Test Automation to Enable Continuous Integration for an Automotive Platform: A Design Science Study of Software Download Function Case

Master's thesis in Computer Systems and Networks

JACOB THOMAS SIMON
ANUSHA BALABHADRAPATRUNI

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis 2017

Test Automation to Enable Continuous Integration for an Automotive Platform: A Design Science Study of Software Download Function Case

JACOB THOMAS SIMON

ANUSHA BALABHADRAPATRUNI

Department of Computer Science and Engineering
Division of Computer Systems and Networks
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2017

Test Automation to Enable Continuous Integration for an Automotive Platform: A Design Science Study of Software Download Function Case

JACOB THOMAS SIMON

ANUSHA BALABHADRAPATRUNI

Supervisor: ERIC KNAUSS, Department of Computer Science and Engineering
Examiner: FRANCISCO GOMES, Department of Computer Science and Engineering

Master's Thesis 2017
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2017

Test Automation to Enable Continuous Integration for an Automotive Platform: A Design Science Study of Software Download Function Case
Jacob Simon
Anusha Balabhadrapatruni
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Continuous integration is being widely used in software industry for frequent product releases and better customer satisfaction through a set of standards which integrates software modules continuously, reduces feedback time between testing and defect fixing and delivers the product successfully into production environment without any errors. The development and testing of software in automotive industry is different from that of software industry mainly because of the proximity of software and hardware and the development of hardware and software modules by many different vendors. Thus adapting continuous integration practices to automotive industry is challenging and studies are going on to address the challenges which may improve the current scenario.

The automotive industry follows a development model in which testing of certain functionality is done often. At the case company where the thesis study is conducted, the software download functionality is very important as it is used to add additional functionality or error correction in the Electronic Control Units or embedded computer system and this functionality is tested on multiple ECUs repeatedly before the car release. The software is updated periodically and the testing cycle has to be repeated after every update. However test engineers spend a substantial amount of time to test the software download functionality manually and the testing includes test execution, result analysis and reporting due to which there is higher feedback time between testing and defect fixing.

In this thesis we identify some of the challenges in adapting continuous integration in automotive platform development and suggest recommendation to solve some of the challenges by implementing them to a sample functionality which is developed at the case company. One of the goals of adapting continuous integration in automotive industry is to reduce feedback time and the thesis focuses on this aspect by automating the software download testing process through test automation framework. The testing process consists of downloading the binary files for flashing, using flashing tool to check the sequence of download according to standards, logging the bus signals in CANoe and Wireshark and generating report that has information about the test steps. The thesis follows design science methodology using which automation framework is developed in three iterations and in each iteration, parts of testing process are automated. Interviews are conducted with test engineers during each iteration to understand the problem and to receive feedback.

Keywords: Continuous Integration, Automation Framework, Software download

Acknowledgements

We would firstly like to thank the case company for giving us the opportunity to pursue our thesis at their office. We would specially like to thank Eric Knauss, our supervisor at Chalmers university of Technology and David Kanyinda, our supervisor at the case company for their guidance and valuable advice throughout the thesis. We would like to thank Francisco Gomes, our examiner at Chalmers University, for his support and advice. Finally we would like to thank the entire team we worked for in the case company for helping us, and actively participating in demo and interviews.

Jacob Simon and Anusha Balabhadrapatruni, Gothenburg, September 2017

Contents

List of Figures

List of Tables

1
Introduction

The automotive industry in general follows a process model that can be characterized as a plan-driven model. This is because of the complex environment where automotive platform developers depend on suppliers for hardware and software components [14]. Intuitively, this leads to a state of entropy where suppliers deliver their products in different time lines which in turn causes delays in testing and feedback. Due to the high complexity and safety critical requirements, testing of vehicle embedded systems is a very crucial process in automotive industries. Modern cars are equipped with electronic control units having re-programmable flash memory to facilitate software updates and this feature is important as it is used to fix software defects which are increasing according to studies conducted by National Highway Traffic Safety Administration (NHTSA) [35]. So it is imperative to have proper testing before cars are released to the markets. Today many automotive companies face challenges in conducting testing activities. This is partly because many companies use manual testing which consumes a lot of time. Automation of the testing processes can be very helpful for testers as it would reduce a considerable amount of time consumed for tracking test case execution and reporting. In this master thesis we have developed an automation framework for testing software download in vehicle embedded system which can run test cases and generate reports without human intervention and the tool can save approximately 8 hours for running several test cases which would take approximately 12 hours if run manually. This report initially explains the main problems, scope and limitations of our master thesis in this chapter. The background and context of the thesis is described in detail in chapter 2. The research methodology and the different phases of the methodology used in the thesis are explained in the chapter 3. The investigated problems, implementation, design artifacts and validation are portrayed in relation to the the research methodology in 4. Further the research questions are answered based on the outcome of the automation in results chapter 5. The conclusion and future work are stated in the conclusion chapter 6.

1.1 Purpose of Study

The provision of software download functionality in modern cars is important for error correction or adding new functionality in aftermarket. The purpose of the thesis is to study the testing strategies and development methodologies used in automotive industry through the case company and to suggest and implement the an automation framework to automate testing of software download. A thorough

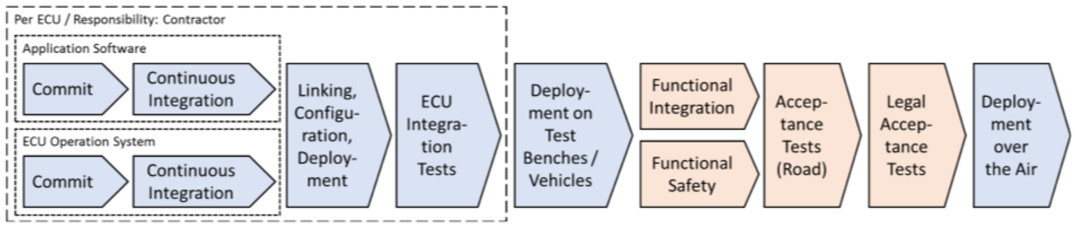


Figure 1.1: Deployment pipeline[28]

investigation is done on how the automation strategy would improve the testing and development process in automotive platform. The use of an automated testing tool can improve the usage of expensive test objects and the time required for testing. Moreover many of the mechanical parts are being replaced with electrical parts in the automobile industries and there is a lot of scope of improvement of the practices used for handling software. For this purpose we have chosen to do our master thesis in collaboration with the case company which specializes in research and development within automotive industry.

1.2 Problem Identification and Goals

System development in automobile industry is different from software industry because of safety criticality and hardware dependency. For example, in mobile phone industry, apps and new functions can be released to user instantly and feedback can be collected from end users. The fail fast approach is widely used in innovative software where, if part of software fails, the failure is made visible through assertions. But in automotive industry, this approach cannot be adopted since it is highly safety critical and hence the software needs to be thoroughly tested before releasing into the market[28].

A deployment pipeline in automotive industry is as shown in Figure 1.1. Application software, operating system and libraries are linked and built into one ECU and on this ECU tests like unit tests and integration tests are done. In the next phase the code is checked into central repository in binary format. From this central repository, codes are flashed into test bench consisting of many ECUS. On the test bench, functional integration tests and safety tests are done. Testing at this stage requires enormous efforts since a total of approximately 10000 tests have to be done. The final phase is deployment and this can be done over the air or through wired connections[28].

In automotive industry the development method follows big-bang integration method where all the individual hardware and software modules are integrated at once and subjected to testing. Due to dependency on hardware, these test cases take a long time to complete. Due to big-bang integration finding, tracing and fixing errors can become tedious. In a system level, the input artifact would be pre-compiled software containers for specific ECUs and the output artifact would be evaluations, analysis and reports of executed test cases. In this thesis we focus on developing test automation framework for a specific functionality that is often tested to find out how this can maximize error detection and reduce the time taken for testing[33].

In the current development process, there are many integration points where hardware and software modules are combined. Software download process is an important functionality that needs to be tested at every integration point because it affects most of the electronic control units in a vehicle. The test data for testing software download, are stored in a word document and a tester checks and enters data in the testing tool for each test case manually, every time a test case is executed. Each step of a test case needs to be checked manually and reported which wastes lot of time and effort for the tester.

A box car is an environment where different modules are combined physically. In the testing process the test item or the object of testing is the box car. Box car is an expensive test item and has limited availability. The usage of this test item is limited to working hours as testers have to be physically present at the test item for testing. An automation framework to use the test object efficiently can potentially reduce the time consumed in testing software download.

Test analysis and report generation are also important activities in the testing process of software download, because this helps in finding the root cause of download failure. Currently test engineers spend a substantial amount of time analyzing test logs to find the root cause, due to lack of proper analysis scripts. By automating the existing reporting procedure, the level of manual intervention in the test case reporting and test monitoring can be reduced. From the case company's perspective this could reduce the testing time and effort. The suppliers may get feedback about the products earlier and this helps the suppliers to fix errors faster and meet the deadline for next integration point. Due to the unavailability of a model to integrate before trigger points, the suppliers who complete their software earlier cannot test their software in an integrated environment like a box car.

1.2.1 Contributions to the case company

- **Developing a generic and scalable Test automation framework:** The thesis aims to develop a test automation framework which is generic and scalable so that it can be reused for different projects. We aim to build the automation framework for one test case in such a way that it can be extended for other test cases with some adjustments. This makes it imperative to identify the generic requirements and corresponding test cases.

- **Developing UI for using the automation framework:** It is impractical to make changes in program for automation framework to enter the parameters required for running automation. Nevertheless, the test engineers in automotive industry may not be equipped with programming capabilities. To use the automation framework easily, it is essential to build a user interface through which the parameters and file or folder locations can be entered. The test engineers must also find it easy and effort efficient to use the UI.

- **Test Results Analysis:** The test engineers spend a substantial amount of time in analyzing test logs generated by test environment. The thesis also aims at implementing or suggesting a method to generate a well structured report using which testers can easily find out the reasons of failure.

1.2.2 Contributions to the Scientific Community

- **Impact of Automation on System Development:** Another goal is to identify the impact of automation process on the overall system development. The impact of test automation greatly depends on the test coverage[19]. We conducted a survey and get feedback from the end users and practitioners of this automation tool. This knowledge can be used to evaluate how automation affects the system development process.

- **Challenges of continuous integration on Automotive platform level** The testing process of software download is done at every integration. There are generally more than three integration points and at each integration point the software files are received from all the suppliers. The development process follows a waterfall model which may delay the time for suppliers to get feedback from testers and improve the code. Adapting continuous integration may improve the process and reduce the feedback time.

- **Factors to be considered while developing a test automation framework in automotive platform** Testing can be automated in order to improve the amount of time consumed after each integration point and there are different projects running in parallel in an automotive platform. A test automation framework can automate different aspects of testing.

- **Solutions to overcome the challenges faced by platform developers with respect to continuous integration** The thesis focuses on implementing test automation framework to enable continuous integration. From the interviews conducted, literature review and implementation of test automation framework, it may be possible to derive some solutions to overcome some of the challenges faced by automotive industry in adapting continuous integration.

1.3 Scope and Limitations

The scope of this thesis is to identify some of the challenges of adapting continuous integration faced by the case company which is an automotive platform developer and to analyze how a test automation framework can be effective in addressing some of them. We develop an automation framework that reduces the effort of testers by automating test execution, log generation, report generation and test analysis. The focus is to suggest and implement an automation strategy for automating system testing process of software download functionality. This strategy is analyzed in terms of the improvement in current development process of the company.

In this thesis we focus on improving the testing process through automation, implementing remote access of box car and by implementing a tool that helps in analysis of test results. By doing so, we learn about general challenges, desired properties, and promising solution candidates for test automation in automotive platform development. In addition we also intend to suggest solutions that can improve the current development process with respect to integration with different suppliers. However we will not implement all the solutions, since it will take longer time and requires agreement among key stakeholders of the company and the suppliers.

2 Background

This chapter covers the technical and scientific concepts required to understand the thesis subject. The section begins with introducing the case company and the nature of technical work done there. Electronic control unit which is a very important part of a car is explained in the next section. In the following section, all the automotive networks like CAN, LIN, MOST and FlexRay are explained briefly. In the next section the current software download process at the case company is described following which, there is a section explaining the software testing methods in and the current methodology of testing followed by the company. Further, a box car and its usage are explained and the section also explains the Hardware in loop or HIL testing. The last section gives context of automated testing and continuous integration that we intend to explore through the thesis.

2.1 Case Company

China Euro Vehicle Technology (CEVT) is a development center located at Gothenburg, Sweden, that covers all aspects of a passenger car development. The EE architecture branch within the Electrical department is responsible for vehicle's entire electrical architectural design. The base technology section within EE architecture is responsible for car network testing, software download, car configuration etc.

The company distributes electrical platforms or models to car manufacturers and the components of the models, called electrical control units, are developed by different vendors based on the case company's requirements and this development model resembles that of an OEM. The components are then assembled at the case company and the whole platform or model is then tested and distributed to car manufacturers.

2.2 Electronic Control Units

The passenger car consists of many embedded control units called Electronic Control Units. ECUs process information through sensors has the ability to control different processes like navigation or temperature control. There has been a continuous increase in features and complexity of ECUs which affect the software implemented in them. To cop up with this design complexity, it is imperative to have a provision to program ECUs during assembly or service. By having a provision of software download on all ECUs new functions can be added even after production and the cost of error correction can be significantly reduced[35].

The software download is done by a software module called bootloader in the ECU. There are two types of bootloaders, primary bootloader (PBL) and secondary bootloader (SBL). The primary bootloader is permanently present in the non-volatile memory and is activated on ECU start or reset. However for security reasons it does not have the access to erase or program the non-volatile memory and can only write to the volatile memory.

The secondary bootloader is downloaded by the primary bootloader into the volatile memory every time software download is performed. The SBL can perform all the operations that can be performed by the primary bootloader and in addition it can erase or write the non-volatile memory. So it is used to perform the software download and after the software download is finished the SBL is erased from the volatile memory.

2.3 Vehicle Network

The vehicle communication network needs to satisfy real time communication requirements, for example, having interference from network environment and being cost-effective. The different performance requirements throughout a vehicle,as well as competition among companies within automotive industry, have led to the design of a large number of communication networks[34]. This section describes the different automotive networks like CAN, Flexray, LIN and MOST which are used in a modern car and the automotive platforms developed at the case company use all the networks mentioned above. The test engineers at the case company have good knowledge of how messages traverse through these networks to different ECUs which is essential to test the functionality implemented in automotive platform. We have used the knowledge of these networks to do test analysis, which is explained later in the report.

2.3.1 Controller Area Network(CAN)

Controller area network was developed by Bosch in 1980 and is by far the most widely used in-vehicle network[34]. CAN became an ISO standard in 1994[5]. The success of CAN network can be attributed to a number of reasons. CAN provides an inexpensive, durable network that helps multiple CAN devices communicate with one another[36]. An advantage to this is that electronic control units (ECUs) can have a single CAN interface rather than analog and digital inputs to every device in the system[36]. This leads to an overall decrease of cost. Even though the amount of data to be transmitted has been increased, CAN is still the primary standard. This is mainly because the number of CAN channels can be increased when there is more data to be transmitted.

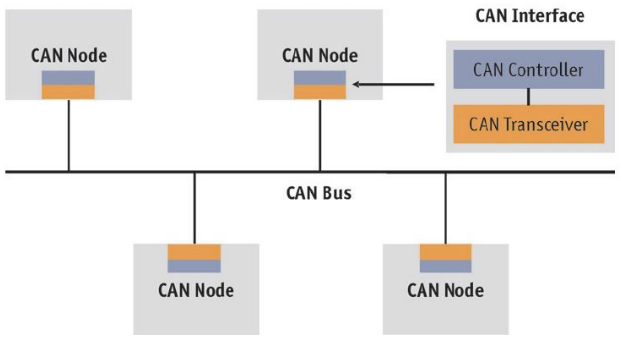


Figure 2.1: CAN Bus Topology from[11]

On CAN the data is segmented in form of frames. These frames can be sent periodically or aperiodically. Every CAN frame is labelled by a number called the identifier which determines the priority of the frame while transmitting.

The network topology of CAN is given above in the Figure 2.1. The CAN network has decentralized structure and all the network nodes can access the bus at any time. To support this communication over a multi-master bus each node has a transceiver and a controller. CAN is an event driven communication protocol. To support the real time behaviour and increase the reaction time to events two bus access techniques are used. These bus access techniques are CSMA/CA(Carrier Sense Multiple Access/Collision Avoidance) and arbitration technique.

In the CSMA/CA technique whenever a node in the CAN network wants to transmit a message it checks first if any other node is transmitting on the bus. If there is any other node communicating then it refrains from sending the message for some time and then checks again for bus availability. Despite using this technique if there is still a collision then the arbitration scheme is used to prioritize the message to be sent.

Based on the arbitration scheme when multiple nodes are transmitting message at same time then the node with the highest priority will continue sending the message. The nodes with lower priority stop sending messages.

The main advantage of CAN is that it is a light-weight and low cost network. Also all the devices in the network have a CAN controller chip, so they can see all the transmitted messages and decide whether to filter it or not. The arbitration scheme allows networks to reduce traffic congestion and meet deterministic timing constraints[36]. CAN also provides error free transmission as each node can check for error based on cyclic redundancy code(CRC) and also send an error frame to transmit the error signal.

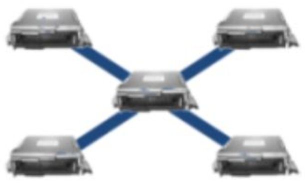


Figure 2.2: Star Topology from [10]

2.3.2 FlexRay

CAN is a dominant protocol for in-vehicle network, but it cannot provide real-time performance, which is essential in safety-critical applications, such as the x-by-wire system and advanced driver assistance system (ADAS)[37]. To solve this problem, FlexRay was designed using a time-division multiple-access (TDMA) mechanism for safety-critical systems[37]. FlexRay is a time triggered communication network that is the activities are driven as time progresses. FlexRay was developed by a consortium of major companies. Currently lot of automotive companies use both CAN and FlexRay networks together based on their requirements.

The FlexRay network has a quite flexible topology and can be used as bus, star or multi-star. In a star topology all nodes communicate through a central node. The bus and star topologies can be combined to form hybrid topology. This can help to take advantage of the ease-of-use and cost advantages of the bus topology while applying the performance and reliability of star networks where needed in a vehicle[25].

FlexRay has the capacity to take the place of multiple high speed CAN busses, reducing the complexity and cost[38]. FlexRay provides support for deterministic communication and higher bandwidth compared to CAN. The FlexRay standard manages communication of multiple nodes or ECUs through one channel. This is done with a pre-set communication cycle that provides a pre-defined space for static and dynamic data[25]. The communication cycle is as shown in the figure 2.3 from [10].

The blue portion of the frame shown in the figure 2.3 are static segments which are used for deterministic communication. The static segment is divided into different time slots. Each time slot is dedicated to a particular ECU. Each ECU transmits messages in the particular time slot only in an communication cycle. In case it misses the time slot it waits until the allocated time slot in next communication cycle. This helps the program to determine how old the transmitted data is.

In vehicle communication networks there are wide variety of data to be transmitted like high speed messages and low speed messages. FlexRay provides dynamic segments to prevent slowing down the communication cycle by more static slots. The dynamic segment is of fixed length and only a fixed amount of data can be sent. The dynamic segment is divided into mini-slots which are pre-assigned to frame of data. The mini-slots are of 1 microsecond length. Higher priority messages are transmitted in the mini-slots closer to the beginning of dynamic segment. The symbol window

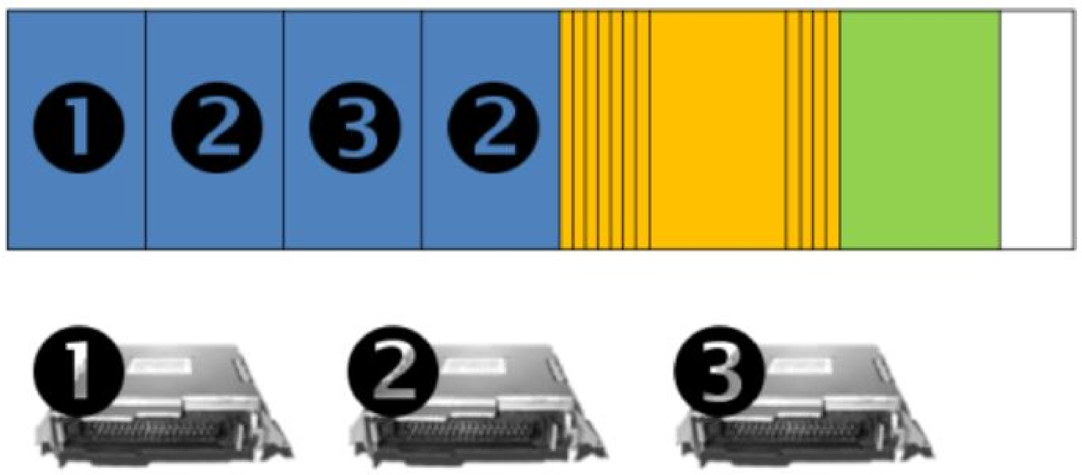


Figure 2.3: Communication Cycle of FlexRay Network

as shown in yellow in figure is used for maintenance of special cycles. It is also used for signalling for starting the network. The network idle time in white in figure is used to maintain synchronization between node clocks.

2.3.3 MOST and LIN

LIN or Local Interconnect Network was developed to be used in applications which do not require high data transfer rate and robust characteristics of CAN network. This reduces the unnecessary cost. Some examples of usage of LIN are seat, door and mirror control.

Media Oriented Systems Transport or MOST is a communications standard used in automotive applications. It is optimized for multimedia and infotainment applications that require high data transfer rates [9]. MOST was developed to provide low-overhead and low-cost interface for simple devices like speakers and microphone. The features of MOST make it suitable for any application, inside or outside the car that needs to network multimedia information along with data and control functions[9].

2.4 Software Download

As mentioned in the above section software download gives a provision to add an additional functionality or perform efficient error correction even after the production. In our case company software download generally involves transfer of information from tester to the ECU. This information transfer is usually done using data stored in a file. Some data from ECU is uploaded from ECU to tester which can be used for debugging. This process is referred to as flashing. In our case company internal flashing tool is used to perform software download.

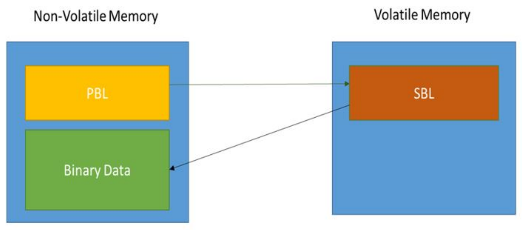


Figure 2.4: Two-Level Bootloader

Software Download is done on the ECU using the two level bootloader as shown in figure 2.4. Usually the ECU can be in two modes, either a default mode or a programming mode. The application of ECU runs in the default mode. The software download is done in the programming mode. When an ECU is reset or started the Primary bootloader or PBL is activated. The software download is done by the secondary bootloader.

Every time a software download is performed the tester has to manually upload the download files to the internal flashing tool. Then a diagnostic request is done to check the mode of the ECU. An ECU reset is requested if it is in the default mode and then software download is performed once it enters the programming mode.

2.4.1 Software Download Sequence:

In the figure 2.5 the different steps in the download sequence are shown.
Enter Program Mode: In this step a diagnostic request is sent to all the ECUs requesting programming session. A positive response is received if it enters the programming mode. This is done as the software download can be done only in the programming mode.
Pre-Programming Sequence: In this step a security access check is performed where the ECU pins are verified and then only the tool can get access to ECU to perform software download. If the option for parallel software download is selected then the security access verification is done parallely on all ECUs.
Programming Sequence: This programming sequence mainly comprises of steps for data download to the ECU. The SBL is always downloaded and activated first followed by other files. Download of the data file step is repeated several times till all the files are flashed. In this case if the parallel download option is selected for all ECUs then the download of SBLs is done in parallel but the activation is done serially. Further rest of the programming sequence is loaded in parallel on all ECUs. Although the sequence on one ECU is loaded in a serial manner.
Post-programming Sequence: The post-programming sequence includes whether the whole data is transferred. It also verifies the software integrity and restarts the vehicle via a reset.
The messages between the tester and the ECU have to pass several gateways which results in certain latency. The latency can be reduced by sending a queued request. This method basically sends an extra request before the response of the first request is received. This saves time of processing the request and then sending the next one.

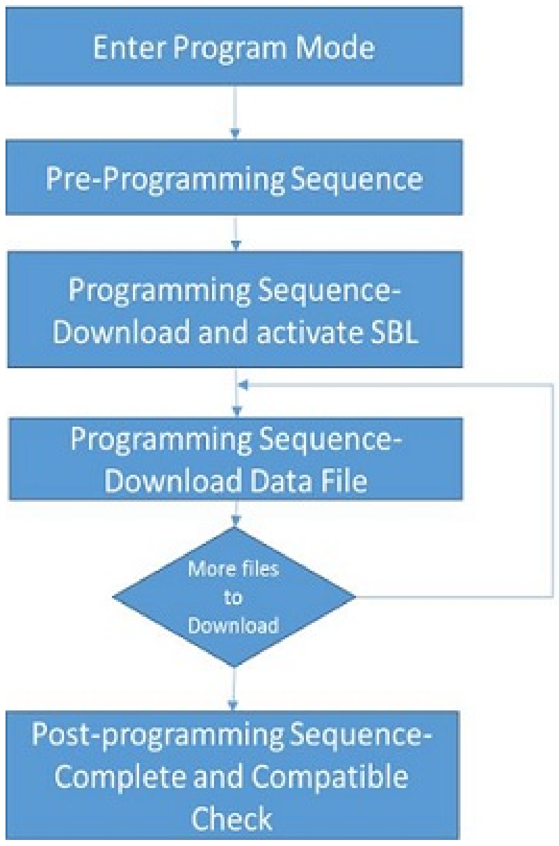

Enter Program Mode
Pre-Programming Sequence
Programming Sequence- Download and activate SBL
Programming Sequence- Download Data File
More files to Download
Post-programming Sequence- Complete and Compatible Check

Figure 2.5: Software Download Sequence

The parallel or queued options speeds up the software download especially when it is done on multiple ECUs.

2.5 Software Testing

Software testing is one of the most important aspects of software development life cycle and it occupies 40% of the budget in most of the development projects[6]. This is mainly because software testing ensures the quality of the product and satisfaction of end users and the test items need to be verified, validated and evaluated at all phases of software development. Testing process can be manual, automated or a mix of both. Manual testing is labor intensive and time consuming and prone to human errors. Automated testing is more efficient and test results are more consistent and reliable which in turn ensures better quality of the product[6].

A test project consists of a number of test sub-processes. A test sub-process can be related to a phase of software development life cycle or a quality of the software being tested. If it is related to software development life cycle, a test sub-process can be unit testing, integration testing, system testing and acceptance testing[16]. A software product is typically made of different software modules that may be, in turn, developed by different developers and testing is carried out at different stages of development. Unit testing is done on individual modules typically by the developer responsible for the module whereas integration testing is done when modules are combined together and the main purpose of integration testing is to make sure that the individual modules work together when they are combined. System testing is done, when all the modules are integrated into the system to make sure that the developed system meets all the system specifications mentioned in the design document[7].

A test item is a set of processes or pieces of code on which testing is performed and dynamic testing is the testing technique where test item is executed in order to perform testing [16]. Software download is a functionality that is being tested and the test item is a system with software download functionality which is a box car. The software and hardware components of individual ECUs are tested by the suppliers that develop them and are assembled together at the automotive platform developer. The team at the automotive platform developer does testing on the system and this testing can be done only if all the ECUs are available and connected together according to design requirements. Hence the test sub-processes are system testing and functional testing and the testing technique used is dynamic testing. The team at the case company, responsible for the thesis does system testing of software download process. Software and hardware components for the system are developed by different suppliers and are integrated at the case company. If some issues are found during system testing, the suppliers are asked to fix the issues during next integration and this process happens more than three times according to testing experts at the case company. The automation framework developed in the thesis is used for automating system testing of the software download functionality.

A test case is the lowest unit of testing activity beyond which a testing activity cannot be divided further. An example of a test case is as follows:

- **Test Case Name:** Parallel Software Download on all ECUs.
- **Test Item:** Boxcar with ECUs connected through different networks.
- **Testing Preconditions:**
  - All ECUs required according to the design are connected
  - All ECUs are in default mode
- **Test Inputs:**
  - Files to be downloaded into the ECUs.
  - List of files to be downloaded into the ECUs.
- **Expected output:**
  - Success: If all the steps of software download are completed for all ECUs.
  - Failure: If all the steps of software download are not completed for all ECUs.

2.6 Current Testing Process and HIL

Software download functionality is tested manually by the team responsible as shown in Figure 2.6. They use an internal flashing tool for flashing the software sequence into ECUs of a box car. Tools like CANoe and wireshark are used for logging the download. The files used for flashing are downloaded manually from an internal repository and loaded into the flashing tool as in the figure 2.6. The files are downloaded based on the part numbers read out from all the ECUs in the box car and the part numbers are read out by sending diagnostic requests to the ECUs in the box car.

System testing of software download typically involves several test cases with each test case having multiple test steps. A test case may involve testing of software download in multiple ECUs parallely or sequentially. There are several steps involved in software download sequence for any ECUS. These steps are defined by unified diagnostic services. Unified diagnostic services or UDS is a protocol that defines the behavior of an ECU within automobile electronics according to ISO 14229-1[39]. The protocol is followed in the internal flashing tool and the sequence of steps in the protocol can be perceived in the GUI of flashing tool. The network analysis tools such as CANoe and Wireshark capture the packets and signals involved during the flashing process. The details of the steps defined in the protocol can be perceived through CANoe or Wireshark logs.

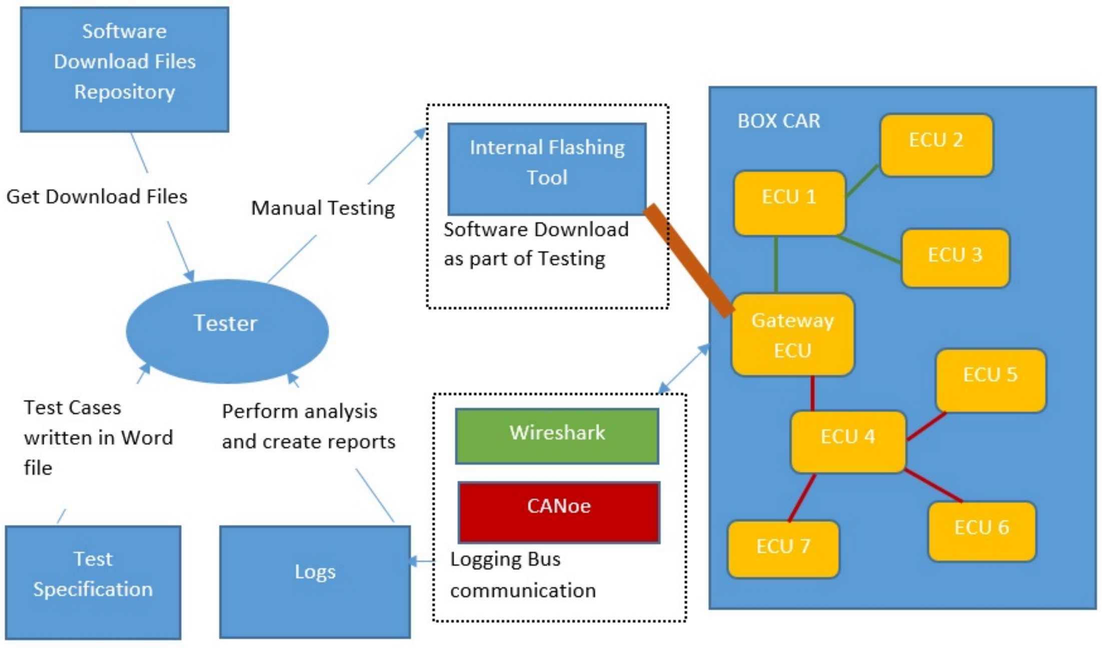

Software Download Files Repository
Get Download Files
Manual Testing
Internal Flashing Tool
Software Download as part of Testing
Tester
Test Cases written in Word file
Perform analysis and create reports
Wireshark
CANoe
Logging Bus communication
Test Specification
Logs
BOX CAR
ECU 1
ECU 2
ECU 3
Gateway ECU
ECU 4
ECU 5
ECU 6
ECU 7

Figure 2.6: Current Testing Process

Another organization that collaborates with the case company uses the internal flashing tool for testing the box car using hardware in loop (HIL). Hardware in loop(HIL) is a technique used to simulate test objects[12]. This technique can be explored to perform testing in a simulated environment. An HIL system uses a real time operating system and an input/output interface for simulating the real time environment for an ECU[13].The company also uses the flashing tool for sending diagnostic requests to test the box car.

2.7 Testing Tools

In this section we explain the state-of-the-art testing tools in automotive industry like CANoe and Wireshark. These tools are currently used in our case company for logging and analysis.

2.7.1 CANoe

CANoe is a software tool developed by Vector Informatik GmbH used in automotive industry for development and testing of various components of an automotive platform. This tool can be used for development, testing and analysis of single ECU or a complete network of ECUs. CANoe also provides a functionality to simulate environment for ECU like in an actual car. This can be very helpful for testing and analysis.

CANoe is a very useful tool when it comes to bus communication. Because of the vector hardware devices available which can be connected to the buses to be logged, the logging process becomes much simple. A configuration file has to be created for the specific networks to log the bus communication. There is also simulation setup while creating a configuration in which signal databases can be added. The signal databases are used to identify the signals logged during communication. Then to start the logging of the buses the measurement setup has to be started. This measurement setup contains different functional blocks and analysis windows which can be activated or deactivated for logging communication.

A test environment can be created in the configuration from the test setup in CANoe. CANoe gives the provision to add test modules which can be written in CAPL,.Net or XML. When a measurement setup is started the test module execution can also be started automatically. CANoe is used in software download testing and hence it needs to be integrated in the automation framework [50].

2.7.2 CAPL

CAN Access Programming Language or CAPL is an event driven programming language used in Vector tools like CANoe and CANalyser. CAPL is based on C-programming language. It has additional features which are used for development of CAN-based embedded systems. Another advantage of CAPL is that there are many built-in functions for analysing the diagnostic communication. A tester often looks at traces of specific signals when software download process executes. Test modules can be written in CAPL to capture signals during software download and a test report is automatically generated by CANoe. Hence writing scripts in CAPL would make it easier for a tester to precisely locate the point of failure by looking at the report generated instead of searching the logs. It is also easier to extend this test framework, as the same CAPL script can be re-used for all test cases with very minute changes [50].

2.7.3 Robot Framework

The Robot Framework is an open source, general purpose test automation framework which is mainly used for acceptance testing [22]. The concept of keyword driven testing makes it easier to create test data driving the test execution [24]. The testing capability of robot framework can be extended by test libraries programmable with ironpython, Jython and Python [23]. Ironpython is a programming language developed by integrating python and .NET framework and is used to access libraries built in both python and .NET. Jython is a programming language implemented in python for accessing libraries of java.
Robot Framework is implemented with Python and is operating system and application independent [23]. So it does not have to know the system under test but uses the test libraries to interact with it.

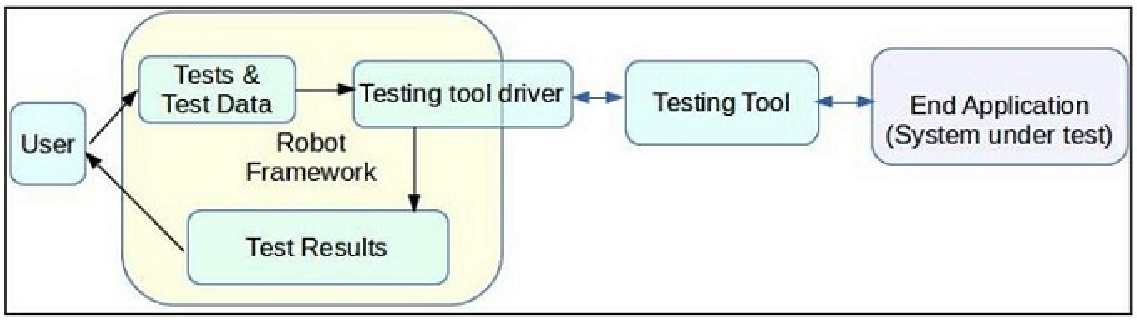


Figure 2.7: Robot Framework Architecture

2.7.4 High Level Overview of Robot Framework

The figure 2.7 from [22] presents a high level overview of robot framework. The different components of the framework are explained below:

- **Test Data:** The test data comprises of test and data files and folders as well as the contents of those which are used for test execution [22].
- **Test Results:** These are the end products of the tests, which are used to determine the results of tests as well as logs that can be used to assess various portions of the test [22].
- **Test Tool Driver:** This provides communication between the framework and the actual tools in place. It can be custom-tailored to meet specific requirements by the testing tool in place [22].
- **System Under Test:** This is the actual software that is to be tested for usability for its acceptance by the client or the end user [22].
- **Testing Tool:** This is the actual software that is used to perform testing on the system under test [22].
- **Robot Framework:** The robot framework is an arrangement which uses test data, invokes testing tools, stores the test results and interacts with the user [22].

Robot Framework's testing capabilities are provided by test libraries and these libraries can either use application interfaces directly or use lower level test tools as drivers [23]. There are many existing libraries, some of which are even bundled with the core framework [23] and these Libraries can be accessed through low-level keywords. A keyword identifies an operation or atomic action in test execution [42]. New Libraries can also be created using Robot Framework's library API which is quite simple and straightforward [23]. Robot Framework has three different test library APIs as stated below:

- **Static API** The simplest approach is to have a module or a class with methods which map directly to keyword names [23].
- **Dynamic API** Dynamic libraries are classes that implement a method to get the names of the keywords they implement, and another method to execute a named keyword with given arguments [23].
- **Hybrid API** This is a hybrid between the static and the dynamic API [23].

2.8 Continuous Practices

Continuous practices are concepts used mainly in software industry to enable development, testing and release of products with high quality, reliably and consistently. Continuous practices involves continuous integration, continuous deployment and continuous delivery. Continuous integration(CI) ensures development activities are integrated and tested, frequently and the process is often automated. Continuous deployment ensures that the application is always ready to be delivered to production and uses test automation, CI etc, whereas continuous delivery ensures a smooth delivery of the final product to the actual environment which is used by customers[27].

In automotive industry the development model is different from that of a typical software industry, because the final product is a mix of hardware and software modules and in most of the cases the different modules are developed by different vendors[28]. The case company provides hardware and software requirements to different vendors and all modules when completed are brought to the case company and integrated. Integration tests and system testing are done at the case company and the results are reported back to individual vendors for defect fixing or, addition or removal of functionality. Since system testing is done often, a quicker feedback can be provided to vendors if the testing can be done faster, consistently and reliably which lead to the idea of implementing a test automation framework.

2.9 Test Automation

Test automation is the process of automating one or many phases of a testing process, but in most of the cases it is concerned with automating test execution[16]. Automation is done based on certain tools, assumptions, concepts and rules, referred to as test automation framework[21]. Test-case design, test scripting, test execution, test evaluation, test results reporting, test management are some of the areas covered by test automation frameworks.

1.Test-case design: In general, testing is done by going through a requirement manually and derive test case. If the test-case design is automated, the automation generates test case with required inputs and expected outputs. However, this activity can be partially automated with testers manually deriving the test cases that cannot be directly executed by testing tool and use automation to make executable test cases. A complete automation of test case design activity can be time consuming and may have less return of investment with respect to automation costs [41].

Model Based Testing can be used for automating test case design in which, manually selected algorithms automatically generate test cases from a set of models of the system under test or its environment [40]. Test designers use standard modelling languages like UML to develop a test model that represents the expected operational behavior of the system under test (SUT) or its environment [40].

2.Test scripting: Test scripts are collection of instructions that needs to be followed for executing test cases [17]. In full automation, test scripting activity produces a set of instructions that can be understood by the test automation tool. However, this activity can also be partially automated with few scripts that need to be manually executed considering the possibility and cost of full automation [41].

There are many tools which can help a tester to generate test scripts automatically by giving test specifications through a Graphical User Interface. There are record-and-playback and keyword-driven automation techniques which can be used for this purpose. In the record-and-playback technique, testers record a test scenario (test case) by interacting with the GUI of the system under test (SUT), while the tool automatically records the test log as a test script in the background [41]. With the technique of keyword-driven test, test scripts can be generated automatically according to the provided keywords [42]. Keywords or action words are defined using a table format or spreadsheet for each function that we would like to execute.

3.Test execution: In general test execution involves loading the testing tools, providing inputs, running the test item and recording the behaviour of the system under test for analysis. This activity can also be partially automated by considering the possibility and the cost for full automation [41].

There are many powerful tools for automated text execution for example, Selenium [44], JUnit [46] and Testdroid [45]. Also the state-of-art integrated development environments(IDE) like Eclipse, Visual Studio include tools that automatically generate empty test classes and methods [43].

4.Test evaluation: Evaluation is an inevitable component of testing process since it determines whether a given test case has passed or failed based on the required output of a test case. There are mainly three approaches for test evaluation as below:

- Manual assessment can be done where tester can decide whether test case has passed or failed.
- The developers incorporate (hard-code) test evaluations as verification points (assertions) in the test code [41].
- The developers build "intelligent" (learning) test oracles, using machine learning [47]. Test oracles are used as a complete and reliable source of expected outputs and a tool to verify the correctness of actual outputs [48]. A comparison between actual and expected outputs is made to verify the correctness. The process of finding correct and reliable expected outputs is called oracle problem [49].

5.Test results reporting: In most testing processes, this is the last step where testers track and report the status of test case for defect fixing. There are several test automation frameworks and libraries available that can automate this activity.For instance, the NBug .NET library automatically creates and sends bug reports and crash reports [41].

There are many other test engineering activities that can be automated, but the activities listed above are important because they form the crux of testing process.

In this thesis, automated test framework aims at developing a framework which can be used to execute the test cases by loading the inputs required for a test case and automatically log the test analysis in CANoe or wireshark. To simplify the analysis, the thesis also intends to develop scripts in CAPL. The idea is to completely automate a test case and show how other test cases can be automated similarly.

3
Research Methodology

3.1 Research Questions

In the case company integration of software files and testing is done at platform level. As the integration is done at the platform level all the software and hardware deliverable have to be received from the suppliers to perform system testing. This can cause delay in the integration process and also cause delay in giving feedback to suppliers. Further adapting the continuous integration process could reduce this feedback time. A research question in this context would be as follows:

- **RQ1:** What are the most pressing challenges of continuous integration on Automotive platform level?

Testing can be automated in order to improve the amount of time consumed after each integration point. There are different projects running in parallel on an automotive platform. A test automation framework can automate different aspects of testing and the research question here is as follows:

- **RQ2:** What are the factors to be considered while developing a test automation framework in automotive platform development?

The thesis focuses on implementing test automation framework to enable continuous integration. From the interviews conducted, literature review and implementation of test automation framework, it may be possible to derive some solutions to overcome some of the challenges faced by automotive industry in adapting to continuous integration. A research question here is as follows:

- **RQ3:** What are the promising solutions to overcome the challenges faced by platform developers with respect to continuous integration?

A study applying the design science research method contains a strict process of constructing artifacts to undertake the mentioned problems, measuring the pattern and explaining the project results to the pointed audience [55]. The expected outcome of this thesis is to automate different phases of testing life-cycle to improve continuous integration, which can be considered as a technical artifact. The fundamental principle of this design science research methodology is that knowledge and understanding of a design problem and its solution are acquired in the building and application of an artifact [2]. As developing and evaluating a technical artifact is one of the major contribution of this thesis, choosing design science methodology would be helpful. This would enable practitioners also to take advantage of the benefits offered by the artifact and it enables researchers to build a cumulative knowledge base for further extension and evaluation [2].

One of the contributions of the thesis to scientific community would be to provide solutions for addressing continuous integration challenges on an automotive platform. A problem solving methodology with focus on developing and evaluating a design artifact like design science would help in identifying the challenges of continuous integration and also help to evaluate the artifact. The design science research methodolgy is explained in detail in the next section. Further the procedure used for evaluation is also explained.

3.2 Design Science Research

The methodology chosen in this thesis is based on design science research. Design science emphasizes the connection between knowledge and practice by showing that we can produce scientific knowledge by designing useful things. Design science research as a research methodology is more concerned with artifacts than other branches of science [1]. We would require this type of problem solving paradigm to efficiently and effectively accomplish this thesis. There is a problem which is not so well defined at the case company. The thesis investigates, identifies and defines the problem. The case company uses a set of tools and resources to perform procedures which form the premise of the problem. The idea is to design a solution using these resources and evaluate the solution. The problem is divided into sub-problems and each sub-problem is investigated. After investigation, a solution is designed, implemented and evaluated for each sub-problem. The approach taken to solve the problem is consistent with design science research. This methodology consists of different regulative cycles. A regulative life-cycle is a framework of the research method that solves the problem in logical structure [1]. Each regulative cycle consists of 4 or 5 phases in general [1, 2, 3, 4]. The outcome of each regulative cycle is used as the basis for the next regulative cycle.

The different phases in design science research are as shown in the figure 3.1. The goal of each iteration for the thesis is also given in the figure. In this thesis we automate three different phases of test automation that is test execution, test analysis and test scripting. We chose these areas of focus based on the problem investigation which was done in form of interviews and documented as user stories. The automation is mainly done to achieve continuous integration process on automotive platform.

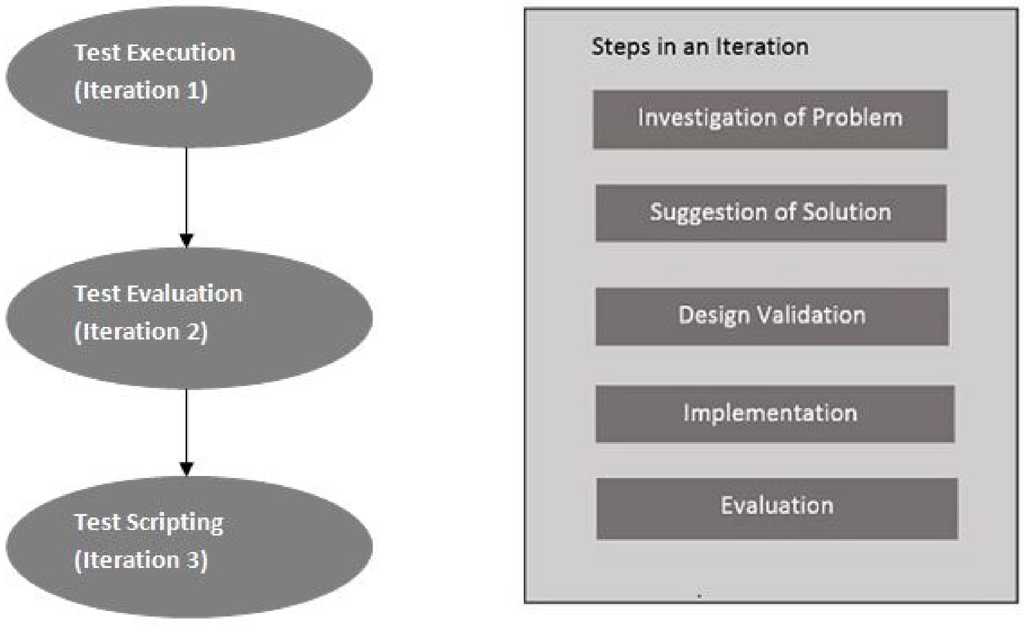


Figure 3.1: Design Science Research Methodology

3.2.1 Investigation of the problem

An awareness of an interesting research problem may come from multiple sources including new developments in industry or in a reference discipline. Reading in an allied discipline may also provide the opportunity for application of new findings to the researcher's field. The output of this phase is a proposal, formal or informal, for a new research effort [2]. In this phase, we analyze the current situation of the company with respect to development cycle. This is done using a series of knowledge transfer sessions, demo sessions, document review and tutorial sessions along with interviews. The goal of this phase is to describe the problem, to explain it, and possibly to predict what would happen if nothing is done about it [2]. We have further investigated the problem in form of user stories(presented in section 4.1) that describe how a specific role would benefit from test automation.

3.2.2 Suggestion of Design

Suggestion is essentially a creative step wherein new functionality is envisioned based on a novel configuration of either existing or new and existing elementst [2]. In this phase we will design a solution based on literature review, knowledge of system gained from the company and the knowledge of networks in cars. The design solution can be suggested after continuous experimentation of possibilities on test item. Exploring related tools like CANoe, internal flashing tools and test scripts would be helpful in suggesting a design. Also thorough literature review of solutions proposed for similar research problems has proven to be helpful.

3.2.3 Design Validation

Design validation is a knowledge task in which we ask whether the specified design, if implemented correctly, would indeed bring stakeholders closer to their goals [3]. In this phase we validate the design by discussion with the company supervisor and the Chalmers supervisor along with available expertise at the company. If the design is acceptable to both parties, we can go ahead to the next phase. If one of the parties disagree, then we would use their feedback to improve the design.

3.2.4 Implementation

The goal of implementation is to develop artifacts based on the design suggestions [2]. The design suggestion from previous phase is implemented in this phase by using the resources available at the company such as box car, tools from different vendors and using appropriate programming languages mentioned in the design phase. In case of problems or obstacles regarding the use of specific tools the guidance of company supervisor can be availed. This is the longest phase of design science methodology with respect to the thesis.

3.2.5 Evaluation

Once constructed, the artifact is evaluated according to criteria based on the awareness gained in the problem phase [2]. In this phase we evaluate the implementation by conducting a series of interviews with practitioners and end users and assess the improvement in the development process. The feedback received can be used to evaluate the impact of the solution on current development process. The interviews are also used to gather suggestions for improvement of the implementation. Based on the user stories we interview the same intended people about on whether their problem has been addressed, and if so, how were they addressed. The outcome of the evaluation phase can be utilized for the next iteration.

The above cycle is iterated to further improve the development process by finding additional strategies to solve the problem. The thesis is done in three iterations. In the first iteration, the test execution is automated. Since the objective is to make a generic framework that can be used in different projects,the design should focus on identifying the generic requirements.

In the second iteration, the test evaluation is automated to generate high abstraction level reports that can be easily analyzed by a tester. In problem investigation, interviews are conducted to analyze what makes it easier for a tester to analyze test reports. Based on this test report generation is modified using existing CANoe tool's reporting mechanism. After every iteration interviews are conducted to evaluate the solution.

In the third iteration, GUI is designed which is used to set up all the files and folders for storing logs, reports, test cases and test-suites. The GUI is also used for connecting to the box car, form executable test cases and to invoke test execution. In third iteration, the framework takes its final form where there is a test-suite to store all the test cases and a mechanism to invoke test execution for each test cases according to an order defined by a test engineer.

3.3 Interviews and Surveys

We have conducted interviews and surveys as part of evaluation in this thesis to collect qualitative data. The surveys are used in research in order to give a strong evidence for the research done [56].

Interviews can be done in two ways. Individual interviews can be conducted where in a single person is interviewed with a common agenda. Whereas, focus group interviews commonly known as group interviews are conducted by gathering people from similar background and discussing on a topic which is set by the researcher [57].

Interviews can be semi-structured, fully-structured or unstructured [59]. In fully structured interviews the questions are determined and ordered. Whereas in semi-structured interviews, the interviewer tries to elicit information as much as possible through conversations though he or she has to prepare some predetermined questions [58]. While in an unstructured interview, the questions are neither determined nor ordered. The questions would be mostly conventional for such kind of interviews [59].

We have conducted a demo and a survey with test engineers at case company, as part of evaluation of the first iteration. This survey questions can be found in Appendix A. Structured interviews were conducted as part of evaluation of second and third iteration of the thesis.

4
Iterations

During planning phase of the thesis, we conducted a series of interviews and knowledge transfer sessions to study the scope of improvement in terms of system testing process followed in the case company. After these discussions we decided to complete the thesis in three iterations. This section describes the activities done in each iteration.

4.1 A setup for Test Execution

The team responsible for software download in the case company which provides base technology support as well as a platform developing integration house. They want to speed up their development cycles, so that they can serve OEMs better to use their platform. The platform should not only include technical components, but also testing infrastructure that makes it easier to further develop the platform and also ease integration in a particular car project. The objective of these iterations is to develop an automation strategy for system testing of software download. The aim for this iteration is to develop an automated setup for test execution for one test case which can be further extended.

4.1.1 Investigation of Problem

In this phase of the iteration, the system development process followed in the case company has been thoroughly studied to dive deep into the problem. This has been done by conducting interviews with different stake holders of software download. Based on the input we received from the stakeholders we spotted some tasks(T) as shown in the Figure 4.1 performed manually by test engineers which consume lot of their effort and time and have overall negative impact. The tasks are further explained in terms of sub-tasks (ST). These sub-tasks are further elaborated in terms of user stories (US). The improvement is done mainly in three phases namely test scripting, test execution and analysis.

T1. Test Scripting: The software download files that are from different suppliers are received at every integration point and a system testing cycle is conducted. We conducted a series of interviews which helped us to frame user stories further explained in this section. These user stories give a big picture of the existing integration process, as well as all the tasks that can be automated.

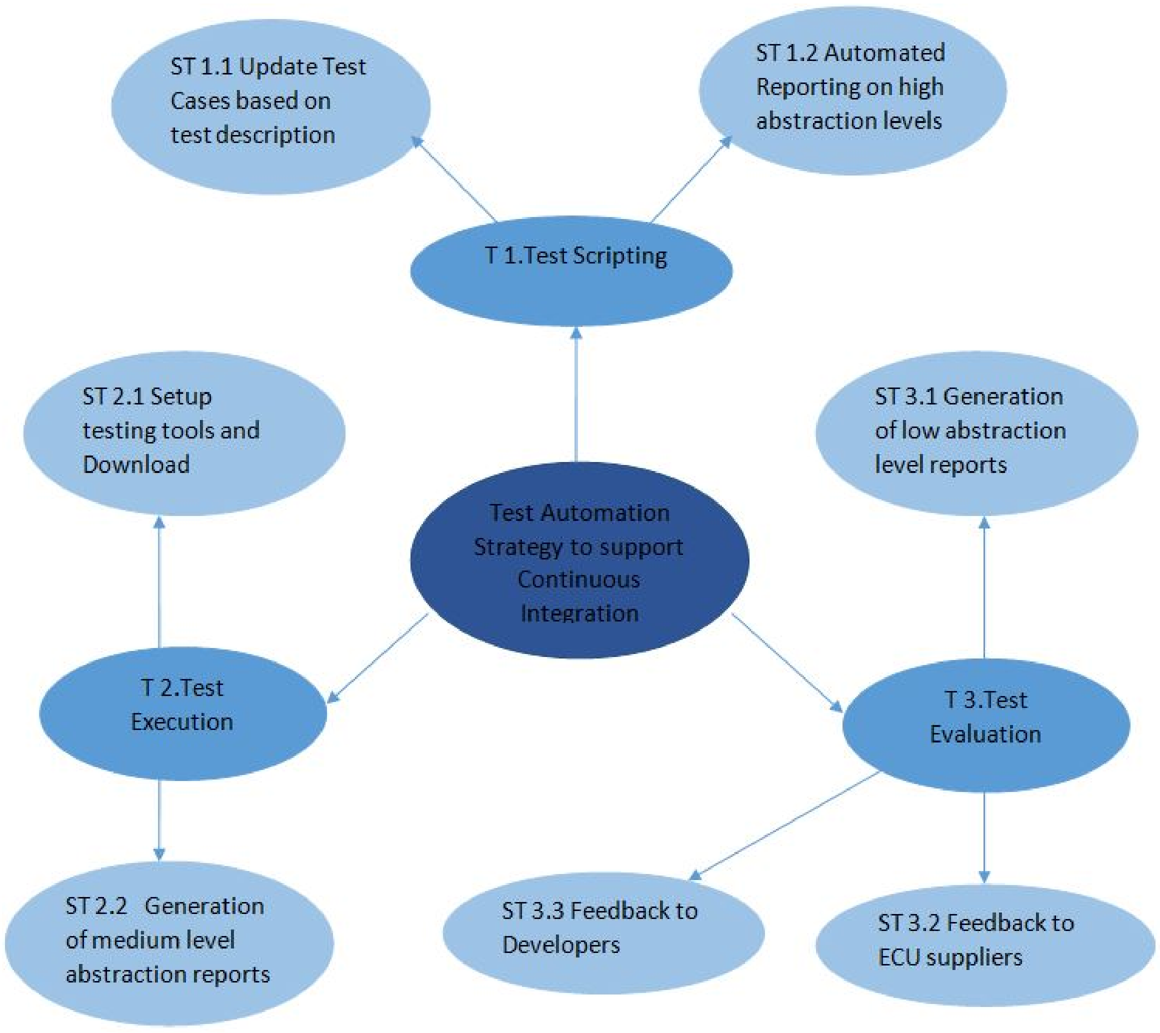

ST 1.1 Update Test Cases based on test description
ST 1.2 Automated Reporting on high abstraction levels
T 1.Test Scripting
ST 2.1 Setup testing tools and Download
ST 3.1 Generation of low abstraction level reports
Test Automation Strategy to support Continuous Integration
T 2.Test Execution
T 3.Test Evaluation
ST 2.2 Generation of medium level abstraction reports
ST 3.3 Feedback to Developers
ST 3.2 Feedback to ECU suppliers

Figure 4.1: Phases of testing lifecycle that can be automated to improve the continuous integration process in the case company are represented as different tasks T1, T2, T3. Further the task T1 test Scripting has other sub-tasks such as automated generation of reports with high abstraction levels. Similarly other Tasks T2 test execution and T3 test evaluation also have sub-tasks.

ST1.1 Update Test case based on test description In the current test setup as there is no proper test suite, the parameters of software download such as the number of iterations, ECUs etc; have to be manually changed in the flashing tool based on the test case description. This consumes a lot of time for the test engineer and also can be error prone. At every integration point a different change sets of software, data or test description are integrated. The effort for testing the integration of change sets can be considerably reduced. This will speed up giving feedback to developers and ECU suppliers.
US1.1:"*As a test engineer, I want automated update of the testing parameters for software download based on the test case description so that it reduces the effort for integrating different change sets and thereby fastens the feedback cycle.*"

ST1.2 Automated Reporting on high abstraction levels Currently each test case has to be manually verified as to whether it has passed or failed as there is no proper test setup. Automating the generation of standard higher abstraction level report can save a lot of time of the test engineers which can be utilized to focus on root cause analysis. This high level report is also helpful for HIL test engineers as they only need to know whether test case has passed or failed. Also further testing and integration of new changes will depend on this. The new change sets can be rejected if the test case fails.
US1.2:"*As a test engineer, I want to generate a report that allows me to see directly whether a software download test case has passed or failed so that I can focus on analyzing root causes for test failure instead of determining the test result.*"

T2. Test Execution Currently the system testing process involves flashing of download sequence through the internal flashing tool and also logging of bus communication through CANoe and Wireshark. Few test cases are executed more often than other test cases. Performing parallel and queued software download on multiple ECUs is one of the most frequently performed test cases.
ST2.1 Setup testing tools and download sequence: Currently both the internal flashing tool and logging tool have to be started for execution of a test case. Also the download sequence is very important for the test case to pass. Correct sequence has to be set. If this setting up tools and download sequence is automated then the it would save a lot of time and also reduce the manual errors. This would considerably reduce the feedback time for the next integration.
US2.1:"*As a test engineer, I want automated setup of testing tools and download sequence during each test case so that reduces the manual errors and speeds up the test execution. This in turn would improve the feedback cycle.*"

ST2.2 Generation of medium level abstraction reports: Currently a medium level abstraction log is available in the flashing tool which gives the information of progress or failure in different software download steps as shown in Figure 2.5. This is further used to track failures in logs based on the step where software download failed. By standardizing the automatic generation of reports along with logs will help in speeding up the evaluation. US2.2:"*As a test engineer, I want automated* *generation of medium level abstraction reports tracking different steps of software download so that based on the report further failure can be tracked down in logs. "*

T3. Test Evaluation: The different buses communication between different ECUs is logged by setting up testing tools such as CANoe and Wireshark. This is mainly done to capture and analyze signals in order to determine the diagnostic communication during the software download. This helps to analyze root cause of download failure.

ST3.1 Generation of low level abstraction reports In the current test setup whenever software download fails the test engineer has to manually go through the test logs in CANoe and Wireshark to find the root cause. However automating generation of a low abstraction level report with only the relevant logs would save a lot of time. This would help in giving valuable feedback as well as remove blockers.
US3.1:"*As a test engineer, I want automated generation of low level abstraction reports based on test execution and signal names so that the root cause of failure can be found out which would help in efficient fix of a failed change set.*"

ST3.2 Feedback to ECU suppliers: Based on the test report the ECU owner has to identify issues in ECU and report it to the ECU supplier. It is imperative to give feedback before definite timelines to the suppliers in order to get the issues fixed by next integration.
Based on the reports and logs issues are identified. Relevant issues and concerns are discussed with supplier. This issues are usually fixed by next iteration. The function owner needs to know if a particular service is supported or not by the ECU. A tester has to perform software download again or check the previous logs to determine this. Based on this the function owner has to determine whether further changes can be done or not to ensure the working of functionality in aftermarket. There is also a timing constraint which has to be considered by the function owner as late changes cannot be done in lower layers.
US3.2:"*As Function owner of software download I want to understand whether a particular bootloader would support the use case scenario so that I can ensure that the function will work in the end by taking appropriate action. However due to lack of proper reporting mechanisms confirming this feasibility causes some amount of delay.*"

ST3.3 Feedback to developers: Developers require a proper feedback mostly downstream from the test engineers in order to evaluate their work. However currently as the test engineers spend a lot of time writing a report, this becomes difficult to give feedback to developers before a specific timeline.
US3.3:"*As a developer, I want to get feedback downstream so that I know whether I am developing in the right direction and also the ability to test intermediate versions of my code quickly and at a low cost. For that, I need detailed information from testers.*"

Based on the above user stories as shown in Figure 4.1 we felt that we can proceed with creating an automation framework and implement one test case to give them a clear picture. Based on the interviews we decided to automate the test execution for this iteration. We identified the most common test case and decided to automate it.

4.1.2 Suggestion of Design

After understanding the problem and exploring different alternatives we figured out three approaches to build the automation framework. They are as below:

- Using existing internal flashing tool APIs for building the new tool.
- Extending the currently available scripts of another internal flashing tool which currently can perform software download only one ECU at a time.
- Converting VBF files to BIN files and creating test cases in CANoe.

During document reviews and knowledge transfer sessions we learned that the existing flashing tool has APIs which can be used for building a new tool. Another alternative was to extend the currently available scripts of another internal flashing tool. This tool currently has scripts only for flashing on single ECU at a time. This extension would consume a lot of time and would also be difficult to extend for other test cases later by the test engineers. Another option was to convert VBF files to BIN files and create test cases in CANoe. However this is also a cumbersome process and also is unfeasible to extend.

We suggested a plan to build an interface using the existing flashing tool's APIs which would take test cases as input, execute them one by one and generate logs. The CANoe application can also be launched using COM server interface built for the application. COM is a standard defined by Microsoft for the communication between different software components [53]. Different programming languages can be used to create such components that can be built by different software developers, independently from each other [53]. The idea is to combine both the interface and the COM server. The APIs of the flashing tool were exposed as C# classes and dlls. The concept of using COM server interface to access CANoe is described in the tool documentation as C# snippets. This lead us to the thought of implementing an application in C# that can combine both COM server and the interface built using APIs.

4.1.3 Validation of the Design

In this phase we validated the design by conducting interviews with testing engineers who has the expertise of CANoe and the flashing tool. They were a bit skeptical about the usage of APIs since it was not properly documented. However, they reassured us with the idea of contacting the API responsible from another organization that collaborates with the case company.

4.1.4 Implementation

During implementation, the first step was to try the APIs with the box car. To do this, we wrote a simple console application in C#. After trying most of the APIs, we analysed APIs in terms of the sequence of steps described in the background section to connect to a box car. The sequence of calling the corresponding APIs were organized according to the steps. The objective of the console application was two fold. First task is to connect to the box car and to do diagnosis. i.e., to send diagnostic requests and receive responses from the box car successfully. This has been built and tested against multiple diagnostic requests. Next step was to implement software download to the box cars using the APIs. During the build we faced several challenges. The dlls were coded in C++, but the application was built in C#. Due to the incompatibility of datatypes between languages, a significant amount of time and effort have been put in understanding how to call functions built in C++ from C#. We also faced some challenges in calling APIs responsible for security access. These problems have been solved based on intuition combined with trial and error.

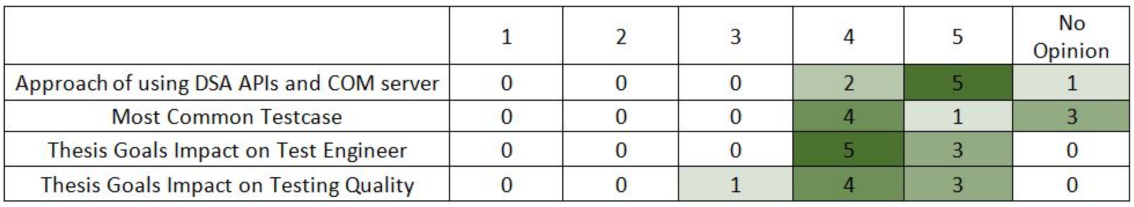
In the second phase of implementation, we built another console application to call the CANoe application, load configuration and to start measurement. We combined both these applications to simultaneously execute software download while logging the network transactions in CANoe.

4.1.5 Evaluation

As part of the evaluation we conducted a demo in our case company for a team of 12 test engineers. As part of the demo we explained the user stories we put forth during problem formulation of this iteration. Based on the use cases we described the major problems we identified in the current testing process and our thesis goals. We also presented the approach we proceeded with in our first iteration in detail. At the end of the presentation we handed over survey papers A where we put forth some rating questions based on the use cases we formulated. We received 8 responses from the test engineers which are summarized in the form of heat map as shown in figure 4.2. We also put some interview questions like suggestions for developing a better test setup, enhancement of our first iteration in our survey paper.

The first row in the heat map is the rating received for the approach of using DSA APIs and COM server. Here 1 is the lowest rating representing not helpful and 5 is the highest representing very helpful for all the questions. Another question was formulated based on the user story that execution of parallel and queued download on all ECUs is done several times. The question was to rate how much our thesis is helpful for executing this most common test case. The third and fourth questions in the heatmap were the extent to which our thesis goals would be helpful in improving the testing quality.

Apart from the ratings we also received written and verbal suggestions for our next iteration.



	1	2	3	4	5	No Opinion
Approach of using DSA APIs and COM server	0	0	0	2	5	1
Most Common Testcase	0	0	0	4	1	3
Thesis Goals Impact on Test Engineer	0	0	0	5	3	0
Thesis Goals Impact on Testing Quality	0	0	1	4	3	0

Figure 4.2: Heat Map of Survey Responses from Test Engineers. Here 1 is the lowest rating representing not helpful and 5 is the highest representing very helpful for all the questions

4.2 A setup for Test Evaluation

The objective of this iteration is to develop analysis scripts to ease the reporting.

4.2.1 Investigation of Problem

Based on the problem investigation done in first iteration analysis of CANoe logs seemed to be a manual task which could be automated. There were many user stories for this manual task which reflected concerns of different stakeholders. However after the completion of first iteration some interviews were conducted to understand the concerns to be addressed in this iteration. Some additional user stories are presented in this subsection.
T3. Test Evaluation: After the first iteration, the CANoe logging is triggered with the software download. However a proper test framework is not developed to track the progress of each test case.

ST3.4 Test Framework Currently there are no test cases or test modules in CANoe which would be triggered once the download starts. There is no test modules or test cases which would track the progress of software download or logging. After the completion of first iteration although the tester can perform download and logging simultaneously the test cases have to be still manually read from a word document. The test engineer has to confirm manually whether each test step has passed or failed.
User Story::"*A*s a test engineer I want a proper test framework so that the software download progress can be tracked as part of a test case. CANoe has a test environment where test modules can be created and test case can be tracked. If this framework can be utilized by capturing the software download signals using analysis scripts, it could be helpful to improve the feedback cycle."
In this phase we decided to automate the test analysis part as this seemed to be the major concern of the stakeholders after the completion of first iteration.

4.2.2 Suggestion of Design

The CANoe tool comprises of a component for testing. Test modules in CAPL, .Net or XML can be created which can be run by system or user during measurement. Test cases can be written as part of this test module. We decided to create test modules in CAPL, as it has maximum support for accessing the communication channels in the measurement setup.

In the current system testing process, the Flexray and CAN signals are captured using the vector CANoe with VN8900 series interface. This VN8900 interface family is real time hardware which can be used for data monitoring, test execution or system simulation. Configuration of the simulation and evaluation are performed on a standard PC (CANoe), while the simulation and test kernel are executed on the VN8900 interface[20].
So we felt that using the same interface and setup and writing scripts for in CAPL is a good approach. However for ethernet signals, currently the logging is done in Wireshark. In order to log the ethernet bus another interface in VN5610 series is required to be connected through the VN8900 interface.
For logging the ethernet signals we had two alternatives:

- Logging ethernet signals in CANoe by using two interfaces.
- Capturing ethernet traffic in Wireshark and use the existing scripts to filter software download data.

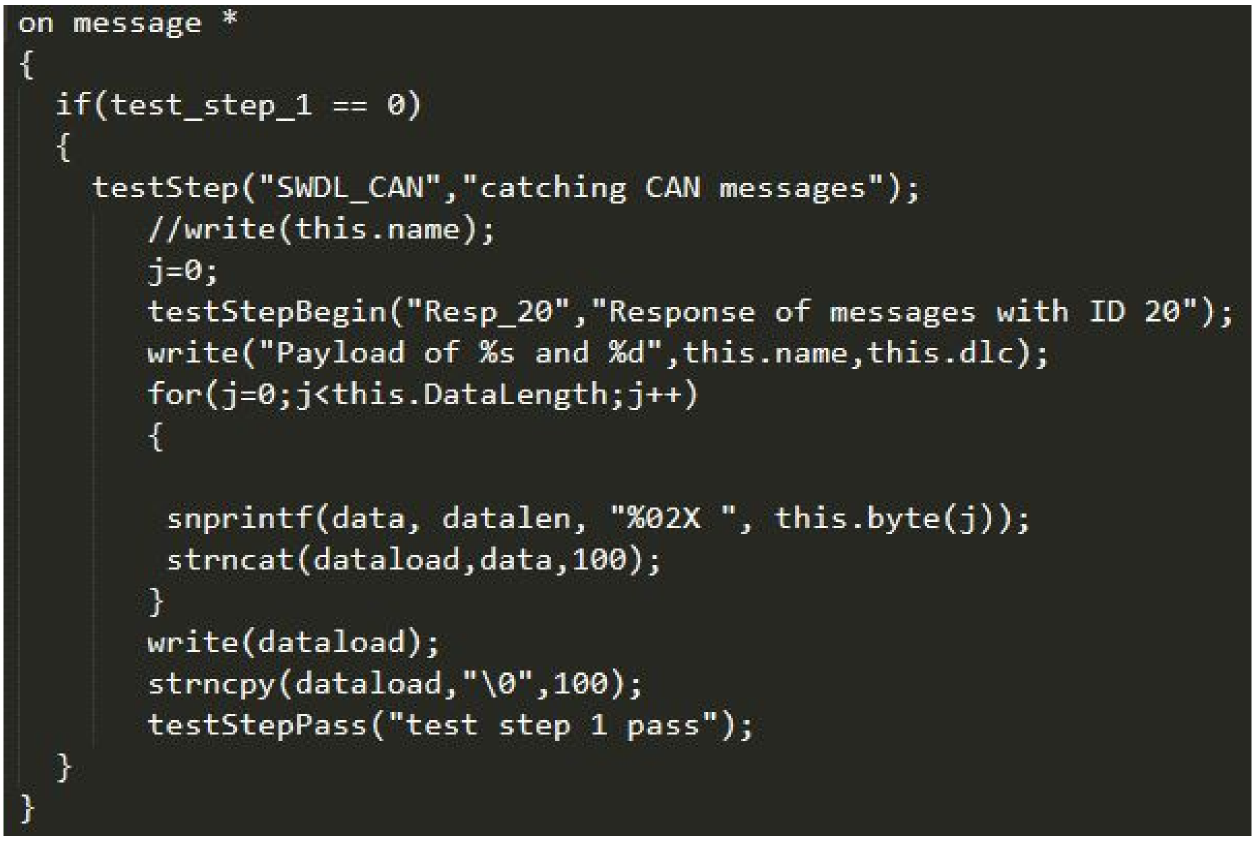
After exploring both the options we felt logging ethernet signals in CANoe and writing analysis script to capture software download signals is better option. As this would maintain an uniformity in the application there would not be two set of separate reports that the test engineers have to go through.

4.2.3 Validation of Design

In this phase we had a discussion with our supervisor in the case company and afew test engineers. They also felt that proceeding with logging ethernet signals in CANoe would be a better option. In their opinion CANoe has a good reporting mechanism which could be utilized to generate more concise and clear reports.

4.2.4 Implementation

In this phase, the first step was to write analysis scripts for capturing flexray signals. There are databases available for the software download signals of both CAN and Flexray. Once the databases are loaded in the simulation setup these signals can be accessed by names in the test module. In CAPL certain event procedures are available which create an event every time a particular signal is captured. For Flexray the event procedure was used and the frame was identified based on the name of the signal in the database. Then the payload data of the frame was accessed by usage of general functions and keywords available in CAPL.
The next step was to capture CAN signals. Similar to flexray even CAN has an available database useful for identifying the software download signals. The database signals can be accessed by names in the test modules once they are assigned to the appropriate channels in the simulation setup. The event procedure as shown in figure 4.3 was used to capture signals either based on the name or the identification number. The payload was accessed by using the keywords available in CAPL. Each message captured is done as a part of the test step.



```
on message *
{
  if(test_step_1 == 0)
  {
    testStep("SWDL_CAN","catching CAN messages");
       //write(this.name);
       j=0;
       testStepBegin("Resp_20","Response of messages with ID 20");
       write("Payload of %s and %d",this.name,this.dlc);
       for(j=0;j<this.DataLength;j++)
       {

        snprintf(data, datalen, "%02X ", this.byte(j));
        strncat(dataload,data,100);
       }
       write(dataload);
       strncpy(dataload,"\0",100);
       testStepPass("test step 1 pass");
  }
}
```

Figure 4.3: These event procedures are used to capture signals either based on the name or the identification number.

Further for ethernet signals as database was not available for the download signals. However the software download messages have an additional udp payload other than the ethernet data. So all the signals having udp payload could be captured as they were all software download signals. So when using the event procedure, we capture all ethernet packets. The udp payload could also be accessed as part of ethernet data. So the packets which only have the udp data are captured.
In CANoe the test modules can be configured to run automatically by the system once the measurement starts. So we configured test modules, so when the measurement is triggered externally through the COM interface the test cases will be run automatically. Both reports and logs will be stored to the specified destination.

4.2.5 Evaluation

Evaluation has been done using interviews while demonstrating the report generated. The following points were assessed during the interview.

1. How useful is the report in identifying the cause of failure or success ?

   The logs from CANoe can be filtered more to pin point the error signal, this would be a feasible but is a time consuming process.

2. Does the automated report generation save time for a test engineer ?

   The report saves time for a test engineer which otherwise has to be done manually by going through logs generated by CANoe.

3. Can the report be sent as a feedback to vendors directly?

   If details about test cases and the ECUS involved in the download process can be included in the report, it can be sent directly to vendors as feedback.

4. Suggestions to improve the report

   It was suggested to conduct research on how to customize the content of the report to highlight details about the test cases and the ECUs involved in the download.

4.3 Test Scripting

The objective of this iteration was to automate test scripting which basically provides set of instructions for executing a test case.

4.3.1 Investigation of Problem

Based on the problem investigation done in first iteration, some concerns were reflected about the test case generation. After the second iteration was completed

we conducted few interviews to understand the expectation for the test framework. Below are additional user story presented based on the interviews done in iteration 2.
T1. Test Scripting After the first and second iteration there is a provision to start the both flashing and logging simultaneously and analysis scripts generate reports containing relevant communication. There is also a way to track the software download through the console application. However currently there is not a proper User Interface to insert locations of download files and CANoe configuration.

ST1.3 Front End Development By having a front end the test engineer can upload input the parameters of software download and location of download files. Also a way to upload the link of CANoe configuration.
User Story::"*A*s a test engineer I need a user interface to input software download parameters so that I can start download sequence and log bus communication at once."

4.3.2 Suggestion of Design

After problem investigation based on the user stories we identified that the following are the requirements identified in accordance with [25] for the test case management and generation. They are as following:

1. Faster test scripts generation
2. Re-usability of the test code
3. Scalable for future test requirements
4. Ease of maintenance
5. Extended Reporting Capabiity

Based on these requirements we felt that making use of robot framework which incorporates a keyword-driven test approach would be a good choice.This framework was built from initial stages to be a tool to test engineers so that they can create automatic test cases. By using available standard test libraries, test cases can be created without programming knowledge. Besides robot framework is a generic framework where test scripts can be written in different programming languages. There is also a good reporting mechanism which generates result reports and logs in html and xml format.

Further to ease the testing process we decided to create the front end User Interface using Qt framework. This is a mature GUI framework. It is cross platform, in the sense that the Qt class library is implemented for several different operating systems [26]. By using Qt the application code can be structured in independent, reusable components [26].

As we decided to design the User Interface in a way it fetches all the software download parameters we also wanted to automate the test script generation process. We decided to use the PySide binding of Qt Framework as Python's built-in high level language and dynamic typing would be helpful. Also the dictionary types are very useful for scripting.

Initially we thought of proceeding with developing the User Interface using .Net Framework. However as we wanted to have a test setup using Robot Framework using Python binding with Qt framework seemed like a better option.

4.3.3 Validation of Design

In this phase we discussed the design suggestion with the supervisor at company. After conducting some interviews with test engineers we received a positive feedback about the design suggestion. Our supervisor advised us to create a technical document about the software and packages required to install, so that it could be easier for the test engineers to setup the environment. Based on the inputs we received from the test engineers we understood that as there are lot inputs to be given to the framework, having it to be fetched from an excel sheet would be a better option.

4.3.4 Implementation

In the initial part of this phase we had setup robot framework with iron python. Then we created a test suite which had a test case to trigger software download and CANoe measurement. This was implemented easily by the usage of available keywords from the built-in libraries. High level abstraction reports and logs were generated. As currently the console from where the testsuite is run only shows the test case pass or fail, we made a batch file to trigger the software download progress in another console. As there are a lot of inputs to be given to the test suite we modified the test script and the software download console application to take inputs from a xml file.

As the second part of this phase we developed the user interface using the PySide binding of Qt Framework. The figure 4.4 shows the user interface developed. The different fields are as explained below:

- **CANoe:** The first field of the widget is used to select the CANoe configuration file which is used during the time of measurement.
- **SWDL Log:** This field is used to select the location of the folder to store the software download logs. This are the logs generated on a second layer of abstraction.
- **PIN File:** This field is used to select the file containing the pins and addresses of the ECU's which will be used for security access during software download.
- **VBF Folder:** Through this field the location of the VBF file or the software download files is given. These files are arranged in correct sequence and a VBS file is created which contains the links to all the files in correct sequence.
- **Test Cases:** This field is used to select the excel file through which the software download inputs are given.

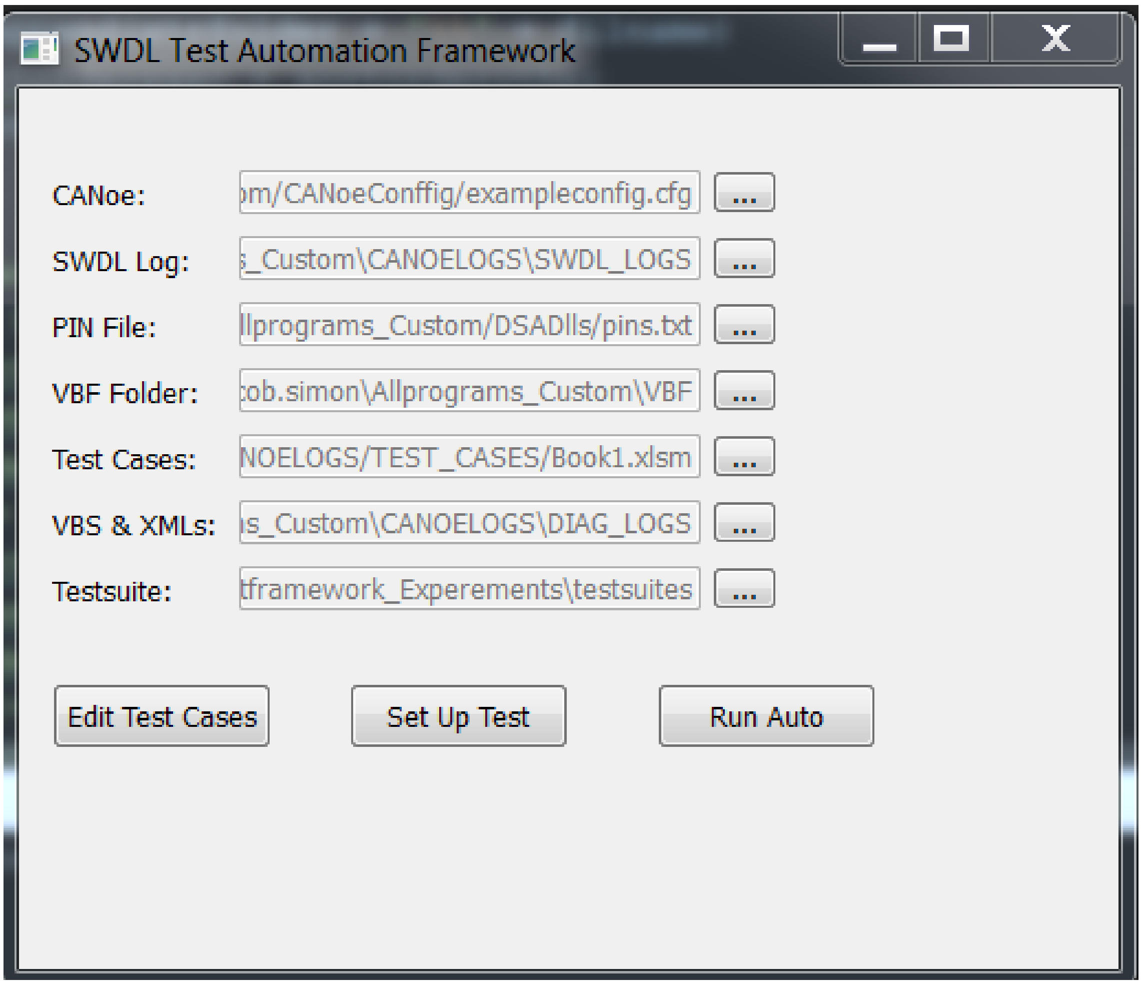

SWDL Test Automation Framework
CANoe:
ɔm/CANoeConffig/exampleconfig.cfg
...
SWDL Log:
s_Custom\CANOELOGS\SWDL_LOGS
...
PIN File:
llprograms_Custom/DSADlls/pins.txt
...
VBF Folder:
:ob.simon\Allprograms_Custom\VBF
...
Test Cases:
NOELOGS/TEST_CASES/Book1.xlsm
...
VBS & XMLs:
ıs_Custom\CANOELOGS\DIAG_LOGS
...
Testsuite:
tframework_Experements\testsuites
...
Edit Test Cases
Set Up Test
Run Auto

Figure 4.4: User Interface of the tool

- **VBS & XML:** This is to upload the desired location where the VBS and XML generated will be saved based on the inputs given through excel sheet.
- **Testsuite:** This field is used to specify the location of the testsuite which contains the executable test cases which are automatically generated.

As seen in the figure 4.4 there are three push buttons Edit Test Cases, Set Up Test and Run Auto. On clicking the Edit Test Cases button the excel sheet opens which can be updated for the software download parameters. Different parameters for software download such as parallel, queued, number of iterations, ECUs, preprogramming, post programming, complete and compatibility check can be chosen in the xls sheet. The column named include tests can be used to select a particular test cases among a group of test cases for automated download. The excel sheet is with a XLSM file extension which is basically a macro-enabled workbook file.

The Set Up Test button will go through each test cases in the excel sheet, generate vbs files, xml files with software download parameters, and generates executable test cases for robot framework. The parameters from the excel file are updated into a XML file. The robot framework uses test cases and each of these test cases has xml file as the input. The executable test cases are generated using a reference file since the test cases are similar with changes only in software download parameters. Robot framework calls each of these test cases, checks if the test case has passed or failed and generates a report accordingly.

The Run Auto button will invoke robot framework with testsuite, which is a set of executable test cases. The testsuite starts the console application for performing software download. The xml containing the parameters is also passed to the console application. Then both the flashing and logging is started simultaneously and reports are generated in three different abstraction levels.

4.3.5 Evaluation

The working of the final tool has been demonstrated to test engineers and the demo included the following items.

1. How to add new test cases related to software download ?
2. How to map folders and files required for download ?
3. How to generate executable test cases for the framework
4. How to run the test suites and check the results ?

After the demonstration, a document has been provided to the test engineers that explains how to set up the tool in a local windows machine.

After using the tool, the test engineers had few suggestions to improve the tool and they are as follows:

1. The test engineers experienced few problems with installation of python libraries, iron python and robot framework. The solution is to provide a package

that would make the installation automatic using windows commands.
2. Few bugs were found related to test tool. One such bug was the tool crashed without indicating any reason when the system was not connected properly to the box car. After improving the exception handling, this bug was fixed. Such bugs mostly related to the usability of the tool have been corrected.
3. It was suggested to name the report using the company standards.
4. It was suggested to include the root cause and the ECU name in the main report instead of showing error codes.

The testing activities have been done manually to test a failure. The logs have been analyzed manually and reports have been generated using the standards followed by the case company. The time taken for completing these activities has been noted down to be approximately around 12 hours. The same activities have been tested using test automation. This activity has been repeated for many tests to detect success and failure. It has been found that the whole testing procedure including running test cases and generating reports can save approximately 8 hours. Also the test cases can be run without manual intervention which would save a lot of tester's time. It was pointed out by the test engineers that many manual errors could be avoided by using the test automation tool.Another advantage of the tool is that test engineer can see the test logs in a console whenever needed while the process goes on.

5 Results

In this section, we discuss the findings of the thesis in terms of research questions and answers that reflect the aspects of continuous integration and test automation in the automobile platform. The research questions are answered based on literature review, interviews conducted through out the thesis, challenges faced while building the test automation framework and the solutions to solve the challenges.

5.1 Research Question 1

What are the most pressing challenges of continuous integration on Automotive platform level ?

The main difference in development process between traditional software development and automobile development is that, in later the process involves both hardware and software [28]. This implies that both hardware and software should be available at the time of system testing on platform level and hence the dependency on vendors that supply hardware and software is unavoidable. Another difference is that testing activity in the automobile platform can be more challenging than traditional software development due to highly manual testing activity, scenarios where testing has to be done without source code and test specifications written in natural language [54]. To bring in the aspect of continuous integration at automobile platform level, efforts should be focused on testing activities performed on the platform. While working on the thesis, it was found that one of the challenging problems faced by the automotive platform is the failure to limit feedback time after testing. Test engineers spent time in test execution, analyzing test logs and making reports to give feedback to developers or vendors.During one of the interviews, it was discussed that manual testing can sometimes lead to incorrect judgments about test results and this can also result in longer feedback time. In some cases the testers may have to provide a quick feedback without going into details and this lack of abstraction can lead to longer feedback cycles. For example, at the case company, a test engineer involved in integration testing may only need to know whether the testing succeeded or failed without going into details.

The vendors that supply components and the case company collaborate through a system, where software and hardware part numbers are maintained. Although the software components are maintained in this system, it is not maintained as a continuous integration system where build and testing are automated. Although the primary intent of a platform developer is to focus on architecture and requirements for the platform, it is important to consider variability elements to succeed in product development[29]. Since automobile industry is focused on safety, the standards set affects the organization's policies concerning development process. The variability element and focus on safety limits agility [30]. Although continuous integration can be implemented to some extent in automobile industry, the cost of implementing continuous integration should balance with that of overall product development.

Some of the testing tools used in automotive industry cannot be easily integrated with continuous integration tools available in the market and this has been brought up during interviews conducted with testing experts who are exposed to CI. This is mainly because these tools were not developed to integrate with CI tools used in software industry. Although test automation can be enabled by modifying the existing tools, the automation tool should work with changes in test cases easily and should have good test coverage.

5.2 Research Question 2

What are the factors to be considered while developing a test automation framework in automotive platform?

Testing in automobile platform is often performed by test engineers who has the expertise of networks, signals and electrical parameters and they may not be programmers. Thus a test automation framework developed for automotive industry must be easy to handle and should abstract technical complications of the framework from a tester: a test engineer should be able to add test cases without going through changes in software used in automation [32]. During the course of the thesis, the idea of a GUI was enthusiastically recommended by test engineers which simplifies the operation of the framework. For example, invoking the automation process, changing folder paths of tools, logs and reports can be made easy through a GUI. The changes in functionality can be frequent and the framework has to be designed so as to build changes easily. The cost versus benefit of using test automation framework depends on the time and effort saved by running automated test cases in any given project and hence test cases have to be chosen accordingly. In automotive industry there is diversity in tools used for testing different functionality and hence test automation framework should be able to integrate multiple tools which also signifies the fact that an automation framework should support different libraries that support these tools [30]. For example, robot framework used in the thesis supports a library named operating system, which can track any process in windows environment.

Analysis of tests, Logs and reporting are important aspects of testing that a test automation framework should cover. During one of the interviews conducted with a function owner within automotive industry, the concept of abstraction of information in reports was brought up. A test automation framework typically runs a test suite which is a set of test cases. So the framework should be able to generate an overall report showing pass or fail of all test cases, individual report for each test case showing details of test steps, logs of test suite and logs of individual test cases. Such reports can be very useful in providing required feedback to people involved in different levels such as developers, team leads and managers.

Test automation framework used for one set of test cases may not be suitable for another set of test cases which leads to more automation frameworks within the same organization and they should have the capability to be integrated with each other. The automation framework developed should also have flexibility to be integrated with CI systems. These features in test automation framework make it expandable for future automation and integration.

5.3 Research Question 3

What are the promising solutions to overcome the challenges faced by platform developers with respect to continuous integration?

To reduce the feedback time of testing to vendors, test automation can be employed. Most frequently used test cases can be automated and test engineers can focus on test cases that are not automated. Test automation framework can be developed to provide necessary abstraction to reports sent out to vendors occupying different positions. Test automation can also reduce manual errors to a great extend since the checks done in test result analysis are standardized. New tools which are developed for testing can be made to have interfaces for integrating with CI systems and test automation frameworks.

A combined ecosystem in which continuous integration and test automation can be put in place which would be beneficial in reducing feedback time and enabling quality of overall development and testing. A CI team can interact with developers and test engineers on a continuous basis to gradually build and maintain CI system and test automation frameworks. Changes in test cases and new test cases can be included in the ecosystem by the CI team.

The table below demonstrates the benefits of implementing test automation framework to support continuous integration.

User Stories	How does Test Automation help?	How this affects continuous integration?	Benefits
1.1	Update test case in suite based on test description	Reduces effort of integrating change sets (Software, Data, Test Description)	Faster Feedback cycle
1.2	Automated, standardized reporting on high abstraction levels. This high level reports is also useful for HIL integration engineers	Give overview on whether Download works. Further testing and integration will depend on this. Reject change otherwise.	Manage Continuous Integration: accept/reject changes quietly
2.1	Automatically bring files for download in correct order and setup testing tools to execute and measure/record.	Reduce the number of manual errors. Speed up test setup/execution.	Faster Feedback cycle
2.2	Automated generation of medium level abstraction reports of software download. Based on this report, failure is tracked down further in logs.	Gives detailed description of different steps in the Download Process. Further test evaluation will depend on this.	Manage Continuous Integration: Speeds up the download evaluation.
3.1	Automated, standardized reporting on low abstraction level based on test execution and signal names	Allow efficient fix of change set that failed to integrate.	Improves average cycle time of change sets and helps to remove blockers
3.2	Automated report generation with the root cause analysis about issues to be reported to ECU suppliers.	Improve the integration process. Speed up reporting issues.	Faster Feedback cycle
3.3	Automated report generation with the root cause analysis about issues to be reported to developers.	Improve the integration process. Speed up reporting issues. Gives more time for the developers.	Manage Continuous Integration: Helps improve the development process

Table 5.1: Impact on Test Automation and Continuous Integration

6 Conclusion

This thesis focused on analyzing and addressing some of the challenges of implementing continuous integration on automotive platform where individual ECU's software and hardware components are integrated through different networks. Through literature review we have looked over the existing challenges in automotive industry and identified the same in the case company. Several challenges have been identified and such as dependency on vendors that supply software and hardware, long feedback time between testing and defect fixing, lack of abstraction, manual errors, safety requirements, variability aspect of product development, difficulty of integrating automotive testing tools with continuous integration tools used in software industry etc.

During the study, it was identified that one of the most pressing challenges is the long feedback time between testing and defect fixing, which in turn delayed release of car. To address this issue, we have come up with a test automation framework to automate the most frequently used test cases. As this test case is usually run in a loop and takes long time, so if such test cases can be automated it can substantially reduce the feedback time and release the car early in the market bringing a competitive edge to the Organization. The thesis was completed following a design science methodology with three iterations and in each iteration, parts of test automation framework was implemented and interviews and demo sessions are conducted to evaluate and to obtain feedback from test engineers and academic experts on the subject. The feedback is used to further develop the framework enabled the test engineers to use it easily and effectively.

The test automation framework has been evaluated, tested and used by the test engineers for testing the functionality and the organization is interested in integrating more test cases and functionality to the framework. However, the possibility of implementing and integrating new test cases for new functionality has to be examined as part of future research. A continuous integration ecosystem where different vendors and the organization can collaborate and to which different test automation framework can be integrated will improve the co-ordination and communication in a distributed environment where developers and testers located across different organizations. The possibility of implementing such an ecosystem has to be studied and pursued as future research.

Bibliography

[1] Roel Wieringa,"Design science as nested problem solving",In Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology (DESRIST '09). ACM, New York, NY, USA, Article 8, 12 pages, https://doi.org/10.1145/1555619.1555630, 2009.

[2] Alan R. Hevner, Salvatore T. March, Jinsoo Park, and Sudha Ram, "Design science in information systems research", MIS Q. 28, no.1, pp. 75-105, March, 2004.

[3] Vijay K. Vaishnavi and William Kuechler, Jr, "Design Science Research Methods and Patterns: Innovating Information and Communication Technology (1st ed.)", Auerbach Publications, Boston, MA, USA, 2007

[4] A. Collins, D. Joseph, and K. Bielaczyc, "Design research: Theoretical and methodological issues, The Journal of the learning sciences", vol. 13, no. 1, pp. 15-42, 2004.

[5] International Standard Organization. ISO 11519-2, "Road Vehicles- Low Speed serial data communication - Part 2: Low Speed Controller Area Network", ISO, 1994.

[6] Goutam Kumar Saha, "Understanding software testing concepts", Ubiquity, Article 2 , http://dx.doi.org/10.1145/1348483.1348484, February, 2008.

[7] Ron Patton, "Software Testing (2nd Edition)". Sams, Indianapolis, IN, USA, 2005.

[8] Vector Informatik GmBh, "Learning module LIN", [Online]. Available: http://elearning.vector.com/index.php?wbt_ls_kapitel_id=1330149&root=378422&seite=vl_lin_introduction_en. [Accessed 13 September 2017].

[9] MOST Cooperation, "Motivation for MOST" [Online]. Available: http://www.mostcooperation.com/technology/introduction/. [Accessed 13 September 2017].

[10] National Instrutments, "FlexRay Automotive Communication Bus Overview" 21 August 2009. [Online]. Available: http://www.ni.com/whitepaper/3352/en/oc1. [Accessed 13 September 2017].

[11] E. Mayer, "Serial Bus Systems in the Automobile - Part 2: Reliable data exchange in the automobile with CAN" Vector GmBh, December 2006. [Online]. Available: http://elearning.vector.com/portal/medien/cmc/press/PTR/SerialBusSystems_Part2_ElektronikAutomotive_200612_PressArticle_EN.pdf. [Accessed 13 September 2017]

[12] Herbert Schuette and Markus Ploeger, "Hardware-in-the-Loop Testing of Engine Control Unit- A Technical Survey ", http://dx.doi.org/10.4271/2007-01-0500, 2007.

[13] LIN Cheng and ZHANG Lipeng, "Hardware-in-the-loop Simulation and Its Application in Electric Vehicle Development", IEEE Vehicle Power and Propulsion Conference (VPPC), 2008.

[14] Eric Knauss and Daniela Damian, "Towards Enabling Cross-Organizational Modeling in Automotive Ecosystems, IEEE Fifth International Workshop on Empirical Requirements Engineering, 2015.

[15] E. Mayer, "Serial Bus Systems in the Automobile - Part 2: Reliable data exchange in the automobile with CAN", Vector GmBh, [Online]. Available:http://elearning.vector.com/portal/medien/cmc/press/PTR/SerialBusSystems_Part2_ElektronikAutomotive_200612_PressArticle_EN.pdf. [Accessed 13 September 2017], December, 2006.

[16] "Software and systems engineering Software testing Part 1:Concepts and definitions", IEEE, Article 2, 1 pages. [Online]. Available: http://dx.doi.org/10.1109/IEEESTD.2013.6588537, September 2013

[17] "Software and systems engineering Software testing Part 2:Test processes", IEEE, Article 2, 1 pages 1-68

[18] N. Tcholtchev, M. A. Schneider and I. Schieferdecker, "Systematic Analysis of Practical Issues in Test Automation for Communication Based Systems," 2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pp. 250-256, Chicago, IL, 2016.

[19] Dudekula Mohammad Rafi, Katam Reddy Kiran Moses, Kai Petersen School of Computing Karlskrona, Sweden, Mika V. Mantyl Lund University Department of Computer Science Lund, "Benefits and Limitations of Automated Software Testing: Systematic Literature Review and Practitioner Survey", Sweden

[20] "VN8900 Interface Family Manual", [Online]. Available: https://vector.com/portal/medien/cmc/manuals/VN89xx_Manual_EN.pdf.

[21] M. D. Tokcan, O. Ozturk and H. Tuna, "MetTest: A Test Automation Framework for Development of a Point-To-Multipoint Radio," 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), Graz, pp. 1-2, [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7102624&isnumber=7102573, 2015.

[22] Bisht, Sumit. "Robot Framework Test Automation", Packt Publishing, ProQuest Ebook Central, [Online]. Available:http://ebookcentral.proquest.com/lib/chalmers/detail.action?docID=1532018, 2013

[23] "Robot Framework User Guide", Nokia Solutions and Networks, [Online]. Available: http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html [Accessed 13 September 2017]

[24] Laukkanen P, "Data-Driven and Keyword-Driven Test Automation Frameworks" Master's Thesis, Helsinki University of Technology - Aalto University, 2006.

[25] "Guidelines to create a Robust Test Automation Framework", Alliance Global Services White Paper

[26] "Qt/Embedded", Trolltech AS White Paper, [Online]. Available: http://ftp.task.gda.pl/site/qt/pdf/QtEWhitepaper.pdf.

[27] M. Shahin, M. Ali Babar and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," in IEEE Access, vol. 5, no. , pp. 3909-3943, doi: 10.1109/ACCESS.2017.2685629, http://ieeexplore.ieee.org.proxy.lib.chalmers.se/stamp/stamp.jsp?tp=&arnumber=7884954&isnumber=7859429, 2017.

[28] Sebastian Vöst, Stefan Wagner, "Towards Continuous Integration and Continuous Delivery in the Automotive Industry", [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1612/1612.04139.pdf.

[29] L. Brownsword and P. Clements, "A Case Study in Successful Product Line Development", Standard CMU/SEI-96-TR-016,ESC-TR-96-016, [Online]. Available: ftp://ftp.sei.cmu.edu/pub/documents/96.reports/ps/tr016.96.ps, 1996.

[30] Eric Knauss, Patrizio Pelliccione, Rogardt Heldal, Magnus Ågren, Sofia Hellman, and Daniel Maniette, "Continuous Integration Beyond the Team: A Tooling Perspective on Challenges in the Automotive Industry", In Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '16). ACM, New York, NY, USA, , Article 43 , 6 pages, [Online]. Available: https://doi.org/10.1145/2961111.2962639, September, 2016.

[31] Sabina, AMARICAI and Radu, CONSTANTINESC, "Designing a Software Test Automation Framework",Informatica economica, Inforec Association, ISSN :1453-1305, 2014.

[32] Graham, Dorothy and Mark Fewster, "Experiences of test automation; case studies of software test automation", Ringgold Inc, vol. 27, issue:2, ISSN :0887-3763, 2012.

[33] Sebastian Vöst, "Vehicle level continuous integration in the automotive industry", In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015), ACM, New York, NY, USA, 1026-1029, 2015.

[34] N. Navet et al, "Trends in Automotive Communication Systems," Proceedings of the IEEE, vol. 93, (6), pp. 1204-1223, 2005

[35] Dakroub, H. and Cadena, R., "Analysis of Software Update in Connected Vehicles," SAE Int. J. Passeng. Cars – Electron. Electr. Syst. 7(2):411-417, , doi:10.4271/2014-01-0256,2014.

[36] "Control Area Network (CAN) Overview", National Instruments, [Online]. Available: http://www.ni.com/white-paper/2732/en/.

[37] J. H. Kim et al, "Gateway Framework for In-Vehicle Networks Based on CAN, FlexRay, and Ethernet," IEEE Transactions on Vehicular Technology, vol. 64, (10), pp. 4472-4486, 2015.

[38] R. Makowitz and C. Temple, "Flexray - A communication network for automotive control systems," in 2006, . DOI: 10.1109/WFCS.2006.1704153.

[39] ISO 14229-1:2013(E) - Road vehicles – Unified Diagnostic Services (UDS) – Part 1: Specification and requirements.

[40] I. Schieferdecker, "Model-Based Testing," IEEE Software, vol. 29, (1), pp. 14-18, 2012.

[41] V. Garousi and F. Elberzhager, "Test Automation: Not Just for Test Execution," IEEE Software, vol. 34, (2), pp. 90-96, 2017.

[42] J. Tang, X. Cao and A. Ma, "Towards adaptive framework of keyword driven automation testing," in 2008, . DOI: 10.1109/ICAL.2008.4636415.

[43] S. J. Galler and B. K. Aichernig, "Survey on test data generation tools: An evaluation of white- and gray-box testing tools for C, C++, Eiffel, and Java," International Journal on Software Tools for Technology Transfer, vol. 16, (6), pp. 727-751, 2014.

[44] Seleniumhq.org. (2017). Selenium - Web Browser Automation. [online] Available at: `http://www.seleniumhq.org/` [Accessed 8 Sep. 2017].

[45] Bitbar. (2017). DevOps for Mobile App Testing and Mobile Monitoring | Bitbar. [online] Available at: `https://bitbar.com/` [Accessed 8 Sep. 2017].

[46] Junit.org. (2017). JUnit. [online] Available: `http://junit.org` [Accessed 8 Sep. 2017].

[47] K. Frounchi et al., "Automating Image Segmentation Verification Research," Proc. IEEE 5th Int'l Conf. Software Testing, Verification and Validation (ICST 12), 2012,pp. 400–409.

[48] S. R. Shahamiri et al, "An automated framework for software test oracle," Information and Software Technology, vol. 53, (7), pp. 774-788, 2011.

[49] P. Ammann, J. Offutt Introduction to Software Testing (first ed.), Cambridge University Press, New York (2008)

[50] [Online]. Available: `https://vector.com/portal/medien/cmc/manuals/CANoe75_Manual_EN.pdf`

[51] [Online]. Available: `http://ironpython.net/`

[52] [Online]. Available: `http://www.jython.org/jythonbook/en/1.0/`

[53] S. White, J. Lin, N. Gulave and M. Kienast, "AN-AND-1-117 CANoe CANalyzer as a COM Server" Available: `https://vector.com/vi_downloadcenter_it.html?product=canoe&formular_treffer_submit=1#`

[54] Lachmann, Remo, and Ina Schaefer. "Towards Efficient and Effective Testing in Automotive Software Development." GI-Jahrestagung, 2014.

[55] K. Peffers et al, "A Design Science Research Methodology for Information Systems Research," Journal of Management Information Systems, vol. 24, (3), pp. 45, 2008.

[56] K. Kelley, B. Clark, V. Brown, and J. Sitzia, "Good practice in the conduct and reporting of survey research," International Journal for Quality in Health Care, vol. 15, no. 3, pp. 261–266, 2003.

[57] S. E. Hove and B. Anda, "Experiences from conducting semi-structured interviews in empirical software engineering research," in Software metrics, 2005. 11th ieee international symposium. IEEE, 2005, pp. 10–pp.

[58] R. Longhurst, "Semi-structured interviews and focus groups," Key methods in geography, pp. 117–132, 2003.

[59] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," Empirical software engineering, vol. 14, no. 2, pp. 131–164, 2009.

A
Appendix 1

The survey questions handed over during evaluation of first iteration are given in the next page.

Master Thesis-Test Automation to Enable Continuous Integration for an Automotive Platform

A Design Science Study of Software Download Function Case

Thesis Goals

1. Develop a Framework to automate test execution such that flashing and logging can be done simultaneously.

2. Develop test cases to log bus communication in CANoe to achieve automation of test evaluation.

3. Develop test scriptis and UI to improve the usability of the tool.

Start this form over.

Survey Questions

The ratings are in increasing order with 1 as the lowest representing not helpful and 5 as the highest rating representing very helpful

1. **From scale of 1 to 5 please rate as to how do you think the thesis goals would affect testing quality?**
   Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

2. **From scale of 1 to 5 please rate as to how do you think the thesis goals would be helpful for a test engineer?**
   Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

3. **From scale of 1 to 5 how would you rate our approach of using DSA APIs and COM server in first iteration?**
   Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

4. **If you have chosen 1 or 2 to the previous question, could you suggest an alternative approach ?**

5. **One of the common testcases is parallel and queued download on multiple ECUs. Is the first iteration helpful in implementing the test case ?**
Mark only one oval.

1	2	3	4	5
⬭	⬭	⬭	⬭	⬭

6. **What do you think can be added as an enhancement to the first iteration ?**

7. **Any suggestions on developing a better test setup?**