Comparing and Contrasting Model-Driven Engineering at Three Large Companies

Håkan Burden
Computer Science and Engineering
University of Gothenburg
Gothenburg, Sweden
burden@cse.gu.se

Rogardt Heldal
Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden
heldal@chalmers.se

Jon Whittle
School of Computing and Communications
Lancaster University
Lancaster, UK
j.n.whittle@lancaster.ac.uk

ABSTRACT

Hutchinson et al. recently carried out an interview-based study of how model-driven engineering is practiced in 17 companies. Their results are revealing: they found that successful MDE companies develop domain-specific languages; are motivated by a clear business case; and are committed at all levels of the organization. Whilst the results are useful, the study is a very broad one, with one or two interviewees per company. This paper supplements Hutchinson's study by focusing on three large companies, but studying the companies in depth through a number of interviews at each. These three companies have a number of things in common – they are all large companies applying MDE and are all undergoing a parallel transition to agile methods. The paper reflects on similarities and differences in the way these companies apply model-driven engineering, but also discusses how findings from this study either validate or refute those of Hutchinson et al.

Keywords

Case study, Agile methods, Domain-specific languages

1. INTRODUCTION

Model-driven engineering (MDE) focuses on the use of high-level abstractions, or models, as primary artefacts for understanding, analyzing and developing software [10]. When applied successfully, MDE can lead to an increase in productivity, better quality software, and more effective reuse of software components. Equally, however, there are many factors – technical, social and organizational – that can cause MDE to hinder a software development effort [18]. In prior work, Hutchinson et al. [7, 8, 9, 18] carried out an extensive study of MDE practice and provided detailed insights as to why some companies adopt MDE successfully whereas others fail.

Hutchinson et al.'s study aimed at providing broad coverage of MDE practice: it surveyed 450 MDE practitioners and interviewed 22 MDE professionals from 17 different companies across 9 industrial sectors. The results are revealing; however, such a broad study is also necessarily shallow. The data is based on a survey of mainly closed questions and on semi-structured interviews, where only one or two people per company were interviewed.

To complement the analysis by Hutchinson et al., this paper reports on a much deeper, albeit less broad, study of MDE practice in three large Swedish companies: Ericsson AB, Volvo Cars Corporation and the Volvo Group. The aim was to limit the investigation to a small number of companies but delve more deeply into how MDE is being applied in each. This paper compares and contrasts MDE practice at the three companies and examines how their experiences either validate or refute findings from the earlier study by Hutchinson et al. As such, this paper is a narrow but deep investigation of MDE practice which complements Hutchinson et al.'s broad but shallow study.

The three companies in question were deliberately chosen to have both similar and contrasting features. Ericsson's Radio Base Station unit was an early adopter of MDE (since the late 1980s), has many years of experience in applying MDE for mobile communications, and has focused on the use of UML and UML profiles. Electronic Propulsion Systems at Volvo Cars, on the other hand, committed to MDE in 2010 within a domain (automotive) which is often highlighted as a success story for MDE, and focuses on models developed using Simulink. Whereas MDE is now an accepted part of the development culture at Volvo Cars, software implementation at the Volvo Group is still mainly a code-centric business with a few pockets of MDE expertise. All three companies develop products in a specialized embedded systems domain. At the time of the study, all three organizations were undergoing a transition to agile development practices.

Our methodology was to study each company through a series of 25 in-depth interviews with software developers and project managers, split across the companies. Interviews were audio recorded and transcribed. A grounded approach was then used to extract useful insights. Additional fieldwork, in the form of attending project meetings, informal follow-on conversations, and reporting back to the companies at formal meetings, also provided data in the form of field notes.

Our results both confirm and refute findings from Hutchinson et al.'s study and the broader literature on MDE practice. We observed a number of key differences in the way that MDE is being applied across the three companies, and that these differences can have a significant effect on the overall success of the MDE effort. Contrary to perceived wisdom, we have also observed that agile methods and MDE can co-exist peacefully although we note a number of tension points where their joint application may aggravate a development process. The value of abstractions is heavily dependent on context, and can in fact complicate and obscure the structure and behaviour of a system instead of promoting clarity. Where and how to apply MDE still seems to be a question without a fine-grained way to determine the answer. However, in one company, MDE had a significant effect on productivity by integrating domain experts with low-level implementation by forming a new unit supported by an MDE framework.

2. COMPANY CONTEXTS

2.1 Volvo Cars Corporation

Within Volvo Cars Corporation, our study focused on a unit named Electric Propulsion Systems, EPS. EPS is a new unit developing software for electric and hybrid cars. At the time of the study, EPS was developing their second generation of electric propulsion vehicles. Whereas the first generation was developed independently of the rest of Volvo Cars, the second generation was being integrated into a new Volvo Cars platform under development. The second generation was the first major project at EPS which applied MDE. Although EPS started from scratch in terms of defining and building their MDE infrastructure, there was a legacy of MDE knowledge within Volvo Cars to build on, since another unit, working on combustion engines, has had experience with MDE since 2005.

At the time of the study, there were two major process changes underway at EPS. First, there was a push to increase the proportion of in-house software development. This was done in a bid to keep expertise and domain knowledge within the company. Secondly, EPS was undergoing a transition to an agile development process in order to shorten lead-times. In both cases, MDE was seen as a key enabler. To explain how agile methods and MDE are applied at EPS, we briefly explain the development process at Volvo Cars.

Development at Volvo Cars has traditionally followed a five phase waterfall model – see Figure 1a. A new concept – e.g., a new engine type – is first scoped out as a set of use cases and Simulink models. Together, these specify the concept. At this stage, both functional and non-functional requirements are formally specified in these models and simulations validate them. Phase 2 (System design) breaks the concept specification down into a set of textual requirements, component interfaces described in a graphical modeling notation, a database model describing the electrical architecture of the car, and a deployment model describing how to map components on to ECUs (Electronic Control Units), defined in a company-specific tool. An ECU is an embedded system that controls an electrical subsystem in the car (e.g., brake control, transmission control). It is implemented by one or more software components. A modern car can contain up to 150 ECUs and the life-time expectancy of an electrical architecture is at least 10 years.

In Phase 2, components are defined black-box. The electrical architecture connects the ECUs to the software components and is defined on top of the AUTOSAR (AUTomotive Open System ARchitecture[1]) standard, a hardware abstraction layer that facilitates updating hardware and software during the electric architecture's lifetime.

Phase 2 is designed to support parallel development of components: it freezes the component interfaces so that components may be developed either in-house or by subcontractors independently. Given the importance of the component interfaces to the overall development process, there is a formal approval process for interface changes, which may take up to 20 weeks.

It is in Phases 3 and 4 (Model-in-the-loop and Integration) where MDE has brought significant changes. Previously, most component development was done by sub-contractors based on the Phase 2 specification. Now, the move is towards defining the internal behavior of components using Simulink in-house. The interfaces of the Simulink models are generated from the system design as black boxes which are then manually filled with behavior according to the textual requirements. The models are then used to generate executables for testing and integration in Phase 4.

Since some of the components interact with components that are developed off-site, their interaction is validated in a simulated environment developed by EPS. To calibrate the behavior of the models, testing on rigs of hardware is required. Without the rigs and simulations, validation would have to wait until both in-house and out-sourced components were available for testing. Ultimately, if a concept is successful, it will go into mass production (Phase 5), although most concepts do not make it that far.

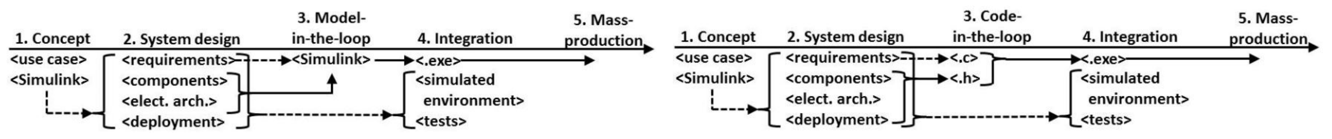
As mentioned above, the organization was going through a transition to an agile development process. In practice, this meant that each team working on a component had a large degree of freedom to choose development methods and styles. Agile methods did not, however, extend across all phases. Component interfaces were still defined in Phase 2 with a lengthy approval process required to change them. As we shall see later, this caused tensions for the engineers, who, on the one hand, were working in a very agile fashion, but, at the same time, were expected to work to previously frozen and difficult-to-change interfaces.

2.2 The Volvo Group

Within the Volvo Group, our study focused on Trucks Technology, which develops software for various truck brands – Volvo, Renault, Mack, UD and Eicher. The five brands are developed on top of a shared platform. The truck platform serves the same purpose and has the same overall structure as the car platform at Volvo Cars but has a longer life expectancy (around 20 years). Trucks Technology take overall responsibility for system design and integration but most of the software is implemented by external suppliers.

The process at Trucks Technology (Figure 1b) is very similar to that at EPS with two exceptions. Firstly, since each truck brand has unique mechanical and hardware configurations (e.g., number of axles, suspension type), each brand needs different parameters and parameter values in the software. With a shared platform for all brands, the system design needs to keep track of all possible configurations. In total, there are 150,000 configuration points.

[1]`http://www.autosar.org`



(a) Process at Volvo Cars

(b) Process at Volvo Trucks

Figure 1: Solid arrows are automated transformations; dashed arrows are manual implementations.

Secondly, at Trucks Technology, Simulink is only used in Phase 1. Implementation is in C. So, instead of generating Simulink from the system design, header files are generated. The specifications of the header files are then realized using C in Phase 3. Even if the implementation is in C, Trucks Technology have developed a simulated environment since they have the same need for early validation and integration of software and hardware as EPS.

After having just successfully launched their first truck on the new platform, at the time of our study, Volvo Trucks was in a time of organizational reflection. Trucks Technology were considering to expand their MDE activities and introduce more executable models, a move that is justified by a need to validate integration earlier and a wish to raise the level of abstraction in their implementation.

2.3 Ericsson AB

The Radio-Base Stations unit (RBS) has used MDE since the late 1980s. RBS delivers software for mobile communications on the GSM, 3G, 4G and multi-standard networks. 90% of the software developed at RBS is deployed on a single hardware node and so there are strict constraints on memory consumption and processing speed. The reason for adapting MDE at RBS was primarily to handle the rising complexity of the products. RBS has defined their own UML profiles using off-the-shelf modeling tools from a major tool vendor[2].

All four generations of mobile communication have undergone the same transition, from inception to becoming established products. RBS uses two types of model: descriptive and prescriptive. In the inception phase, models are descriptive – that is, they are used mainly for communication and documentation. In this phase, however, the focus is on developing new features and so models are kept lightweight to avoid slowing the process down. Component diagrams and collaboration or sequence diagrams are used in this case. During the main development phase, implementation may be code-centric or model-centric. In the latter case, prescriptive models are used – that is, models that precisely describe the system to be built. Code may be generated from prescriptive UML state machines and custom-made DSLs are used to test the model. When it is clear that a product will survive on the market and will need to be maintained for many years, the need for more accurate documentation arises, which necessitates creating new models or improving existing models.

A year before the interviews were conducted at Ericsson the interviewees had been reorganized into cross-functional teams, XFTs [15]. At Ericsson, an XFT consists of one software architect (often the domain expert), around five developers and two testers. By combining different skill sets, the aim is to reduce the number of manual hand-overs in the development process as well as to shorten lead times and have a better integration of hardware and software. The transition to XFTs meant that the development teams now have a short-term commitment towards features instead of long-term responsibility for a certain phase of the development process.

XFTs have not changed the need for consistent model-based documentation, neither have they enforced a move towards code-centric development. The choice of implementation method is a question of legacy: if a sub-system started as an MDE project, it will continue to be developed using the same MDE tools (or upgraded versions); if the sub-system started out in code-centric fashion, it will remain so.

[2]not named, for reasons of confidentiality

3. METHOD

3.1 Aim

Our study aims to investigate the following two questions: (i) What are the similarities and differences in the way that the three companies have adopted and applied MDE? (ii) How does the nature of practice at the three companies agree or disagree with the findings from Hutchinson et al.?

Recall that our study – as a deep study at three large companies – is intended to supplement that of Hutchinson et al., which is broad and shallow. Note that we only studied particular sub-units within each company, as described in Section 2. For ease of writing, we henceforth refer to Ericsson RBS, Volvo Cars EPS, and Volvo Group Trucks Technology as Radio, Cars and Trucks, respectively.

3.2 Study Design

The three companies were deliberately chosen because they are similar in some respects but different in others. All three are large companies developing products in an embedded systems domain. However, the way they have adopted MDE (see Section 2) is different in each case. These characteristics allowed us to compare and contrast MDE practices across the three companies. Another reason for choosing these companies is more pragmatic: we had existing relationships with key individuals in all cases. We approached our contacts in each company. In the case of Radio and Trucks, based on a discussion of the aims of the study, our contacts suggested sub-units to study. In the case of Cars, our contact already worked in an appropriate sub-unit, which was therefore self-selected. At the end of this selection process, each company had assigned a point of contact responsible for coordinating the study within the company. This responsibility included selecting suitable interviewees, supporting site visits, and coordinating formal and informal progress meetings within the company.

Data was collected from each sub-unit using a variety of methods: (i) semi-structured interviews with staff; (ii) conversations with staff in informal settings (e.g., over coffee or lunch); (iii) observations of design and development ac-

Role	MDE Exper.	Organization Radio	Cars	Trucks
Architect	5+	4	2	2
	0-5	0	1	0
Developer	5+	2	2	0
	0-5	2	5	1
Tester	5+	1	0	1
	0-5	0	1	0

Table 1: Roles and MDE experience of the interviewees

tivities, and team meetings; (iv) formal progress meetings with the company points of contact and additional managers or engineers depending on their availability and interest; (v) presentations of results to broader audiences at technical meetings. We used the interview data as the primary source: findings emerged from the interview data; we then used the other sources to validate or elaborate on these findings. Field notes were taken in (ii)-(v); we followed established practices for documenting informal conversations and observations [4, 12]. Data collection was primarily carried out by the first author; the remaining authors participated in activities related to (ii), (iv) and (v). A non-disclosure agreement was signed with each company to allow for open and honest sharing.

We interviewed a total of 25 individuals across the three companies – 9 at Radio, 12 at Cars and 4 at Trucks. Interviewees were selected to represent a range of architects, developers and testers, all of which had been involved in an MDE effort. Table 1 summarizes the distribution of interviewees across these roles as well as the number of years experience each has using MDE in this role. The relatively low number of interviewees at Trucks is due to the fact that Trucks is not yet active in using MDE during the development phase; hence, we could not interview MDE developers there. At Radio and Trucks, the interviewees were dominated by more experienced engineers (5+ years of MDE) but each company assigned at least one engineer with limited MDE experience. At Trucks, the experienced engineers had often worked with MDE at other companies before joining their present employer. Due to how MDE was adopted at Cars, the less experienced interviewees outnumbered the more experienced ones seven to four. One interviewee at Cars volunteered to participate out of curiosity.

At Cars and Radio, the interviews took place between January and March, 2013. For Trucks, the timing was slightly different – interviews took place between early March and June. None of the interviewees spoke English as their native language. Even so, the majority of the interviews were conducted in English so that the content could be analyzed by all the researchers. All interviews were conducted by the first author and were carried out on-site.

The interviews lasted an hour each on average and were semi-structured in nature. The interviewer began with general questions - such as the interviewee's current role and background - and then delved into MDE-specific topics, such as their experience of MDE, their motivation for applying MDE, the benefits and challenges of applying MDE in practice etc. Each interview followed a different direction according to the interests of the interviewee and what the interviewer picked up as particularly interesting. In some cases, the interview was followed by a second interview or an informal meeting in order to clarify statements. An initial analysis of each interview was conducted immediately after carrying it out; this enabled emerging themes to be explored in subsequent interviews, thus allowing later interviews to confirm or deny earlier ones, as advocated by Robson [14].

The interviews were each recorded and transcribed. They were then analyzed using a lightweight version of grounded theory [6, 16]. Each interview was analyzed and annotated by at least two researchers; the annotations were then grouped into themes. Themes, as well as the comments from where they originated, were discussed extensively amongst the three researchers and a consensus was reached, in each case, on which findings were the most interesting to document. In each case, findings were validated or further explored using data collected in (ii)-(v) above. The themes were also contrasted with the findings of Hutchinson et al.'s study.

4. FINDINGS

In this section, we present the most prominent findings from the data collected during our study. Each subsection describes a recurring theme where we found similarities and/or differences in the way different companies were applying MDE. We illustrate the themes with exemplars taken from the interview data; where it helps to illuminate key issues, quotes are included in italics. These quotes are taken verbatim from the interviews. In some cases, we mark an interviewee's response in a quote with *"I"* and the researcher's question with *"R"*.

4.1 MDE to Bring Development In-House

In earlier work [18], Hutchinson et al. noted that MDE can allow some companies to bring out-sourced development back in-house. The reason given for this in [18] is that out-sourced tasks are often relatively straightforward development jobs; these 'simpler' jobs are easier for subcontractors to carry out independently, but are also easier to automate using MDE. In our new study, we also found that MDE can bring out-sourced jobs back in-house; however, different reasons for this were discovered. Indeed, one of the primary motivating factors for introducing MDE at Cars was to bring development tasks in-house. Traditionally, the models developed in Phase 2 of Cars' process (Sec. 1a) were sent out to external suppliers for implementation. This is currently still the case at Trucks. However, at Cars, MDE has reduced the dependency on these external suppliers since the out-sourced tasks are now done in-house – using Simulink to automatically generate development activities previously left to the suppliers.

Cars have found two key benefits from this. Firstly, Cars now has more control over the development process: they are no longer dependent on suppliers who may not deliver on time and who may not deliver sufficiently high quality components. Secondly, in-house MDE development supports Cars' move towards agile development. In agile development, external dependencies – such as on suppliers – can be a major point of tension. Using suppliers as part of an agile process threatens to break two key agile principles: (1) it slows things down because the team becomes dependent on delivery schedules of the supplier; and (2) it forces the team to write specifications for the supplier to work to, which can lead to overly heavy documentation. Hence, the most effective agile process is one in which all development is carried out in-house (not necessarily physically co-located). MDE has allowed Cars to develop both the Simulink specification and the generated implementations in-house. This has given them a lot more control over their process leading to productivity gains. It is also a success story in which agile and model-driven methods can not only co-exist but can actually mutually support each other.

Compare, for example, the following quotes. At Trucks, which is still dependent on external suppliers in India, there are issues because the Simulink specification sent to India requires domain knowledge to properly understand it. Contrary to perceived wisdom, a formal model is not necessarily understandable without the domain background because there are all kinds of hidden assumptions in the definition and integration of Simulink blocks. This can lead to problems where suppliers without domain expertise improvise incorrectly around the specification: *"...you can just get a feeling for if it works correctly or not. But usually in Bangalore, they don't ... Many of them haven't seen a truck or played around with it at least, so they don't know. They don't have this natural feeling for ... how it should work".*

In contrast, at Cars, bringing the development in-house, using MDE as an enabler, was a way to increase productivity by controlling tightly scheduled iterative development loops: *"That's the key objective, so to say, with this whole in-house software development ... That's where the big increase in speed is from a Volvo perspective."* MDE was also seen to have other benefits than simply speed: it led to cost savings and a higher quality product because Cars could now produce software more relevant to its business, rather than software adapted by a supplier from another customer *"in some cases ... [suppliers] are just reusing an old software that they have delivered to some other OEM [Original Equipment Manufacturer]."*

4.2 Leveraging Domain Experts

A key benefit of MDE at Cars is that it allows domain experts, who specify requirements, to be directly involved in the development process because they understand the Simulink model and can work together with developers to generate code from it. This again has two major benefits. Firstly, it means that domain experts are much more closely involved in the implementation process: developers and domain experts work together on developing the Simulink models and generating code from them. This in turn leads to a higher quality product. One interviewee put it well: *"In my opinion, the only reason to work with Simulink, is that the system designer and tester and all other stakeholders at Volvo actually understand what they see. You can sit around a Simulink model and discuss an implementation and review it, and so on. If we would have been working in C or something like that, it's just the programmer who understands what is happening in the code."*

Hutchinson et al. noted the importance of having two key skill-sets to make MDE work: both modeling/abstraction skills as well as compiler/optimization developer skills [18]. Both skills are needed to apply MDE successfully in practice because a company has to come up with good models but also typically needs either to develop its own model transformations or adapt off-the-shelf transformations. Whereas Hutchinson et al. recommend both skill sets to be possessed by the individual – what they term the 'MDE guru' – Cars has been successful even when these two skill sets are held by different people. The critical factor that makes this work, however, is that MDE brings the modeling (or domain) expert much closer to the software expert. The use of executable models reduces misinterpretations of the specification and allows domain experts and developers to 'try things out' in close cooperation.

The second benefit is once again that it supports the agile processes at Cars because those specifying the requirements and those developing the implementation work closely alongside each other. Indeed, many of the employees at Cars come from an electrical or mechanical engineering background and have been trained in Matlab/Simulink rather than programming languages like C. Hutchinson et al. [7] found a similar story at another large automotive company in which MDE was introduced precisely to address the lack of software engineers compared to electrical engineers: *"You couldn't find a computer scientist if you went on a search party."* MDE allowed this company to build software using domain experts who understood Simulink and a relatively small team of software specialists could build the generators. The case is similar at Cars. MDE reduces the need for *"pure ... software guys"* because it allows domain experts to get directly involved in (some) implementation tasks.

4.3 Secondary Software Supporting MDE

Previous research has pointed out that the introduction of MDE tools requires significant effort in adapting them to the context of the organization [18, 17]; commercial MDE tools cannot be simply taken 'off-the-shelf' but may require major tailoring to a company's existing processes. In our study, we found further evidence of the need for this, but also found that it is not just single tools that need tailoring, but that an organization may need an entire suite of supporting applications – which we term secondary software – to make MDE work in practice. Crucially, much of this supporting software is needed precisely to allow domain experts to participate in the development process.

As an example, scripts play a major role in tailoring MDE at Cars: *"they have actually just made some scripts to make life easier".* These scripts have different origins – *"a few scripts are from the supplier and a few scripts are developed by us, yeah, and by Volvo"* – and are used in various ways, from massaging XML files into the correct format to flashing Simulink models onto hardware or for diagnostic purposes (*"When you generate code and deploy it on hardware, you also through the scripts add a lot of debug information"*). The message is clear: there is a lot of "grunt work" needed around the MDE tools to support both integration of MDE tooling into a broader context and to provide support for domain experts being part of the process. This is an interesting observation because one of the commonly perceived benefits of MDE is to automate 'grunt work.' Here, the grunt work is automated, but the need for it arises directly because of the use of MDE.

Interestingly, Kuhn et al. [11] found that one of the frictions that engineers at General Motors encountered was the lack of support for developing scripts and small applications. From our study, it seems that this is a question of organizational support and not the accessibility to domain-appropriate technologies.

The motivation for these scripts is to automate recurring tasks that are complicated or tedious. (This supports Hutchinson et al., who state that one of the success stories of MDE is when small DSLs are used "bottom-up" to automate small but repetitive tasks [18]). Furthermore, the scripts form an important part of the secondary software that domain experts at Cars depend on to be active contributors in the development process – since the scripts compensate for their lack of programming skills by automating many tasks that might be straightforward for programmers but not for domain experts. These simple tasks can range from extracting a view of the signals from the overall system model, through to "trimming" a Simulink model to ensure performance in the generated code: *"Yeah. Exactly. So you have a model, and then your inputs to that model are calibration variables. So I've been working with them and optimizing the engine ... so I get the perfect output ... "*. Taken together, the scripts provide a chain of functionality, which is absolutely essential: *"But the benefits I perceive from modeling your software, they will not be utilized as much as possible if you don't have a good framework to work with."*

In this way the MDE framework encodes the software development process, from requirements down to deployment of binaries and ensures that the information stays in-house. Another consequence is a shift of focus when it comes to reuse – the MDE infrastructure enables fast re-implementation of functionality so the reuse and maintenance of existing models is not as important as the reuse and maintenance of the tool chain. A lesson here is that the framework should developed in an agile way to quickly supply new features as the complexity of the system grows.

In contrast to the emphasis at Cars on developing and maintaining secondary software, Radio, as early adopters of MDE in 1980s, arguably underestimated the need for secondary software to tailor tool suites: *"I: Because that is not the core business for Ericsson ... R: Perhaps that is something that scares the management at Ericsson, it is a huge invest– I: – yeah. It isn't always that easy to define the business case."* In other words, the development of secondary software can seem like a distraction from the organization's core business. It is essential, however, for companies to recognize that secondary software in MDE *is* part of the core business.

The exception at Radio is a UML profile used for defining the management interface for the base stations. The profile has its own organizational unit for its development and maintenance, supported by a set of tools and model transformations enabling the interface specifications to be ported to different textual formats: *"We need to be very careful with what we change because it will have an impact on customer tools ... We do have a process for how to change it and we review the changes very carefully. For new functions we want it to look similar, we want to follow certain design rules and have it so it fits in with the rest."*.

4.4 Legacy

In adopting new processes in a large organization, a critical challenge is how to deal with legacy. We saw interesting contrasts, for example, in how the three companies integrated new development methods, e.g. agile processes and MDE, with legacy processes, such as a waterfall.

A legacy of the waterfall model at Cars, for instance, is that Cars freezes the component interfaces in Phase 2. The motivation is that sub-contractors and suppliers can then independently develop software in Phase 3 using their own processes and technologies as long as they fulfill the interfaces. In principle, this allows a degree of organizational control over the process. This process of freezing interfaces remains at Cars, despite a move towards agile MDE where components are developed by domain experts using Simulink models and code generation. But a tension point is the need to adhere to frozen interface definitions, which are not easy to change.

The interfaces and electric architecture of the car are frozen around twice a year. This freeze fixes the entire architecture of the car. Although interface changes can be requested between freezes, this involves a lengthy negotiation process between all teams that are dependent on the interfaces and so teams are often reluctant to undertake this: *"R: So you can't change it if you come to understand that, wait a minute, I need to have this parameter as well. I: Absolutely. R: Or I need a new signal to answer this signal in case of - I: Exactly."* Instead, developers try to predict what they will need and add extra interface parameter requests; these parameters may ultimately turn out to be redundant. Hence, there is a problem of interface bloat and Cars ends up with agile teams working effectively but constrained by a highly non-agile infrastructure. The same issue of frozen interfaces was described at Trucks.

Radio uses XFTs, which appears to make the process of changing interfaces straightforward – *"as an XFT team, when I do some changes on the interface, I do both sides"* – because the XFT has control over an entire feature, whereas, previously, developers only had ownership of a small part of a feature and could not update other aspects: *"Somebody had to decide and define down to bits and pieces in a document and assign that work item to you ... And then you had to be finished at the same time because you have to deliver at the same time. So you had to synch that."*

The lesson here seems to be that conflicting process cultures can lead developers to subvert processes: the Cars case of agile developers trying to work around non-agile interface freezes is a case in point.

4.5 System Comprehension and Abstraction

Abstraction is usually argued to be a major benefit of MDE. High-level, abstract models should aid system comprehension and allow stakeholders to see 'the big picture'. Although raising the level of abstraction of system development is undoubtedly a worthy goal, and MDE certainly goes some way towards supporting this, our data contains numerous examples where models and modeling tools can actually work against system comprehension. Once again, there are both similarities and differences in how this manifests itself at each of the three companies.

Interviewees at Radio, for example, reported multiple problems of comprehension, ranging from overly complex model specifications to difficulties in merging and navigating models to the irreversibility of design decisions when modeling. The sheer size of the models used at Ericsson became a barrier to understanding them: *"We had one model that we always updated and tried not to branch it too much because it became impossible to merge all the changes that we had. And we also had like 8000 sequence diagrams that we should maintain."* Eventually, developers drifted back to Microsoft Word because it was simpler to maintain: *"And I think that got people to be a little bit afraid of the amount, all this big amount of sequence diagrams and all this big amount of information that we need. Of course we needed the informa-*

tion, but it became so painful to update it for every feature that we tried to add to the system . . . So I think that is the reason why people started to use Word again."

Another Radio interviewee talked about the difficulty of reversing design decisions: *"So someplace there you have to decide is this capsule actually consisting of many capsules or are there actually two capsules because there are probably parallel state machines. But if the system is analyzed wrongly on top and you try to realize it by using software UML, then it ends up with something that's very, very strange. And it costs a lot to handle that . . . Refactoring stuff in that situation is really hard."* The reason for this is that current MDE tools (a UML tool from a major tool vendor in Radio's case) force the developer to make decisions but these cannot easily be undone later because there are so many different models that are mutually dependent on each other across multiple levels: it can be hard to predict the effect of a refactoring and to make sure that all views are consistently updated. This is in contrast to refactoring code because developers are not forced to edit according to the abstract syntax tree: they can cut/paste text freely.

Even predicting the effects of changes is hard because it can be very difficult to navigate through models due to multiple levels and viewpoints: *"If you look at it, you just see one point. It's like playing football with a cone. You just see this part. And then you double click to open something else. And double click to open something else. And then you look at that. And then you've forgotten this . . . Your brain, my brain, at least, can't handle that."* In addition, the non-linear nature of graphical models makes them difficult to read because there is no obvious place to start: *"When you document, it puts it on the table. Start at page one. And it goes to page 200. And then you're finished. But a model isn't like that."* A similar phenomenon was reported by Kuhn et al. [11], who found that a modeler at General Motors created his own linear numbering scheme for Simulink blocks so he could remember which block to start reading from.

The result is that developers at Radio often ended up struggling with the models and the modeling tools. As one interviewee put it: *"Because in textual coding the code is the main artefact. But when you go to graphical modeling, the tool becomes the main artefact. Because you can't really export your models to a new tool."* An open research question, therefore, is how to free MDE from the constraints of tools?

Similar problems were experienced at Trucks. Although MDE adoption at Trucks is still in its early days, there is still a lot of modeling of requirements in Simulink; these Simulink models are typically sent to suppliers for implementation. These requirement models can become very large: *"If we take out the generated specification from this tool . . . it is 3,000-plus pages."*. The sheer size of this specification means that suppliers cannot or do not look at everything: *"And he [supplier] says, 'I've never read it. Because I wouldn't do anything but just reading it."'* Typically, Trucks extracts what they think is the relevant part of the specification for each supplier, but this can cause problems if information is missing: *"They're looking through a peep-hole really."*

An additional comprehension problem reported at Trucks is related to finding the right level of abstraction. A key issue appears to be that different developers model at different abstraction levels, which can cause mismatches or incorrect interpretations: *"We probably have 60 function developers right now actually working. And they each . . . have a little bit of different dialect when they are writing. Somebody goes a little bit lower. Some a little higher. Different experience. And probably you have at least 50 of these persons all writing requirements in this . . . area. So you have 50 dialects of writing requirements in this spec."*

4.6 Craftsmanship

One of the key messages that arises from our interviews is a lack of knowledge about when and where to apply MDE. MDE is not suitable for all development tasks – and indeed, different types of MDE are more suitable in some cases than others – but there is currently a lack of experience in the MDE industry as to how to apply MDE most effectively where it matters most. This manifests itself in four ways: (i) knowing which parts of a system are most appropriate for MDE and which are not; (ii) knowing how best to apply MDE once a decision is made; (iii) encoding best practices to transfer MDE "craftmanship" to future projects; (iv) avoiding over-generalization by assuming that if MDE worked well for one project, it will work for others.

Cars has partially overcome some of these problems by adapting the experiences of MDE from other units within Volvo, such as Powertrain, which develops software for combustion engines: *"It's like this. Right now we are in a learning process of what these Simulink patterns look like to generate efficient code. Powertrain, which have been code generating for many years, have patterns for TargetLink, who are competitors, and there Volvo has a very good knowledge of what is good and bad design patterns. When it comes to the world of Simulink, it is something that we currently are obtaining, that experience, what we should and should not do in a Simulink model."*

The introduction of Simulink at Cars is therefore based on the concrete experiences from previous projects within the company and the skills of individual engineers. However, this does not imply that Simulink is suitable for all domains within Volvo Cars. Our interviews show, for example, that the transition into MDE is more complicated within the active safety domain. In this case, there are challenging runtime constraints which generated code must satisfy, since the underlying algorithms demand more CPU and memory than the algorithms used at Cars. Even within automotive, some sub-domains then are more suitable than others because of (e.g.) the requirements place on the generated code. It appears that as an MDE community, we have very little way of assessing, in a fine-grained way, the requirements for each sub-domain, how those requirements affect the applicability of MDE, and what are the MDE patterns most appropriate for that particular sub-domain.

Contrast this with the analogous situation for programming languages at Radio. There, the organization has spent many years building up a wide-spread knowledge of how best to match programming languages to specific sub-domains: *"We cannot use one language for every part because they're not suitable for that . . . It's great to write test kits in Erlang . . . When close to the OSS [Operation Support System] and GUI, of course, Java is excellent."* When transitioning into MDE this knowledge has to be rediscovered. An example of this comes from Cars where a consultant with prior experience of both C and Simulink stated that *"I found out making timers, it was tough in the C code . . . it maybe took you 20 minutes to set up one single timer. And in the modeling world, it's just a matter of seconds to make a timer . . . On* *the other hand, if you're doing some data, like moving in an array or something, . . . I found that quite complicated in the modeling world. That was more practical in C."*

According to one project owner at Radio, there are two questions to ask when considering modeling a certain subdomain: (i) Will the generated code be efficient enough, and (ii) Do the developers have the right competence: *"You have to take that into account when you form the business case for the start of the project. In some areas, the cost for developing that competence is too big . . . If MDE is not something that shows black figures in the end, we shouldn't do it."* Clearly, there are other considerations too (for example, interviewees at all three companies said that applications with strict non-functional requirements on memory handling and processing capacity are known to be difficult to implement using MDE); the point is the application of MDE on a large-scale in practice still appears to be in a relatively immature state – one in which elements of MDE "craftmanship" have not yet been formalized and shared.

4.7 Applying MDE in a New Unit

Other authors have written about strategies for introducing MDE into an organization. Hutchinson et al., for instance, talk of the need to place MDE on the critical path in a development effort – however small it may be – to ensure that the MDE team is not assigned the weakest staff [18]. Aranda et al., in a study of MDE at General Motors, report tensions when engineers were asked to redefine their role in an organization due to the introduction of MDE [1]. This came about because GM employed domain experts with a background in physics and mechanical engineering as software developers, which was made possible because they knew Simulink from their University training. Software engineers were employed to develop the secondary software. This division of resources, which was similarly applied at Cars, required engineers to redefine their roles.

Cars, perhaps by chance, came up with an interesting and rather successful strategy for introducing MDE. Because Electric Propulsion Systems was a new business area for Cars, a completely new unit was created for the development work. The creation of a new unit meant that many of the classical issues associated with change management – such as requiring engineers to redefine their roles, or requiring culture changes within a team – naturally did not exist. The new unit was not tied down to legacy, either in terms of software, organizational culture, existing process, or external relations. In other words, the new unit more or less had a clean slate to develop new ways of working as needed. Crucially, however, such a new unit could not succeed in a vacuum. It was not enough simply to create the new team and expect great things to happen; rather, the team had access to historical expertise in MDE from other units within Volvo. But the new unit allowed the team to pick up best practices as they wished, and to exclude practices that did not resonate with their new way of thinking. An example of this is how the new unit at Cars defined their own agile processes on top of MDE processes from Powertrain. Although the creation of a new unit may not always be feasible, or even desirable, this way of handling the introduction of MDE is consistent with the recommendations of Christensen, who states that when changes become too extensive within an existing organization the solution is to start afresh [3].

5. RELATED WORK

5.1 Hutchinson et al.

The most closely related work to ours is that of Hutchinson et al. [7, 8, 9, 18]. This subsection therefore gives a detailed comparison of the findings from the two studies. The main conclusions of Hutchinson et al. are given in Table 2, grouped according to different aspects of introducing MDE. The final column in the table indicates whether our findings either validated (V) Hutchinson's study or refuted (R) it. Although most of our results validate Hutchinson, we both (i) offer additional evidence to support the finding (a form of replication), and (ii) typically offer deeper or additional insights related to the finding in each case.

Earlier sections of the paper have elaborated on validated findings. In this section, therefore, we focus on findings which refute Hutchinson et al. The refuted findings come exclusively under the categories of Control and Training.

In relation to Control, we discovered that software architects at Radio who are used to specifying their domain knowledge in textual documents found MDE tools and languages intimidating. The introduction of MDE enforced on them a level of formality with which they were not comfortable. In contrast to Hutchinson, who argues that middle managers are often a bottleneck when introducing MDE because they are risk-averse, our study shows that this need not be the case – again at Radio, we found that middle managers can be champions of MDE precisely because it can minimize risks related to human error or external organizational dependencies. We also found that code gurus are not necessarily averse to MDE; software engineers employed at Cars, for instance, do not mind building secondary software to support domain experts because they have specifically been employed as consultants to do this.

In relation to Training, Hutchinson et al. have criticized UML education in Universities because educators often tend to focus on teaching the syntax of UML rather than problem solving. At Cars, the domain experts have been trained at University in Simulink and, contrary to UML, appeared to be well-equipped to enter the company and transition, using MDE and Simulink, to a more software development-oriented role. Hutchinson et al. have also argued that Universities unnecessarily separate modeling/abstraction skills from compiler/optimization skills in their curricula. This can cause problems, but does not appear to do so at Cars because Simulink allows the domain experts and implementors to work very closely together.

Similarly, at Cars, there is a supply of Simulink-trained engineers leaving University, which is exactly what Cars needs to hire. This conflicts with many situations in the Hutchinson study where companies hired engineers skilled in a particular technology, but then required them to work on modeling, for which they were not trained.

5.2 Other Related Work

There are a very large number of papers describing case studies of applying MDE in practice, as well as a smaller number of empirical studies on industrial use of MDE. To limit the scope, we here describe only the most prominent of these papers, with a particular emphasis on those that go beyond the pure technical aspects of MDE in that they also include organizational and social factors, especially as relates to the automotive and telecoms sectors.

	Findings from Hutchinson et al.	**V**alidated or **R**efuted
Domain	Successful MDE tends to favor DSLs over General-Purpose Languages [18]	V
	When developing DSLs, focus on narrow, well-understood domains [8, 9, 18]	V
	Domain experts already model: they use informal DSLs so can transition easily to MDE [18]	No data
Process	MDE more successful if driven bottom-up, by developers (not from managers) [7, 18]	V
	Put MDE on a critical path to get the best resources [7, 8, 9, 18]	V
	Productivity gains from MDE can be lost elsewhere (e.g., unreadable generated code) [8, 18]	V
	Most MDE project failures are at scale-up [8, 18]	V
	Organizations must constantly evolve MDE infrastructure as understanding grows [7]	V
Motivation	MDE needs a clear business driver [7, 18]	V
	MDE works well for domain-specific products, rather than general software [9]	V
	Target MDE where it can have maximum impact quickly [9, 18]	V
	Organizational buy-in for MDE is important at all levels [7, 9]	V
Control	MDE requires significant customization of tools to an organization's context [18]	V
	MDE can bring outsourced development activities back in-house [18]	V
	Architects like MDE: the generator lets them control architectural rules more easily [18]	R
	Code gurus do not like MDE: they lose control to the generator [18]	R
	Middle managers are risk-averse and reluctant to try new techniques such as MDE [18]	R
Abstraction	If the models are too close in abstraction level to code, there are limited benefits [7]	V
	Simplicity in models can be counter-productive; managers may see it as laziness [18]	No data
	MDE engineers need a mix of abstraction and compiler skills [9, 18]	V
Training	MDE training is too often focused on tool idiosyncrasies, rather than problem solving [7]	R
	Universities do not adequately train engineers in MDE [18]	R
	Businesses hire too much based on knowledge of specific technologies of the day, which may not be relevant to modeling [18]	R

Table 2: Comparing the conclusions of Hutchinson et al. with data from Radio, Cars and Trucks.

In a recent publication, we presented a taxonomy of challenges and potentials when applying MDE tools in industry [17]. The taxonomy was deduced from the interviews conducted by Hutchinson et al. and validated through a limited set of the data obtained in this study. The findings emphasize how current MDE tools are lacking in both usefulness and usability and concludes that more research is needed to understand the interplay between organizations, human factors and the technical features of the tools.

Kuhn et al. [11] and Aranda et al. [1] report on two parallel studies of applying MDE at General Motors. The former focus on individual perceptions of the adoption of MDE at GM; the latter reports on how MDE induced changes at the organizational level. At the individual level, engineers experienced both forces and frictions related to MDE tools and languages – for example, a lack of support for developing secondary software (see Sec. 4.3). At the organizational level, [1] found, for instance, that software developers felt "downgraded" when MDE was introduced, because they were now asked to focus on secondary software whereas domain experts focused on primary functionality.

Baker et al. [2] describes a process of introducing MDE at Motorola. They also report on the lack of organizational maturity in terms of which processes to use in combination with MDE, the difficulties in adapting existing skills to the new challenges of MDE and the unwillingness of the organization to change in order to make the most out of the transition into MDE. A previous study conducted at Radio can be found in [13]. Here the emphasis is on MDE in relation to architectural concerns (deployment, algorithms, performance etc.) but the authors point out some organizational aspects of MDE, such as the possibility that the cost of modeling can outweigh the benefits.

6. VALIDITY

In terms of threats to internal validity, we followed a systematic approach in setting up the study and followed best practice guidelines in both data collection and analysis. A systematic approach is straightforward to follow in the case of the interviews (i), as strict protocols can be set up. In the case of the more informal aspects of the fieldwork – namely, (ii) and (iii) – there are inevitable critiques of representativeness and rigor. However, we were not able to carry out more rigorous, formal experiments within the companies because this would have been too disruptive. In such cases, informal interactions are an established way to gather data [4, 12]. Indeed, a benefit of informal interaction is that it lets the respondent be in control, which, in turn, enables the discourse to lead to new topics not anticipated by the researcher [5]. The informal settings allowed the researchers to interact with employees that were not assigned as interviewees. This gave an opportunity to validate if data collected through the formal interviews represented a shared understanding or a minority view. Another benefit of the informal interactions was that often engineers from other units would be present, which gave the opportunity to see if findings carried across to other units or not.

We have taken great care in our study to validate emerging findings from the interviews. This mitigates to a large extent any misinterpretations that the researchers may have made. Throughout the study, the companies have been involved in validating our analysis: in seminars with the respective contact persons and through recurring interactions with engineers in both formal and informal settings. All three companies approved this paper.

In terms of threats to external validity, the three companies all develop large-scale software for embedded systems and this software is expected to have a long lifetime. Therefore, our findings may not apply to MDE in small companies or for software with a shorter life expectancy. All three companies were transitioning into a combination of agile practices and MDE; although we have some findings related to this aspect, it may be too early to fully understand how these practices work together. Another limitation is that we studied only one unit at each company; other units might have very different experiences of MDE. We have mitigated this to some extent by interviewing a broad spectra of engineers within each organization.

7. CONCLUSIONS

In general our results validate the conclusions of Hutchinson et al. However, in two ares our data refutes their observations: how to introduce MDE so that the engineers feel they do not loose control and in to which extent engineers have the right training for MDE.

Additionally, our own data illustrates how it is possible to involve the domain experts directly in market leading software implementation. The resulting productivity gains are possible due to investments in secondary software that compensates for the domain experts lack of programming skills. In this way the secondary software captures both the domain and the best practices of implementing the same.

Our findings present initial insights in how MDE and agile practices can naturally coexist in the context of embedded software. A future direction of research is then to further explore how domain experts can be included not only in the specification of new features but become directly involved in their implementation - from innovative idea to integration on hardware. A special case of the above is how agile practices and MDE can combine to enable innovation in industrial sectors where external organizations play an important role in the overall development.

Acknowledgments

The authors would like to thank all those who took part in the interviews, including those who facilitated the study at Radio, Cars and Trucks. The study was partly funded by the Software Center, Gothenburg, Sweden.

8. REFERENCES

[1] J. Aranda, D. Damian, and A. Borici. Transition to Model-Driven Engineering - What Is Revolutionary, What Remains the Same? In *MODELS 2012, 15th International Conference on Model Driven Engineering Languages and Systems*, pages 692–708. Springer, October 2012.

[2] P. Baker, S. Loh, and F. Weil. Model-Driven Engineering in a Large Industrial Context – Motorola Case Study. In *Proceedings of the 8th international conference on Model Driven Engineering Languages and Systems*, MoDELS'05, pages 476–491, Berlin, Heidelberg, 2005. Springer-Verlag.

[3] C. M. Christensen. *The innovator's dilemma: when new technologies cause great firms to fail.* Harvard Business School Press, Boston, Massachusetts, USA, 1997.

[4] K. Davis. Methods for studying informal communication. *Journal of Communication*, 28(1):112–116, 1978.

[5] K. DeWalt and B. DeWalt. *Participant Observation: A Guide for Fieldworkers.* Anthropology / Ethnography. Rowman & Littlefield Pub Incorporated, 2002.

[6] B. Glaser and A. Strauss. *The Discovery of Grounded Theory: Strategies for Qualitative Research.* Observations (Chicago, Ill.). Aldine Transaction, 7 edition, 2009.

[7] J. Hutchinson, M. Rouncefield, and J. Whittle. Model-driven Engineering Practices in Industry. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 633–642, New York, NY, USA, 2011. ACM.

[8] J. Hutchinson, J. Whittle, and M. Rouncefield. Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. *Science of Computer Programming*, 2013. Accepted for publication.

[9] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen. Empirical assessment of MDE in industry. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 471–480, New York, NY, USA, 2011. ACM.

[10] A. Kleppe, J. Warmer, and W. Bast. *MDA Explained: The Model Driven Architecture^{TM}: Practice and Promise.* Addison-Wesley Professional, 2005.

[11] A. Kuhn, G. C. Murphy, and C. A. Thompson. An exploratory study of forces and frictions affecting large-scale model-driven development. In *Proceedings of the 15th international conference on Model Driven Engineering Languages and Systems*, MODELS'12, pages 352–367, Berlin, Heidelberg, 2012. Springer-Verlag.

[12] G. Michelson and V. S. Mouly. 'You Didn't Hear it From Us But...': Towards an Understanding of Rumour and Gossip in Organisations. *Australian Journal of Management*, 27(1 suppl):57–65, 2002.

[13] L. Pareto, P. Eriksson, and S. Ehnebom. Concern coverage in base station development: an empirical investigation. *Software and Systems Modeling*, 11(3):409–429, 2012.

[14] C. Robson. *Real World Research: A Resource for Social Scientists and Practitioner-Researchers.* Regional Surveys of the World Series. Blackwell Publishers, 2002.

[15] K. Schwaber and M. Beedle. *Agile Software Development with Scrum.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2001.

[16] A. Strauss and J. Corbin. *Basics of qualitative research: grounded theory procedures and techniques.* Sage Publications, 17 edition, 1990.

[17] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden, and R. Heldal. 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, October 2013.

[18] J. Whittle, M. Rouncefield, and J. Hutchinson. The state of practice in model-driven engineering. *IEEE Software*, 2013.