CHALMERS
UNIVERSITY OF TECHNOLOGY

An Exploration for Improving Robustness of AUTOSAR Software Components with Design by Contract

Master's thesis in Software Engineering

YULAI ZHOU

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis 2016:NN

An Exploration for Improving Robustness of AUTOSAR Software Components with Design by Contract

YULAI ZHOU

Department of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden 2016

An Exploration for Improving Robustness of AUTOSAR Software Components with Design by Contract

YULAI ZHOU

Supervisor: Johan Haraldsson, Volvo Group Trucks Technology
Mafijul Islam, Volvo Group Trucks Technology
Patrizio Pelliccione, Chalmers University of Technology

Examiner: Miroslaw Staron

Master's Thesis 2016:NN
Department of Computer Science and Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Department of Computer Science and Engineering
Gothenburg, Sweden 2016

Abstract

The increasing volume of software in vehicles makes robustness a significant quality attribute for vehicle software. In order for high quality and high development efficiency of the vehicle embedded software, Automotive Open System Architecture (AUTOSAR) was put forward by several large manufacturers and suppliers in the automotive industry around the world. In this thesis, Design by Contract is applied to improve robustness of existing AUTOSAR software components. The main idea of Design by Contract is to view the relationship between two components as a formal contract which expresses each component's right and obligations. The specific way is to separate input, output and invariant checks from the main processing component and build additional components for them. Functions for checking pre-conditions, post-conditions and invariants are defined in these components respectively. Each function is invoked every time the corresponding check is needed. The proposed solution is validated by conducting testings for the original and modified components in the unit testing tool ARUnit and comparing the results. The results prove Design by Contract greatly increase the robustness of AUTOSAR software components. None of the testings for the modified software components failed. Certainly, this method has weaknesses such as possible errors brought by the newly-built components. And also, it is hard to modify the components of which the code is automatically generated from some model tools.

Keywords: AUTOSAR, Design by Contract, robustness, ARUnit

Acknowledgements

I would like to express my sincere appreciation to my supervisors at Volvo Group Trucks Technology, Mafijul Islam and Johan Haraldsson for their great help and guidance throughout the thesis work. I wish to thank my supervisor at the university Patrizio Pelliccione for his encouragement and academic support on the ways of doing the thesis work. I would also like to thank my examiner Miroslaw Staron for his help in the mid-time seminar, final presentation and the thesis report. Finally, I also would like to express my gratitude to Daniel Blomqvist and other staff at the company for their help with the technical difficulties during the thesis work.

Yulai Zhou, Gothenburg, 2016

Contents

1 Introduction

Software volume in vehicles has been keeping increasing for years. The volume is expected to increase by 50% by 2o2o[1]. In line with the trend of increasing software volume in vehicles, more and more requirements for the robustness of the software are elicited. According to some reports, software errors led to almost 60-70% of all the recalls of vehicles in Europe and North America [1]. It endangers people's lives, affects manufacturers' reputation and leads to enormous economic losses .

The robustness of the vehicle embedded software is just part of the quality requirements for the vehicle embedded system. In order for high quality and high development efficiency of the vehicle embedded system, many large manufacturers and suppliers in the automotive industry in Europe have been joined up to establish a shared standard for vehicle system architecture since 2003. Then, Automotive Open System Architecture (AUTOSAR) was put forward. Its goal is to get a de-facto open industry standard for automotive E/E architectures [12], by which automotive systems can get better modularity, scalability, transferability and re-usability. Since then, AUTOSAR has been a popular open standard in the automotive industry because of its great value and development potential. More and more manufacturers and suppliers join and become partners of this project[2].

However, challenges also come with the popularity of AUTOSAR. As AUTOSAR just defines the architecture of the vehicle software system, the implementation of the functionality is done by the manufacturers and suppliers themselves, of which the quality is hard to ensure. This brings great challenges on the robustness of the AUTOSAR software they developed. AUTOSAR is mature, but such AUTOSAR software still needs to be improved. Researchers and developers are always willing to develop AUTOSAR software components with good robustness, they have been trying many different design ideas for development.

In these design ideas, Design by Contract may be a good design idea for AUTOSAR software components. In Design by Contract, the relationship between a class and its clients is viewed as a formal agreement in which each party's right and obligations are described[3]. In practice, it sets precise conditions for the input and output of the components. As the definition of robustness is "the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions" [4]. The output of one component is often the input of another component. Design by Contract helps check the input and output of one software component. Also, it checks the invariants inside the software component.

To achieve a high degree of robustness, Design by Contract is worth a try. The aim of this thesis is to answer the research question:

- **Is the use of Design by Contract a methodology to increase the robustness of AUTOSAR software components?**

The work in this thesis addresses this question. I apply Design by Contract to the components of two AUTOSAR applications, a Brake-Lighting application and a Brake-By-Wire system. With the design idea of Design by Contract, conditions for input and output are set for the Brake-Pedal-Input-Handler component in the Brake-By-Wire system and the Brake-Light-Control component in the Brake-Lighting application. As the Brake-Pedal-Input-Handler component is also used in the Brake-Lighting application, the connection of these two components is also considered and implemented. These components which are modified with input and output checking, are tested by black-box testing with ARUnit testing tool. By comparing the test results of the original and modified components, the answer to the research question can be gotten.

1.1 Related Work

Design by Contract(DbC) was firstly described by Bertrand Meyer [5] in his several articles starting from 1986, which is introduced together with his Eiffel programming language. Later in 1992, in the article Applying Design by Contract [6], Bertrand Meyer introduced the application of Design by Contract. In this article, he emphasized the significance of software reliability which includes robustness and showed how to reduce bugs by building software components on the basis of carefully designed contracts. He defined the contract as the obligations and the benefits for the client and the supplier. Assertions, which include pre-conditions, post-conditions and invariants, were described by him to express contracts for software.

Liu et al.[7] presented how to specify the functionality of software components with the theory and methods of the Design by Contract approach in their paper in 2002. By their way of understanding Design by Contract, they concluded that if the operations are encapsulated within the components and the communications are made through the interfaces, it will make the components more reliable and reusable. Cheon et al.[8] introduced a method in 2005, to model program variables to write and check DbC assertions without referring to the program states which makes the assertions more readable and maintainable.

Benveniste et al.[9] wrote one paper in 2011 about contract-based design which is similar to Design by Contract and its uses to address the challenges faced in designing large-scale complex embedded systems. Concepts and the key steps of contract-based design were introduced in this paper by giving three real examples. Thüm et al.[10] introduced some approaches of integrating the Design by Contract approach with feature-oriented programming by defining contracts of methods and their refinements to increase the reliability. Some case studies were also performed by them to gain and then share the insights.

The work of this thesis is based on the ideas and implementations from these related articles and work. As Design by Contract is used for objected-oriented languages in most situations and AUTOSAR SW-Cs are developed by C without any existing third-party tools that support Design by Contract, the author of this thesis needs to explore a new way for applying Design by Contract in AUTOSAR SW-Cs and evaluates the effect of improvement of robustness.

1.2 Research Methodology

After studying several research methodologies and considering the context of this thesis, the research methodology described in [11], which is used for design research, is selected. The process of the research methodology for this thesis is described in Figure 1.1. The stages of the research methodology include problem identification, discussion & suggestion, design & development, evaluation, and conclusion & report.

In design research, the solution for solving one particular problem is investigated during the process of design and implementation [11]. In the context of this thesis, the problem is to verify if the use of Design by Contract is a methodology to increase the robustness of AUTOSAR software components. This problem also concerns how to implement software to apply Design by Contract to the components. In order to address the problem, different programming methods are proposed, discussed and implemented. After that, the AUTOSAR software components, which are enhanced with the programming method for Design by Contract, are tested with the unit test tool ARUnit. The robustness is evaluated from the results of the test. From the results, we can know whether Design by Contract helps improve robustness of AUTOSAR software components. The process and conclusion are documented and presented.

Besides some preparations for starting the thesis, the thesis work is done with 3 iterations. During the first iteration, assert() was discussed and used to apply the Design by Contract approach to the original AUTOSAR SW-Cs. In the second iteration, I tried to set independent pre-condition components for every type of inputs. These two attempts were abandoned for their weaknesses described in Chapter 3. In the third iteration, the method presented in Chapter 4 was used and implemented. Besides the iterations, I have weekly meeting with the supervisors at Chalmers and Volvo for delivering what has done, getting feedback and planning for next work.

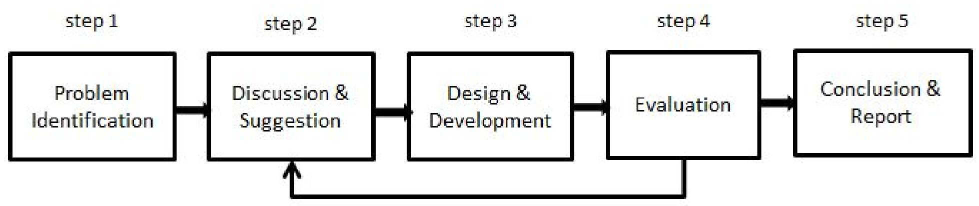


Figure 1.1: Research Methodology

1.2.1 Problem Identification

In this stage, the primary strategy is to collect information about AUTOSAR system and Design by Contract through deep literature studying. On the one hand, I learned knowledge about Design by Contract from the literature that the supervisor at Chalmers provided. On the other hand, I studied about AUTOSAR and robustness evaluation method for AUTOSAR software components from the literature that supervisors at Volvo provided. The possible programming methods to apply Design by Contract are searched by myself on the Internet.

Another part of significant work is to select the AUTOSAR software components that I work on. After reading the documented specification of DEDICATE framework project and discussing with supervisors, components in two AUTOSAR applications, a Brake-By-Wire system and a Brake-Lighting application, are selected.

1.2.2 Discussion and Suggestion

The main strategy that I used in this stage is weekly meeting with the supervisor at Chalmers and Volvo. During the weekly meeting, the supervisors at Volvo presented the AUTOSAR system and relevant information about it, and the supervisor at Chalmers presented the design idea of Design by Contract. Some suggestions were also proposed by them about how to apply Design by Contract.

Based on the literature of applying Design by Contract to embedded system which uses C programming language, one possible programming method was discussed with the supervisors. It was to use assert() in the codes for contracts checking. For several weaknesses of this method, it was abandoned. Later, the method of building independent components for every input was tried. It was abandoned for some weaknesses as well. After that, redesign of the components according to some reference patterns was discussed and considered. We decided to try it in the third iteration and evaluate the results.

1.2.3 Design and Development

In this stage, programming work is done to apply Design by Contract to AUTOSAR software components. It contains 4 steps: raising issues of the original components, condition identification, design and programming and testing. In the first step, issues of the original components are raised to see what needs to be improved in the new design. In the second step, pre-conditions, post-conditions and invariants are identified for the selected AUTOSAR software components. In the third step, by designing the new components and programming, these conditions become code in the components. In the final step, testings are conducted to get results.

1.2.4 Evaluation

In the evaluation stage, black-box testing is performed with ARUnit test tool. The code of the software components is also in ARUnit which is based on Eclipse. The steps of robustness evaluation are:

- In ARUnit, perform robustness testing of the original version of the software component.
- Input a list of valid and invalid data, and calculate what percent of the output data is the result of successful running and also in the expected range. Keep this percentage as D1.
- In Eclipse, replace the original version of the software component with the software component which has been enhanced with Design by Contract.
- Input a list of valid and invalid data, and calculate what percent of the output data is the result of successful running and also in the expected range. Keep this percentage as D2.
- Compare D1 and D2. If D2 is greater than D1, it means the software component which is enhanced with Design by Contract has better robustness.

At the same time, the evaluation results also help improve programming method for applying Design by Contract.

1.2.5 Conclusion and Report

In this stage, conclusion is gotten from the evaluation results. The availability of Design by Contract for better robustness of AUTOSAR software components is verified. All the process and results are documented and further work is discussed.

2 Concepts of Related Technologies

In this chapter, some introductions to the concepts related to the thesis are described to help the readers better understand the thesis. Also, together with the concepts are some examples.

2.1 AUTOSAR

AUTOSAR (AUTomotive Open System ARchitecture) [12] is a collaborative project initiated by several large manufactures and suppliers in automotive industry to establish a shared standard for automotive E/E architectures. It is driven by the intention for getting better flexibility, scalability, reliability and quality when the complexity of E/E system is greatly increasing. This kind of increased complexity is mainly concerned with the growth of the functional scope. Besides the goal of making the developers concentrate on the realization of the functionality rather than the design of the architectures, the standard of AUTOSAR also makes components developed by different manufacturers or software companies be able to be integrated with well-defined interfaces.

2.1.1 AUTOSAR Abstraction

AUTOSAR is a standard architecture to make vehicle software applications independent of the hardware. Every AUTOSAR application is distributed to one or more Electronic Control Units (ECUs). Communication between different ECUs are conducted with a shared visual bus, which consists of hardware interfaces provided by the basic software in AUTOSAR infrastructure. The Runtime Environment (RTE) is an implementation of the Virtual Functional Bus (VFB). It provides a uniform environment for communication between components [12]. So that when moving a component to another ECU, developers do not need to change any code of the component. The layered architecture of the AUTOSAR software for an ECU is shown in Figure 2.1.

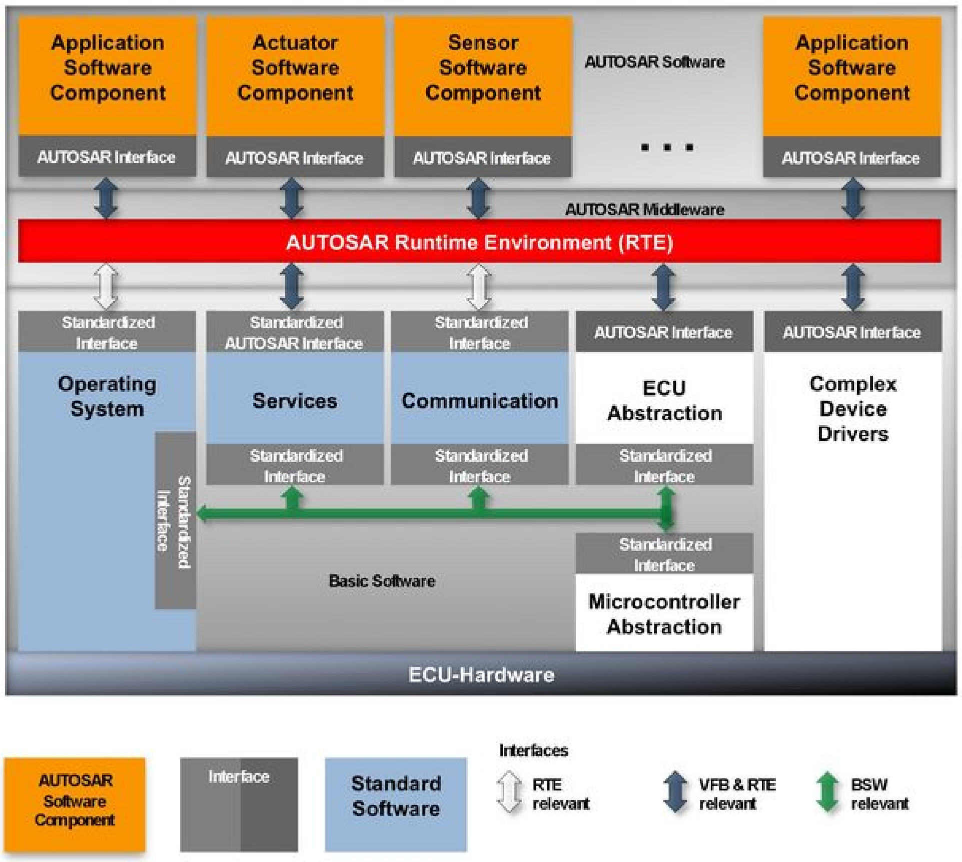


Figure 2.1: The layers of AUTOSAR architecture for an ECU.[12]

The thesis focuses on the application layer which consists of application software components. Examples of components in AUTOSAR applications are given. But for better understanding of how the system runs, short descriptions of related concepts of others layers are also given.

2.1.2 AUTOSAR Software Component

AUTOSAR software component is defined as the encapsulation of part of the functionality of the AUTOSAR application[12]. An AUTOSAR application is composed of one or several SW-Cs. How to describe the interfaces of these AUTOSAR SW-Cs is defined and standardized within AUTOSAR. Each component can only be distributed to one AUTOSAR ECU. This is the reason why the AUTOSAR SW-C is called as "Atomic Software Component" [12]. AUTOSAR does not prescribe the size of the SW-Cs and how the SW-Cs are implemented. But in order to be able to integrate several AUTOSAR SW-Cs correctly, one formal and complete description for one SW-C is needed when it is implemented. The description introduces how to configure the infrastructure for the component when building the system. The introductions to the components that I work on are given below.

2.1.2.1 Brake-By-Wire Application

In order to better understand the functionality of the selected AUTOSAR software components in this thesis, the software application that includes these components should be introduced. Here is the Brake-By-Wire application.

The Brake-By-Wire application is a research framework developed by the DEDICATE project [13] that implements a brake-by-wire function distributed over five ECUs. It is not the real system that is used in the real trucks. It is proposed to give a example of distributed safety-critical system for validating research projects. The BBW application also includes an environment model of the vehicle in order to simulate the behaviour of the entire vehicle with regards to acceleration and braking [13]. When using this application, the Brake Pedal ECU gets the signal of braking, does calculation and then sends a corresponding brake force request to each wheel.

2.1.2.2 Brake-Pedal-Input-Handler Component

The distribution of the Brake-Pedal-Input-Handler component in the Brake-By-Wire system is shown in Figure 2.2. The function of this component is to convert the hardware pedal input into a pedal position (0-100%). The input of this component is an integer with 12 bits and the output is a percentage from 0% to 100%. It provides input for the Brake-Torque-Calculation component and the Brake-Light-Control component.

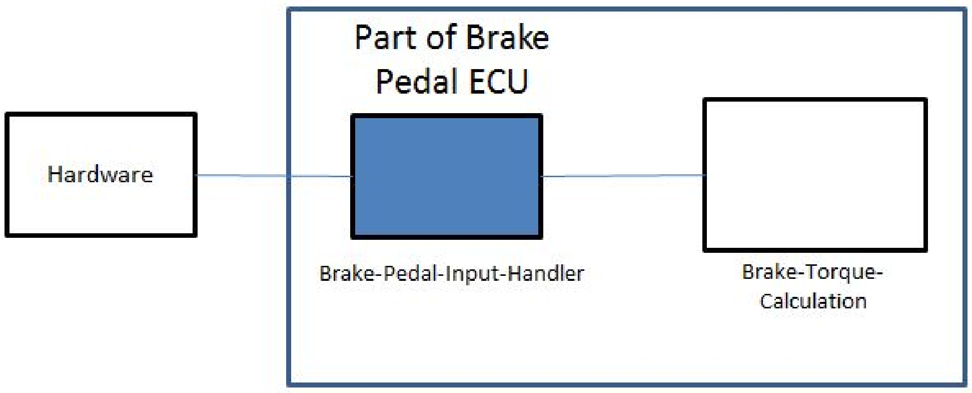


Figure 2.2: Brake-Pedal-Input-Handler component distribution on ECU.

2.1.2.3 Brake-Light-Control Component

The distribution of the SW-Cs in Brake-Lighting application is shown in Figure 2.3. The Brake-Light-Control Component is located on the BrakePedalECU. It inputs vehicle speed and brake pedal position (0-100%) and outputs ON or OFF for the brake lights according to some rules. The basic rules [13] are : (1)The brake light is always OFF when the pedal input is 0%. (2)The brake light is always fixed ON whenever the pedal input > 0% and the vehicle speed is < 10km/h. (3)From 10km/h and above the brake light will blink ON/OFF if emergency braking is active otherwise it is fixed ON.

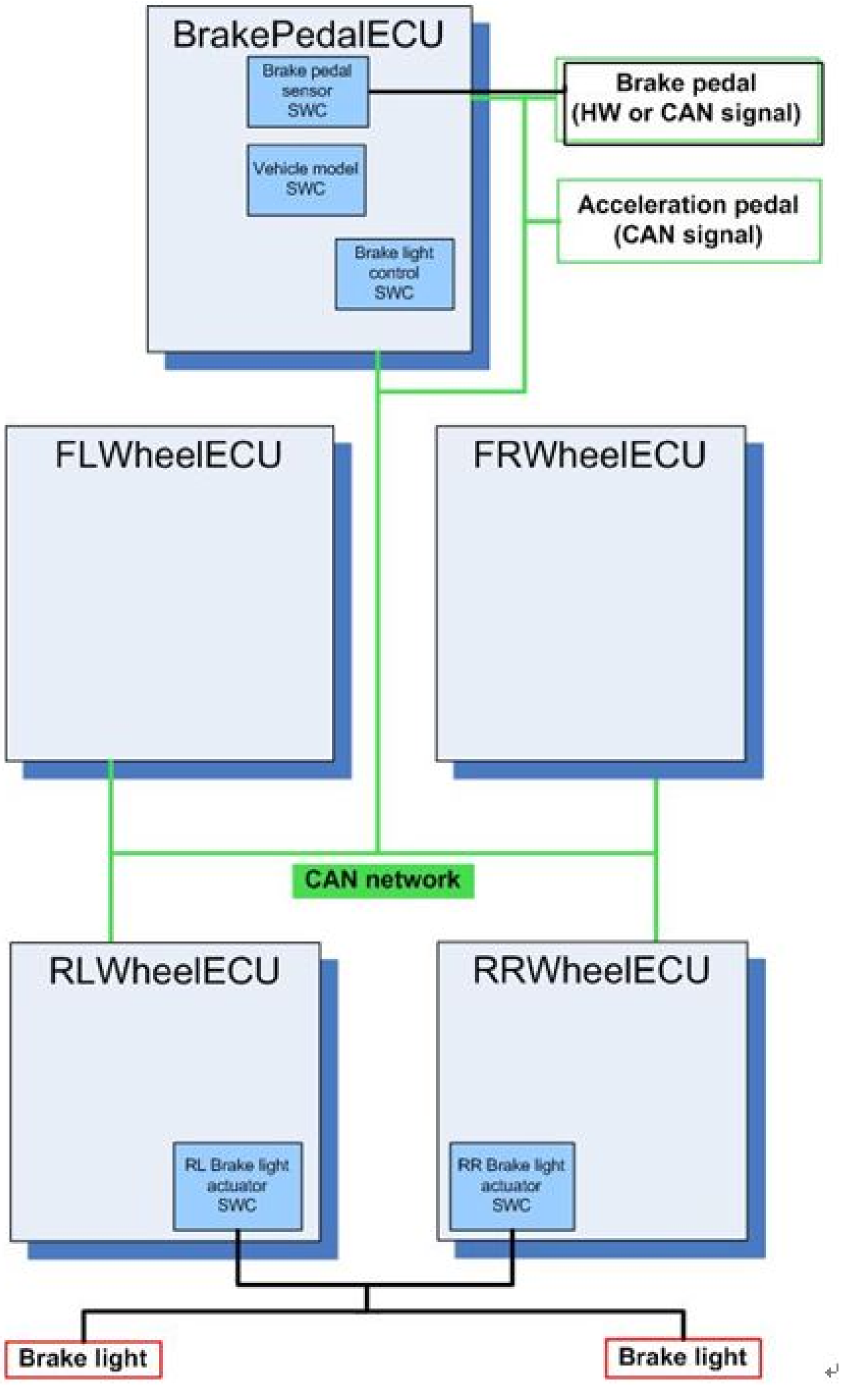


Figure 2.3: Brake lighting SW-Cs distribution on ECUs.[13]

2.1.3 AUTOSAR Software Component Communication

Communication between AUTOSAR software components are conducted by well-defined ports. A port is defined by an AUTOSAR interface. It can either be a

Provider Port which provides data or a Required Port which requires data. There are two main types of communication patterns supported by AUTOSAR. One is Client-Server and another one is Sender-Receiver. In the Client-Server pattern, the client will send a request for service, and the server will perform the requested service after receiving it and then respond to the request. A SW-C can be both a client and a server. The Sender-Receiver pattern realizes asynchronous communication. A sender will send information to one or several receivers without getting a response from the receivers. And also, the time and way to use the received information are decided by the receivers.

2.1.4 Virtual Functional Bus

The virtual functional bus (VFB) is defined as the abstraction of the AUTOSAR SW-Cs interconnections [12] on the vehicle. It enables the components on different ECUs to communicate with each other independent of the hardware and which ECUs they locate.

2.1.5 Runtime Environment

The runtime environment (RTE) is an information exchange center for communication inside one ECU or between different ECUs. It is the implementation of the VFB on a specific ECU [12]. It provides the same interface for the SW-Cs to the AUTOSAR infrastructure and hardware despite where the SW-Cs locate. As different components are used in different applications, the RTE will be tailored in order to use resources more efficiently when the RTE is generated by the RTE generation tool. Sometimes, the RTE is tailored according to the configuration. All these make differences on the generated RTE of different ECUs.

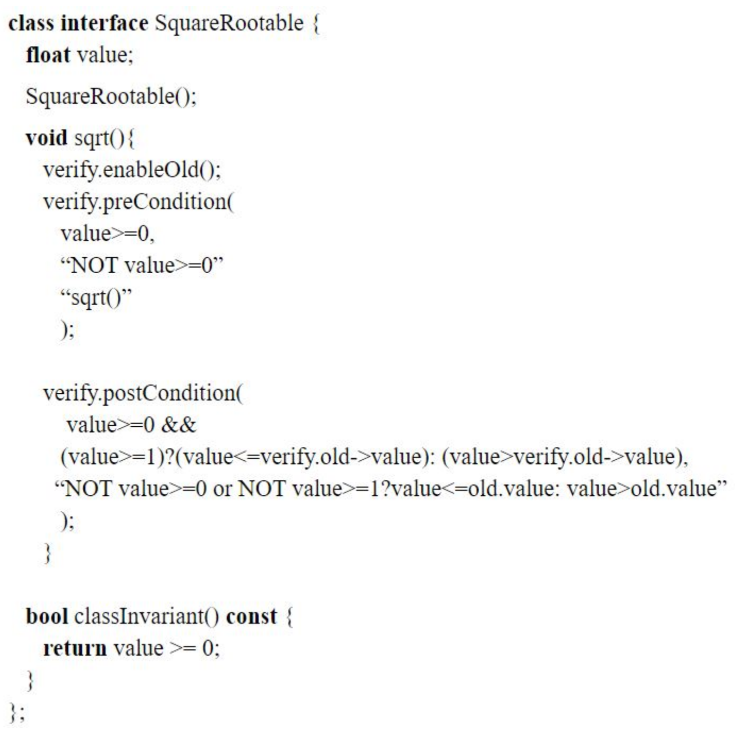
2.1.6 AUTOSAR Infrastructure

AUTOSAR infrastructure is a collection of basic software which mainly consists of services, communication, operating system, ECU abstraction, microcontroller abstraction and device drivers [12]. Its function is to provide platform services to SW-Cs.

2.2 Design By Contract

Design by Contract, also known as programming by contract, is an approach for designing software, by which software can get better robustness. The key concept is “viewing the relationship between a class and its clients as a formal agreement, expressing each party’s right and obligations” [3]. The agreements are similar to the contracts in business. These contracts set conditions for input and output of software components. The conditions have three types, pre-condition, post-condition and invariant. When a client component calls an operation on a server component, the client component needs to meet the pre-condition which is specific for that operation. For the return of that operation, the requirements of the post-condition need to be met, which is an obligation for the server component. Invariant is a certain property that are met for both of the two components. In this way, different components of a software system can collaborate with each other with high robustness.

When Design by Contract was pioneered by Bertrand Meyer in the late 1980's, it was firstly used in his design of the Eiffel programming language [6]. Later, this design philosophy of software starts to be popular in languages with native support or with third-party support. The following code [14] is a simple example of how Design by Contract works in C++. It is a short form of a class which omits some code not relevant to the example.



```
class interface SquareRootable {
  float value;
  SquareRootable();
  void sqrt(){
    verify.enableOld();
    verify.preCondition(
      value>=0,
      “NOT value>=0”
      “sqrt()”
      );

    verify.postCondition(
       value>=0 &&
      (value>=1)?(value<=verify.old->value): (value>verify.old->value),
    “NOT value>=0 or NOT value>=1?value<=old.value: value>old.value”
      );
    }

  bool classInvariant() const {
    return value >= 0;
  }
};
```

This component functions as checking the input and output value of square root calculation. In this component, all the three types of conditions are set for the class. The pre-condition is that the input value should not be less than 0. The post-conditions are that the output value should be greater than 1 when the output value is less than the input value, and the output value should be greater than 0 and less than 1 when the output value is greater than the input value. The invariant for such a class is that the value should not be less than 0. If these criteria are not met, the calculation will not be conducted. This is a simple example for understanding such a design idea.

In C programming language, Design by Contract can be applied by assertions. For the reason that there are no classes in C, the subjects of the conditions are the functions. But the principles are similar. The caller function must meet all the preconditions of the callee function, and the callee function must meet its own post-conditions [15]. The failure of either party of the contract is a bug in the software

[3]. Invariants in C are the conditions that must be hold for a structure or type [15].

When giving examples of setting conditions for software components in AUTOSAR, the conditions should derive from all the possible input (valid and invalid) and requirements specification. Using the Brake-Pedal-Input-Handler component as an example, the valid input is an integer from 0 to 4095 and the invalid input may be less than 0 or greater than 4095. The pre-condition can limit the input value in the valid range. Similarly, as the conversion calculation may make the output value less than 0% or greater than 100%. Obviously, the post-condition should check the output in the range of 0%-100%.

2.3 Robustness

Robustness is defined as "the degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions" in IEEE standard [4]. The definition is similar to what is described in ISO 26262-1. The understanding of it is a bit different for software and hardware. For software, robustness is the ability to respond to abnormal inputs and conditions. For hardware, robustness is the ability to be immune to environmental stress and stable over the service life within design limits [16].

In this thesis, I just work on the software part. If discussing the robustness of software, on the one hand, good robustness means the system or component can handle the data correctly even the input volume is very large. On the other hand, it should be able to handle invalid inputs to ensure the successful running of the system or component. As in the vehicle embedded system, most applications run repeatedly over a time period. The time period can be 5ms, 10ms or 20ms according to the requirements of the application. Considering the definition of Design by Contract, my work in this thesis mainly focuses on improving the capability of handling invalid inputs and ensuring valid outputs. That is how Design by Contract is applied.

The robustness of AUTOSAR software components is evaluated according to the rules described below. ARUnit is used to run black-box testing for the components. A list of data which may be valid or invalid is inputted to the component. For every input, the output can be data in the expected range, data outside the expected range or error. Better robustness means more output data in the expected range, less output data outside the expected range and less errors. The comparison of them shows the differences of different components' robustness.

2.4 ARUnit

ARUnit is a unit testing tool which provides a lightweight testing environment for AUTOSAR software components[17]. It is based on Eclipse. After importing the components that need to be tested, it can compile the components and generate the run-time environment for each single AUTOSAR software component. Of course, test cases can be defined in it as well for the reason that it provides an API to stimulate and query the state of the RTE from the outside[17]. It is a quite convenient tool for operating unit testing effectively and efficiently. Figure 2.4 shows how ARUnit generates RTE for AUTOSAR software components. The way of how the data handled and exchanged inside between component A and component B

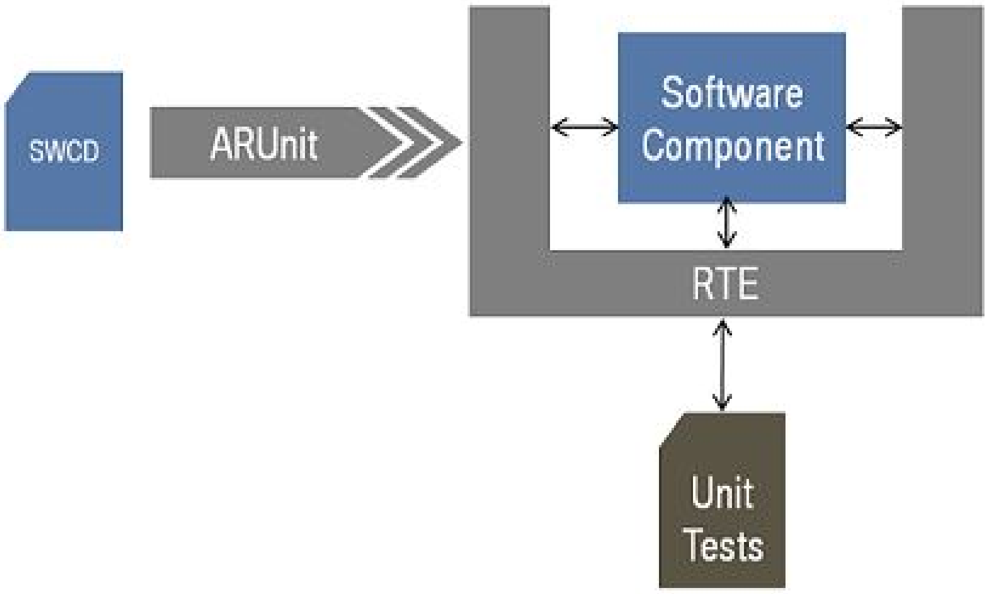


Figure 2.4: ARUnit provides RTE for single Software Component.

3

Designing the Solution: Design by Contract for AUTOSAR Software Components

In this chapter, three attempts of applying Design by Contract to AUTOSAR software components with different methods are described. The first and second attempts are abandoned for the problems and limitations they have. The final attempt which builds new pre-condition, post-condition and invariant components is adopted. The descriptions and figures are used to explain the solution in this chapter.

3.1 First Attempt

3.1.1 Description

As described in [15], C programming language does not provide language features that Design by Contract needs. What C language has is assert(). The first attempt is to directly use assert() to add input and output checks into the code. For example, there are two types of input and one type of output for one component. And the requirements for them are input_1 > 0, input_2 < 0 and output > 0. Then I need to add assert(input_1 > 0 & input_2 <0) at the beginning of the code and assert(output > 0) at the end of the code. And also, if there are C-style structures or types in any places of the code, assertions are needed at these places as well. These assertions are used as the pre-conditions, post-conditions and invariants for this component.

3.1.2 Problems & Limitations

An C-style assertion is not suitable for error handling especially in embedded software. In most situations, there are not any screens available to show the information of the errors. What such software needs is an approach to detect and handle the errors. And also, there are many weaknesses of using assertions. They are lack of robustness, intermixing application code with contracts and code redundancy [15]. As using assert() needs to add extra code into the original component which may also possibly bring errors when running the preconditions, post-conditions and invariants checks. And also, assert statements tend to intermix with application code

[15] which is not good for readability, understandability and reusability of the code. Moreover, duplicate code is needed when invariants for a common structure or type exist in many different places in the code. Thus, assert() in C programming language does not make Design by Contract reach desired effects to improve software components' robustness in AUTOSAR.

3.2 Second Attempt

3.2.1 Description

In the second attempt, I tried to set independent components for every type of input, output and structures in the original AUTOSAR software components. Using the same example in Section 3.1.1, there should be 3 components around the original component. In each of these components, there is a function which is used to check the value. They are function_1(input_1), function_2(input_2) and function_3(output). These functions in different components are invoked when they are needed by the original component.

3.2.2 Problems & Limitations

There are problems for this method. The most important one is that if there are a huge number of types of input, output and structures for one AUTOSAR software component, there will be the same number of components around the original component. It makes it hard to manage so many components. And also, in most conditions the requirements for one type of input, output or structure are not complex. It is not worth the effort building so many new components just for one original component. Other problems, such as redundancy of invoking these functions in the code of the original component and bad readability of the code, also exist.

3.3 Final Attempt

In this final attempt, I tried to build a pre-condition component, a post-condition component and an invariant component for one original component. The pre-condition component contains a function to check all the types of input. The post-condition component contains a function to check all the types of output. And the invariant component has functions of checking all the structures or types in the original AUTOSAR software components. This method effectively limits the number of newly-built components and functions. It is also the method that I finally used in the implementation period.

3.3.1 Reference Patterns

As mentioned in Section 2.2, the Design by Contract approach views the two sides of the contract as the caller and the callee (or the client and the server). For this reason, the traditional client-server pattern in the software architecture design is a very good reference pattern. In the Client-server pattern, the component types are clients and servers, and the principal connector type for it is a data connector driven by a request/reply protocol used for invoking services [18]. A client is defined as a component that invokes services from a server component. A server is a component that provides services to clients. One component can be both a client and a server. This pattern is wildly used as it "factors out common services which are reusable" [18].

Another pattern that I drew lessons from is the Proxy patten. It is not how the components in this pattern distribute, but the way it handles the invoked service that is worthy of learning. As shown in Figure 3.1, when a client component invokes a service from a server component, the proxy component will make pre-processing for the input and post-processing for the output. The pre-processing and post-processing can serve many purposes including converting formats [19]. That is why I think it can also serve as input and output checking. This approach can combine with the Design by Contract approach.

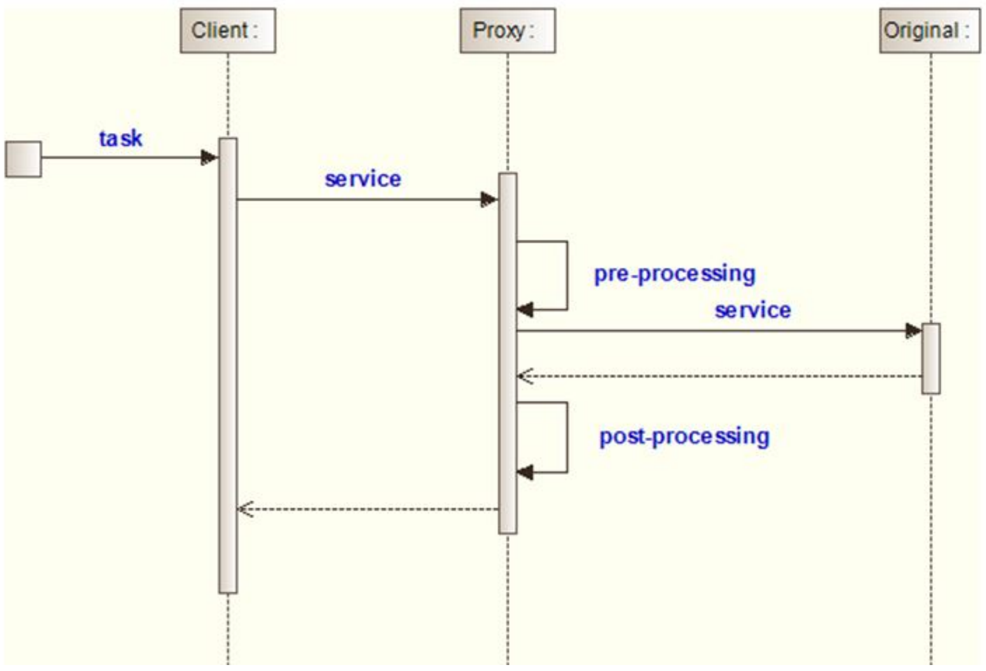


Figure 3.1: Process of how proxy pattern handles tasks [19]

3.3.2 Description of the Final Solution

When combining the Design by Contract approach with the two reference patterns, the most significant point is where to define the pre-conditions, post-conditions and invariants. Figure 3.2 shows the design for the new components enhanced with Design by Contract and how the components work together. The main processing component is almost same as the original component. It is responsible for calculating or handling the input data and generating the output data. The newly-built pre-condition, post-condition and invariant components around it are responsible for data check.

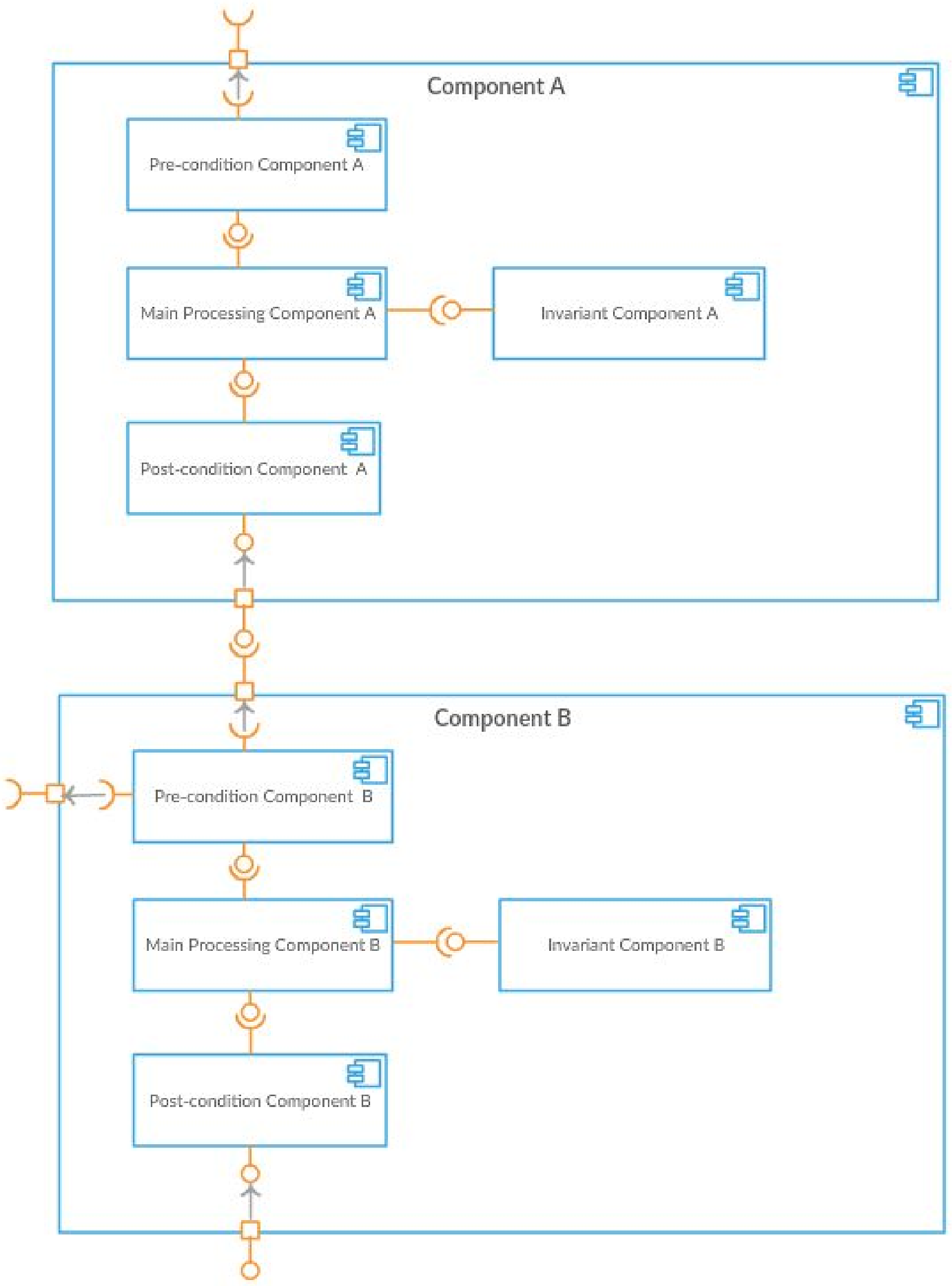

Component A
Pre-condition Component A
Main Processing Component A
Invariant Component A
Post-condition Component A
Component B
Pre-condition Component B
Main Processing Component B
Invariant Component B
Post-condition Component B

Figure 3.2: Design for AUTOSAR SW-Cs with Design by Contract

In the pre-condition component, there is a function that works for checking all the input data. If the input is invalid or erroneous, it can throw it away or make it into a default value. How to deal with it depends on the requirements. It will give the checked input data to the main processing component for further calculation. In the variant component, one or more functions are defined. Each function is used for checking one structure or type in the code of the main processing component. When there is a structure or type in the code, it will invoke the corresponding function to check this structure before using it. In the post-condition component, there is a function that works for checking all the output data. It will make sure that the output is in the reasonable range.

Considering how Proxy pattern handles input and output data, the newly-built pre-condition, post-condition and invariant components can be viewed as a proxy component. If one component needs the output data of another component as its input in an AUTOSAR software application, these two components can be seen as a client and a server. Figure 3.3 shows how the input data are handled and exchanged between the components.

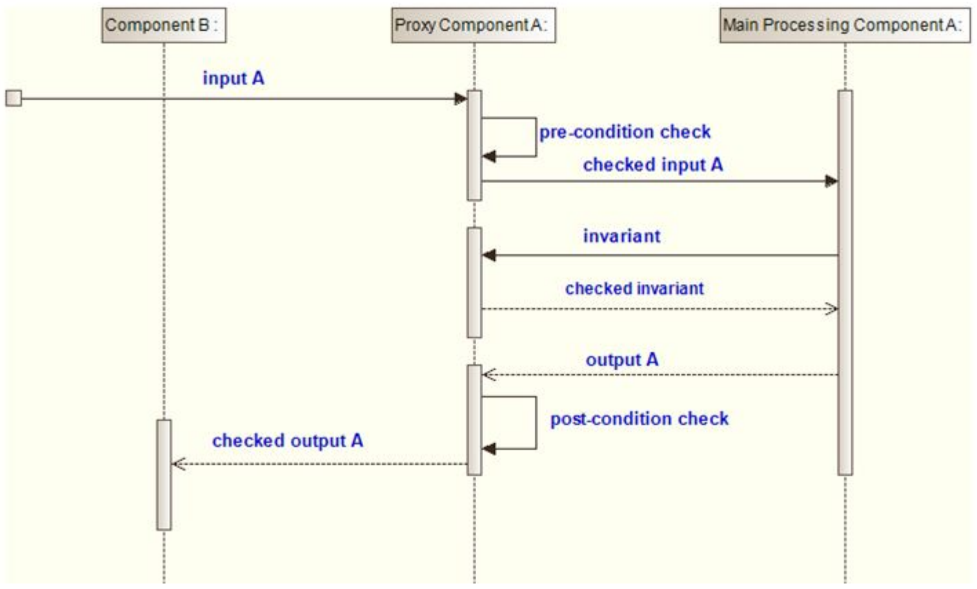


Figure 3.3: How data handled and exchanged between components

4 Implementation

This chapter introduces the process of setting up the development environment including exporting the Brake-Pedal-Input-Handler component and Brake-Light-Control component from the Arctic Studio, and importing them into ARUnit. Then, the implementation includes the identification of pre-conditions and post-conditions, and the design, modification and testing of the two software components.

4.1 Environment Setup and Components Import

The PC uses Windows 7 as the operating system and the two main development tools, Arctic Studio and ARUnit, are installed correctly on it. Arctic Studio provides a complete embedded software development environment for automotive embedded software based on AUTOSAR [20]. Another development tool, ARUnit, has been introduced in section 2.4.

Arctic Studio is the original development tool for the existing AUTOSAR SW-Cs. The whole AUTOSAR software package that was developed in the DEDICATE framework project is in this tool. It makes the architecture easy to understand and the components easy to recognize and read. By reading through the code of the components and the DEDICATE framework description [13], I selected the Brake-Pedal-Input-Handler Component and the Brake-Light-Control Component from the applications in this package as the components I would modify in ARUnit.

As Arctic Studio and ARUnit are built for different purposes and ARUnit is more efficient for running and testing one single component or certain components, in order to modify and test the two selected components independent of the relevant components in the applications, the ECU and the real running environment, I exported them from the whole package in Arctic Studio and imported them into ARUnit. The files that I imported into ARUint are the source files of the components and the software component description files of them. ARUint will generate the run-time environment for them according to these files when running them. And also, code files used for testings are built in ARUint as well.

In the software package of the DEDICATE framework project that the company gave me, I did not find a structure or type that should be checked with invariants in the components. That is why in the implementation part there are not descriptions about it. As how the functions in the invariant component work is similar to the functions in the pre-condition and post-condition components, it will not affect the verification of this method. In the AUTOSAR software components of other projects or applications, there are structures or types. That means invariant component can be used in those software components though it is not used here.

4.2 Process of Modifying the Brake-Pedal-Input-Handler Component

In this section, analysis of the original component, process of design, modifying and testing the Brake-Pedal-Input-Handler component with the Design by Contract approach are described.

4.2.1 Issues of the Current Component

Several issues were raised when reviewing the original component that may threaten the realization of the expected functionality and the robustness of the whole component. One issue is that there are not complete input check for the component. Inside the component, it does not handle the input in all the possible ranges that are mentioned in the DEDICATE framework description [13]. In other words, the input check is too simple to deal with all the possible conditions.

Another issue is that there is not output check to ensure the data gotten from the component is completely correct for the next component that uses the data. Although the calculation in this component is not complex, the errors can not be completely avoided at run time. That is why output check is necessary.

Finally, the issue concerns the internal logic and the readability of the component code. In the original component, the simple and incomplete input checks are mixed with the code which is responsible for calculations. It makes the developers hard to read and understand, which may also threaten the robustness when modifying the code.

4.2.2 Identification of Pre-conditions and Post-conditions

According to the DEDICATE framework description [13] and the package of all the program code, the Brake-Pedal-Input-Handler component is used to convert the analogue input from the pedal into a pedal position which is from 0% to 100%. The pedal provides an analogue input with the range from 10% to 90% of supply voltage (5V direct current), which means the voltage is about from 0.5V to 4.5V. If the analogue input is 0-0.5V, it means it is open circuit or short to ground. If the analogue input is 4.5-5V, it means it is short to battery. Both of them are errors. For the reason that the AD (Analog-to-Digital) converter of the microcontroller has not been calibrated, this inaccuracy has to be considered when building the software [13]. The output from the AD converter is the input for the software component

I analyse and modify here. The input is a 12 bit value which uses 0 to represent 0V and uses 4095 to represent 5V. Of course, if considering that the input values of the test cases for this component can also from the ARUnit, the input can possibly be less than 0 or greater than 4095. Thus, input values in this range are seen invalid.

Range of Input Value
value < 0
0 <= value <= 400
401 <= value <= 499
500 <= value <= 3500
3501 <= value <= 3700
3701 <= value <= 4095
value > 4095

Table 4.1: Ranges of input value

Table 4.1 shows all the possible inputs of the Brake-Pedal-Input-Handler component. What I need to do next is to set contracts for the component. That is to say, pre-conditions and post-conditions will be discussed in detail. When setting pre-condition part of the contracts, the information from requirements specification should be carefully considered to cover all the possible inputs. Here, pre-condition is that only the input value between 401 and 3700 is seen as valid and faultless. When the input value is less than 0 or greater than 4095, it is an invalid input. Input value in this range just appears in the testing environment in ARUnit. When input value is in the range of 0-400 and 3701-4095, it is erroneous. Input value in this range represents 0-0.5V or 4.5-5V. It is generated by the errors of hardware in the vehicles. For the post-condition, it should meet two requirements. Firstly, the output of the software component should be an integer from 0 to 100 to represent from 0% to 100%. Then, the correctness check for the calculation within the component is needed. There are not structures or types used in this component. Hence, I do not need to set invariant check for it.

4.2.3 Design and Modification

The aim of the modified software component is to solve the issues that exist in the original component with the Design by Contract approach. As mentioned in Section 3.3, the separation of the contracts and the component itself is a good idea. The architecture design of the new component is shown in Figure 4.1.

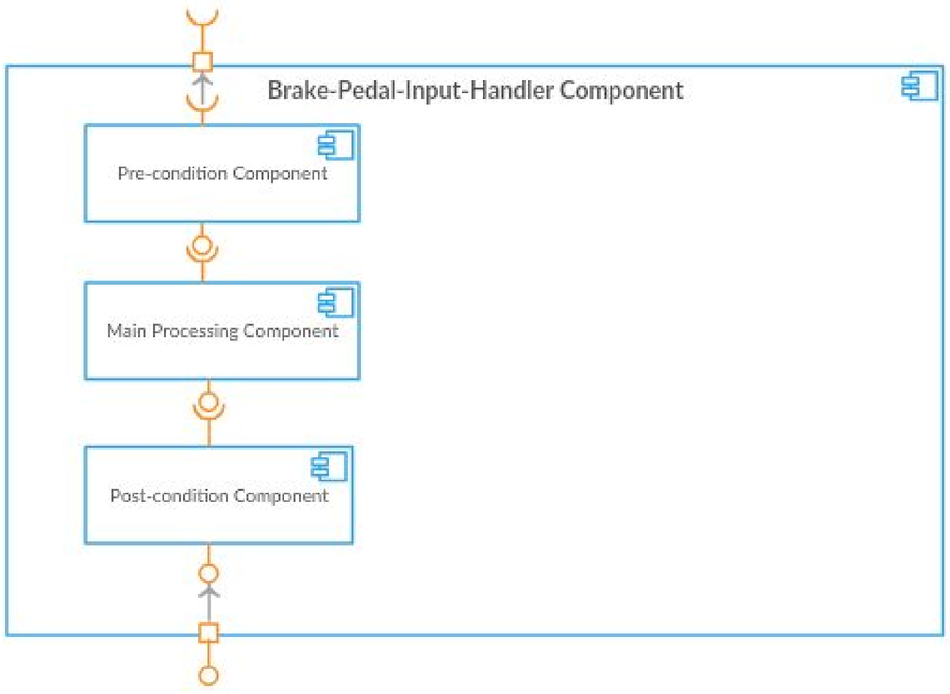

Brake-Pedal-Input-Handler Component
Pre-condition Component
Main Processing Component
Post-condition Component

Figure 4.1: Design for the Brake-Pedal-Input-Handler Component.

In the pre-condition component, there is a function. It gets the pedal signal and verifies it for the main processing component. Only the valid and faultless data, which are greater than 401 and less than 3700, can enter into the main processing component. The invalid and erroneous data are detected and handled correctly. In order to see the testing results intuitively, in my design it directly shows in the console of ARUnit that it is invalid or erroneous. For example, if the input is -100, the console shows it is an invalid input. But when running in the real ECU, other approaches should be used to handle an invalid or erroneous input because of lack of a screen. The possible approaches may be the correction of the data or getting the next input data after some time. The main processing component works for calculation of the data and giving the results to the post-condition component for checks. The code for data processing in the main processing component is from the original component. In the post-condition component, a function used for checking the calculation results of the main processing component is defined. It checks if the calculation is correct and the output is in the range of 0%-100%.

4.2.4 Test

In order to get to know if the modified component improves the robustness of the component, a testing program is created for the original component and the modified component in ARUnit. When running the testing program, the input data are sent into the component and the output data are shown on the console in ARUnit through the testing program. For better readability of the output data, some additional comments are attached with the output data to show the status of this running. Of course, this is used just in the testing environment to help develop the component and collect information. In the real ECU, there are not such additional comments at run time.

In Table 4.2, some examples of input data in all possible ranges of the Brake-Pedal-Input-Handler Component are shown. According to the DEDICATE framework description [13], the expected outputs of the software component are also included to help readers better understand the testing.

Range of Input Value	Input Example	Expected Output
value < 0	-100	invalid input
0 <= value <= 400	200	erroneous input
401 <= value <= 499	450	0, successful
500 <= value <= 3500	2100	53, successful
3501 <= value <= 3700	3600	100, successful
3701 <= value <= 4095	3900	erroneous input
value > 4095	6000	invalid input

Table 4.2: Examples of input for the Brake-Pedal-Input-Handler component and the expected output

In the testing period, 70 different input data are tested for both the original component and the modified component. If the output gotten from the component is the same as the expected output, it means it is a successful running. Besides the testings that get valid output data such as 0, 53 and 100 which can be used by other components are seen as successful testings, the testings that successfully detect invalid input or erroneous input are also seen as successful testings. The results of the testing are described in section 5.1. The robustness of the tested software component can be seen through how many of the test cases are successful.

4.3 Process of Modifying the Brake-Light-Control Component

In this section, the process of modifying the Brake-Light-Control component with the Design by Contract approach will be described. Since this section emphasizes on the collaboration between the Brake-Light-Control component and the Brake-Pedal-Input-Handler component, some descriptions similar to Section 4.2 are omitted.

4.3.1 Analysis and Identification of Pre-conditions and Post-conditions

The existing issues for the Brake-Light-Control component is similar to the Brake-Pedal-Input-Handler component. It does not have complete input checks for the input data to cover all the possible input data. Some simple input data checks are mixed with the program code. Also, it lacks output checks. The modification for the original component is expected to solve these issues.

According to the DEDICATE framework description [13], the Brake-Pedal-Input-Handler component is used to control the brake lights by the rules described in Section 2.1.2.2. Its input should be the pedal position and the vehicle speed. The pedal position is output of the Brake-Pedal-Input-Handler component. It is very easy to know the pedal position should be between 0%-100%. The range of the vehicle speed depends on different situations. Here, I set the highest vehicle speed as 300 km/h. Another factor that affects the output of the component is the status of emergency braking. The status of emergency braking can be active or inactive. In order to concentrate on the collaboration of the two modified components, it is directly sent into the Brake-Light-Control component as another input without being included in the pre-condition. Thus, the pre-condition for this component is that the pedal position should be 0%-100% and the vehicle speed should be 0 km/h-300 km/h. For the post-condition, it should check if the calculation in the component is correct.

4.3.2 Design and Modification

The design of the new Brake-Light-Control component is similar to the new Brake-Pedal-Input-Handler component. The pre-condition and post-condition are separated from the main processing component as the pre-condition component and the post-condition component. Figure 4.2 shows how these components work together.

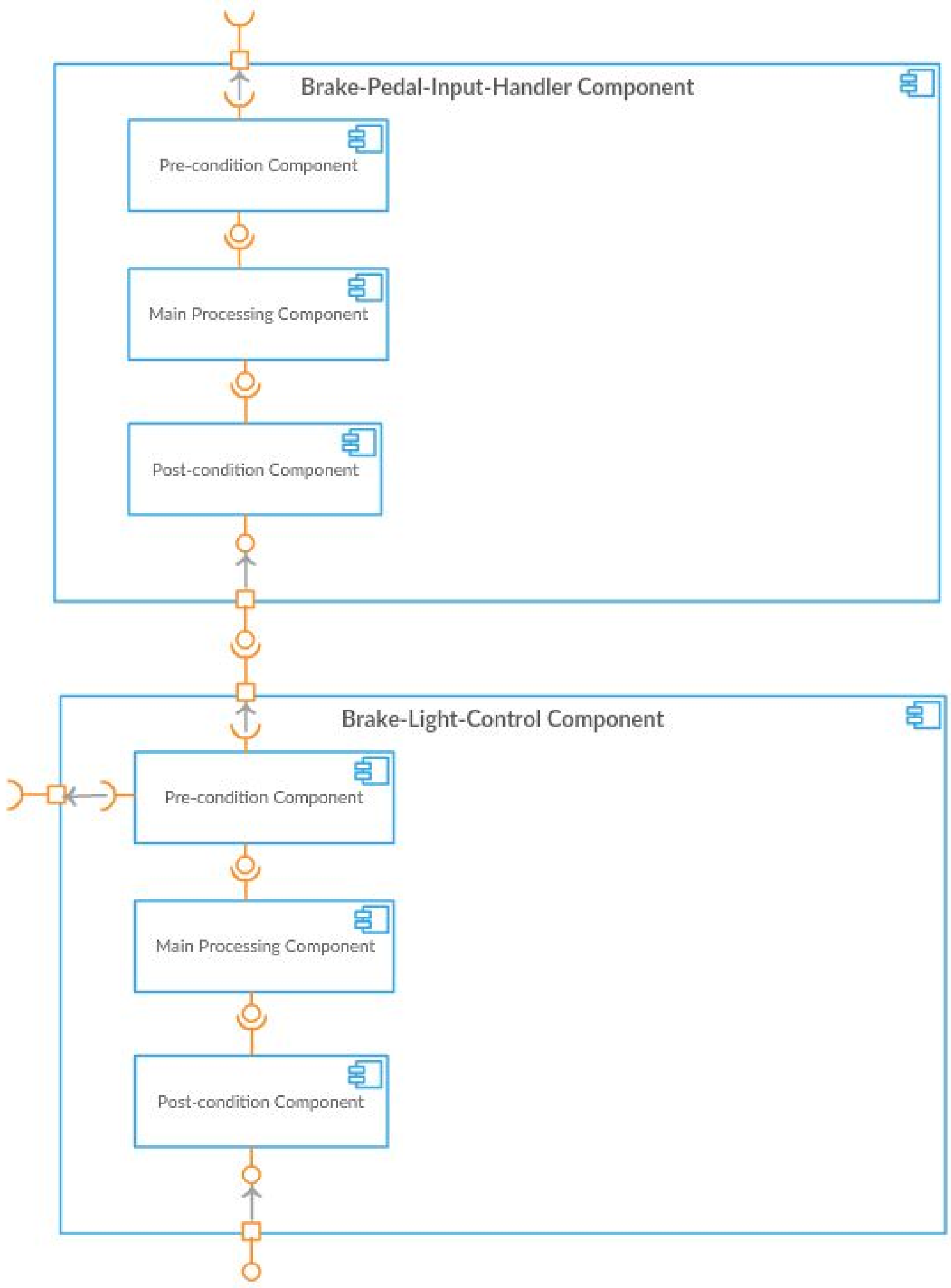

Brake-Pedal-Input-Handler Component
Pre-condition Component
Main Processing Component
Post-condition Component
Brake-Light-Control Component
Pre-condition Component
Main Processing Component
Post-condition Component

Figure 4.2: Design for the two selected components in the testing environment

There is a function in the pre-condition component of the Brake-Light-Control component. It gets the input data from the Brake-Pedal-Input-Handler component and other sources, and then verifies the data for the main processing component. Only the input data with pedal position from 0% to 100% and vehicle speed from 0 km/h-300 km/h are valid. The code for data processing in the main processing component is from the original component. In the post-condition component, a function used for checking the calculation results of the main processing component is defined. It checks if the status of braking light is correct.

4.3.3 Test

In order to know if the two modified components can collaborate with each other and improve the robustness, a testing program is created for the original components and the modified components in ARUnit. When running the testing program, the input data are sent into both the two components. The similar data as Section 4.2.4 are passed to the the Brake-Pedal-Input-Handler component. Its output data are used as the input data for the Brake-Light-Control component with the vehicle speed and the emergency braking status. The output is the status of the brake lights. It can be ON/OFF/BLINK. Some comments are attached to the output to know which input is detected as invalid or erroneous.

In Table 4.3, some examples of input data are shown. According to the DEDICATE framework description [13], the expected outputs of the software component are also included to help readers better understand the testing.

Brake Pedal Input	Vehicle Speed	Emergency Braking Status	Expected Output
2000	5	active	ON
450	35	inactive	OFF
3000	35	active	BLINK
200	35	inactive	erroneous brake pedal input
500	400	inactive	erroneous vehicle speed

Table 4.3: Examples of input for the two selected components and the expected output

In the testing period, 30 different sets of input data are tested for both the original components and the modified components. If the output gotten from the components is the same as the expected output, it means it is a successful testing. If it successfully detects the invalid or erroneous input, it is still seen as a successful testing. The results of the testing are described in section 5.2. The robustness of the tested software component can be seen through how many of the test cases are successful.

5 Results and Evaluation

In this chapter, the results of the testings described in Section 4.2.4 and Section 4.3.3 are shown. And also, the analysis and evaluation for the modification of the software components are conducted.

5.1 Testing results of the Brake-Pedal-Input-Handler Component

The testing results for the Brake-Pedal-Input-Handler component are shown in Table 5.1. For the original component, it failed 15 times in the 70 test cases. The modified component failed 0 time in the 70 test cases. The success rates of them are 78.6% and 100.0% respectively. Obviously, the modified component has better robustness and can handle more input data successfully. The analysis and evaluation are conducted in Section 5.3.

	Successful Testings	Total Testings	Success Rates
Original Component	55	70	78.6%
Modified Component	70	70	100.0%

Table 5.1: Results of testings for the Brake-Pedal-Input-Handler component

5.2 Testing results of the two modified Components

The testing results for the two modified components are shown in Table 5.2. For the original components, it failed 9 times in the 30 test cases. The modified components failed 0 time in the 30 test cases. The success rates of them are 70.0% and 100% respectively. Obviously, the modified components have better robustness and can handle more input data successfully.

	Successful Testings	Total Testings	Success Rates
Original Component	21	30	70%
Modified Component	30	30	100%

Table 5.2: Results of testings for the two modified components

5.3 Evaluation

For the testing results, all input data of the failed test cases are in the ranges of invalid input or erroneous input. The reason of why getting such results is that the original component has incomplete and inaccurate input check in it. It does not cover all the possible input data from different ranges. Although these invalid or erroneous input data seldom appear or will be replaced by the next input quickly in the real running in the ECU, they cannot be ignored.

There are two main reasons that make the modified components get better results. The first one is that all possible input data have been considered by carefully analysing the documented specification when designing this new component. The invalid input data and erroneous input data have been handled in the pre-condition component and do not have the chance to get into the main processing component. The second reason is that the data from the main processing component are checked again in the post-condition component to ensure its correctness. The pre-condition and post-condition components are like two guards that check all the input and output data of the main processing component.

The prerequisite of setting such pre-condition and post-condition components that can accurately cover all the possible input and output data, is that we need to have complete requirements for the components. Some relevant information from the company can prove the two software components used in this thesis are representative. When developing AUTOSAR software components in the company, the bottom line for the requirements of the components is that the whole range of every input and output must be specified. And if they are not, someone will revise or update the requirements. It means that almost all the requirements specify the ranges of the input and output, and can be used to help set accurate pre-condition and post-condition for the components like what I did in this thesis.

If evaluating the design of the components, the components themselves and the collaboration among different components work well in the testing environment. For this design, there are clear and complete input and output checks that are conducted by the pre-condition and post-condition components. The main processing component can also focus on data processing. Moreover, the component code becomes more readable and easier to modify when necessary. What is different from the real components in the ECUs is that in order to read the checking results more easily, the components in this thesis directly shows the results in the console. The way of detecting errors is the same. But in the real ECUs, if we detect errors, the component will use other ways to handle these invalid data. For example, in the Brake-Pedal-Input-Handler component, the component will switch to another sensor to get the input if the input from one sensor is invalid. How to handle the invalid data depends on the requirements. That is another reason why the well-specified requirements are important.

Some strengths and weaknesses of using Design by Contract in AUTOSAR software

Strengths	Weaknesses
• Better robustness of the components and applications	• Add more components which may also bring errors to the components
• Increase readability and understandability of the code	• Strict data checks may slow the components and applications down
• Convenient to refactor manually coded software components	• Hard to modify the components of which the code is automatically generated from some models
• Low redundancy	

Table 5.3: Strengths and weaknesses of the Design by Contract approach in AUTOSAR

components have been mentioned in this thesis report more or less. A summary is listed in Table 5.3.

Another thing that needs to be evaluated here is in which situation the Design by Contract approach can be used to improve AUTOSAR software components' robustness. As known from the DEDICATE framework description [13], the code of some components is generated from TargetLink which is a modeling and development tool. Such kind of code is hard to read, import into ARUnit and modify manually. These components that have code automatically generated by TargetLink should follow a different approach. Contracts including pre-conditions and post-conditions should be embedded in code generation instead of trying to modify the code a-posteriori, as done by the approach proposed in this thesis. In the source base, nearly 50% of the AUTOSAR software components are generated from TargetLink. Thus, at least 50% of the components of which the code is written manually are able to be modified with the Design by Contract approach to improve their robustness. Moreover, new AUTOSAR software components can certainly be designed and implemented with the Design by Contract approach.

6 Conclusion and Future Work

In this chapter, the conclusion that the Design by Contract approach can be applied to build and modify AUTOSAR software components for better robustness is proved. And possible future work for relevant topics is also described.

6.1 Conclusion

The Design by Contract approach can be applied to build and modify AUTOSAR software components for better robustness. My way in the thesis is to build additional pre-condition, post-condition and invariant components around the main processing component. In other words, checks for input, output and invariant are separated from the original component. By testing the components modified in this way, the results prove that it improves robustness. This new design of components has strengths as better robustness, better readability, better understandability and low redundancy. It is very easy to apply this way to modify existing software components that are manually coded. Two points are worth mentioning when applying this way. One is that the developers should analyse the documented specification or the stakeholders' requirements carefully to know all the possible values of the input, output and invariant. Another is that the pre-condition, post-condition and invariant components should be well-defined to cover all the values known from the first point. Certainly, the weaknesses such as the possibility of the errors brought by additional components, also exist. That is why we need to go through careful consideration when applying this way of Design by Contract. The Design by Contract approach has been wildly used in many different software applications and tools supported by programming languages themselves or third-party tools. There may be other more effective ways of applying Design by Contract to AUTOSAR software components to improve robustness. That is worthy of further exploration.

6.2 Future Work

Many AUTOSAR software components are automatically generated from models in modeling tools. For example, some software components in the DEDICATE framework are modeled in Simulink or TargetLink and C source code is generated from TargetLink [13]. These components are hard to modify with my way of applying Design by Contract. If the pre-condition, post-condition and invariant components can be modeled in the modeling tools and C source code can also be automatically generated, this will be a great progress.

Further more, when searching for Design by Contract on the Internet, there are many third-party tools that can be used to support the programming languages that do not have Design by Contract language features. Developing a tool that can support C language in AUTOSAR system to directly define pre-conditions, post-conditions and invariants for AUTOSAR software components is worth of trying.

Bibliography

[1] B. Fleming, An overview of advances in automotive electronics, Vehicular Technology Magazine, IEEE 9 (1) (2014) 4-9.

[2] AUTOSAR, https://www.autosar.org, accessed: 2016-04-02.

[3] Meyer, Bertand. 1997. Object-Oriented Software Contruction (2nd ed.). Upper Saddle River, NJ: Prentice-Hall.

[4] IEEE Standard Glossary of Software Engineering Terminology, IEEE Std 610.12-1990.

[5] Meyer, Bertrand: Design by Contract, Technical Report TR-EI-12/CO, Interactive Software Engineering Inc., 1986.

[6] Meyer, Bertrand: Applying "Design by Contract", in Computer (IEEE), 25, 10, October 1992, pp. 40–51.

[7] Liu, Yi, and H. Conrad Cunningham. "Software component specification using design by contract." Proceeding of the SouthEast Software Engineering Conference, Tennessee Valley Chapter, National Defense Industry Association. Vol. 6. sn, 2002.

[8] Cheon, Yoonsik, et al. "Model variables: Cleanly supporting abstraction in design by contract." Software: Practice and Experience 35.6 (2005): 583-599.

[9] Benveniste, Albert, et al. "Contracts for the design of embedded systems part i: Methodology and use cases." Contract 2 (2011): G1.

[10] Thüm, Thomas, et al. "Applying design by contract to feature-oriented programming." International Conference on Fundamental Approaches to Software Engineering. Springer Berlin Heidelberg, 2012.

[11] A. Collins, D. Joseph, and K. Bielaczyc, Design Research: Theoretical and Methodological Issues. Journal of the Learning Science 13, 1(2004), pp. 15-42.

[12] AUTOSAR, AUTOSAR Technical Overview v2.2.2. 2012.

[13] M. Jones and J. Haraldsson, D2.4 DEDICATE Framework Description, 2012.

[14] http://www.dinisio.net/nicola/papers/verify12-en.pdf, accessed: 2016-05-30

[15] http://www.onlamp.com, accessed: 2016-05-06.

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

[17] https://www.artop.org/arunit, accessed: 2016-06-10.

[18] Reviewer-Herzog, Jared. "Software Architecture in Practice Third Edition Written by Len Bass, Paul Clements, Rick Kazman." ACM SIGSOFT Software Engineering Notes 40.1 (2015): 181-183.

[19] Riccardo Scandariato. "Tricks of the trade: architectural patterns" Advanced Software Architecture slides, 2015.

[20] http://www.arccore.com/products/arctic-studio, accessed: 2016-08-30.

[21] 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.

[22] P. Koopman, J. Sung, C. Dingman, D. Siewiorek, and T. Marz, “Comparing operating systems using robustness benchmarks, In: Proceedings of the 16th Symposium on Reliable Distributed Systems, pp. 72-79, IEEE, 1997.

A

Terminology and Abbreviations

The terminology and their abbreviations used in this thesis are listed in the following table.

Abbreviation	**Description**
AUTOSAR	AUTomotive Open System ARchitecture
BBW	Brake by Wire
DbC	Design by Contract
DEDICATE	Dependability and Diagnostics Concept Assessment and Test
ECU	Electronic Control Unit
E/E	Electrical and/or Electronic
PC	Personal Computer
RTE	Runtime Environment
SW-C	Software Component
VAP	Volvo AUTOSAR Platform
VFB	Virtual Functional Bus

Table A.1: Terms and abbreviations