Robustness Testing of AUTOSAR Software Components

Master of Science Thesis
Computer Systems and Networks Programme

VICTOR JANSSON
JERRY LINDAHL

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden, 2013

Robustness Testing of AUTOSAR Software Components

VICTOR JANSSON
JERRY LINDAHL

Examiner: JOHAN KARLSSON

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone +46 (0)31-772 10 00

Cover:
A photo taken by Lothar Schaack from the VW factory test track in 1973
Bundesarchiv, B 145 Bild-F038805-0031 CC-BY-SA
License: Creative Commons Attribution-Share Alike 3.0

Department of Computer Science and Engineering
Gothenburg, Sweden June 2013

Abstract

The increasing complexity of software in modern automotives is currently leading to a greater interest in robustness testing of software components. Furthermore, the introduction of industry standards within functional safety, such as ISO 26262, raises the need for methods suitable for software testing within the latest frameworks. In this master's thesis an automatic tool for robustness testing of AUTOSAR software components (SW-C) is presented. By using the interface specification delivered with every SW-C the tool executes a robustness testing campaign based on the data types of each port of the interface. As an interface specification is attached to every SW-C, both white-box and black-box components can be tested. A wrapper surounding the SW-C under test injects input selected from a library consisting of carefully selected values. The tool demonstrate that data-type based robustness testing is a highly suitable method for future software development in the AUTOSAR standard.

Keywords: ISO 26262, AUTOSAR, robustness testing, fault injection, software components

Acknowledgements

We would like to express great appreciation to our supervisor Prof. Johan Karlsson at Chalmers University of Technology. We are particularly grateful for the assistance given by Johan Haraldsson and Mafijul Islam who continuously helped the project move forward by giving both encouragement and technical assistance. It was a joy working at Volvo Group Trucks Technology and with the staff at the department of Electrical and Embedded Systems of Advance Research and Technology who provided all we needed and also allowed great freedom during the project. A special acknowledgement is given to Vinnova who provided financial support and made the project possible.

Contents

1 Introduction

An important trend in the modern automotive industry is the rapidly increasing amount of software in new cars, trucks, and buses. Common expectations are that this trend will continue for future decades [1], and as more and more features are introduced through software the system complexity increases. Furthermore, as digital systems are given responsibility over essential and safety related functions, the requirements for their reliability grow. This shift towards increased reliance on digital systems and software has encouraged the development of the industry-wide ISO 26262 standard [2] which aim to maintain the functional safety of road vehicles. ISO 26262 concerns the design, development, and production process of electric/electrical (E/E) systems and provides requirements on the complete life-cycle of E/E systems within road vehicles.

AUTOSAR [3] is an evolving standard for software development in the automotive industry and is motivated by the increased complexity of automotive E/E systems. It is designed as a new approach to software development within vehicles and aims to deal with several issues which the industry faces, including compliance to ISO 26262. The automotive industry considers AUTOSAR to be an important step towards a modular approach to software development [3]. Modularity facilitates code reuse and integration of third-party components which is likely to reduce the development cost when introducing new features.

ISO 26262 require robustness testing of the software units of a system and the industry see a need for tools performing robustness testing of AUTOSAR components. Robustness of systems and software is an established area of research in academia. In 1996 a DARPA funded project called Ballista [4] was started at the Carnegie Mellon University. The project aimed primarily on robustness testing of operating system interfaces by looking at the data types of the parameters to each interface. This testing method proved to be successful at finding defects even in supposedly well tested and robust interfaces. Koopman et al. [4] conclude the choice of basing test values on parameter data types was especially successful.

AUTOSAR components have clearly defined interfaces and communication is provided by a virtual bus. This enables middleware to monitor and modify communication between software components, which allows implementation of robustness testing. A successful robustness testing suite will be useful to developers for improving robustness and functional safety of software, or for evaluation and comparison of component robustness when deciding upon the component suitability.

In this thesis we present a tool which can perform robustness testing of

AUTOSAR SW-Cs using a technique inspired by the Ballista project. We call this technique data-type based robustness testing as test values are based on the data type specification of the interfaces to SW-Cs.

1.1 Research Question and Scope

The focus of this master's thesis is to examine the possibilities of the data-type based approach to software robustness testing within the AUTOSAR framework. We do this by presenting a robustness testing tool for AUTOSAR software components. In the tool the test cases are automatically generated by analyzing of the interfaces specified in the AUTOSAR software components' XML interface specification.

1.2 Related Work

The first research covering fault injection can be traced back [5, 6] to Mills [7] work at IBM in 1972. For two decades fault injection was mainly used as a way to emulate hardware faults. In 1996 Koopman et al. [4] started the Ballista project at Carnegie Mellon University. Software faults were intentionally injected into operating systems APIs for robustness testing. The Ballista projected was considered successful as it found several vulnerabilities in commercial software. Koopman also introduced the CRASH scale in [8] which is a common way for grading the severity of robustness vulnerabilities. Yu et al. [5] published in 2003 a paper summarizing the state of the art of fault injection as by then. The definitions given in that paper are still highly useful. Shahrokni and Feldt [9] recently published a systematic overview of research on software robustness and conclude the need for research on real world systems. Cotroneo et al. describes in [10] a method of assessing the accuracy when performing software fault injection on black boxes compared to white boxes. The accuracy of the injection of a specific software fault is lower when working with compiled binary code compared to when the source code is available. Fuzzing [11] is another approach to robustness testing where random input is directed into applications. A drawback of this approach is the large amount of excessive tests [12]. Lu et al. [13] introduces a framework to design robust software for multi-layered software in the automotive domain such as AUTOSAR. Piper et al. discuss how an AUTOSAR dependability assessment may be performed of AUTOSAR components by instrumenting either source code, header files or object files [14]. In [12] a methodology to evaluate the quality of commercial off-the-shelf (COTS) is presented by Voas. The use of system-level fault injection is promoted for such an evaluation.

The work presented in this report builds upon the conclusions and implementations from the master's thesis by Haraldsson and Thorvaldsson [15]. They evaluated software fault injection to be a viable technique for AUTOSAR based systems. The authors to this report recognize a need for easy and practical robustness testing of such systems in a real world setting.

1.3 Stakeholders

The Department of Electrical and Embedded Systems of Advanced Technology and Research at Volvo Group Trucks Technology is currently taking part in a research project called BeSafe which aims *"to identify benchmark targets in automotive electronics, define benchmark measures and a methodology for performing and using such benchmarks"* [16]. BeSafe is funded by Vinnova (Swedish Governmental Agency for Innovation Systems) and the consortium consists of six partners: Volvo AB, Volvo Cars Corporation, Scania AB, QRTECH, Chalmers University of Technology and SP Technical Research Institute of Sweden. This thesis aims to assist the goal of BeSafe by investigating data-type based fault injection techniques for AUTOSAR software components.

1.4 Project Procedure

Peffer et al. [17] presents a methodology with six activities suitable for conduction of design science research in information systems. This methodology is constructed as a tool for research where the creation of an artifact is in focus. The six activities, shown in figure 1, and often referred to as steps, include problem identification and motivation, definition of the objectives of a solution, design and development, demonstration, evaluation, and communication. This master's thesis is initiated by the stakeholders who have identified and motivated the project. The authors of this thesis have together with the stakeholders agreed upon the definition of the objective. The design and development is an important and essential part of our project process as we implement a proof of concept software, i.e., the previously referred to artifact. The ordinary procedure at Chalmers University of Technology deals with academic demonstrations, but meetings with the supervisors are also part of the demonstration step. Evaluation is done on a regular basis by supervisors, the authors of this thesis, stakeholders and ultimately the examiner. The printed thesis together with the oral presentations represents the last activity, namely communication. Peffer et al. [17] explains further that depending on the nature of the project, the first step isn't necessarily

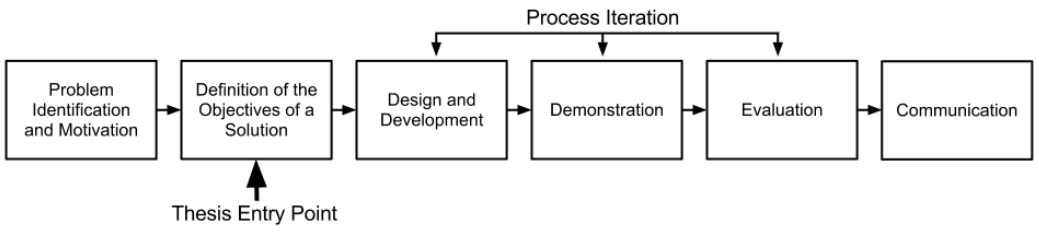


FIGURE 1: The methodology presented by Peffer et al. in [17].

the first activity. In this case the entry point for this thesis is the second step, namely the definition of the objectives of the solution.

1.5 Report Outline

An introduction to the field of dependable computing is given in chapter 2 by a brief description of the common taxonomy. Chapter 3 explains important techniques and concepts related to the work of this thesis. Chapter 4 introduces the tool that has been developed to perform data-type based robustness testing of SW-Cs in AUTOSAR. An evaluation of the tool is further given in chapter 5. This is followed by a discussion in chapter 6 and the conclusions are highlighted in chapter 7. Finally some proposals for future work are given in chapter 8.

2 Taxonomy of Dependable Computing

This chapter will introduce common terminology used in the field of dependability and is relevant for the understanding of this report. It is pointed out by Shahrokni et al. [9] that critical systems deals to great extent with the quality attribute dependability. Avižienis et al. [18] defines dependability as the “ability of a system to avoid service failures that are more frequent and more severe than is acceptable”. The dependability of a system can be described using the following attributes:

- **availability:** readiness for correct service.
- **reliability:** continuity of correct service.
- **safety:** absence of catastrophic consequences on the users(s) and the environment.
- **integrity:** absence of improper system alterations.
- **maintainability:** ability to undergo modifications and repairs.

Dependability is by Avižienis et al. [18] given a specializing secondary attribute referred to as robustness. The definition of robustness generally conformed to is stated in IEEE Std 610.12 [19] as “the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions”. Thus robustness can be seen as an extension to the dependability attribute of software in presence of invalid input [9]. Fault injection can be used as a validation technique of the robustness of software. Faults are intentionally injected into a system under test and the following behavior is observed [5]. Fault tolerant mechanism for known faults can thereby be tested effectively and systematically.

The root of a service failure is always a fault followed by an error. This cause-effect chain is sometimes titled the pathology of failure [18] and is shown in figure 2. A fault can be introduced to a system in many ways e.g., by broken wires or solder, corruption of memory, or originate from a faulty system design. Additionally, several faults and errors may be the source of just one service failure. Note that a fault or error does not necessary imply a service failure. A reason for this may be that the faulty value is never read, or it is overwritten by a correct value before it’s used. A fault is dormant if it is not currently the cause of an error. An active fault is on the contrary a fault which is causing an error.

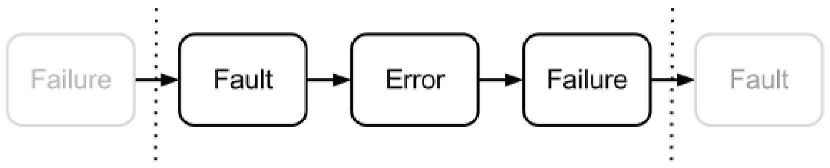


Figure 2: The failure pathology

Faults can be categorized into physical faults, development faults and interaction faults. Physical faults are further classified into permanent, transient, or intermittent faults. A permanent fault is caused by an irreversible damage, and is therefore present permanently. Transient faults are caused by environmental disturbances, e.g., electromagnetic interference. Intermittent faults come from unstable hardware which alternates between correct behavior and incorrect behavior. Development faults are in all cases the result of incorrect design [5]. It is a common experience to software developers that 'bugs' may lie dormant for long periods of time and manifest as errors for the first time during execution in exceptional situations, such as heavy load. This cause these faults to potentially be overseen during the development phase. Interaction faults are fault introduced during operation of a system. A system may implement functionality such as fault tolerance to avoid a fault to set the system to an erroneous or a system failure state.

Considering that a system is built upon smaller components a failure in one component can be considered a fault to another component or to the global system. The output of one misbehaving component may propagate a fault through a chain of components. This process is known as fault propagation.

3 Related Technologies

This chapter provides an overview of technical concepts used in this thesis and aims to facilitate the reader's understanding of the thesis.

3.1 AUTOSAR

AUTOSAR is a collective initiative by several large actors in the automotive industry and aim to be a shared software architecture framework within the industry. The reason behind the effort is the increased complexity of E/E systems as the functional scope of such systems is growing larger and larger. A mutually agreed upon architecture framework allows manufacturers to compete on functionality rather than architecture. The benefits of a shared platform include the possibility for a wide use of COTS. The manufacturers, but also independent specialist software companies, can develop general software suitable for the automotive market at large. The following subsections introduce the main concepts of AUTOSAR and how it improves flexibility, scalability and quality of E/E systems.

3.1.1 The AUTOSAR Abstraction

An AUTOSAR system is deployed onto one or several electronic control units (ECUs) and each ECU encloses one or several software components (SW-C). A shared physical bus enables communication between the ECUs by using the hardware interface provided by the basic software (BSW) component. The BSW consequently uses a communication protocol such as CAN, Flexray or LIN. The run-time environment (RTE) layer implements the virtual functional bus (VFB) which provides a uniform environment for SW-Cs communication. The VFB makes the system highly flexible as it allows SW-Cs to be transferred to other ECUs or be updated without the need of additional code changes. A schematic of an AUTOSAR system is shown in figure 4.

3.1.2 The AUTOSAR ECU Abstraction

AUTOSAR specifies a layered abstraction of the ECU as shown in figure 3. The top level encloses applications which are providing the end-user with functionality. Isolation of the application level allows applications to be developed independently of the surrounding environment. The applications use an interface to the RTE for communication between applications. The RTE also connect the applications to the operating system and hardware. In AUTOSAR the applications are implemented by one or more SW-Cs. [3]

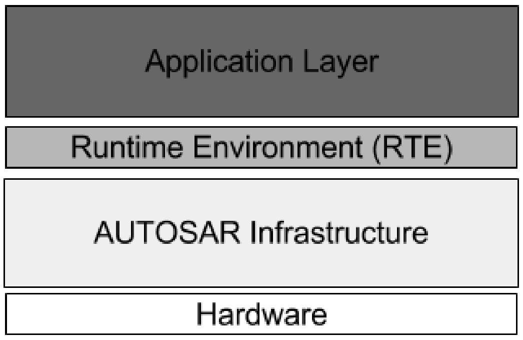


FIGURE 3: The layered architecture of AUTOSAR

3.1.3 Software Component

The SW-Cs are implementations of application functionality, however AUTOSAR does not specify the size of the SW-Cs. Depending on the situation SW-Cs can be implementations of a limited function or a large set of several functions. SW-Cs are always atomic and cannot be distributed over more than one ECU. It is important to note that AUTOSAR does not specify how to implement a SW-C but merely delivers the framework for successful integration of SW-Cs over the system of ECUs. Delivery of a SW-C always comes with a formal specification of the infrastructural dependencies required by the SW-C via XML files. The specification is used when building the complete AUTOSAR system.

3.1.4 Software Component Communication

Each SW-C includes a well-defined interface for inter-component communication. The interface consists of ports that are either providing (PPort) or requiring (RPort) data. The SW-Cs need to implement either a Client-Server or a Sender-Receiver communication pattern for each port. Both communication patterns are well known techniques for data communication. In the Client-Server pattern the initiating actor is always the client which requests a service from a server. The client may be blocked until the service is delivered by the server (synchronous), or it may continue its execution independently of the server (asynchronous). The Sender-Receiver pattern facilitates multicasting, i.e., asynchronous messaging from one sender to multiple receivers. The receivers may or may not act on incoming messages.

3.1.5 Run-time Environment

The run-time environment (RTE) provides an interface for the SW-Cs to the operating system, hardware and other SW-Cs. The RTE is customized automatically by an RTE generation tool for each ECU to support the SW-Cs dedicated to that ECU. An AUTOSAR standardized XML file specify each SW-C and is used during the RTE generation. The RTE is customized according to which ports are used and which SW-Cs the actual SW-C need to communicate with. The implementation of the RTE is dependent on which AUTOSAR vendor tool is used. Furthermore a vendor tool will generate the RTE differently based on which options are used. Such an option include optimizations of the generated code which is explained further in section 3.1.7.

3.1.6 Virtual Functional Bus

The aggregation of several RTEs on different ECUs implements a distributed communication bus, the VFB. The VFB enables software components to communicate independently of the underlying hardware and the organization of ECUs. In the perspective of one SW-C it does not matter on which ECU it is executed, the VFB will provide the means necessary for communication with the required components. An additional benefit of the VFB is the possibility for relocation of SW-Cs to other ECUs. Relocation of SW-Cs enhances the modularity of AUTOSAR and is useful for resource balancing between ECUs.

3.1.7 Optimizations during RTE Generation

When several SW-Cs running on the same ECU communicate internally their ports are normally realized as function calls. This can be optimized by the RTE generator. Instead of read and writes on ports realized as more expensive function calls, the RTE generator may replace function calls with less expensive macros. These macros make use of global variables which are more efficient but also less flexible for fault injection. Furthermore the macro optimization is vendor specific which makes automatic generation of fault injection more complex. To make a wrapping based approach more workable this optimization of the RTE can be disabled. One additional strong reason for disabling optimization is that the generation relies on source code for the SW-Cs being available. This may not be the case when testing COTS. Optimizations are therefore discouraged.

The disabling of optimizations will cause overhead as more code will be generated, possibly taking up more space than available on the ECU. An-

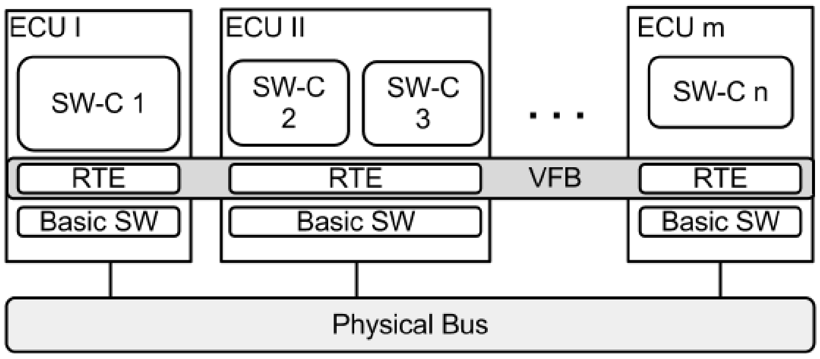


FIGURE 4: An overview of the AUTOSAR architecture as described in the AUTOSAR specification [3].

other cause of concern is that AUTOSAR software is often constrained by some form of real-time scheduling requirement. It is possible that disabling optimization can cause this scheduling to be delayed; therefore this effect must be considered for every new system tested.

3.2 Fault Injection

Fault injection is an effective technique for dependability testing and validation of E/E systems and components [5]. In a fault injection experiment a fault is intentionally inserted into the system under test and the consequent behavior observed. A serie of experiments is usually called a campaign. A campaign can be run on a simulation of a system, referred to simulation based fault injection, or on an actual deployed system, referred to execution based fault injection.

It is useful to differentiate between hardware fault injection and software fault injection. Hardware faults can be implemented by hardware-level modifications or by software emulating hardware faults. The injection of software faults implies testing the resilience for software design faults.

Additionally, the use of fault injection can be categorized as invasive or noninvasive. A noninvasive fault injection technique does not alter the system implementation in any way. The fault is directly transferred to the system, like in the case of pin-level injection or fault injection by ion radiation. However, it is at times impossible for practical reasons to not alter the implementation in any way when performing fault injection. Figure 5 shows the six attributes of a fault injection technique as grouped by [5].

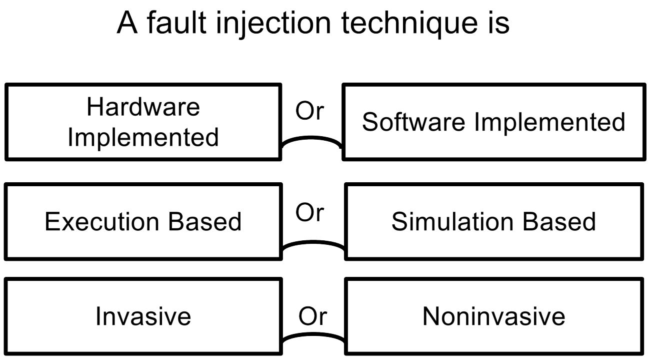


FIGURE 5: The categories of fault injection

3.2.1 The General Fault Injection Environment

With the maturity of fault injection techniques the common components of the fault injection environment have crystallized. The components as presented by Hsueh et al. [20] are enumerated below:

- **fault injector:** injects fault into the target
- **fault library:** a repository of faults
- **workload generator:** generates work for the target
- **workload library:** stores sample workload for the target
- **controller:** controls the experiment
- **monitor:** supervise and tracks the fault injection campaign
- **data collector:** performs data collection
- **data analyzer:** analyze and process the collected data

During construction of a fault injection tool all above components should be considered, as pointed out by Yu and Johnson [5]. The location of the components in our tool presented in this report is shown in figure 6. Depending on specific requirements components may be implemented on a separate machine or on the actual module under test (MuT). A component implemented directly on a MuT will naturally have a larger memory footprint on the MuT than if the component is implemented on a separate machine. However, code executing directly on the MuT may have greater abilities to alter inputs.

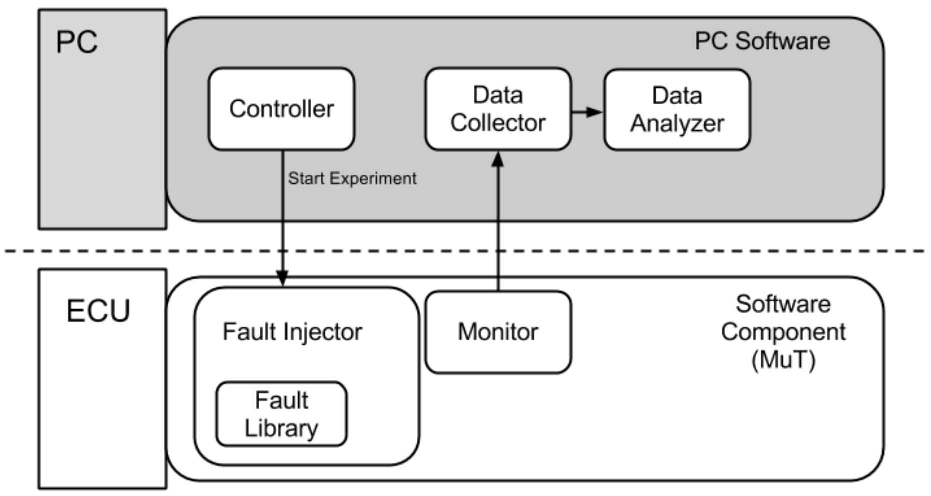


FIGURE 6: The architecture of our tool with respect to the components of the general fault injection environment.

3.2.2 Fault/Error Model

For any fault injection experiment it is of importance to choose an effective fault/error model. As described in section 2 there is a distinct differentiation between faults and errors, where a fault is the cause of an error. When discussing fault/error models this distinction is often quite loose. We will hereby refer to this concept as solely a fault model, even if the nature of the fault is an error.

A fault model can be seen as a pool from where faults are extracted. It is very hard to prove that the model correctly represent the real fault space. According to Yu and Johnson [5] it is common practice to assume that a fault model is sufficient and representative to the greatest extent possible with the experiment data, the historic data, or the results published in literature.

The effectiveness of a fault model can be measured by e.g., implementation costs (time or memory) and the details of the results. Three prominent classes of fault models in the field of robustness testing are bit-flip, fuzzing, and data-type based models [21]. In the bit-flip model the impact of a fault is simulated from changing one or several bits of a parameter. The bit-flip model closely relates to hardware errors where a stored or transmitted bit for an often unknown reason is changed, resulting in an incorrect value. Fuzzing implies the injection of random values as faulty input parameters. In the data type fault model faults are chosen dependent on the data type specification of the parameter. The data type fault model doesn't include randomness but comes with a predefined list of interesting fault values for each data type.

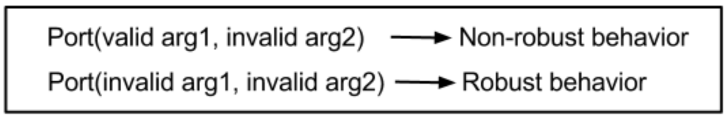

Port(valid arg1, invalid arg2) → Non-robust behavior
Port(invalid arg1, invalid arg2) → Robust behavior

FIGURE 7: An example of fault masking for a function with robust handling of invalid inputs only of the first argument

3.3 Selection of Test Values

Testing all possible combinations of input is in any non-simple system a much too time consuming task to be feasible. The goal of down-scaling a campaign is to keep the coverage of faults high while using a smaller set of inputs. Thus, a practical and scalable test campaign needs to limit the number of input values. This implies the need for a sophisticated selection of test values.

3.3.1 Behavioral Specification

According to Kropp et al. [22] four components are needed for an automatic generation of software tests. There must exist a MuT which is accompanied with a behavioral specification. A test generation mechanism is required and also a mechanism for comparing the behavioral specification with the results of the execution in the MuT. When testing COTS the complete behavior specification may be hard to obtain [22], as it is often unavailable. Fortunately, robustness testing can successfully be performed with a simple and generic behavioral specification [22]. Such a specification could simply state that a component should not crash or hang during operation.

3.3.2 Input Categorization and Fault Masking

Input can be categorized into two groups, namely valid input and invalid input. During a test campaign it is important to systematically generate input from both groups. It may be suggested that there is only a need for testing of invalid inputs. This could however result in lower coverage than expected. As a module can have several input ports, a fault on one port can be masked by a fault on another port. Kropp [22] demonstrates this behavior by a module taking two arguments, where the first argument is invalid. If the module returns an error, which would be customary for a robust node, the second argument might not be concerned. In the case that the module would show non-robust behavior with invalid input on the second argument that test case would be masked by the fault on the first input. An example of this situation is shown in figure 7.

TABLE 1: A data type test value table

Data Type	Test Values
Integer	0, 1, -1, MaxInt, MinInt
Float	0, 1.0, -1.0, $\pm$MaxDBL, $\pm$MinDBL, π, e

3.3.3 Data-Type Based Test Values

In data-type based robustness testing proper test values need to be selected for each type. The implementation of data types may differ between systems depending on, for example, choice of programming language or compiler. Therefore, knowledge about the specific implementation and use of a data type is needed for each system under test. Drawn from this knowledge input can be categorized as valid or invalid values. Common areas of interest are values around the boundaries of a data type. One example is the value between positive and negative values, namely zero. Kropp et al. [22] note that boundary values are identified from experience and give the following three criteria when choosing data values:

- implement at least one valid value
- implement at least one of each type of invalid value
- implement boundary values

Following the above criteria the first step is to choose valid input. In the case of an integer valid values include most values such as 2, 42, 64 and 1024. Selection of invalid input requires knowledge about how integers are used in the specific environment. If such knowledge is not known we can't choose any specific invalid values. Most values are selected from boundary values. This include 0, 1, -1, minimum and maximum integer values. A table with some of the normally interesting values [23] for testing of an integer or float port is shown in table 1.

3.4 Evaluation of Functional Safety

Safety is of great concern in the automotive industry and so is the need to display proof that substantial safety measures has been taken for a certain product. Functional safety and benchmarking are essential concepts for the development of safer vehicles. ISO 26262 defines functional safety to "absence of unreasonable risk due to hazards caused by malfunctioning behavior of E/E systems".

3.4.1 Functional Safety

The functional safety of a system is said to be valid according to a functional safety standard. The functional safety of a system can only be claimed after an independent reassurance that the requirements of the actual standard have been met. If such an evaluation is passed the systems is said to be certified to a certain safety level within the standard.

3.4.2 ISO 26262

ISO 26262 is a recently introduced functional safety standard for E/E systems in road vehicles. The standard introduces requirements for evaluation of the functional safety of an E/E system. ISO 26262 embody the full life-cycle of the system, including management, development, production, operation, service, and decommission of the system [2]. Product development at the software level is considered in Part 6 of ISO 26262. It includes requirements for initiation, specification, architectural design, unit design and implementation, unit testing, integration, and verification.

3.4.3 Benchmarking and BeSafe

Benchmarking of safety related properties for E/E systems in the automotive industry is essential for the development of safer vehicles. Benchmarking provides means of comparison between systems and components, facilitating proper selection of systems or components. BeSafe (see section 1.3) is a project initiated by several actors within the industry aiming to provide the foundations for benchmarking of functional safety. Consequently benchmarking is a way to tell if systems are fulfilling the requirements of standards such as ISO 26262.

4 Implementation

We examined the viability of data-based robustness testing in the AUTOSAR environment by extending an existing tool provided by Volvo GTT called DFEAT. DFEAT had the capability to extract and wrap the interface of a SW-C. The wrapping code allows DFEAT to trigger fault injection experiments on the SW-Cs from a PC and monitor its progress. We extended the functionality of DFEAT to include automatic data-type based robustness testing of AUTOSAR SW-Cs. DFEAT was also extended with an analyzing module that can analyze the progress of data-type based campaigns and report failures to the user.

4.1 Hardware Setup

The tool is executing on a PC connected to a physical ECU by two CAN-busses using a USB to CAN adapter. The ECU is using a Freescale MPC5517 CPU and is further connected to the PC via a debugger module (D.M.) via the JTAG interface of the ECU. The debugger module enables easy downloading of software onto the ECU and the use of additional debugger features, such as adding break points and monitoring variable values. The two CAN-buses isolate inter-ECU communication from communication between the PC and the fault injector residing on the ECU under test. Figure 8 shows the schematic of the setup.

4.2 Developement Software

The PC runs Microsoft Windows XP as operating system and the development environment for the tool is Microsoft Visual Studio 2010. The tool which generates AUTOSAR code in the C programming language was developed using C#. Additionally, the debugging features and downloading of software onto the ECU is enabled by TRACE32 PowerView R2010. The CAN-bus for the fault injector is monitored in the PC using CANoe 7.6. The full AUTOSAR system is generated using Vector DaVinci Developer 3.0.19.

4.3 AUTOSAR Systems Used During Development

During the development and testing of the tool we used two different AUTOSAR systems. The first system implements two SW-Cs providing the function of a simple calculator and an adder. The calculator sends the two terms to be added on the in-ports of the adder. The adder calculates the sum and

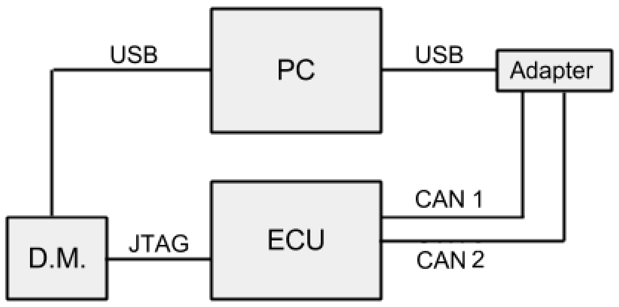


Figure 8: The schematic of the hardware setup

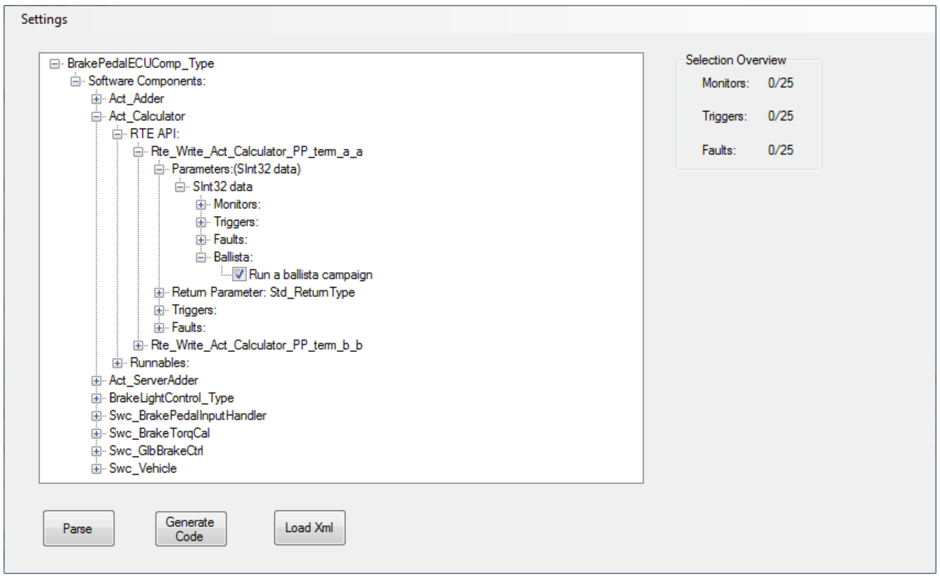
returns the sum on its out-port which is connected to the in-port of the calculator. The second AUTOSAR system was a more complex brake-by-wire system provided by Volvo GTT. The system consists of one ECU located at each wheel and one at the brake pedal. Each ECU executes several SW-Cs providing the required functionality such as braking the wheels, brake light indicator, wheel speed etc.

4.4 Tool Architecture

The basic components of a system fault injection tool are introduced in section 3.2.1. The schematics of the tool is shown in figure 6. Depending on the location of the components attributes such as intrusiveness, timing and controllability are affected. The tool is divided into two separate domains, the PC and the ECU. A PC program is configured for generation of wrappers for SW-Cs and additional code allowing for data-type based robustness testing. The compiled program from the generated code is flashed onto and executed on an ECU. The fault injector is located in code that is wrapped to the SW-Cs and consequently located on the ECU. Included in the wrapper is also functionality for monitoring, the wrapper continuously sends information on the proceedings of the experiment from the ECU to the supervising PC. The fault injector is using a fault library also residing on the ECU. On the PC the monitoring data is collected, allowing for analysis of the data. Located on the PC is also the controller which can start and stop an experiment.

4.5 User Process

The tool requires two steps by the user to perform robustness testing of AUTOSAR SW-Cs. The first step allow the user to select which SW-C ports to be tested and select a pre-written generic xml file describing the data-type selection. Data-type selection is explained in more detail in section 3.3.3.



Settings

```
BrakePedalECUComp_Type
  Software Components:
    Act_Adder
    Act_Calculator
      RTE API:
        Rte_Write_Act_Calculator_PP_term_a_a
          Parameters:(SInt32 data)
            SInt32 data
              Monitors:
              Triggers:
              Faults:
              Ballista:
                [✓] Run a ballista campaign
          Return Parameter: Std_ReturnType
          Triggers:
          Faults:
        Rte_Write_Act_Calculator_PP_term_b_b
      Runnables:
    Act_ServerAdder
    BrakeLightControl_Type
    Swc_BrakePedalInputHandler
    Swc_BrakeTorqCal
    Swc_GlbBrakeCtrl
    Swc_Vehicle
```

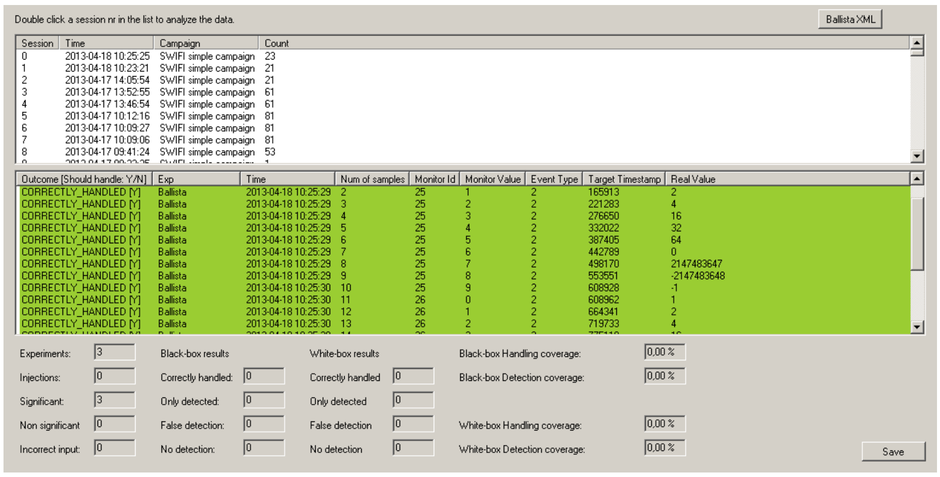
Selection Overview

Monitors:	0/25
Triggers:	0/25
Faults:	0/25

Figure 9: The configurator view showing the selection of performing a ballista test campaign on the port named Rte_Write_Act_Calculator_PP_term_a_a.

The second step concern execution and analysis of the test. The selection of ports and data-type selection file is done using the part of the tool named the configuration. Depending on the selections by the user the configurator generates the code necessary for the execution of the test. The code includes wrappers for the SW-Cs and RTE header files substituting function calls with wrapped function calls. The wrapper is explained in more detail in section 4.6. The configurator also generates a XML file describing the selection made by the user and information needed in the analysis step after the test is done. The code generated by the configurator is manually compiled together with the rest of the AUTOSAR system and flashed onto the target ECU. Figure 9 presents the configurator window.

The test is controlled by a second part of the tool named the campaign runner. The campaign runner can start and stop the execution of the test. It also monitors the test by storing values sent back from the SW-Cs under test in a database. Before the analyzing step the XML file generated by the configurator is needed for mapping of the data stored in the database with the corresponding test value data. The mapping function is explained further in section 4.8. Figure 10 shows the result after a successful robustness test.



Double click a session nr in the list to analyze the data.

Ballista XML

Session	Time	Campaign	Count
0	2013-04-18 10:25:25	SWIFI simple campaign	23
1	2013-04-18 10:23:21	SWIFI simple campaign	21
2	2013-04-17 14:05:54	SWIFI simple campaign	21
3	2013-04-17 13:52:55	SWIFI simple campaign	61
4	2013-04-17 13:46:54	SWIFI simple campaign	61
5	2013-04-17 10:12:16	SWIFI simple campaign	81
6	2013-04-17 10:09:27	SWIFI simple campaign	81
7	2013-04-17 10:09:06	SWIFI simple campaign	81
8	2013-04-17 09:41:24	SWIFI simple campaign	53

Outcome [Should handle: Y/N]	Exp	Time	Num of samples	Monitor Id	Monitor Value	Event Type	Target Timestamp	Real Value
CORRECTLY_HANDLED [Y]	Ballista	2013-04-18 10:25:29	2	25	1	2	165913	2
CORRECTLY_HANDLED [Y]	Ballista	2013-04-18 10:25:29	3	25	2	2	221283	4
CORRECTLY_HANDLED [Y]	Ballista	2013-04-18 10:25:29	4	25	3	2	276650	16
CORRECTLY_HANDLED [Y]	Ballista	2013-04-18 10:25:29	5	25	4	2	332022	32
CORRECTLY_HANDLED [Y]	Ballista	2013-04-18 10:25:29	6	25	5	2	387405	64
CORRECTLY_HANDLED [Y]	Ballista	2013-04-18 10:25:29	7	25	6	2	442789	0
CORRECTLY_HANDLED [Y]	Ballista	2013-04-18 10:25:29	8	25	7	2	498170	2147483647
CORRECTLY_HANDLED [Y]	Ballista	2013-04-18 10:25:29	9	25	8	2	553551	-2147483648
CORRECTLY_HANDLED [Y]	Ballista	2013-04-18 10:25:30	10	25	9	2	608928	-1
CORRECTLY_HANDLED [Y]	Ballista	2013-04-18 10:25:30	11	26	0	2	608962	1
CORRECTLY_HANDLED [Y]	Ballista	2013-04-18 10:25:30	12	26	1	2	664341	2
CORRECTLY_HANDLED [Y]	Ballista	2013-04-18 10:25:30	13	26	2	2	719733	4

Experiments:	3	Black-box results		White-box results		Black-box Handling coverage:	0,00 %
Injections:	0	Correctly handled:	0	Correctly handled	0	Black-box Detection coverage:	0,00 %
Significant:	3	Only detected:	0	Only detected	0		
Non significant	0	False detection:	0	False detection	0	White-box Handling coverage:	0,00 %
Incorrect input:	0	No detection:	0	No detection	0	White-box Detection coverage:	0,00 %

FIGURE 10: The view of the campaign runner after a successful test campaign

4.6 The Wrapper

Wrappers have appeared in fault injection tools before [24, 25, 26] as wrappers are an effective way of collecting and modifying data on the edge, i.e., input and output ports, of a software component or software layer. The AUTOSAR specification enforces a precise naming scheme for RTE function calls, which are also the standard mean of communication between SW-Cs. As the RTE header files are accessible these functions can be replaced using C programming language macros. The modified code includes a wrapper which can intercept and modify both function parameters and the function's return value. It is noticeable that this technique doesn't require access to the source code of the SW-Cs. The SW-Cs are always delivered with an XML-file which specifies its ports and the ports are accessed through to function calls in the RTE. As the naming scheme is set by the AUTOSAR specification, the macros can, by using the xml file, easily be automatically generated to replace RTE function calls with wrapped versions. The wrapper code is attached only to ports selected by the user during the configuration to allow a small degree of intrusiveness.

4.7 The Test Algorithm

The algorithm we implemented in the tool creates a deterministic campaign depending on the number of arguments and the types of the arguments. The tool will only inject one faulty value on one port each round. The algorithm of the tool is given in the following pseudo code.

TABLE 2: Sequence of test values injected on three ports of the same type

Seq. No.	Port 1	Port 2	Port 3
1	A	A	A
2	B	A	A
3	C	A	A
4	A	B	A
5	A	C	A
6	A	A	B
7	A	A	C

```
//init:
for each port
    set valid input on port
execute test

//main:
for each port
    for each test value of port type
        set test value on port
        execute test
    set valid input on port
```

Table 2 shows an example using three ports of the same type. The type has three different test values which are defined as A, B, and C. A is a considered a valid input, and B and C are invalid inputs.

4.8 Mapping of Monitor Values

During the execution of a test campaign the values injected into the MuT are also sent to the PC. This enables monitoring and evaluation of the campaign. The CAN bus has package size limitations and a problem may occur if a test value is larger than the package size. This problem is avoided by not sending the real injected value back to the PC, but simply the sequence number of the test. The campaign runner can map the sequence number to the actual test values. This mapping can be done because the test algorithm and the test values for each types are known from the XML files created by the code generator.

5 Evaluation

Evaluation of our tool was carried out using a small experimental system. The system was provided by the researchers at Volvo GTT. This chapter present our results from evaluating the tool.

5.1 Measurements

The subsections below provide measurements of intrusiveness and performance of the tool.

5.1.1 Campaign Time Duration

A robustness testing tool for commercial off-the-shelf (COTS) should be fast. Speed is important because a main attraction for companies opting for COTS is to save development time. The time to set up and run a full test campaign with the tool varies with each system and a simple campaign on a small system can take as little as a few minutes. Large campaigns could possible take several days. The actual execution time of a test campaign depends on several parameters, the number of test values for each data type, the number of ports to be tested, how frequent the ports are called, and in what order. In a simple system with a single SW-C reading two ports, A and B, every Tms, the campaign duration time is Tms multiplied with the number of test values of the type of port A added with Tms multiplied with the number of tests values of the type of port B. Given that no timing constraints (see section 5.1.3) are violated during the execution of a campaign the campaign duration can be estimated with the following formula:

$$\text{Campaign time duration} = \sum_{i=1}^{n} P_n * T_n$$

Where n is the number of ports, P_n is number of tests for port n, and T_n the execution frequency of port n.

5.1.2 Memory Footprint

The memory footprint of adding the testing controller to the system is relatively small (∼9KB). Every port included in the test campaign uses approximately 1KB. For the smallest ECUs this could potentially be a problem, but running the test on a ECU with more memory of the same architecture would avoid memory issues and still expose robustness weaknesses.

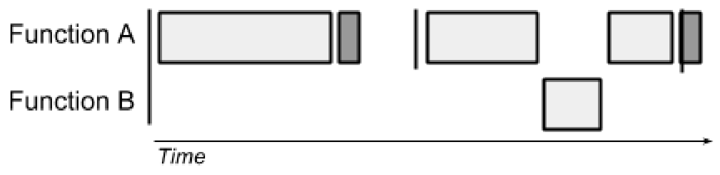


FIGURE 11: An execution diagram showing how the additional overhead (dark color) extend function A's completion time past its own period. In this example the period of function A is equal to its maximum allowed completion time.

5.1.3 Timing Constraints

When testing real-time systems extra care must be taken to ensure that the test code does not force the system to violate its timing constraints [27]. AUTOSAR SW-Cs consists of functions that are called periodically by the RTE. It is often essential for functions to complete before the start of the next cycle. Additionally, the functions may have a specific individual time constraint for their completion. Functions may be interrupted by other functions, which will prolong the completion of the previous function. It is up to the designer of the AUTOSAR system and its scheduler to make sure no constraints are violated. Because ports are realized as functions the testing code is wrapped around every function under test and will extended execution time of the function. Figure 11 shows an example where the timing constraints are violated by the additional code.

The wrapper code is constructed to minimize intrusiveness when it comes to execution time. The testing code for injection of a value consists of 235 machine instructions and affects one port at a time. Ports that are not currently under test return after 12 machine instructions as they do not need to retrieve a new value. For comparison reading a value without the wrapper is approximately 300 machine instructions long. It may be noted that the code for reading a value may be optimized by the RTE generator as described in section 3.1.7 and therefore faster than 300 machine instructions. Regardless, the additional code is still considered to only marginally affect the execution time of a function.

5.2 Experimental Experiences

Testing of the tool has been carried out on a physical implementation of the brake-by-wire system described in section 4.3. Experiments show that lack of redundant mechanism leaves the system vulnerable for numerous faults. By running a campaign on a single wheel, it could be shown that it was easy to make the single wheel break. A single wheel breaking would force the vehicle to rotate and potentially end up on the roadside or crash into another vehicle. While the brake-by-wire system is more complex than the calculator and adder (see section 4.3) it is still logically trivial. No logical fallacy was discovered in the system. Intentionally vulnerable code was introduced for demonstration of the possible exposure of logical mistakes. The code did not have proper boundary values, and a large value would leave the system in a hanged state. The tool did successfully discover this weakness and we believe that as systems becomes more complex, the likelihood of vulnerabilities such as the one artificially introduced is increased.

5.3 Using the Tool for Fuzzing

An important aspect to any fault injection tool is the choice of fault model. Common fault models used by fault injectors are explained in section 3.2.2. An alternative to the data-type based model is fuzzing. Fuzzing rely on a random generation of test values. The fault space of the data-type based fault model can be seen as a subset of the fault space of the fuzz fault model, as the randomly generated values may result in the same values as in the data-type based fault model. The relationship between the two fault models encouraged an exploration of changing the fault model used by the tool. By randomly generating the values in the type value file the tool could be transformed into a fuzzer for AUTOSAR SW-Cs. The approach of generating fuzz data before the execution of the test have the advantage that it is easier to trace back a faulty behavior to a specific faulty value. The discovery led to the addition of a module to the tool for the generation of a fuzz-style value selection file.

6 Discussion

Presented in this thesis is a tool for robustness testing of SW-Cs in AUTOSAR. The tool uses a method of data type based fault injection on the interfaces of the SW-Cs under test. To perform the injection a wrapper is used to substitute the current values on the ports of the interface to values from a library. A complete campaign includes the injection of all test values for each data type on all corresponding ports.

A significant advantage of using a testing method based on the data type specification is that no further assumptions on design are taken. A more common alternative approach is to construct a test derived from the functional specification of a component. By using the data type based method we avoid relying on design that may be incorrect in the first place. We believe that extra care should always be taken when there is risk of introduction of human error, such as in the design phase.

Using a tool like the one presented in this thesis can be used to expose vulnerabilities within a system and thereby facilitate the development process towards more robust systems. Components in AUTOSAR are often constructed of smaller sub-components. By using the tool it is possible to inject faults directly into the sub-components even when they are part of a larger component. This can potentially be used to increase the understanding of how errors propagate inside a component.

By using the tool it is apparent that a system often consists of many single points of failure. During the evaluation of the tool we had the opportunity to test it on a more complex physical brake-by-wire prototype. We saw that we could easily make a single wheel break, imposing great risk on the driver. Discovering these vulnerabilities may help the designer to understand where to better locate countermeasures. One such countermeasure may be the introduction of voters. Voters have been used extensively in the space industry and are a common area of research. As vehicles are getting more and more complex there may be a need of using voters also in the automotive industry.

During the evaluation of the tool simple data types were tested. As the AUTOSAR standard expands, more types may be introduced. Pointers and strings are possible data types which are traditionally known to introduce 'bugs' in PC software. With the introduction of more complex data types we see an even larger potential for data type based testing. AUTOSAR is a standard followed by several vendors resulting in many different implementations. However, the standard is not complete and every implementation will have its specific features. The differences between implementations can at times be subtle and it may be hard for a designer to know all peculiarities of a specific implementation. This is especially true as data types gets more complex. In this environment we see a potential for the data-type based method of robustness testing explored in this report. As the tool can be run successfully without prior knowledge on the design of the component under test, the implementation specifics are not needed to be considered by the user of the tool.

Because of the independence to the functional specification there is small need of configuration before running a test. This allows the data-type based method to be easy to automate. Configuring a test on a previously untested SW-C only takes a few clicks when using a tool like the one we present in this report. Automation is of great importance as it leaves few openings for mistakes by the user. Furthermore, development time and costs are important factors when opting for third-party components, and it is valuable that the testing of such components does not take significant time.

The tool wraps SW-C interfaces and every call to a port of the interface is manipulated by the wrapper. However, this is not the only way to implement fault injection on AUTOSAR systems. In [28] Lanigan et al. provides their experiences with a technique provided by AUTOSAR called hooks in combination with the software development tool CANoe. They conclude hooks to be a feasible technique but also promote a in-depth study of the most appropriate method for fault-injection in AUTOSAR. We believe such a study to be highly valuable.

7 Conclusion

Robustness testing of AUTOSAR systems and its components is required for compliance with the evolving standards within functional safety such as ISO 26262. In this thesis we provide a tool which can perform robustness testing based on the interfaces delivered with AUTOSAR software components (SW-Cs). The tool examines the SW-C interfaces and constructs a testing campaign based on the data types of the ports delivered by each interface. The data-type based method is shown to be suitable for AUTOSAR SW-Cs, and the testing of components can be made highly automatic. An advantage of using a data-type based method is that testing can be performed without consideration to logical design, only the data types of the ports are considered. The injection of faults is made possible using a wrapper which surrounds the SW-Cs under test. This allows testing of components delivered as both white-boxes and black-boxes. The compatibility to black-boxes is an important step towards a higher use of proprietary and third-party components, a strong need recognized by the industry. In the construction of a robust system all components should be robust, thus robustness tested. Furthermore, components are often delivered in a group of sub-components. By using the wrapping technique it is possible to target single components inside a chain of components. This allows interesting test cases as the outcome from faults inside a chain of components can be monitored. Data-type based robustness testing is a highly promising technique in the AUTOSAR environment and further research towards use on actual products is encouraged.

8 Future Work

The AUTOSAR environment consists of several types of components and robustness testing of a complete system may be performed on all the components. An example of a type of component of particular interest is the BSW as it is used as a layer between hardware to SW-Cs. The wrapper technique used in the tool presented in this thesis is suitable for SW-Cs but may not be suitable for other types of components. An architectural analysis of the different components is needed to find out what technique can be used.

An important aspect of data-type based testing is the test value selection. A thorough analysis of the data types of AUTOSAR and suitable test values would greatly enhance the effectiveness of a data-type based robustness testing tool.

The aim of a robustness testing suite is to detect weaknesses in the system under test. A case study of performing data-type based testing on actual products is recommended. Such a study is useful for evaluation of using the data-type based method in the industry.

Propagation of errors is also an inviting area of future work. Knowing the path of errors in a system may for instance be used to help in the decision to where to locate error detection mechanisms [29]. The tool as implemented today is restricted to testing one ECU at a time. This is strictly an implementation limitation. With little work the tool could be extended to test several ECUs at the same time. Such a tool could be used to study several ECUs during one campaign and gather insights on how ECU robustness issues affect other ECUs in the same system.

The tool presented in this master's thesis is in large parts automated. However, some manual steps are still needed. A fully automatic suite may prove to be highly useful for the industry. An attempt for a fully vendor independent tool is regarded to also be valuable. An analysis of different implementations of AUTOSAR is proposed as a step towards a vendor independent robustness testing tool for AUTOSAR systems.

References

[1] A. Biagosch, S. Knupfer, P. Radtke, U. Näher, and A. E. Zielke, "Automotive electronics—managing innovations on the road," *McKinsey Brochure*, 2005.

[2] ISO, *Road vehicles - Functional Safety - 26262-6.* ISO, 2011.

[3] AUTOSAR, *AUTOSAR Technical Overview v2.2.2.* 2012.

[4] P. Koopman, K. DeVale, and J. DeVale, "Interface robustness testing: Experiences and lessons learned from the ballista project," *Dependability Benchmarking for Computer Systems*, p. 201, 2008.

[5] Y. Yu and B. W. Johnson, "A perspective on the state of research on fault injection techniques," tech. rep., Research Report, 2001.

[6] J. M. Voas and G. McGraw, *Software fault injection: inoculating programs against errors.* John Wiley & Sons, Inc., 1997.

[7] H. Mills, *On the Statistical Validation of Computer Programs.* IBM, 1972.

[8] P. Koopman, J. Sung, C. Dingman, D. Siewiorek, and T. Marz, "Comparing operating systems using robustness benchmarks," in *Reliable Distributed Systems, 1997. Proceedings., The Sixteenth Symposium on*, pp. 72–79, IEEE, 1997.

[9] A. Shahrokni and R. Feldt, "A systematic review of software robustness," *Information and Software Technology*, 2012.

[10] D. Cotroneo, A. Lanzaro, R. Natella, and R. Barbosa, "Experimental analysis of binary-level software fault injection in complex software," in *Dependable Computing Conference (EDCC), 2012 Ninth European*, pp. 162–172, IEEE, 2012.

[11] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of unix utilities," *Communications of the ACM*, vol. 33, no. 12, pp. 32–44, 1990.

[12] J. M. Voas, "Certifying off-the-shelf software components," *Computer*, vol. 31, no. 6, pp. 53–59, 1998.

[13] C. Lu, J.-C. Fabre, and M.-O. Killijian, "Robustness of modular multi-layered software in the automotive domain: a wrapping-based approach," in *Emerging Technologies & Factory Automation, 2009. ETFA 2009. IEEE Conference on*, pp. 1–8, IEEE, 2009.

[14] T. Piper, S. Winter, P. Manns, and N. Suri, "Instrumenting autosar for dependability assessment: A guidance framework," in *Dependable Systems and Networks (DSN), 2012 42nd Annual IEEE/IFIP International Conference on*, pp. 1–12, IEEE, 2012.

[15] J. Haraldsson and S. Thorvaldsson, *Software implemented fault injection for AUTOSAR based systems.* Chalmers University of Technology, 2012.

[16] "Besafe - benchmarking of functional safety," 2011. Available online at `http://www.chalmers.se/safer/EN/projects/pre-crash-safety/associated-projects/besafe-benchmarking` Retrieved 2013-06-08.

[17] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of management information systems*, vol. 24, no. 3, pp. 45–77, 2007.

[18] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *Dependable and Secure Computing, IEEE Transactions on*, vol. 1, no. 1, pp. 11–33, 2004.

[19] IEEE, *IEEE Standard Glossary of Software Engineering Technology Terminology. IEEE Std 610.12-1900.* IEEE Computer Society, 1990.

[20] M.-C. Hsueh, T. K. Tsai, and R. K. Iyer, "Fault injection techniques and tools," *Computer*, vol. 30, no. 4, pp. 75–82, 1997.

[21] A. Johansson, N. Suri, and B. Murphy, "On the selection of error model (s) for os robustness evaluation," in *Dependable Systems and Networks, 2007. DSN'07. 37th Annual IEEE/IFIP International Conference on*, pp. 502–511, IEEE, 2007.

[22] N. P. Kropp, P. J. Koopman, and D. P. Siewiorek, "Automated robustness testing of off-the-shelf software components," in *Fault-Tolerant Computing, 1998. Digest of Papers. Twenty-Eighth Annual International Symposium on*, pp. 230–239, IEEE, 1998.

[23] D. Adams, *The Hitchhiker's Guide to the Galaxy.* San Val, 1995.

[24] A. Baldini, A. Benso, S. Chiusano, and P. Prinetto, "Bond: An interposition agents based fault injector for windows nt," in *Defect and Fault Tolerance in VLSI Systems, 2000. Proceedings. IEEE International Symposium on*, pp. 387–395, IEEE, 2000.

[25] S. Han, K. G. Shin, and H. A. Rosenberg, "Doctor: An integrated software fault injection environment for distributed real-time systems," in *Computer Performance and Dependability Symposium, 1995. Proceedings., International*, pp. 204–213, IEEE, 1995.

[26] J. Arlat, J.-C. Fabre, M. Rodríguez, and F. Salles, "Mafalda: a series of prototype tools for the assessment of real time cots microkernel-based systems," in *Fault injection techniques and tools for embedded systems reliability evaluation*, pp. 141–156, Springer, 2004.

[27] R. Hexel, "Fits: a fault injection architecture for time-triggered systems," in *Proceedings of the 26th Australasian computer science conference-Volume 16*, pp. 333–338, Australian Computer Society, Inc., 2003.

[28] P. E. Lanigan, P. Narasimhan, and T. E. Fuhrman, "Experiences with a canoe-based fault injection framework for autosar," in *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, pp. 569–574, IEEE, 2010.

[29] M. Hiller, A. Jhumkas, and N. Suri, "Dependable computing systems," pp. 407–428, 2005.