECUs' RAM while executing a test case manoeuvre with a car. The result of the test is determined by evaluating the recorded data.

The amount of collected data can turn the evaluation of the recorded test case data into a time consuming challenge. In current solutions, the result of the evaluation can be classified as "passed" or "failed" In case of a passed classification, the recorded data show that the System under Test (SuT), e.g., an ECU, exhibited the expected behaviour described by the requirements. The classification failed shows a deviation of the measured data from the expected values and hence from the expected test result. But especially if human beings are involved in test execution, the recorded data might be "invalid" if there was a significant mistake during the test case execution. In this case, the evaluation of the recorded data is impossible with respect to the test case's definition.

Figure 1 shows the classification of the test results that can occur in in-vehicle tests. In the first step, the recorded data is usually examined manually by an engineer if it meets the constraints of a valid data record. Some possible cases for invalid data records are:

- The test driver has not driven the test case correctly
- The data logger configuration was incorrect
- The recorded data was incomplete because the measurement has stopped while the tests case was executed

If the recorded data is valid it can be compared with the expected results of the test case. The result of the test evaluation is passed if the system works as expected or failed if the system has not the expected behaviour.

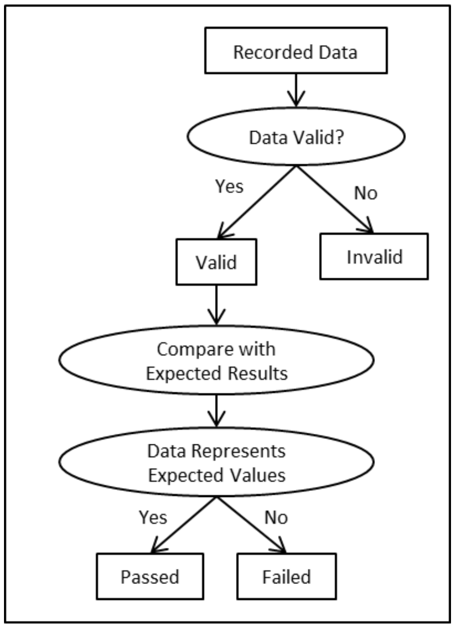
Recorded Data
Data Valid?
Yes
No
Valid
Invalid
Compare with Expected Results
Data Represents Expected Values
Yes
No
Passed
Failed

Figure 1. Classification of test results of in-vehicle tests

To minimize the cases of an invalid data record, and therefore the time for the test case execution and evaluation, the data logger can be augmented with additional functionality to monitor the correct execution of the test case. The necessary conditions are to be defined by the test engineer before test execution. This is possible if the data logger can be extended with instructions supervising relevant signals. For these signals boundaries may be defined. A test case can, e.g., be successfully accomplished if the signal stays within these boundaries. However, the goal is not to test the driver's behaviour as mentioned in [4]. The goals are to give instructions to the test case executor, which may be a driver, a robot or a test automation tool, and to additionally supervise the execution's correspondence to the conditions predefined by the test engineer. Especially a human driver is one of the — corresponding to our experiences — biggest error source in a vehicle during a test case execution. In the following, we show how the augmented data logger can help to avoid unnecessary work by evaluating a test case at runtime for being valid or invalid. If the augmented data logger is not only able to supervise the driver while executing a test case, but also guides him through the test case, the augmented data logger even helps to minimize invalid test executions.

In addition to meeting the challenges of in-vehicle testing, the introduced Augmented Data Logger (ADL) shall seamlessly integrate into a typical development process of the automotive industry. A short overview of the relevant aspects shall be given within the next paragraphs.

Figure 2 shows an example of a system development process according to the V-Model as shown in [5]. In this example, the test on vehicle level is the last level of testing within the integration process. Before this stage, many other tests have already taken place on lower integration levels. For efficiency reasons, it would be helpful if the test engineer could reuse test cases developed on lower integration levels, e.g., test cases from Hardware in the Loop (HiL) tests [6]. The reuse of these test cases minimizes the work for the test engineer to adopt the test cases to the desired test platform. The reuse also enables the comparability of the test results from a vehicle test with the results from lower integration levels. For guaranteeing the reusability of the test cases it is essential to specify the test cases platform independently. A test case language is needed, which is both platforms independent and suitable for all testing platforms. Figure 2 shows typical levels in an automotive V-model and the corresponding testing platforms.

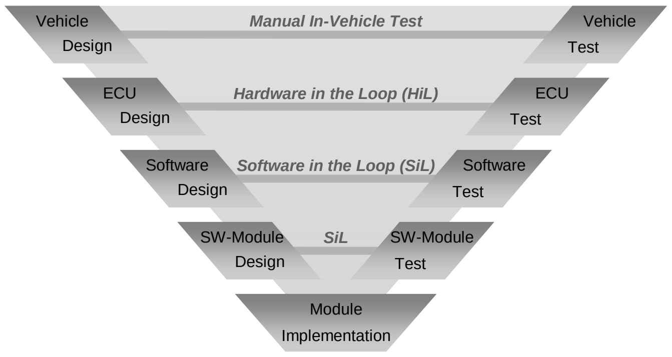

Figure 2. Commonly used application of the V-Model in the automotive industry

The solution described in this article is based on a test case language, which allows the reuse of test cases on Software and ECU levels. Within this article, a solution for extending this approach to "Vehicle Test" is discussed. The solution is based on test cases from lower integration levels by adding information to guide the driver through the test case and by adding instruction to supervise the actions of the driver. The article begins with a description of the state of the art for data loggers and discusses two prototypes of the augmented data logger. The added features are supported by a case study. The paper ends with a summary and ideas for future work.

II. DATALOGGER STATE OF THE ART

Today in-vehicle tests are usually executed without the support of a software tool for giving feedback on the quality of the test execution or a tool that guides the driver through a test case. This conclusion is based on our experience from several automotive companies and suppliers. Instead, the test cases are often written in plain human readable text which describes what a tester has to do in the vehicle to fulfil the test case. These textual test cases are stored for example in a database. For taking a set of test cases to the car, they are either printed out or downloaded to a robust handheld computer. In both cases, they are read before or during a driving manoeuvre. The quality of the execution of the manoeuvre thus depends on the skills of the test driver. Details of the execution quality can be determined offline on a parking lot or by evaluating the information on the data logger. Especially if test driver and test engineer is not the same person, this process is error-prone and time consuming. Since the test cases are in natural language there is enough room for misunderstandings between a test manager who writes the test cases and a test driver who has to execute the manoeuvre. This fact tends to result in multiple iterations of in vehicle tests of the same test case.

There are several solutions that have the aim to optimize in vehicle tests and to minimize the time overhead. A touch-display can be used in vehicles getting rid of the printed check lists and directly sending the results of the test steps to a database. A more advanced system is shown in [7], which comprises of a driver guidance system and a feature to immediately evaluate if the test is passed or failed.

For testing driver assistance functions, manoeuvres have to be executed very precisely by the test driver. That means in a significant number of tests the tests are failed not because the system is not working correctly but the test driver has made a mistake. To minimize this number of invalid tests this paper describes a way to detect deviations of the given test case during its execution. This avoids a usually time consuming evaluation of invalid test cases.

Another solution is described in [8]. This paper describes a system of a car and robot. The robot drives the car inside a restricted area. Within this area the robot performs test cases very precisely. The robot is controlled by engineers from a base station. The approach needs a restricted area because the robot does not recognize its surrounding. This system was developed for executing very dangerous tests, e.g., collision mitigation/prevention tests at high speed rates. Since the system is very expensive and restricted to special test areas it is an addition for human driven cars but cannot replace the human tests.

A. *Datalogger Setup*

Current data loggers [3] are designed for recording data and neither for interpreting it nor for participating in the testing process. This section describes a way of augmenting the functionality of the data logger in order to support the testing process and to seamlessly integrate the vehicle tests in the system integration and testing process.

A data logger to record digital information in vehicles might be designed in the way described in [9]: i.e., a host computer is connected via a network interface, e.g., Ethernet, to the data logger. Over this connection, the data logger can be controlled and configured. The configuration defines which signals are stored in the data storage and on which bus interface the signals can be received. The host computer is mainly used to start and stop the data logger and to visualize an excerpt of the recorded data on the fly. The data logger hardware is responsible for the real time processing of the data. A commonly found feature is a trigger that starts a measurement when a predefined condition becomes true as it is described in [10].

For evaluating the trigger conditions the data logger needs information about the connected data buses and the data that is transferred over a particular data bus. Usually, this information is available in form of configuration and signal files that are interpreted by the host computer and transferred to the data logger.

In some parts of a data logger execution in real-time is mandatory. This is necessary because the test engineer needs to know exactly when some data have been transmitted on a particular bus. A common solution is that the communication on a bus system is recorded together with timestamps, which indicate the time instance when a message is transferred over a bus [11]. Figure 3 shows the procedure of recording a message from a bus. If the data logger receives a message a timestamp is taken. For the evaluation of the recorded data it is possible to correlate in time the different recordings with the help of the timestamps, which means that the more precisely the timestamp is taken the more precisely the situation can be reproduced and evaluated.

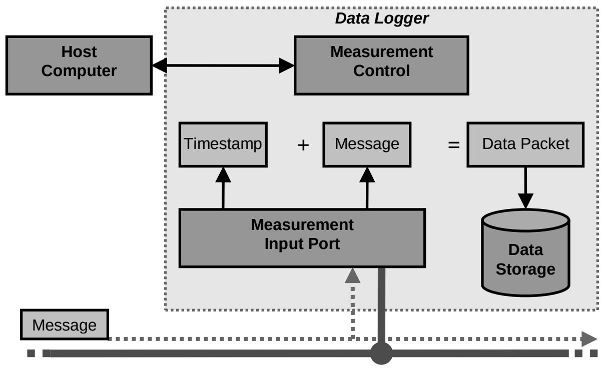

Figure 3. Schematic procedure of measuring a message on the bus

The example in Figure 3 shows a host computer that is connected over a communication interface with the measurement control unit within the data logger. The host computer is commonly a PC or a notebook with an operating system that does not support real time tasks. Via the host computer the engineer has access to and control over the data logger. Additionally, the host computer can access measurement data and visualize them to the user. Evaluating this data while conducting a manoeuvre is almost impossible since, in this case, the driver would have to fully concentrate on the monitor instead on his driving task.

B. *Current Testing Process on Vehicles*

In the common testing process, the test engineer starts looking at the requirements for the SuT. Based on these requirements the test engineer creates the corresponding test cases. How the test engineer writes down these test cases for in-vehicle tests is mostly not defined. In some way, the test cases have to be readable by the driver while he is executing the manoeuvre in the vehicle. After finishing writing a test case, the test engineer has to hand over the test case to the driver who executes the manoeuvre specified in the test case in the vehicle. This is usually supported by tools, which allow configured testing and sending them, e.g., to handheld devices. This test set is executed by the driver. The role of the test engineer and of the driver might be taken by the same person or by different ones. If the test engineer and the driver are different persons who write and execute a test case, the test case must be well defined to prevent misunderstanding. If the test case specification is not complete and therefore, the driver does not execute the test case as intended by the test engineer, the following work might be unavailing.

After having recorded the data of the manoeuvre that is specified in the test case the driver hands over the recordings to the test engineer. Afterwards, the test engineer evaluates the data. Usually, this is done manually. The test engineer has to search through a database of signals with probably more than 10,000 entries. If the result of the test case is passed, the test case will be documented and closed. In case the result is failed, the test engineer has to find the exact reason. The SuT can either have a bug or the test case has not been executed accurately, which means that the test is invalid. If the test case was executed within all defined constraints by the test engineer the test case is valid and hence failed. Both cases generate lots of work of analysing and documentation for the test engineer. Especially, the work for the first case can be minimized by finding out the validity of the test case in an earlier stage of the process.

Generally, the biggest drawback of finding invalid test runs late in the process is the time that the test engineer spends on one test case. It must be considered that the number of test cases that must be performed for each major release can be up to several hundred test cases. As a conclusion two main issues can be identified that can be possibly optimized:

- The time for evaluating the test results by avoiding invalid test cases
- The number of times moving from the office to the vehicle and to the test track for repeating invalid test case

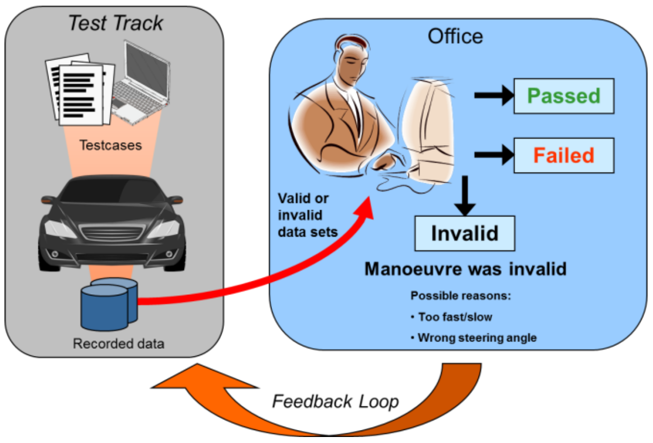

Figure 4. Sample of the current testing process on vehicle level

Figure 4 visualizes the current testing process. The test cases are executed in a vehicle and the recorded data is stored on a local disk of the test system. Later the data is transferred to a computer in the office for evaluation. An engineer evaluates the data and removes invalid data sets. The tests corresponding to the invalid data sets usually have to be executed again. This means going back to the vehicle on the test track. The feedback loop in this example is between two different places, which is time consuming as stated above. One approach to avoid going from the test track to the office and back for several times would be to evaluate the tests in the vehicle after having executed a test set. But while evaluating the tests, the vehicle cannot be used for executing other test sets. Since most of the time test vehicles equipped with measurement systems are rare and have to be shared by many engineers, this approach seems even more inefficient.

The introduced testing process on vehicle level is very different from the test processes on lower integration levels of the development process shown in Figure 1. In the lower levels, i.e., HiL or SiL, a test case is written in a defined way. The test case can be reused and usually returns a reproducible result. Another point is that the test result is directly available after the test has been finished. It can be said that the processes on different levels have mainly five important parts [12]:

- The SuT itself
- Test case execution system
- Environment simulation that simulates the environment of the SuT
- Measurement and data logging system
- Evaluation system

The evaluation system compares the measured values with the ones that are specified in the test case for the SuT. The test case execution system reads the test case and controls the environment simulation that affects the SuT. In a vehicle, the parts for the test process are different. The test case execution system in a vehicle is the test driver. The test driver has control over the environment of the SuT. The evaluation system in a vehicle test is the test engineer who evaluates the measurements.

The measurement and data logging system might be the same as the one used in the vehicle. For the in-vehicle test, an environment simulation is not necessary because the vehicle is used in a real environment. Sometimes both environments are mixed for the vehicle tests, e.g., foot passengers are simulated with synthetic dolls or imaginary sensor information.

III. AUGMENTED DATALOGGER VERSION I

This section introduces the first prototype of the ADL implementation. The focus of this prototype is the implementation of the basic features for giving feedback to the driver. The attached display is not optimized for intuitive feedback and only shows basic text output.

A. *System Design*

The first design of the data logger version was focused on the test case execution inside the vehicle. In this case, a test case is a sequence of instruction the driver has to execute. It also includes a set of rules that has to be met for a valid execution. Each step is s In case of an invalid hown inside a small display as a text message. In case of an invalid execution of the test case, the driver gets a response and the test case execution stops. A laptop is necessary in this version to control and configure the data logger. Only one test case can be stored on the data logger. So the test case selection and loading has to be done by the driver manually. In this first prototype, the test case description has to be converted by a code generator into executable code before the test case execution can start. This approach was good enough for first experiments but far too slow for efficient in-vehicle testing.

B. *Testing Process Supported by the ADL*

To reduce the time for testing and evaluating of in-vehicle testing a new approach for the testing work flow should be considered. The first aspect is the form how the test case is written. A uniform platform independent language (see Section III C. for more detailed information) is used to define the test cases. With this uniform language, the test engineer can precisely describe the test case. The test case is now not only human readable but also machine-readable and can be interpreted by a program. Additional instructions extend the abilities of the data logger. The system now knows about the manoeuvre that has to be executed for a particular test case. With the knowledge of how a test case must be performed, driving errors can be detected directly and time can be saved.

The new work flow has a strict separation between the office work and the work in the vehicle. Right after performing a test case, the driver gets a result if the test case was executed accurately. The feedback also includes the information why the test has been invalid. This information depends on the test case description from the test engineer. If the test engineer describes the test case in many details more driving errors can be detected without looking at the whole measured data back in the office. The advantage of this new approach is that the driver:

- Is guided through the test case execution process through a unified notification
- Gets a response directly after the manoeuvre if the test is executed correctly and hence valid
- Gets the reason why a test case was classified as invalid

This reduces the evaluation work and the test case execution work. Since the data logger instructs and checks the manoeuvre, it makes the execution more precise.

For this new approach, parts of the evaluation system and the test case execution system are added to the data logger. The schematic of a data logger shown in Figure 2 can be extended to execute additional instructions given by the test engineer, which controls the data logger and guides the test driver through the manoeuvre. Figure 5 shows a simplified version of such a measuring system. The CPU (Central Processing Unit) has to fetch the messages from the bus, add a timestamp to each message and extract relevant signals. The values of the signals are internally decoded from the coded bus signals and provided for the test case code.

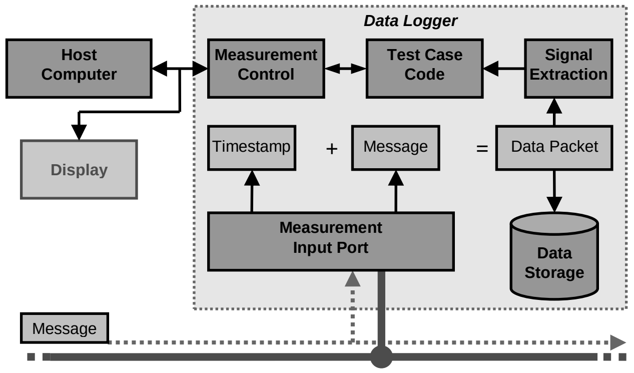

Figure 5. Schematic measuring system extended with the test case code

To control and configure the data logger the test case needs a connection to the measurement control module. On the first hand, the measurement has to be started at the beginning of a test case and stopped when it has ended. On the other hand, the measurement control module is responsible for monitoring the execution of the test case. In detail the measurement control module compares target values defined in the test case (in the following called “rules”) with the corresponding signals transmitted on the vehicle bus. Furthermore, the measurement control module generates instructions for the driver depending on the current test step within the test case. These instructions are extracted from the test case and are provided to the driver, e.g., via a display. The ADL version 1 has an attached display that shows only human readable text generated from the machine readable test case description.

Figure 6 shows the testing process corresponding to an augmented datalogger. The test case is supplemented with instructions for the driver and with conditions for being valid. Based on this the test engineer is guided through the test while driving the car and the evaluation of the test case for being executed correctly is done by the data logger on the fly. Immediately after a violation of a rule within a test case the test driver gets informed and has the choice whether to finish the manoeuvre or stop immediately and start from the beginning.

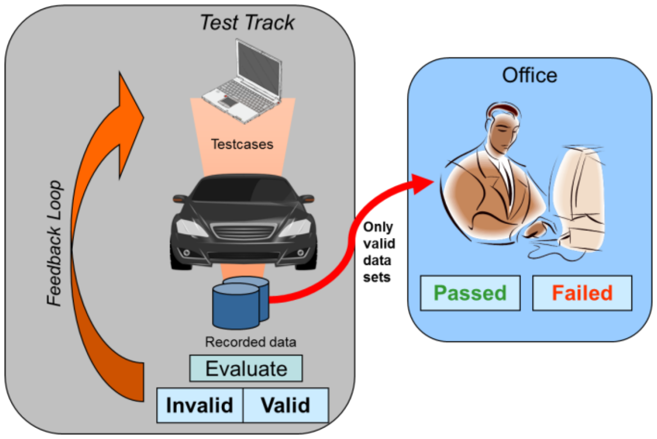

Figure 6. Optimized testing process

C. *Test Case Implementation and Execution*

In this section, the test case implementation and execution is shown using the following example:

Test Step 1:	Start engine
Test Step 2:	Accelerate to 60 km/h
Test Step 3:	60 km/h reached?
Test Step 4:	Full braking
Rule:	Steering wheel straight

Such a manoeuvre is used, e.g., to measure data of an Anti-Blocking System (ABS) and to evaluate if it has performed accurately during its intervention. A possible criterion for an invalid ABS-test execution is defined by looking at the steering angle. If the data show that the car did not drive straight, the test case has not been executed accurately. The manoeuvre can be described in a state chart manner represented by an XML (Extensible Markup Language) file [13].

The example in Figure 7 shows the ABS-manoeuvre in XML code. The definition of the XML code is described by Ruf [14] for Hardware in the Loop tests. The test case is composed of states, actions, events and rule. For the above test case the rule checks the steering wheel angle during the whole test case. The states are following in chronological order. Each state has one or more actions that have to be performed by the test driver. If the condition of an event is fulfilled the state machine enters the next state.

```
<?xml version="1.0" encoding="UTF-8"?>
<Testcase xmlns="http://www.ebtb.de/adl"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ebtb.de/adl
http://www.ebtb.de/adl">

  <Rule SteeringAngle_deg_eqal="0" Tolerance_deg="5"/>

  <State num="1">
    <Action text="Get ready to start the manoeuvre"/>
    <Event wait_seconds="5"/>
  </State>

  <State num="2">
    <Action text="Start the engine"/>
    <Event wait_seconds="5"/>
  </State>

  <State num="3">
    <Action text="Accelerate to 60km/h"/>
    <Event velocity_kmh_equal ="60"/>
  </State>

  <State num="4">
    <Action text="Full braking"/>
    <Event velocity_kmh_equal ="0"/>
  </State>

  <State num="5">
    <Action text="Turn-off engine"/>
    <Event wait_seconds="5"/>
  </State>

  <State num="6">
    <Action text="Manoeuvre finished"/>
    <Event wait_seconds="3"/>
  </State>

</Testcase>
```

Figure 7. Listing of a test case in XML

D. *Case Study*

After having implemented a prototype of the described data logger with its additional features, a case study has been performed to determine the benefits of the augmented measurement system for test drivers. The case study was conducted with a group of eleven candidates. The group consisted of team leaders, developers and testers. In the first step, the content of the executed XML test case was explained to the candidates. With this knowledge the candidates were guided by the augmented measurement system to execute the test case in the role of a test driver.

1) *Manoeuvre*

The selected manoeuvre for the case study was more complex than the sample test case in Figure 7. The test addresses the safety deactivation of the cruise control when engaging the hand brake. For one test case example the test steps are as follows:

Test Step 1:	Start engine
Test Step 2:	Accelerate to 50 km/h
Rule:	Speed less than 55 km/h
Rule:	Steering wheel straight
Test Step 3:	Activate the Cruise Control with the "SET" button
Test Step 4:	Remove foot from acceleration pedal
Rule:	Don't turn the Cruise Control lever up
Comment:	*Cruise Control active*
Test Step 5:	Engage the hand brake
Rule:	Speed more than 45 km/h and less 55 km/h
Comment:	*Cruise Control disabled*
Test Step 6:	Decelerate to zero
Test Step 6:	Turn off engine

In this test case, the Driver should accelerate to the target speed of 50 km/h. But in the test the driver should not accelerate to a speed greater than 55 km/h and should not turn the steering wheel.

In most cars with a Cruise Control lever, the function can be activated on two ways:

1. Pressing the "SET" button to use the current speed as reference
2. Tip the lever up to activate the Cruise Control and accelerate or to resume to the speed set before

The implementation might differ between manufacturers. But if there is more than one way to activate the Cruise Control the test case shall address exactly one. The other ways are different test cases.

Engaging the hand brake turns the vehicle into the following situation: The brake leads to a vehicle deceleration. If the Cruise Control will not be disabled the controller tries to match the speed that is set and accelerates. To avoid this situation the Cruise Control has to be disabled if the hand brake is engaged.

2) *Feedback*

The vehicle for the case study was equipped with an extra display that is attached to the windscreen. The setup in the vehicle looks similar to an external navigation system. In this setup, the display shows the instructions and the current state of the running test case. The execution of the test case was done on a locked test track. This ensures a save environment and that the candidates are not disturbed by surrounding vehicles.

After executing, the test cases multiple times the candidate was interviewed about his experience with the augmented measurement system. The collected feedback is summarized in Figure 8.

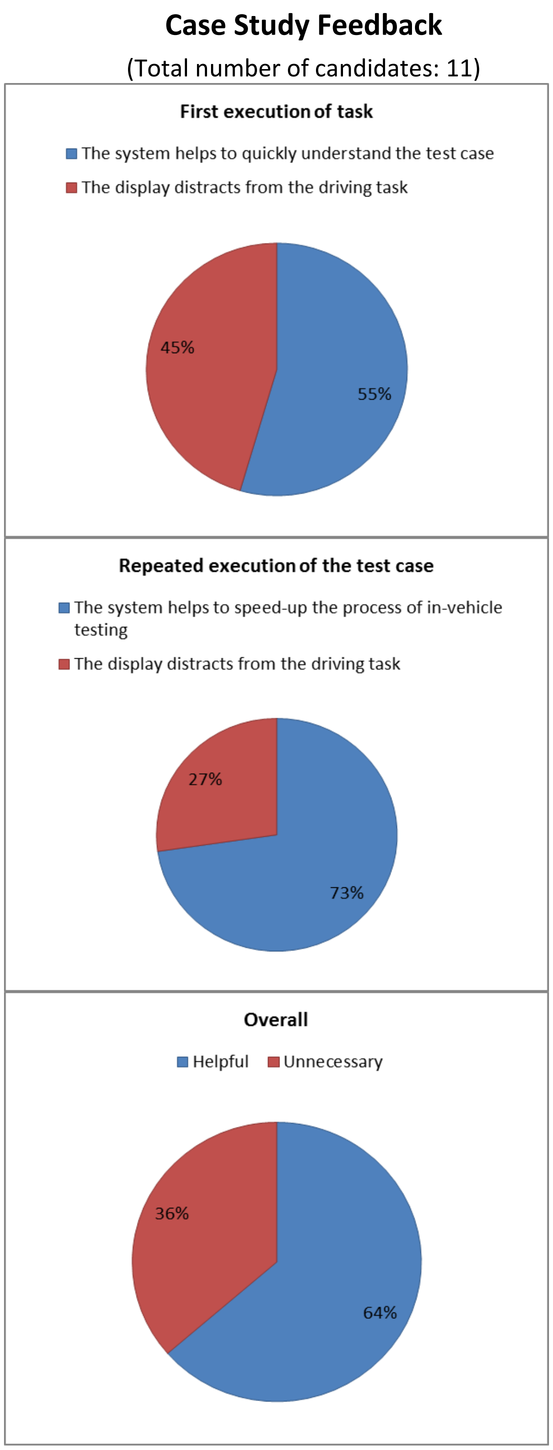

Figure 8. Case Study feedback results

Most candidates are confused and distracted by the information shown in the display driving the test case for the first time. The reason might be that the candidates do not yet intuitively follow the instructions on the display. As soon as the instructions are known to the candidate he can concentrate less on the display and more on his driving task. After a short learning curve the confidence and sureness working with the augmented data logger raised. In summary, 7 candidates are seeing a benefit of such a system to speed up and assist them in their daily work.

Furthermore, the feedback also includes suggestions for improvements. The four mostly mentioned suggestions were:

- Additional speech output for instructions
- Direct connection to quality and lifecycle management tools
- More detailed information in case of a invalid result
- Using LCD glasses instead of a display attached to the windscreen
- Adding a test case automation for processing several test cases in a sequence

The feedback of the case study indicates that the augmented data logger helps to speed-up the testing process for in-vehicle testing.

IV. Augmented Datalogger Version II

The second version of the ADL has several improvements. The display shows more detailed information of the current state. To handle these information most actions, events and rules are displayed as icons. This might increase the reaction time of the driver while executing a first manoeuvre but after getting used to the icons they can be instantly understood. The icons have been designed in cooperation with the University of Applied Sciences Karlsruhe. They shall be intuitive to new drivers. It is planned to add the speech output for instructions at a later time and optimize the required visual aspect. As an additional feature in the second prototype a testing automation has been implemented. The driver has the possibility to execute several test cases in a sequence. Finally, the implementation can filter test cases from lower integration levels and skip test cases that are not suitable for the in-vehicle setup.

Another major improvement of the ADL version II is the way how test cases are loaded in the data logger. The first version generates code from an XML test case description and executes that code on the data logger. In the second version, the XML test case is transferred to the data logger. The data logger interprets the test case and executes it directly. This is a big benefit because the code generation step is no longer necessary.

A. Display icons

Figure 9 shows an example how a test case state might look on a display. Two actions should be executed:

- Switch gear selector in state “D”
- Accelerate to 80 km/h

The event occurs if 75 km/h are reached. The grey number indicates the degree of fulfilment. The two "R" inside the squares depict the active rules:

- Steering wheel straight ±5 degrees (Tolerance is not shown in Figure 9)
- Speed less than 85 km/h

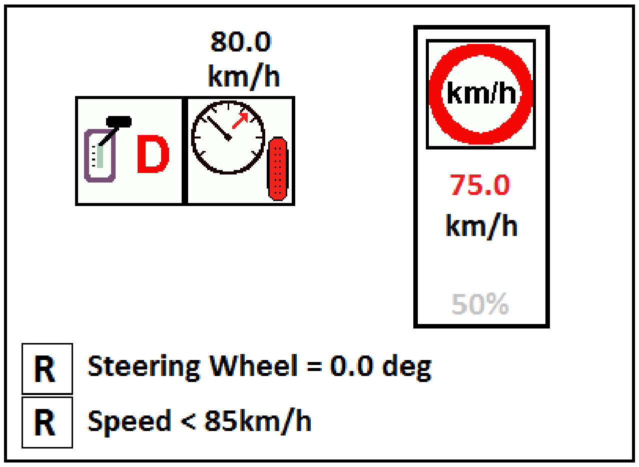

Figure 9. Possible display screen with actions, rules and one event

If a rule is not fulfilled the test case execution stops and shows a red screen with the broken rule. If all test steps are executed and no rule is broken the driver gets a green screen. Both the green and the red screen terminate with the test automation screen.

B. *Test Automation*

To give the test driver the ability to easily switch between the available test cases on the data logger, a test case automation system [15] was implemented and is shown in Figure 10.

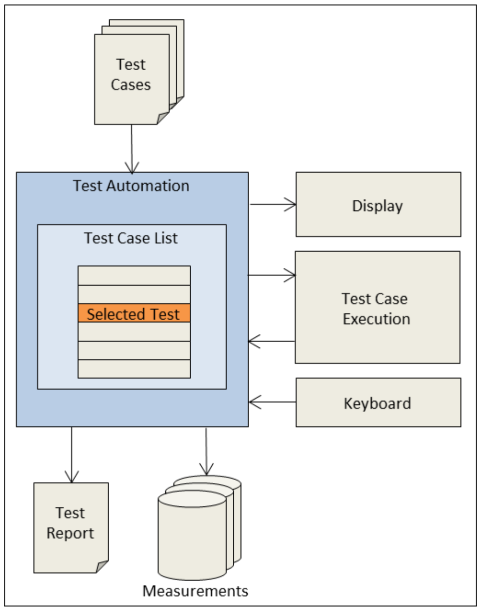

Figure 10. Test case sequence automation

The test driver can select a list of test cases for execution and upload these test cases to the data logger. Beginning with the first test case the driver performs all tests one after another. If the test is passed, the following test case is loaded for execution. In case of a failure, the driver has the choice to drive the test case again or to go to the next test case.

The results of the test runs are stored and will be presented in a report showing the valid and invalid test cases. The measurements are stored for both cases, valid and invalid tests. Running one test case multiple times will produce multiple test reports and measurements. It is up to the test engineer to select the relevant reports and measurements for evaluation.

C. *Test Case Filters*

As explained in the sections before, the test cases for in-vehicle testing can be reused from lower integration levels. Since the underlying test case language is manoeuvre-based, a large number of test cases can be reused without any change. But there are test cases that cannot be reused directly.

One reason is that test cases might use special actions that cannot be performed in each vehicle setup. These types of test cases can be called "platform dependent test cases". For example on a HiL platform the bus signals sent from the HiL's bus interface can be manipulated easily because the HiL simulates all other ECUs on the specific bus. In a vehicle these ECUs are existent. This means that the vehicle needs a special hardware that separates the bus between the SuT and all other ECUs on the bus. This hardware manipulates all incoming messages and signals to the SuT as specified in the test case. If this hardware is not available within a certain test vehicle a test case, which needs this signal manipulation cannot be executed.

Another example is a hardware interface manipulation. Some HiL platforms are able to apply hardware errors to the interface of the SuT such as a defective contact. These tests are platform specific and as well as can only be tested automatically inside a vehicle if the corresponding manipulation hardware is installed.

Complex test cases are using a predefined environment to test driver assistance functions. The environment can be a given driving track or surrounding objects like other vehicles, motor cycles and trucks. Lower integration levels are using simulated sensor ECUs to simulate the environment. Inside a test vehicle the sensors are real and will detect the given environment. A basic example is an adaptive cruise control manoeuvre. The test vehicle is behind another vehicle — called object vehicle. The object vehicle accelerates or decelerates and the test vehicle should do the same automatically to keep a safe distance between the two vehicles.

One approach for testing the acceleration algorithm in a vehicle is replacing the sensor ECUs with the environment simulation of a HiL. These test cases can only be executed in an in-vehicle test if the vehicle is modified as described. Another idea, which is left for future work is to distribute the test case to several ADLs in several vehicles.

The three examples show the ADLs ability of executing a test case is depending on the content of the test case and the setup of the vehicle. With the help of a filter test cases can be automatically sorted corresponding to the required test equipment.

A different use case for applying a filter is the way how the test engineers write the tests. As explained above, the test cases can have a variable amount of actions in a state that will be performed simultaneously. On SiL/HiL platforms it is possible to perform many actions at the same time because the simulated driver can perform the actions simultaneously. A real test driver gets all the information what he has to do in a certain state at once and has to perform all these tasks as fast as possible. The more actions are within a test case state the more tasks the test driver has to execute. He has to gather all the information and perform the required physical actions. The risk to forget to execute an action rises with the amount of actions in a state.

One result of the work with the University of Applied Sciences Karlsruhe has been that there should be only one action in a state, which has to be executed by the driver. Skilled drivers might be able to execute up to three actions. Since the test cases can be executed on HiL-platforms as well, it is important to know that Actions are executed on a HiL platform immediately after entering the state. A human driver has recognition and response times. These times are rising with each additional action. First tests and the case study show a very high range of the reaction times. These reaction times may be critical to get a valid test case result. An analysis of existing test cases showed that even the HiL test cases are modelled with a maximum of three driver actions per state. This means that the display layout can be optimized to show one to three actions at a time. Due to these limitations the number of actions within a state is limited to three for test cases that shall be used on vehicle level. Test cases with more than three driver actions per state are refused. An example, which shall clarify this statement, is the following test case:

1. Switch the gear selector to “R”
2. Press the turn indicator to right blinking
3. Press the brake pedal
4. Release the tightened parking brake

For a human driver it is almost impossible to perform all these actions simultaneously. In this case, the test case writer has to check the test specification whether the actions can be split into two states without altering the expected test case results. According to our experience splitting one state with several driver actions into several states with one driver action is mostly possible.

There are two ways to address the limitation of driver actions within the software tools, either globally limiting the allowed amount of driver actions within one state to three or by adding a filter to the in-vehicle test automation, which suggests skipping test cases with more than three driver actions. One topic for future work is to automatically detect and convert these test cases and to inform the author immediately after writing such a test case that an in-vehicle execution is not possible. Based on this approach, one future work might be to automatically forecast the dynamic criticality of a test case. The dynamic criticality is a factor that indicates how risky the execution of the test case is for the driver himself and the surrounding environment. Manoeuvres marked with a high dynamic criticality have a high potential for injury and vehicle or environmental damage. For example, a test case that requires high deceleration or gear movements at high velocities might be automatically marked as dangerous and only suitable for adequate test tracks.

This approach will enable the test automation to filter the test cases for the required environment. For example, all test cases can be executed, which have to be driven on a high-speed track. A first prototype of this semantic filter of the test cases has been implemented. Defining a metric for the combination of all driver actions and its evaluation is left for future work and prototypes.

V. Conclusion and Future Work

This work shows an approach how the process of in-vehicle testing can be improved. The introduced approach shows a way to reduce the costs for the testing process by reusing test cases from other testing platforms and by optimizing the workflow of in-vehicle testing. As a rule of thumb, we experienced that for complex testing scenarios comprising about 100 test cases over 30 per cent of the test cases are invalid when they are evaluated manually after test execution. A major part in the optimized workflow is the possibility for declaring a test case invalid.

The extended classification of a test case enables an early feedback about the quality of the executed test case and hence makes sure that only valid test cases are evaluated. In the introduced approach, a test case can be classified as "passed", "failed", "valid" and "invalid". The first two classifications are based on the requirements and can only be evaluated if the data is valid for the SuT, while the other two classifications reveal if the test case is executed within defined constraints that are based on additional testing requirements. The test engineer has only to look at the measurements of the test cases that are classified as valid. This helps to reduce the evaluation time especially if the test case manoeuvre is very complex or time critical.

A first prototype of the Augmented Data Logger has been discussed, which allow to use test case descriptions from lower integration levels and use them as a basis for the in-vehicle test. The test engineer needs no knowledge in programming languages for implementing and running a test case on the introduced augmented data logger.

While driving a test case the test driver has precise instructions on his current tasks and is guided through the test case manoeuvre. The test driver has immediate feedback if the constraints of the test case added by the test engineer are fulfilled. The augmented data logger observes the execution and the driver gets a response if the manoeuvre is valid or if the test driver has made a mistake during the execution. It is then up to the test driver to decide if he wants to immediately repeat the manoeuvre or continue with the next test case.

A case study shows that the approach is useful and has potential for improvements. The second version of the ADL improves the visual recognition by using icons instead of text messages. The tool chain has been extended by a test automation that supports the driver by defining test sequences that can be executed at once.

The use of test cases from lower integration levels shows that they can be reused if the technical conditions are met. To detect these conditions the idea is to implement filters for the test cases. A filter can select the test cases that are suitable to run in the test vehicle.

For future work, a distributed ADL can be considered to support the in-vehicle test of advanced driver assistance systems where several vehicles are involved. Furthermore, augmented reality glasses instead of a display might be considered for informing the test driver. A semantic interpretation of the test cases might help to forecast the dynamic criticality of a manoeuvre and to recommend a test track.

REFERENCES

[1] K. Hünlich, D. Ulmer, S. Wittel, and U. Bröckl,, “Optimized testing process in vehicles using an augmented data logger”, IARIA ICONS Conference, Febuary 2012, ISBN 978-1-61208-184-7

[2] K. Athanasas, “Fast prototyping methodology for the verification of complex vehicle systems”, Dissertation, Brunel University, West London, UK, March 2005

[3] S. McBeath, “Competition car data logging: a practical handbook”, J. H. Haynes & Co., 2002, ISBN 1-85960-653-9.

[4] L. Petersson, L. Fletcher, and A. Zelinsky, “A framework for driver-in-the-loop driver assistance systems”, Intelligent Transportation System Conference 2005: Proceeding of an IEEE International conference Vienna (Austria), September 2005, pp. 771 – 776.

[5] E. Meier, „V-Modelle in Automotive-Projekten, AUTOMOBIL-ELEKTRONIK“, Journal, February 2008, pp. 36 – 37.

[6] M. Schlager, „Hardware-in-the-Loop simulation“, VDM Verlag Dr. Mueller e.K., 2008, ISBN-13: 978-3836462167.

[7] mm-lab, “Driver guidance system”, Automotive Testing Technology International, September 2009, page 89.

[8] H-P. Schöner, S. Neads and N Schretter, “Testing and verification of active safety with coordinated automated driving”, NHTSA ESV21 Conference 2009, http://www-nrd.nhtsa.dot.gov/pdf/esv/esv21/09-0187.pdf

[9] J. Park, and S. Mackay, “Practical data acquisition for instrumentation and control systems”, An imprint of Elvester, 2003, ISBN-10: 075-0657-960.

[10] M. Koch, and A. Theissler, “Mit Tedradis dem Fehler auf der Spur”, Automotive Journal, Carl Hanser Verlag, September 2007, pp. 28 – 30.

[11] D. Ulmer, A. Theissler, and K. Hünlich, “PC-Based measuring and test system for high-precision recording and in-the-loop-simulation of driver assistance functions”, Embedded World Conference, March 2010.

[12] S. Dangel, H. Keller, and D. Ulmer, “Wie sag’ ich’s meinem Prüfstand?”, RD Inside, April/Mai, 2010.

[13] B. Ruf, H. Keller, D. Ulmer, and M. Dausmann, “Ereignisbasierte Testfallbedatung - ein MINT-Projekt der Daimler AG und der Fakultät Informationstechnik”. spektrum 33/2011, pp. 68–70.

[14] B. Ruf, H. Keller, D. Ulmer, and M. Dausmann, “Ereignisbasierte Testfallbedatung”, Spektrum 33/2011, pp. 67 – 68.

[15] M. Spachtholz, “Mission Control - Automatisiertes Testen von Fahrerassistenzsystemen im Fahrzeug”, Bachelor Thesis, University of Applied Sciences Esslingen, 2012