CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Testing Strategies to Support Continuous Integration for Complex Systems

Master's thesis in Software Engineering

HARITHA GANGINENI

SARAH JAMIL

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

Master's thesis 2016:NN

Testing Strategies to Support Continuous Integration for Complex Systems

HARITHA GANGINENI
SARAH JAMIL

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

Testing Strategies to Support Continuous Integration for Complex Systems

HARITHA GANGINENI
SARAH JAMIL

Supervisor: ERIC KNAUSS, Department of Computer Science and Engineering
Examiner: REGINA HEBIG, Department of Computer Science and Engineering

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

Testing Strategies to Support Continuous Integration for Complex Systems

HARITHA GANGINENI
SARAH JAMIL
Department of Computer Science and Engineering
Division of Software Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG

Abstract

Software development companies strive to implement and embrace continuous integration, which aims to integrate different parts of software in a continuous way in order to increase the frequency and productivity of development process and to improve the communication to customer and market. The testing strategy is considered as one of the most important aspects for successful continuous integration. Yet, today many companies face challenges regarding testing activities. The testing of complex software that are hardware dependent, creates a long feedback loop because the software needs to run on different hardware. This in turn causes a delay in delivery of the product.

When integrating the parts of the software into one system may lead to errors and breaking down the system. Furthermore, causing a problem to the developer to figure out where the problem lies. These problems lead to increased work effort and unpredictable release times, which prevent the achievement of continuous integration. The aim of this study is to identify and design a useful test strategy for continuous integration that helps address the complexities and get immediate feedback from the test results. This study was conducted in collaboration with a company that is looking to improve their existing test strategy to achieve continuous integration.

The results of this study are: an investigation of current test activities is conducted, automated tests are developed and implemented in order to reduce the feedback loop and further the requirement traceability is improved through mapping the requirements with the developed tests. Furthermore, a suitable test strategy for continuous integration should allow to visualise the test activities, have automated tests at system level and provide a clear traceability of the requirements.

The methodology followed for this study is design science research (DSR) with 5 phases in each regulative cycle. The data collected is based on related literature as well as a survey and interviews with key employees. The outcome was gathered based on the progress with the methodology (the test strategy). The results of this study was successfully achieved and the feedback loop was reduced.

Keywords: Software Testing, Testing strategies, Continuous Integration.

Acknowledgements

We would 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 for his guidance and valuable advice that he provided to us throughout the thesis.

Haritha Gangineni and Sarah Jamil, Gothenburg, June 2016

Contents

1 Introduction **1**
- 1.1 Purpose of Study . . . 1
- 1.2 Statement of Problem . . . 2
- 1.3 Research Questions . . . 2
- 1.4 Scope and Limitations . . . 3
- 1.5 Structure and Contributions . . . 3

2 Background **5**
- 2.1 Case Company . . . 5
- 2.2 AUTOSAR . . . 5
  - 2.2.1 Software Architecture . . . 6
  - 2.2.2 Basic Software Layer . . . 6
    - 2.2.2.1 Services layer . . . 7
    - 2.2.2.2 The ECU Abstraction Layer (ECUAL) . . . 7
    - 2.2.2.3 Microcontroller Abstraction Layer (MCAL) . . . 7
- 2.3 CIViT-Continuous Integration Visualization Technique . . . 7
- 2.4 Testing Strategies . . . 8
- 2.5 Software Testing . . . 9
  - 2.5.1 Unit Testing . . . 9
  - 2.5.2 Integration Testing . . . 9
  - 2.5.3 System Testing . . . 10
- 2.6 Automated Testing . . . 10
- 2.7 Continuous Integration . . . 10

3 Research Methodology **13**
- 3.1 Design Science Research . . . 13
  - 3.1.1 Investigation of the Problem . . . 14
  - 3.1.2 Suggestion of a Design Solution . . . 15
  - 3.1.3 Validation of the Design . . . 15
  - 3.1.4 Implementation . . . 15
  - 3.1.5 Evaluation . . . 16
- 3.2 Interviews and Surveys . . . 16

4 Iterations **19**
- 4.1 Iteration 1: Implementing CIViT . . . 19
  - 4.1.1 Investigation of Problem . . . 19

4.1.2 Suggesting a Design Solution 19
4.1.3 Validation of the Design 20
4.1.4 Implementation 20
4.1.5 Evaluation of Iteration 1 21
4.2 Iteration 2 23
4.2.1 Investigation of Problem 23
4.2.2 Suggesting a Design Solution 24
4.2.3 Validation of the Design 24
4.2.4 Implementation 24
4.2.5 Evaluation of Iteration 2 25
4.3 Iteration 3 28
4.3.1 Investigation of Problem 28
4.3.2 Suggesting a Design Solution 28
4.3.3 Validation of the Design 28
4.3.4 Implementation 29
4.3.5 Evaluation of Iteration 3 29

5 Results **31**
5.1 Research Questions 31
5.1.1 Research Question 1 31
5.1.2 Research Question 2 32
5.1.3 Research Question 3 34

6 Discussion **37**
6.1 Threats to Validity 39

7 Conclusion **41**

Bibliography **I**

A Appendix **I**
A.1 Workshop questionnaire I
A.2 Survey questionnaire I

1 Introduction

Software development companies strive to implement high quality software and to deliver final products to the market quicker. Among the best practices to do so, many embrace continuous integration as one of the agile practices [1, 2] which aims to integrate the different parts of the software in a continuous way on a daily basis. This in turn increases the frequency and productivity of the development process as well as to improve communication with customer and market. Furthermore, testing strategy is considered as one of the most important aspects for successful continuous integration. Yet, today not many companies have invested in setting up a testing strategy and consequently face challenges regarding testing activities [3]. For example, one of the problems of continuous integration is that, testing of complex software creates a long feedback loop; and lack of sequence feed-backs if some new development is breaking something in the system. These problems may lead to increased work efforts to fix the broken system and unpredictable release times because of late feed-back. Since, testing has a high impact on achieving continuous integration in the development process, having a continuous test strategy is useful to address this complexity and get immediate feedback. Furthermore, the test strategy can be used to check the accomplishment of system requirements. These strategies can be used within embedded systems and to identify which requirements are tested by which test cases, and to check the success of all code line executing during the test.

1.1 Purpose of Study

The purpose of this study is to address the obstacles that prevent the achievement of continuous integration in complex systems. Furthermore, through dealing with the issues regarding the testing and implementing a suitable test strategy which helps them to achieve faster delivery to the customer, reduce the number of defects and to improve the quality of the software development process. The aim of this study, is to evaluate the current testing activities and to propose suitable test strategies and improvements needed in order to reach the continuous integration. For this, we chose to pursue this research at a company in order to have a clear understanding of the software industry.

1.2 Statement of Problem

The case company where the study took place, deals with providing AUTomotive Open System ARchitecture (AUTOSAR) products to the automotive market. Currently, the company has two main components in the platform software: the Basic Software (BSW) platform and the Microcontroller Abstraction Layer (MCAL). The BSW contains about 40 modules and all these are hardware independent. The major challenge is that each module is configurable and there are endless ways of combining these modules into one specific platform.

In contrast, the MCAL consists of about 10 modules that are all hardware dependent. The challenge here lies that the company supports about 20 different Micro Controller Units (MCUs) and only limited automation is implemented at this point. This means that they do not have any continuous integration for quicker releases and the testing of the MCUs is carried out semi automatic before each release. Hence, leading to identify the defects late while the release testing is a very intense period.

The company follows a V-model in their development process and currently they follow three testing scopes: Unit testing, Integration testing and System Software testing. The Unit testing covers the majority effort i.e. 80 percent of the time is allocated and is continually integrated. The base of these tests depends on the requirements base as well as code complexity and code coverage. Whereas, 15 percent of the time is allocated for the integration tests. This is tested semi automatic and it lacks inputs. In contrast, the system software testing is very little automated and is based on the Software requirement specification(SRS) by AUTOSAR; and is required to check the complete system.

As the customer demands are increasing and the need for delivering to the market as fast as possible without fault is much a requirement from them. Though, there are tests available today, the requirements are being mapped with the test cases i.e. checking which requirements are tested by that particular test case. Further, the code coverage is being used to make sure that all the code lines are executed during the unit tests i.e. branch and state coverage. All these processes are in fact taking time which in turn are showing impact on the product delivery. The company is looking for being able to deliver fully verified software on all MCUs every night, through improving their hardware testing which is considered as a main blocker.

1.3 Research Questions

The research questions, according to the thesis objective, are the following:

RQ 1: What are the current test strategies?

RQ 2: With respect to test strategies what are the challenges that prevent the success of Continuous integration?

RQ 3: What is a/how is a suitable test strategy for Continuous Integration(CI) characterized?

1.4 Scope and Limitations

As mentioned in Section 1.2, that the case company currently follows three types of testing strategies and it is known that the testing at the unit level is satisfactory and needs no further improvement for now. In contrast, since some parts of the system needs a support of the hardware while testing, there exists some problems in those parts of the system.

The scope of this study is to asses the current test activities in order to identify the problems and challenges being faced by the developers/testers. Also, to find out a suitable test strategy for the development process to help for quicker delivery to the customer.

1.5 Structure and Contributions

Chapter 2 gives a rundown of the basic knowledge about the AUTOSAR and its architecture, short description about the CIViT and the different test strategies. It further gives an insight about the software testing and continuous integration. Chapter 3 explains the research methodology followed in this thesis i.e. the design science research and a description of the iteration process that it follows. In Chapter 4, each iteration followed during the study is described. It also focuses on the unique test strategy that is implemented and evaluated during each iteration. Chapter 5 provides the results attained after end of each iteration and further the findings from each iteration are also presented. The chapter also consists of the result of the survey conducted to evaluate the confidence level of the developers/testers. Each research question is discussed in Chapter 6. The contribution to the research community and the threats to validity are also discussed. Finally, Chapter 7 concludes the study and provides a short summary of the thesis work carried out and also about the future work.

2
Background

In this chapter, the theory and the background information related to this thesis project is presented. Section 2.1 consists of the general context of the thesis project. Whereas in Section 2.2, a short description of AUTOSAR and its software architecture are presented and in Section 2.3, an introduction to CIViT has been presented. Furthermore, in Section 2.4, various testing strategies are explained. In Section 2.5, automated testing is described and in Section 2.6, a brief explanation about continuous integration is described.

2.1 Case Company

The study is conducted in collaboration with a company located at Gothenburg, Sweden. The company mainly works with developing AUTOSAR (Automotive Open Source Architecture) products. AUTOSAR is a worldwide automotive industry standard for automotive suppliers, manufacturers as well as other companies who are involved in software development for the vehicles [4].
Furthermore, the company has their own product which is an Eclipse based configuration environment tool for automotive software. The specific challenge is that their AUTOSAR basic software needs to run on a large number of different hardware which makes continuous integration particularly difficult.

2.2 AUTOSAR

AUTOSAR refers to "AUTomotive Open System ARchitecture" which is an embedded platform reference architecture for the Electronic Control Unit (ECU) software; founded in the year 2003. The software for the ECU can be developed as completely hardware independent for the AUTOSAR layered architecture. The idea behind AUTOSAR is to standardise the functionality of basic software, make it easy to transfer the software, support different functional domains and to develop highly dependable systems [4].
The introduction of AUTOSAR helps to handle the complexity of the architecture and thus reducing the time and cost of production [5]. Furthermore, by adapting AUTOSAR as a standard, the suppliers, manufacturers and the tool developers benefit with regards to re-usability of the software, providing more opportunity to design flexibility and gives clear rules for integration [4].

2.2.1 Software Architecture

The AUTOSAR software is divided into three layers.

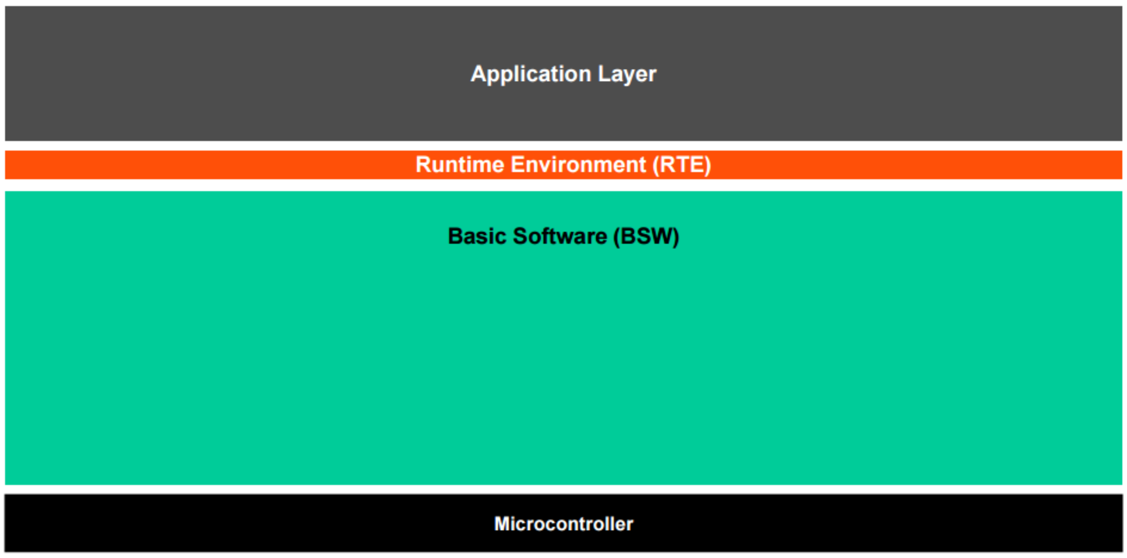


Figure 2.1: Layered Software Architecture [6]

As shown in the Figure 2.1, the top layer is the *Application layer*, which is considered as the external or out of scope for the AUTOSAR standard. Furthermore, the actual applications are contained in this layer which are used to run on the ECU. Whereas, *Run time Environment (RTE)* is the second layer which is responsible for providing the communication services to the above layer which is the application layer. The communication between software components at each ECU and between other ECUs occur via runtime environment [7]. The third layer is the *Basic Software Layer (BSW)* which is responsible for providing an abstraction that is hardware independent to other layers. Furthermore, the BSW interacts with the microcontroller to make it hardware dependent and thus implementing this layer depends on the type of hardware used.

2.2.2 Basic Software Layer

The Basic Software Layer is further divided into three more layers. The Figure 2.2 shows the clear division of the layers. The layers are: Services layer, ECU Abstraction Layer and MCAL.

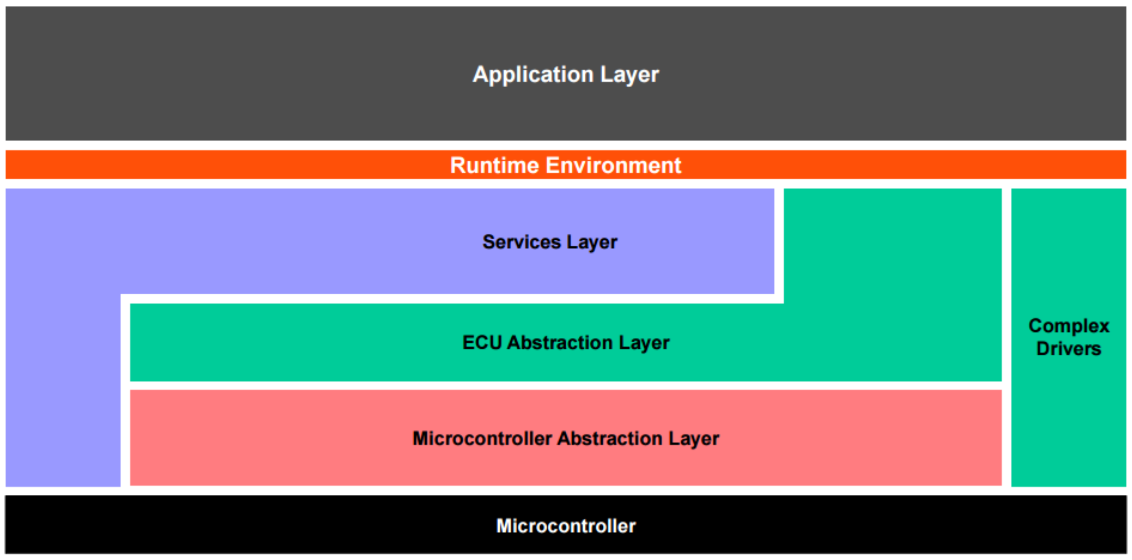


Figure 2.2: Basic Software Layers [6]

2.2.2.1 Services layer

It is the top most layer of the Basic Software that consists of the modules for Memory services, ECU state management, Communication services and the Operating systems. Its major responsibility is to deliver primary services for applications, RTE and basic software modules [6].

2.2.2.2 The ECU Abstraction Layer (ECUAL)

This layer acts as an interface between the drivers in the Microcontroller Abstraction Layer (MCAL) and to that of the layers present above the ECU Abstraction Layer. Furthermore, the ECU Abstraction Layer serves the purpose of making the upper software layers free of the ECU hardware design [6]

2.2.2.3 Microcontroller Abstraction Layer (MCAL)

The MCAL is situated at the bottom of the BSW layer which contains the internal drivers and the modules present in this layer communicate with the MCU. Furthermore, this layer makes the other higher software layers independent of the hardware [6]

2.3 CIViT-Continuous Integration Visualization Technique

The CIViT model is exclusively used to visualize the testing activities that are taking place inside the company while developing the product and further makes it easy to identify the present condition of the product in terms of quality.

Furthermore, it provides a stage for discussion on where to improve the testing strategy and to speedup the production process [8].
The CIViT model is concentrated on four types of testing: functionality, legacy functionality, quality attributes, and edge cases. The figure below shows how these elements are arranged in a CIViT model.

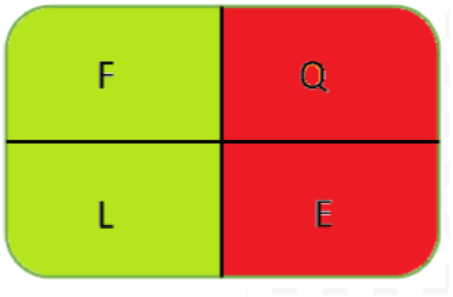


Figure 2.3: Arrangement of the types of testing activities in CIViT model

The functionality testing is referred to testing of the functionality of the system which is currently under development. Whereas, Legacy functionality testing is concerned about the functions that already exists and checks if they still work as per the requirements even after adding new functionality. Furthermore, quality attributes testing like the performance, safety, reliability and security aspects ensures that the system under test is in line with the specified quality specifications. Finally, the edge case testing ensures if there are any unusual circumstances that have come up from defects that were not detected in the early stage and have been noticed only after some time [8]. The green color inside the box depicts that the requirements are fully covered, whereas the red shows that there is no coverage of the requirements. Orange color represents that the requirements are partially covered. Furthermore, the border line of the box explains about the level of automation at each stage. Green color shows that the level is fully automated, whereas red illustrates that the level is manually tested. Orange portraits that the level is partially automated. For the complete picture of the CIViT and understandability, check Figure 4.1.

2.4 Testing Strategies

A test strategy is a process of drawing the software testing approach. In this paper we use the following definition of test strategy based on [9, 10, 11] This approach helps the developers and the testers to achieve the desired goal of testing. Mainly the test strategy is based on the requirements specification of the development process [9]. Having a test strategy means having a clear path for the testing levels, as well as explicate the individual roles and responsibilities during test planning [9, 11]. Furthermore, a good test strategy explains the risks involved in the product being developed in a documented way and describes how these risks can be reduced through detecting the errors at earlier stage. A test strategy is used as a technique to test the specification with user requirements. These techniques can be either automated or manual or a mix of both [11]. Furthermore, for big systems, executing huge number of test cases at one time might be difficult, and so the test strategy is used to schedule the systems tests in a planned way [10]. This means that there exists only one test strategy to follow for large systems and this strategy shall have more than one test plan [9]. These plans are implemented based on the goals to achieve.

Furthermore, a test strategy should be capable to support different levels of testing i.e. acceptance tests, module testing, unit testing, integration testing and system testing [11]

Having these levels within one test strategy will help to:

- make sure that the developed system is fully tested.
- describe the level of the tests to be conducted.
- have a clear path for validating the system.
- have a clear connection between the specification and customer requirements, also the specification and the developed product.
- take care of prioritizing and scheduling the tests and allows for further updating in the system testing.
- increase the system maturity.
- allow to automate the tests in writing and executing them.

2.5 Software Testing

Software testing is an important phase in any software development process. Testing of software is important because in this phase the main focus is to asses the software for success, failing or any risk in the product at any step of the development process [12]. Furthermore, such kind of testing helps to check if the final product meets the customer requirements. The main outcome of testing the software is to attain the required quality in the product being developed and discover any potential threat or defects in the software. There are different kinds of testing which can be implemented at different stages of the development process. The selection of the type of test is based on the purpose of testing and at each level of development.

2.5.1 Unit Testing

Testing the individual software components or a group of components is defined as unit testing [13]. It is a method used to increase the accuracy and quality of software [14]. This is the primary level of testing and the testing takes place by performing unit tests remotely [15].

2.5.2 Integration Testing

Whittaker defines integration testing as: "Integration testing tests multiple components that have each received prior and separate unit testing. In general, the focus is on the subset of the domain that represents communication between the components" [13] . Integration testing is performed to test the integrated parts of the software that are submitted by developers during the development process. The implementation of integration testing is important in order to detect the errors at an early stage until all the single components collaborate to form an executable program. Such kind of testing is important in companies where the development process is based on continuous integration. Here the testing takes place at the module level and gives importance to the communication between modules and their interfaces [16].

2.5.3 System Testing

System testing is also a kind of testing which is implemented at the end of development process. The testing of the system is based on hardware and software components or only software. It is mainly performed to check if the system/product is developed as specified to the requirements. This is usually considered as the final stage of testing, since the entire product is being tested and verified whether it fulfills the purpose or not. Usually, the complete domain is considered here [13].

2.6 Automated Testing

Another type of testing is "Automated Testing" which is considered as a key to continuous integration. The Automated testing helps to test the running code [17]. Furthermore, it helps to shorten the system build so that the testing can be run frequently. The automated testing allows the frequent testing which in turn allows to early checking for the quality of the product and allows frequent deliveries. Other benefits of having automated testing is to have reduced time for testing, and make the tests much easier and funnier to work with. Furthermore, it improves the visibility of the development process and communication.

2.7 Continuous Integration

Continuous Integration is an agile software engineering practice that most software companies are working towards. The idea behind the continuous integration is to allow the developers to integrate their code several times in a day. The frequent integrating allows frequent builds and tests of the code. Martin Fowler in his words defined continuous integration as -

"Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible." [2]

The frequent integration of the code will allow to speed up the development process, increase the value of testing by executing the tests from earlier stage. It also becomes easy to locate and fix the errors as soon as possible [2]. Implementing such practices in the organization heps to deliver faster to the market.

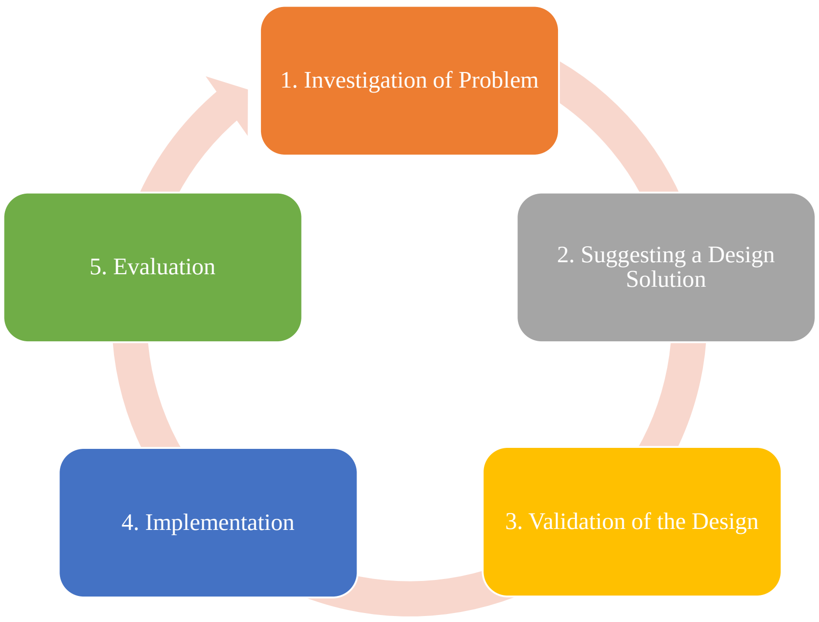
3

Research Methodology

In this chapter, a detailed description of the research methodology followed during the thesis has been presented. The methodology used is the design science research (DSR).

3.1 Design Science Research

The research methodology followed in this study is the design science research (DSR) which is performed as an iterative process and each iteration consists of four to five phases [18, 19, 20, 21]. The different phases in the design science research are shown in Figure 3.1.



1. Investigation of Problem
2. Suggesting a Design Solution
3. Validation of the Design
4. Implementation
5. Evaluation

Figure 3.1: Different phases in design science research

For each regulative cycle, we implement the five phases: Investigation of the Problem, Suggesting a design solution, Validation of the design, Implementation, and last phase is Evaluation. Each iteration shall have a different focus on the research. The output of each regulative cycle is considered as an input for the next regulative cycle.

The primary aim of this study is to understand how to build a suitable test strategy for complex systems to adopt complete continuous integration. The main element that differentiates the methodology used in this study from that of the other methodologies is that, it has a unique way to deal with problems in order to solve them. The method was found to be more suitable for this study because of its design to find appropriate approaches to the study of learning phenomenon which serves the human objectives [21]. The design research method helps to analyse carefully the current status of testing processes in the study and how the different elements of automated architecture are working out. Through this research methodology, the study to identify the weakness in development process and the areas that need further improvement in the software industry could be found explicitly. Furthermore, the "Regulative Cycle" is a framework of the research method that solves the problem in logical structure [19]. All this together makes this method much suitable for this research. The different phases are explained in detailed in the following subsections.

3.1.1 Investigation of the Problem

The main focus of this phase is to understand the problem statement and its impact on the software development process, while asking all the questions that helps to understand the current problem in more deeper sense. The reason for investigating the problem is that, the company has a goal to improve continuous integration process through improving their testing activities. Furthermore, the focus is to diagnose the company's goals. So, this shows that the investigation of the problem is "goal driven" [19]. With respect to our study, during the problem of investigation we started by searching related literature regarding continuous integration and different test strategies. The results of this phase during all the iterations were to have an understanding regarding the above mentioned domains and further to investigate the kind of obstacles and challenges that prevent the success of continuous integration. Furthermore, at this phase we gained a picture about different test strategies and its benefits and drawbacks. During all the iterations that took place while performing the study, the investigation phases helped us to find and analyse the problem in a much deeper way.

In the Iteration 1, we analysed the current status of testing activities at the company to find the challenges that prevent to achieve continuous integration. Furthermore, the literature has provided extensive understanding regarding test strategies within continuous integration. Whereas in the Iteration 2, we investigated to find the possible solution for the problems that are related to testing, through assessing the impact of these problems. In the Iteration 3, we investigated more on how to improve the requirement traceability and its impact on the tests. The outcome of this phase is the key for the next phase which is the "Suggestion of a Design Solution".

3.1.2 Suggestion of a Design Solution

The focus of this phase is to come up with a set of solutions to the problem. These solutions are identified based on the gathered information from the employees at the case company and also from the literature review. The solution design is the problem solver responsibility to come up with possible solutions to the company. During this research the solution design is identified based on the outcome of previous phase "Investigation of the problem". The suggested solutions design varies at each iteration based on the results of problem investigation at each regulative cycle. An important thing that should be taken into consideration at this phase is that, there are key employees to discuss and prioritise the suggested solutions, also not all of these suggestions are able to be implementable [19]. It depends on the discussions with employees in the industry who decides the most appropriate or suitable solution. The outcome of this phase is an outline of a plan which will be evaluated in the next phase.

The solution design of the Iteration 1 is to implement the CIViT workshop at the case company with key employees involved. Details about the CIViT workshop can be found in Section 4.1.4. While in the Iteration 2, the suggested design becomes much clearer for all the parties to develop automated tests for the system level. During the Iteration 3, requirement traceability for the tests developed was suggested.

3.1.3 Validation of the Design

In this stage, the suggested solution design will be evaluated by related people to decide whether the proposed solution helps the company to reach its goals [19]. During the research, in order to be able to determine the validity of the suggested solution, we focused on having answers on how and would such suggestions satisfy the criteria which were identified in the investigation of the problem phase. For the first iteration, the design is validated with the industry and the academic people, and they agreed upon the idea of implementing the CIViT model as a starting point to visualize the current status of the testing activities at the company and to come up with all problems in the specific areas the employees were engaged with. The validation of the second iteration is that, the company agreed upon the modification of long feedback loop found at system and release level and further, the importance of creating the automated tests in these levels instead of manual tests which prevent the continuous integration. In the Iteration 3, the suggestion to trace the requirements with the developed tests was validated and accepted by all the parties involved and carried out to the next phase.

3.1.4 Implementation

The suggested and evaluated design solution will be implemented in this phase. The complexity of the implementation phase depends on the complexity of the identified problem and design solution for that problem. In all iterations, usually the implementation phase is represented in different forms such as, developing a prototype, developing a software or developing a model. The implementation of

Iteration 1, is conducting a CIViT workshop at the case company with key employees. More details about the workshop are found in Section (4.1.4). While in the Iteration 2, the implementation is to develop automated tests. More details about this can be found in Section (4.2.4). The automated tests will be run on daily basis instead of waiting until the release time. This in turn will reduce the feedback loop and reduce the challenges that prevent the success of continuous integration at the company. The implementation of Iteration 3, is to map the requirements to the tests and report the results in HTML format. The detailed results of this can be found in Section 4.3.4. The outcome of this phase is to validate whether the implemented design solution addresses the desired goals.

3.1.5 Evaluation

This is the last phase of the regulative cycle. The outcome of the previous phase i.e. Implementation phase, will be evaluated in order to analyse if the implemented design solves the problem addressed. During all the iterations, this phase was carefully performed in order to be sure about the design that is implemented met the desired goal. The study that is performed in all the iterations gives us a clear result that is attained during each phase of the iteration. This in turn helped us to gain a comprehensible picture of what can be done for the following iterations. Thus, the outcome of this phase is considered as an input for the next iterative cycle.

3.2 Interviews and Surveys

In this study, both interviews and surveys are used to collect the qualitative data. The surveys are used in research in order to give a strong evidence for the research done [22].

Interviews are popular ways to collect qualitative data. There exists two types of interviews that can be conducted to elicit data. The individual interviews are conducted where an interviewer interviews a single person 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. These interviews are conducted by either one or two interviewers [23, 24].

Interviews can be structured in different ways. They can be semi-structured, fully-structured or unstructured [25]. 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 [25, 24]. While in an unstructured interview, the questions are neither determined nor ordered. The questions would be mostly conventional for such kind of interviews [25].

During the study, a focus group interview was conducted as part of iteration one in order to bring up the CIViT model for the company. The interview is in the form of a workshop which lasted for an hour. It requires active involvement of researcher during the group interview which is considered as a main ingredient for good data collection [26]. Furthermore, the focused group interview is semi-structured which are almost identical in the way that they are quite casual in nature. This involves participants to respond in their own views and bring up long conversations [24]. The focus group interview conducted at the company involved 5 key employees who have more than four years of experience. Some semi-structured questions are prepared and asked during the interview to elicit the information needed. While as part of iteration 2, a survey was carried out to measure the confidence level of the employee with respect to the company's product and fixing the broken builds. The questionnaire used for the survey can be found in Appendix A.2.

4
Iterations

In this chapter, the work done during each regulative cycle is presented. Furthermore, the results attained for each cycle have also been discussed.

4.1 Iteration 1: Implementing CIViT

The main purpose of this regulative cycle is to assess the current test activities and be able to identify the key problems which needs improvement.

4.1.1 Investigation of Problem

In order to understand the current testing activities at the case company, we firstly had a meeting with the industry related people and further participated in the introduction week organised by the company to get familiar with their tools and to have a better understanding of AUTOSAR. Furthermore, by using their tool, which is an Eclipse based configuration environment for automotive software has provided us experience to start up the research to understand the problems that prevent continuous integration from the perspective of testing. Furthermore, to envision the obstacles in testing. From the literature [17], mentioned the major factors that effect the adoption of continuous integration as related to the mindset of the developers, the tools for testing and the capability of the infrastructure. The slow feedback loops of test results, testing the quality attributes lately and misunderstanding of the end-to-end testing process are identified as challenges for achieving successful testing [8].

4.1.2 Suggesting a Design Solution

In this phase, we aimed at further understanding the testing activities through reading more related work. During the research, we found that the purpose of CIViT model is to visualise the end-to-end activities of testing and further to focus on the best ways to support the transformation towards continuous integration [8]. As a result, we concluded that the model can be used as a good starting point as a test strategy that can guide us into knowing the company's testing activities and to have a visualised picture of their testing activities at different levels of development processes and also build a stage for all the developers to bring up discussions about the processes they follow.

4.1.3 Validation of the Design

The idea was then discussed with supervisors at the case company as well as at Chalmers university about the CIViT workshop and how the results of the model would be providing us a full understanding about the processes that need further improvement and identifying the problems involved. Apart from this, we attended a workshop conducted by one of the authors of CIViT model and we got an opportunity to validate our design against the original model. During the discussions, it was found that the suggested design is appropriate and all parties agreed upon the design.

4.1.4 Implementation

In this phase, the implementation of the design solution, i.e. the CIViT workshop took place. The key employees at the company were involved in the discussion and it lasted for an hour. The testing activities were discussed at each level of development process. Through several discussions and debates, the CIViT model was developed and by the end of the workshop, the model is validated again. The outcome was discussed with the employees present during the workshop and turned out satisfactory. The developed CIViT model at the CIViT workshop is shown in Figure 4.1.

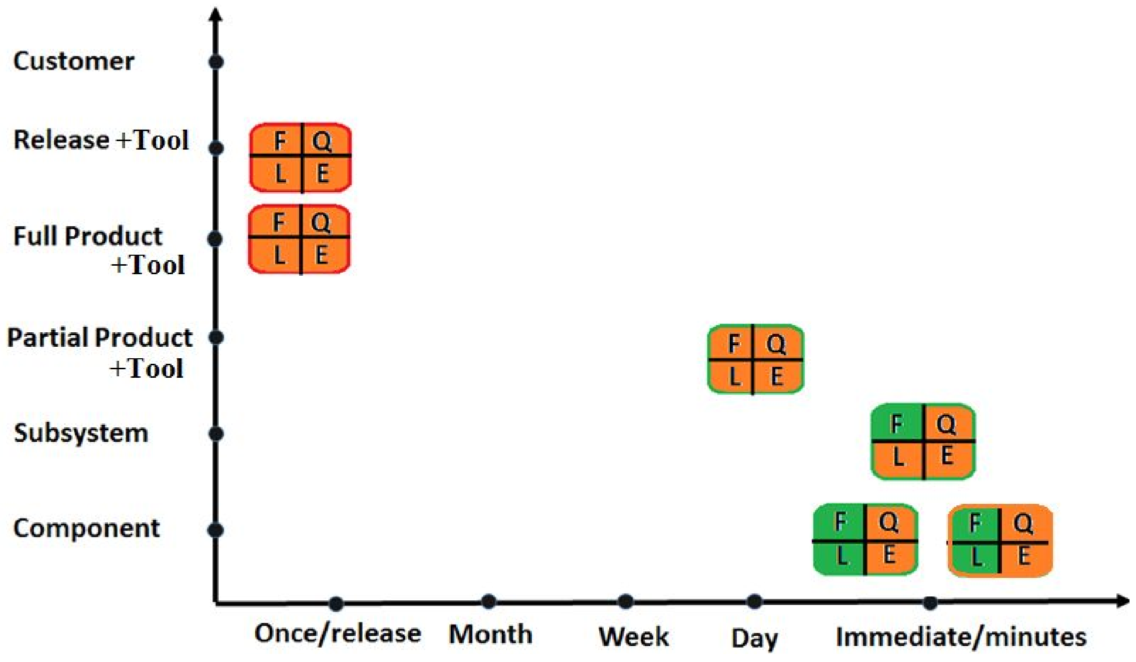


Figure 4.1: CIViT model from the case company

On the y-axis, the different types of testing activities (component level to customer) are present and on the x-axis, the time taken (Once per release to immediate/minutes) to perform the tests is present. The result is depicted as follows: At the component level, the time taken for the tests is immediate and some tests are fully automated whereas some of the tests involve the hardware, so making it partially automated. At the subsystem level, the feedback for the tests is at an immediate level and here the tests are fully automated. Furthermore, at the testing of the partial product is carried out on a daily basis and the tests are fully automated. The testing at the full product and the release level is carried out once per release and is carried out manually. At the component and the subsystem level, the functional and the legacy requirements (Except for the subsystem level it is partially covered) are covered completely. Whereas, for partial, full product and the release levels, functional and legacy requirements are covered partially. Furthermore, the quality and the edge cases are not covered at all.

4.1.5 Evaluation of Iteration 1

The goal of this iteration is to identify and assess the current test activities and challenges involved. According to the developed CIViT model during the workshop, it provided a rich and clear picture about different testing activities at the company and the way they are arranged in the organization. The CIViT workshop was used as a part of the test strategy in the study which has helped the developers to gain an understanding regarding their tests and to find the missing parts which needs more focus in future. The CIViT model specifically allowed to assess and visualise about the improvements needed in testing activities in order to address it and attain better support for continuous integration. Furthermore, from the Figure 4.1, it is evident that the case company has partial testing activities on *"Edge Cases"* and *"Quality"* which were only realised after the CIViT workshop. This in fact has provided a clear information to the company on the areas to focus in the future. Furthermore, the testing at the Full Product and Release level are carried out at the same time with no enough time for improvement if any defects are found at Full Product level testing. This is explicitly visible in the CIViT model and found to be a need for improvement on the first hand. During the workshop, it was also found that the developers have no explicit view of the testing activities i.e. the visibility of the end-to-end testing activities is missing.

The model also enabled the company to check if any duplication of the testing is taking place. The developers found that the CIViT workshop has given them an opportunity to be able to discuss about the testing activities at different stages of development processes and share the difficulties and solutions they found to resolve some common problems. It was clear that there exists no duplicates in testing activities and is a good sign for the company. Whereas, the feedback loops at product level which is tested once in three months seems to take long time and needed immediate action. Furthermore, testing done at this level is manual and the need to automate the tests was recognised which also helps to move the Full Product level testing to daily. It was further found from the results that on the sub system level, building the tests takes long time which is not desirable by all developers.

The major findings from the implementation of CIViT workshop in the Iteration 1 are summarised as shown in Table 4.1.

Index	Findings
1.1	Provided better understanding of the testing activities
1.2	The CIViT workshop can be used to assess and visualise test strategy
1.3	Partial testing activities on ”Edge Cases” and ”Quality”
1.4	Tagging of the requirements is difficult when having several unit tests for each module
1.5	Full Product and Release level testing is carried out at the same time
1.6	No explicit view of the end-to-end testing activities
1.7	No duplication of the testing exists

Table 4.1: Summary of the findings from Iteration 1.

The CIViT workshop further allowed us to discuss the challenges involved in the development stages and we found one of the challenges is to map the requirements specification with the test cases. We found that the software testing is an important aspect of software development process which improves the correctness of software and evaluates whether the software meets its requirement specifications [27]. Thus, mapping the requirements with respect to the tests executed, is also considered important in order to be able to trace the requirements easily and lessen the time and effort to trace them later. The identified problem is addressed in Iteration 3. We further found that there is a need for improving the testing effort for better software development process to take place.

Later, after careful evaluation of the developed CIViT model, we concluded that there are a set of challenges that are needed to be addressed. So we prioritised the challenges based on the discussions with employees at the company.

Firstly, it is found that there is a need for automating the Full product level. Currently, it is manually tested and through automating the tests, it is much likely that the feedback loop time is reduced and an improvement in continuous integration can be seen. Since, a right test strategy with good level of automation helps to ensure the efficiency of test strategy which is considered to be very important [28]. Secondly, another challenge that was found is to map the requirement specifications with the test framework in order to be able to trace the requirements with respect to the tests. Considering the level of severity and the need, we prioritised to work on automating the tests in second iteration and further with mapping of the requirements with the tests in the third iteration. Furthermore, in order to execute the third iteration plan, it is needed to have automated tests.

A questionnaire is also sent to the employees that participated in the CIViT workshop to elicit their thoughts about testing and also to know if they face any more challenges regarding feedback loops and tests that they did not want to discuss in an open group discussion in workshop. The questionnaire can be found in Appendix A.1. Considering all these issues, we prioritised the tasks that needed improvement and we decided to go further with automating the tests as our first priority.

According to the CIViT literature, the authors have successfully implemented the

CIViT workshop to visualize the end-to-end test strategies at six companies [3]. In our study, we implemented a similar workshop to visualize the test activities at the company. Furthermore, we used the CIViT as a test strategy that helps to visualize the activities as well as to find the challenges included in development process. These challenges were later prioritised and implemented in the later iterations. The suggested changes for making the full product to automated and moving the testing activities to monthly or weekly is shown in the Figure 4.2.

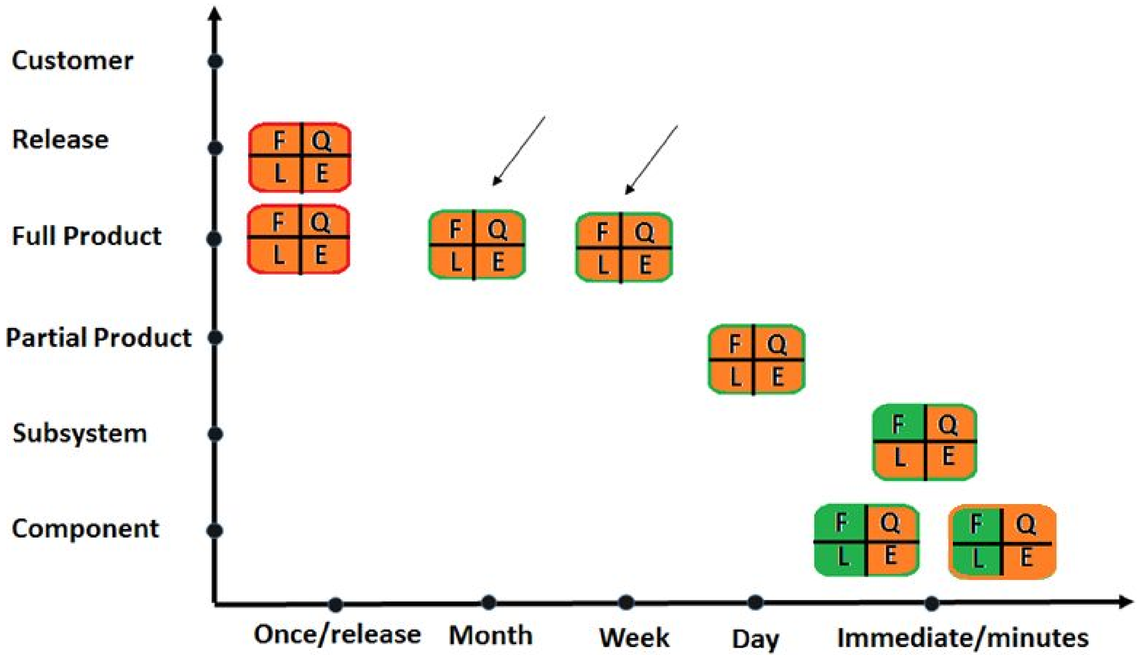


Figure 4.2: CIViT model with suggested changes to the company

Apart from this, the results obtained by implementing the CIViT workshop were satisfactory for the company, since they were able to visualise their current status in the testing activities and have a clear overview of the work processes. They would like to implement such workshop on an yearly basis.

4.2 Iteration 2

The aim of this iteration is to reduce the testing efforts by making parts of system level testing to automated from being manually tested.

4.2.1 Investigation of Problem

In this phase, the result from the first iteration is taken as a starting point for refining the problem investigation in the second iteration. As mentioned in Section 5.1, it is found that there is a need for more focus on full product and release levels. Since the testing at these two levels is carried out manually and have the longest feedback loop. So, the problem of investigation is to find an appropriate way to solve the problem.

4.2.2 Suggesting a Design Solution

The suggestion in this iteration is to automate the tests at the full product and release levels to reduce the effort needed to test in the last moment i.e. once in three months. Furthermore, the plan to automate the tests is considered as a part of the test plan that need to accomplish an effective test strategy. For the suggestion to be implemented, there is a need to identify which tests should be automated in the AUTOSAR Basic Software. After discussions with the employees at the company, we found that the Abstraction Layer can be the first step to be automated and tested. So, a suggestion was made to reduce the feedback time and a plan is suggested to test the specifications on a daily or a weekly basis.

4.2.3 Validation of the Design

After suggesting the design, it is again validated by correlating it to the results and discussions from the employees during the CIViT workshop. The reasons for focusing on these two levels (Full Product and the Release) is that, the hardware is included during the testing of the software and the lack of automated tests is in fact taking lot of time to give feedback. Furthermore, making it difficult to fix the errors that are found at these levels, due to the tests taking place in the last moment before the release. Furthermore, we found from the discussion from the employees during the CIViT workshop, the testing at the release level requires extra effort from the employees to set up the test environment. All these indicate to validate our suggested design.

4.2.4 Implementation

In this phase, the task to develop automated tests for the system level is implemented. The initial plan was to develop and run them on a daily basis and check the performance of the tests. As a first step in implementation, we automated tests of *Can Communication Stack* along with the *Memory Stack* and the *Watchdog*. Furthermore, as a later part of the implementation we automated the tests for *Ethernet*. These tests are developed based on the test specification that are followed for manually testing them at the case company. Furthermore, a python test framework was taken as a base for automating the tests and to reduce the effort for automation. During this phase, the software is run and tested on the hardware i.e. the Electronic Control Unit (ECU). Finally, the test cases are developed and the intended plan to automate the test specifications was implemented.

In contrast, during this phase we also conducted a survey to measure the confidence level of the employees in order to assess the impact of confidence level of employees on fixing the broken builds. Details about the results of the survey are found in Section 4.2.5.

4.2.5 Evaluation of Iteration 2

The goal of this Iteration was to do automation at system level and with an objective to getting to weekly or to a daily scope. Furthermore, we evaluated if this is possible and whether tests could be run on a daily basis. The outcome is successful and we developed automated tests using python framework, that are used for testing the software at system level. The system included two parts: the hardware and the software.

The tests were developed on top of the python test framework, based on a set of test specification followed at the case company. Since, the automation on the system level was missing; during this study we drew the path for the testing infrastructure. This makes it easier to add the automated tests in future. As discussed during the CIViT workshop in iteration one, the system level testing is carried out just before the release, which possesses a major risk during the release time. Errors could be found in the last moment and the delivery to the customer can be delayed. In order to reduce this risk and further to improve the feedback loop time, we automated the test specifications. Figure 4.3, shows the difference between before and after test automation. The figure in the left panel shows the result of the CIViT model after the workshop in Iteration 1. Whereas the figure in the right panel, presents the changed CIViT model after implementing the automated tests. In this, the testing at the full product level has been moved to a daily basis. The tests on the system level are successfully automated. According to the outcome suggested in Section 4.1.5, that the tests will be run on a weekly or monthly basis; the suggested plan was modified and the tests are run on a daily basis. The reason behind to shift the decision is to speed up the process of fixing the errors as the information is fresh on the developers mind. This servers the main core of achieving continuous integration.

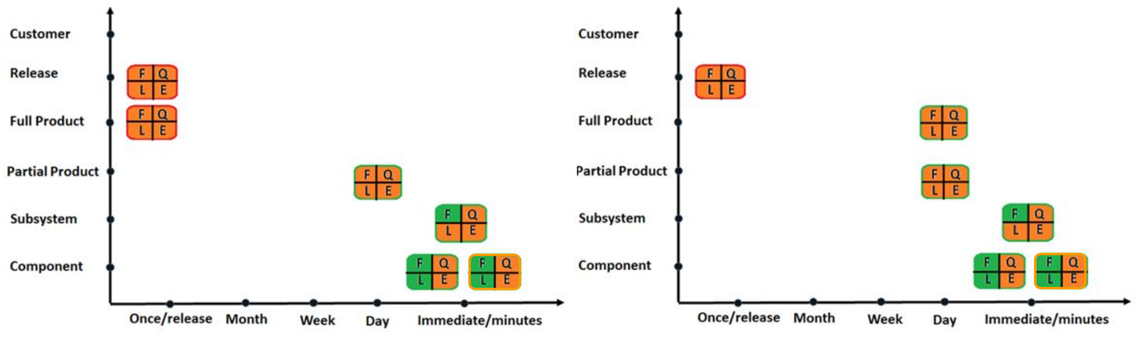


Figure 4.3: Left panel: CIViT model from the case company before test automation. Right panel: CiViT model after after test automation.

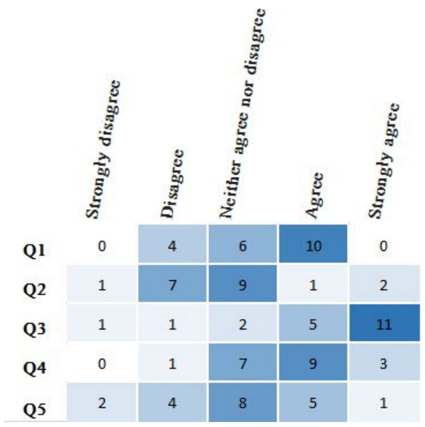
Furthermore, before the tests were automated, it took almost an hour to perform them manually. The time taken also differs from developer to developer to perform these tests manually. If the system is set and everything required for the tests is in place, it would take half an hour to an hour. In this case, if done by an experienced developer. Whereas, if the developer is new to the system and everything has to be set up from the scratch; it would take nearly half a day to perform these tests. This shows that the experience and time taken are directly related in performing the tests. After we developed the automated tests, the time taken to run these tests is less than a minute and this indeed satisfied the need of the developers as well as reduced the effort needed in testing the system.

Furthermore, as a part of the results that we obtained in this iteration: there are many hardware modules that are hard to implement and test. There is a need to have the test automation on the system level for supporting these difficulties. Otherwise, it is hard to perform the testing. Implementing the test for the system level at an early stage allows to test the software gradually during the development process and including the hardware. Also detecting errors as early as possible helps to mitigate the faults faster when everything is still fresh on the developer's mind. As a part of our conclusion, it is possible to automate all the tests and it took four weeks of our effort to develop these automated tests and run them successfully.

As discussed in Section 4.2.4, we focused on conducting a survey to assess the confidence level of the developers within embedded systems (Appendix A.2). The purpose of the survey is to analyse the behaviour of the developers and their interest to fix a broken builds which can have an affect on the development process. Also, if there is a lack of confidence level of employees then it would be hard for them to follow the test strategy and understand its value in correct way. This in turn can have an effect on the achievement of continuous integration. The survey was conducted among 20 employees at the case company located at Gothenburg and the experience level of the developers ranges between few months to ten years of experience. This variation in the experience level, can also show an effect on the results attained.

The confidence level may vary from one developer to another based on their experience, work environment, backgrounds and many other reasons. The Figure 5.4, below shows the results of the survey and by carefully analysing it, we found that a majority of the employees think that it is their responsibility to fix the build.

When asked about if they would really take responsibility to fix the build, majority of them neither agreed nor disagreed. This shows that they were interested in fixing the build but the confidence in fixing the build is actually affecting it directly. Furthermore, forcing only few employees to take responsibility. This in fact has provided a strong base for us to argue that testing of system software is much more important to do in the early stages of the development, in order to avoid the last minute rush to fix the errors.



	Strongly disagree	Disagree	Neither agree nor disagree	Agree	Strongly agree
Q1	0	4	6	10	0
Q2	1	7	9	1	2
Q3	1	1	2	5	11
Q4	0	1	7	9	3
Q5	2	4	8	5	1

Figure 4.4: Evaluation of the survey for measuring confidence level of the employees

In order to analyse the above founded results from the survey, we conducted semi-structured interviews with three employees at the company. These interviews helped us to investigate the reasons behind the lack of confidence level in the employees. The major findings were that the complexity of the system and the lack of overview of documented system architecture makes it hard for the developers to understand the complete system. As well as, lack of expertise in other parts of the system also leads to failure in fixing the builds as fast as expected. For example, each employee feels that they are responsible for the part of the project they take up and it is hard to find someone who is well versed with the complete system knowledge. Apart from this we also found that the mindset of the employees has an impact on the confidence level as well as taking the responsibility in fixing the builds. For example, one interviewee said

"I mean some people that have that mindset, I am not the cleaning guy and someone else has to do the cleaning, I only do a new development. If you have that mindset, it will have to cost lots of cleaning work for others."

This shows that, the mindset is directly effecting the success of continuous integration. Furthermore, having the time to fix the builds is also found as a crucial factor for all the above said aspects. Since, the developers find that the work they are allotted to complete is their first preference and do not have time to fix the broken builds. An other crucial finding is that, there is a lack of testing. According to one of the interviewees, the code is developed and unit tested by the same person. Making it difficult to cover all the test cases. This effects the confidence level because they are missing different perspectives in testing which in turn the developers are unable to assure that they cover every part of the specifications.

Thus, from the results of iteration two and the result of assessment of confidence level of the employees, it is evident that the confidence level of employees and their performance during the development process have an impact on achieving continuous integration. The major findings from the Iteration 2 are summarised as shown in the Table 4.2.

Index	Findings
2.1	Developing the automated test for the system level helped to reduce the time and effort taken for testing.
2.2	Low confidence level of developers in fixing the build is directly affecting the development process
2.3	Full Product and Release level are carried out at the same time.
2.4	Complexity in the systems prevent from fixing the problems quicker.

Table 4.2: Summary of the findings from Iteration 2.

The outcome of the Iteration 3 helped the developers to reduce the time taken for testing as well as it meets their need for gaining quick feedback.

4.3 Iteration 3

The aim of this regulative cycle is to further improve the tests that are developed during the Iteration two, through mapping the specified requirements in the database to the corresponding test framework.

4.3.1 Investigation of Problem

In this phase, part of the outcome of the second iteration was considered as an input for investigation for the current iteration. As mentioned from the Iteration 1, the challenge of requirement traceability is taken as a problem to investigate and solve. Furthermore, during the Iteration 2, we faced some problems in tracing the corresponding requirements to the tests that failed. Thus, a need for mapping of the requirement specification to the tests developed was identified.

4.3.2 Suggesting a Design Solution

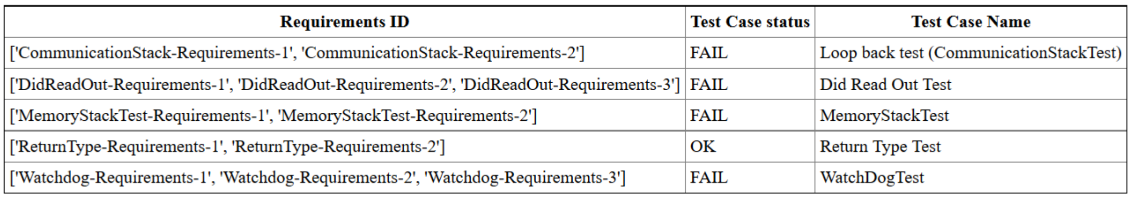
During this phase, the suggestion was to map the requirements and create a connection between the requirement specification in the database and the test framework. Furthermore, the result of the tests and the corresponding requirements to these tests was planned to be visualised as a HTML report.

4.3.3 Validation of the Design

After suggesting the design, it was then discussed with the supervisors to validate the value the design and assess its impact on solving the problem. Furthermore, this problem has also been identified as one of the challenges faced during the CIViT workshop and the plan has been approved as a value for improving continuous integration.

4.3.4 Implementation

During this phase, the suggested design to map and visualise the results of the requirements covered was implemented. Due to the time constraint, the implementation of this iteration has been divided into two parts. The first part is to extract the requirements from the database and further run the tests framework. The result of the tests have been combined with the related requirements and visualized in the HTML report. The second part is to take the results of the tests and re-sending them to the database. Thus, for this phase, we mainly focused on to visualise the results in HTML report. The first part was successfully accomplished. The Figure 5.5 below shows the final outcome of the implementation.



Requirements ID	Test Case status	Test Case Name
['CommunicationStack-Requirements-1', 'CommunicationStack-Requirements-2']	FAIL	Loop back test (CommunicationStackTest)
['DidReadOut-Requirements-1', 'DidReadOut-Requirements-2', 'DidReadOut-Requirements-3']	FAIL	Did Read Out Test
['MemoryStackTest-Requirements-1', 'MemoryStackTest-Requirements-2']	FAIL	MemoryStackTest
['ReturnType-Requirements-1', 'ReturnType-Requirements-2']	OK	Return Type Test
['Watchdog-Requirements-1', 'Watchdog-Requirements-2', 'Watchdog-Requirements-3']	FAIL	WatchDogTest

Figure 4.5: Result in HTML

As you can see from the Figure 4.5, the first column 'Requirement ID' represents the requirements from the database and the second column provides the 'Test Case Status' which is 'OK' or 'FAIL'. Whereas, the third column represents the 'Test Case Name' from the executed test framework.

4.3.5 Evaluation of Iteration 3

The outcome of the third iteration was satisfactory. The results of the test framework, from the Iteration 2 were taken as a base and further these results have been merged with the related requirements in the database. Furthermore, all these results have been developed into HTML table.
The solution now helps to visualise the requirements that are covered by the test cases and in turn become much easier for developer/testers to check which requirements have been covered corresponding to the test cases. The test cases that have been passes or failed are also visible. This solution increase the traceability of requirements during the development process. The ability of being able to trace the requirements is considered as one of the important aspects of the automated test and for attaining continuous integration. This is the last iteration of this study and all the identified challenges (Figures 4.1 and 4.2) have been addressed by the end of the iteration.
Apart from this, we also took feedback from the employees about the outcome of the work carried out during the Iteration 3. The employees were happy and satisfied with the outcome, since the implementation has added a value to connect the requirements specifications to the tests. This further makes it easier for them to locate the tests that failed and having a clear summary of the tests outcome.

5 Results

5.1 Research Questions

In this section, the results of the thesis with regards to the research questions are discussed. In the following subsections each research question has been discussed.

5.1.1 Research Question 1

What is the current test strategy?

This question answers about the current test strategy that are used during the development processes at the organizations. The literature highlights the importance of the test strategy to detect the errors and fix bugs in order to deliver a product faster according to the customer requirements. Throughout our study, we found an implicit test strategy in order to attain the goal of continuous integration.

Firstly, in the Iteration 1 by conducting a CIViT workshop, we visualised the test activities that take place within the entire development process and according to the literature, the CIViT workshop was implemented among six companies and the workshop successfully helped to visualize the test activities [8]. From this, we concluded that the CIViT workshop can be applied as a visualising tool for testing which can be implemented as a part of test strategy. The reason behind this is, the CIViT provides a bird's eye view of the current test activities and further provides a clear picture of the backlogs and challenges being faced during the development process. The CIViT model further is considered as a light way of documentation for the test strategy in this study. Furthermore, helping us to prioritise the work to be carried out in the future iterations. Conducting such kind of CIViT workshop within a test strategy, shall definitely help any software developing organization to have a visual path for their testing levels and allows to continuously improve the test processes.

Secondly, as discussed under the Section 4.2 in Iteration 2, the plan to automate the system tests for the software modules was chosen. Test automation and execution is considered as one of the important aspects to achieve the goal of testing and to reduce the time and effort taken to test the software by the developers/testers. This will also help to fix the errors as early as possible. From the literature, we found that there are many benefits from automating the tests such as saving the time, improvements in the system implementation and also reduces the effort needed for the developers to execute the tests [10, 29, 30]. Furthermore, for attaining a successful test strategy, having a good level of

automation is very critical, especially in automotive testing [10]. This enhances the importance of the automated testing as a strategy for complex systems.
Lastly, for the Iteration three, we mapped the requirements corresponding to the tests executed and be able to visualise the results in HTML report. In order to address the issues of requirement traceability and to make sure that the key requirements are not ignored, the plan to map the tests and the requirements has helped to solve those issues.
From all the three iterations that took place during the study, a good test strategy for continuous integration should be able to visualize the different test activities of the testing at various levels of development process. This helps to support the cross functional teams and further helps the developers/testers to identify the weaknesses and show a path for further improvement. Furthermore, the test strategy should allow to automate the tests on the system level in order to facilitate quick feedback and remove delays from manual testing. Having such an automation shall help to reduce the effort needed in testing and to speed up the development process. In turn, allowing to deliver the product to the customer on time with minimal errors. Apart from this, a test strategy should be able to map the requirements specification with the related tests in a proper way. Additionally, such strategy shall ensure to have a good level of test coverage of requirements and improves the system correctness in order to achieve the customer satisfaction [31]. Furthermore, executing the tests and making sure that the requirements are met, is a key to have a successful product. This in turn shall help to attain high level of quality [32]. This in fact helps the developers to navigate easily and act quickly to fix the errors where necessary. All these above mentioned points are considered as vital ingredients for all the software companies that strive to obtain suitable test strategy for continuous integration.

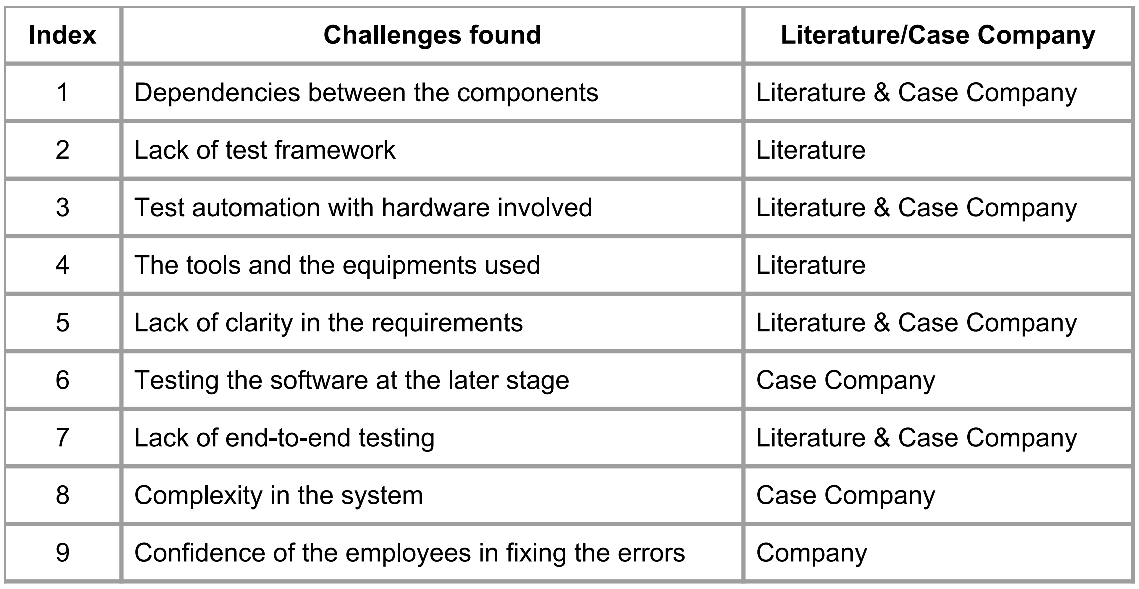
5.1.2 Research Question 2

With respect to test strategies what are the challenges that prevent the success of Continuous integration?
The answer for this question relies on the gathered data from two parts in the study conducted. Firstly, from the literature review and secondly, from the the findings at the case company. From the literature, we found that there are many challenges that exists during the development process that prevent the success of achieving continuous integration. One of the challenges identified is handling of dependencies between the components of the system during different stages of development process. Furthermore, lack of test framework is considered as one of the most important challenges that affect the success of achieving continuous integration [17]. Furthermore, the challenge also lies in the tools used for reviewing the integrated code and its infrastructure. Having too many tools for the tests will also effects the desired goal of testing. These are also considered as equally important for attaining the continuous integration [33]. Furthermore, similar results were identified that the lack of the testing tools and also lack of availability of the test equipments may lead to affect the ability of automated tests. Also, the lack of clarity in the requirements during the development process makes it hard to

be traceable. The testing of the software at the later stage of software development shall also cost lot of time and effort to fix the errors [31]. These challenges in turn affect the probability of efficiency of test strategy that can be followed during the software development process. Furthermore, preventing the success of continuous integration.

Secondly from the study that took place during the thesis, it was also found that the clarity of the requirements is an other challenge as well. The automated tests emphasises on checking and testing the code on a daily basis. Especially with companies in embedded software design, it is very important that the software is tested on the hardware before the release and automating these tests from manual, is considered as a qualification to attain continuous integration. This in turn can enhance the quality and maintainability of software product during the development. There were several other challenges that we found. The CIViT workshop that was conducted during the first iteration has indeed helped us to interact with the employees and trigger discussions on difficulties they have during development process. Firstly, there was a lack of end to end testing. Not every developer has a detailed insight of the testing activities. Secondly, the lack of availability of automated test suites especially in the full product and release levels. These levels are dependent on the hardware for the tests to take place and the tests are carried out semi automatic which leads to consume a lot of time and effort in testing the product.

Furthermore, from the survey conducted to measure the confidence level of the employees at the case company has given a base to us to understand the challenges that are being faced by them. We found that, due to the complexity of the system; the employees find it difficult to fix the problems quicker in-spite of having a genuine interest in fixing it. This also reflects on the confidence level of the employees in fixing the errors. Lastly, we found that not everyone commit their work every day which hinders the goal to attain the continuous integration. From the literature and case company there are several challenges that were identified in the Figure 5.1 below and also from where we gain these results.



Index	Challenges found	Literature/Case Company
1	Dependencies between the components	Literature & Case Company
2	Lack of test framework	Literature
3	Test automation with hardware involved	Literature & Case Company
4	The tools and the equipments used	Literature
5	Lack of clarity in the requirements	Literature & Case Company
6	Testing the software at the later stage	Case Company
7	Lack of end-to-end testing	Literature & Case Company
8	Complexity in the system	Case Company
9	Confidence of the employees in fixing the errors	Company

Figure 5.1: Summary of the challenges found from Literature and Study from Case Company

5.1.3 Research Question 3

What is a/how is a suitable test strategy for Continuous Integration(CI) characterised?

The answer for this question has been gathered from the three iterations conducted during the study as well as from the literature review.

Based on the literature, the test strategy is all about illustrating the path of the tests and defining the steps for the developers/testers [11, 12, 10]. It follows some unique characteristics that make it suitable for supporting continuous integration. Otherwise, it will be hard for the developers and testers to have an appropriate testing that guarantee achieving a successful continuous integration.

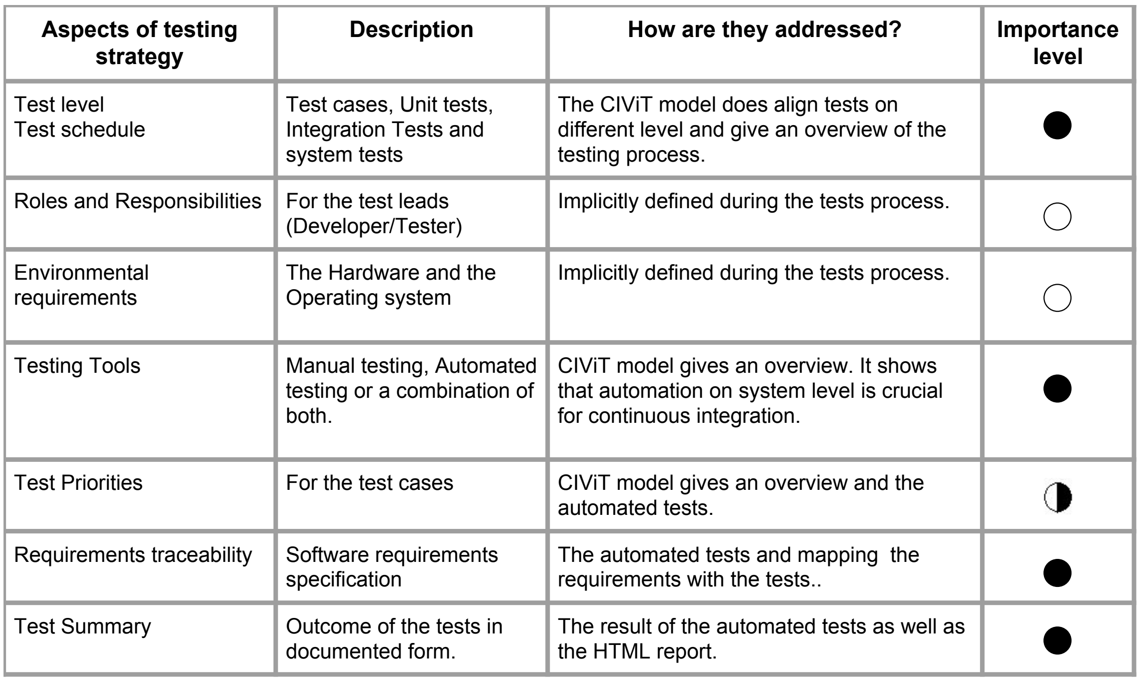
A good test strategy for continuous integration should explicitly define the roles of the developers/testers and further possess a clear description about what is to be performed during the testing. The test environment is also equally important for a good test strategy. Furthermore, to execute the tests there exists certain tools such as manual, automated or mix of both [11]. The test strategy for continuous integration is important to be automated to support its principles of getting a continuous feedback on the tests. Having a clear description of the test level is one of the important characteristics of test strategy for continuous integration. Furthermore, a test strategy must have a clear implementation plan that helps to specify the required time for testing and estimates the time taken for testing. The test strategy should have a reasonable number of test cases to be run with a high level of prioritization [30]. This helps the testers/developers to understand the most important tests that need focus on to achieve a good quality of the code when continuously integrating it. The clarity in requirements and clear definition of the relation between the requirements and test activities is important. For example, requirements that are not described properly and testing that takes place without traceability of requirements is considered as a challenge, that makes it hard for developers to detect the location of errors and fixing them on time. [31]. From the case company, the environment is also found to be an important characteristic for a successful test strategy. For example, in order to perform the implementation phase in the Iteration 2, we needed Windows x86 MSI installer and Python 2.7.11 to run the automated tests successfully. Finally, we also found that, a test strategy must possess a clear synopsis of the tests results on daily basis. These reports can be either presented in a PDF or HTML form for the convenience of the reader and further these results shall include the related information of the tests that are executed as well as describe the purpose of the tests and the outcome of it.

6 Discussion

Through this study we can summarize the test strategy as a process of drawing the software testing approach. Having a properly defined test strategy is consider as an important part of testing process. Generally, a test strategy for any software development process follows standard characteristics. When it comes to continuous integration, not all these characteristics are equally important to support the test process. The test strategy for continuous integration should allow testing frequently the software and be able to visualize all the test activities to make it clear for the developers/testers to understand the plan of testing. Furthermore, it should also provide a clear traceability between the requirements specification of the development process and the tests.

The test strategy should describe clearly all the test levels and the purpose of these levels. Furthermore, it should also define the individual responsibilities for the people involved. A clear plan for scheduling the time for executing the tests and the testers should be able to prioritise the tests while following the test strategy. From the Iteration 1, it is evident that the implementation of the CIViT workshop helped to achieve the above mentioned points. Thus, the CIViT model that is developed during the study can be considered as a part of the test strategy that provides a simple way of documentation. It visualises various tests types at different levels of development process, its periodicity, coverage level of tests and the level of automation of the tests. Apart from this, we also found that a test strategy for complex systems should allow the automation of tests and further provide a clear report on the test results that are executed. Having a clear traceability for the requirements is also equally important for the test strategy. Which means, it should have a clear mapping between the running tests and its related requirements. From the Iteration 2, the automation tests on the system level meet these characteristics that are defined above. The implementation of the second iteration, helped to have fully automated tests on the system level of the software which reduced the time for testing and to improve the quality of the test framework. Furthermore, the evaluation of Iteration 3, helped to address the issue of requirement traceability. In this iteration, we provided a visibility between the requirements and the related test cases. Furthermore, the results are reported in a HTML form. This kind of reporting provides a clear information about the requirements that are covered with regards to the executed test cases. It further provides the status, if the tests have been passed or failed.

The figure below summarises various aspects of testing strategy that were found during our study [9, 11, 10]. These aspects are important to follow for a test strategy that supports continuous integration and how these aspects have been addressed during the study and the importance level of these aspects during the study are indicted in the Figure 6.1. The filled circle is for high importance(i.e. has been covered during the study), half filled circle means partially covered, while the empty circle indicates that it is covered implicitly by CI test strategy.



Aspects of testing strategy	Description	How are they addressed?	Importance level
Test level Test schedule	Test cases, Unit tests, Integration Tests and system tests	The CIViT model does align tests on different level and give an overview of the testing process.	●
Roles and Responsibilities	For the test leads (Developer/Tester)	Implicitly defined during the tests process.	○
Environmental requirements	The Hardware and the Operating system	Implicitly defined during the tests process.	○
Testing Tools	Manual testing, Automated testing or a combination of both.	CIViT model gives an overview. It shows that automation on system level is crucial for continuous integration.	●
Test Priorities	For the test cases	CIViT model gives an overview and the automated tests.	◑
Requirements traceability	Software requirements specification	The automated tests and mapping the requirements with the tests..	●
Test Summary	Outcome of the tests in documented form.	The result of the automated tests as well as the HTML report.	●

Figure 6.1: The summary of test strategy characteristics for continuous integration

The study shows an explicit way to have a test strategy that helps to support continuous integration. The implication of implementing the CIViT workshop on continuous integration is to have more emphasise on different testing levels. For example, integration tests and system tests. Furthermore, it helps to empower cross functional teams by clearly communicating test strategy. The automated tests are equally important to meet the core of continuous integration, which is frequent testing and short feedback loops. Apart from this, mapping of the test cases to the requirements gives a clear layout to locate the errors directly and this further makes it easier to fix the problems in a shorter time. Furthermore, shall help to quickly meet the requirements specification and identify the one which missed to be testes during the development process. In contrast, this improves the quality of test strategy to support continuous integration through integrating frequently and providing error free parts of the system and that will increase the maintainability of the product.

Thus, from this study we conclude that having an explicitly defined test strategy and well documented shall help to improve the testing strategy in any software organization to support continuous integration.

6.1 Threats to Validity

This section defines the validity threats based on the explanation by Runeson and Höst [25].

Construct Validity: The evaluation of each iteration during our study took place in the form of discussions among the academic supervisor as well as the supervisor at the case company. The discussions also involved one of the key employees to validate the work carried out during each iteration. These discussions in fact have helped us to make sure that we have good study. Furthermore, a questionnaire was sent to the participants of the CIViT workshop during the iteration one, to evaluate the work carried out. Since, the workshop was in the form of group interview, there is a risk here that the employees to be cautious in what their talks. There is a high chance of not able to elicit proper opinion from them. For this reason, a questionnaire was sent to validate their thoughts and this in fact mitigated the risk of losing valuable information from them. Furthermore, the survey questionnaire that was conducted to measure the confidence of the employees with regards to fixing the errors; is filled by those who have less than a year experience and more than four years of experience. The difference in the experience, could have been shown an effect on the results attained.

Internal Validity: The tools used (such as the drivers) during for the testing activity in the study, are new to us and took time to learn about them. Furthermore, there are limited number of drivers that are available at the company and it was shared between the employees and us. This has in fact showed a little delay in our work, waiting for our turn to test.

External Validity: The study has been carried out at only one company and thus the results obtained during the CIViT workshop, as well as the survey results would be biased and seen in the context of the employees working in that particular company. Thus, the findings of this thesis are largely based on the information gathered from the case company and comparing the literature from the academics. We have further tested our work at the case company and by running our tests on two different environments(using different drivers). Due to the time constraint and the availability of the concerned engineers, we limited the testing of the automated tests on only one engineers system.

Reliability: The study was carried out by hugely relying on the feedback and the suggestions from one engineer at the case company, since he is the only one who is currently working on developing the tests. We have conducted the survey among the developers present at the Gothenburg office of the case company; although it has offices located in three other cities globally. The results obtained for the survey are hugely relying on the developers opinion at Gothenburg office only. Since, the majority of the developers are present in Gothenburg office and it possesses a low threat to our study in terms of validation.

7
Conclusion

This study focused on assessing the best test strategies for complex system, that strives to accomplish continuous integration. There exists many test strategies that could help to attain the goal. During this study, we identified that a test strategy should be able to visualise a clear path of the testing activities for the developer/tester, have a good level of automation with the ability to trace the requirements. The CIViT, automation of tests and mapping the requirements with the tests are considered as an important part of test strategy that supports to attain continuous integration. Furthermore, during our study we found some challenges that prevent the success of continuous integration from the side of testing perspective. These challenges can be summaries as follows: Dependencies between the components, lack of test framework, test automation with hardware involved, the tools and the equipments used, lack of clarity in the requirement, testing the software at the later stage, lack of end-to-end testing, complexity in the system and confidence of the employees in fixing the errors.

Furthermore, we found that a good test strategy must have certain characteristics. Using these characteristics as part of the test strategy shall improve the productivity of this test strategy and makes it more attractive to use. These characteristics can be summarized as follows: the need for clear documentation and high traceability of the requirements, high flexibility in implementing an fixing the errors, explicitly define the roles and responsibilities of the testers as well as the project managers and choosing the correct tools for testing.

If all the above mentioned factors have been achieved, this will lead to have a useful test strategy that guarantee achieving of a successful test environment which is considered as a challenge that prevents the success on continuous integration.

A
Appendix

In this Chapter, the questionnaires that were sent out to the employees at the case company during the study are presented.

A.1 Workshop questionnaire

The questionnaire was sent to the employees that participated during the CIViT workshop to elicit their thoughts.
1. What is your position at the company?

☐ Developer
☐ Tester
☐ Combination of both
☐ Other

2. What are you responsible for? Please provide a short description of your day to day activities at the company.
3. Does the long feedback loops at any point of development cycle disrupt your workflow?

☐ Yes
☐ NO

4. If Yes. Please explain and give your thoughts. 5. If No. How do you avoid disruption? 6. Was the CIViT workshop productive for you?

☐ Yes
☐ NO

7. If No, what was missing?

A.2 Survey questionnaire

The survey was conducted by giving out the questionnaire to the employees to fill during the breakfast meeting at the case company.
Q1. Usually, I am strongly confident about the quality of product including my latest change.

☐ Strongly Agree
☐ Agree
☐ Neither Agree Nor Disagree
☐ Disagree
☐ Strongly Disagree

Q2. I always fear that the change introduce new problems.

☐ Strongly Agree
☐ Agree
☐ Neither Agree Nor Disagree
☐ Disagree
☐ Strongly Disagree

Q3. I think it is my responsibility to fix the build when failed after tests.

☐ Strongly Agree
☐ Agree
☐ Neither Agree Nor Disagree
☐ Disagree
☐ Strongly Disagree

Q4. I am confident enough to fix a build after the feedback from tests.

☐ Strongly Agree
☐ Agree
☐ Neither Agree Nor Disagree
☐ Disagree
☐ Strongly Disagree

Q5. I should take responsibility to fix the large build.

☐ Strongly Agree
☐ Agree
☐ Neither Agree Nor Disagree
☐ Disagree
☐ Strongly Disagree

Bibliography

[1] K. Beck, M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries *et al.*, “The agile manifesto,” 2001.

[2] M. Fowler, “Continuous integration [ONLINE],” "http://www.martinfowler.com/articles/originalContinuousIntegration.html", (Date last accessed: 2016-03-23).

[3] A. Nilsson, J. Bosch, and C. Berger, “Visualizing testing activities to support continuous integration: A multiple case study,” in *Agile Processes in Software Engineering and Extreme Programming.* Springer, 2014, pp. 171–186.

[4] AUTOSAR, “Autosar home [ONLINE],” http://www.autosar.org/, (Date last accessed: 2016-03-24).

[5] D. Kum, G.-M. Park, S. Lee, and W. Jung, “Autosar migration from existing automotive software,” in *Control, Automation and Systems, (ICCAS 2008). in Proc. International Conference on.* IEEE, 2008, pp. 558–562.

[6] AUTOSAR, “Autosar home [ONLINE],” http://www.autosar.org/fileadmin/files/releases/4-0/software-architecture/general/auxiliary/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf, (Date last accessed: 2016-05-12).

[7] S. Bunzel, “Autosar – the standardized software architecture,” *Informatik-Spektrum*, vol. 34, no. 1, pp. 79–83, 2011.

[8] A. Nilsson, *The CIViT Model in a Nutshell: Visualizing Testing Activities to Support Continuous Integration.* Springer International Publishing, 2014, ch. 8.

[9] T. Excellence, “Testing excellence [ONLINE],” http://www.testingexcellence.com/test-strategy-and-test-plan/, (Date last accessed: 2016-05-24).

[10] S. S. Barhate, “Effective test strategy for testing automotive software,” in *Industrial Instrumentation and Control (ICIC), 2015 International Conference on.* IEEE, 2015, pp. 645–649.

[11] P. Ammann and J. Offutt, *Introduction to software testing.* Cambridge University Press, 2008.

[12] A. Mathrani and S. Mathrani, “Test strategies in distributed software development environments,” *Computers in Industry*, vol. 64, no. 1, pp. 1–9, 2013.

[13] J. A. Whittaker, “What is software testing? and why is it so hard?” *IEEE Software*, vol. 17, no. 1, pp. 70–79, 2000.

[14] Y. Cheon and G. T. Leavens, “A simple and practical approach to unit testing: The jml and junit way,” in *ECOOP 2002—Object-Oriented Programming.* Springer, 2002, pp. 231–255.

[15] K. Naik and P. Tripathy, *Software testing and quality assurance: theory and practice.* John Wiley & Sons, 2011.

[16] H. K. Leung and L. White, “A study of integration testing and software regression at the integration level,” in *Software Maintenance, 1990, Proceedings., Conference on.* IEEE, 1990, pp. 290–301.

[17] A. Debbiche, M. Dienér, and R. B. Svensson, “Challenges when adopting continuous integration: A case study,” in *Product-Focused Software Process Improvement.* Springer, 2014, pp. 17–32.

[18] R. H. von Alan, S. T. March, J. Park, and S. Ram, “Design science in information systems research,” *MIS quarterly*, vol. 28, no. 1, pp. 75–105, 2004.

[19] R. Wieringa, “Design science as nested problem solving,” in *Proceedings of the 4th international conference on design science research in information systems and technology.* ACM, 2009, p. 8.

[20] V. K. Vaishnavi and W. Kuechler, *Design science research methods and patterns: innovating information and communication technology.* Crc Press, 2015.

[21] 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.

[22] 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.

[23] 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.

[24] R. Longhurst, “Semi-structured interviews and focus groups,” *Key methods in geography*, pp. 117–132, 2003.

[25] 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.

[26] I. McLafferty, “Focus group interviews as a data collecting strategy,” *Journal of advanced nursing*, vol. 48, no. 2, pp. 187–194, 2004.

[27] B. Zachariah, “Analysis of software testing strategies through attained failure size,” *Reliability, IEEE Transactions on*, vol. 61, no. 2, pp. 569–579, 2012.

[28] M. E. Khan, “Different forms of software testing techniques for finding errors,” *International Journal of Computer Science Issues*, vol. 7, no. 3, pp. 11–16, 2010.

[29] E. H. Kim, J. C. Na, and S. M. Ryoo, “Test automation framework for implementing continuous integration,” in *Information Technology: New Generations, 2009. ITNG’09. Sixth International Conference on.* IEEE, 2009, pp. 784–789.

[30] A. A. Sawant, P. H. Bari, and P. Chawan, “Software testing techniques and strategies,” *International Journal of Engineering Research and Applications (IJERA)*, vol. 2, no. 3, pp. 980–986, 2012.

[31] A. Kasoju, K. Petersen, and M. V. Mäntylä, “Analyzing an automotive testing process with evidence-based software engineering,” *Information and Software Technology*, vol. 55, no. 7, pp. 1237–1259, 2013.

[32] Z. A. Barmi, A. H. Ebrahimi, and R. Feldt, “Alignment of requirements specification and testing: A systematic mapping study,” in *Software Testing, Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on.* IEEE, 2011, pp. 476–485.

[33] H. H. Olsson, H. Alahyari, and J. Bosch, “Climbing the" stairway to heaven"–a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software,” in *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on.* IEEE, 2012, pp. 392–399.