CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Automated Testing for Automotive Infotainment Systems

Master's thesis in Embedded Electronic System Design

Ning Yin

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY AND UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2018

Master's thesis 2018

Automated testing for automotive infotainment systems

Ning Yin

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden 2018

Automated testing for automotive infotainment systems
NING YIN

Supervisor: Lena Peterson, Chalmers University of Technology
Examiner: Per Larsson-Edefors, Chalmers University of Technology

Master's Thesis 2018
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Automated testing for automotive infotainment systems
NING YIN
Department of Computer Science and Engineering
Chalmers University of Technology

Abstract

With the development of automotive industry, the complexity of infotainment systems is increasing due to the growing number of electronic control units (ECUs). In-vehicle infotainment (IVI) is gradually becoming one of the main features in high-class vehicles nowadays. Automotive companies find it a challenge to test these complex functions for ensuring product quality before start of production. Therefore, it is highly demanded to carry out high volume of infotainment tests by test automation to shorten test cycles, improve the quality and save resources.

To address this challenge, one purpose of this thesis is to investigate a suitable area for test automation within the infotainment area. In the first part, a testing framework of an ECU diagnosis functionality is introduced. Test cost and effort estimation is also considered in this part. The second part is to develop a Python script which is applied in an ECU test for communication and surveillance purposes. The outcome of the first task can be utilized as a background material for ÅF business in related areas while the Python script can be employed in a future in-house development project.

Keywords: Test automation, Infotainment system, Diagnosis, Robot Framework

Acknowledgements

This thesis project was performed by Ning Yin from Chalmers University of Technology; it was carried out at ÅF in Trollhättan and Gothenburg. First of all, I would like to thank to my supervisor Ola Wennberg, Manager of Infotainment, SW & HMI at ÅF Automotive department who gave me the chance to perform this master thesis. I am grateful for all the guidance and valuable information offered by Dan Carlsson, Mikael Karlsson and Smitha Mohan from ÅF infotainment team. Second, I would like to take the opportunity to thank Johansson Andreas LV, Oskar Andersson and their colleagues for helping me with the second task of this master's thesis. Third, I want to express thanks to my supervisor Lena Peterson at Chalmers. Thank you for helping me with report writing and giving continuous feedback along with the project. Last, I am also thankful to my examiner Per Larsson-Edefors for reviewing and giving feedback on my thesis report.

Ning Yin, Trollhättan & Gothenburg, July 2017

Contents

List of Acronyms

ADAS	Advanced Driver Assistance Systems
AIN	Analog Input
API	Application Program Interface
ATDD	Acceptance Test-driven Development
AUT	Application Under Test
CAN	Controller Area Network
CAPL	Communication Access Programming Language
CSMA/CD	Carrier-sense Multiple Access with Collision Detection
DAQ	Data Acquisition
DC	Direct Current
DDT	Data-driven Testing
DTC	Diagnostic Trouble Codes
ECU	Electronic Control Unit
EMI	Electromagnetic Interference
FIO	Flexible I/O
HMI	Human Machine Interface
HTML	Hypertext Markup Language
ICE	In-car Entertainment
IVI	In-vehicle Infotainment
LAN	Local Area Networking
LIN	Local Interconnect Network
MBT	Model-based Testing
MOST	Media Oriented Systems Transport
OBD	On-board Diagnostics
ODX	Open Diagnostic Data Exchange
OSI	Open Systems Interconnection
POF	Plastic Optic Fiber
QoS	Quality of Service
ROI	Return On Investment
RTR	Remote Transmission Request
SID	Service Identifiers
SOF	Start of Frame
SOP	Start of Production
SUT	System Under Test
TSV	Tab-Separated Values
UDS	Unified Diagnostic Service
XML	Extensible Mark-up Language

1
Introduction

Automotive infotainment systems are a combination of "information" and "entertainment" [1]. Although a fault in infotainment systems can hardly threaten people's lives, they still influence drivers' user experience in the car. For example, an incoming phone call should not be blocked by a radio function; the Bluetooth function should enable external devices (e.g. phone, iPad) connected to the vehicle to control music, answering calls from the phone; an unexpected low operating voltage should be reported to the driver and shown on the display screen. From research released by GFK Automotive (Gesellschaft für Konsumforschung), five out of top ten purchasing decisions for cars are associated with infotainment features [2, 3]. With an increasing complexity and diversity of electric and electronic systems, conventional manual testing has a hard time meeting quality and time requirements. Therefore, test automation comes into the picture to improve the efficiency and reduce required human resources. However, it is worth noting that manual testing cannot be replaced completely with automated testing. Short-term projects, judging requirement from human intuition and thinking are possible reasons behind this phenomenon.

1.1 Project Background

ÅF is an engineering and consulting company that delivers consulting service in the automotive field. One of the main tasks in the automotive department is to explore the next generation of vehicle infotainment systems. Due to the advancement of automotive electronics technology and rising of personalized requirements on customer service, functions such as climate control, wireless communication, digital video and navigation are integrated into today's IVI systems [4]. The design of infotainment systems is gradually becoming one of the key-areas in the automotive industry. Infotainment systems are composed by numerous ECUs connected through in-car network. Different protocols can communicate with each other through gateway ECUs. How to ensure correct functionality of these ECUs is the root of having a flawless and seamless connectivity IVI system.

The testing of infotainment systems is conventionally carried out by experts manually. This approach has significant limitations because the testing takes a long time while still having limited test coverage [5]. In addition, the accuracy requirement is difficult to meet along with continuous tests and shorter delivery in a sprint (or iteration), which is a basic development unit in an agile project framework. Therefore,

test automation with support of proprietary tools and scripts is highly demanded by car manufacturers. An automated testing system would decrease the need of human resources and the intervention of experts as well as lead to enhancement of efficiency, performance and shorter lead time [6].

1.2 Motivation

The customer expectations on new entertainment application and services in their cars forces the automobile infotainment industry to evolve continuously. Meanwhile increasing complexity of in-vehicle electronics makes infotainment system integration difficult. In today's superb vehicles, the infotainment system works as a distributed system and integrated system, where hardware and software components interact by means of in-vehicle network. The typical problems such as interruption from another feature, concurrency and consistency usually occur in service interaction. Functional testing, regression testing and robustness testing need to be performed on these features against erroneous action. Hence, new testing methods should be adopted for replacing old hardware or software test approaches with more flexibility. For instance, a new testing method for supporting multiple kinds of input/output, such as GPS data from navigation, information from advanced driver assistance systems (ADAS) in a single IVI system. Another example is finding a common solution to ever-increasing types of infotainment standards and connectivity protocols [7]. If a tester wants to check car performance under circumstance of running multiple tasks, i.e. a driver uses GPS while connecting the phone to Bluetooth and adjusting indoor temperature at the same time, testing environment needs to be simulated involving all task features. This means that the combination of navigation system, Bluetooth connectivity and communication network must have correct synchronization. Therefore, new tests should take this into account and analyze the exact occurrence timing when these tasks are carried out.

The operation of infotainment systems depends on different embedded software capabilities. Due to frequent development and change of software before production, it is essential to validate changes and test features after each update. Moreover, regression test has to be performed to ensure all previous functions are intact. Therefore, a challenge is introduced for adapting new changes fast while securing other functions remain intact. It can be tedious and error prone with manual work and thus test automation is needed through development to reduce time and improve quality.

Another challenge lies in the automation oracle problem, which is also known as automatic generation of test cases [8]. When applying automated testing in an agile project, efforts need to be made to quickly react to consecutive changes to the system requirements. New test cases are invoked correspondingly and should be added to the existing test automation in parallel with the development of the project. For example, tests will fail if the order of two buttons in test case is changed while this change has not been generated as a new test case. The test-generation problem becomes more intractable when part of the testing environment is out of engineers' control or the testing environment is indeterminate [9].

The decision of what tests to automate should be taken at the inception stage and based on the comparison between the value of test automation and effort to produce them. The initial phase of assessing different testing goals within the infotainment area is vital since it gives the direction where testing is laborious and time consuming to perform manually. Other challenges such as architecture of test code, test code language and test environment specification must also be considered in this project [10].

1.3 Project Goals

With a planning stage and literature study phase in the beginning, sufficient background knowledge within automotive infotainment field is acquired. The project itself can be generally divided into two parts, which correspond to two goals respectively.

- The first aim is to analyze a suitable area using automated test for IVI system. The chosen area should be agreed on based on motivation and rationale of the domain. Afterwards, the area deemed as suitable needs further investigation regarding how automation should be applied and consider test effort estimation.

- The second goal is to create a small useful Python script which can be applied to an ECU test in an automotive system. The validation of the script will be achieved by comparing test results with expected behavior of the application in the vehicle test environment.

1.4 Limitations

In this project, the limitations are:

- In the first part of this thesis, the author will not have access to the real infotainment systems and hence this part work is a purely theoretical implementation.

- The coverage of infotainment systems will be provided by ÅF engineers, which means the scope of test automation discussed in this thesis project is confined, even if the study would show boundless viable areas can be investigated.

- By the time the author finishes the thesis, the script has not be performed on the complete automotive system due to the project progress in the company. However, the test principle is the same and the existing testing result proves that the method can be applied to future ECU test when the hardware is ready.

1.5 Ethical Aspects

Due to the automated testing features and characters of IVI in this project, it is necessary to take into account ethical implications. First, the sustainable development should be considered. A good choice of an infotainment automated testing area should always take into account the durability and frequency of the test. A manual testing is recommended when the test is not performed often so as not to waste the resource. Second, a trade-off between the quality and cost of test automation is necessary to be made. Third, the coverage of test automation must be borne in mind that automated testing may cover other areas not done by manual testing and vice versa. As a result, it comes to ethics if the testing is thorough to assure the system related with human drivers reliable. Fourth, the thesis project will receive support from engineers at ÅF company, however, the work should be done individually and comply with the privacy policy within the company. Regarding to the safety standard for electrical and electronic systems, ISO 26262 is not especially required in this project.

1.6 Report Layout

The thesis report is composed of following chapters:

Chapter 2 starts with an introduction to the domain of automotive electronics. In-car infotainment as a part of automobile electronics application is then expanded to details regarding its architecture, network used for realizing entertainment features, the gateway function and diagnosis function inside ECUs. This chapter also includes information on test automation technology.

Chapter 3 describes the methodology used for conducting the thesis work and outlines the research questions.

Chapter 4 demonstrates the testing framework for ECU diagnosis. A new test cost estimation model is also presented in this chapter.

Chapter 5 illustrates the implementation and execution of an ECU test automation with communication and surveillance purpose .

Chapter 6 analyzes the result achieved and answers the research questions from Chapter 3.

Chapter 7 arrives at the conclusion made from this project and proposes areas to investigate in future.

2

Technical Background and Current Situation

This chapter introduces background knowledge related to the thesis work. The first section refers to automotive electronics, especially automotive infotainment system. Relevant understanding includes diagnosis functionality, various network communication and network gateway. Next comes the state of automated testing technology. The comprehension of these two sections will ease the implementation of automated test framework raised in the later chapter of this thesis.

2.1 Automotive Electronics Technology

Automotive electronics are organized as distributed system and developed for controlling vehicles by various embedded computer units. These embedded computer units are known as ECUs. Today, there are typically more than 100 ECUs integrated in a modern car via in-vehicle network for different kinds of purposes. The purposes can be mainly classified into: Active and Passive Safety, Powertrain and Chassis Control, Passenger Comfortableness, Infotainment System and Engine Electronics. In the following parts, infotainment system will be explained in detail.

2.1.1 Automotive Infotainment

Automotive infotainment, which is also known as IVI or in-car entertainment (ICE), it is a multi-functional interactive hardware and software system that provides information services, communication services and entertainment services [11]. Entertainment services include audio/video, radio, rear-seat entertainment etc. Communication services such as phone calling using voice control technology allow drivers to have hands-free answering. Another well-known communication service is offered by Bluetooth, which realizes information interchange with the vehicle by wireless predefined communication [12]. When the driver gets a call, all other audio related services are turned mute and the calling from phone gets transmitted to car stereo system. Rear parking assistance from information services can detect the presence of vehicles and warn drivers of danger. This vehicle-to-vehicle communication introduces an innovative method for traffic safety and should be utilized without potential risk of distracting the drivers. The objects at the rear part of vehicle are recorded by a camera and displayed on the central display. Other information services provide vehicle related information such as the overall distance the vehicle can cover with

the fuel level, door security, traffic information and weather forecasts are required by sensors and shown on the display screen.

Generally, IVI systems must have the help of human machine interface (HMI) to deliver and process the services mentioned above. The usability of HMIs has increased considerably due to the features of IVI systems. The correctness of a user interface is essential to have a faultless experience of IVI system. A typical HMI consists of central touchscreens, control unit, keypads, multi-function buttons, etc. Figure 2.1 shows an infotainment system example of Volvo V60.

Figure 2.1: Infotainment system of Volvo S90, where 1. Voice control unit 2. Touch screen and 3. Keypads. Photograph taken by author.

2.1.2 IVI architecture

The applications such as navigation system and audio/visual function are accessed by either hardkey buttons or via a panel display located in the center console [13]. The design of tactual modalities, such as touch-screen and key-console for human-computer interaction, is optimized towards simple one-hand manipulation in order to prevent the driver from being distracted as much as possible. IVI systems are regarded as modal systems that support interaction among human, exterior environment and circumstances inside the vehicle. The complexity of IVI systems increases not only due to growing number of infotainment features but also caused by multi-haptic modalities (e.g. knobs, buttons, touchscreens), which must be synchronized correctly and consistently to present the system status to the driver. The connection between IVI systems and other vehicle functions, for instance engine control or driver assistance systems, also identify a need of a reliable IVI system to facilitate the information acquisition. Figure 2.2 shows three main structural parts of an infotainment system and their interaction. The detailed explanation is shown below:

- The user interface provides all input modalities such as touching screen and haptic buttons for entering command from drivers. The output modalities like

speaker are integrated into user interface as well.

- Applications can be divided into two groups generally. One is called head unit that offers driver control over the vehicle's entertainment media [14]. It is usually integrated in the center of the dashboard. The other is electronic instrument cluster unit which includes a set of instrumentation such as speedometer, odometer and instrument.

- Management part refers to administration functions such as management of external devices. The connection master has a table containing information about all existing connections [15]. The file system stores social media documents such as photos, musics and videos.

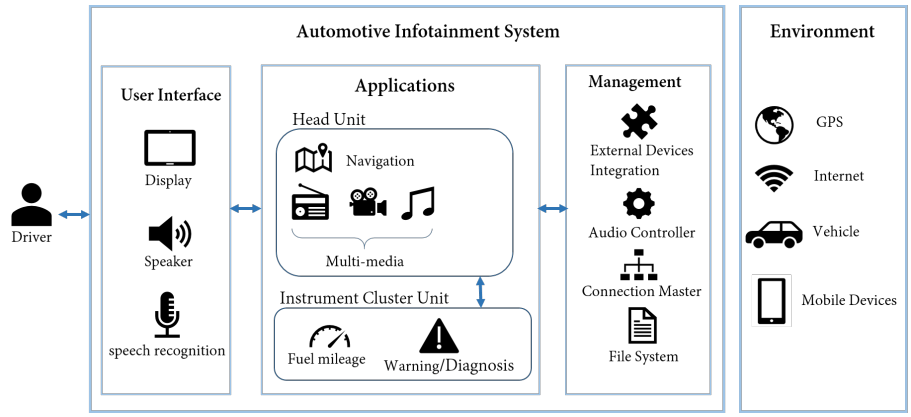


Figure 2.2: Infotainment system structure

In each part, different embedded ECUs interact via one or more communication buses such as media oriented systems transport (MOST) and controller area network (CAN) shown in Fig. 2.3. The amplifiers are to increase the signal amplitude provided by the head unit [16]. The head unit communicates with other functions inside the vehicle through the CAN bus.

2.1.3 In-vehicle Networks

To realize the features of IVI system and architecture mentioned in the last two sections, in-vehicle networks are required for exchanging information among ECUs. The development of automotive networks started in the early 90's. Before that, point-to-point communication connection was adopted by in-vehicle ECUs. $(N^2\text{-}N)/2$ links are needed for N ECUs. Therefore, the number of ECUs and links in this strategy has exponential relationship, which cannot be used to deal with growing number of ECUs [17]. This becomes the motivation of using multiplexed communication mechanism over a shared medium.

Different communication network protocols are used to meet requirements of bandwidth and implementation of distributed systems throughout the car. For example, around 40 ECUs are integrated in Volvo XC90 by interconnection of a MOST bus,

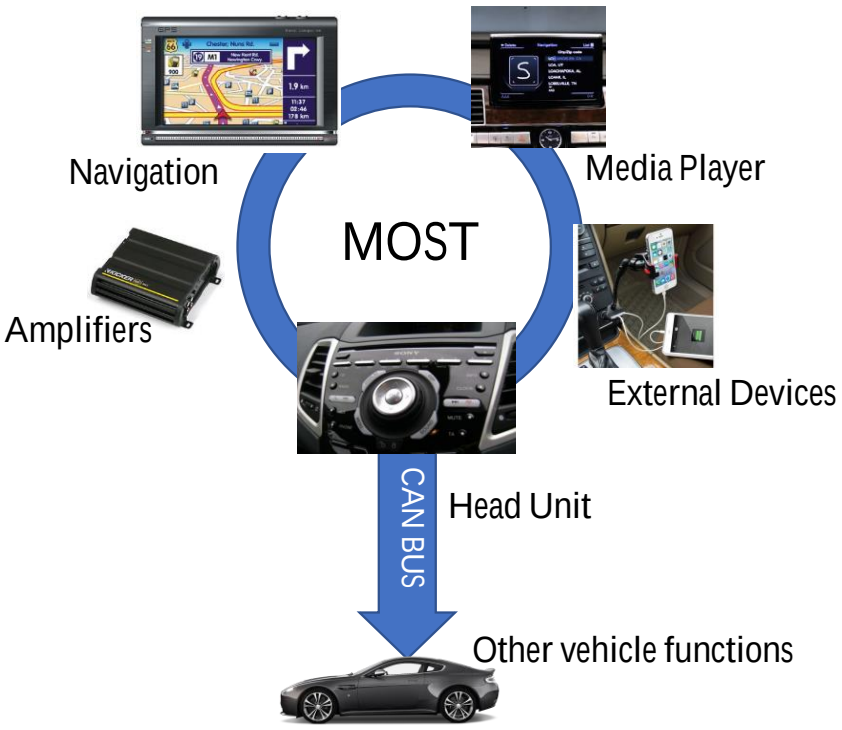


Figure 2.3: Infotainment system interaction architecture

both low-speed and high-speed CAN bus and a local interconnect network (LIN) bus. Rodelgo-Lacrus et al. [18] classified networks into several levels based on data rate and functions. LIN is an example of Class A network, which is used for simple data transmission with a speed lower than 10 kbit/s. It is the cheapest among automotive networks and used when there is no high versatility needed. Class B network such as low-speed CAN offers transmission rates up to 125 kbit/s. Together with high-speed CAN belonging to Class C network which operates from 125 kbit/s to 1 Mbit/s it composes the CAN bus network. CAN is also called multi-master broadcast serial bus since each node can transmit and receive messages with fault tolerance. Class D networks such as MOST, are designed for speed exceeding 1 Mbit/s, mainly serving as gateways between subsystems and carriers for audio, video or other media data. In the following part, a more detailed introduction for CAN, MOST and Ethernet is presented.

CAN Network

CAN is an event-triggered bus system for real-time nodes such as temperature sensors, driver door module. It was first released in 1986 by Robert Bosch GmbH and applied in vehicles. After that other application areas, for instance trams, undergrounds and aircraft, started adopting it for many applications. The CAN network provides a single interface for ECUs instead of having several input nodes on the device. The broadcast communication feature of CAN makes every device linked to the network notice the transmitted messages and decide if the message should be

accepted or neglected [20]. Therefore, the additional nodes can be added without changing the topology of the network. CAN offers non-interrupted transmission of messages that frames with the highest priority get access to the CAN bus and transmitted in broadcast. Another advantage of CAN network is error-detection capabilities supported by cyclic redundancy check (CRC) to detect global and local transmission errors. The topology of CAN network with both high speed and low speed bus is illustrated in Figure 2.4.

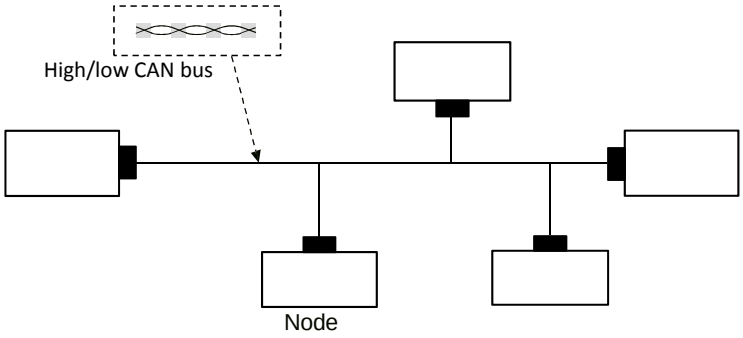


Figure 2.4: CAN topology

CAN message has four types denoted as: data frame, remote frame, error frame and overload frame [19]. Data frame, as the name suggests, is used for transmitting data to other nodes in the network. Remote frame is similar to data frame except that the remote transmission request (RTR) bit and missing of data field. Error frame is generated when a node detects an error and triggers all other nodes to send error frame as well. The purpose of overload frame is to require more time for a busy node which causes extra delay between messages [21][22].

MOST Network

MOST network is used for multimedia and infotainment applications such as videos, radios and other GPS navigation in the car. The using of plastic optic fiber (POF) cables offers a better performance against electromagnetic interference (EMI). It has a ring topology which can manage up to 64 devices (nodes). The MOST networking technology eases the way of connecting multiple devices in today's infotainment systems by plug and play functionality. MOST25, MOST50 and MOST150 are three versions of MOST network. The bandwidth is 25 Mbit/s for MOST25, 50 Mbit/s and 150 Mbit/s for MOST50 and MOST150. Not only the bandwidth is improved from the first generation of MOST, but also the frame length is increased to 3072 bits. Thus, MOST becomes more efficient when handling the increasing streaming of audio and video data.

In a MOST ring, the distance distance between two nodes is 20 meters. The communication direction is one-way transmission as shown in Figure 2.5. A time master sends MOST frames to the next node in the logical ring with a consistent frame rate

(44 kHz-48 kHz) and all other time slaves with different sampling rates synchronize their operation with the frame preamble [23]. The data can be sent through synchronous, asynchronous and control channel by different bandwidths [24]. The data is called synchronous data, asynchronous data and control data correspondingly.

- Synchronous data: Transmission of multimedia data without error detection.
- Asynchronous data: Transmission of multimedia data with high data rate and error handling.
- Control data: Management of commands and diagnostic information between network ports with error detection.

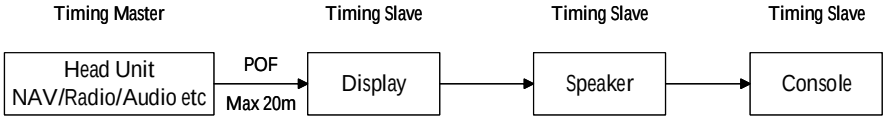
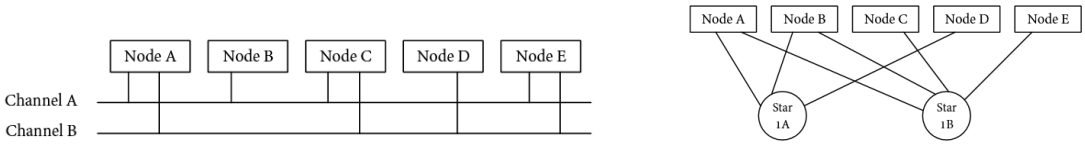


Figure 2.5: MOST timing master and slaves

FlexRay

FlexRay is a time-triggered communication network having a fixed delay when transmitting data. In contrast to unpredictable latency in CAN network, FlexRay is intended for use in safety-related and fault-tolerant systems (e.g. brake-by-wire). The protocol provides both synchronous and asynchronous data transfer. Moreover, both static and dynamic communication segments are provided in a communication cycle with a pre-defined space for static and dynamic data [25][26]. In this way, the static communication segment provides bounded delay for deterministic data while dynamic segment adjusts bandwidth requirement to meet the demand of event-based data without determinism.

FlexRay has versatile topologies such as passive bus and active star type. Figure 2.6 shows two basic layouts of FlexRay. In Figure 2.6a a node can connect to one or both of the channels. Node A, Node C and Node E are connected to both channels while Node B and Node D only connected to either Channel A or Channel B. The active star structure shown in Figure 2.6b is free from closed ring. The received signal from one node can be transmitted to all other nodes connected. Similarly, a node can link to any other channel in the topology. FlexRay can be implemented as a combination of communication bus system which improves the flexibility and adaptivity for more applications. Figure 2.7 is an example of hybrid configuration of FlexRay.



(a) Passive bus topology **(b)** Active star topology

Figure 2.6: Basic FlexRay layout

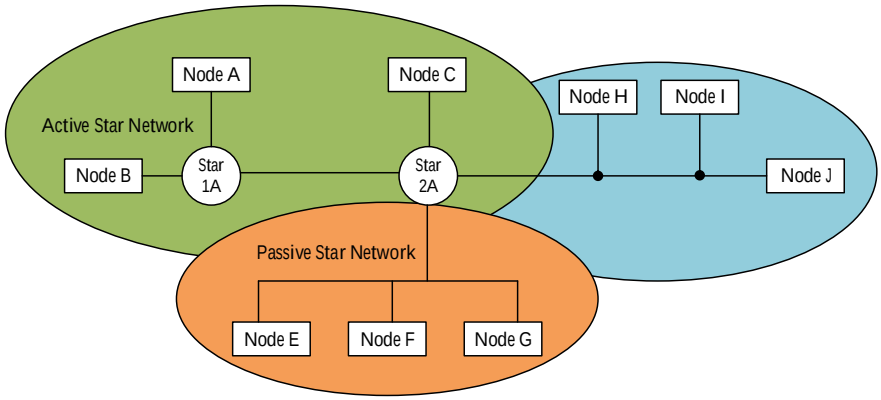


Figure 2.7: Hybrid configuration of FlexRay topology

Ethernet

Ethernet is a high-speed system with data rate 100 times higher than that of a CAN bus, which is necessary for infotainment and active safety application. The low-cost and specific quality of Service (QoS) are also the motivation for Ethernet. Ethernet is also widely used in diagnostics [27] by using local area networking (LAN) technology.

Two common Ethernet topologies are depicted in Figure 2.8. In a bus style configuration, all the nodes connected to the bus share one channel under Carrier-sense Multiple Access with Collision Detection (CSMA/CD) method. In infotainment systems, with the help of Ethernet switch, the messages from head unit can be broken down into small packets and sent to the target address. The transmission is simultaneous and bidirectional. The example in Figure 2.9 shows that there are two frames in flight on the bus between display and console node. No frame exists between head unit and speaker node since they are not involved in the transaction. The function of switch also makes Ethernet more flexible and scalable than other network topologies.

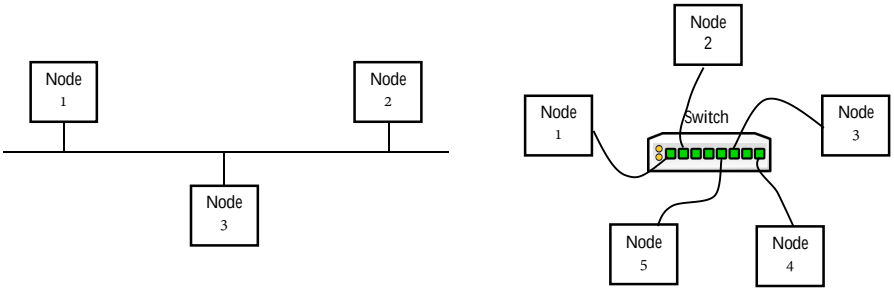


Figure 2.8: Two common Ethernet topologies: bus and star based

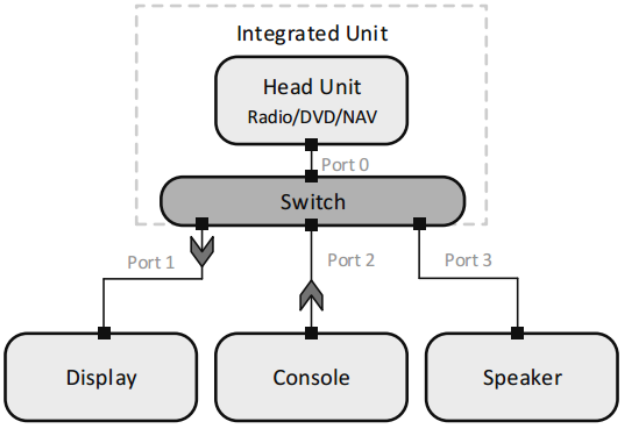


Figure 2.9: The transaction of packets by Ethernet switch [28]

Comparison of In-vehcile network

Table 2.1 gives an overview of typical networks used in today's vehicles. The CAN network has high flexibility, compared with Ethernet, multi-master mechanism enables ECUs to be added to the CAN network easily without requiring a port for each switch. However, CAN has low performance due to limited "short" messages (max message size is 8 bytes) and low maximum speed.

Although MOST has long been regarded a wise option for infotainment systems, it is on the way out due to less flexibility and expensive cost of optical fiber. The ring topology of MOST leads to disconnected connection if a problem found in one node. Volvo and Geely, for example, have reduced the number of MOST bus for modern vehicles.

The main advantage of FlexRay is that it has built-in redundancy by two channels while Ethernet needs to add additional switch path (extra cost) to achieve the same performance. However, the drawback of FlexRay is low versatility. FlexRay lacks the bandwidth and protocols to support the purposes other than X-by-wire and safety-critical applications.

Ethernet is well suited to the rising number of advanced infotainment applications

due to its high bandwidth. Moreover, the configuration is easy to change based on different requirements. Nevertheless, the restriction that every node must rendezvous at switch-point makes it less flexible and incurs significant costs for the added switches for a complex network configuration.

From the comparison, it can be concluded that the benefit of Ethernet in future automotive electronic market outweighs other automotive networks. The lack of bandwidth network CAN can be solved by Ethernet, which makes it more adaptable to ever-increasing functionality and performance controlled by ECUs in future vehicles. Furthermore, Ethernet enables emerging driving-related features in a car, for example, autonomous driving with a real-time video camera and ADAS. The broadened applications supported by Ethernet become an attraction for consumers and a battleground for car manufactures as well. However, it is not easy to adopt a new networking technology. Transitioning Ethernet into vehicles means the test method should also be changed accordingly. The existing data analysis tools used among different networks cannot meet the requirement of Ethernet. The special physical and IP protocol layer of Ethernet call for new test tools for verifying the correct integration between Ethernet and other protocols.

Table 2.1: Comparison of in-vehicle network

Parameter	**CAN**	**MOST**	**FlexRay**	**Ethernet**
Medium Access Control	Multi-master	Timing-master	Multi-master	Master-slave
Execution Control	Event-triggered	/	- Time-triggered - Event-triggered	Time-triggered
Redundant Channel	Not supported	Not supported	Two channels	Not supported
Typical Physical Layer	Twisted dual wire	Optical fiber	Twisted pair of POF	Optical fiber
Typical speed	Up to 1Mbit/s	25 to 150 Mbit/s	10 Mbit/s	1 Mbit/s to 100 Gbit/s
Application examples	- Driving assistance - Powertrain	- Infotainment - Navigation	- X-by-wire - Chassis (Active safety)	- Car sensor system - Infotainment system
Access scheduling	- CSMA/CD	CSMA/CA	- TDMA (static) - FTDMA (dynamic)	CSMA/CD
Topology	Bus	Ring/Star	- Bus/Star/Hybird	Point-to-point/Star/Bus
Error detection	- 16-bit CRC - Error counter - bus-off state schemes	- CRC - Plug & Play feature	- 24-bit CRC - Bus guardians	- CRC - Bit error detection
Transfer Mode	Asynchronous	- Asynchronous - Synchronous	- Asynchronous - Synchronous	Synchronous

2.1.4 Gateway ECU

For satisfying heterogeneous networks used for in-vehicle applications, a gateway ECU is necessary to be implemented for these protocols communication. The automotive gateway is an ECU for linking different communication interfaces and it realizes the control of information exchange of one or multiple protocols.

Gateways are similar to routers but have more complex configuration. A gateway can be applied on the networks with more than one protocol technology. The data frame format of one protocol needs to be translated to another protocol before it can be read. For network using the same protocol, the role of gateway is to translate traffic between multiple buses of the same protocol, for example, whether the transfer speed is too fast or there is traffic congestion on the bus.

Figure 2.10 is an example of an automotive gateway. Both high and low speed CAN, FlexRay, LIN and MOST can be connected together by a central gateway. All the protocol translation is performed by only one gateway ECU, thus this architecture has low fault-tolerance, which means the communication will be off if the central gateway fails. FlexRay or Ethernet backbone gateway can be used to share the load of central gateway. Figure 2.11 shows a combination of ECUs and network mentioned in this chapter.

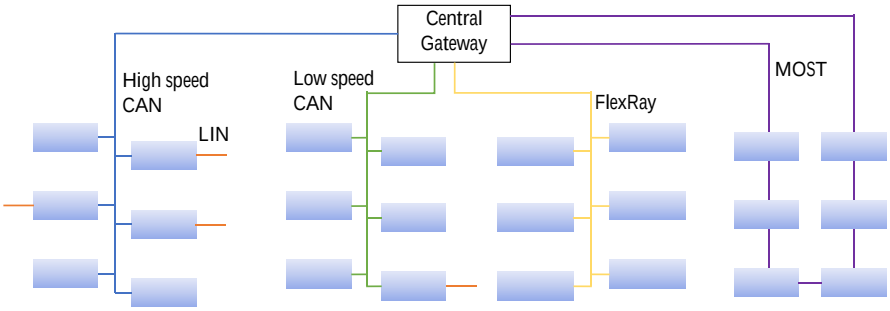


Figure 2.10: Central gateway layout

The basic function of gateway includes diagnostics, routing and network management etc. The detailed description of main function is presented as below:

- Diagnostic tester: A gateway can be equipped with a self-diagnostic connector to detect any error occurs during the data transmission. For example, if the message sent is received correctly by the target node. This functionality can also prevent the error propagate through the networks.

- Message Routing: The message routing ensures the message goes to the correct network bus. For example, one engineer determines the message path coming from MOST bus by certain algorithm to the CAN bus with frame identifier 4.

- Packet Routing: The gateway sends the data packet to the destination node by a routing table. The routing table mainly contains network ID, the desired

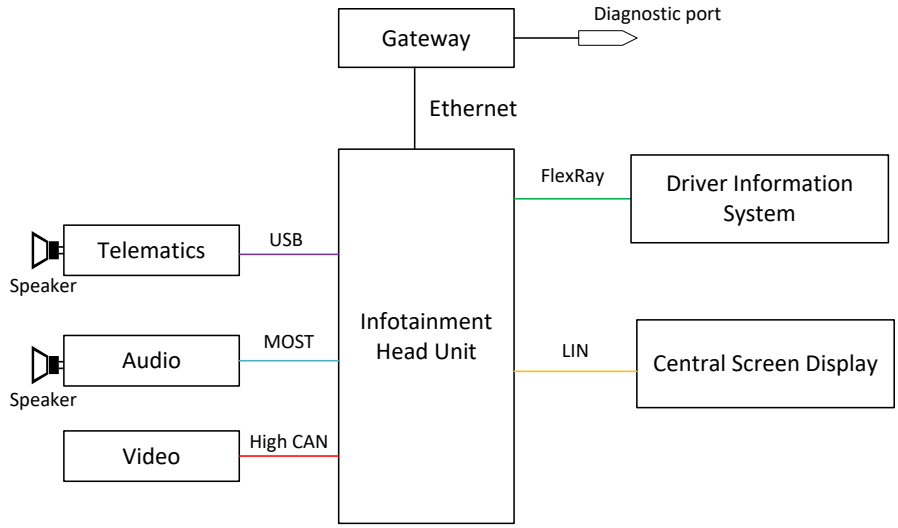


Figure 2.11: A typical infotainment ECUs and network topology

address and cost. Thus the gateway can have the record and keep track of how the data packets are transferred [29].

- Network Management: This enables to handle send or receive requests that appear on the network.

- Message Translator: The most important function of a gateway is the realization of translating message between two different protocols [30]. By this feature, the gateway can divide long message into several small sections based on the requirement of protocols.

2.1.5 ECU Diagnosis

In the infotainment system, it is essential to ensure the status of different ECU modules whether they work as expected and prevent any malfunction in future [31]. Diagnosis function in a vehicle refers to a vehicle's self-diagnostic capability to identify faults and initiate appropriate countermeasures as well as record the fault. The record offers maintenance with access to the status. There are three diagnosis procedure in a vehicle [32]:

1. The current vehicle behavior is compared with the expected performance. When there is any deviation and abnormal conditions detected, the observed discrepancies are noted as symptoms.

2. Analyze underlying fault based on the symptoms.

3. If the fault is defined, the driver should be informed with an alert. (e.g. warnings shown on the central panel)

Diagnostic Protocol

International Standard ISO 14229, unified diagnostic service (UDS) protocol has been established on the fifth and seventh layers of open systems interconnection (OSI) model for requirements of diagnostic services. This model allows a diagnostic tester (client) to control diagnostic functions in an on-vehicle ECU (server) [33]. The action from ECU is represented by service identifiers (SID) and Sub-function ID. The diagnostic protocol uses "request-respons" model for checking. As shown in Figure 2.12 , the request is composed of SID, sub-function and request parameter. The response is different to the request with and without sub-function, the detailed requirement can be found in protocol ISO14229-1.

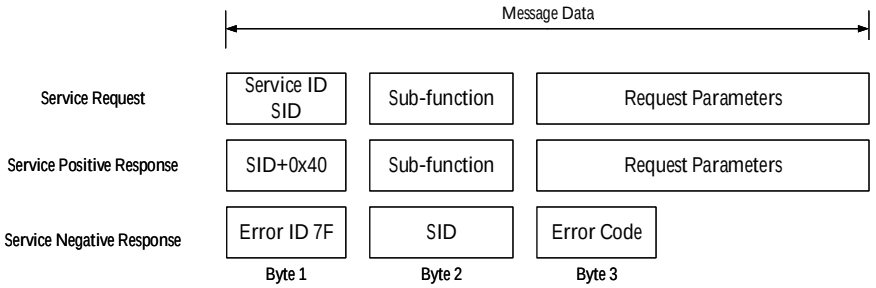


Figure 2.12: UDS Diagnostic Protocol [34]

2.2 Test Automation

Test automation is a way of using software to manage the test execution [35]. The process mainly includes: test automation feasibility analysis, appropriate tool selection, automation framework development, test scripts building, test execution and result analysis [36]. Test automation requires less human resources and results in higher efficiency and resource-savings compared with manual test. By taking advantage of test tools and framework, testers can complete execution of basic test cases in a short time and thus realize integration test as early as possible. However, the decision should be made whether to carry tests automatically or manually for each test suite. Generally, maintenance effort and upfront costs (test tools, environment setting) outweigh the other features when making a choice. Quality and time spent are also considered as important factors.

2.2.1 Scope of test automation

Some prerequisites need to be considered for making decision of taking test automation. First, the change of software should not be too frequent. The stability of test script influences the expenditure of automated test maintenance, which can be regarded as a new code development. In this case, modification, debug and test framework improvement are required during maintenance, which cost a lot of human resource and money. Therefore, the investment is higher than what can be

saved from automated test. Second, the project cycle should be long enough. The time spent on determination of test requirements, test framework development, test script building and debug ought to be guaranteed during the project development. Generally, the benefit of test automation will not be observed immediately. Third, test scripts ought to be re-utilized. Low rate usage of test scripts will lead to huge cost put on script development becoming wasted, and thus losing the meaning of test automation. Flexibility and compatibility should be taken into account when developing test scripts. Fourth, tests with low cost-benefit should not be automated. The following guidance summarizes the typical test types suitable for automated test and manual test.

Tests suitable for automated test

- Regression tests: To ensure that a recent modification in the system does not have adverse affection on existing behaviors, the regression testing is performed. Test automation is a perfect solution for regression tests since the same testing process is needed until a fault is fixed [37]. Due to the fact that the test cases of regression test and expected result are decided in advance, the efficiency of conducting this type of test is enhanced by reducing execution time and labor cost.

- Continuous integration tests: During an agile project development, feedback associated with integration errors should be given after each test building. There are usually multiple integration events carried out daily since different engineers have different projects to integrate each day. By using automated tests along the project iteration, the test time frame is decreased and testers can get results after each new commit. Moreover, running test automation for each integration makes less impact on overall systems when there is a serious issue to fix. This is because that fast execution of test automation gives immediate result for testers, which prevent to solve all problems at the end of the project. By test automation, continuous integration tests can also be executed during night, which gives more time for testers to analyze the issue during day time.

- Monkey tests: Random inputs are used during the testing and testers check the performance of system accordingly. Test automation is deemed as profitable way for stochastic input data and enormous steps in these tests.

- API based tests: Application Program Interface (API) specifies the interaction methods among software components. A range of requests and extreme inputs are utilized in testing to verify if the responds from software is correct. Test automation is recognized as suitable for API based tests. Unit testing and functional testing are involved in API based tests.

Tests suitable for manual test

- Maintenance, installation and setup tests: Usually these tests require human intervention when re-configuring system architecture and installing software/hardware is need.

- Localization tests: If the test target related with specific language or culture, only a specialist can judge whether the translation is reasonable and without culture biased.

Although at first glance there are more areas where automated test can be employed, manual test still plays an important role when tests require human sense, thinking and knowledge. Moreover the goal of having tests is to improve the quality of product. Automated test only verifies the relation between test result and expected outcome without showing how to enhance accuracy and project quality. Even the test result is the same as anticipated behavior, test cases can be improved by imagination and creativity of human and thus need further verification. Other tests need high maintenance cost and have low utilization frequency should also be performed manually.

2.2.2 Advantage and Disadvantage of Automated Test

From literature review, pros and cons of automated testing have been found. In the first place, the main merit of test automation is high speed of test execution, which in turn saves time to run tests. Conducting tests by a computer is much faster than by a tester, which means the number of tests performed by automation is more than a manual test in the same period of time. Automated test can be applied to tests which need to run twenty-four hours or over weekend without tester's operation [38]. On the other hand, the reliability has not been reduced by the fast speed of automated test. For instance, a tester can hardly find the difference of responding time of 0.3s and 0.5s. However, if carried out by software, every nuance will not be missed and finally captured by computers. Third, the repeatability of test execution is another advantage of test automation. Since the procedure and content of each test execution are the same, testers do not need to worry about human error, forgetting action steps and other negligence during the execution. The quality of testing process is therefore increased and manpower can be utilized more effectively. At last, reusable automated scripts also make automated testing productive. Good scripts with compatibility can be used in different projects for the same purpose.

However, automated test has limitation in tests which only need one-time effort and involve human thought, which means manual testing cannot fully replaced by automated testing [38]. Another drawback is enormous cost of buying tools and investment of how test automation can be done at early stage. Third, due to the fact that the performance of test automation relies largely on the quality of scripts, there will be higher technical requirement on engineers to develop test cases and framework [39]. The education time for new testers will also add cost to automated test.

2.2.3 Test Automation Framework

Test automation framework is an integration of test design thinking and test method that sets the rules of automation for a specific system under test (SUT). The framework combines test object, test libraries and test cases into an architecture, where the communication is controlled by arguments transmission. Different open-source test automation frameworks are available for different purposes. For example, RedwoodHQ and Sahi are frameworks target for web application tests. Another framework called Citrus is applied to communication protocol testing such as HTTP.

The common test automation frameworks can be classified into record and playback, data-driven and keyword-driven in terms of approach. There is also a new technology of testing software which relies on model-based testing (MBT) framework. The right choice of test automation framework should be based on reusability, maintainability, extensibility, repeatability and stability. Moreover, the framework should also be easy to understand by non-testers such as customers and business stake holders. An effective test framework results in a success of test automation, otherwise it can cause deviation of test objective.

Common Test Framework

Record and playback is the first generation of linear test automation framework which is based on the concept of simulation. It captures the users' action on PC and replay it. The main problem is that it is difficult to have any change on the system due to strong dependency on system environment. Therefore the framework is hard to maintain because of large amount of separate test scripts and non-reusable modules. The test execution cannot be iterative.

Data-driven testing (DDT) framework is also known as "table driven" type of test. The tests are executed in terms of data tables, which provide test input and output values from data files. The test table is then loaded into variables in the driver test scripts. DDT allows the same test to be executed multiple times with different dataset. The creation of test cases is no longer dependent on systems as well as becomes more flexible to fix bugs. The conception of DDT is shown in Figure 2.13. However, new driver test scripts are required for new kinds of tests to be understood. That means adaption is needed when either changing driver test scripts or introducing new test data files.

Keyword-driven testing framework is an extension of DDT by increasing reusability and maintainability of the framework. The biggest limitation of DDT is removed by feeding both test data and directives into a driver script. The directives, i.e. keywords or action words, direct how to execute the test data parsed from test scripts [40]. The keywords can be created by Python, Java, .Net or through own scripting language. The driver script interprets these keywords and execute test by assigned arguments [40]. Figure 2.14 shows the architecture of keyword-driven framework. However testers need to deal with more complicated frameworks and more complex test cases due to the increased flexibility, which is time consuming as

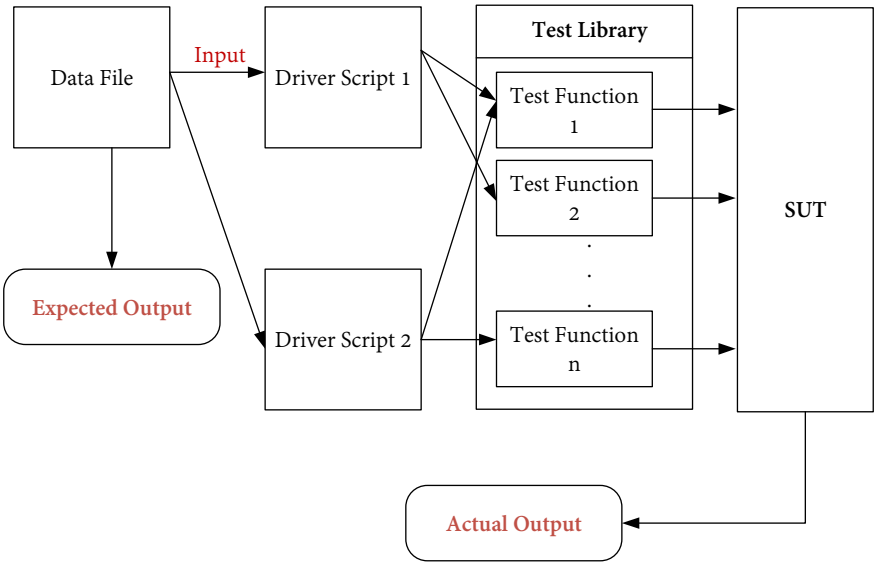


Figure 2.13: Data-driven testing framework

well as needs great effort for development.

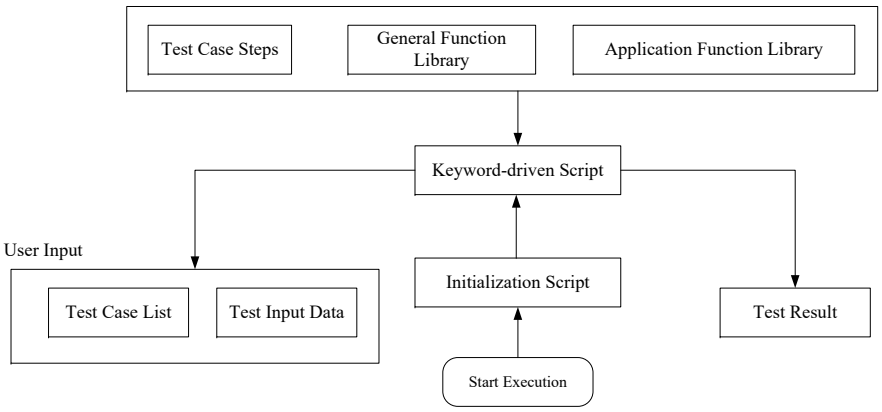


Figure 2.14: Keyword-driven testing framework

Compared to script-based test automation framework, a new approach MBT comes to the stage in recent years. The working flow of MBT is shown in Figure 2.15. The core of this testing framework is the model of system behavior illustration, which is done by manual. In order to make sure that the system description is consistent with machine-level model, a feedback from model to original requirements is needed. The next step is to generate test suites containing test sequences and test oracle, which are also regarded as input and expected outputs. The test sequences are used for controlling the execution steps of SUT while test oracle observes the results from the implementation. Afterwards, the results obtained by test oracle will be compared with the expected outputs and give a pass or fail conclusion. The failure

information is sent back to implementation stage as well as system model and initial system requirements stage to find reason of failure.

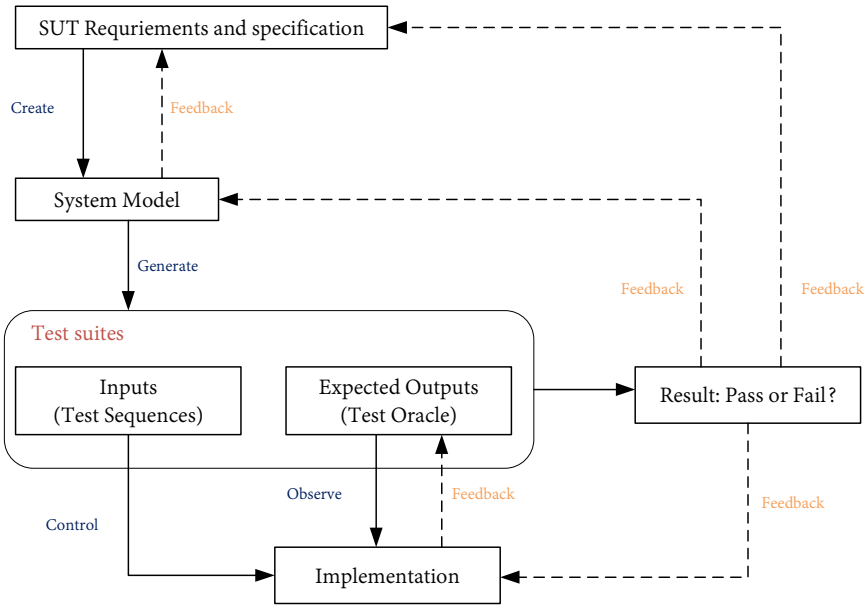


Figure 2.15: Model-based testing conception

In conclusion, a testing framework is an independent system from application under test (AUT). With the help of libraries, common tools, assumptions in the framework, make it easier for the testers to develop test cases. Supposing that each application has a new automation environment, the testing framework not only allows testers to expand specification from one application to another, but also avoids duplicated test cases among different applications.

The Robot Framework

Based on literature review, a keyword-driven testing framework called Robot Framework is chosen. It is used for acceptance level testing [41], which is also known as acceptance test-driven development (ATDD). The aim of acceptance testing is to evaluate the system's performance and check if it is available to be delivered according to the business requirements. The Robot Framework principle is based on table-driven keywords, which define each step of execution in advance. Figure 2.16 shows the architecture of Robot Framework. Test data layer is presented in tabular format as a dictation of test execution. Different tabular syntax can be used depends on the context, such as plain text, tab-seprated values (TSV) and hypertext markup language (HTML). Plain text format is chosen in this project since it is easy to edit by editors. Test data contains settings, variables, test cases and keywords. Figure 2.17 is an example of test data file. Table 2.2 lists functionality of each element in test data. The test data will be processed by the framework when it starts.

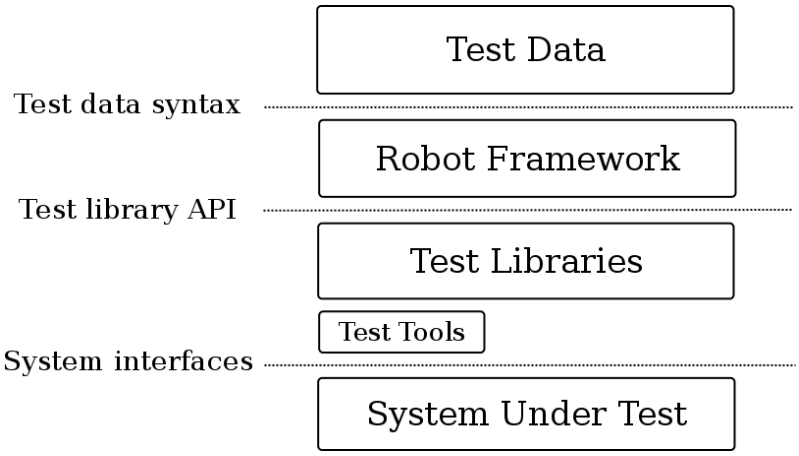


Figure 2.16: Robot Framework architecture [42]

```
***  Settings  ***
Library                  Selenium2Library
*** Variables ***
${URL}                   URL                        # URL of page
${USERNAME}              USERNAME                   # Username for login
${PW}                    PASSWORD                   # Password for login
*** Test Cases ***
Login to the page
*** Keywords***
Login to the page
       Put Username      id=loginusername           ${USERNAME}
       Put Password      Id=loginpassword           ${PW}
```

Figure 2.17: Robot Framework test data file

Table 2.2: Robot Framework test data elements

Table element	Functionality
Settings	1) Importing test libraries, resource files 2) Managing test setup
Variables	Defining argument described in test case and keywords
Test Case	Generating test cases
Keywords	Creating user keywords based on existing keywords

Test libraries work as a communication bridge between the framework and application under test AUT. The interaction is handled by test library keywords from both standard and external test libraries supported by Python or Java. Standard libraries contain methods of dealing with operating system, string manipulation and verification, telnet communication etc. [43]. In this way the framework can access system under test without knowing the details, which makes it easier for non-testers to understand tests. The communication can either be direct or indirect access by test tools. Robot Framework provides four built-in tools, i.e., Rebot, Testdoc, Libdoc and Tidy to ease building tests.

Features of the Robot Framework

- Easy-to-use: Test cases can be created in the same style due to tabular syntax.
- Reusability: High-level keywords can be derived from existing keywords [43].
- Independence: The Robot Platform is separated from the test application.
- Automatic report generation: The result and log are generated automatically and provided in HTML format.
- High adaptability: The Robot Framework is suitable not only for acceptance level testing but also cater to web testing, GUI testing, Telnet etc.
- Resourceful built-in function: Variables for testing different environments and a simple library of API are offered [43].

2.3 Chapter Summary

This chapter is divided into two sections which are the foundation of this thesis project.

First, automotive electronics especially automotive infotainment systems were introduced. For supplementing the understanding of IVI system, different in-vehicle networks were discussed in detail. The comparison between these networks was presented in this chapter as well. Second, the knowledge of test automation was provided, which laid the background of the usage of test automation and test framework.

3
Methodology

This chapter introduces the methodology used for conducting research. The project begins in a planning stage and followed by a literature study phase, which give sufficient background knowledge within infotainment-related areas. A detailed description of research method and study method is explained in the following.

3.1 Research Questions

In the initial step of a research project, having clear and well organized research questions within a reasonable scope will facilitate the subsequent study. The research questions determine the area will be looked into and identify the specific inquiry or objectives the project will answer.

Since the existing studies and surveys on testing methods used in automotive industry are limited, the following research questions are essential for the future test automation and ÅF business:

- *Why is implementing test automation for automotive infotainment systems difficult?*

  The answer of this question could be helpful and interesting for the engineers in ÅF. By analyzing the challenges behind automated testing in vehicle infotainment system, the company can get enough information to decide whether or not to expand their business in the market of interest.

- *How to decide whether to automate a test or run it manually within the considered systems?*

  Due to the wide coverage of in-vehicle systems, some tests require a test automation method to enhance the efficiency while others need manual intervention. It is a perennial challenge to find a balance between these two approaches and to decide when to focus on test automation. A standard including cost-benefit analysis and quality assurance need to be made during the project to solve this question.

3.2 Research Method

Research method shows the way of getting and analyzing collected information and data during the project. The procedure of completing the thesis and methods used are summarized as below:

- Planning. The initial planning stage determines the thesis research question, limitations, motivation and timing plan.
- Literature study (Prestudy). Gathering information about automotive infotainment systems including architecture, in-vehicle network and test automation workflow etc. The outcoming from online literature studies and previous publications lay solid foundation for solving tasks afterwards.
- Down-scoping. Investigating and determining test areas that benefit from automation. It is impossible to automate all tests since some are hard to automate and may require manual attention. The suitability of test automation depends on several factors such as system architecture, how many times the given tests will be performed, the cost and quality of automation etc.
- Analysis. Discussing the chosen automation method and determined automated areas (Diagnosis) with ÅF engineers in more details. For instance, the way of how automation will be done with regard to cost, timing and quality.
- Implementation. Developing an automated test based on Python script which will be applied in further development. The test results will be compared with expected behavior of the application. The performance (e.g. time-, cost- efficiency) will be taken into account.
- Evaluation. The study results is continuously held along with the thesis project to make sure the requirements are met.

3.3 Tools and resources

In the first theoretical part of thesis project, no special tools were needed. However, the author had continuous supervision and feedback from engineers at infotainment team in ÅF. In the second implementation part, one laptop with access to Robot Framework, Gerrit and Jenkins was provided.

4

Diagnosis Test Automation and Automation Cost

In this chapter, the implementation of the first goal in this thesis is introduced. Infotainment ECUs diagnosis testing is selected for test automation based on motivation and feasibility. A framework of how diagnosis function test should be done is illustrated from a theoretical perspective. During the first phase of thesis project, the automated test estimation especially in terms of money is also one of the concerns. Based on literature review, a revised way of calculating test automation benefit is provided in this chapter as well.

4.1 Motivation of Diagnostic Automated Testing

As mentioned in the introduction chapter, electronic functions such as navigation, Bluetooth, audio/video are booming in today's IVI systems. Increasing functionality of each ECU on account of customer entertainment and comfort needs is one of the challenges within automobile industry [44]. It naturally becomes of the essence to report malfunctional communication inside the car and give drivers warning indication. The exceptions that occur due to incorrect data transmission should be analyzed and captured by self-diagnostic function in a vehicle. A popular way of ECU diagnosis is through on-board diagnostics (OBD). It not only reports the vehicle problems to drivers but also stores the trouble code and gives access to technician later for repairing the error subsystems. The execution of OBD relies on a digital connector to transfer the ECU's message together with standard diagnostic trouble codes (DTC), which is also known as fault code [45]. By reading these DTCs the engineers are able to identify the particular ECU problem.

In order to make ECUs implemented with correct diagnosis function, testers must test diagnosis services during the product development. However, current in-car diagnosis testing technology has technical bottlenecks in several aspects. For example, different diagnosis protocols and data formats among various ECUs; test coverage of diagnosis testing; low efficiency when testing large amount of ECUs etc. Hence, automated diagnostic testing facilitates the improvement of diagnosis features among on-vehicle ECUs.

4.2 Infotainment ECUs Diagnosis Test

The diagnosis test of infotainment ECUs mainly contains two sections: diagnosis protocol test and diagnosis functionality test. In this project, the research focus is put on diagnosis functionality.

4.2.1 Diagnostic Protocol Test

Diagnostic protocol is a formalized standard to regulate ECU diagnosis services, such as what should be sent to the ECU when reading DTC or data message. The diagnostic protocol for CAN bus is called UDS. The purpose of ECUs diagnosis protocol test is to verify if the protocol communicating mechanism operates correctly and reliably according to the ISO standard. The test mainly relies on CANdelaStudio, CANoe.Diva and CANoe.

The first step of ECU diagnostic protocol test is gathering ECU diagnosis specification such as ECU configuration data or DTC [46]. The data is sent to CANdelaStudio, which is a tool provided by Vector for editing ECU diagnostic description, to generate .cdd files. The output files are added together to become a diagnosis database. CANoe.Diva is an extension of CANoe which can support different in-vehicle networks and is used for generating test cases based on a diagnosis database. CANoe is used for generating testing environment. Testers can select wanted test cases in CANoe as well as determine test flow. In the final step of a protocol test, a report is produced automatically with analysis and comparison between expected result and actual outcome. The test flow is shown in Figure 4.1.

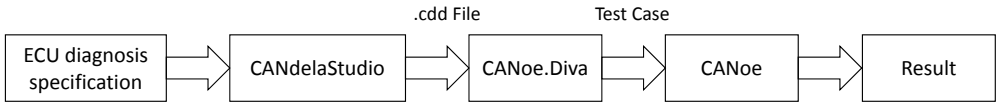


Figure 4.1: Diagnosis protocol automation test flow

4.2.2 Diagnosis Function Test

In contrast to ECU diagnosis protocol test, diagnosis function test needs special stimulus of input and output from ECUs. In order to validate the performance of ECU diagnosis function under erroneous conditions, testers need to insert errors on ECUs in advance. The fault injection was usually done manually in the early days. However, this approach has poor repeatability and low test coverage when it comes to complicated setting procedure. Moreover, it has high possibility to destroy ECUs due to misoperation during the test. Thus having automated testing for ECU diagnosis function is necessary to enhance the test accuracy, efficiency and stability with the help of software.

Based on the characteristics of diagnosis service, four basic categories of testing

(test cases) can be defined as below, however, it is important to mention that ECU function test cases vary from ECUs to ECUs.

Test cases for diagnosis function test

- Input/output DTC test: The test case is used to evaluate if ECUs could set correct DTC when there is a short circuit to power supply or ground between input/output node of ECUs. Alternatively, if DTC will be set when there is an open circuit or other damaged electrical wiring. The test is also intended to check whether the function recovers according to the ECU specification after faults have been solved.

- Abnormal-state DTC set test: The purpose of this test is to inspect whether ECUs can set DTC as required in diagnosis strategy under abnormal working operation (e.g. too low voltage).

- State reading test: The aim of examining ECU state reading is to check that ECU function status records is the same as ECU current working condition.

- Input/output control test: This type of test is to detect if the input/output of ECU can be enabled or disabled by diagnostic system based on the diagnosis standard.

Framework of diagnosis function test

For investigating diagnosis function in ECUs, it is essential to have similar working environment for the ECUs and their peripherals. The working environment setup includes power supply, stimulation, load and fault simulation. After that, fault injection is applied to the inputs and outputs of ECUs.

Figure 4.2 shows a conceptual framework for the diagnosis functional test based on the four test cases above. The main facilities are: The PC, VT system created by the company *Vector*, CANcase XL, programmable power supply and direct current (DC) power supply. PC works as a control interface including both hardware and software applications such as CANoe. ECUs under test can either be virtual or real by editing configuration in CANoe. Vector system consists of different modules which simulate diverse digital and analogue inputs/outputs. The transmission and generation of CAN messages can be done through CANCase XL who has two CAN controllers. The CAN message can use either 11 bit or 29 bit identifiers [47].

Modules of Vector system selected for the test

- VT8012A: Backplane. Supplying voltage of 12V for at most twelve modules on VT system and communicating with computer. The communication is sup-

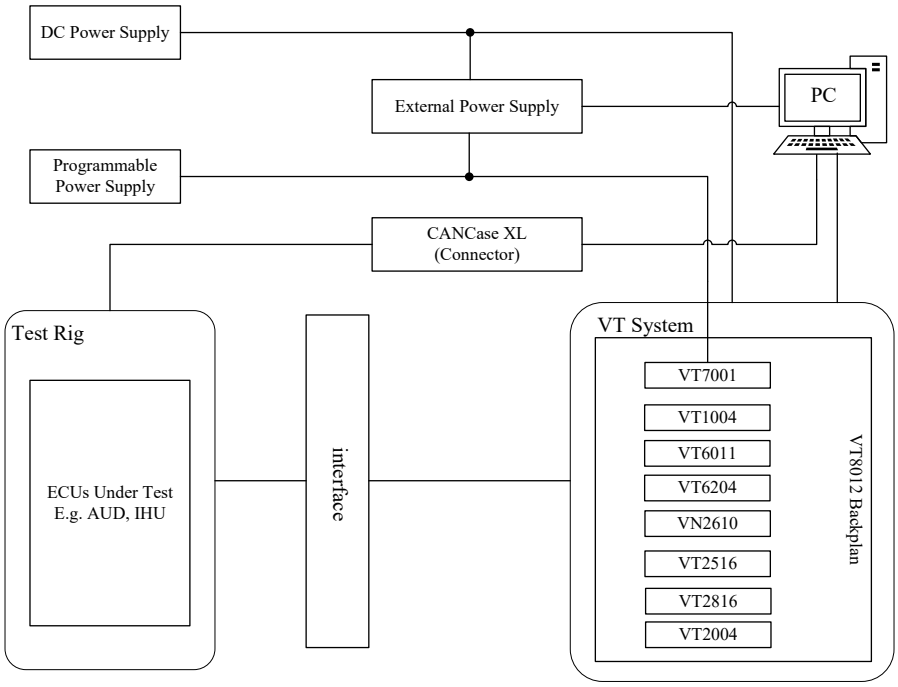


Figure 4.2: Architecture of a test automation for diagnostic function test.

ported by Ethernet interface on backplane. Another option of backplane is VT8006A, which can support at most six Vector VT modules.

- VT7001: Power Supply Module. It has two output sections used for up to two ECUs with different voltage levels. The supporting voltage for ECU under test comes from internal power supply by VT7001. VT7001 also can be used for current measurement.

- VT1004: Load and Measurement Module. The module simulates loads to ECUs and measures the output voltage of ECUs. It has four channels which can generate short circuits to ground or battery voltage [48].

- VT2004: Simulation Module. The utility of this module is to simulate resistance and short circuit stimulation.

- VT6104/VT6204: Network Module. Both modules provide common independent network interface, i.e. CAN, LIN for ECUs while VT6204 supports FlexRay additionally.

- VN2610: MOST Interface Module. The module is specially for connecting MOST bus from ECUs to VT system.

- VT6011: Real-time Module. The module has two USB2.0 ports which can support interface of module VN2610. VT6011 is regarded as a PC module on VT system for real-time execution of CANoe [48]. The module is connected

with PC via Ethernet port.

- VT2816: General Purpose Analog I/O Module. The module provides 12 input channels for voltage measurement and 4 output channels. Eight out of twelve channels can also be used for current measurement by shunt.

- VT2516: Digital Module. The modules involves 16 channels, which can simulate digitally used I/Os of ECUs, short circuit between input and ground or power supply.

Similar with the choice of test cases, alternative VT system modules can be selected other than what has been mentioned above according to certain ECU diagnosis function test.

Detailed implementation of test framework

For the test cases discussed before, the detailed implementation of VT system is:

- Input/output DTC test: The module VT2004 can be used as an analog input while VT2516 can be utilized as an input for digital signal. Module VT1004 is used for output load and can connect four outputs of ECUs at most. When faults are injected on Vector system either to input signal or output load, DTC should be set according to the error type. Otherwise, the diagnosis test fails.

- Abnormal-state DTC set test: Generally, the test modules are the same as the first test except that programmable power supply is needed for changing voltage level of VT7001. The successful tests should have right DTC if the power is lower than normal.

- State reading test: As the same for input/output DTC test, input signals should be simulated for detecting the current of data. Moreover, the CAN message needs to be generated by CANcase XL if the output is triggered by CAN network signal. The analog input is emulated from VT2004 and VT2516 for digital input. Both VT2816 and VT2516 modules can be used as an output of and ECU.

- Input/output control test: VT1004 or VT2516 can be used for simulation of ECUs output load.

Procedure of diagnosis functional test

1. Choosing suitable VT system modules and connect each module with input/output of ECUs.

2. Writing diagnostic test cases in CANoe using communication access program-

ming language (CAPL)based on predefined test steps.

3. Making configuration in CANoe by selecting real or virtual ECUs.

4. Running test cases in CANoe and generating testing report.

5. Analyzing report from step 3 and finding fault reasons. In this step, revision could be done for test cases and test environment.

4.3 Test Automation Cost

In the previous chapters, the advantages and necessity of having test automation are discussed. However, the automation process should take costs and risks into account. The usual way of evaluating the automated test value is by calculating return on Investment (ROI). ROI is a mathematical way of identifying how much benefit we can get from investing in an automated test. By understanding ROI factor, testers can evaluate the value of the project and adjust test plan to achieve higher gain. It has been used widely at early planning stage for test automation. Utilizing ROI before starting test automation testers can judge how it pays off as well as to decide the percentage of areas for which to employ automated test. The basic principle is to divide benefit by investment cost. In test automation, the benefit comes from both tangible and intangible factors. The cost of tangible factors such as labor cost and equipment purchasing cost are easy to calculate. However, there is no agreement on how to calculate the intangible benefit and expenditure since many elements are not quantifiable and oversimplified. An inaccurate calculation leads to improper implementation of test automation and thus causes huge losses ranging from individual to the whole project.

4.3.1 Previous Work on ROI Calculation

From literature study, we find that there are some previous work related to ROI calculation. In the following part, two representative calculation methods are analyzed in detail to show that earlier work on ROI calculation is not perfect.

In his “Test Automation ROI” paper, Dion Johnson proposed three ways of calculating ROI [49]. In “Simple ROI”, the calculations are considered in terms of monetary savings. It takes fixed costs such as tools, training, machines into account and converts the time factor of automated testing into the form of money. The merit of “Simple ROI” calculation is that it makes the project investment more intuitive to the upper level management. However, this method assumes that automated test can completely replaced manual tests, which oversimplifies the cases when tests are done with a combination of manual and automated test. In contrast, “Efficiency ROI” only considers time investment and calculates the benefit from that to assess test efficiency. The way of calculating is only suitable when test tools are used long enough to be neglected. “Efficiency ROI” allows testers to estimate the project ‘budget’ and present benefits of doing test automation in terms of days. Nevertheless,

the method is based on the assumption that full regression can be performed during test cycles even without test automation, which is rarely true. At last, Dion introduced a "Risk Reduction ROI" which assesses the risk of not having test automation and calculates the loss caused by the risk. For example, doing automated test saves a huge amount of time for execution while providing more time to do analysis and test framework development. This could help to increase the test coverage and thus reduces the risk of project failures. One disadvantage of this model is that it is hard for testers to estimate how much money will be lost. Moreover, the lack of comparison between manual and automated test cannot answer of the reason for choosing automated test instead of manual test.

$$E_n = \frac{A_a}{A_m} = \frac{(V_a + n \times D_a)}{(V_m + n \times D_m)} \tag{4.1}$$

$$E_{n'} = \frac{A_a}{A_m} = \frac{(V_a + n_1 \times D_a)}{(V_m + n_2 \times D_m)} \tag{4.2}$$

$$ROI_{automation} = \frac{B_a}{C_a} \tag{4.3}$$

$$ROI_{automation'} = \frac{\triangle B_a}{\triangle C_a} \tag{4.4}$$

Douglas Hoffman once introduced four equations for calculating ROI, which are listed in equations 4.1 to 4.4 [50]. However, each equation has several drawbacks so that it is not accurate enough to show the cost benefits from test automation. In equations 4.1 and 4.2, A_a stands for automated costs and A_m is manual costs. The expenditure for preparation before manual tests is expressed as V_m; V_a is noted as implementation cost for automated test. Analysis work after automated test is represented by D_a and D_m means the execution of manual tests. In equation 4.1 and 4.2, test automation is regarded suitable to implement when ratios are smaller than one. The difference between these two equations is the test time for automated and manual tests, which is the same in equation 4.1 while different in the other. Nevertheless, the assumption is often not applicable in real situation. Moreover, overhead spend such as hardware and software expenditure and maintenance cost for test automation are not included in any of these two equations. Equations 4.3 and 4.4 are calculated basically the same, which takes the ratio of benefit from automation over cost of automation. The improvement of equation 4.4 is using added benefit and added cost of automation over manual instead. The shortcoming of both equations lies in its practicality, which means it is hard to compute the benefit in figures absolutely. All four equations are too general to show what should be considered when calculating the cost-benefit. In addition, when comparing two ROI figures with the same number, the hasty decisions to automate both projects can be made without taking time into account. Therefore, the time factor should be included to make calculation more reliable.

4.3.2 New calculation model for ROI

In order to improve the accuracy of existing ROI computation model, a more advanced model is presented in this section. The basic idea behind the new ROI calculation is to take the ratio of gain and investment of test automation, which is shown in equation below:

$$ROI_{automation} = \frac{\text{Gain}}{\text{Investment}} = \frac{C_m - C_a}{C_a} \tag{4.5}$$

In the equation, $\boldsymbol{C_m}$ represents cost of manual testing and $\boldsymbol{C_a}$ represents cost of test automation. The unit of both factors is man-hour. Manual testing cost is composed of manual test cases creation cost, $\boldsymbol{C_{mcc}}$ and manual tests execution cost, $\boldsymbol{C_{me}}$, as described by equation 4.6. The overall execution cost consists of expenditure for the first-time testing and afterwards regression testing. The automated testing cost contains more elements:

$\boldsymbol{C_{hs}}$: It can be regarded as a fixed cost of hardware and software used for automated testing. Machine, tool license and acquisition costs are counted in this item. The unit is man-hour. The depreciation factor of C_{hs} can be noted as k.

$\boldsymbol{C_t}$: The notation is the cost spent on training. When the new test automation platform, tools, framework etc. are introduced, testers need to spend time on mastering these new technology. For an already-existed and familiar testing environment, this term can be ignored. The unit is man-hour.

$\boldsymbol{C_{afc}}$: It is the measurement of research, design and creation cost for the first-time test automation framework development. This item generally accounts for a bulk of expenditure during the initial stage of an automated test while it goes down in subsequent executions. The unit is man-hour.

$\boldsymbol{C_{acc}}$: The item stands for cost of creating test cases in automated testing. It depends on the number of automated test cases, $\boldsymbol{N_{ac}}$ and average cost of individual automated test case development cost $\boldsymbol{C_{acca}}$. During the test cases development, testers need to write test scripts and debug them. The unit for this term is man-hour.

$\boldsymbol{C_{ae}}$: This notation represents the overall execution cost for automated test. Generally, test automation is introduced after several iterations of manual testing and is applied from M round. Therefore, the execution cost of automated test is calculated from round M to the end of test round N rather than from the first test. The number of automated tests for calculating ROI can be expressed by $N - M + 1$.

After replacing these notations into $\boldsymbol{C_m}$ and $\boldsymbol{C_a}$, we get:

$$C_m = C_{mcc} + C_{me} \tag{4.6}$$

$$C_a = C_{hs} \cdot k + C_t + C_{afc} + C_{acc} + C_{ae} \tag{4.7}$$

The number of test cases varies for manual and automated test due to the capability of the two test methods. Before exploring detailed of equations, there are three assumptions for this calculation model [51]:

- First, the number of errors is exponentially suppressed as the test times increase. Suppose that there are 40 check points and error rate is 50%, in consequence there are 20 errors found and need to be fixed after the first test. For the second test, 10 errors (40*0.5*0.5) will be checked out of 20. After the third test, only 5 errors will remain. The test will go on until the errors are all fixed. If $\boldsymbol{d}$% is average error probability, then the number of fault is d%*N_p for $\boldsymbol{N_p}$ checkpoints. The number of error-prone points after i-times test automation is N_p*d^i, where i is the test order. The illustration is shown in Figure 4.3.

- Second, assuming that the same number of errors, d%*N_p, can be found either from a manual test or an automated test. This assumption holds even though the time spent on manual test is longer than that on automated test.

- Third, since analyzing after test execution is needed for both test ways, the average cost of analyzing a fault, $\boldsymbol{C_{epa\_sa}}$, after test is the same. Combining with previous assumptions, the overall erroneous problems (erroneous checkpoints) analysis cost, $\boldsymbol{C_{epa}}$, is the same for both test methods.

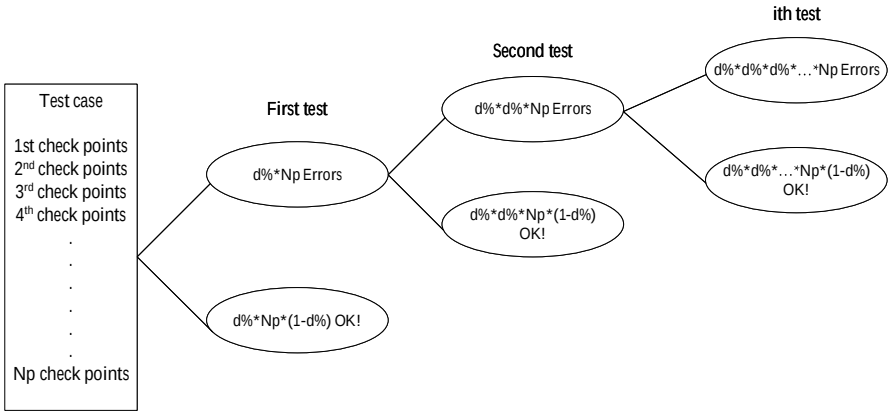


Figure 4.3: Error-prone check points after the i^{th} test iteration

Manual Test Cost

In equation 4.6, the manual test cases creation cost, $\boldsymbol{C_{mcc}}$, is determined by the number of test cases, $\boldsymbol{N_{mc}}$, and the average cost of a test case design $\boldsymbol{C_{mcca}}$. When it comes to manual test execution cost, $\boldsymbol{C_{me}}$, it depends on the number of tests, single-test execution cost, $\boldsymbol{C_{mce}}$, and single-test maintenance cost, $\boldsymbol{C_{msm}}$. For simplicity, problem analysis cost, $\boldsymbol{C_{epa}}$, is also counted in the term $\boldsymbol{C_{me}}$. It is calculated

by multiplying number of fault check-points with the average unit analysis cost, $\boldsymbol{C_{epa\_sa}}$. After each test, manual test cases need to be maintained due to requirement changes and modifications in the test. Thus, maintenance cost is the product of quantity of test cases, $\boldsymbol{N_{mc}}$, and the average single-test-case maintenance cost, $\boldsymbol{C_{msm\_sa}}$. Similarly, The single manual test execution cost is calculated by multiplying $\boldsymbol{N_{mc}}$ with the average cost of single-test-case execution cost, $\boldsymbol{C_{mse\_sa}}$.

As a result, the manual test cases creation expenditure $\boldsymbol{C_{mcc}}$ is:

$$\begin{aligned} C_{mcc} &= (\text{Number of test cases}) \cdot (\text{Average single test case design cost}) \\ &= N_{mc} \cdot C_{mcca} \end{aligned} \tag{4.8}$$

In order to compare with cost of automated test, manual test execution cost, $\boldsymbol{C_{me}}$, is calculated with the same period as automated test. It can be expressed as:

$$\begin{aligned} C_{me} &= (\text{Number of manual tests}) \cdot [(\text{Single manual test execution cost}) \\ &\quad + (\text{Single manual test maintenance cost})] + (\text{Problems analysis cost}) \\ &= (N - M + 1) \cdot (C_{mse} \ + \ C_{msm}) \ + \ C_{epa} \end{aligned} \tag{4.9}$$

After replacing equation 4.6 with equations 4.8 and 4.9, the cost of manual testing can be expressed as:

$$C_m = N_{mc} \cdot C_{mcca} + (N - M + 1) \cdot (C_{mse} \ + \ C_{msm}) \ + \ C_{epa} \tag{4.10}$$

Automated Test Cost

To expand the automated test cost in equation 4.7, detailed form of test case creation cost, $\boldsymbol{C_{acc}}$, and test cases execution cost, $\boldsymbol{C_{ae}}$ need to be investigated. The automated test cases creation can be calculated when average unit cost of test case design, $\boldsymbol{C_{acca}}$, and the number of test cases, $\boldsymbol{N_{ac}}$, are provided. The total cost of automated test cases execution is derived from five variables: the number of test cases, single-test automated framework maintenance cost, $\boldsymbol{C_{afsm}}$, single-test automated test cases maintenance cost, $\boldsymbol{C_{acsm}}$, single-test automated test cases execution cost, $\boldsymbol{C_{acse}}$ and overall problem analysis cost, $\boldsymbol{C_{epa}}$.

Therefore, automated test cases creation cost $\boldsymbol{C_{acc}}$ is represented as:

$$\begin{aligned} C_{acc} &= (\text{Number of automated test cases}) \cdot (\text{Average single test case design cost}) \\ &= N_{ac} \cdot C_{acca} \end{aligned} \tag{4.11}$$

and the expense of automated test execution starting from the Mth to the Nth test is indicated as:

$$\begin{aligned} C_{ae} &= (N - M + 1) \cdot (C_{afsm} + C_{acsm} + C_{acse}) + C_{epa} \\ &= (N - M + 1) \cdot [C_{afsm} + N_{ac} * C_{acsm\_sa} + N_{ac} * C_{acse\_sa}] + C_{epa} \end{aligned} \tag{4.12}$$

where $\boldsymbol{C_{acsm\_sa}}$ is average maintenance cost for individual automated test case and $\boldsymbol{C_{acse\_sa}}$ is average execution cost for each automated test case.

Final expression

The expansion of equation 4.5 considering the manual testing time factor, h_m, and automated testing time factor, h_a, is shown in equation 4.13, which can be regarded as an ultimate form of new ROI calculation described above. It shows the key factors which influence the test automation and manual test cost computation. The bigger the value of ROI is, the higher the benefit got from test automation. In order to make the value big, the nominator needs to be positive and as big as possible while the denominator should be as small as possible. The prerequisite is that all the cost associated with manual test cannot be increased, which is reasonable in industry. Table 4.1, 4.2 and 4.3 list the notations used in this calculation model.

$$\begin{aligned} ROI &= \frac{C_m - C_a}{C_a} \\ &= \frac{(C_{mcc}+C_{me})\cdot h_m - (C_{hs}\cdot k + C_t + C_{afc} + C_{acc} + C_{ae})\cdot h_a}{(C_{hs}\cdot k + C_t + C_{afc} + C_{acc} + C_{ae})\cdot h_a} \\ &= \frac{C_{mcc}\cdot h_m - (C_{hs}\cdot k + C_t + C_{afc} + C_{acc})\cdot h_a + (C_{me}\cdot h_m - C_{ae}\cdot h_a)}{(C_{hs}\cdot k + C_t + C_{afc} + C_{acc} + C_{ae})\cdot h_a} \\ &= \frac{[C_{mcc}\cdot h_m - (C_{hs}\cdot k + C_t + C_{afc} + C_{acc})\cdot h_a] + (N-M+1)\cdot[(C_{msm}+C_{mse}\cdot h_m - (C_{afsm}+C_{acsm}+C_{acse})\cdot h_a]}{(C_{hs}\cdot k + C_t + C_{afc} + C_{acc}) + (N-M+1)\cdot(C_{afsm}+C_{acsm}+C_{acse})\cdot h_a + N_p\cdot C_{epa\_sa}\cdot \frac{M\cdot d\%(1-d\%)(N-M+1))}{1-d\%}} \end{aligned} \tag{4.13}$$

General	
Term Notation	**Denoting**
C_m	Manual test overall cost
C_a	Automated test overall cost
N	Number of overall test times
M	Number of automated test starts point
N_p	Number of check points
d%	Error Rate
C_{epa_sa}	Average cost of single error-checkpoints analysis
C_{epa}	Total error-checkpoints analysis cost

Table 4.1: Summary of general notation for the model

Discussion

For enlarging nominator part, four elements related with test automation (i.e. C_{hs}, C_t, C_{afc}, C_{acc}) need to be decreased. That means, testers need to improve resources using efficiency and use familiar automated environment as much as possible to reduce the front-cost. In addition, automated framework creation cost can be reduced by avoiding using too complicated test framework. The test cases are supposed to be simple but effective to reduce the script debugging time. On the other hand, three contributors (i.e. C_{afsm}, C_{acsm}, C_{acse}) indicate that the development of test

Automated Test	
Term Notation	**Denoting**
N_{ac}	Number of automated test cases
C_{hs}	Hardware and software equipment cost
C_t	Automation training cost
C_{afc}	Automated test framework creation cost
C_{acc}	Automated test case creation cost
C_{acca}	Average cost of single test case creation cost
C_{ae}	Automated test execution cost
C_{afsm}	Automated test framework single-time maintenance cost
C_{acsm}	Automated test cases single-time maintenance cost
C_{acse}	Automated test case single-time execution cost
C_{acsm_sa}	Average cost of single test case maintenance cost
C_{acse_sa}	Average cost of single test execution maintenance cost

Table 4.2: Summary of automated test cost notation

Manual Test	
Term Notation	**Denoting**
N_{mc}	Number of manual test cases
C_{mcc}	Manual test case creation cost
C_{mcca}	Average cost of single manual test case creation cost
C_{me}	Manual test execution cost
C_{msm}	Manual test single-time maintenance cost
C_{msm_sa}	Average cost of single test-case maintenance cost
C_{mse}	Manual test case execution cost
C_{mse_sa}	Average cost of single-test-case execution cost

Table 4.3: Summary of manual test cost notation

framework and test cases should be flexible so that it is easy to change them when modifications are required. Executing test cases in batch can abate the expenditure spent on test cases execution in a single test. Moreover, increasing the number of automated tests $(N - M + 1)$ is another important way of increasing nominator. It can be concluded that if the execution number of test cases is low, it is not suggested to have automated test due to low ROI result.

For denominator part, the first two polynomials are reduced by the strategies mentioned above. Cost of analysis in the last term, shows that the automated test cannot be started too early. Too small value of M makes too much cost on the problem analysis.

For the reason that manual test takes longer time to finish the same amount of tests than test automation does, h_m is larger than h_a. Therefore, time elements result in even larger nominator and smaller denominator, which makes ROI fac-

tor bigger. Although it cannot be seen directly from the equation that automated testing runs more tests per day, since the time spent on the same number of tests is shorter by test automation, it can be inferred that automated testing outweighs manual testing in terms of quantity of test execution.

With the new ROI equation, although sometimes it is hard for testers to know each item unit cost, it benefits testers in several aspects. First, the equation shows that the testers can make trade-off between test automation and manual test running hours. Second, the manager can build a better budget plan and divided monthly budget more reasonably since the formula gives overall expected investment in terms of one-time cost and continuous input. In this way, the project manager is able to have rational resource distribution in different period of the project. ROI value reflects the task scheduling so that the test leader can give feedback or adjust on testing cycle and weigh the amount of time. In addition, ROI formula calculation tells the bottom line of getting benefit so engineers have more confidence about testing strategy.

5

Automated Test for an ECU

In this chapter, the implementation of an automated test of a vehicle ECU will be illustrated.

5.1 System Description

The electronic control unit going to be tested is integrated in a vehicle for communication and surveillance purpose. It connects to other systems in the vehicle through several electrical interfaces. The main feature provided by the electronic control unit is to route eight inputs to any of eight analog video outputs. The enclosure of the ECU, shown in Figure 5.1, consists of a milled aluminum box and an aluminum top cover. The unit is mainly composed of a computer module, DC/DC modules, internal cable harness, connectors for external power, video and data signals interface. The computer module contains a CPU module, video interfaces and data bus interfaces. The primary function of the computer module is to interface with and communicate data through different data buses as well as perform a video-multiplexer function.

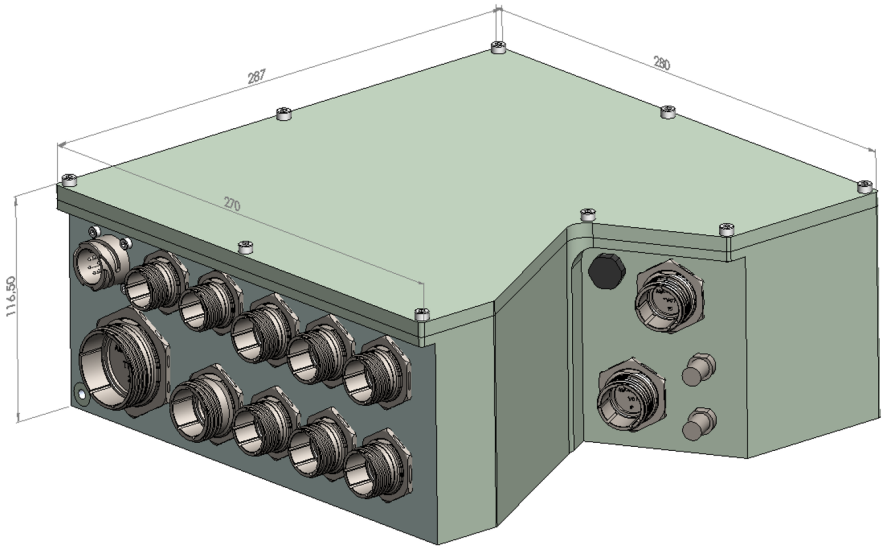


Figure 5.1: Electronic unit hardware enclosure

5.2 Automated Test Scope and Setup

Among several interfaces, there are three types of logical input and outputs: GPIO, DRU enable and presence check. The automated test area in this project focuses on DRU enable and presence check.

The DRU enable is a relay output from the computer module. If the electronic unit activates this signal, the relay will short the two output pins connected to one of the connectors. The presence check input of the computer module serves as a detection input of a specific IRV camera connected to the electronic unit. One of the two different cameras has a jumper inside that will pull the presence check signal low to indicate a connection. If the other camera is connected lacking of jumper, the presence signal will return 1.

Instead of testing the function of the DRU enable and presence check inside the electronic unit, data acquisition (DAQ) is used for testing the relay itself without access to the real hardware. The test principle is the same for the real electronic unit, which is shown in Figure 5.2. The DAQ board can be divided into two parts: switch control and switch check, which corresponds to the DRU enable and presence check function. The computer PC works as a test controller to set switch commands to the DAQ board. The python script running on the PC can send commands and read data automatically from the DAQ board to check the status of the relay. Test cases are shown in table 5.1.

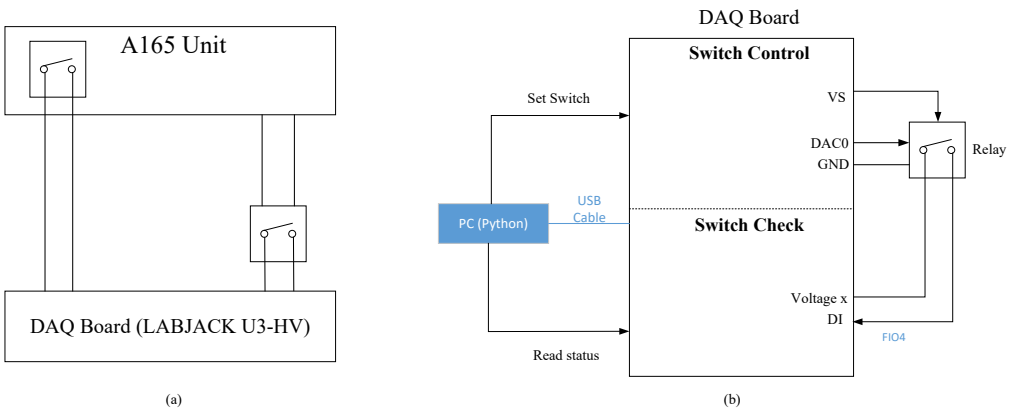


Figure 5.2: Test principle of real hardware and DAQ itself

Table 5.1: Test cases for the electronic unit test

Step	Description	Expected Result
1	Send 'open' signal from PC to DAQ output	Relay is opened
2	Check DAQ input of the received signal	The status of relay is low
3	Send 'close' control signal to DAQ output	Relay is closed
4	Check DAQ input of the received signal	The status of relay is high

The DAQ used in this project is called LabJack U3-HV. The equipment and the pin illustration are shown in Figure 5.3. LabJack U3-HV has 8 flexible I/O (FIO), 4 analog input (AIN), 2 analog output called DAC0 and DAC1. Each analog output can be set to a voltage up to 4.95 volts. The first four FIO can only be configured as analog inputs. FIO4 is selected to be a digital input pin to check the relay status. DAC0 is set to be an analog output with voltage either 5V or 0V, which corresponds to closing or opening the relay. The VS terminal is connected to the positive control of the relay to support the operating voltage of the PCB board. The physical connection is shown in Figure 5.4. The LabJack U3-HV is connected to the computer by an USB cable. A simple circuit diagram is depicted in Figure 5.5. The signal relay used is from TE connectivity which has 3V coil voltage. It provides high dielectric and surge capability depends on different contacts.

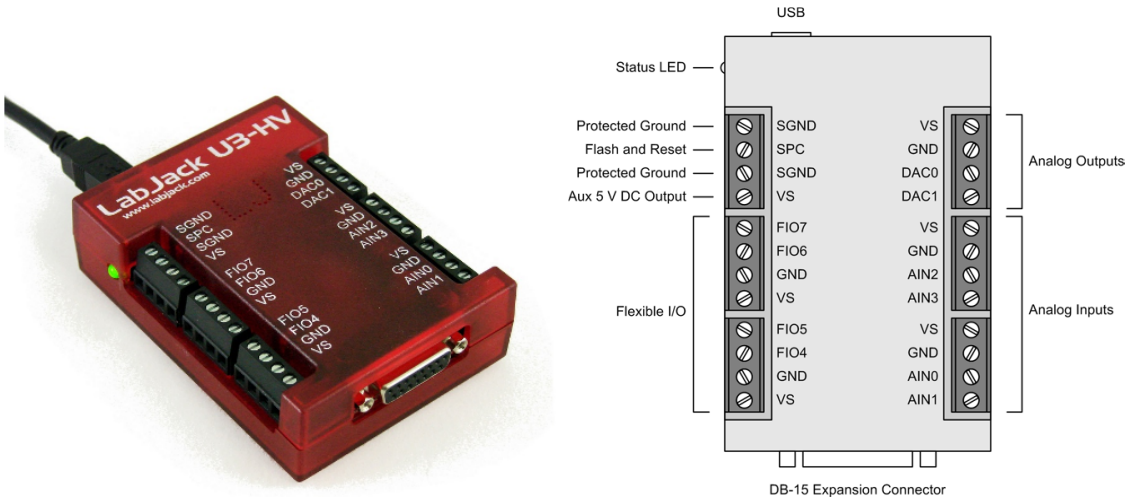


Figure 5.3: LABJACK U3-HV

5.3 Test Automation Setup

In this section, how the test automation of electronic unit is done is explained.

5.3.1 Tools

The software tools needed for test automation are Git, Gerrit, Jenkins, Python and JIRA. During an agile software development project, developers need to make changes for each build. With Git, a version control system, these changes are recorded and the files are saved in a version database [52]. The developers can recall a specific version at any time on any computers as well as prevent the situation of script missing. Gerrit is a code-review system which is built on the Git system [53]. The function of Gerrit is to review code file before it is committed. All committed files are saved at a central source repository, which is regarded as an authoritative copy of the project content [54]. The illustration of how Gerrit works is shown in Figure 5.6. There are two developers called Pumbaa and Timon, both of them can fetch the code from authoritative repository separately and edit

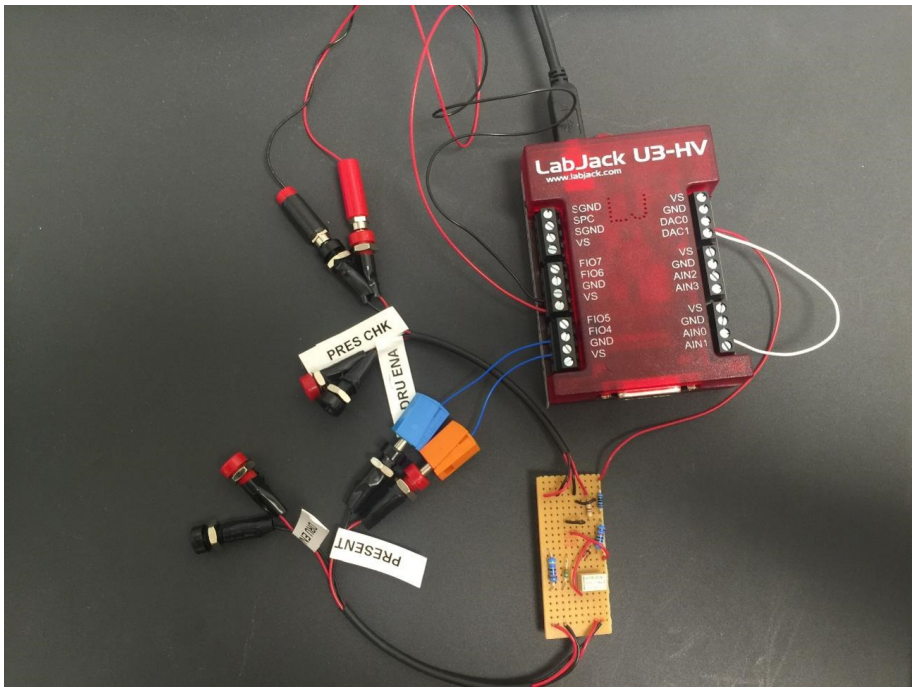


Figure 5.4: Connection of Labjack U3-HV and relay

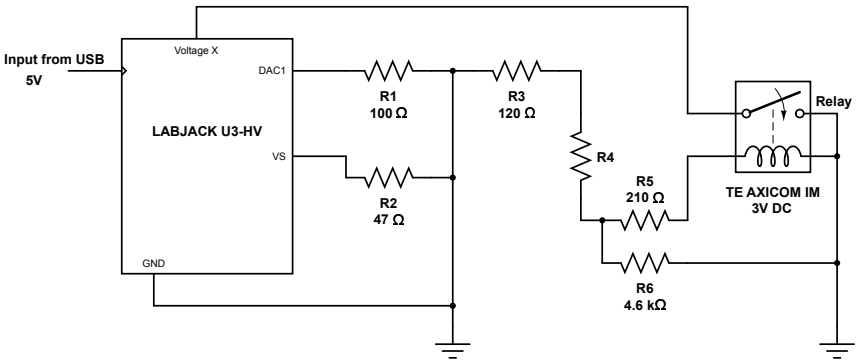


Figure 5.5: Circuit of Labjack U3-HV and realy

the script on their local computer. After the changes made, they push the modified scripts, which will be temporarily stored until another developer (reviewer) reviews the code. The reviewer should leave a verdict to either approve or reject in order to make the pending changes to be submitted to the final repository. Jenkins is a build management system used for continuous integration [55]. The main function of Jenkins is to trigger testing build automatically and give the test report. JIRA is a task management system, which is a development tool used for planning, tracking, test and issue management [56].

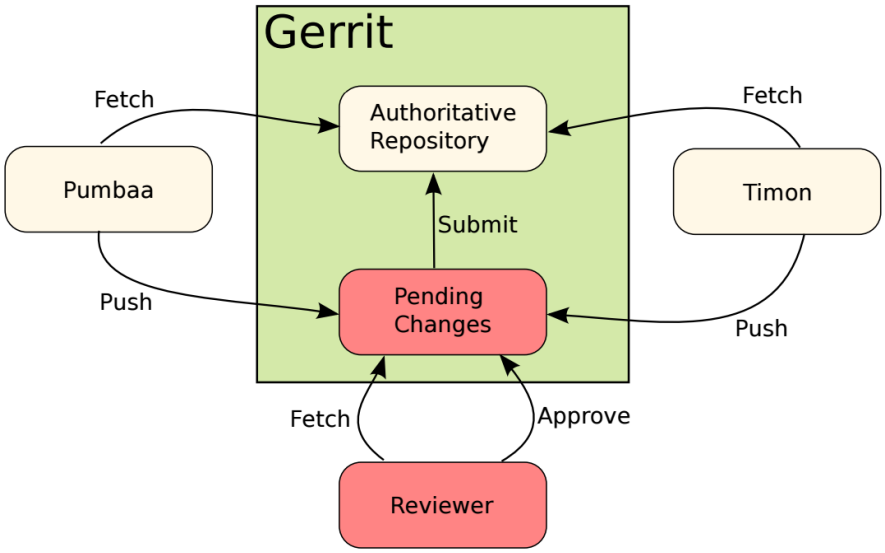


Figure 5.6: Gerrit workflow

5.3.2 Test Automation Implementation

The flowchart of test automation is shown in Figure 5.7. The developer first gets a task from the JIRA system and fetches the source code file from the Gerrit repository. The developer then updates code on the local computer and pushes the code to Gerrit. Afterwards, another software developer who is regarded as a reviewer fetches the code file from Gerrit, reviews the code and leaves a verdict. The code is also sent to Jenkins and given a test verdict back from it in parallel. If and only if two verdicts are postive, the code can be merged to Gerrit central repository and is public for everyone. The verdict mechanism is depicted in Figure 5.8.

The Python script for test cases, described in table 5.1, is executed in a test rig and the robot test automation framework discussed in section 2.2.3.1 is applied as a bridge between Jenkins and the Python script. The relation between Jenkins, the Python script and the robot framework is displayed in Figure 5.9. After the test is triggered in Jenkins, it sends command to robot framework to run tagged test cases on selected test application. The Robot framework executes the Python script based on the characteristic of keywords defined in the robot test data file. Afterwards, the feedback is given from the Python script and robot framework. Test results will be presented as a html report by Jenkins automatically. Finally, the report is handled by testers for checking the outcome.

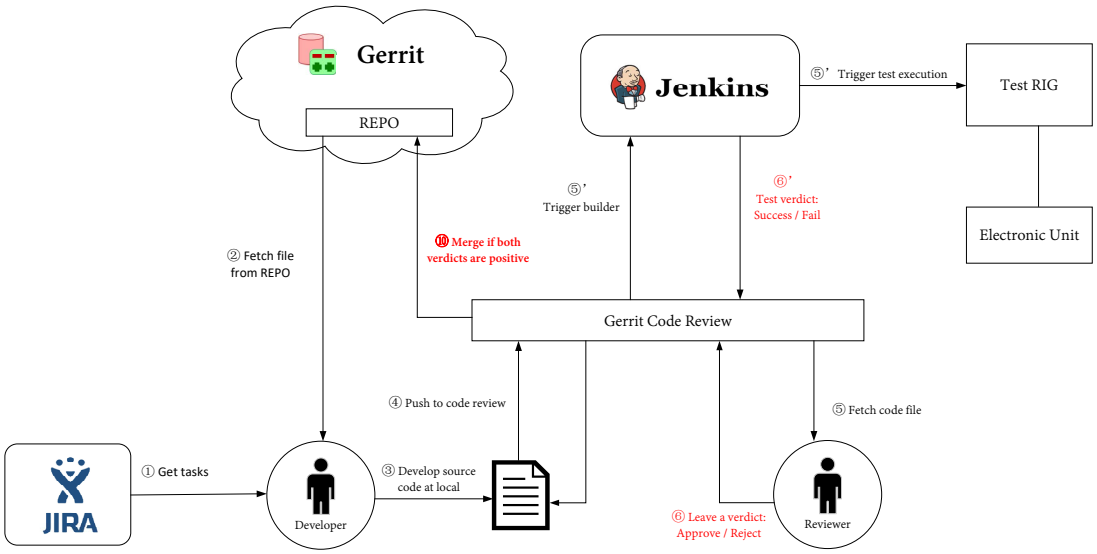


Figure 5.7: Electronic unit test automation procedure using Gerrit, Jenkins, JIRA

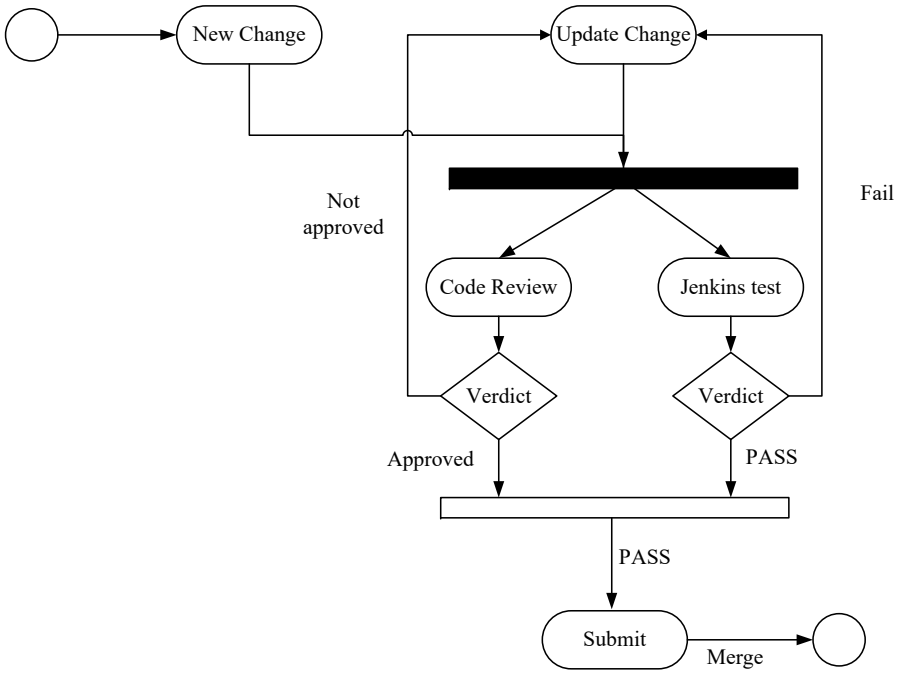


Figure 5.8: Verdict mechanism of test automation

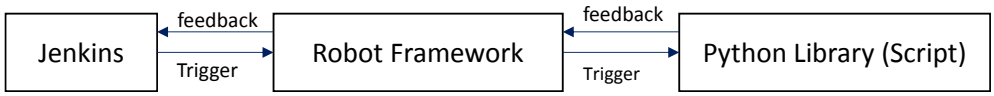


Figure 5.9: Relation between RobotFramework, Jenkins and Python

5.3.3 Test Automation Result

The command and window terminal of robot framework execution is presented in Figure 5.10. The test report generated by the robot framework is shown in Figure 5.11. It can be concluded that all test cases pass. The test framework input library is the python script. "Send command to control switch" and "Read switch status" are two keywords set in the robot framework file. The variable called "COMMAND_to_SWITCH" is used to send command to the relay.

With the keywords defined in the robot file, it is straightforward for testers and other people who are not programmers to understand the function without knowing the details of the scripts. The revision control system Git makes it possible to track all changes to perform merge operations and revert changes that go wrong. The introduction of Jenkins facilitate new tests to be integrated in a project continuous integration pipeline. The comprehensive reports generated by Jenkins simplifies the analysis and allows stakeholders and testers to quickly understand the result. Jenkins also saves all the execution history to provide a better view of product testing process.

```
C:\Users\A547476\Desktop\test_ver\Test_Cases>robot DAQ_Test.robot
==============================================================================
DAQ Test :: Tests the DAQ function
==============================================================================
Test if DAQ and switch respond to control signal from PC              | PASS |
------------------------------------------------------------------------------
DAQ Test :: Tests the DAQ function                                    | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  C:\Users\A547476\Desktop\test_ver\Test_Cases\output.xml
Log:     C:\Users\A547476\Desktop\test_ver\Test_Cases\log.html
Report:  C:\Users\A547476\Desktop\test_ver\Test_Cases\report.html
```

Figure 5.10: Command terminal of robot framework execution

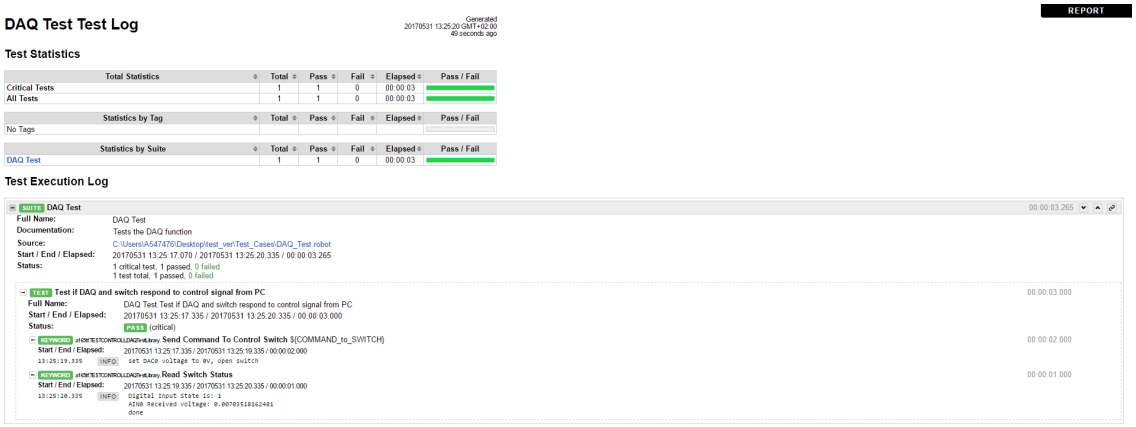


Figure 5.11: Successful test report generated by test framework

6

Analysis and Discussion

This chapter discusses the outcomes mentioned in the previous chapter and answers the research questions raised in section 3.1.

6.1 Answer to Research Question 1

RQ1 - Why is implementing test automation for automotive infotainment systems difficult?

The diverse functionality of infotainment ECUs is the main reason that makes automated test execution hard to realize. The ECU function can either be independently implemented or mutually related with other ECUs to provide more features, which in case challenge the integration testing. Another difficulty lies in simulating the complex user interface such as mobile device and center display console.

In this project, the implementation of ECU diagnosis test is one of the examples of test cases suitable for test automation. This is because the input and output signals can be simulated with the help of software tools and test cases can be easily decided. The execution includes neither the cognition nor the creativity of humans. However, this is not true for the other test areas within infotainment systems, for example, Apple carplay compatibility test, bluetooth devices compatibility test, media system test, hands-free performance test, over-the-air update test etc. The reason for the first two is that there are different infotainment platforms, networks and software releases in different targeted markets. In addition, the essence of an excellent infotainment system is good user experience. The user experience can sometimes only be acquired by testing the infotainment system manually. Examples of this are media system test and hands-free performance test. HMI, sound and display are the elements required for media system test. In order to define the types of HMI errors which test automation should face, it is necessary to find out all HMI errors happening in real situations. However, it is not easy to locate faults in HMI, for instance, just for menu navigation, there are probably hundreds of errors caused by inconsistency, language problem, wrong pop-up dialog windows, overlap text, erroneous next-level menu etc.

6.2 Answer to Research Question 2

How to decide whether to automate a test or run it manually within the considered systems?

Based on the answer to research question 1 and the literature study mentioned in the previous chapter, we can concluded that if a test is frequently needed and does not need human intervention, it is suggested to have test automation. However, it is also important to make sure that the project time range is long enough to develop test automation framework and scripts. In the second place, if a test frequency is high and needs human interaction, it is recommended to have a combination of automated test and manual test. The starting time of automated test can be decided by calculating M factor, which is the beginning point of test automation, in the new ROI calculation model mentioned in equation 4.13. In contrast, suppose that a test frequency is low and needs human interaction, it is generally not suggested to conduct test manually.

7
Conclusions

This chapter lists the achievements in this thesis project and future work which can be done.

7.1 Achievements

In the beginning of the thesis project, two objectives were set:

- Analyze suitable areas for automated test of vehicle infotainment systems. The chosen areas should be agreed on based on motivation and rationale of the domain.

- Create a Python script which can be applied to an ECU test in an automotive system.

This thesis first considered suitable areas for test automation within infotainment field. ECU diagnosis test has been further investigated and a theoretical implementation of test automation is demonstrated. However due to lack of supporting equipment, it can not be concluded that the test framework is valid for a real situation.

In this thesis, a new more detailed ROI calculation model used to calculate cost-benefit factor is provided based on predecessor research results. From the feedback and evaluation of ÅF engineers, it can be concluded that the model is good for showing all the elements need to be taken care but not very easy to conduct in industry. The challenge is that it is quite hard to define the cost of each unit described in the final equation, for example, average cost of single manual test case development cost. Nevertheless, the calculation model is effective if the unit price can be defined to some extent.

Another achievement in this thesis is realizing a test automation of an electronic unit for communication and surveillance purpose. Although there was no hardware support during the time I carried out this task, the test result is approved that the same test framework and test principle can be applied to later hardware version. With the combination of Jenkins, Gerrit and Robot Framework, the test process can be executed automatically and a test report is generated in HTML format.

7.2 Future Work

After reviewing the achievements, there are some which can be further developed after this projects.

First, the theoretical test framework can be applied in a real infotainment ECU test for validation purpose. Vector system modules need to be changed for different infotainment ECUs. The test cases defined need transition to language in CANoe, which is known as CAPL. The configuration of testing environment is necessary to be built in CANoe to emulate the ECUs working situation. The testers need to decide whether using real ECUs or simulated ECUs by CANoe.

Second, the new ROI calculation model can be used in real projects to calculate the cost-benefit factor. Based on the ROI factor and analysis discussed in section 6.2, testers can make a choice between manual test and automated test. In addition, since ROI calculation plays an important role of making decision for test automation availability, the accuracy should be improved in future work. Future research focus can be put on the factors that influence ROI calculation accuracy; the accuracy level of investment and benefit separately; a framework to estimate the erroneousness of ROI calculations etc.

Third, the simple test framework developed in section 5.2 can be used for a real hardware ECU test. The real outcome can be compared with the model testing method.

Bibliography

[1] D. Mauser, A. Klaus, K. Holl, & R. Zhang (2013). GUI failures of in-vehicle infotainment: Analysis, Classification, Challenges, and Capabilities.

[2] D. Sims. Infotainment features are top consideration among americans shopping for a new vehicle, 2008. [Online]. Available: `http://marketresearchworld.net/content/view/2210/77`. [Accessed: 2017-02-09].

[3] W. Waxenberger. What do Italian drivers want from tomorrow's car infotainment systems?, 2015. [Online]. Available: `https://blog.gfk.com/2015/01/what-do-italian-drivers-want-from-tomorrows-car-infotainment-systems`. [Accessed: 2017-04-25].

[4] S. R. Garzon (2012, June). Intelligent in-car-infotainment system: A prototypical implementation. In *Intelligent Environments (IE), 2012 8th International Conference on* (pp. 371-374). IEEE.

[5] Y. Huang, R. McMurran, M. Amor-Segan. "Development of an automated testing system for vehicle infotainment system" in *Advanced Manufacture Technology*. Springer. 2010, pp.223.

[6] D. M. Rafi, K. R. K. Moses, K. Petersen, & M. V. Mäntylä (2012, June). Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In *Proceedings of the 7th International Workshop on Automation of Software Test* (pp. 36-42). IEEE Press.

[7] National Instruments. Addressing key challenges in automotive infotainment test with the NI PXI platform, 2014. [Online]. Available: `http://www.ni.com/white-paper/51912/en`. [Accessed: 2017-06-20]

[8] A. Singhal and A. Bansal. "A Study of Various Automated Test Oracle", in *2014 5th International Conference- Confluence The Next Generation Information Technology Summit*. IEEE, 2014.

[9] J. Rushby (2005). Automated test generation and verified software. In *Working Conference on Verified Software: Theories, Tools, and Experiments* (pp.

161-172). Springer Berlin Heidelberg.

[10] S. Eldh, K. Andersson, K. Wiklund. "Towards a test automation improvement model", in *2014 International Conference on Software Testing, Verification, and Validation Workshops*. IEEE, 2014.

[11] S. Reñé Vicente (2015). Handoff management for infotainment services over vehicular networks.

[12] A. V. Vasilakos, M. Parashar, S. Karnouskos, & W. Pedrycz(Eds.) (2009). *Autonomic Communication* (Vol. 51, p. 171-172). Springer Science & Business Media.

[13] S. R. Garzon (2012). Intelligent in-car-infotainment systems: A contextual personalized approach. In *Intelligent Environments (IE), 2012 8th International Conference on* (pp. 315-318). IEEE.

[14] B. Fleming (2013). Smarter cars: Incredible infotainment, wireless device charging, satellite-based road taxes, and better EV batteries [Automotive Electronics]. *IEEE Vehicular Technology Magazine*, 8(2), 5-13.

[15] K. Reif (2015). Automotive mechatronics: Automotive networking, driving stability, systems. *Electronics. Springer Vieweg.* (pp. 106).

[16] "Car audio basics: Head units, amplifiers, and speakers", 2018. [Online]. Available: `https://www.lifewire.com/car-audio-basics-head-units-534562`. [Accessed: 2018-05-23].

[17] N. Navet, & F. Simonot-Lion (2013). In-vehicle communication networks-a historical perspective and review. University of Luxembourg.

[18] M. Rodelgo-Lacruz, et al. "Base technologies for vehicular networking applications: review and case studies." *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on.* IEEE, 2007.

[19] H. Chen, & J. Tian (2009). Research on the controller area network. In *Networking and Digital Society, 2009. ICNDS'09. International Conference on* (Vol. 2, pp. 251-254). IEEE.

[20] "Controller area network (CAN) overview", 2014. [Online]. Available: `http://www.ni.com/white-paper/2732/en/#toc3`. [Accessed: 2018-06-21].

[21] K. Pazul (1999). Controller area network (CAN) basics. *Microchip Technology Inc, 1.*

[22] S. Corrigan (2008). Introduction to the controller area network (CAN). *Texas Instrument, Application Report.*

[23] Cooperation, M. O. S. T. (2008). Media oriented systems transport (MOST).

[24] A. Sumorek & M. Buczaj (2012). New elements in vehicle communication "Media Oriented Systems Transport" protocol. *Teka Komisji Motoryzacji i Energetyki Rolnictwa*, 12(1).

[25] R. Shaw (2009). Improving the reliability and performance of FlexRay vehicle network applications using simulation techniques. Doctoral dissertation. *Waterford Institute of Technology.*

[26] "FlexRay automotive communication bus overview", 2016. [Online]. Available: `http://www.ni.com/white-paper/3352/en/`. [Accessed: 2018-03-10].

[27] S. Moradpour Chahaki (2012). On-Board diagnostics over Ethernet.

[28] B. Metcalfe, et al. (2014). Automotive Ethernet - the definitive guide. *Intrepid Control Systems.*

[29] S. H. Seo, J. H. Kim, T. Y. Moon, S. H. Hwang, K. H. Kwon, & J. W. Jeon (2008). A reliable gateway for in-vehicle networks. *IFAC Proceedings Volumes*, 41(2), 12081-12086.

[30] S. H. Kim, S. H. Seo, J. H. Kim, T. Y. Moon, C. W. Son, S. H. Hwang, & J. W. Jeon (2008). A gateway system for an automotive system: LIN, CAN, and FlexRay. In *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on* (pp. 967-972). IEEE.

[31] K. Swathi & M. Narasimhan (2015). Implementation of diagnostics module in car-infotainment system. In *International Journal of Scientific & Engineering Reserach* Volume 6, Issue 5.

[32] B. Stein, O. Niggemann, & H. Balzer (2006). Diagnosis in automotive applications. In *3rd Monet Workshop on Model-Based Systems (MBS-06) at the ECAI* (Vol. 6).

[33] ISO14229-1. Road vehicles - Unified diagnostic services (UDS)-Part 1: Specification and requirements[S]. 2013

[34] M. Ring, T. Rensen, & R. Kriesten (2014). Evaluation of vehicle diagnostics security – Implementation of a reproducible security access. *SECURWARE 2014*, 213.

[35] A. Ieshin, M. Gerenko, & V. Dmitriev (2009). Test automation: Flexible way. In *Software Engineering Conference in Russia (CEE-SECR), 2009 5th Central and Eastern European* (pp. 249-252). IEEE.

[36] D. H. Kum, J. Son, S. B. Lee, & I. Wilson (2006). Automated testing for automotive embedded systems. In *SICE-ICASE, 2006. International Joint Conference* (pp. 4414-4418). IEEE.

[37] R. Pelau (2015). Automated testing in automotive in *Advanced embedded systems.*

[38] D. M. Rafi, et al. (2012). Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. In *Proceedings of the 7th International Workshop on Automation of Software Test* (pp. 36-42). IEEE Press.

[39] S. Berner, R. Weber, & R. K. Keller (2005). Observations and lessons learned from automated testing. In *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*(pp. 571-579). IEEE.

[40] P. Laukkanen (2006). Data-driven and keyword-driven test automation frameworks. *Master's thesis. Helsinki University of Technology.*

[41] T. Pajunen, T. Takala, & M. Katara (2011). Model-based testing with a general purpose keyword-driven test automation framework. In *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference* on (pp. 242-251). IEEE.

[42] R. Framework. Robot Framework, 2017. [Online]. Available: `http://robotframework.org`. [Accessed: 2017-10-10].

[43] Robot Framework user guide (version 3.0.2). Robot Framework Foundation.

[44] R. Hegde, & K. S. Gurumurthy (2009). Load balancing towards ECU integration. In *Advances in Recent Technologies in Communication and Computing, 2009. ARTCom'09. International Conference* on (pp. 521-524). IEEE.

[45] A. X. A. Sim, & B. Sitohang (2014). OBD-II standard car engine diagnostic software development. In *Data and Software Engineering (ICODSE), 2014 International Conference* on (pp. 1-5). IEEE.

[46] ISO 22901-1: 2008(en) Road vehicles - Open diagnostic data exchange (ODX) - Part 1: Data model specification

[47] Manual CANcaseXL/log - vector (version 5.2). *Vector Informatik GmbH.*

[48] Vector system user manual (version 1.14). *Vector Informatik GmbH.*

[49] D. Johnson. Test automation ROI. *Test Automation ROI*, 1.

[50] D. Hoffman (1999). Cost benefits analysis of test automation. *STAR West, 99.*

[51] M. Cui & C. Wang (2015). Cost-benefit evaluation model for automated testing based on test case prioritization. *Journal of Software Engineering*, 9(4), 808-817.

[52] S. Chacon & B. Straub (2014). Pro git on (Chapter 1 pp.1). Apress.

[53] P. Thongtanunam, et al (2015). Who should review my code? A file location-based code-reviewer recommendation approach for modern code review. In *Software Analysis, Evolution and Reengineering (SANER), 2015 IEEE 22nd International Conference on*(pp. 141-150). IEEE.

[54] M. B. Zanjani, H. Kagdi, & C. Bird (2016). *Automatically recommending peer reviewers in modern code review.* IEEE Transactions on Software Engineering, 42(6), 530-543.

[55] V. Armenise (2015). Continuous delivery with jenkins: Jenkins solutions to implement continuous delivery. In *Release Engineering (RELENG), 2015 IEEE/ACM 3rd International Workshop on*(pp. 24-27). IEEE.

[56] H. M. Sarkan, T. P. S. Ahmad, & A. A. Bakar (2011). Using JIRA and Redmine in requirement development for agile methodology. In *Software Engineering (MySEC), 2011 5th Malaysian Conference in*(pp. 408-413). IEEE.