Agile Model-Driven Engineering in Mechatronic Systems - An Industrial Case Study

Ulf Eliasson[1], Rogardt Heldal[2], Jonn Lantz[1], and Christian Berger[3]

[1] Volvo Car Group, Sweden
ulf.eliasson@volvocars.com jonn.lantz@volvocars.com
[2] Chalmers University of Technology, Sweden
heldal@chalmers.se
[3] University of Gothenburg, Sweden
christian.berger@gu.se

Abstract. Model-driven engineering focuses on structuring systems as well as permitting domain experts to be directly involved in the software development. Agile methods aim for fast feedback and providing crucial knowledge early in the project. In our study, we have seen a successful combination of MDE and agile methods to support the development of complex, software-driven mechatronic systems. We have investigated how combining MDE and agile methods can reduce the number of issues caused by erroneous assumptions in the software of these mechatronic systems. Our results show that plant models to simulate mechanical systems are needed to enable agile MDE during the mechatronic development. They enable developers to run, verify, and validate models before the mechanical systems are delivered from suppliers. While two case studies conducted at Volvo Car Group confirm that combining MDE and agile works, there are still challenges e.g. how to optimize the development of plant models.

Keywords: Model Driven Engineering, Agile, Mechatronic Software Development, Virtual Testing, Assumptions, Plant Models.

1 Introduction

Developers working on complex embedded systems are consciously and unconsciously making assumptions early in the project about properties and behavior of other components in the system [1, 17, 21]. For example, a developer might make the implicit assumption that the velocity signal from one component is supposed to be in km/h, but it is in fact provided as m/s. These assumptions are often difficult to verify until later phases in a project, such as subsystem integration, because it is common that development is distributed within and between companies. When assumptions lead to unexpected behavior, the costs for fixing these issues increase rapidly the longer they go undetected [19]. Therefore, there is a clear need to address the negative impact of assumptions already early during the development.

Today's automotive industry undergoes a rapid transformation from a mainly mechanical industry into a computerized electromechanical industry where cars are composed of several mechatronic systems, which interact with their surroundings, e.g. autonomous emergency braking. Already twelve years ago it was estimated that 80% of the automotive innovation stems from electronics [13], mainly driven by software, and the size and complexity of software in cars have continued to grow exponential [7]. For example, a modern hybrid electric car has more than 100 electronic control units (ECU), collaborating in a complex in-vehicle network and executing several gigabytes of software.

Sequential development processes that are traditionally used during vehicle development have shown to be insufficient for handling such an exponential growth of software [5, 7]. Furthermore, car manufacturers consider vehicle functions powered by software as a competitive advantage and hence, they tend to develop an increasing amount of this software in-house. Thus, innovation cycles can be shortened as well.

We conducted a case study at the Volvo Car Group (VCG) in Sweden at the Department of Electric Development to better understand challenges originating from combining MDE with agile methods. The main findings from this study are:

– It is possible to combine MDE and agile methods during the development of complex mechatronic systems at automotive original equipment manufacturers (OEM) to gain more knowledge earlier in the projects and decrease the number assumptions.
– Virtual test environments to validate software components are a flexible instrument to support MDE and agile to improve the quality of software regarding wrong assumptions.

Overview: In Sec. 2, we present the development process at VCG to provide the background on how software development is usually carried out at automotive OEMs and Sec. 3 outlines the problem domain and motivation. In Sec. 4, we present our research questions and the methods we used to address them. Sec. 5 reports about the design and results from an exploratory study that was conducted as a pre-study to address RQ-1 and to derive RQ-2 and RQ-3. To address these derived research questions, we did two case studies at VCG, which are reported in Sec. 6. The findings from the investigated cases are analyzed and discussed in Sec. 7. Related work is described in Sec. 8 before a summary and conclusion is provided.

2 Background

Mechatronic development, such as for a modern car, involves software, hardware and mechanical development. The widely adopted model for this distributed development is the V-model, see Fig. 1. The V-shape represents the journey for each component in the car, from the high level requirement and design, via

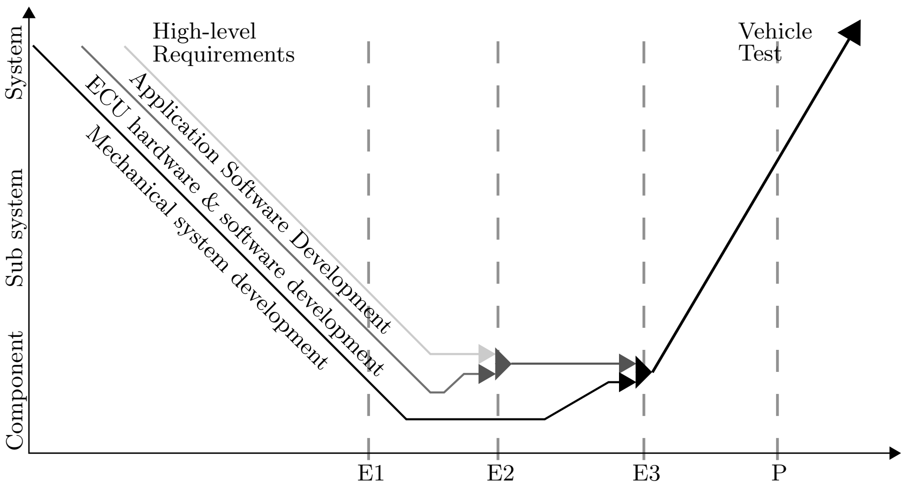
System
Sub system
Component
High-level Requirements
Application Software Development
ECU hardware & software development
Mechanical system development
Vehicle Test
E1
E2
E3
P

Fig. 1. The V-model as it is implemented at VCG for a car development project. Software, hardware and mechanical development happens in parallel and is integrated at certain points during the project.

the detailed component development and up to verification of the integrated component in the product.

As shown in Fig. 1, the overall system development process at VCG is a sequential process with a number of points in time where artifacts are to be delivered. There are three tracks of parallel development: (a) the software components (SWC), (b) the hardware like ECU, and (c) mechanical parts. E1-E3 are electronic integration points and P is where the software should be production ready. The OEM has an overall system responsibility but purchases mechanical systems, hardware and software from suppliers. The suppliers then deliver their components for integration in later phases of the project. VCG has traditionally ordered all the software from suppliers, except for the engines. Recently, an initiative for developing more software in-house has been initiated to keep domain knowledge as a competitive advantage within the company and to improve the speed of innovation.

To meet deadlines in the project, the in-house software development needs to be started before suppliers deliver their components. This is a major challenge for the development teams because they do not have the components available to validate that their assumptions on the behavior of the component are correct.

At the beginning, all requirements and the system design for the next iteration is captured as a model inside a custom-made tool, referred to as SysTool. This model contains, among other things, software components, the requirements they should realize, their deployment on ECUs within the car, and the communication between them. At a certain point in time, this model is frozen and no new changes are allowed until the next iteration starts. These freezes usually last for 20 weeks.

The SysTool model is the single point of truth with respect to two aspects: Firstly, it is used to schedule the communication on the in-vehicle networks; secondly, the component model is transformed into Simulink model skeletons. Each Simulink model represents an ECU with skeletons of the deployed software components including ports and connections. These models are then complemented with functionality by developers as specified by textual requirements. If the system model changes, the Simulink models are updated to reflect these adaptations and preserve any existing implementation.

The executable Simulink models are tested in a virtual, model-in-the-loop (MIL), environment. In MIL-testing the software models are executed within the modeling tool. So-called plant models are used for simulating the surroundings of the ECUs, including software and mechanical components. This enables the developer to get instant feedback by running and testing their models on their PCs. MIL-testing is the focus for this article in terms of investigating and understanding the role and impact of assumptions.

The suppliers provide their software as binaries to protect their intellectual property (IP). Code is generated from in-house models, compiled to a binary and linked with the supplier modules. The resulting software is transferred to the hardware and tested in hardware-in-the-loop (HIL) test rigs. HIL testing means that the code is executed on the intended ECU-hardware and uses real network buses but the rest of the environment surrounding the ECU is simulated. Due to the fact that suppliers provide binaries, this is the first time that both, in-house and supplier developed software, can be integrated and validated together.

The final phase of testing is when software and hardware is integrated with the mechanical systems in a complete prototype vehicle and tested. In these prototypical cars, the whole system is tested altogether.

3 Motivation and Problem Domain

Many systems in today's vehicles are so-called cyber-physical systems (CPS), which use sensors and mechatronic parts to realize their functionality. These systems include assistant systems like adaptive cruise control, safety-critical systems like autonomous emergency braking, but also mechatronic systems like electronically supported steering. While some of the software development for a car is conducted in-house, other parts of the software as well as a majority of the hardware and mechanical components are developed and provided by suppliers. When the software development takes place in-house, mechatronic components, which are used to obtain data from the surroundings or to interact via actuators with the surroundings, are not available yet and hence, the software development would be partially based on assumptions about the behavior and data to be expected from such hardware and mechatronic components. The in-house development cannot be delayed until suppliers have delivered their finished systems as this would make it impossible to meet project deadlines.

Additionally, a standardization of hardware and software platforms like AUTOSAR [9] would have the potential of facilitating the integration and lower the number of platform-related assumptions. However, the automotive industry has a long tradition of optimizing on component price, since the hardware cost is still believed to dominate the total production costs and thus, packing and weight are valuable. Therefore, there is no breakthrough of component standardization yet, which could help to reduce the potential risk of relying on assumptions.

In order to achieve faster feedback as well as better utilizing the expertize of domain experts in-house VCG has decided to use the executable software modeling language Simulink. This permits one to simulate and test the code continuously and translates it automatically into C code whenever necessary. It also enables in-house engineers with domain expertise to implement solutions themselves.

Having the above in mind we wanted to improve on the way of developing mechatronic systems by combining elements from MDE with agile development methods. There are different methods and techniques called agile but they share a number of concepts, such as having working software early, rapid feedback reducing the time between decisions and seeing the consequences, taking advantage of faster and less formal communication and focusing on how to best utilize the skills and talents of the people in the organization [6,10]. The focus in this study has been on having working systems early in contrast to writing specifications and requirement documents up front, faster feedback by short iterations and more direct communication with stakeholders.

We are only looking at agile MDE within single ECUs. The input and output to the ECUs are exported from the SysTool and the design process on this level follows the overall system process common for the whole car project. However, even when being constrained to one ECU it is complicated to achieve agile development for mechatronic systems, since the connected physical systems are not always available and therefore require plant models as part of the virtual test environment. Without these plant models of mechanical parts one will not have an executable system early.

4 Research Questions and Methods

With the said-to-be standard tooling of Matlab Simulink in the automotive industry, the OEMs and their suppliers are successful adopters of model-driven engineering (MDE). Considering the aforementioned problem domain, the goal of this study is to investigate the challenges for MDE at an automotive OEM, when (un-)consciously depending on assumptions during in-house software development.

RQ-1: What are the causes that lead to assumptions in distributed mechatronic development and what are the consequences?

RQ-2: Does the combination of MDE and agile methods increase the knowledge in earlier phases of the project compared to a plan driven process?

RQ-3: What impact does faulty assumptions within the test environment have on the product and process?

Our study incorporates an exploratory study as a pre-study and two case studies. The exploratory study was conducted to address RQ-1, investigating the challenges regarding software, hardware, and mechanical assumptions by conducting interviews with developers, requirement engineers, testers, and architects. The design of the interview was semi-structured with open-ended questions, which allowed the interviewees to focus on topics as they arose during the discussion. The design and results for the exploratory study are described in Sec. 5.

The results from the semi-structured interviews were used to cluster topics to identify main challenges. From these topics, the research questions RQ-2 and RQ-3 were derived and two case studies were designed to follow upon on these topics. In the case studies, data was collected by observation and complementary semi-structured interviews. The results were validated by discussions with involved participants in the projects. The design and results for the case studies are described in Sec. 6.

The findings from the application of both methods are analyzed and discussed in Sec. 7.

5 Exploratory Study to Prepare the Case Studies

To address RQ-1, we conducted an exploratory study as a series of semi-structured expert interviews with open-ended topics. In this section, we report about the design and results of this exploratory study according to the guidelines of Shull et al. [19].

5.1 Design of the Semi-structured Interviews

The exploratory study consisted of eight semi-structured interviews with eight engineers. In general the interviewees had a background in electronics, physics, automation or mechanical engineering and a long-term experience in mechatronic and automotive development. The interviewees were sampled from different roles and working on different levels in the development project covering engineers working with software development and writing detailed requirements, testers on different levels, as well as developers working on the electronic architecture for the complete vehicle.

The following topics were open-ended discussed and recorded for post-processing:

- Background of the interviewee including role and responsibilities
- Type of involvement during the development process
- Experiences with assumptions:
    - Concrete examples for assumptions during the development
    - How wrong assumptions were identified and corrected
    - How corrected assumptions were documented afterwards

The interviews took on average 45 minutes and were very informative in terms of initiating reflection processes while discussing with the interviewees.

5.2 Results from the Semi-structured Interviews

Motivational factors for challenges stated from interviewees could be boiled down to having their roots in the sequential development process used on system level at VCG. To pass the project stage gates, decisions have to be taken before having the complete knowledge for deriving well-informed decisions. Thus, this gap of missing knowledge is filled in by assumptions as depicted by Fig. 2.

The earlier an engineer has to make a decision the higher is the risk of faulty assumptions that lead to unwanted side-effects of defects, which need to be fixed later. Furthermore, these assumptions can only be verified at the integration points when the different parts of the system are delivered. Since the integration often happens late in the development project, any serious issues that are discovered at this time are obviously costly and time-consuming to fix.

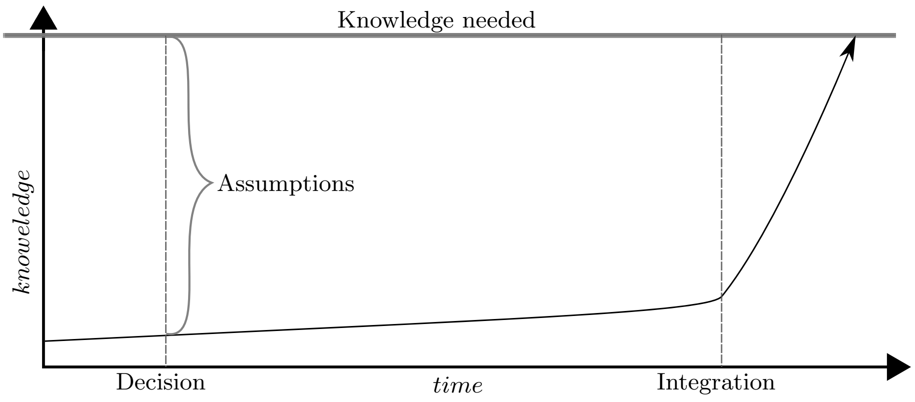

Fig. 2. A process that require early decisions regarding requirements, design or implementation details, before having any activities on building the knowledge needed will inevitably force its participants to make assumptions. These assumptions can be faulty, consequences of which will be visible first after the first real integration, usually leading to a huge increase in needed activities at the final stages of the project.

The main finding, confirmed with the involved interviewees, was that there is a clear need for earlier and faster feedback to the developers and designers of systems. This clearly shows that there is a need for agile methods to support delivering and integrating in increments early to build knowledge for making more informed decisions and getting early feedback on if their solution is heading in the right direction.

From the feedback of the interviewees in this exploratory study, we derived the research questions RQ-2 and RQ-3, which shall be followed up on with two subsequent case studies.

6 Case Studies

From our exploratory study, we derived the research questions RQ-2 and RQ-3, which are addressed by two case studies conducted at VCG. Here, we report about the case studies according to the guidelines from Runeson and Höst [18].

6.1 Case Study Design

The two cases are from different units in electric development department at VCG. The first case (Case 1) is from Electrical Propulsion Systems (EPS). EPS is the unit at VCG that develops the components for electrical and hybrid vehicles. The second case (Case 2) is from Central Electronic Module (CEM). CEM is responsible for an ECU within the car, which holds a collection of functionality that is central, such as locking and headlight control.

The software developers in the case studies have different backgrounds. A majority is educated in electrical engineering, some in computer science, and one has studied physics. The developers have between 2-10 years of automotive or embedded systems development experience. The software development part of Case 1 was about five man years, the one for Case 2 is two man years. Additionally, resources from other groups include test engineers to specify and execute component tests in MIL and HIL environment and computer aided engineering (CAE) engineers for constructing simulation models that are used in MIL and HIL testing.

6.2 Case 1: Clutch Control for Electric Drivetrain

The first case study was conducted at the group responsible for the electric part of the drivetrain in the hybrid car under development. Between the traditional and electrical drive train there is a clutch that is required to safely engage when the electric engine should be used. The study investigated the development of the software that controls the clutch.

Way of Working. The team used MDE techniques to develop their software. The wrappers for the behavior models were automatically generated from a system model and then complemented by the actual behavior implementation. A virtual environment was built to quickly iterate, test, and get feedback on their implementation. This virtual environment contains a number of models that simulate mechanical and software systems (plant models) for those components that have not been delivered from suppliers yet. The models needed for the simulation were developed by an in-house team specializing in CAE and verification. The basic functionality was implemented and tested against the plant models to avoid waiting for the first prototype vehicle with the real hardware and mechanical systems.

Results. In relation to research question RQ-2, we saw that there was a strong development of the plant models, e.g. of the clutch, from a simple model to a more detailed and realistically parameterized model. The work bench utilizing the plant models was used to continuously test the software models during the development work. Although the early executable models were not completely matching the final mechanics, they allowed flexible testing of the early controller software in a way which would have been very difficult or impossible without the virtual environment.

The test bench that was designed to run the controller software and plant models was optimized in collaboration between all involved parties, spanning over three groups. The methods were very different e.g. from Case 2, which demonstrates the importance of agility. Later in the project, when faulty assumptions were discovered, both controller code and plant model could be corrected locally with continuous testing on the developers PCs. One should note that both CAE team and control system developers were at that time used to the model design and had the understanding required to adjust it rapidly.

Considering RQ-3, in early car tests when the controller software was combined with a real clutch, there were indeed problems. The clutch did not behave as the CAE team had thought, since assumptions on the behavior of the clutch's teeth was proved wrong. This prevented the clutch from engaging properly. It took about one month of calendar time for identifying what the problem was, implementing a fix and get it deployed on the prototype vehicle. However, as soon as the problem was identified the resulting changes to fix the controller code were small and mainly parameter-related and the overall design was not affected.

6.3 Case 2: Active High Beam Headlight

The second case study investigated the way how the software for the active high beam headlights (AHBH) was developed. AHBH are used to let the headlight use the high beam but still not blind fellow motorists. This is done by identifying other vehicles using the cameras in the car and then mechanically obstruct the light to put the other vehicles in shade. This helps to keep the visibility at the sides of the road high at the same time as not blinding other road users.

Way of Working. This project was in the beginning a requirement engineering project following a classical waterfall process. The functional hardware, camera and image analysis system was developed, with well defined APIs and reasonably good specifications. Hence, detailed specifications existed of the sensor and actuator systems. The function was specified at high level, mainly using PowerPoint presentations describing customer use cases, but the control system, intended to be developed in a VCG ECU, was undefined. The solution was to engage an external expert in MDE to be part of the development team for a few months, creating a simulation environment, a demonstration/visualization environment and a first version of the control software, which was based on the supplied API and high level requirements. Later on the project was taken over by the original team, strengthened with a software modeling engineer. The second half of the project was carried out mainly in the laboratory, switching between software modeling and test in HIL rigs and on a real vehicle in a garage. Hence, plant models were used to little extent. The modeling environment was used for software modeling and visualization, and for simulations running with videos and recorded data from the supplier device and camera as input.

Results. With respect to RQ-2, much was learned using the available demonstration model by quickly trying out different solutions and algorithms. Before the simulation and demonstration environment was available the team struggled with getting approval from management to go ahead. The project was literally stuck. When the test environment with the demonstrator, including also the first version of the control software, was built it did not only help to start up the project work but also to communicate the ideas and to obtain approval.

Regarding RQ-3, the lack of a real, closed loop, plant model created assumptions of ideal actuators and sensors in the system. As a direct consequence, the early versions of the controller software was over-engineered in terms of implemented algorithms and code that the mechanical system could not support. This also resulted in an unnecessary complex architecture, judging from the design of the final version. The accuracy of the system actuators especially, regarding reaction speed, finally led to a significant simplification of the controller algorithm. The architecture and design of the first and more advanced controller software influenced also later versions, still showing signs of over-engineering.

Another assumption was caused by the visualization, which was created using simple light shape overlay on a 2D video with no 3D compensation or more advanced light rendering. Moreover, the edges of the overlay were unrealistically sharp, as the calculated image represented a plane only about five meters in front of the car. When the developers finally got into a car with the essential equipment installed, it was obvious that the spread of the light was higher than they had assumed by basing it on their visualization and the solution had to be rethought. Regarding this assumption one should also note that it was not unveiled during laboratory tests, although the real equipment was present, since the beam was projected on a screen inside the laboratory.

6.4 Interpretation of Results

In both of the case studies we found that the use of virtual test environments significantly improved the knowledge in early stages of the projects. The test environments enabled reliable tests of prototypes and assumptions about externally developed technology.

We found that the development team in Case 2 had little progress at all before gaining access to a virtual environment for testing. Looking at Case 1 one could see that having virtual testing available from the start enabled the software models to grow faster and more organically than if they had followed the overall system process where decisions had to be made upfront, such as in the start of Case 2. The software model in Case 1 also showed less signs of over-engineering compared to Case 2. Furthermore, in Case 1, more effort was spent on a realistic plant model, compared to Case 2 where the visualization was prioritized. Observing the progress, we note more late issues caused by faulty assumptions related to hardware and mechanics in Case 2, which were also re-worked in longer time. Finally, Case 2 suffered from late issues, found in traffic situations, which theoretically could have been discovered much earlier using more advanced simulation environments.

By integrating and executing the software models in a virtual environment the developers were forced to address holes in their knowledge. Hence correct knowledge was gained although models or virtual environments included assumptions. This gain of knowledge is visualized in Fig. 3. Faulty assumptions were indeed discovered in vehicle tests in both use cases. Nevertheless, the time to fix these issues was shortened by a realistic simulation environment in Case 1.

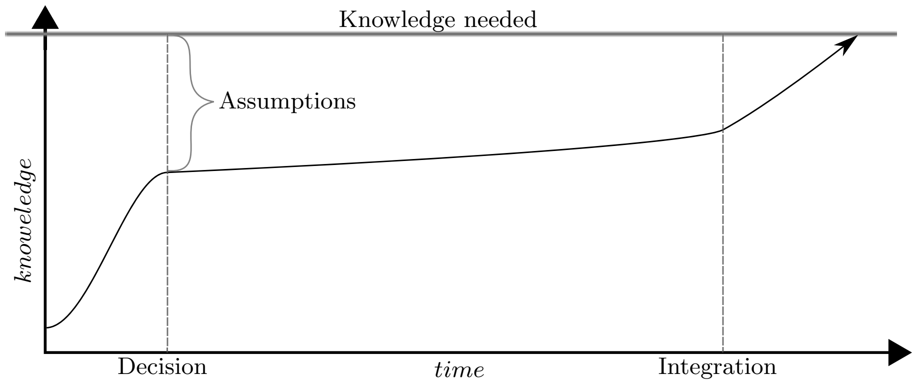

Fig. 3. By being able to integrate and execute tests more frequently using plant models and virtual environments the developers can build and validate knowledge also early in the project. Although numerous assumptions still have to be made, since the mechanical system or other parts of the mechatronic system are unknown to the developers, the ability to conduct simulations will increase the knowledge. At the first real integration there will obviously still be a risk for faulty assumptions, but the knowledge gap is significantly reduced.

The plant models in Case 1 was constructed from specifications, intended for the external supplier developing the equipment, and suffered both from smaller inconsistencies and incompleteness. Some simplifications were also made, forced by limited understanding of the mechanical behavior of the clutch. Furthermore, the plant model developer and the function developers worked in separate groups. Hence, the team developing plant models lacked some contextual background for the intended use case of the plant models as well as direct contact with the supplier. However, the benefit of having software tested with plant models before deploying in real vehicles greatly outweighed losses due to an incomplete process or problems with communication.

Finally one should note that some architecture decisions, e.g. regarding communication between ECUs, still had to be done upfront. The teams expressed frustration over this but a solution would require changing the overall systems process as we reported on in our previous work [8].

7 Analysis and Discussion

The case studies unveiled that applying MDE in combination with agile helped to address the challenge of shortened development cycles. Together with a virtual test environment, it was possible to relax the dependency on mechanical components provided by suppliers later during the development. This would not be possible in mechatronic development otherwise.

Furthermore, this environment enabled short iterations to experiment with different solutions and get feedback. This helps to gain knowledge earlier in the project and to reduce the number of assumptions that had to be made.

However, it turned out that is important to understand the weaknesses of the plant model. Any issues in the plant model, for example caused by faulty assumptions, heavily impact the software models that are tested against them. This can lead to issues in the software that are not visible until the integration phase with the real components later. Thus, the software is believed to be more mature than it actually is.

We have also seen in Case 2 that assumptions on the capabilities of the mechanical systems that exceed reality can result in a more complex and resource-demanding implementation than the mechanical components can support. This in turn could result in that the requirements for the ECU-hardware are higher than needed and a more expensive ECU is bought when a cheaper one could have been used. Because of the number of cars produced, even small increases in cost for a component have an impact in the profits of the OEM. E.g. if an ECU costs 10 USD more than a less expensive one that would suffice, and 100 000 cars are produced, the profit shrinks with 1 000 000 USD just caused by one component.

Delaying software development until the mechanical systems are delivered is not an option, as that would delay the whole project. But there are some potential methods that could possibly decrease the number of assumptions in the virtual testing environment.

One solution is that the suppliers deliver plant models of their mechanical systems. This should in theory give the OEM more accurate plant models to base their development on. However, for a number of reasons not all suppliers are willing to share their models. This could often be traced back to a way of protecting their IP. The other one is that, in the cases where suppliers do deliver plant models, they only provide them as black boxes. This makes it impossible for the OEM to verify that the model is actually working correctly. There have been cases where a model delivered from a manufacturer of a mechanical system was shown to be incorrect when compared to the real component and without insight into a model it is hard for the OEM to investigate where the issue lies, in the software model or the plant model. It also prevents the OEM to see if the plant model is fit for the testing. E.g. if the plant model provided does not simulate the teeth within the clutch it would not be possible to verify that the algorithm within the software model could calculate the correct offset needed for successfully engaging the clutch.

Alternatively, plant models could be built in-house but verified by the supplier. Most suppliers of mechanical systems have environments where they could execute such models. This would have the potential of avoiding structural and fundamental mistakes in the plant models.

Even though using plant models and MIL-testing for rapid prototyping seems to be promising, there are also aspects that need to be considered. It is time-consuming to construct and maintain the plant models and integrate them in the MIL environment; furthermore, developers or testers need to develop the test cases for the MIL-testing. Here, the task seems to be rather less attractive because the implementation will be tested on the prototypical real vehicle later anyway; however, the gains of having an early validation in a virtual test environment needs to be underlined properly. This is also confirmed by the participants in our studies who see otherwise the risk of not meeting deadlines in the demand of shorter development cycles.

As the groups studied in the case studies have not had in-house software development before, there was no previous software development process that needed to be adjusted to allow for MDE and agile. Furthermore, the toolchain could also be built from the ground up to support agility.

To address threats to validity in our studies, we follow the guidelines from Wohlin et al. and Runeson and Höst [18, 22, 23].

Construct Validity: The subjects in our exploratory study were experts in how to build embedded software for automotive systems. We validated our result by discussing our findings with the interviewees. The case studies were conducted on two projects where the resulting artifacts of the projects were components that will be used in cars delivered to customers at VCG. By having two studies, effects, which might be present only in one case could be reduced.

Internal Validity: The main objective for the studied cases was primarily on delivering a running system with high quality in the end. Effects that would favor MDE or agile methods could be reduced.

External Validity: There are a lot of companies where hardware, mechanical parts, and software need to interplay well like trucks, radars, and pumps etc. For many of these companies our findings can be of interest and stimulate similar studies.

Reliability: The analysis was conducted by the authors. Our findings and conclusions were confirmed with the participants of the studies in feedback and discussion rounds to reduce the risk of dependency on the conducting researchers.

8 Related Work

Matinnejad [16] has done a review on the existing agile model-driven development (AMDD) processes. They think that an intelligent compromise can be done to gain the advantages from the two different approaches to software development. We have seen that there might not be such a big contradiction between MDE and agile. Instead they two complement each other well, especially in a distributed mechatronic domain where it enables to develop and test against yet to be delivered components. However, we agree that there are benefits from combining the two and that it is an area that needs more interest and research.

Zhang et al. [24] applied agile MDD on a development project for real-time telecommunication. The project was considered successful and was delivered on time. MDD was only applied to two of five components, the other three components were "hand-coded". They saw a threefold increase in productivity in terms of lines of code compared to hand-coding and that the defect density was lower in code generated from models. They also saw that there was a steep learning curve for the organization to adopt to MDD and agile methods. Therefore they thought that short-term benefits were not likely but the long-term benefits still made it worthwhile. We however observed that the engineers at VCG instead could get started faster with software development due to the fact that they were allowed to work in a tool that was close to their domain and that was familiar from their engineering education.

Kulkarni et al. [11] reported on their experience with agile MDD in their development projects of business systems. They found that some activities were not suitable to be conducted in short sprints and that code generation and transformations can be a bottleneck. They solved this by introducing meta-sprints that do not necessarily have to deliver working software but can comprise for example design documents or an evaluation of a set of design strategies. In our studies and experience from VCG a major point in the success of MDE is that they use a product for their software modeling that is mature and where the C-code generation is already tested and stable.

Auweraer et al. [2] have also identified the need for virtual testing to enable concurrent development of mechanical components and the software for controlling it. However, their motivation is to accelerate the design process and not to enable software development at the OEM while the mechanical component is developed somewhere else. Therefore they do not discuss the problem of the OEM having a gap of knowledge of the mechanical system, or software processes.

Virtual test environments are proved to be successful especially for highly complex cyber-physical systems like self-driving cars as outlined by Berger, 2010 [3] and Berger and Rumpe 2012 [4]. These systems will play an increasingly important role in the future with more and more diverse product families and shorter development cycles. Faster and precise feedback of the quality for implementation models when using these simulation-based virtual test environments will be the competitive advantage for automotive OEMs.

Research has been done on different ways of modeling assumptions to verify them [12, 14, 20]. These approaches can be divided into two classes, formal or semi-formal [15]. Formal approaches, such as Tirumala 2006 [20] try to capture assumptions in a model and formalize their attributes, so that these can be verified automatically. Semi-formal approaches, such as Lewis 2004 [14] and Lago 2005 [12] capture the assumptions in a model but their attributes is free text and could therefore not be checked by a machine. Such approaches were initially considered, however we found them not feasible. Firstly, without having the properties of the real system such a model will in itself contain assumptions and the result of a verification could not be trusted. Secondly, it would add a new tool and language for the engineers to not only use but also to learn. With them already being on a tight schedule it was deemed impossible to introduce such tools. It could be a good exercise to make the developers more aware of the assumptions they are making, but the ones that will cause problems are the ones that are not explicitly thought about. Experience from practice shows that assumptions will always be made and capturing them all seems impossible.

9 Conclusion and Future Work

In our case studies we have seen that MDE and agile methods can successfully be combined to develop software for complex mechatronic systems. Together with virtual test environments, it enables an organization to start developing and test their software before they receive deliveries of mechanical and software components from their suppliers. We saw that it was possible for the developers to quickly iterate their implementation and get feedback. This made the developers aware of holes in their knowledge that needed to be addressed, and it allowed for exploring solutions and build knowledge before writing the final requirements and committing to a design.

It was obvious that assumptions made when constructing the models for the simulated environment had great impact on the software models constructed. Faulty assumptions in the simulation of yet to be delivered components caused issues in the software. These issues were discovered at integration but feedback from involved engineers let us conclude that the advantages with fast feedback, building knowledge, and a more mature solution earlier outweigh the fact that not all issues are caught in the virtual test environment.

For the future, we aim for automating further parts of the testing and reach continuous integration and deployment. Both in a model based virtual test environment but also for hardware-in-the-loop testing and finally to deploy the new software to prototypical vehicles.

For the problems caused by faulty plant models there are two directions to further explore with better collaboration between suppliers and OEMs. One would be to get access to their white box models of their mechanical components as part of the business agreement. The other potential way is to keep the construction of plant models in-house but have the supplier on-site to verify them.

References

1. Albayrak, O., Kurtoglu, H., Biaki, M.: Incomplete software requirements and assumptions made by software engineers. In: Asia-Pacific Software Engineering Conference, APSEC 2009, pp. 333–339. IEEE (December 2009)
2. Van der Auweraer, H., Anthonis, J., Bruyne, S.D., Leuridan, J.: Virtual engineering at work: The challenges for designing mechatronic products. Engineering with Computers 29(3), 389–408 (2013)
3. Berger, C.: Automating Acceptance Tests for Sensor-and Actuator-based Systems on the Example of Autonomous Vehicles. Citeseer (2010)
4. Berger, C., Rumpe, B.: Engineering autonomous driving software. In: Experience from the DARPA Urban Challenge, pp. 243–271. Springer (2012)
5. Broy, M.: Challenges in automotive software engineering. In: Proceedings of the 28th International Conference on Software Engineering, ICSE 2006, pp. 33–42. ACM, New York (2006)
6. Cockburn, A., Highsmith, J.: Agile software development, the people factor. Computer 34(11), 131–133 (2001)
7. Ebert, C., Jones, C.: Embedded software: Facts, figures, and future. IEEE Computer 42(4), 42–52 (2009)
8. Eliasson, U., Burden, H.: Extending agile practices in automotive MDE. In: XM 2013 Extreme Modeling Workshop, p. 11 (2013)
9. Fürst, S., Mössinger, J., Bunzel, S., Weber, T., Kirschke-Biller, F., Heitkämper, P., Kinkelin, G., Nishikawa, K., Lange, K.: AUTOSARA worldwide standard is on the road. In: 14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden (2009)
10. Highsmith, J., Cockburn, A.: Agile software development: the business of innovation. Computer 34(9), 120–127 (2001)
11. Kulkarni, V., Barat, S., Ramteerthkar, U.: Early experience with agile methodology in a model-driven approach. In: Whittle, J., Clark, T., Kühne, T. (eds.) MODELS 2011. LNCS, vol. 6981, pp. 578–590. Springer, Heidelberg (2011)
12. Lago, P., van Vliet, H.: Explicit assumptions enrich architectural models. In: Proceedings of the 27th International Conference on Software Engineering, ICSE 2005, pp. 206–214. ACM, New York (2005)
13. Leen, G., Heffernan, D.: Expanding automotive electronic systems. Computer 35(1), 88–93 (2002)
14. Lewis, G., Mahatham, T., Wrage, L.: Assumptions management in software development. Software Engineering Institute (August 2004)
15. Mamun, M.A.A., Hansson, J.: Review and challenges of assumptions in software development. In: The Second Analytic Virtual Integration of Cyber-Physical Systems Workshop (2011)
16. Matinnejad, R.: Agile model driven development: An intelligent compromise. In: 2011 9th International Conference on Software Engineering Research, Management and Applications (SERA), pp. 197–202 (August 2011)
17. Miranskyy, A., Madhavji, N., Davison, M., Reesor, M.: Modelling assumptions and requirements in the context of project risk. In: Proceedings of the 13th IEEE International Conference on Requirements Engineering, pp. 471–472. IEEE (September 2005)
18. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering 14, 131–164 (2008)

19. Shull, F., Singer, J., Sjøberg, D.I.: Guide to advanced empirical software engineering. Springer (2008)
20. Tirumala, A.S.: An Assumptions Management Framework for Systems Software. Ph.D., University of Illinois at Urbana-Champaign, United States – Illinois (2006)
21. Uchitel, S., Yankelevich, D.: Enhancing architectural mismatch detection with assumptions. In: Seventh IEEE International Conference and Workshop on the Proceedings of the Engineering of Computer Based Systems, ECBS 2000, pp. 138–146. IEEE (2000)
22. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in software engineering. Springer (2012)
23. Yin, R.K.: Case study research: Design and methods, vol. 5. Sage (2009)
24. Zhang, Y., Patel, S.: Agile model-driven development in practice. IEEE Software 28(2), 84–91 (2011)