# Extending Agile Practices in Automotive MDE

Ulf Eliasson[1] and Håkan Burden[2]

[1] Volvo Car Corporation, Sweden
ulf.eliasson@volvocars.com
[2] University of Gothenburg, Sweden
burden@cse.gu.se

**Abstract.** The size and functionality of the software in a modern car increases with each new generation. To stay competitive automotive manufactures must deliver new and better features to their customers at the same speed or faster than their competitors. A traditional waterfall process is not suitable for this speed challenge - a more agile way of working is desirable. By introducing MDE we have seen how individual teams at Volvo Cars adopt agile practices, resulting in tensions while the organization at large still uses a waterfall process. In an exploratory case study we interviewed 17 engineers to better understand how agile practices can be extended beyond individual teams. Where the tensions have their source in the technical specification of the software components and their interfaces, it turns out that it is company culture and mindsets that are the main hurdles to overcome.

**Keywords:** Interface Specification, Tailoring, Exploratory Case Study

## 1 Introduction

The size and functionality of the software in a modern car increases with each new generation [1] and the software can be executing on in the order of 100 Electronic Control Units (ECUs) spread out in the car. This causes software development to take an increasing part of the total R&D budget for new car models [2]. Automotive manufacturers are traditionally mechanical and hardware oriented companies. The software processes often resemble the traditional waterfall since that works for mechanical development - but it might not necessary be the best for developing software. In a world that moves faster and faster it is crucial to push new features out faster than the competitors. One way that automotive manufactures can speed up their software process is to develop more software in-house, making it possible to iterate their software and introduce new features faster than when ordering the same from a supplier. By introducing MDE we have seen that the domain experts are directly involved in the software implementation and how lead times become shorter.

## 2 Automotive Software Development at VCC

The system development process at Volvo Car Corporation (VCC) is a waterfall process with a number of integration points throughout where artifacts should

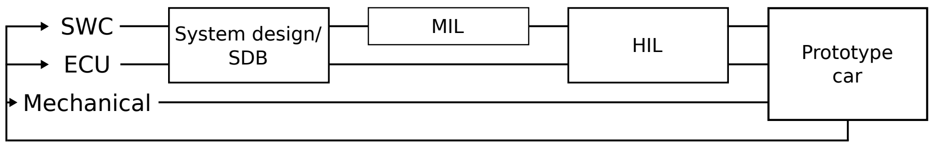

**Fig. 1.** Overall process view.

be delivered. There are three tracks of parallel development - the software components (SWC), the hardware (known as electronic control units or ECU) and the mechanical parts. Parts of the software is developed in-house while the rest of the software, and most hardware and mechanical systems are developed by sub-contractors. Each iteration of the process has a number of integration steps, as shown in Fig. 1. The system design and signal database (SDB) is the first integration step where the software and its interfaces are designed. Model-In-the-Loop (MIL) is where the implementation Simulink-models are integrated together with other models, either single components together with test models for testing, or in complete MIL with models of different components from different teams. After code generation from the implementation models they are integrated in Hardware-In-the-Loop (HIL) executing on hardware but with the environment around it simulated with models executing in real-time. Finally everything is integrated in a prototype car.

Early on, all the requirements and the system design for the next iteration is captured as a model inside a custom-made tool, referred to as SysTool. The model contains, among many things, software components, their responsibilities as requirements, the software components deployment on ECUs, and the communication between them as ports and signals. The signals are basically periodic data that is sent, and they are described down to the byte size of their data types. The signal definition also includes timing requirements, such as frequency and maximum latency. At a certain point in time the 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 used for two things. Firstly, signals, ports and their connections and timing requirements are used to implement the SDB. The SDB is used by the software on the ECUs and the internal network nodes to schedule all the signals passing through the networks in the car. The packing and scheduling of the signals is done manually. It takes 9 weeks from a freeze until the SDB is delivered and ready to be used. 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 filled in with functionality by the developers as specified by textual requirements. If the system model changes the Simulink models are updated to reflect this by a click of a button and any implementation already done is kept intact. The implementation must be updated manually if it is dependent on signals that are removed or changed.

The executable Simulink models are tested together with plant models. The plant models represent the environment surrounding the ECUs, including electrical, physical and mechanical systems. This enables the developer to get instant feedback by running and testing the models s/he developed in isolation on their own PCs. The models are also integrated in a virtual car MIL environment where all models developed in-house are integrated and executed. The time frame for getting feedback after delivering a model to the virtual car is counted in days. This gives the developers a possibility to do requirements and design exploration and validation on component level. However, sub-contractors developing software do not normally provide models of their software meaning that where suppliers are developing functionality there are holes in the MIL environment. These holes are filled in by models describing how to develop their software as they see fit, as long as they can fit their work in to the larger overall process and deliver in time. Therefore development within the Simulink models for one ECU can and is conducted agile. However, as soon as there is a need to extend or modify the SDB the developers need to adapt to the overall waterfall process.

The suppliers deliver their software as binaries. Code is generated from the in-house models, built to binary and then the two are linked together. The finished software is loaded on hardware and is then tested on HIL rigs, see Fig 1. Because the supplier only delivers binaries this is the first time that in-house and supplier developed software can be integrated and validated together. Later, software and hardware is integrated with the mechanical systems in a complete prototype vehicle and tested. This is the first time that the whole system is tested together. Each full iteration in Fig 1 has a deadline referred to as a gate and the time between the gates is referred to as E-series.

The use of MDE in the teams makes it possible to break free from the suppliers making it possible to execute and test the in-house software without having to wait for the hardware and software from the suppliers. This enables the team to be agile and work in short iterations.

## 3 Method

Given that agile practices have shown to be possible at the level of individual teams, we wanted to answer the question:

**RQ:** Which are the challenges and possibilities for a more agile software development process on a system level?

The question was answered by two exploratory case studies [3]. The main data was collected through two parallel interview series, conducted by the two authors independently of each other. The first set of interviews was launched internally by Volvo Cars in order to identify factors that could improve the existing process. The second study was initiated as a collaboration between Volvo Cars and academia to evaluate and improve the ongoing transition into Model-Driven Engineering, MDE. Both interview sets included eight interviewees, without an overlap between interviewees. In general the interviewees had a background in electronics, physics, automation or mechanical engineering with a limited training in software development from VCC. All interviews were conducted at VCC in Gothenburg, Sweden, and complemented by active participation by one of the authors and on-site observations by the other.

The study focusing on process conducted the first interview in May 2012, the study on MDE started in January 2013. Both sets of interviews were finished by April 2013. Since the interviews were conducted over a longer time frame, collecting and analyzing the data was done in an incremental fashion [3]. From the first interviews of each study a preliminary analysis emerged, identifying themes and concepts that the engineers found challenging in the current combination of process and model-driven implementation. Using a semi-structured format allowed the interviewers to explore new topics as they arose but also to see if spiring hypotheses could be confirmed [4,5].

The two studies were aligned in May 2013 by a manager who recognized that both lines of inquiry had come to the same conclusion independently of each other, despite the fact that one study was an internal effort to improve the existing process and the other study was an academic collaboration investigating MDE. After the analysis of the interviews had been completed (see section 4) a system architect involved in the scheduling of the SDB was interviewed to reflect on the outcome of the analysis (presented in section 5).

## 4 Challenges for Agile MDE Practices

From our interviews we discovered a number of issues that have their roots in the waterfall process used on a system level. The most prominent issue, repeated by different interviewees, concerned the SDB in the SysTool. By freezing requirements and system design such a long time before delivery developers are forced to take premature decisions on what data they need to receive or send and where. Because the signals are the interfaces between different components, developed by different teams or sub-contractors, any negative impacts caused by premature decisions are not discovered until late in integration and therefore expensive to change.

The practice of freezing the interface implies that the engineers have to specify the interface they need before they fully understand the internal behaviour of the component being developed. This means that the interfaces are defined based on the assumptions the developer have at that time and subsequently there are signals that will never be used but still have a share of the limited capacity. This causes two problems. First extra signals need to be scheduled on the network, wasting time for the SDB group in their work as well as causing extra congestion on the network. It also makes it difficult for a developer on an ECU to know which signals are used or not used.

*Q: So do you overload the interface? Throw in a signal just in case?*

*A: Yes, that is what we do. At least I do it [. . . ] and then you end up with the problem knowing which signal it is you should actually use.*

As with the spare signals above, developers add extra data-elements to their signals they create to future-proof them. This causes the same problems as with the spare signals but is also harder to redeem because one can't just remove a signal, the signal needs to be modified. It is also harder to check if a data element is used or not compared to seeing if a whole signal is used.

*A: Also an old problem we have here at Volvo is that when someone wants to add a new signal, they know it will be hard to change the signal later. So to be prepared they add a few extra data bits to the signal, just in case.*

Developers that figure they need to send or receive some data that they did not think about before the freeze do not want to halt their implementation until the next freeze, instead they use existing signals in creative ways. This includes using signals and data-elements in ways that are not described in the requirements making them behave differently than intended. When other groups depend on these signals misinterpretations occur which causes problems.

*A: But we have a text document that's about 300 or 400 pages in total if you take all the documents. And that hasn't been updated for a couple of years. So this is wrong. This document is not correct.*

Since the textual documentation is inconsistent with the implementation and the interface is overloaded the engineers start to mistrust the artifacts that are supposed to support them in their development. As a consequence one of the other interviewees had developed a work-around for handling that the interface specification was constantly outdated. The solution is to sieve through a second document after the information that concerns the interface being developed and translate that information into a new, temporary, specification.

*A: We have in our requirements a list of signals used in the requirement. Now that list is seldom updated. It's hardly ever, so they're always out of date. So I don't actually read them anymore. I just go in through the specific sub-requirements and I read what is asked for my functionality. This is asked. What do I need? I need this and this. So, yeah, so I do it manually, I guess.*

The reoccurring theme behind these issues are that developers are forced to make unfounded assumptions about what the SDB will or needs to contain and how the signals in it will be used. Also a shared view between the developers was that it is the development of mechanical and hardware systems and the MDE tools that forced the use of a waterfall process. The issues caused by these assumptions are not found until late in the process with a considerable cost in both time and money as the result.

## 5 Possibilities for Extending Agile MDE Practices

Based on the results from previous interviews we interviewed an architect responsible for tool and process development at VCC and brought up the identified challenges. He did not see the technical problems of the SysTool and its models to be the main obstacle to overcome, rather it is the culture and the way of thinking in the organization that needs to change.

*Q: Why do we have these freezes?*

*A: We have a traditional gated process, and then you have freezes and gates for everything, period. [. . . ] If we think waterfall it is very logical that we have these freezes, that is what we have to rethink completely. What I'm thinking is that we need to change our system definition process so that it is agile and we can make changes whenever we want [. . . ] and then we can drive this in different speed with different suppliers but it shouldn't be our process that is stopping us.*

He also did not think that the hardware development done in parallel requires a waterfall system development process with gates. However, working with suppliers is one reason for having the gated system development process.

*Q: How much does it have with working against suppliers and developing hardware?*

*A: Yes, that is part of the answer. The connection to hardware I do not see as very strong, because the hardware development is not really in the same cycles anyway. Of course, we have hardware upgrades and when it affects the software then it has that connection, if there is some sensor or something that is replaced, but many of these problems are about changes to signals on the network and that is not connected to hardware at all really, at least not at that level. So the hardware is not a big factor. But supplier interaction is of course a factor, because we have a way of working with the supplier where we tell them that on this day we will send the specification and this day you will deliver, and there is a number of weeks between when we send the requirements and they should deliver and then we need a freeze so that we have something to send.*

The time between freezing the database until there is a finished implementation of that version, is 9 weeks. Because the developers know that this is their last chance for a while to change the signals they wait until the last minute to put any requests for new or changing signals as close to the freeze as possible. Obviously this results in a lot of change requests to be processed at the same time. It is not until after the freeze that signals and other changes are checked for compatibility and consistency, which might result in a lot of work to make the system design model consistent again.

*A: If we say that at this point in time you should submit all your change requests then you get all of these the Friday before instead of getting them more continuously and then you have these weeks of job ahead of you. [. . . ] We need to start thinking about the SDB and frame packing as a product, like any node. So the nodes deliver their software and the SDB delivers the frame packing as a component that integrates with the other stuff. So instead of having a process where the frame packing and everything needs to be finished and done before you can start making a node it should be something that you integrate with the other stuff. [. . . ] Integration is something that already today is happening continuously. It is not a specific day we integrate it is something you try during the E-series and in the end you make it work and then it is time for the next one. [. . . ] At the gate between E-series we do a refactoring of the SDB, clean it up and pack it. What I think is that, we should continuously allow ourselves to add signals, and also allow each other to add redundant signals. If one signal is wrong we do not remove that signal, because the components are expecting to get this signal.*

*But we can all the time allow ourselves to add signals and therefore we can get double, triple or quadruple signals when we find our way forward. This is roughly as you work with product lines, you allow yourself to over-specify interfaces and so on.*

In this way the SDB is tidied up at each gate instead of defined at the system design phase within each iteration.

*A: Working like this we can basically end up at releasing a SDB every day throughout the series and as long as a test or E-series environment is alive we can release a SDB daily where we can add new signals to test. Then we can allow ourself to deliver in what frequency we want.*

*Q: It sounds like most things that need to change are soft issues, are there no technical obstacles for this?*

*A: No, not really. There is a need for more support in SysTool to make sure that additions that you make are backward compatible. Today this is mostly a manual process. So you need to build in locks so, for this E-series you can only introduce backward compatible changes so that we all the time can export a correct export. And a lot of the stuff that we manually need to clean up is checked automatically. So there are some small tool changes. But this is not the big thing, it is how we think and how we work.*

The system model doesn't have to be executable, a non-executable model is enough for the checks that are needed. Also, because the system model is used to generate shells that are filled with executable Simulink models, developers might not see a need or purpose with having more executable models.

*A: We will still have these E-series where there will be some refactoring, and to release such an E-series will still take two to three weeks to pack, so it might not be nine weeks but it might be four to five weeks before an E-series release that you need to say what you want in that release. However, the difference from now is that this is not your last chance to get things in it, it is just what will be in it from day one in this series. Then after this you will be able to get things into it as long as it is living.*

Because the software development cycles are not bound to the hardware or mechanical development there is no need to follow a similar waterfall process. Also, a more agile system development process would not force the suppliers to be more agile, instead it would make it possible for the teams and their suppliers to work out the best way of working between them. Therefore we can use a more agile process, as already practiced by some of the ECU teams, on a system level. To enable this transition the SysTool needs to support static consistency checks, as proposed by [6], to give the developers and architects confidence in that the changes they make are backward compatible and will not break the integration.

## 6 Related work

Pernstål [7] has conducted a literature study of lean and agile practices in large-scale software development, such as in the automotive industry, concluding that there is a lack of empirical research within this area.

Eklund and Bosch [8] have developed a method and a set of measures to introduce agile software development in mass-produced embedded systems development, including automotive development. They have discovered that part of introducing agile methods is to gain acceptance in the organization for gradual growth and polishing of requirements instead of using a waterfall approach. They also say that it is possible to have agile software development even though the product as a whole is driven by a plan-driven process.

Kuhn et al. [9] and Arnanda et al. [10] have investigated MDE at General Motors. However, they have not looked at how MDE could change the process and help overcome some of the problems with the traditional process for developing automotive systems.

## 7 Conclusion and Future Work

The interviewees often thought that the agile challenges were related to MDE or the used tools. However, during the interview with one of the responsible architects for the tools and processes he identified them as caused by the process used on a system level. MDE would be the enabler for a more agile way of working as it provides the teams with a way of testing and iterate their design without having to wait for supplier software or hardware. To get the suppliers on board in such a way of working will take time. But a more agile system development process would not force the teams and suppliers to change their current way of working, but it enables teams and suppliers to agree on a way of working that suits them best instead of forcing them to fit in to the waterfall development process.

We have during our research discovered that agile MDE can be beneficial for automotive development. Using a language close to the domain, such as Simulink, enables engineers trained in the specific domain to work in an environment they recognize and can express their solution in. A model based environment where one can do physical modeling also enables the developers to quickly test their solutions on their own PCs. These are enablers for a more agile development process than is currently the norm in the automotive industry.

We have also discovered that the hardware or mechanical development does not force the software development to follow a waterfall process. The software development is so independent it could have its own process on top of hardware and mechanical development. A more agile software development process would also make it easier to adapt to changes in hardware or mechanical systems.

The interaction with sub-contractors is one of the obstacles that needs to be bridged before agile can happen on all levels. However, a more agile software process on the system level would permit the individual teams and sub-contractors to interact in any way they would think is best instead of forcing them to follow and fit it in to the waterfall process. Some systems are naturally less appropriate to develop in a completely agile way, such as brakes, and others are so stable and well known that there is little benefit or need for agile development. But having the software process on the system level agile doesn't mean that these domains have to be developed agile, the sub-processes can still be as strict as they need.

The natural thing would be to try to spread the agile MIL environment in all directions and tailor the process [11] to utilize the possibilities of the tools. A first step to enable such a transition would be to allow for faster iterations of the SDB and extending the SysTool to do static consistency checks.

For the future, we plan to implement some of the proposed changes in section 5 to the process at parts of VCC and evaluate them, including looking at how external actors can be integrated in a more agile way of working. Changing a large organization will take time [12]. By starting bottom up we hope that the acceptance for change will be easier to achieve in respect to in-house developers [8] but also for building trust with sub-contractors [13].

## References

1. Ebert, C., Jones, C.: Embedded software: Facts, figures, and future. IEEE Computer **42** (2009) 42–52
2. Broy, M.: Challenges in automotive software engineering. In: Proceedings of the 28th international conference on Software engineering. ICSE '06, New York, NY, USA, ACM (2006) 33–42
3. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering **14** (2008) 131–164
4. Robson, C.: Real World Research. 2nd edn. Regional Surveys of the World Series. Blackwell Publishers (2002)
5. Seaman, C.: Qualitative methods in empirical studies of software engineering. IEEE Transactions on Software Engineering **25** (1999) 557–572
6. Rumpe, B.: Agile modeling with the UML. In Wirsing, M., Knapp, A., Balsamo, S., eds.: Radical Innovations of Software and Systems Engineering in the Future. Number 2941 in Lecture Notes in Computer Science. Springer Berlin Heidelberg (2004) 297–309
7. Pernstål, J.: Towards Managing the Interaction between Manufacturing and Development Organizations in Automotive Software Development. PhD thesis, Chalmers University of Technology, Gothenburg, Sweden (2013)
8. Eklund, U., Bosch, J.: Applying agile development in mass-produced embedded systems. In Wohlin, C., ed.: Agile Processes in Software Engineering and Extreme Programming. Number 111 in Lecture Notes in Business Information Processing. Springer Berlin Heidelberg (2012) 31–46
9. Kuhn, A., Murphy, G.C., Thompson, C.A.: An exploratory study of forces and frictions affecting large-scale model-driven development. In France, R.B., Kazmeier, J., Breu, R., Atkinson, C., eds.: Model Driven Engineering Languages and Systems. Number 7590 in Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 352–367
10. Aranda, J., Damian, D., Borici, A.: Transition to model-driven engineering: what is revolutionary, what remains the same? In: Proceedings of the 15th international conference on Model Driven Engineering Languages and Systems. MODELS'12, Berlin, Heidelberg, Springer-Verlag (2012) 692–708
11. Whittle, J., Hutchinson, J., Rouncefield, M., Burden, H., Heldal, R.: Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem? In:

MODELS 2013, 16th International Conference on Model Driven Engineering Languages and Systems, Miami, USA (2013)

12. Aaen, I., Börjesson, A., Mathiassen, L.: SPI agility: How to navigate improvement projects. Software Process: Improvement and Practice **12** (2007) 267–281
13. Christopher, M.: The agile supply chain: Competing in volatile markets. Industrial Marketing Management **29** (2000) 37–44