CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Practicing Continuous Integration in a Multi-Supplier Environment for the Development of Automotive Software

Master's Thesis in Software Engineering and Technology

Evio Abazi

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Master's thesis 2019

Practicing Continuous Integration in a Multi-Supplier Environment for the Development of Automotive Software

Evio Abazi

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

Practicing Continuous Integration in a Multi-Supplier Environment for the Development of Automotive Software
Evio Abazi

Supervisor: Darko Durisic, Miroslaw Staron, Department of Computer Science and Engineering
Examiner: Jan-Philipp Steghöfer, Department of Computer Science and Engineering

Master's Thesis 2019
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2019

Practicing Continuous Integration in a Multi-Supplier Environment for the Development of Automotive Software
Evio Abazi
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Background: Software development in the automotive industry is in transition from the traditional V-Model to the adoption of Agile methods. Continuous Integration (CI) is one of the most adopted practices when working Agile since it enables faster feedback, workflow automation and allows constant testing activities. However, in the automotive software development process are involved OEMs and multiple suppliers; the practice of continuous integration in similar projects may present several challenges.

Objective: The study aims to investigate the problems related to the practice of continuous integration in a multi-supplier environment for the development of automotive software. Moreover, it identifies the root causes of such problems and provides potential solutions for preventing them.

Methodology: The thesis is based on qualitative research. An explorative case study was conducted at the development sections of an OEM company and a Tier 2 software vendor. Observations and interviews at both companies allowed to discover the main problems related to the adoption of continuous integration. A Pareto analysis, in combination with Cause-and-Effect diagrams, identified the root causes that had a major impact on the discovered challenges.

Results: Eight challenges are identified; their root causes are related to the development tools adopted, lack of synchronization and barriers to effective communication. In addition, potential solutions for the causes are provided and implemented in a demonstrative CI environment for the development of a sample application.

Conclusions: The study reports the challenges of practicing continuous integration in automotive software development. In addition, the results show opportunities for improvement for the identified problems. Since it is a single case study, the generalizability of the results is still limited; however, inputs are provided to the companies for improving their development process, and valuable research insights are provided into the obstacles associated with the practice of continuous integration in automotive software development.

Keywords: software development, automotive, embedded systems, agile, continuous integration, cinders, challenges

Acknowledgements

I would like to express my sincere gratitude to both my supervisors, Darko Durisic and Miroslaw Staron, for the great patience and constant feedback during the project.

A special thanks also to Anders Kallerdahl, for his support and for sharing his time and expertise. I was very fortunate to work with you.

Evio Abazi, Gothenburg, June 2019

Contents

List of Figures

List of Tables

1
Introduction

1.1 Problem Description

During the last two decades, in the automotive industry, significant changes took place in the way of developing ECU[1](Electronic Control Unit) software. Advanced functionalities are getting more and more software-based, and new features need to be continuously provided, raising complex requirements for next-generation vehicles [2]. In fact, the quantity of software in cars is growing exponentially and software can be a key differentiator in the automotive manufacturing industry [3] [18]. This leads to software-intensive ECUs with a significant level of complexity and interdependency, which is not manageable by the traditional development approach [4]. For example, autonomous driving systems and the vehicle-to-everything technology (V2X) require high processing power and communication systems that can support large bandwidth. In this regard, modern high-performance ECUs combined with the new software architecture introduced by AUTOSAR [1] - a standardized reference architecture for the development of automotive software - are a strong foundation for building complex systems.

The traditional software development methodology includes the use of V-Model where the process is based on sequential development phases, followed by testing activities corresponding to each stage. The V-Model approach is serving the automotive industry well and has become one of the most commonly used methodologies. However, the market is in significant transition, with requirements frequently changing and less predictable [5]. Automotive companies need to deliver products within a shorter time scale, and the current development methodology has limited capabilities in solving these challenges. Therefore, it is necessary to make a change in the way of working; it is required an incremental deployment strategy that provides rapid response to changes and allows a continuous software development through iterative cycles [6].

Agile development practices have brought benefits in different domains and can solve the aforementioned challenges in the automotive industry [7]; they offer the required flexibility to handle the fast-changing requirements, provide shorter time-to-market and produce high-quality software [5, 8]. One of the most adopted

[1]Electronic Control Unit (ECU) is an embedded device installed in the vehicle that executes a specific functionality by controlling one or more electronic systems. A modern car can have more than 100 ECUs [18].

practices for the development of complex systems is Continuous Integration (CI) [11], especially when the development process involves multiple development teams. This practice requires developers to integrate their work frequently and enables automated builds and verification activities at every software increment [9].

Automotive software development requires the involvement of multiple hardware and software suppliers. They are responsible for implementing specific components of the final system, and, usually, each supplier has its private continuous integration infrastructure for their internal software development. Consequently, the development process is based on a multi-vendor environment where interaction between different CIs needs to be established, and all the deliveries assembled into the final system. However, the implementation of a continuous integration environment with a distributed development is not easy to achieve, and it may present different challenges [11, 12]. For instance, Debbiche et al. [12] reported challenges related to software requirements, adopted tools and tasks synchronization, as well as problems at the team level, such as the developers' mindset. Moreover, research papers in this topic indicate the need for investigation due to the lack of scientific study in the field of CI associated with the embedded system software development [13].

1.2 Purpose

The transition to Agile in the automotive industry and the next generation vehicles software system require a collaborative and coordinated workflow between OEMs and the involved software suppliers [16]. While in the traditional V-Model development model, the OEMs are responsible for the software verification and validation, with an agile approach, they work concurrently with the suppliers and control the overall software integration process. In fact, the development process requires an interaction between the continuous integration systems of the different software suppliers and the automobile manufacturer [16]. This study investigates the work process of the software development section at an OEM company and the Tier 2 software supplier with the aim of understanding the challenges related to such working environment. This is achieved by collecting data through observations, semi-structured interviews, and by modeling the CI infrastructure of the two companies using the architectural framework Cinders [14]. Moreover, the model of the continuous integration system can be valuable for both companies to have a better understanding of the integration pipeline[2]and a view of the specific stages where the problems are identified.

The reported challenges are subsequently analyzed with the aim of identifying, for each of them, their causes. Determining the root causes and understanding how problems relate to each other can be beneficial for the case companies to determine what contributed to the challenge formation and what necessary improvements can

[2]A pipeline is the set of organized activities composing the CI process

be implemented to avoid the recurrence in the future. In addition, this thesis targets an essential topic as continuous integration, a widely adopted practice, but which lacks research when implemented in automotive software development [13, **?**]. Ergo, the results of this study will add knowledge to the field of automotive software engineering by reporting the challenges of adopting CI for the development of embedded software.

1.3 Research Questions

This thesis addresses the following research questions:

RQ1: What are the challenges of practicing Continuous Integration in a multi-supplier working environment for the development of automotive software?

RQ2: What are the root causes of the challenges of practicing Continuous Integration in a multi-supplier working environment for the development of automotive software?

RQ3: Which solutions can be adopted to address the causes and mitigate the challenges?

The aim of RQ1 is to identify the problems related to the practice of Continuous Integration in a multi-vendor working environment for automotive software development. Once the challenges are described, it follows an analysis in order to understand the root causes of such challenges, this leads to RQ2. Lastly, once the potential causes are known, with RQ3 alternative solutions are provided to overcome the listed challenges.

1.4 Industrial Context

The thesis work was conducted at the development section of a Tier 2 software supplier, a company specialized in providing product engineering solutions to the automotive industry, in collaboration with a world's leading car manufacturer company. For confidentiality, the names of the companies are not given and they are referred to as "Tier 2 supplier" and "OEM company" throughout this paper.

The object of study was the working environment of the two companies for the development of AUTOSAR-based software. The Tier 2 supplier developed the AUTOSAR software platform required by the OEM to integrate the software application built on top of it. Both the case companies used an Agile iterative development approach, had their private continuous integration infrastructure, and worked with different working cycles. After the integration of the software platform and the application, it was generated the ECU software that run on ECU hardware installed in vehicles, as shown in Figure 1.1.

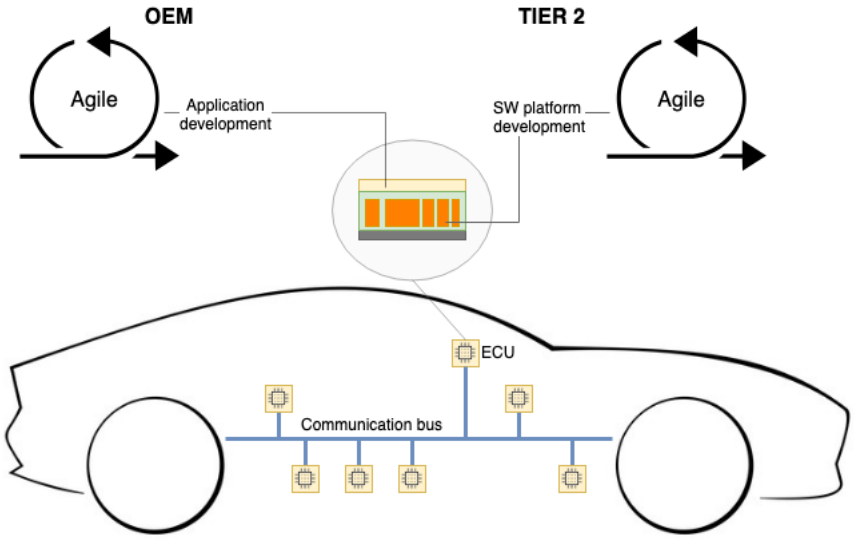

Figure 1.1: Simplified topology of a vehicle network and ECU architecture

1.5 Thesis outline

This thesis is organized as follows. Chapter 2 introduces the terminology and provides the background theory on the subject of the thesis topic, while in Chapter 3, the research approach is described. In Chapter 4, the results are discussed and the research questions answered. Finally, Chapter 5 presents the conclusions.

2

Background

In the following sections are provided the definitions and an explanation of the fundamental concepts. Firstly, Section 2.1 introduces the concept of AUTOSAR-based ECU software. Section 2.2 explores the concept of the traditional software development and the transition to Agile methods. Next, the practice of continuous is discussed. Lastly, in Section 2.5 is presented the framework Cinders and the application of continuous integration modeling.

2.1 Automotive AUTOSAR-Compliant ECU Development

The ECU software development process is organized into phases at the end of which artifacts are delivered [18]. In addition, it is uncommon that a single automotive vendor is responsible for the entire development process; different other companies, such as software and hardware suppliers, are involved in the process. Based on their responsibility, in the automotive domain, it is possible to identify primarily four roles: Original Equipment Manufacturer (OEM), Tier 1, Tier 2 and Tier 3 supplier [18]. The main activities performed by each one of them and the overall development process are presented in Figure 2.1.

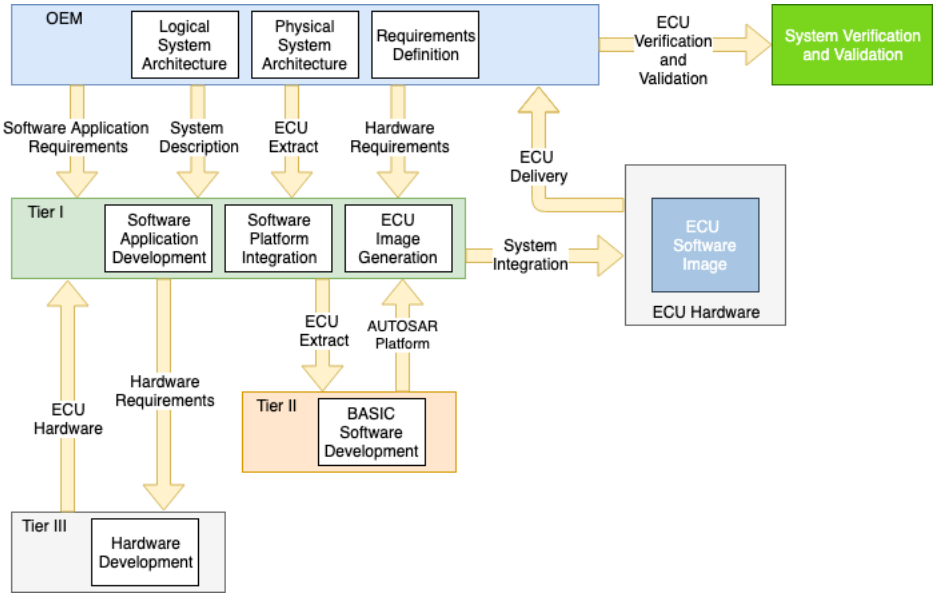

Figure 2.1: The supply chain in the automotive industry

The OEM delivers the system requirements through uses cases and textual requirements [18, 24]. Once the requirements have been collected and clearly defined, they are organized and mapped to features that need to be implemented into the ECUs. In addition, the OEM is also responsible for designing the Logical and the Physical System Architecture. In this regard, AUTOSAR defines the methodology of the overall implementation process and provides design rules.

The logical system architecture includes the design of the various software components and the definition of the communication interfaces needed by them to exchange data and interact with each other. The application software component (SWC) consists of the implementation of the functional software of an ECU, that is, the functionality that an ECU will be performing, such as the electronic brake system, engine control or door locking. The code implementation of the SWC can be done manually using C/C++ programming language or through model-based development (MDD) where the code is auto-generated from models by advanced tools, such as Simulink[3]. Multiple SWCs are subsequently grouped into subsystems and allocated to one or more ECUs [18]. Then, once the logical architecture of the system has been defined, it follows the design of the physical architecture of the system and the definition of the required hardware. The output of these activities generates the configuration of the complete ECU network system, known as the System Description. This artifact is then divided into multiple ECU software models called ECU extracts, and delivered to the Tier 1 suppliers together with the hardware requirements. Every ECU extract contains the description of the single ECU instance in accordance to the AUTOSAR development process [6].

Tier 1 are suppliers that take the responsibility for the physical design of the single or multiple ECUs, and of the implementation of the functional application specified by ECU extract. They are in direct contact with OEMs with a collaboration with Tier 2 and Tier 3 suppliers, and they are generally responsible for the delivery of the final ECU [24]. Tier 2 are suppliers responsible for the development of the AUTOSAR-compliant basic software platform on top of which the application software components run. In fact, the basic software platform does not perform any application functionality itself, and consists of software modules offering services utilized by the software applications, such as communication service. The Tier 1 can assign the development of the basic software platform to a single Tier 2 or involve multiple suppliers and distribute the development various modules. Tier 3 or silicon vendors, are usually companies who take the responsibility of the hardware development and delivery, based on the specifications provided by the Tier 1.

Once the code implementation is completed, the basic software modules are then integrated with the application into a single ECU software image. The final delivery of the process will be the ECU software image running into the ECU hardware, ready for the verification and validation activities conducted by the OEM.

[3]https://se.mathworks.com/products/simulink.html

As aforementioned, the development of automotive software systems is standardized by the reference architecture AUTOSAR (AUTomotive Open System ARchitecture) [18]. The automotive standard provides a set of specifications defining the basic methodology of the software development and architecture of the internal Electronic Control Units (ECUs). The aim of this initiative is to manage the growing complexity of the automotive software, meet the new vehicle requirements in terms of software upgrades, safety, availability and increase the reusability of high-quality and efficient software [19]. The reference architecture has been established in the industry by a development partnership which consists of 10 Core Partners and more than other 150 Partners, including both car manufacturers (OEMs), software and hardware suppliers.

Since the first AUTOSAR-compliant vehicles in 2008, automotive ECUs have predominantly used the Classic AUTOSAR software platform (CP) [21], where software development is based on data and mapped signal received directly from the ECU. In the last decade, the CP has brought to the industry several benefits such as improved quality, software reusability, and better communication among stakeholders [20]. However, new infotainment systems and advanced features, as autonomous driving, have to deal with a continuous large amount of data, requiring high-performance computing capabilities and continuous deployment of over-the-air software updates, and therefore not manageable by the existing platforms [2, 23]. To fulfill these needs and complement the existing Classic platform, AUTOSAR released the Adaptive Platform (AP). In fact, the new software platform introduces the use of superior hardware, as high-performance ECUs, enabling parallel processing and real-time problem solutions [6, 23]. These enable over-the-air updates and cloud connectivity, allowing the deploy of updates to individual software components during all the vehicle life cycle [2, 23].

In addition, in order to achieve customer satisfaction, high quality products and international credibility, automotive industry companies follow ISO/IEC standards [17, 18], such as automotive SPICE[4], for the development processes, ISO 26262[5], for functional safety, SQuaRE[6], for quality requirements and evaluation, and many others.

2.1.1 Traditional development approach and Agile

The traditional development follows the V-Model approach where the development is done incrementally, based on defined phases at the end of the which artifacts are delivered [18, **?**]. As already introduced, the development is not done entirely by a single company; the design of the software components and code implementation are commissioned to external software suppliers, while the OEM takes responsibility for the requirements specification, the design of the system, and the software verification and validation through integration and system testing activities [4]. Moreover, the

[4]http://www.automotivespice.com/
[5]https://www.iso.org/standard/68383.html
[6]https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=484884

verification is also done incrementally; specific tests are conducted for each delivery, starting from the code implementation to the complete system testing, as shown in Figure 2.2.

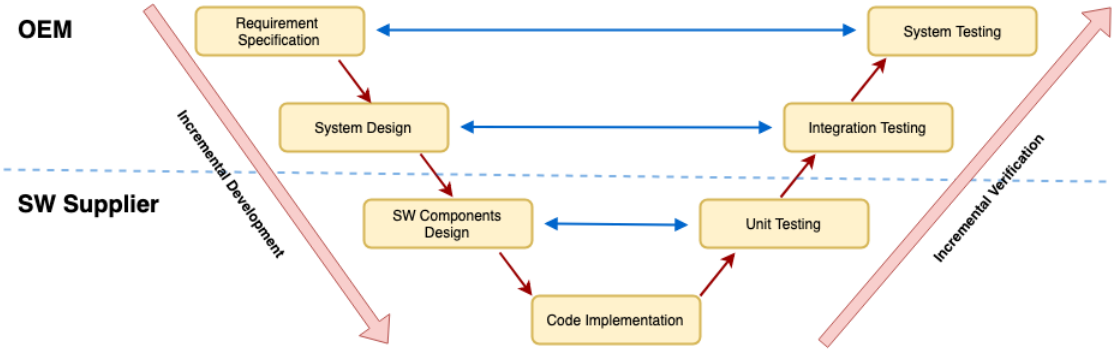

Figure 2.2: V-Model development process

This development approach has been widely adopted and even standardized by a series of automotive standards, such as the ISO 26262, that use the V-Model as a reference development process model [26]. The reason for this is that this kind of approach puts high emphasis on the testing activities; at the end of each development phase, the result is verified and approved before continuing to the next one. As a result, it provides traceability, facilitates planning activities and allows a better product quality assurance [17]. However, these benefits come at a price: the certification of safety-critical system and the sequential way of working slow down the development process and reduce frequency of the software updates [32]. In addition, these drawbacks are becoming impermissible considering the fast-changing market needs, with OEMs trying to shorten development times in order to deliver rapidly new functionalities and obtain quick feedback on decisions [25]. The natural consequence of this is the adoption of fast development processes and the conduction of frequent verification, as obtained when using agile software development methodologies [29, 30].

Agile practices have their foundation on the Agile Manifesto [10]; they base the development on short iterations and incremental software development, thereby improving flexibility, reducing development efforts and allowing at the same time frequent verification and deployment [8]. Starting from the successful transition to agile of small development teams, OEMs and Tier suppliers are currently adopting agile methods at large scale [27, 5, 28]. Figure 2.3 illustrates an agile workflow between OEM and software suppliers; they work with an internal agile environment and are able to integrate their software at the end of each iteration. Moreover, the responsibilities remain unchanged: the OEM determines the requirements specification and the overall system design, while the software suppliers are responsible for the implementation of the code which will run on the ECU. However, the orientation of the software delivery and integration of the ECU software may vary since it usually depends on agreements and the type of contract. Car manufacturers can decide to develop internally the applications, assign the development of the basic software to Tier 2, and at the same time be the of the

system. In other occasions, Tier 1s can be involved in the development process and be responsible for the integration.

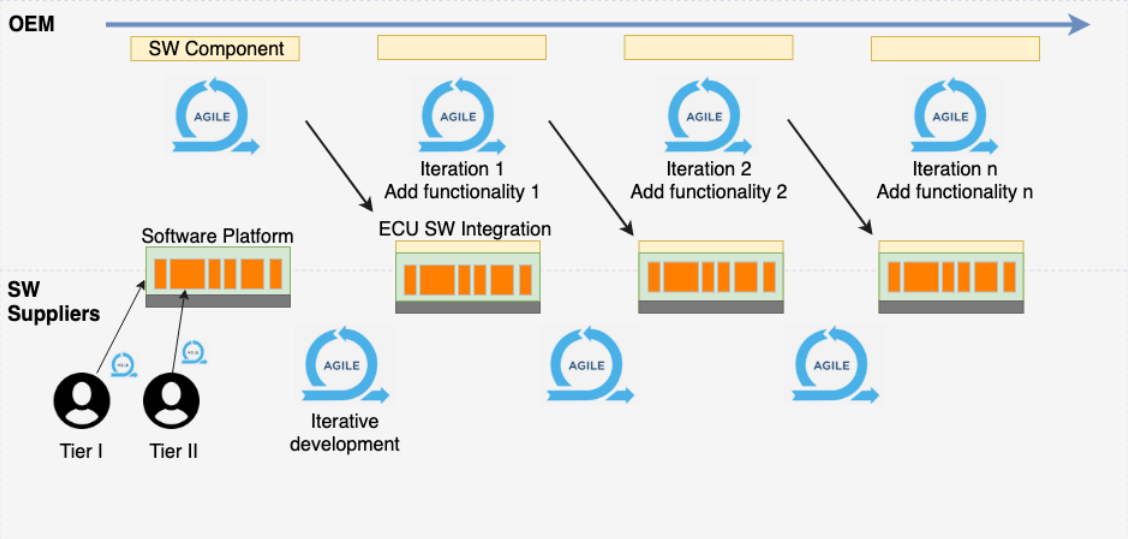

Figure 2.3: Agile workflow between OEM and software suppliers

2.2 Continuous Integration

Agile software development is based on the idea of rapid software delivery and therefore, it requires the adoption of Continuous Integration (CI) [32]. Continuous integration is a practice introduced with the adoption of eXtreme Programming (XP) software development methodology [34]. Essentially, in this development practice developers integrate frequently their work during the day in a central source repository. Moreover, the CI working cycle requires a complete build of the system and execution of unit tests at every code integration in order to verify that the new changes do not break anything [38]. Build and tests should be automated and allow the developers to continue working without the need of waiting for its completion; dead time and productivity drops in this phase are almost entirely avoided [35, 36, 37]. A standard continuous integration flow is shown in Figure 2.4.

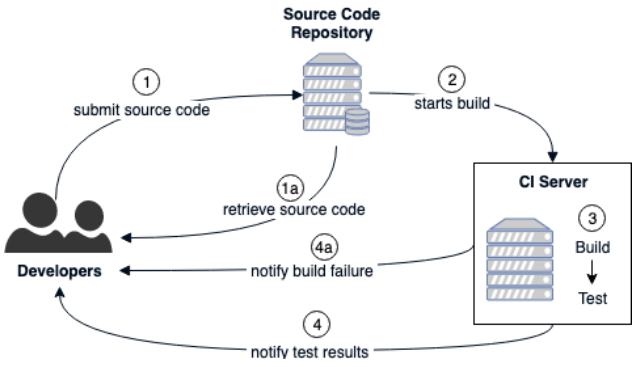

Figure 2.4: Continuous integration workflow [9] [15]

Developers retrieve the latest code version from the source code repository and can add new code starting from that version (1a). Popular versions of source code management tools used in CI pipelines are Git[7], Subversion[8] and AWS CodeCommit[9]. Developers can integrate new source code several times per day into the shared source repository (1). At every code integration, the source code repository verifies the new additions and assures that there are no conflicts with the current version. Next, an automated system build is triggered to detect eventual build issues at each commit and maintain a working version of the system (2). The CI server performs a software build and the configured tasks (3). Continuous integration servers are supported by powerful tools that orchestrate the entire CI process. Popular versions of these are Jenkins[10], Travis CI[11] and TeamCity[12]. The build phase includes the conduction of automated testing activities, which are an essential part of the continuous integration chain. Automated testing ensures that the software meets the initial requirements and executes with the desired performance. In case of a build failure, the developers are immediately notified, and they can take actions to fix the errors (4a). Lastly, in case of a successful system build, a notification with the test results is sent to the developers (4).

The benefits of this practice have been reported and described in other research papers [11, 35, 36, 39]; these include better communication, improved productivity, better software quality and quicker release cycles. However, the transition to Agile and the adoption of CI arise challenges. Bosch et al. [36, 40] argue that these challenges can differ based on the size of the project as well as provide different advantages and disadvantages if continuous integration is adopted at a system or single application level. Moreover, companies tend to report in a different way the disadvantages and faced challenges. This is because CI is not a heterogeneous practice; it incorporates various activities which can be implemented in a multitude of ways [41]. For this reason, Ståhl and Bosch propose Cinders, an architecture framework for describing and better understating the design and implementation of a CI environment from multiple viewpoints [14, 41]. Cinders is also used in this case study for representing the integration flow of the software development of the two companies.

2.3 CI Modeling: Cinders

As aforementioned, modeling continuous integration can provide a better understanding and evaluation of this practice. The architecture framework Cinders was designed to provide tools to model an integration system and analyze it from multiple viewpoints [14]. Other modeling techniques for continuous integration

[7] https://git-scm.com/
[8] https://subversion.apache.org/
[9] https://aws.amazon.com/codecommit/
[10] https://jenkins.io/
[11] https://travis-ci.org/
[12] https://www.jetbrains.com/teamcity/

have already been presented in the past, such as Continuous Integration Visualization Technique (CIViT) and Automated Software Integration Flows (ASIF). However, Sthål and Bosch assert limitations for both the existing frameworks. In their paper [14], they report experiences of how the adoption of CIViT and ASIF does not fully meet the needs of the industry. Therefore, Cinders aims to be an improvement; it combines and extends the valuable elements of both existing frameworks with a more detailed and complete overview of the CI processes and activities [14]. Cinders provides four architectural viewpoints: the causality viewpoint, the product line viewpoint, the test capabilities viewpoint and the instances viewpoints. Every view directs a specific aspect of the continuous integration system.

2.3.1 Causality Viewpoint

The Causality Viewpoint represents the relationships in the CI setup. It includes all the tasks performed by the activity nodes inside the CI setup and the set of events triggering the start of the activity. Additionally, the node relationships are not limited to show the causal relationships, but they also include a short description of their scope and function. This provides a level of detail absent in the other existing frameworks. In figure 2.5, the meta-model of the Causality Viewpoint is represented.

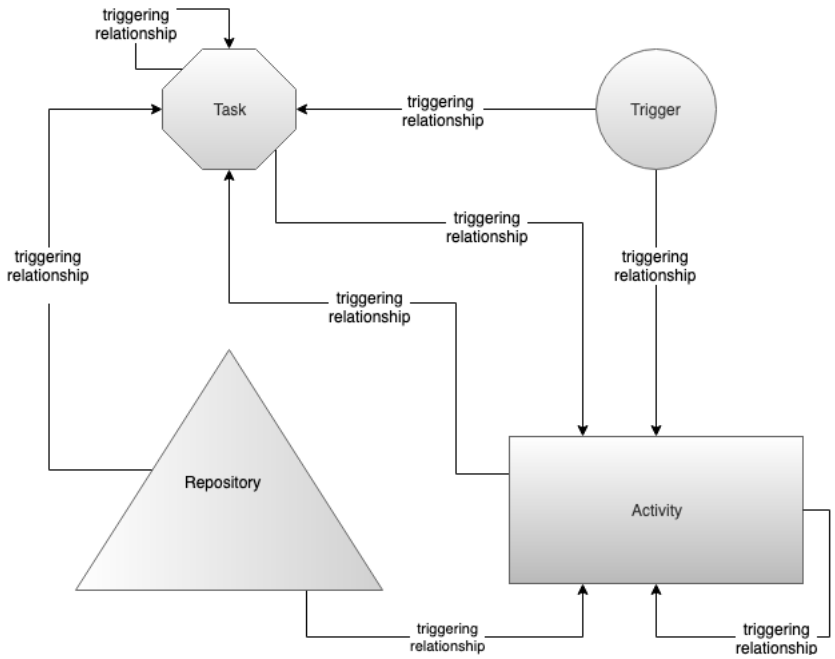

Figure 2.5: Cinders - a meta-model of the Causality Viewpoint

As shown, triggering nodes are circular, activities are rectangular, repositories are triangular and task nodes are octagonal. These are connected by unidirectional edges representing the triggering relationship described with a short annotation.

2.3.2 Product Line Viewpoint

The Product Line Viewpoint contains the same nodes shown in the Causality Viewpoint, but with different relationship types. Triggering relationships are replaced by consuming relationship; these represent how data is transferred between the nodes, and what is the artifacts' flow in the system. The meta-model of the Product Line Viewpoint is shown in figure 2.6.

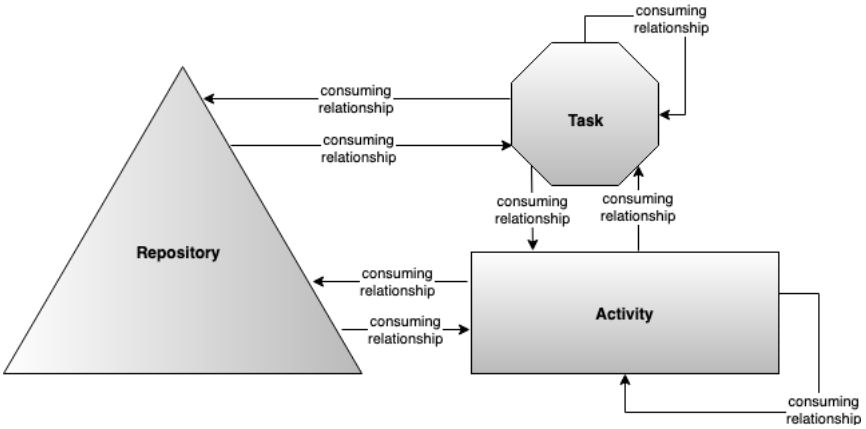

Figure 2.6: Cinders - a meta-model of the Product Line Viewpoint

2.3.3 Test Capabilities Viewpoint

The Test Capabilities Viewpoint provides an overview of the software testing conducted in the system; it provides the results of the test activities and their execution time. In figure 2.7, it is shown the meta-model of the Test Capabilities Viewpoint; the color of the outer borders represents the level of automation, while the color of the solid boxes represents the level of Functional or Non-functional confidence of the specific test activity.

Figure 2.7: Cinders - a meta-model of the Test Capabilities Viewpoint

2.3.4 Instances Viewpoint

This Viewpoint is to be considered optional, and it includes a more detailed view of the nodes and relationship of the system. The Instance Viewpoint includes also the nodes and triggering relationships of the Causality Viewpoint as well as the consuming relationships of the Product Line Viewpoint.

3
Research Approach

This section introduces the approach of this research study. First, the methodology and the data collection are introduced. Secondly, the approaches that are used to analyze the data are explained. Lastly, the threats to validity of the study are presented.

3.1 Research Methodology and Study Design

The objective of this study was to identify the main problems faced by the two companies when practicing continuous integration for a collaborative software development. Therefore, the object of study were the development process and the CI environment adopted by the two companies for real-life project development. In order to achieve the goal, case study was selected as a research method. In empirical software engineering, the conduction of a case study allows the exploration and investigation of a particular phenomenon in its natural context [47]. Moreover, case study is equivalently used for defining fields studies and observational studies, since it consists on the examination of the study object in its ordinary conditions through observations and detailed analysis [42, 44, 45, 47].

Runeson et al. [47] define the of case study in software engineering as method of investigation of a software engineering phenomenon using multiple source of evidence within its natural context. Additionally, for investigational purposes, Runeson and Höst suggest the survey and action research as research methodologies [47]. However, both the alternative methodologies were excluded. Surveys are limited to a standardized data collection through interviews or questionnaires, while the case study focuses on multiple sources of data collection and on the monitoring of the entire process. Therefore, surveys are limited in providing an overview and do not allow a detailed analysis [47]. The adoption of the action research could have been considered a valid alternative for the scope of the work, however, it was still discarded due to the complexity of the process and the impossibility of applying actively changes.

3.1.1 Study Design

The process of conducting a case study includes five main steps: case study design, preparation for data collection, data collection, analysis of data and reporting of the

results. These steps are also followed in this study.

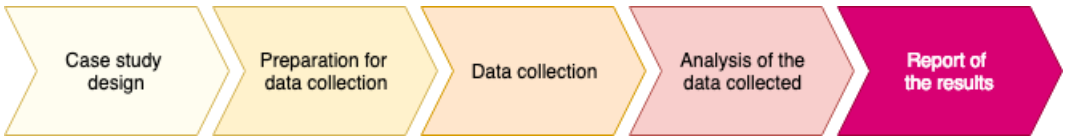

Figure 3.1: Case study steps

In addition, it is possible to classify a case study based on its main purpose. Robson distinguishes the Exploratory, Descriptive, Explanatory and Improving research methodology [45]. This thesis work is presented as an Exploratory research study since it aims to investigate and analyze causalities or particular aspects of a problem.

A successful case study requires a good planning, and there are many elements that need to be preventively prepared. Robson suggests six elements that need to be taken in consideration during the design of a case study: the objective - what is the main goal; the case - what is the object of study; theory - frame of reference; the research questions - areas of concern; methods of data collection and the strategy of data selection [45, 47]. Defining the frame of a reference through the use of theories is not well-established in the field of software engineering [22]. However, for this study the frame of reference was expressed through the conducted literature review and the background of the researcher. In regards to the objective, the case and the research questions have already been covered in the previous section, while the data selection and the selection strategy are presented in Table 3.2. Each research question was addressed with a specific approach and the respective contributions to the findings are presented.

Research question	**Methods of data collection**
What are the challenges of practicing Continuous Integration in a distributed environment for the development of automotive software based on the AUTOSAR Standard?	Conduction of non-participant observations, analysis of internal documentations. Interviews with the engineers and modeling of the CI infrastructure using Cinders.
What are the main causes related to the challenges for practicing Continuous Integration in a distributed environment for the development of automotive software based on the AUTOSAR Standard?	Analysis of the collected qualitative data and RCA using Fishbone diagrams in combination with a Pareto Analysis.
Which solutions can be adopted to address the causes and mitigate the challenges?	Brainstorming workshop with the development teams

Table 3.1: Research questions and the methods of data collection

RQ1 aims to investigate and find out which are the problems related to the practice of CI from the two software suppliers. In order to achieve this, the conduction of observations allowed to have an understanding of the CI chain and the development workflow followed by the development teams. Among other benefits, modeling can provide better understanding of a software system or architecture [49]. Therefore, during the data collection, a model of the continuous integration workflow was done using the available tool Cinders [50]. At the same time, interviews provided a secondary source of data; the conduction of interviews allowed the extrapolation of issues not transpiring during the observations.

Answering RQ1 contributed with a list of challenges and issues in various stages of the development process. Once all the problems were listed, RQ2 was addressed and a root-cause analysis (RCA) was performed in order to identify the main cause for each of the reported problem. Fishbone diagrams are one of the most efficient tools and techniques used for identifying the root cause of a specific problems [51]. These diagrams were used in combination with a Pareto analysis to show the potential causes and to which extent each cause contributed to the problem. The produced diagrams were presented and evaluated with the members of the teams. Subsequently, the last research questions had the objective of finding possible solutions for the reported causes. RQ3 was approached with a brainstorming workshop where possible solutions were identified for the causes. In addition, the workshop session served also as a validation of the results. Lastly, in order to verify the outcomes of the results, a development part consisting on a proof of concept of the workflow was developed. The scope of the it was to replicate the CI workflow and part of the software development process and evaluate and validate the results and proposed solutions. The case study design flow can be observed in the figure 3.2.

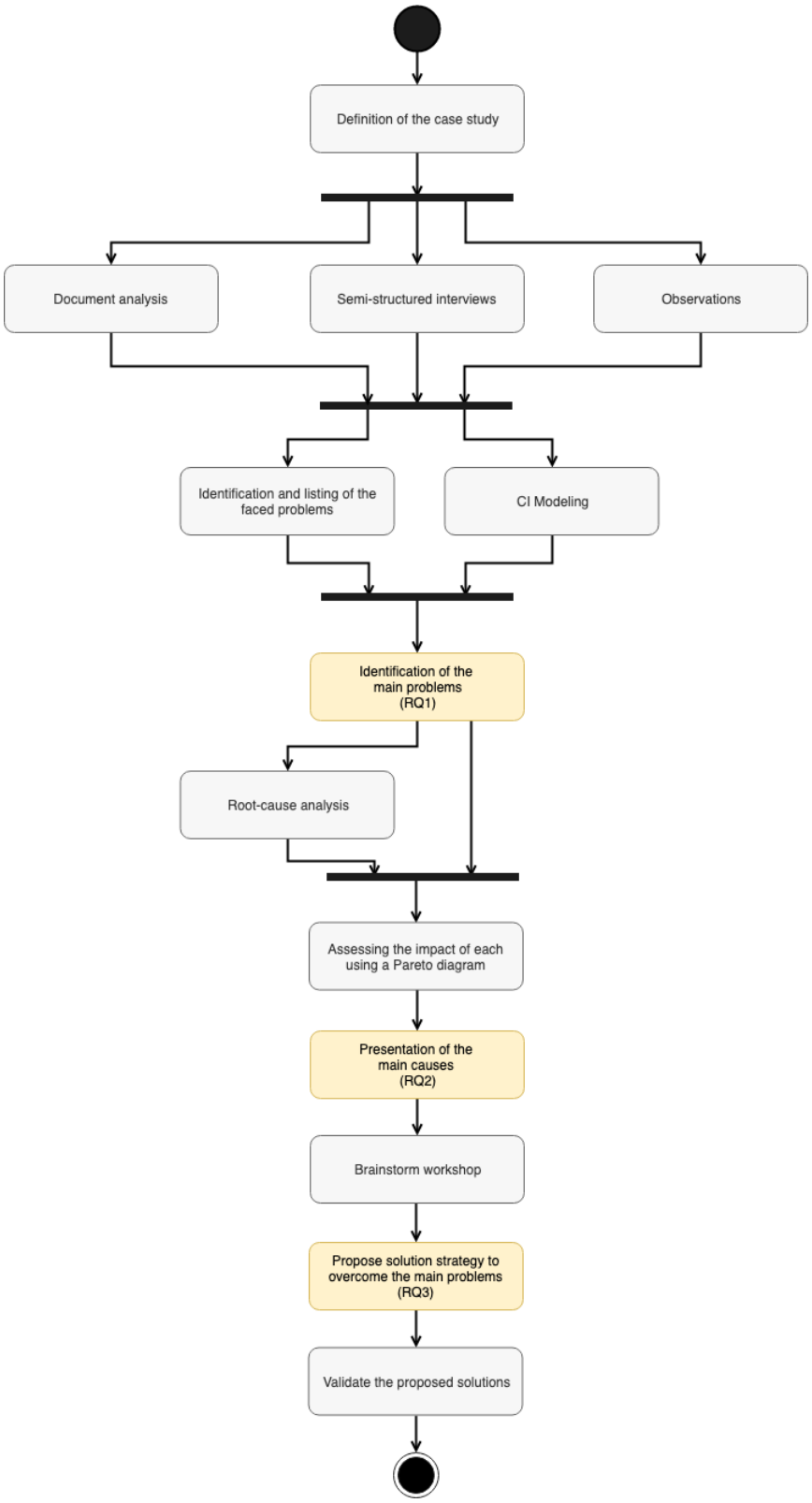

Figure 3.2: Study design flow

3.2 Preparation for data collection

The preparation for data collection varies on the complexity of the research; Runeson et al. [47] defines this phase as the set of activities necessary as a preparation for the data collection phase, but also for the next case study stages. These include training on how to approach the research instruments, the respect of ethical guidelines and the arrangements with the companies and individuals that will be part of the study [47].

Before approaching the study, the researcher attended a course in empirical software engineering, participated in a seminar on research methods as well as followed a case study protocol specific for the field of software engineering. In addition, the two industrial managers managed the agreements and arrangements with the companies.

3.3 Data Collection

There are different ways of collecting data, and the choice of the appropriate methods is done in relation to the expectation of data available [47]. Runeson et al. [47] discuss six main sources of evidence, frequently used in case studies: documentation, archival records, interviews, direct observations, participant-observations and physical artifacts. In addition, Verner et al. [48] suggest adopting multiple collection methods in order to have diversified sources of evidence. In this way, a bigger amount of data can be collected, but also the drawn conclusions are stronger if they are based on multiple sources [47].

Lethbridge et al [43] present three degrees for categorizing the procedure of data collection. The first degree type are data collected directly with the subjects (e.g. interviewee) through interaction in real time. In the second degree are categorized data gathered directly from the source but without direct contact (e.g. video recordings). In the last degree are included data collected through the analysis of documentation of other types of artifacts. The first and second degree methods can be used as a primary source of data; they provide the better quality of data and allow the researcher to have exact control of what type of data is collected [47]. However, they require high effort for both the researcher and from the case companies. The third degree type provides a lower quality and the researcher does not have control of the data; they can be used as supportive methods and provide completeness [47].

In this particular study, direct observations and semi-structured interviews were used as a primary source of data in triangulation with the available documentation provided by the companies, and supported by a literature review on the topic. This strategy would allow to have multiple sources and at the same time cover the three levels of evidence introduced by Lethbridge et al [43]. The interview questions are listed in Appendix 1.

3.3.1 Document analysis

Documentation analysis consists of an independent examination of work artifacts already available and that was collected for purposes not related to the research study [47]. In this study, document analysis was used as the first data collection method and subsequently combined with the observations and interviews. After requesting specific documentation on the organizational structure of the teams and on the development process, the researcher had access to the internal documentation wiki of the Tier 2 supplier, where various guides, technical documentation and process documentation were stored. Based on their importance and utility to the study, they were graded and saved in a bookmarks folder for easier access and analysis.

3.3.2 Interviews

Interviews consist of a conversation where the researcher asks a set of questions to the subject about the topic of interest of the case study. The advantage of offering interaction directly with the respondent, allows the researcher to control the data collected by formulating questions on the specific area of study [47]. This results with a more in-depth investigation, better understanding and higher quality of data. According to Robson [45], interviews can be unstructured, semi-structured and fully-structured. As aforementioned, in this study semi-structured interviews were used. In this type of interviews, the researcher plans the interview and prepares a list of questions that will be asked during the conversation. However, the researcher decides during the interview in which order to ask the questions, and, eventually, this type of interviews allow improvisation and the inclusion of questions that may arise at the moment to examine specific points [47].

The interviews were held with the employees of both companies with the aim of understanding the development process and the adopted continuous integration environment, investigating the difficulties and barriers perceived at the team or project level, and find out which can be the causes based on the view of the interviewee. The interview sessions had a duration of approximately 1 hour, were conducted in English, and they were divided into three phases. In the first phase, of the duration of 5-10 minutes, the researcher gave some personal background and presented the objective of the case study. Next, after a short discussion on the subject, the main interview questions were asked; this phase had a duration of 30-40 minutes. In the last phase, it followed a summary of the discussion and it was collected eventual feedback from the interviewee. The majority of the interviews were recorded in order to ensure that all the details were collected, especially the ones that were not possible to include in the notes taken in real time. Eleven individual interviews were performed with engineers of both companies. The interviewees had experience in automotive engineering, covering roles such as technical expert, software architect, project manager, verification and quality assurance manager, system developer and system tester, as shown in Table 3.2.

Company	Job title	Work Experience
Tier 2	Subject matter expert	> 10 years
Tier 2	Software architect	> 10 years
Tier 2	Principal Architect	2 - 5 years
Tier 2	Project manager	> 10 years
Tier 2	Product owner	5 - 10 years
Tier 2	Developer / Tester	5 - 10 years
OEM	Product owner	> 10 years
OEM	BaseTech verification manager	5 - 10 years
OEM	System architect	5 - 10 years
OEM	System tester	2 - 5 years
OEM	Developer	2 - 5 years

Table 3.2: Interviewee's profile

3.3.3 Observations

Another source of data are observations. Throughout observations, it is possible for the researcher to investigate a phenomenon by being in direct contact and collect data in real time [47]. Runeson and Höst divide observations in four categories based on the interaction level of the researcher and the awareness of the subjects of being observed [47].

In this study, there was no interaction between the researcher and the observed subjects; they were aware of the conduction of the study and the presence of the researcher, but he was always considered external and seen only as a researcher. Initially, the observations helped to understand the development process, how teams interacted and served as input for the formulation of the interview questions. Subsequently, during and after conducting the interviews, the observations had a focus on the emerged issues and activities that were missed during the first phase. In this way, it was possible to triangulate the data obtained from different sources and at the same time have evidence of what was reported during the interviews and real situations.

The observations were conducted on small groups, from 7 to 10 people, and they were spread over a period of four months. The observed meetings were in major part stand-up meetings and had a duration of 20-30 minutes; they were hold daily by the teams, but attended on five occasions and on irregular basis by the researcher. In addition, a sprint planning meeting and a sprint retrospective meeting were also attended in order to cover a full iteration of the development cycle. During the observations, notes were taken and they were afterwards revisited directly after the meetings.

3.4 Data Analysis

The aim of data analysis is to derive conclusions from the collected data. However, the data analysis should not be conducted as a sequential phase after the data collection process. The analysis may bring out unexpected information that was not previously considered and that needs to be further investigated [22]. Therefore, in this study the analysis process was done in parallel while conducting the interviews and during the observations. In fact, some of the interview questions were updated, other questions needed to be added and specific internal documentation availability was requested.

In this study, the data extraction and the report of findings related to the challenges were done using qualitative coding and Robson's guidelines [45]. The collected data was coded based on the problems faced when adopting or practicing continuous integration. For instance, the following quote was coded using the codes: "many target variants", "problem", "automation flow difficulties", "manual configuration required".

> *"We have 44 variants to test. The problem is not only related to the difficulty of having an automated flow, but also to the manual configuration required every time for each variant."*

Once the data was coded, it was connected to the researcher comments in order to provide hypothesis of preliminary results. The aim of the researcher is to provide systematic evidence [47]; the generated hypothesis need to be supported by the successive data collection phase. Therefore, this process is a continuous iterative process and, as aforementioned, with data collection and analysis done in parallel. Lastly, the generation of a set of hypothesis allows the formulation and the report of the findings necessary to answer RQ1, that is, the list of the challenges when practicing CI. The main five steps followed during the qualitative analysis are shown in figure 3.3.

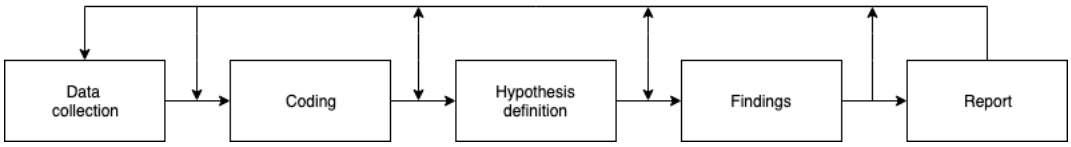

Figure 3.3: Robson's five steps of data analysis [47]

After reporting the problems, it was necessary to determine their possible main causes and identify their cause-effect relationship. This was achieved by conducing a Cause-and-Effect analysis using fishbone diagrams. The construction of the cause effect diagram was done using three steps. Firstly, the problems previously defined were classified by seriousness and prioritized; they were considered the starting point of the fishbone diagram. Next, main categories to group the causes were defined. Lastly, the potential causes were drawn for each of the main groups, and the final diagram showed of the causes correlated the problem.

3.5 Threats to Validity

The validity of a study determines the reliability and accuracy of the results, therefore, it important to discuss the threats that might compromise them [47]. In this section the threats to the internal validity, external validity and construct validity are analyzed.

3.5.1 Construct Validity

Construct validity refers to the degree to which operational measures in the study reflect the initial intent of the researcher and the addressed research questions [47]. In this study, the construct validity relies on multiple method and data source triangulation [47]; the data was collected from multiple independent sources, over a continuous period of time, using multiple methods and under different circumstances.

The triangulation method involved the combination of data collected through observations, interviews and document analysis. Observations were conducted for a period of 4 months, and covered different stages of the development cycles. The results of these were then compared with the data collected through the semi-structured interviews and the documentation available at both companies.
The source triangulation was used by collecting data through different set of participants or documentation provided by different companies. In this study teams belonging to two different companies were involved, and the interviews had subjects various software engineers occupying different roles.
Moreover, the study of the researcher was continuously monitored by three supervisors; the lack of professional work experience of the researcher in the automotive domain was compensated by the supervisors and managers from the industrial partners. They refined the interviews questions and participated to some of the interviews to ensure that there was no misinterpretation.

3.5.2 Internal Validity

This study covers the analysis of casual relationships, therefore, there is a risk of the presence of internal validity threats that might compromise the results [46]. The researcher investigated the causality factors that generate a particular problem, and there is the risk that the reported factors might be affected by eventual third elements that did not transpire during the study. In order to minimize the threats to internal validity, the researcher reevaluated the questions after each interview, covering new emerged explanatory variables.

3.5.3 External Validity

External validity refers to the degree of generalization of the findings and the utility of the industry or other individuals external of the study [47]. This study is based on a case study and has as case object the development process of a

single project involving two companies. This represents only a small part of the industry population, therefore, there is a risk of the presence of external validity threats that might limit the degree of generalization of the study. However, it is important to acknowledge that even though the study involves two single companies, the development process is based on the outline and guidelines provided by the AUTOSAR consortium, which are at the same time adopted by the whole industry.

3.5.4 Reliability

Reliability refers to the degree of dependence between the data analysis and the researcher [47]. In other words, if other researchers analyses the same data, they should draw the same conclusions.

In order to ensure reliability, the industrial managers supervising the study participated during the interviews as observers and clarified the eventual unclear questions. In addition, the reliability was further improved by the evaluation of the findings with some of the participants and the industrial managers.

4
Results

4.0.1 The ECU software development workflow

ECU software development at the case companies is primarily based on the AUTOSAR standard, therefore their methodology presents similarities in the type of activities and how they are organized. This is due to the fact that the AUTOSAR methodology requires also standardized use of artifacts such as the ECU extract and the application manifest.

The development team at the OEM company is responsible for the design and implementation of application software component. The fist step consists of the application design; this is done independently of the software platform and includes the implementation of the functionality that needs to be executed. Part of the code implementation is auto-generated by the modeling software tools while other are parts are coded manually by the developers using C++. In addition to the source code and the application design, the developers generate also the application manifest. This artifact provides the modules configuration required by the application to run onto the software platform; application deployment and execution specifics are included together with other properties such as the network configuration and access roles. The produced files are stored into a source code repository. Generally, each software component is synchronized with a dedicated GitLab repository which stores all the generated files at the latest version. The various repositories are connected to the CI automation server Jenkins which is responsible for the build and the tests execution. The continuous integration system is modeled using Cinders and the Causality Viewpoint is presented in Figure 4.1.

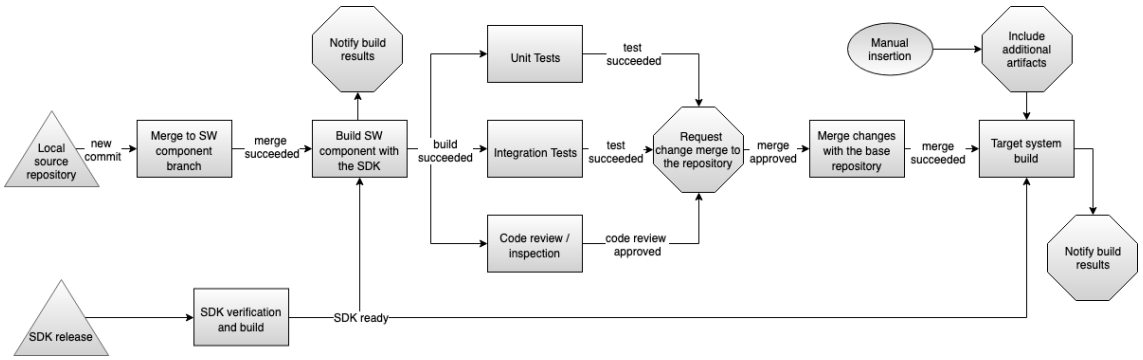

Figure 4.1: Causality Viewpoint of the CI flow for the software component

The Causality Viewpoint shows the integration process of a software component

from the code submission to the software build. The code change submitted by a developer - procedure called commit - needs to be built, verified and approved by other developers before being merged with the existing base version. Every developer works on an independent line of development - called branch - where every local change needs to be stored. Once the developer submits the changes, the software component is built using the latest verified version of target SDK, and the build results are thereafter notified. In case of a successful build, unit tests are performed in order to verify the correct execution of the software. The changes that are successful and pass all the test activities are ready to be merged with the base repository. The base version of the software component is built, and in this case the development team is notified of the build results. The generated binaries are ready to be integrated into a single system with the software platform developed by the software supplier.

The Tier 2 supplier is responsible for the development of the various functional clusters or modules composing the software platform. Multiple development teams work on single software module and every module is stored in a separate GitLab repository. The build process is very similar, however, differently from the software component, the various modules needs to be assembled before the build and result in a single executable. The case company uses Yocto Project [53] as a build framework and the build is done using CMake and based on various recipes. CMake is a build tool that uses code scripts - called Makefiles - to generate executables, while recipes are text files containing a set of instructions, such as dependencies definitions, required to perform the build. The development teams at the Tier 2 supplier use GitLab also as a CI automation server. The Causality Viewpoint of the CI flow for the software platform is presented in Figure 4.2.

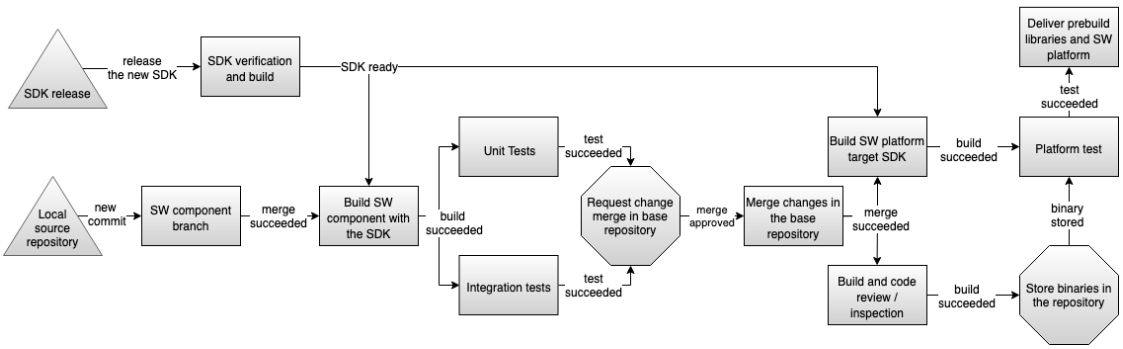

Figure 4.2: Causality Viewpoint of the CI flow for the software platform

Once both the component and platform are binaries are generated for the software component and the application software, they need to be integrated into a single ECU image. Depending from the ECU and the type of the contract, the integration responsibility can be assumed by the OEM or by a Tier 1 supplier.

4.0.2 Identified challenges and their causes

As described in section 3.1.1, the conduction of non-participant observations and interviews had the aim of understanding the development process and identifying the challenges related to the adoption of the continuous integration. The identified challenges were classified into five categories: Build Design, Integration Process, Testing, System Design and Organizational Structure.

Id	Category	Challenge
CH1	Build Design	Complicated build system and process
CH2	Integration Process	Broken builds
CH3	Integration Process	Interruption of the development flow
CH4	Integration Process	Arduous system integration process
CH5	Testing	Lack of automation
CH6	Testing	Time consuming testing
CH7	Testing	Late defect discovery
CH8	Organizational Structure	Delayed response time

Table 4.1: Identified problems in the CI flow with the current development process

4.0.2.1 Complicated build system and process (CH1)

The main reported problem associated with the build system was relative to the high complexity of the build setup and the overall build process. At every software update, a full system build is necessary for the required hardware targets, and consequently, the ECU software images need to be generated. The integration team mentioned in the interviews that setting up a single development environment for a cross-development toolchain was not easy to achieve and the final result is a complex build process composed of several steps. They reported that it requires time and expertise to write recipes and configuration files, avoid common issues such as missing dependencies, and achieve the successful build. They provide internal auto-configuration scripts and have automated all the build steps; developers do not have to deal with the complexity of the overall system and can focus only on the single software component. However, the challenge of the complex system build remains and it needs to be maintained and it is constantly updated. The causes of Challenge 1 are shown in figure 4.3.

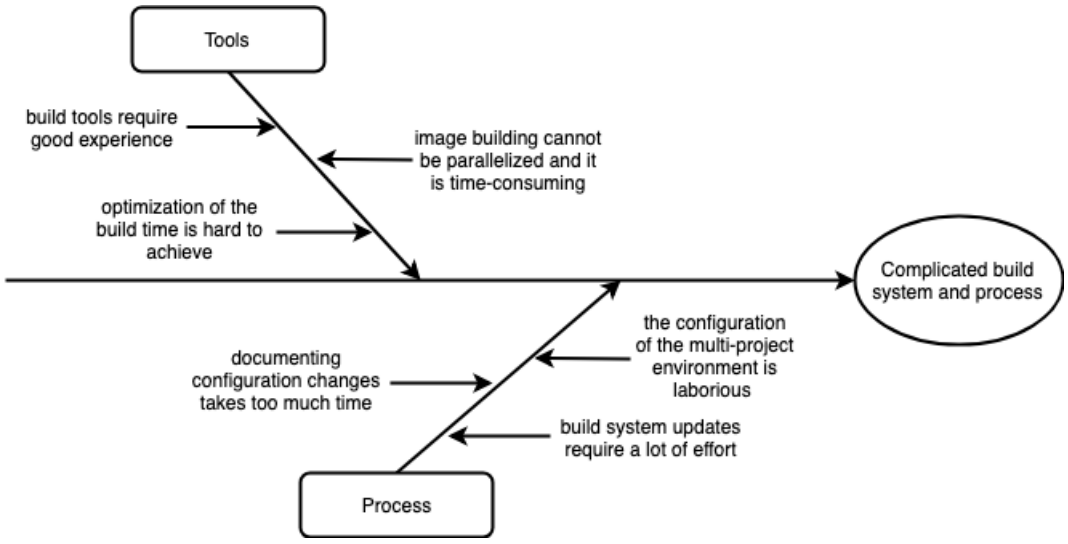

Figure 4.3: The fishbone diagram showing the causes of Challenge 1

Build tools require good experience. One of the causes reported during the observations and emerged during the interviews was the complexity of the tools used. In particular, developers reported difficulties related to the use of Yocto; they often make mistakes when configuring and writing tool specifications, even after having used it for months. Some of the build instructions are edited manually and require meticulous specification of the components that need to be included in the build, such as package dependencies.

> *"[...] Inexperienced developers make usually mistakes. [...] they struggle with editing recipes [...] sometimes they include the wrong dependencies"*

Optimization of the build time is hard to achieve. The build performance is dependent on many variables and does not only depend on hardware capabilities. Caching size limits and other specific build characteristics are big obstacles that need to be considered. Software components are highly dependent on each other and a code change in one of them may require to rebuild of the whole module.

> *"Using virtual machines, the first build takes 3 hours. [...] The use of cache reduces it, and it can take from 2 hours to 20 minutes. [...] Sometimes a small change requires a fresh build of the whole module and it is not possible to avoid this."*

Image building cannot be parallelized and it is time consuming. The generation of an ECU images cannot be parallelized and the software packages need to be installed in a sequential way. The interviewed developer mentioned that it is possible to parallelize the build of multiple images, but the waiting time corresponds to the build time of the biggest ECU image. However, this requires a dedicated repository for each image.

The configuration of a multi-project environment is laborious. The integration

team reported that another element increasing complexity of the build process is the development and maintenance of multiple projects. They deliver their software platform to multiple Tier 1 / OEMs and several software components are shared. Each project has its own repository; they need to synchronize them with the ongoing development state and it takes time and effort.

> *"We have multiple projects for different customers. [...] We create a dedicated repository, [...] some are shared components, some are not. The shared components are stored in a dedicated development repository."*

Documenting configuration changes takes too much time. Maintaining multiple documentation requires a lot of resources. Build systems relies on the specific version of tools and of the internal software development kit (SDK); they should always be in-sync and every alteration should be stored.

> *"We have documented each stage of the internal process flow and we maintain an updated internal wiki. [...] It takes resources, but most of the issues can be resolved just by consulting it."*

Build system updates require a lot of effort. It takes a significant amount of effort to update configurations and tool versions. Interviewed members of the team mentioned that updates of the SDK are one of the most time consuming procedures. It is not avoidable to change version without the need to do consequent adjustments and have a working build system.

> *"We follow the AUTOSAR standard and the consortium releases twice a year. [...] They just upgraded to Yocto Project version 2.4, Rocko, and we are doing the same. [...] We have to update our SDKs to that version."*

4.0.2.2 Broken builds (CH2)

Developers reported experiencing broken builds on a fortnightly basis and it takes effort to fix them. However, the work of the developers is constantly monitored in order to reduce the number of broken build occasions. Also, in order to minimize the build failures, multiple integration tests are at multiple levels, i.e. component, module, system level. However, there were identified build failure problems related to the tools usage, software dependencies and verification coverage. The causes of Challenge 2 are shown in figure 4.5.

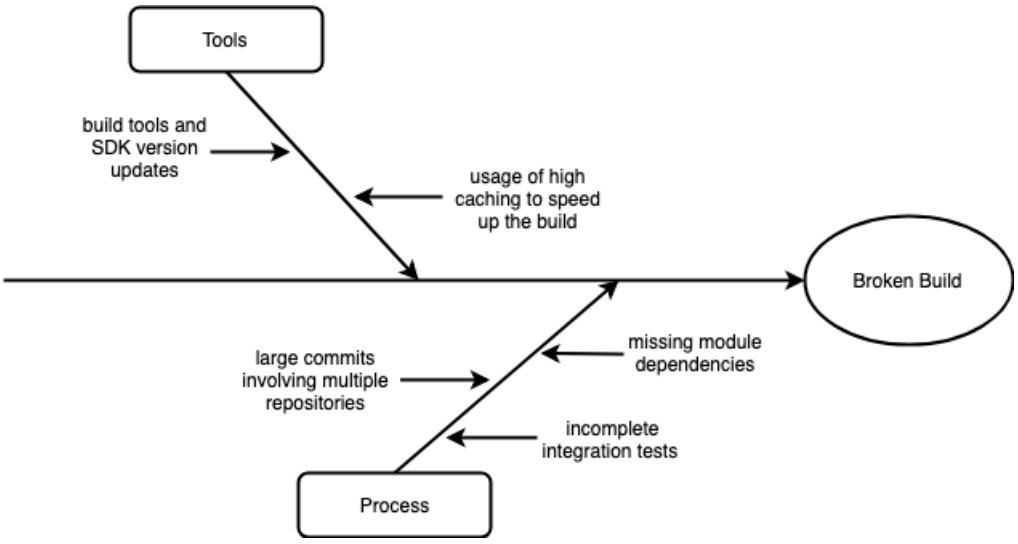

Figure 4.4: The fishbone diagram showing the causes of Challenge 2

Build tools and SDK version updates. New versions for the build tools are released every six months and it is necessary approximately one month for the case companies to have the updated build environment. During this period, the team mentioned to have frequent build failures due to changes in the build instructions.

High usage of caching to speed the build. The integration teams mentioned that caching is a very important element to be considered in order to have an optimized build system. Large volume of shared cache allows to reduce build time since it is not necessary to rebuild packages or modules when changes have not been introduced for them. In fact, the build time can decrease from 2 hours to 5-10 minutes if only minor changes have been made. On the other hand, cache has also been reported to be the cause of multiple build failures.

> *"The use of cache reduces it (the build time), and it can take from 20 minutes (in presence of cached files) to 2 hours (in case of fresh build). [...] Sometimes happens changes are not detected and it fails [...] it is necessary to clean the build directory or delete the tmp folder."*

Large commits involving multiple repositories. As aforementioned, developers' work is constantly monitored and they have specific deadlines to meet and deliver their code. In the interviews, managers explained that they always try to push for regular, small commits. However, this is not always possible to achieve and some engineers opt for developing a complete functionality before checking-in their code. Similar commits involve changes in multiple repositories, and it was reported that is common for the first build to fail due to issues such as merge conflicts. Developers have to put more effort and solve the problem.

Missing module dependencies. The issue with missing dependencies has been mentioned multiple times during the interviews and reported as a cause of build failures. In specific, engineers said that inexperienced developers forget to add

dependencies when writing the recipes; missing packages or not executed tasks generate immediately a broken build. In addition, dependency issues have been reported also in case of incremental builds. The interdependency between software components often requires a complete build and not just an incremental build since a software component may rely on specific versions of other components.

4.0.2.3 Interruption of the development flow (CH3)

Issues in the CI workflow frequently generate an interruption of the development flow. This has been reported both at the team and at the project level. In fact engineers at the OEM company stated that the development of specific components can be interrupted for days, even weeks, before getting a software delivery from the suppliers. The causes of this challenge are shown in figure 4.5.

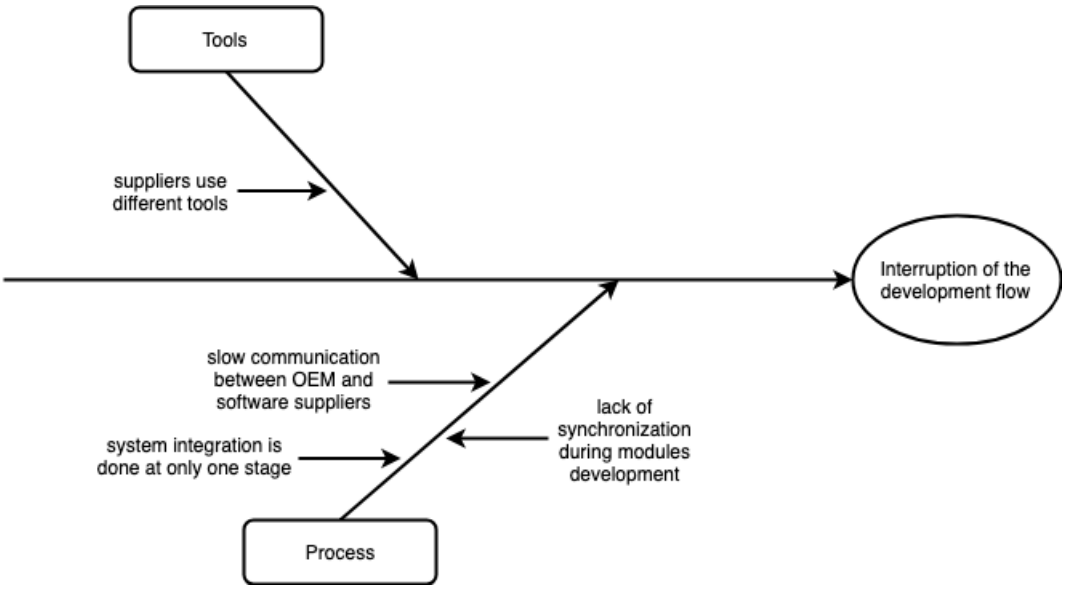

Figure 4.5: The fishbone diagram showing the causes of Challenge 3

Suppliers use different tools. During the interviews it was mentioned that there is a difference in the tools used by the OEMs and the other suppliers. Some artifacts - such as ECU extracts - are delivered by OEMs or exchanged between software suppliers; they need to be processed and converted in order to be accessed and used by other tools. Interviewees reported that a change in the deliveries interrupts the workflow since the software responsible for the conversion needs to be updated and be able to process the file with the new changes.

> *"[...] we use [in-house developed tool] to read the ECU configuration files [...] the file is parsed in order to be read by our editor"*

4.0.2.4 Difficult system integration (CH4)

The involvement of multiple parties in the development process requires the integration of all the software produced by each supplier. The responsibility of the final software integration depends on the contract and the type of project. Generally OEMs usually take this responsibility for the development of some ECUs, while for others they rely on Tier 1 suppliers. However, the software integration is not always easy; engineers at the OEM company reported faults due to inadequate interfaces, missing implementations due to bad synchronization with the development, and performance issues. The causes of CH4 are visible in figure 4.6.

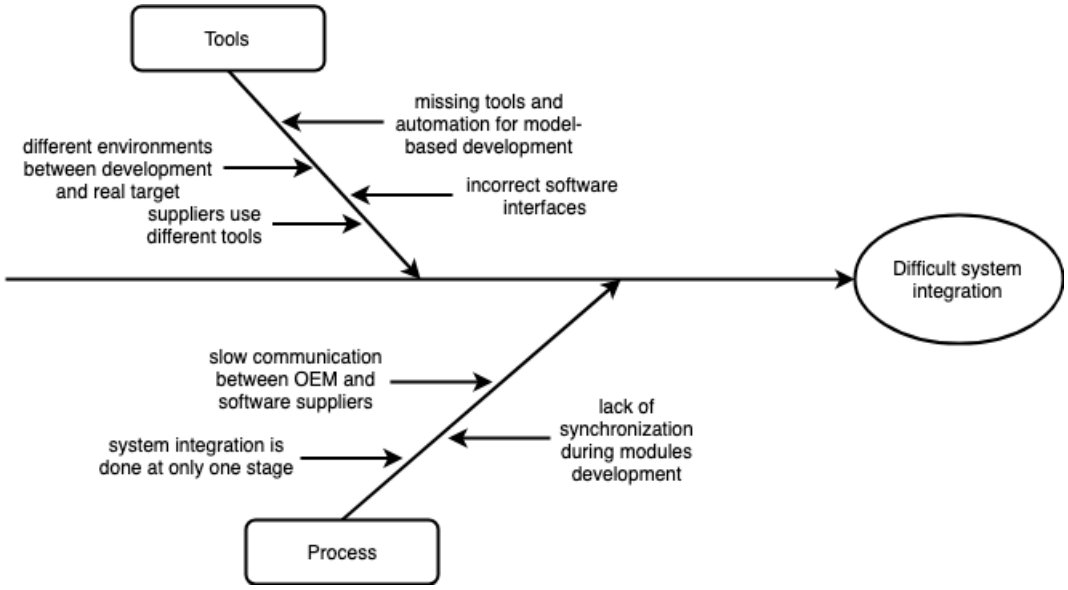

Figure 4.6: The fishbone diagram showing the causes of Challenge 4

Missing tools and automation for model-based development. The growing complexity of the systems has increased the use of physical modeling and meta-modeling environment for designing the system architecture. This has brought benefits for handling major challenges, however, during the interviews it was also mentioned that this has brought also issues. The lack of available tools for testing and having an automated integration process when using model-based development represent a problem during the system integration process. In fact, it is not possible to have an automated continuous integration pipeline if, during the system integration process, faults need to be fixed and separately verified.

> *"[...] there are not many tools to test generated code when using DSL-modeling [...]"*

Incorrect software interfaces. Another cause which transpired during the interviews was the interface incompatibility, especially when multiple software suppliers are involved in the development of the software platform. The integration of the software modules and software components often presents problems of inadequate software interfaces. Software suppliers focus on the verification of the internal interfaces and communication between components, however, the external

interfaces and the intercommunication between the various software components were reported to generate verification fails. Similar issues were reported by the supplier; they mentioned that slow communication between suppliers cannot handle the fast changing requirements and specifications are not always aligned with the development state.

> *"[...] the development includes several software suppliers and it is important to consider the interface problems [...]"*

> *"Yes, the AUTOSAR standard helps when it comes to some modules, but other software components change and everything should be aligned."*

Different environment between development and real target. The development environment is mostly based on the use of emulators since they allow the developers to produce new code and verify instantly in the same machine. During the observations, there were noticed issues when this environment did not precisely coincide with the compiling and deploying environment. This resulted with time-consuming activities necessary to coordinate the two environments.

4.0.2.5 Lack of automation (CH5)

A successful CI environment should aim to achieve a fully automated development process, during the code implementation, system building, testing, and finally provide an automated deployment. Therefore, the lack of automation can be considered one of the major problems currently faced. In fact, both the case companies reported the importance of overcoming it. The causes are shown in figure 4.7.

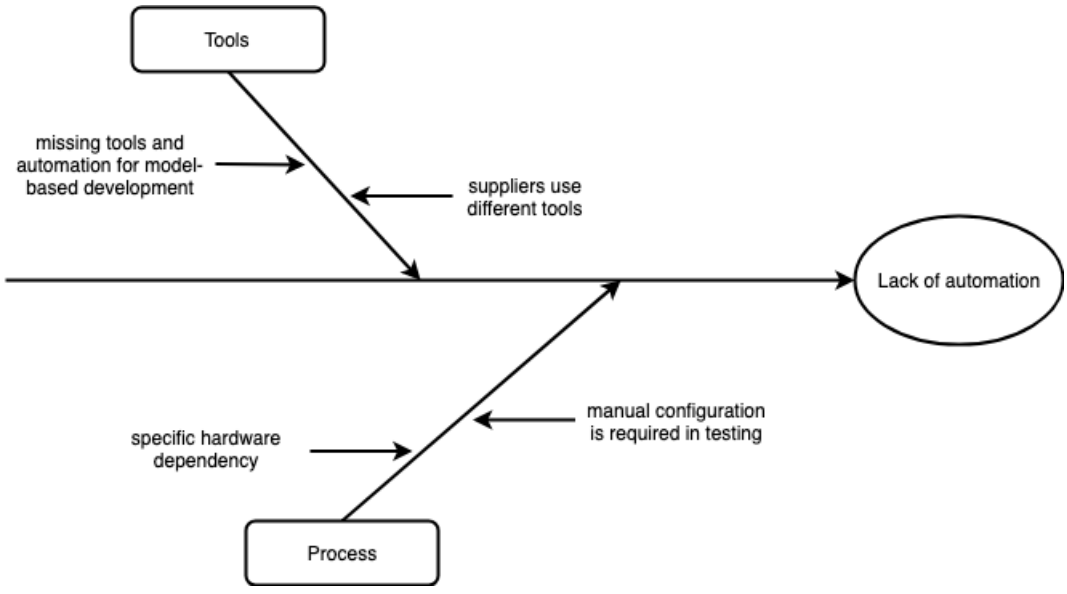

Figure 4.7: The fishbone diagram showing the causes of Challenge 5

Manual configuration is required in testing. The last phases consist of the verification and validation activities which require manual configuration from testers. In specific, testing is done at the system level and requires several ECU nodes interconnected and specific hardware configurations. The test bench is not currently possible to automate, and the inclusion of manual testing activities does not allow a fully automated workflow.

> *"We have 47 possible combinations that need to be tested. [...] I manually configure the boxcar based on the testing activity I have to do. [...] We are working on this, but for the moment it is not possible to have it automated."*

Specific hardware dependency. During testing phases, the software needs to be run on ECU hardware in order to verify that it completely satisfies the specifications. At the case companies, the development and part of the testing are done using system emulators (QEMU), however, such activities need also to be run on real hardware ECU devices. The deployment of the software is not possible for all the target devices, and this results in another cause of not having a complete automation.

> *"We have currently full automation on 4 hardware targets but we are trying to target all the inolved ECU hardware"*

4.0.2.6 Time consuming testing (CH6)

Another challenge identified during the conduction of the observations, and reported also during the interviews, was the large amount of time required by the testing activities. Certainly, this is also dependent to the fact that an extensive amount of testing is required for safety critical systems, however, other also causes are involved and they are shown in the fishbone diagram in figure 4.8.

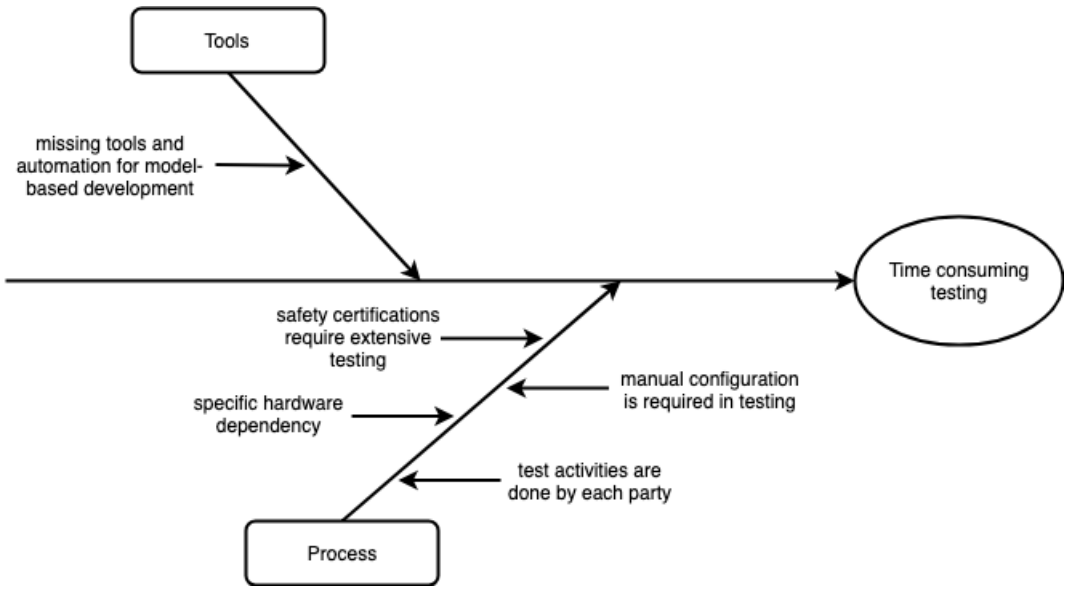

Figure 4.8: The fishbone diagram showing the causes of Challenge 6

Safety certifications require extensive testing. Functional safety certification standards such as ISO 26262 or IEC 61508 require a continuous and big amount of testing activities. This means that, even for a small code integration, all the system testing activities need to be redone. Being able to provide safety certified software is a big advantage for any software supplier, but usually, it is responsibility of the OEM to assure the standard compliance. In fact, at the OEM company it was reported the big amount of time spent by the teams in testing and the impact of it in the overall integration process. They mentioned that this starts from the requirements definition, it impacts during the development and then requires evaulation of the delivery of every software supplier.

> *"[at the interviewer question] No, we cannot test only the delta update or the specific software component, it is necessary to have all green tests again [...] a software component might affect another software components"*

Testing activities are done by each party. Another identified cause making test activities having a big time impact in the integration process is due to the fact that test activities are performed multiple times by each party. During the interviews emerged that software is always delivered along with test reports, however, engineers at the OEM company reported that tested activities are necessary to be performed again in order to verify the test reports.

> *"[...] we test again, to assure that the test reports are correct"*

4.0.2.7 Late defect discovery (CH7)

Late defect discovery represents a problem with the current development process. As already mentioned, the verification and validation of the final system are done uniquely at the last stages by the OEM; software suppliers have testing capabilities limited to the software component they are responsible of developing. For this reason, certain faults can be identified only by tests done by the OEM and the software suppliers cannot have immediate feedback. The causes of this problem are shown in the figure 4.9.
Testing on target is not extensive enough. During the development, software suppliers perform testing activities on only few hardware targets. As a result some faults such as performance issues can be detected only when the software run on ECU hardware.

> *"[...] everything works on the qemu [single target emulator] for a single component. Problems come when you have multiple ECUs networked together or when it runs on real hardware. [...] generally quality requirements are not fully satisfied. For example, we say booting time should be 3 seconds, but when we test it on our boxcar it takes 6 seconds."*

System-level testing activities are done at the last development stages. Test activities at the system-level can be performed only after that every software supplier has

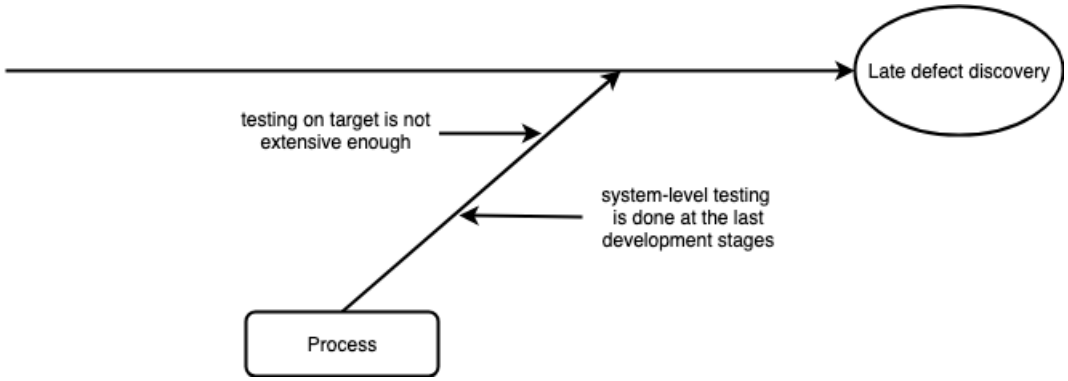

Figure 4.9: The fishbone diagram showing the causes of Challenge 7

deployed and the ECU is delivered. The impossibility of performing system-level test activities also during the development can be considered a major cause of having late feedback on test reports.

4.0.2.8 Delayed response time (CH8)

The reason of having delayed response time is mainly due to the way how the development process is organized. The three-tiered model has been very successful in the industry; the responsibilities are well split, allowing the OEM to focus on requirements elicitation and the design of the system, letting software suppliers be responsible for the code implementation. However, this supply chain seems to include communication difficulties. In fact, during the interviews it was reported the response time from the other software suppliers is very slow since they are generally in direct communication with only the Tier 1. Every specification change or fault report has to pass through multiple communication layers before reaching the team responsible for the development.

> *"[...] once we had a big bug that need to be fixed [...] we had to wait 3 weeks before they fixed it"*

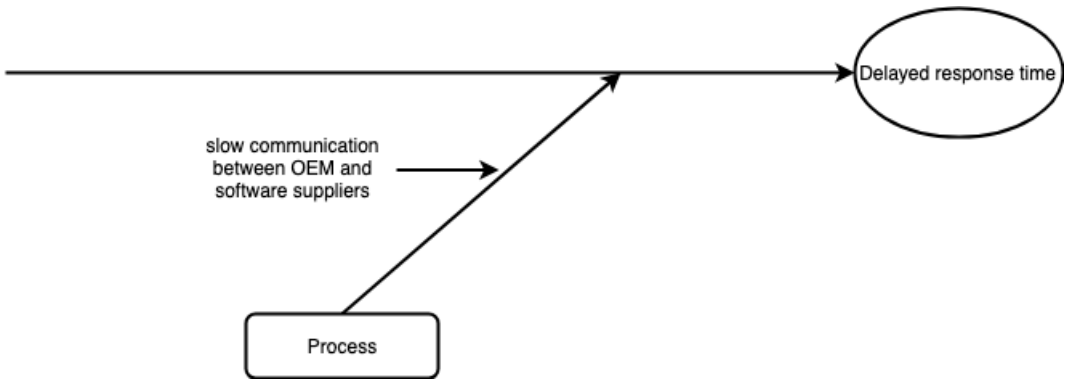

Figure 4.10: The fishbone diagram showing the causes of Challenge 8

4.0.3 Causes frequency

The Cause-and-Effect analysis conducted for the eight identified challenges allows the identification of 24 potential causes. The major part of them are specific to the investigated problem, however, others were connected to two or three challenges. The list of causes with the respective frequency is presented in Table 4.2.

Cause code	**Cause title**	**Cause frequency**
Cause01	Suppliers use different tools	3
Cause02	Slow communication between OEM and software suppliers	3
Cause03	Missing tools and automation for model-based development	3
Cause04	Lack of synchronization during modules development	2
Cause05	System integration is done at only one stage	2
Cause06	Specific hardware dependency	2
Cause07	Manual configuration is required in testing	2
Cause08	Build tools require good experience	1
Cause09	Optimization of the build time is hard to achieve	1
Cause10	Image building cannot be parallelized and it is time-consuming	1
Cause11	Documenting configuration changes takes to much time	1
Cause12	The configuration of the multi-project environment is laborious	1
Cause13	Build system updates require a lot of effort	1
Cause14	Build tools and SDK version updates	1
Cause15	Usage of high caching to speed up the build	1
Cause16	Large commits involving multiple repositories	1
Cause17	Incomplete integration tests	1
Cause18	Missing modules dependencies	1
Cause19	Different environments between development and real target	1
Cause20	Incorrect software interfaces	1
Cause21	Safety certifications require extensive testing	1
Cause22	Test activities are done by each party	1
Cause23	Testing on target is not extensive enough	1
Cause24	System-level testing is done at the last development stages	1

Table 4.2: List of identified causes sorted by frequency

Of course, not all the causes have the same impact on the problems, and only the companies truly know what are the crucial challenges that need to be targeted first. However, instead of addressing individually the causes and proposing potential solutions, a Pareto analysis is done in order to prioritize and show what causes have the major impact on the challenges. In this way, a starting input is given to the companies on what to focus first and try to get numerically the maximum result. The Pareto chart is a simple tool that enables a visualization based on their frequency [54]. It is based on the 80/20 principle where 80% of the problems can be solved by addressing the 20% of the causes. In figure 4.11, a Pareto chart is plot based on the previously introduced table.

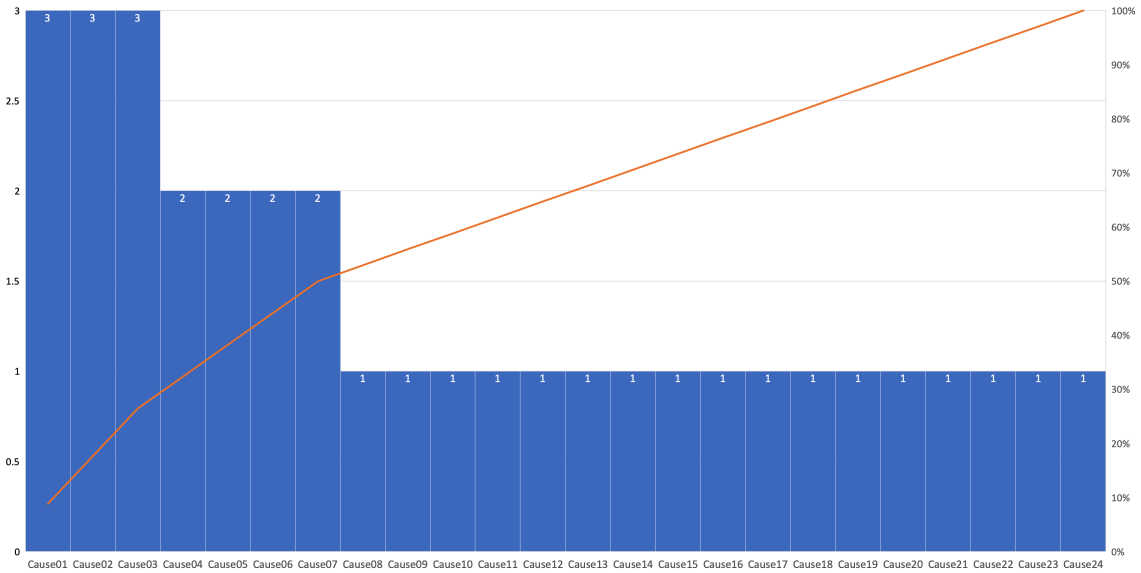

Figure 4.11: Pareto chart showing the frequency of causes of the challenges

4.0.4 Proposed solutions

Once the challenges and their potential root-causes are presented, it is possible to address RQ3 and propose potential solutions in order to overcome or mitigate the impact of the listed challenges. It is important to note, however, that there is no silver bullet. This section aims to give insights into the causes, and provide alternative proposals and suggestions which can serve as input to the case companies for improving their current development process. In fact, some problems present opportunities of improvement.

As previously discussed, in order to reach a solution, it is necessary to address the root causes of the problem and take corrective actions. Moreover, by targeting the causes with the highest frequency, corrective actions can be extended to multiple challenges. For this reason, the first 7 causes in the Pareto chart will be addressed; as shown in Figure 4.11, they have an impact of the 50% on the problems, and they are cause of at least two challenges. The causes and proposed solutions are presented in Table 4.3.

Cause title	Proposed solution
Suppliers use different tools	Automate the data parsing and extraction between the various tools
Slow communication between OEM and software suppliers	Adopt single development pipeline with development teams in direct interaction and communication
Missing tools and automation for model-based development	Include DSL Test Automation software platform in the development process
Lack of synchronization during modules development	Adopt single development pipeline with development teams in direct interaction and communication.
System integration is done at only one stage	Adopt single development pipeline with development teams in direct interaction and communication
Specific hardware dependency	Adopt cross-compilation framework and target multiple hardware devices at the same time
Manual configuration is required in testing	Integrate simulation test activities in the development pipeline

Table 4.3: List of potential solutions proposed

(1) Automate the data parsing and extraction between the various tools. Of course, tool standardization is the most effective solution and would permit direct and simple artifacts exchange, avoiding data discrepancy. However, this is not always possible. As already mentioned, the software companies use in-house developed and maintained tools, built to meet their needs specifically. Considering this, an alternative solution is to facilitate the exchange of artifacts between the OEM and software suppliers by automating the data transformation process. This can be seen as a preparation phase where the output is converted and delivered in the specific format required by the supplier. For instance, outputs can be artifacts such as system description, system and ECU extracts. The architect is aware at the earliest stage of the changes in the artifact, and the supplier does not have to deal with data discrepancy. Once the artifacts are produced, compatible versions for suppliers tools are also automatically generated. In this way, the integration process is facilitated, and the development flow is not interrupted for the time required to the supplier to adapt to the new changes.

(2) Adopt single development pipeline with development teams in direct interaction and communication. This alternative consists of unifying the development of the various suppliers and the OEM into a single continuous integration pipeline. However, every vendor works with its own development process and internal CI environment. The difference from the standard development stands on the idea of using a shared mainline, a single build and deployment pipeline and the CIs interacting with each other. In fact, the system build is required and triggered at every update of any of the software modules and components. In this way,

development teams are in direct interaction and have immediate feedback at each software delivery. Moreover, it would enable testing at the system level at every integration acceptance and regression testing being fundamental. In this scenario, the responsibilities would also remain unchanged during the development process, with the system integration assigned to the Tier 1 or hold by the OEM.

(3) Include automated simulation-based tests of models in the CI pipeline. Model-driven engineering requires testing activities based on specific testing tools such as Simulink Test[13]. These need to be integrated into the continuous integration setup and run every time changes are committed. With this implementation, it is possible to have automated testing activities of models without any human interaction. The workflow of this setup is visible in the figure 4.12.

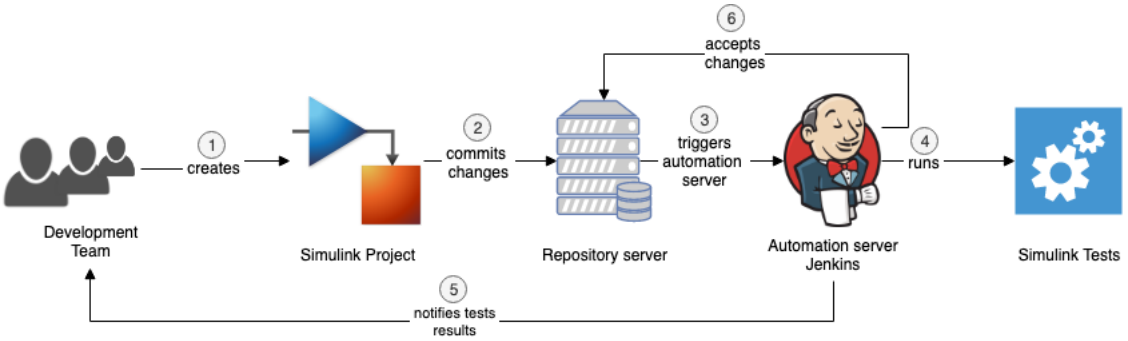

Figure 4.12: Causal relationship between challenges

(4) Adopt cross-compilation framework and target multiple hardware devices at the same time. The use of cross-development and cross-compilation toolchains offered by frameworks such as BuildRoot or Yocto Projet allow to have a single development and targeting multiple hardware. Multiple custom SDK have to be configured on the same development workflow and the build target multiple hardware at the same time.

(5) Integrate simulation test activities in the development pipeline. In order to reach the maximum automation possible and reduce the amount of resources required during test activities, manual configuration needs to be avoided. Verification activities are automatically performed on a simulation-based environment where ECUs are virtualized or physical ECUs are connected to a simulator. These type of testing systems, such as the one provided by dSPACE[14], allow quick and scalable configurations of single ECUs or complete ECU networks.

[13]https://se.mathworks.com/products/simulink-test.html
[14]https://www.dspace.com/en/inc/home/products/systems/ecutest.cfm

4.0.5 Validation of the solution and proposed Proof of Concept

To validate the proposed alternatives, a demonstrative CI workflow of a sample project was implementated. The objective of the project was the development of an ECU that executes a sample Adaptive AUTOSAR Application following the alternative solutions provided in the previous section. It was not possible to include the use of specific commercial licensed software and advanced hardware devices, however, the scope of this setup was to provide the configuration and the design of a CI pipeline rather that reporting tool functionalities and hardware capabilities.

The case companies were involved during the process and they assumed their traditional roles; the OEM was responsible for providing the specifications and for the development of the software application, while the software supplier for the development of the AUTOSAR Platform. The researcher assumed the role of the integrator and set up the CI infrastructure for the project. The setup consisted of an automated multi-stage CI with the two companies in direct interaction. In fact, both companies worked with their own CI infrastructure during their development, but the integration was done on a single working mainline.

At the end of the project, two workshops were conducted with team members of the development teams of both companies in order to evaluate the realization of the proof of concept. It was important to understand to what extent the solutions provided and how much of the PoC implementation can be adopted in a real world development scenario.

4.0.5.1 Source Code Management

As source code management, the case companies used GitLab for the internal projects, and the same tool was adopted for the integration. Software modules and the software component were all stored in separates GitLab repositories and then assembled in a single base repository. The use of a multiple repositories was a key factor for having a better modularity and a better version control of each software release.

4.0.5.2 Build system

The build system was based on Yocto Project, which uses OpenEmbedded[15], a linux-based cross-compilation build automation framework. The Yocto meta layers were cloned into the project directory from open source repositories. These include also the meta layers required to build for the selected target hardware. However, in case of advanced hardware, proprietary meta layers would have been required. The build system was automated using the open source tool Jenkins; it allowed a fully configurable environment, including pre and post build tasks execution. The build was done on a physical machine consisting of 4 CPUs and 16GB of RAM, which

[15]https://www.openembedded.org/wiki/Main$_{P}age$

also stored all cached files.

4.0.5.3 Target hardware

The target hardware were a Raspberry Pi B+[16] and MinnowBoard Turbot Dual-Core[17]. The selection of the this hardware was due to the simple fact that they are officially supported by the AUTOSAR consortium. Two custom SDK were built and generated for each of the targeted hardware device.

4.0.5.4 CI Workflow

The build process was divided in organized tasks carried out by Jenkins through Jobs. The first step consists of the delivery of software by the development teams of the OEM and Software suppliers (2). The version of the software that is delivered is expected to be a working version, where verification activities at the platform and software component level have already been done by the development teams (1). In fact, this phase includes the commit of test reports and changelogs made. Code Review activities are performed at this stage (3) to make sure that there are no merging issues. Once the changes have been been approved a system build is triggered (4). All the software modules and components are assembled in a single git repository, together with the meta layers required for the build (5). Two Custom SDK versions targeting the two different were used to perform the builds (6). The build were parallelized using two Jobs tasks scheduled on Jenkins. Each system build generated the ECU software image for the specific target hardware (7), and the results were published in a dedicated dashboard screen (8). It was then possible to perform system level testing on the single physical hardware by deploying the image into the ECU and running tests (9). In parallel test activities could have been performed using a simulation-based environment and verifying the functionality of when inserted in a ECU network (10). The results of the testing activities were sent to the integration and release managers (11). Once these are approved, it is possible to deploy the software on production hardware (12). Figure 4.13 shows the workflow, while in Figure 4.14 the Causality Viewpoint of the CI flow is presented.

[16]https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus
[17]https://minnowboard.org/

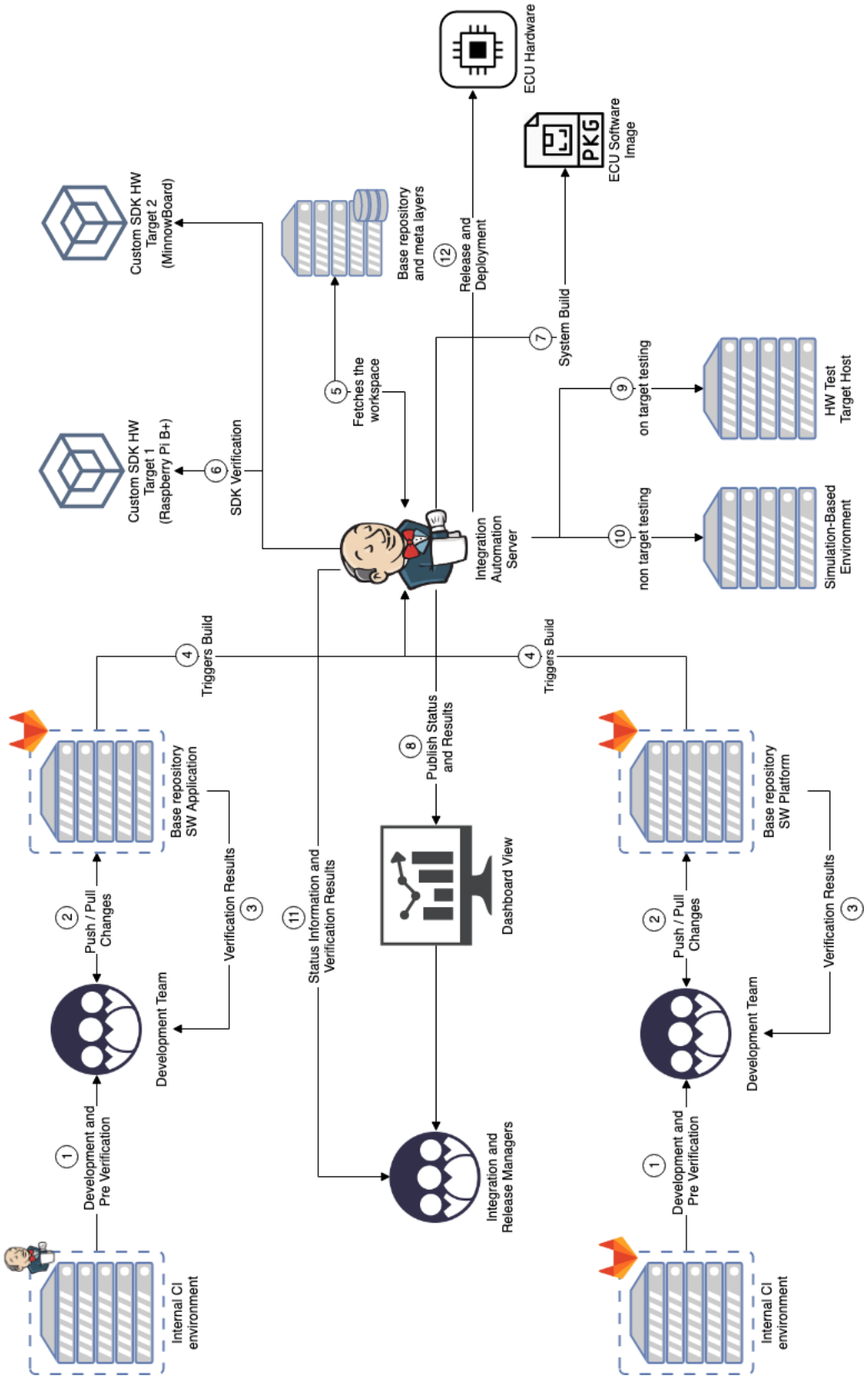

Figure 4.13: Continuous Integration environment setup

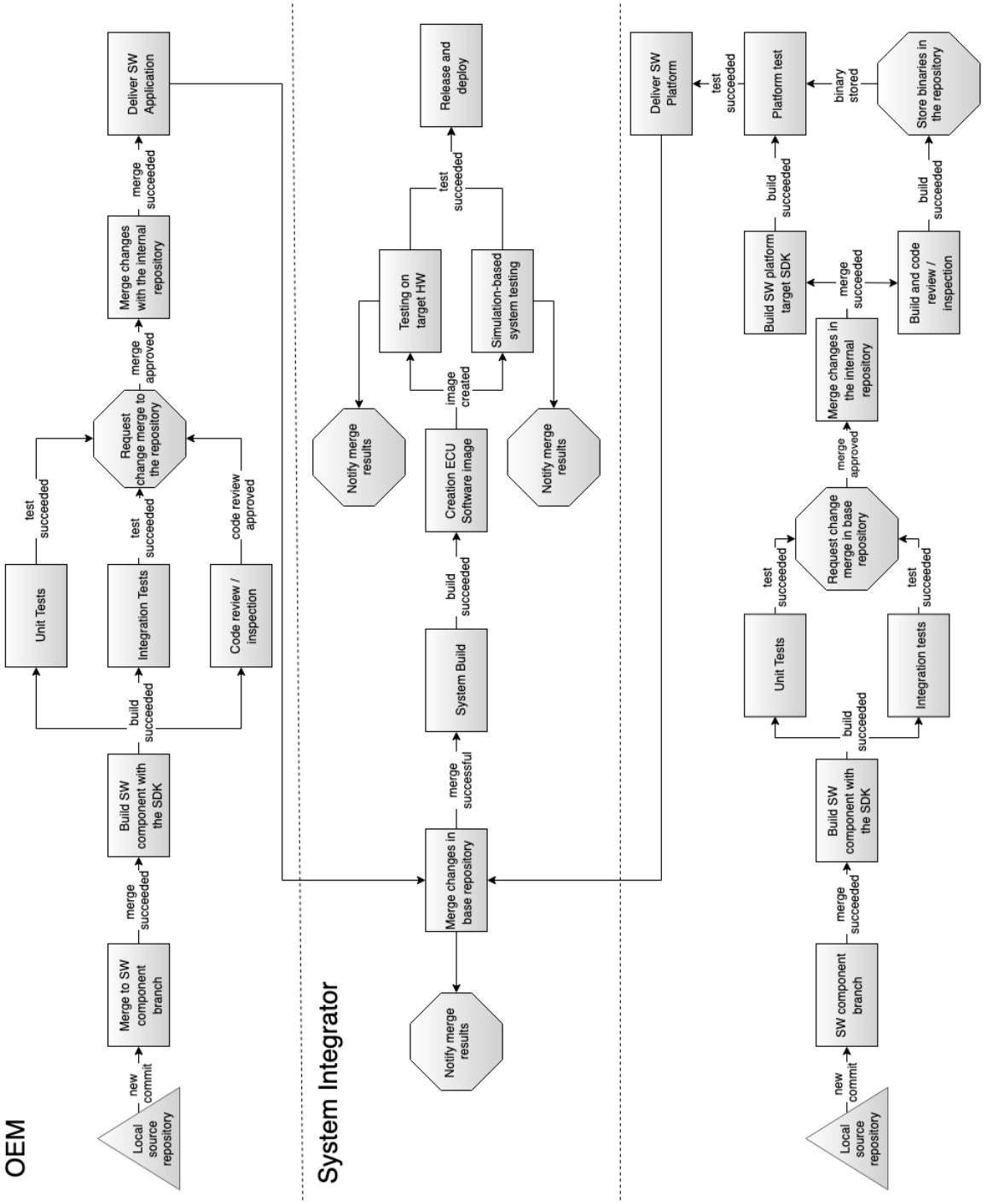

Figure 4.14: Cinders - Modeling of the CI infrastructure

5
Discussion

In this section, the findings are discussed. First, the research questions are answered and for each of them, the results are evaluated. Then, the implementation of the proof of concept and its evaluation are discussed.

5.0.1 RQ1: Challenges of CI adoption

Eight CI adoption challenges are identified and they are classified based on the stage of the CI workflow that they are identified. The classification consists of four groups: build design, integration process, testing and organizational structure.

Complicated build system and process. One of the challenges when adopting CI for large-scale projects according to Bosch et al. [32] concerns the build and the integration flow; in such projects, the final product is an assembly of various subsystems and these need to be part of a CI flow. In addition, continuous integration is even more challenging when the software is constrained to physical and hardware parts [32]. Similarly, findings from this study show that a challenge faced by the case companies is relative to the complexity of the overall build process. The integration of different sub-modules into a software platform, and subsequently the integration of the software platform with the software application into an ECU image are reported by the case companies to be challenging.

Broken builds. Another identified challenge is the occurrence of broken builds. According to Claps et al. [56] broken builds have the highest priority and to be promptly resolved; the complete development team has to be involved and be responsible for fixing it. At the case companies, broken builds are reported to be frequent and the teams have to put hours, even days on particular occasions, in order to have a working version of the system. Based on the cause, the responsibility of resolving is assigned to the single developer causing the failure or the whole team in case of major CI environment changes.

Interruption of the development workflow. Frequent interruptions of the development workflow are another challenge identified in this study. Meyer et al. [52] report that interruptions can lead to productivity drops, higher error rate and slow resumption. At the case companies, interruptions of the workflow can occur for days, and this clearly affects the teams' performance.

Difficult system integration. According to Debbiche et al. [12] coordinating code

dependencies becomes more difficult when adopting CI; it is important to take into consideration how the work is divided among the developers and how it affects the integration process. These difficulties are found also in this study. The integration process is reported to be problematic, especially considering that it is done at one stage by a single company; the work of several development teams, also belonging to different companies, has to be integrated into the final system.

Lack of automation. Olsson et al. [33] reports that physical hardware can represent a barrier for test activities when practicing CI. The absence of test automation and the lack of tools and support are a challenge for the companies. The specific configuration and hardware dependency require system test activities to be run manually by the developers; the absence of automation and human work involvement can certainly limit the benefits of practicing CI [9].

Time consuming testing. Another issue found is relative to the large amount of resources required to perform testing. One of the benefits of CI is the possibility of automating long testing tasks, allowing the team to run them in parallel and be operative while waiting for the results [33]. According to the findings, most of the system testing activities require continuous human interaction. Furthermore, part of the automotive software is safety-critical; testing and safety certifying such software is both difficult and time consuming [57].

Late defect discovery. Early fault detection is a key element that software companies try to achieve; research studies show that the costs of resolving defects late can increase the project costs up to 50 percent [58]. The results show that with the current CI workflow defects are discovered at the last development stages. Software is tested independently by each vendor, and the system testing is done only after their integration. As a result, defects are discoverable only at this phase.

Delayed response time. Results from this study show that slow communication among vendors represents a current challenge. Bergadano et al. [55] report the impact and the advantages of efficient communication in a distributed, Agile multi-site development. The delays in response time found at the case company make collaboration difficult, impacting tasks coordination, activities control and the overall productivity of the development teams.

5.0.2 RQ2: The causes of the challenges

The aim of RQ2 is to identify the root causes of the challenges that are provided in answer to RQ1. In total, 24 causes are identified by analyzing the reported challenges and they are listed in Table 4.2. As shown, some causes are unique to specific challenges, while others have bigger effects and are the cause for multiple challenges. For instance, the difference in tools adopted by the various vendors and the absence of model-based testing ones are identified in three challenges. However, these are known issues in the industry, and related problems are already presented in other research papers [12, 28]; Debbiche et al. [12] report the maturity of the

tools and the infrastructure as a challenge itself when adopting CI. In addition to the lack of maturity, Shahin et al [62] considers a problem also the absence of approaches and practices in order to facilitate the collaboration in distributed teams using different tools.

Another two major causes are the lack of synchronization and slow communication between the companies. Problems with communication in large-scale projects between OEM and suppliers have already been reported in requirements engineering [63]. However, these challenges seem to extend also to other the development phases, especially considering that collaborative development is becoming more and more essential for the development of automotive software.

5.0.3 RQ3: Solution to the causes

RQ3 aims to propose solutions and solve the major causes that generate the challenges. In this study potential solutions are provided for the seven root-causes that had the highest frequency.

Slow communication among software vendors, lack of synchronization, and the single-stage system integration are addressed by a common potential solution consisting of the adoption of a single working pipeline among the various development teams. In this way, direct interaction and immediate feedback can be provided at every software integration. Furthermore, the work needs to be coordinated and the development teams have to ensure that the code dependencies are respected in order to perform a full system build.

The adoption of cross-compilation framework capable of targeting multiple hardware devices and the use of simulation-based environment are provided as alternative solutions for the hardware dependency problem and the manual configuration required for testing activities. It is important to mention, however, that solutions consisting of the implementation simulation-based environment for testing have already been successfully implemented by some automotive vendors as well as for safety-critical software of other domains [59, 60].

The inclusion in the CI workflow of available tools capable of automating model-based testing activities is provided as a solution for the system-level testing. Research studies [61] on Model-based testing (MBT) show promising results in automating multiple model-based testing activities.

The solutions are implemented in a proof of concept CI workflow for the development of a demo application software. The implementation takes inspiration from the workflow adopted by the AUTOSAR consortium for the development of the AUTOSAR Adaptive Platform. In particular, it reproduces the type of modularization and the assembly of the companies work in a single development pipeline. The proposed proof of concept extends this design to a full product development; the software deliveries of every supplier are integrated into the single

pipeline flow, including all the software applications. Testing and automation of simulation-based tests activities proposed as solutions are also included in the CI workflow. Furthermore, the adoption of cross-platform tools showed capabilities of targeting multiple hardware targets with a single development pipeline.

Finally, the implementation of a diagnostic software application integrated with the Adaptive Platform was done in order to evaluate the PoC with the case companies. The requirements for the CI setup were the following:

- involve and integrate both companies development work with automatic interaction between the internal CIs;
- incorporate all the development activities, from the application development to the ECU software image generation;
- include the proposed solutions;
- immediate build feedback to the developers;
- flexibility to include new SDKs for targeting other hardware devices;
- possibility to perform full builds of the system, for all the targeted devices;
- possibility to perform individual builds of the modules and software components.

The CI environment was analyzed during two evaluation workshops with the integration managers at Tier 2 company and with the development team at the OEM company. The Causality Viewpoint of the CI demo presented in Figure 4.14 was used as a support during the assessment of the setup configuration. After the evaluation, the requirements were all considered fulfilled, and the assessment of the CI setup and execution were positive. At the Tier 2 company the engineers commented that the CI setup presents interesting inputs that could be used for a production development; similar feedback was received also from the OEM company even though based to their opinion, the use of a more complex application would have provided more prominent results. However, this could not be done since none of the two companies could provide any of the software applications they have in production. Therefore, part of the feedback is left as possible direction of future work.

6
Conclusion

In this thesis work, the challenges of practicing continuous integration in a multi-vendor environment for the development of automotive software are addressed. The study is conducted at the development section of two leading companies in the automotive industry, and the problems related to the development process and the CI infrastructure are assessed. In addition, the causes of such challenges are determined and potential solutions to solve them are provided.

Based on semi-structured interviews and continuous observations, the findings show the case companies facing eight major challenges in different stages of the development process. Moreover, modeling the continuous integration infrastructures using the architecture framework Cinders contributes to better understand the workflow and to which stage the problems belong. The assessed challenges are relative to the build system complexity, difficulties in the integration process, frequent interruption of the development flow, lack of automation and late defect discovery. For each challenge, Fishbone diagrams are used to establish the potential root causes. The identification of the causes is considered as the first step for solving a problem. Furthermore, correcting the causes is an effective way of solving problems, and in case this is not entirely possible, mitigate them.

Once the causes are listed and analyzed, they are sorted by frequency, and a Pareto chart is plotted to show their impact and the cumulative effect on the problems. The causes with the higher frequency (reported in at least two problems) are addressed and potential solutions for them are suggested. The adoption of a single development pipeline with development teams in direct communication and interaction is provided as a common solution for three main causes. In addition, specific testing alternatives and the use of cross-compilation frameworks are proposed in order to automate testing activities and facilitate the system integration process.

After presenting the solutions, these are implemented in a sample development scenario with the contribution of the two companies. The researcher set up the Continuous Integration environment and the project is successfully carried out. The conduction of two workshops validated the results and established that the proposed solutions can be considered valid and applied to a real-world project development.

This is a single case study, and despite the fact that the results can be considered

positive for the case companies, these are still generalizable to a limited extent. However, valuable input is for the companies to improve their development process and the validation of the results provides good insights for researchers to which can be the obstacles when adopting the practice of continuous integration and what potential solutions can be adopted. The field of CI lacks of research, especially when applied to the development of embedded systems software.

Future research if to further examine the effect of the solutions in concrete large-scale development projects; different scheduling and synchronization strategies might be required when the work of several suppliers needs to be integrated into a single CI pipeline. In addition, mitigation solutions are provided for only a part of the challenges. The build design and related problems can be investigated more, and solutions can be provided to the causes that still remain unsolved.

Bibliography

[1] Lorenz Slansky, Thomas Scharnhorst (2017). AUTOSAR for Intelligent Vehicles.

[2] AUTOSAR. https://www.autosar.org/

[3] Bosch, J., Eklund, U. (2012). Eternal Embedded Software: Towards Innovation Experiment Systems. Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change Lecture Notes in Computer Science.

[4] Rana, Rakesh Staron, Miroslaw Berger, Christian Hansson, Jörgen Nilsson, Martin Törner, Fredrik. (2013). Increasing Efficiency of ISO 26262 Verification and Validation by Combining Fault Injection and Mutation Testing with Model Based Development.

[5] Eklund, U., Olsson, H. H., Strøm, N. J. (2014). Industrial Challenges of Scaling Agile in Mass-Produced Embedded Systems. Lecture Notes in Business Information Processing Agile Methods. Large-Scale Development, Refactoring, Testing, and Estimation.

[6] AUTOSAR. Adaptive Platform: Autosar. https://www.autosar.org/standards/adaptive-platform/

[7] Katumba, B., Knauss, E. (2014). Agile Development in Automotive Software Development: Challenges and Opportunities. Product-Focused Software Process Improvement Lecture Notes in Computer Science.

[8] Fogelström, N. D., Gorschek, T., Svahnberg, M., Olsson, P. (2010). The impact of agile principles on market-driven software product development. Journal of Software Maintenance and Evolution: Research and Practice, 22(1).

[9] M. Fowler. (2006). Continuous integration.

[10] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J. Thomas, D. (2001). Manifesto for Agile Software Development Manifesto for Agile Software Development.

[11] Ståhl, D., Bosch, J. (2014). Continuous Integration Flows. Continuous Software Engineering, 107-115.

[12] Debbiche, A., Dienér, M., Svensson, R. B. (2014). Challenges When Adopting Continuous Integration: A Case Study. Product-Focused Software Process Improvement Lecture Notes in Computer Science.

[13] Martensson, T., Stahl, D., Bosch, J. (2017). The EMFIS Model — Enable More Frequent Integration of Software. 2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA).

[14] Ståhl, D., Bosch, J. (2017). Cinders: The continuous integration and delivery architecture framework. Information and Software Technology, 83, 76-93.
[15] Shahin, Mojtaba Ali Babar, Muhammad Zhu, Liming. (2017). Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices.
[16] Kallerdahl, A. (2018). Adaptive AUTOSAR - A new way of working. https://www.kpit.com/articles/adaptive-autosar-a-new-way-of-working
[17] Nicolaescu, S. S., Palade, H. C., Dumitrascu, D. D., Kifor, C. V. (2017). A new project management approach for RD software projects in the automotive industry - continuous V-model. International Journal of Web Engineering and Technology, 12(2).
[18] Staron, M. (2017). Evaluation of Automotive Software Architectures. Automotive Software Architectures.
[19] AUTOSAR. Motivation and Goals. https://www.autosar.org/about/basics/motivation-goals/
[20] Martínez-Fernández, S., Ayala, C. P., Franch, X., Nakagawa, E. Y. (2015). A Survey on the Benefits and Drawbacks of AUTOSAR. Proceedings of the First International Workshop on Automotive Software Architecture - WASA 15.
[21] Menon, S., Venugopal, P. (2014). AUTOSAR Software Platform Adoption: Systems Engineering Strategies. SAE Technical Paper Series.
[22] Gronniger, H., Hartmann, J., Holger, K., Kriebel, S., Rothhardt, L., Rumpe, B. (2008). View-Centric Modeling of Automotive Logical Architectures.
[23] Furst, S., Bechter, M. (2016). AUTOSAR for Connected and Autonomous Vehicles: The AUTOSAR Adaptive Platform. 2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshop (DSN-W).
[24] AUTOSAR. Methodology for Adaptive Platform. https://www.autosar.org/standards/adaptive-platform/
[25] Pelliccione, P., Knauss, E., Heldal, R., Ågren, S. M., Mallozzi, P., Alminger, A., Borgentun, D. (2017). Automotive Architecture Framework: The experience of Volvo Cars. Journal of Systems Architecture.
[26] IEEE Guide:–Adoption of ISO/IEC TR 24748-3:2011, Systems and software engineering-Life cycle management (Software life cycle processes).
[27] Agren, S. M., Knauss, E., Heldal, R., Pelliccione, P., Malmqvist, G., Boden, J. (2018). The Manager Perspective on Requirements Impact on Automotive Systems Development Speed. 2018 IEEE 26th International Requirements Engineering Conference (RE).
[28] Berger, C., Eklund, U. (2015). Expectations and Challenges from Scaling Agile in Mechatronics-Driven Companies – A Comparative Case Study. Lecture Notes in Business Information Processing Agile Processes in Software Engineering and Extreme Programming.
[29] Paasivaara, M., Lassenius, C. (2016). Challenges and Success Factors for Large-scale Agile Transformations. Proceedings of the Scientific Workshop Proceedings of XP 2016 on - XP 16 Workshops.
[30] Meyer, B. (2014). The Ugly, the Hype and the Good: An assessment of the agile approach. Agile!

[31] Mahally, M.M., Staron, M., Bosch, J. (2015). Barriers and enablers for shortening software development lead-time in mechatronics organizations: A case study. Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015.
[32] Bosch, J. (2016). Continuous software engineering: An Introduction.
[33] Olsson, H.H, Alahyari, H., Bosch, J. (2012) Climbing the “Stairway to Heaven": Evolving From Agile Development to Continuous Deployment of Software.
[34] Beck, K. (1999). “Embracing Change with Extreme Programming.”
[35] Leppanen, Marko, et al. (2015). The Highways and Country Roads to Continuous Deployment.
[36] Ståhl, Daniel Bosch, Jan. (2013). Experienced Benefits of Continuous Integration in Industry Software Product Development: A Case Study.
[37] Miller,A., 2008. A hundred days of continuous integration.
[38] Farley, D., Humble, J. (2010). Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional.
[39] Humble, J., Farley, D. (2015). Continuous delivery: Reliable software releases through build, test, and deployment automation. Upper Saddle River, NJ: Addison-Wesley.
[40] T. Mårtensson, D. Ståhl and J. Bosch. (2017) Continuous Integration Impediments in Large-Scale Industry Projects. IEEE International Conference on Software Architecture (ICSA).
[41] Ståhl, Daniel Bosch, Jan. (2014). Modeling continuous integration practice differences in industry software development.
[42] Runeson, P., Höst, M. (2008). Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering, 14(2).
[43] Lethbridge, T. C., Sim, S. E., Singer, J. (2005). Studying Software Engineers: Data Collection Techniques for Software Field Studies. Empirical Software Engineering, 10(3).
[44] Yin, R. K. (2003). Case study research: Design and methods. Thousand Oaks, CA: SAGE Publications.
[45] Colin Robson (2011). Real World Research. A Resource for Social Scientists and Practitioner-Researchers (Third Edition).
[46] Merriam, S. B. (1998). Qualitative research and case study applications in education.
[47] Runeson, P., Höst, M., Rainer, A., Regnell, B. (2012). Case Study Research in Software Engineering.
[48] Verner, J., Sampson, J., Tosic, V., Bakar, N. A., Kitchenham, B. (2009). Guidelines for industrially-based multiple case studies in software engineering. 2009 Third International Conference on Research Challenges in Information Science.
[49] Gomaa, H. (n.d.). Overview of Software Modeling and Design Method. Software Modeling and Design.
[50] Ståhl, D., Bosch, J. (2018). Cinders. Proceedings of the 2018 International Conference on Software and System Process - ICSSP 18.
[51] Andersen, B., Fagerhaug, T. (2002). Root Cause Analysis: Simplified Tools

and Techniques. Journal For Healthcare Quality, 24(3).
[52] Meyer, Andre E. Barton, Laura Murphy, Gail Zimmermann, Thomas Fritz, Thomas. (2017). The Work Life of Developers: Activities, Switches and Perceived Productivity. IEEE Transactions on Software Engineering.
[53] Yocto Project. https://www.yoctoproject.org/
[54] Powell, Taman Sammut-Bonnici, Tanya. (2015). Pareto Analysis.
[55] Bergadano, F Bosio, G Spagnolo, S. (2014). Supporting collaboration between customers and developers: A framework for distributed, Agile software development. International Journal of Distributed Systems and Technologies.
[56] Claps, G. G., Svensson, R. B., Aurum, A. (2015). On the journey to continuous deployment: Technical and social challenges along the way, Information and Software Technology.
[57] Notander, J. P., Höst, M., Runeson, P. (2013). Challenges in Flexible Safety-Critical Software Development – An Industrial Qualitative Survey.
[58] Damm, Lars-Ola. (2007). Early and Cost-Effective Software Fault Detection - Measurement and Implementation in an Industrial Setting.
[59] Tong, D. S., Awatsu, L., Hubrechts, J., Bhave, A., Van der Auweraer, H. (2017). A simulation-based testing and validation framework for ADAS development.
[60] Lee, H. S., Lee, S. J., Park, J., Lee, E. Kang, H. G. (2018). Development of simulation-based testing environment for safety-critical software, Nuclear Engineering and Technology.
[61] Pröll, R., Bauer, B., (2018). A Model-Based Test Case Management Approach for Integrated Sets of Domain-Specific Models. IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW).
[62] M. Shahin, M. Ali Babar and L. Zhu, "Continuous Integration, Delivery and Deployment: A Systematic Review on Approaches, Tools, Challenges and Practices," in IEEE Access, vol. 5, pp. 3909-3943, 2017.
[63] Liebel, G., Tichy, M., Knauss, E., Ljungkrantz, O., Stieglbauer, G. (2016). Organisation and communication problems in automotive requirements engineering. Requirements Engineering, 23(1).

A

Appendix 1

(Common) Background level interview questions:
1. What is your name and what is your current job title?
2. How long have you been working at the company?
1. How long have you been working in the field of automotive software engineering?
2. Which is your team, what is your role and how does it fit into the project?
3. Can you briefly describe which agile methodologies your current team has used?
4a. (if CI is mentioned) How long have you been using CI in your team?
4b. (if CI is not mentioned) Is your team using CI to integrate software?
5. (if more than this project) How long have you been working with AUTOSAR?
6. Do you have experience with the AUTOSAR Classic architecture?
7. What benefits are gained from the Adaptive standard – both for OEMs and software suppliers?
(Common) Organizational and team level interview questions:
8. Which is the current CI workflow of your system?
9. How often does the team integrate?
10. What do you think are the benefits of it?
11. Have you or your team faced any difficulty when using CI?
12a. (if yes) How did you solve?
12b. (if no) Is there anything that would you change?
13. Do you have any guide or workflow explanation for new developers?
14a. (if yes) Do you think that is sufficient to have a clear view of the integration process?
14b. (if no) Do you think the company should have some documentation in these cases?
15. How long does it take to a new developer to start contributing and integrate new code?
16. (intro about the integration with the other company's software: the application will be integrated with the adaptive platform) How do you think the synchronization would work with the other teams?
17. How would the CI system work when you need to release new software for the supplier / OEM?
18. Can you identify any challenge or barriers in a similar environment?
(Common) Integration process level interview questions:
19. What building environment do you use for production and testing?
20. How long does it require to generate a build?

21. How do you and your team deal with a broken build?
22. Who is responsible to fix it?
23. How much does a broken build affect the team workflow?
24. Who is responsible for testing and what kind of testing does the team do?
25. How much of your workflow is automated? Is there any human manual work involved in the deployment process?
26. (if yes) Can also that process be automated?
27. How do you synchronize the CI for different software components of the same software product?
(Common)Integration process level interview questions:
28. What building environment do you use for production and testing? 29. How long does it require to generate a build?
30. How do you and your team deal with a broken build?
31. Who is responsible to fix it?
32. How much does a broken build affect the team workflow?
33. Who is responsible for testing and what kind of testing does the team do?
34. How much of your workflow is automated? Is there any human manual work involved in the deployment process?
35. (if yes) Can also that process be automated?
36. How do you synchronize the CI for different software components of the same software product?

Specific interview questions at Tier2:
1. Which is the development process for the new Adaptive Platform?
2. How do you do the software verification of the software components?
3. How do you integrate the different software components and build the full system?
4. What quality assurance checks do you perform after each build?
5. Do you do any kind of testing for real target environments (hardware?)
6. How often do you release or plan to release?
7. How do you integrate the new software or software updates from the OEM?
8. Do you go through a complete system build and testing at each update?

Specific interview questions at the OEM company:
1. How do you integrate the different software components and build the full system?
2. How do you do the software verification of the software components?
3. What quality assurance checks do you perform after each build?
4. Do you do any kind of testing for real target environments (hardware?)
5. How do you integrate and synchronize the new software or software updates from the OEM and other software suppliers?
6. Which is the development process for the Adaptive Applications?
7. Will there be any dynamic testing of the application after the deployment? How about functional safety checks?

8. Are you using Model-based development?
9. Do you think there are some impediments of this methodology when combined to CI?