Transition to model-driven engineering
What is revolutionary, what remains the same?

Jorge Aranda, Daniela Damian, and Arber Borici

University of Victoria
{jaranda, danielad, borici}@uvic.ca

Abstract. A considerable amount of research has been dedicated to bring the vision of model-driven engineering (MDE) to fruition. However, the practical experiences of organizations that transition to MDE are underreported. This paper presents a case study of the organizational consequences experienced by one large organization after transitioning to MDE. We present four findings from our case study. First, MDE brings development closer to the domain experts, but software engineers are still necessary for many tasks. Second, though MDE presents an opportunity to achieve incremental improvements in productivity, the organizational challenges of software development remain unchanged. Third, switching to MDE may disrupt the balance of the organizational structure, creating morale and power problems. Fourth, the cultural and institutional infrastructure of MDE is underdeveloped, and until MDE becomes better established, transitioning organizations need to exert additional adoption efforts. We offer several observations of relevance to researchers and practitioners based on these findings.

1 Introduction

Model-driven engineering (MDE)—the proposal to guide the development of software-intensive systems with model-based abstractions, combining models, process, analysis, and architecture [18, 5]—shows much promise [11]. As abstractions, models could be more efficiently created and modified than lines of code, driving down costs. If the abstractions are appropriate, models could also be easier to understand than code, which would result in an increase of clarity and quality.

As the proceedings of this conference over the years have demonstrated, there has been a considerable amount of research dedicated to make the MDE vision happen. However, there is a dearth of reports on its practical successes and limitations [4], and specifically, about the effect that MDE has had on the software development process of the organizations that adhere to it and on their resulting socio-technical structures. This is despite the fact that such feedback would be considerably valuable to practitioners exploring the feasibility of MDE in their settings and to researchers looking for accounts of the real-life performance of the tools they help create.

In this paper, we report on an interview-based empirical case study of MDE and software development activities in two teams at General Motors, the well-known car manufacturer, which has invested a significant effort in transitioning to MDE in its development process. Our case study allowed us to explore several benefits and drawbacks of MDE, and to gain insights into the ways in which coordination dynamics are altered by the introduction of MDE into the development process. We present these findings after the following discussions on previous field studies of MDE and on the methodological details of our own fieldwork.

2 Related Work

Although there has been much research into analyzing the formal aspects of MDE proposals, modelling languages, and model transformation techniques, as well as into evaluating the comprehensibility of several model representations, there are few accounts of what happens when projects actually transition into this engineering approach [25]. The social, organizational, and political implications of a technology as potentially disruptive as MDE are large, yet practitioners and researchers have little information to help guide them on this process.

There are, however, several notable industrial field studies that report on the factors that contribute to the success or failure of MDE adoption in large software organizations. Hutchinson *et al.* [16, 17] report on an interview-based study in which they explore the experiences using and adapting to MDE. Among other things, they warn against adopting MDE wholesale, recommending instead a progressive and iterative approach. They also warn against transitioning without proper organizational support or without motivation from the developers.

Similarly, Staron [28] reports on different experiences of two companies transitioning to MDE. One backed out, the other continued. Staron argues that the state of the art (in 2006) in MDE technology did not support an efficient transition, and that the problem was exacerbated if the transitioning company had to maintain a large base of legacy code. This issue was previously reported by MacDonald *et al.* [22], who claim that MDE does not lead to an improvement in efficiency, effectiveness, or productivity—at least not in the context of projects with a large amount of legacy code.

Other studies of adoption of MDE in industrial settings include Cheng *et al.* [6] who report on the adoption of automated analysis of object-oriented design models and their effect on software design quality. UML class diagrams from two large industrial models at different developmental stages are employed in a utility analysis carried with DesignAdvisor. The authors find that the quantity of severe errors increases proportionally with design complexity and that the utility of design patters greatly contributed to lower the number of errors in the models wherein the patterns were used.

Kulkarni and colleagues [20] describe strategies for scaling up MDE at Tata Consulting and describe elements of their MDE infrastructure, as well as their experience of using it to deliver large business applications over a period of 15 years.

Closest to our study, the work of Baker *et al.* [2] reveals the impact of MDE adoption at a large organization, Motorola. However, their findings are mostly about tool and language feasibility to support large-scale development. They do mention "team inexperience" as an issue Motorola experienced in their deployment of MDE. By inexperience they mean lack of a well defined process, missing skill sets, and organizational lack of flexibility.

We note that, in parallel to this paper, Kuhn *et al.* [19] drew from the same pool of participants we interviewed to study complementary and non-overlapping research questions: while they focused on human aspects primarily at the individual level, considering cognitive aspects of using MDE technology, we focused on human aspects focused on the organizational consequences of MDE adoption. Kuhn *et al.* found several forces and points of friction with respect to cognitive issues of MDE technology. They discovered that model diffing should be a key feature of MDE tools, that there is a need for problem-specific expressivity, that there continues to be a need for exploration late in the product development cycle, and that point-to-point traceability is a fundamental need that becomes even more acute under MDE.

The majority of the related work, to the best of our knowledge, is concerned with the adoptability of MDE in industry, whereas our report focuses on the change of dynamics of an organization that has determined to adopt MDE. Our approach to studying MDE is different—we focus not on factors of adoption, but on the organizational *consequences* of adoption. We describe our research questions in the next section and the case study design and findings in the remainder of the paper.

3 Research Questions

To complement the existing empirical evidence about MDE in industry, the focus of our investigation was on the issues related to consequences in the development processes and the socio-technical structures enabled or affected by the introduction of MDE in large organizations. We formulated the following three research questions to guide our data collection and analysis:

RQ1: How does MDE adoption look like in practice in large-scale projects? To what extent does MDE alter the development landscape?

RQ2: How do the coordination dynamics and the division of labour change under a transition to MDE?

RQ3: What issues, beyond those reported previously in the literature, are relevant for organizations considering an MDE transition?

4 Case Study Design and Execution

We performed an empirical study of the organizational and coordination consequences of transitioning to MDE. A data-rich qualitative study was the most appropriate for our investigation, given the contextualized and multivariate nature of the phenomena that we wished to study.

We followed Yin's case study methodology [30]. Specifically, we executed an exploratory single-case case study. In an exploratory case study, as in other qualitative empirical methodologies such as Grounded Theory [13], one begins with a set of research questions and no hypotheses or propositions to test. In contrast with Grounded Theory, the goal of an exploratory study is not to produce a new theory based on the data. Instead, the researcher collects data from a previously under-explored domain with the goal of reporting insights that can be tested as hypotheses in future studies.

We collected our data from a single organization: General Motors (GM). As part of a larger research project on MDE in industry, the first author of this paper, Aranda, visited the offices of GM in Michigan, along with two researchers from the University of British Columbia, who were interested in studying issues of cognition and MDE. These three researchers conducted a total of ten interviews together, each lasting about two hours, with control engineers, software engineers, and managers of two teams (and in two campuses) at GM. Table 1 summarizes basic information from the interviewees. The first half of the interviews consisted of questions pertinent to this paper and was directed by Aranda; the second half, directed by the UBC researchers, was concerned with cognitive issues that will not be reported here.

The interviews were semi-structured. The interview guide is not included in this paper for space reasons, but we have made it available online [1]. All the interviews but one were audio recorded. A single researcher (Aranda) annotated and coded them, and analyzed the interviews and notes guided by the research questions stated above. We looked for robust findings: insights supported by the observations of several interviewees, as opposed to single-source reports. All the findings reported here are robust in this sense.

4.1 Details from the study site

For this paper, we studied the people and activities relating to software development, testing, management, and process definition in two product development groups at GM. For a reader not acquainted with modern automobile manufacturing, studying software development at GM may appear odd: this is a well-known automobile manufacturing corporation, not a software company. In truth, GM (as other automobile manufacturers) is now a hardware and software development company, and to an outsider it may be difficult to imagine the extent to which software controls its products. GM cars increasingly rely on software to perform their functionality, and correspondingly, GM increasingly depends on its software development groups. Furthermore, GM transitioned to MDE in the two years previous to our data collection, making the organization a prime candidate for the study of the consequences of MDE adoption. Today, most software at GM is developed in model-driven tools (such as MATLAB's Simulink and IBM Rational Rhapsody). One manager and process designer explained it this way:

Table 1. Interviewees for our study

ID	Team	Position
P1	Core	Control Engineer (Algorithm Development Engineer)
P2	Core	Control Engineer (Algorithm Development Engineer)
P3	Core	Manager (Software Engineering)
P4	Core	Software Engineer
P5	Core	Software Engineer
P6	Aux	Software Design Lead
P7	Aux	Software Engineer
P8	Aux	Software Engineer
P9	Aux	Software Readiness Engineer (Testing)
P10	N/A	Manager (Process Definition)

> *We made the rule that the model is the code; you want to make a change, you change the model, you don't change the code. And then you just regenerate.* —P10, Manager

However, as we will see, the MDE transition is still being negotiated in terms of tool adoption, process agreement, and role definition.

As we stated above, the interviews span two product development groups. One of them develops one type of core driving features[1] (we will henceforth refer to it as the "core functionality group"), the other develops a subset of auxiliary functionality (we refer to it as the "auxiliary functionality group"). Both groups follow the same high-level software process. It is based on a V development model [23], with a workflow that goes down from requirements definition to implementation and then back up to testing, but it is adapted to account for the rest of the hardware demands of product design. Specifically, the organization places a much greater emphasis on what it refers to as "the Physics" (the equations and other engineering considerations required in the design of automobiles), which need to be embodied by code, and in calibrating and testing the software in the particular hardware in which it will be run.

The core functionality group is collocated at the building level, and it is divided in two main teams, which we will call Team A and Team B. Broadly, Team A workers are in charge of designing the equations and their interactions with other features, while Team B workers are in charge of implementation.

[1] We are not more precise purposefully, to obfuscate some internal details of the GM structure and of the teams we studied. By "one type of core driving features" we mean features that help a car perform its essential transportation functionality.

There are other groups and roles as well, but they are more detached from MDE and from development in general, and will not be considered here.

The auxiliary functionality group is globally distributed: it has engineers in one of the main company campuses in Michigan and a team of offshore engineers in Asia. Although the group also has people in charge of designing the equations and others in charge of their implementation, they are not separated by teams in the way that the core functionality group is. This group also has testers playing a more prominent part in feature development.

5 Findings

In this section we present four findings on transitioning to MDE that we believe should be of interest for researchers and practitioners in the area.

First, MDE succeeds in bringing software development closer to the subject matter experts [25], but an important (if at times menial) subset of software development activities still needs to be performed by people other than the subject matter experts—people with significant software development skills. These software engineers still play an important role under an MDE structure.

Second, the basic processes and challenges of organizational structure and interaction remain unchanged with MDE: software development uses largely the same organizational forms [29] and processes as traditional software development, and it is still difficult to coordinate, to clarify requirements, and to get teams of professionals to deliver high quality software. In other words, though MDE can bring important benefits under some situations, by abstracting away some software development obstacles [11], it presents at best an incremental improvement in software development, in the case we studied.

Third, switching to MDE may disrupt the organizational structure and alter its balance, which creates morale and power problems that transitioning groups should consider.

Fourth, MDE represents a migration to an underpopulated cultural and institutional landscape. The tools, training, and expectations of professionals under MDE are not as well developed and established as those under more traditional software development dynamics. We expect MDE transitions to be generally problematic for this reason, now and until the cultural and institutional infrastructure of MDE becomes better established.

The following subsections expand on each of these findings.

5.1 Bringing development closer to the subject matter experts

One of the most ambitious visions of MDE is that subject matter experts (or, in lieu of them, requirements engineers or designers) will be able to model the behaviour they wish their software artifacts to exhibit, using an accessible and appropriate abstraction, most probably represented in diagrammatic form, and that just with the push of a button their code will be auto-generated for them and ready to deploy or use. Such a vision would, of course, bring in huge savings

to the software development process in terms of, for instance, efficiency, quality, and clarity [24]. Just as today nobody uses assembly language to program their software if they can avoid it, we will, at some point in the near future, look back to lines of code as an antiquated, needlessly detailed, and cumbersome mechanism to capture the behaviour of software. Of course, this MDE vision does not need to be realized in full to start yielding benefits. If developing software using models is beneficial, a partial application might well bring partial benefits, too.

In practice, in GM to date, MDE has certainly brought development closer to the subject matter experts at work. In some groups within GM, control engineers (the mechanical or electrical engineers in charge of facing the hardware and other physical and design constraints, and of describing and supervising the production of automobile features) can now work with their simulations and, with relative ease, auto-generate the code that will be deployed. This is in contrast to their previous dynamic, in which control engineers would specify their requirements, determine the equations that should be implemented if needed, and communicate them to software engineers, who would be tasked with implementing them in full.

The extent to which this new dynamic is established varies across GM, partly due to GM's flexibility in allowing different groups to transition towards MDE according to their contexts. Both team and personal factors seem to affect the dynamic's variation. At one extreme, for some collaborations between control and software engineers, the control engineer now models all the desired functionality, auto-generates the code, and passes it on to the software engineer to do some more menial work—ensuring that the model adheres to standards, that its integration with other code is handled properly, that its functionality satisfies the description of the work item appropriately, and so on:

> *I prefer to just do it all myself. I do all the algorithm design and [the software engineers] do checks and coding standards. I do the work and [the software engineer] just tracks it.* —P1 (Control Engineer)

At the other extreme, there has been no approximation of development to subject matter experts at all: the control engineer continues to specify requirements and functionality in free text, or even verbally, and the software engineer implements them using a modelling tool:

> *I'm the dullest knife in the drawer when it comes to modelling simulation and coding. I know how the physics work but I depend on these young kids to make it manifest in software. [...] I would like a better separation of responsibilities. I would like to work on requirements interfacing on design, and somebody else create software which is the manifestation of the algorithm and then get back to me and show me what they've done. That's when the system works best.* —P2 (Control Engineer)

In between there are other variations. Some control engineers make their model changes in a mock version of the models, and the software engineers use those mock changes as their blueprint to implement the real changes. Other

control engineers only model some component of their features of particular interest, and leave the rest to the software engineers.

Beyond the extent to which some control engineers have successfully engaged in developing software with a modelling language, there are several tasks that will be difficult to bring under their scope. In other words, a software development "middle man" might be still needed in an MDE framework. This is because domain experts are unlikely to have the software development training, nor the time, nor the professional inclination, to involve themselves with implementation issues. An organization developing software at this scale requires modelling conventions and standards, quality controls, hardware-software calibrations, integration conflict resolutions, and involvement in necessarily bureaucratic processes such as change management boards. Some of these issues may be resolved with appropriate tooling, and others may be addressed by managerial mandate, but this does not mean that the solutions are simple, painless, or even feasible in the short term, and transitioning organizations need to account for this.

5.2 Persistence of the traditional organizational forms

The increased closeness of domain experts to software development work brought about by MDE raises the question of the extent to which MDE has revolutionized the software development landscape. For a long time, organizational scientists have observed that the various groups that belong to the same industry tend to follow similar patterns of interaction, to structure themselves in similar ways, and to encounter common problems and challenges [26]. In other words, they have the same *organizational form* [29]. Revolutionary technologies can bring about new organizational forms, with different challenges and strategies [14].

At the outset, it is unclear whether MDE can be one such revolutionary technology. On one hand, MDE could upset the whole communication and coordination structure, bringing many roles into obsolescence, and eliminating the need for time-consuming and inefficient structures. On the other hand, one could construe MDE more like a change in representation (that is, a change from traditional coding to modelling), and it is irrational to expect it to tackle the fundamental problems of software development [11].

We found that the latter is indeed the case at GM: while MDE *does* bring benefits, it cannot be considered a *revolutionary* solution with respect to the organizational challenges, processes, and structures of software development work. As we mention above, MDE brings development closer to subject matter experts, in abstractions that are closer to their domain, but these experts still must overcome the difficulties of defining, negotiating, and clarifying their activities [7, 8], and of coordinating with other professionals when their work outputs diverge from expectations in the real world. In fact, coordination, which has been increasingly recognized as a central problem in software development [15], seems to be as challenging for GM under MDE as for other large organizations using traditional development approaches. It is still hard to coordinate, especially with remote sites that may be missing contextual information, and with which

communication is necessarily slower, as reported in previous studies of requirements engineering in global software teams [9]. Engineers still need to clarify requirements with each other. Parallel streams of work mean that several people tweaking the same pieces of code brings out conflicts.

In other words, what we found striking was how little difference there was between GM and other organizations' forms, despite the fact that GM now adheres to an approach that is in some respects radically different from the traditional one. The tools and the language are different, but the organizational structure and challenges remain largely the same. While MDE may provide productivity gains, judging from this case it does not seem to lead to a radically different work arrangement for the software industry.

We found two important caveats to this observation, however. We will deal with them in the coming sections.

5.3 Coordination and division of labour

For GM, the switch to MDE caused an interesting organizational disruption. As stated above, before the introduction of MDE, the core functionality group that we studied had settled into a bipartite division of labour, organized by domain: one team (Team A) tackled "the Physics" involved in designing automobile features, and a second team (Team B) addressed the software implementation. One person from each team coupled with each other to work on a feature together. The Team A engineers (the Control Engineers) worked on the equations necessary for the appropriate functioning of their features, as well as on the hardware constraints and the interdependencies with other features. The Team B engineers (the Software Engineers), in turn, focused on "hand-coding" the equations and constraints from their partners into software. They also unit tested their own code, ensured it adhered to their conventions, and were responsible for any changes made to it. In conventional terms, and simplifying the collaboration, the Team A engineer in each couple worked on the analysis and design of a feature, while the Team B engineer worked on its development.

Eventually, MDE tools and processes were introduced to the group. The tools allowed engineers to model the behaviour of their systems graphically, and to auto-generate code that implemented the desired behaviour. The auto-generated code rarely needed to be rewritten. Furthermore, the models could be initially tested in a simulated environment, relaxing GM's dependence on physical tests with real hardware.

This introduction of MDE tools, however, brought a disruption in the work arrangement we described above. The Team A engineer now prepared "the Physics" as a computer model, and auto-generated the code that implements it, but the Team B engineer was still necessary: there were aspects of software development that Team A engineers did not have the training, the time, nor the inclination to tackle: issues such as coding conventions, unit testing, versioning, and code dependencies. The fact that these were largely clerical issues did not escape members of either team. For instance, according to one Team A engineer:

There's a software engineer who manages that [module]. We're supposed to be working with them to design stuff, but they really are acting like bookkeepers and code checkers rather than designers, and they don't seem to be interested in what we're doing. (...) probably because their job is boring. —P1 (Control Engineer)

In effect, the balance in the partnership was lost: Team A increased its conceptual power and dismissed the work of Team B, whereas Team B exercised its remaining structural and technical power [21] as a gateway through which all changes must flow. The collaboration, according to parties on both sides, was at an all-time low.

The group attempted to solve this problem. Its main strategy was to loosen the division of labour between both teams, so that Team A engineers would become more involved with the implementation issues of their models, and Team B engineers would become competent about the mechanical domain and could contribute to the analysis and design of their features. In partnerships with the right motivation and mentorship efforts, this approach worked. For instance, according to one Team B engineer:

Sometimes I do development, that's one of the things we tried to do when we switched to models, to have [Team A] and [Team B] to take on development. Every now and then we help [Team A], others we do the changes ourselves. I've done a couple of those. —P4 (Software Engineer)

In other partnerships, however, the pattern we observed (which we named the *Modeller-Clerk* pattern) remains. At the time of our interviews, GM continued to negotiate the transition to its new normality.

5.4 Cultural and institutional problems

Cultural and institutional forces exert a powerful influence over the activities and decisions of a software organization, though our community tends to overlook them. Many problems of adoption, adaptation, morale, and process redefinition in software development can be traced to institutionalization issues. In order to make our point, however, we first cover the basics of institutionalization theory, as we find it is not well known in our field.

Briefly, and simplifying institutionalization theory [10], in our day-to-day activities we are faced with numerous decisions to make and data to ponder. In general, an efficient strategy to deal with this overwhelming abundance of decisions and data is to repeat the behaviours that worked in the past to deal with similar situations [27]. We choose once (which text editor to use, how to respond to a bug report, when to hold team meetings), and, as long as our choice was at least somewhat successful, every time life throws a similar scenario at us, we respond in the same way.

An essential point of institutionalization theory is that this habit-forming tendency we have extends naturally to our peer interactions. Once a team has

satisfactorily dealt with a situation, it is likely to replicate the patterns of interaction that led to its resolution, and the more like situations it tackles, the more the pattern is reinforced: each team member learns what to expect from her colleagues, and would be surprised, or perhaps angered, if a teammate deviates from the established behaviour [3, 12]. The team pattern is recognized as *the* way to deal with the situation—it is institutionalized.

Furthermore, these patterns extend beyond single teams and into the wider society. A team requires its recruits to have certain skills in order to perform its patterns appropriately, and it pushes educational institutions to train their students to meet its criteria. (Alternatively, educational institutions can push other organizations to accept *their* definition of what counts as valuable skills to have, and the other organizations shift their patterns accordingly.) In time, most aspects of the domain become institutionalized: what counts as knowledge and accepted wisdom, what are proper career paths, what are the tools of the trade, what are the roles people can take, and which ways do they interact with their peers. As long as a kind of situation arises with frequency, the community will coalesce into a set of institutional forms to deal with it.

One can easily see, after these considerations, that any kind of significant organizational change is difficult. It entails the breakdown of many small negotiated successes, it casts the organization's shared understanding into disarray, and it can even cause strong emotional reactions from people forced to reexamine and rebuild many aspects of their professional lives.

We found that MDE adoption falls under the kind of significant organizational change that causes these kinds of reactions at GM. In many respects, MDE is still a pioneering strategy, and the mainstream of the software development field is not familiar with the tools and the practices required for it.

To begin with, tooling capabilities and modelling conventions are still not comparable to their traditional-coding counterparts:

> *One complaint was that changes took very long. Some people would say, if it takes me one hour to make a change in C code, it will take me four hours in MATLAB.* —P4 (Software Engineer)

But the organizational toll may be a greater problem. For GM, hiring software engineers from the mainstream would be difficult; it chooses to hire mostly electrical or computer engineers, which not only have a better understanding of the domain, but may be more familiar with modelling tools such as Simulink. However, these engineers may be less attuned to software development practices and habits. Even then, many (if not most) of GM's engineers have had to learn MDE on the job.

Other engineers have been building automotive software for years, or even decades, and among them resistance to MDE may be greater. They are used to coding in C, and they are effective and efficient with it. The new approach asks them to move towards tools they see as inferior, rightly or not. They find that both small and large code changes take a longer time than they should. They are not used to some of the abstractions embodied by their new tools. In short,

despite their wealth of expertise, both in the automotive and in the software domains, they suddenly feel incompetent. The new cultural and institutional infrastructure is not yet set up to exploit this wealth of expertise properly:

> *It was clear that for people who'd been here for a while... it was all mature, things worked, all that. [With] the MATLAB environment, we needed to redesign the process [...]. For some new people, they were OK with that, others are still struggling.* —P3 (Manager)

Incidentally, the new approach also blocks the old one from functioning. Once a module transitions to MDE, the auto-generated C code, by all accounts, is terrible to work with directly. Furthermore, since the organization has committed to a model *driven* strategy, its tools increasingly enforce a process that requires model-based activity to move items towards their resolution.

It is unclear what an organization at this stage can do other than mitigation. GM is aware that rolling back to traditional coding appears to be an expensive solution—it would require the translation of large swaths of models into human-*readable* C code, and the re-training of new engineers who are now used to working primarily with models. A more likely scenario is to carry forward, absorbing the costs of institutionalization gradually, in the hope that the new approach will yield greater benefits than the old.

This is, of course, a consequence of being at "the bleeding edge" of innovation. These are not intractable problems. If MDE catches on, most of these issues (tooling, process, conventions, education) will dissipate. Our study merely points out that, to date, these cultural and institutional issues still stand forcefully in the way of a smooth transition towards MDE.

6 Discussion

Any technological approach that touches so many aspects of a socio-technical structure as MDE does for software development deserves a close scrutiny of its consequences. In the case of GM's adoption of MDE, we found a set of organizational benefits and problems that we think are of relevance for other organizations considering an MDE transition and to the research community.

At this point, it is useful to revisit our original research questions and provide some answers based on our exploratory study.

RQ1: *How does MDE adoption look like in practice in large-scale projects? To what extent does MDE alter the development landscape?*

Though technically MDE presents several interesting innovations and challenges, judging from our data, large-scale projects that adopt MDE look mostly similar to more traditional large-scale projects. The mainstream structure of software development teams and processes remains in place. That is, MDE brought a shift in emphasis to the activities of GM professionals, a shift that allows Control Engineers to design automotive software features in a tool and in a manner convergent with the product that will eventually be released, but it did not eliminate

the need for the software development structure (of software engineers, testers, maintainers, and the like) that supports traditional software organizations.

Furthermore, we found that the organization we studied, under MDE, still struggles with the same issues that most traditional software organizations struggle with. Under the right circumstances, MDE may help relieve some problems in software development, but it leaves its basic organizational form unchanged.

RQ2: *How do the coordination dynamics and the division of labour change under a transition to MDE?*

Some inter-team coordination dynamics change after a transition to MDE. Models are more used for communication and coordination, although there is still significant reliance on face-to-face communication to clear potential misunderstandings, and on natural language to explain several aspects of architecture and functionality.

MDE, however, may be expected to alter the current division of labour in a transitioning organization, and the resulting imbalance may be hard to negotiate, at least temporarily.

RQ3: *What issues, beyond those reported previously in the literature, are relevant for organizations considering an MDE transition?*

The division of labour imbalance issue should be relevant for organizations considering transitioning to MDE. Another issue we discovered is the extent to which the transitioning organization will find frictions as it migrates and attempts to re-establish itself in an underdeveloped cultural and institutional setting. Previous studies [2] have pointed out that one of the most important challenges for organizations adopting MDE is that change is difficult. Our results go a step forward, by helping to explain in what sense is organizational change difficult, and for which reasons (an underdeveloped infrastructure for the new institutions, and a disruption with the old and well-established institutions).

6.1 Threats to Validity

As with any empirical results, one should exercise caution in the interpretation of our findings. We discuss three threats to their validity that we were concerned with as we performed our data collection and analysis: their generalizability, the number of interviews we performed, and the fact that we only performed interviews after the transition to MDE was well underway.

First, there is the natural question of whether one can generalize from a single case study (or indeed from a single study of any kind) of a single organization to the whole MDE field. The short answer is that one cannot. In a setting as complex as that of organizational and technological change, there are too many confounding factors, qualifications, and provisions. The experiences of other transitioning organizations may be quite different from those we observed at GM. Nevertheless, we believe there is much value in reports such as these, for two reasons: first, because they provide rich information when there was little or none, and second, because they begin the scientist's task of uncovering the *reasons* for which the probable consequences of MDE adoption will be experienced. For instance, to say that "organizational change is hard" is trivial, but

to uncover *in what ways* it is hard for MDE, what are the likely *causes* for the hardships, and *when* are they likely to dissipate, is valuable. Our report does not provide certain answers to these questions, but we believe it helps reach them despite its necessarily limited scope.

Second, we performed only ten interviews, and these were constrained to only two teams within the organization. Of course, a greater number of interviews and of teams would be preferable. We were fortunate, however, to be able to interview professionals in a wide variety of positions, which provided us with a multiple number of views on the inner workings of the organization. The interviews themselves, as can be inferred from the findings we reported above, were frank and wide-ranging. Although further studies at this or other organizations should help us improve the confidence on our findings, we think they are appropriate for the current, exploratory stage of our research.

Finally, we were only able to perform interviews after the transition to MDE was a "done thing". We did not interview professionals before the transition began. We had to resort to their recollections of how things were different before MDE, which may be biased by current events. We had to come to terms with this necessary evil, given the otherwise unprecedented opportunity we had in having access to a transitioning MDE organization and in being able to ask its members both wide and sensitive questions on their work life.

6.2 Implications

For researchers: Our findings point towards several research directions that we believe are worth pursuing. First, they show that studying the practical consequences of MDE is an important endeavour, and they call for further studies to build a richer experience bank and improve our generalization power.

Second, the disruption of organizational balance is worth exploring. The extent to which it is unique to GM, or to which it can be avoided with careful organizational planning and design is unclear. The direct and indirect costs that the transitioning organization needs to absorb to deal with it are similarly unclear.

Third, in the case that improvements brought about by MDE are not radical (as our GM data suggests), there is the question of the conditions under which a transition to MDE is advisable, and the realistic advantages that the transitioning organization can be expected to reap from its efforts. We note that these viability questions are severely under explored in the MDE literature. Further research should help provide answers to these questions.

For practitioners: Naturally, a transition to MDE will have costs and benefits—the questions, as usual, are what costs and benefits are there, and under which conditions the latter overcome the former. We did not delve into the technical costs and benefits of the model representation (though our colleagues from UBC have), but organizationally speaking, we found that introducing MDE does, in effect, bring development closer to subject matter experts, but does not eliminate the need for work tasks that domain experts are not qualified or expected to perform. If the GM experience is indicative of more general cases,

it is unlikely that there will be savings from staff reductions or from a leaner organizational structure.

Indeed, we found that the software development process remains largely indistinguishable from that of traditional software development, with its same challenges and issues, though organizationally speaking the introduction of MDE may alter the balance in undesirable ways.

Finally, while MDE has matured over the years to the point where it can sustain the development of products of critical importance and of high quality, as GM's automotive software needs to be, its institutional infrastructure is still underdeveloped, and transitioning practitioners will find that, both technically and organizationally, many things they took for granted need to be built again.

7 Conclusion

As the viability of MDE continues to increase, we will need more field studies that report on the consequences of their adoption by leading practitioners. Only by identifying the current strengths and weaknesses of MDE can researchers expect to refine the MDE approach to fulfill its vision.

By pointing out several positive and negative organizational consequences of MDE adoption, this paper is a step in that direction. MDE brings development closer to subject-matter experts, and although it does not alter the organizational form of software development groups, it shifts the balance of power and the division of labour within them in ways that may be conflictive, at least temporarily. Furthermore, adoption leads to institutional and cultural frictions that will not be resolved in the short term. We argue that these organizational consequences are far from negligible, and that transitioning organizations and researchers need to consider them in their efforts to advance this domain.

Acknowledgements

We are grateful to our interviewees for their time and information, and to Joe D'Ambrosio and Shige Wang at General Motors for their dedicated coordination to help us execute this study. We thank Adrian Kuhn and C. Albert Thompson for their collaboration during our joint visit to GM. Funding for this study was provided by the NECSIS Network.

References

1. J. Aranda and D. Damian. *Interview Guide for GM October - November 2011 Visit*. `URL:http://home.segal.uvic.ca/~jorge/materials/guide.pdf`, 2012.
2. P. Baker, S. Loh, and F. Weil. Model-driven engineering in a large industrial context - Motorola case study. In *Proceedings of the Model Driven Engineering Languages and Systems 2005 Conference (MoDELS 2005)*, 2005.
3. P. L. Berger and T. Luckmann. *The Social Construction of Reality*. Doubleday Anchor, 1967.

4. S. Beydeda, M. Book, and V. Gruhn. *Model-Driven Software Development*. Springer, 2005.
5. J. Bézivin. Model-driven engineering: An emerging technical space. In *Generative and Transformational Techniques in Software Engineering (GTTSE)*, 2005.
6. B. Cheng, R. Stephenson, and B. Berenbach. Lessons learned from automated analysis of industrial UML class models (an experience report). In L. Briand and C. Williams, editors, *Model Driven Engineering Languages and Systems*, volume 3713 of *Lecture Notes in Computer Science*, pages 324–338. Springer Berlin / Heidelberg, 2005.
7. B. H. C. Cheng and J. M. Atlee. Research directions in requirements engineering. In *FOSE 2007: Future of Software Engineering*, 2007.
8. B. Curtis, H. Krasner, and N. Iscoe. A field study of the software design process for large systems. *Communications of the ACM*, 31(11):1268–1287, 1988.
9. D. Damian and D. Zowghi. Requirements engineering challenges in multi-site software development organizations. *Requirements Engineering Journal*, 8(3):149–160, 2003.
10. P. J. DiMaggio and W. W. Powell. Introduction. In W. W. Powell and P. J. DiMaggio, editors, *The New Institutionalism in Organizational Analysis*, chapter 1, pages 1–38. University of Chicago Press, 1991.
11. R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *FOSE 2007: Future of Software Engineering*, 2007.
12. A. Giddens. *The Constitution of Society*. Polity Press, 1984.
13. B. Glaser. *Basics of Grounded Theory Analysis*. Sociology Press, 1992.
14. M. T. Hannan and J. Freeman. *Organizational Ecology*. Harvard, 1989.
15. J. D. Herbsleb. Global software engineering: The future of socio-technical coordination. In *FOSE 2007: Future of Software Engineering*, 2007.
16. J. Hutchinson, M. Rouncefield, and J. Whittle. Model-driven engineering practices in industry. In *ICSE '11: Proceedings of the 33rd International Conference on Software Engineering*, Honolulu, HI, USA, 2011.
17. J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen. Empirical assessment of MDE in industry. In *ICSE '11: Proceedings of the 33rd International Conference on Software Engineering*, Honolulu, HI, USA, 2011.
18. S. Kent. Model driven engineering. In *Third International Conference on Integrated Formal Methods (IFM 2002)*, 2002.
19. A. Kuhn, G. C. Murphy, and C. A. Thompson. An exploratory study of forces and frictions affecting large-scale model-driven development. In *MODELS'12*, 2012. Under submission.
20. V. Kulkarni, S. Reddy, and A. Rajbhoj. Scaling up model driven engineering - experience and lessons learnt. In *Proceedings of MoDELS 2010*, 2010.
21. L. A. Macaulay. *Requirements Engineering*. Springer, 1996.
22. A. MacDonald, D. Russell, and B. Atchison. Model-driven development within a legacy system: an industrial experience report. In *Proceedings of the 2005 Australian Software Engineering Conference*, 2005.
23. R. Pressman. *Software Engineering: A Practitioner's Approach*. McGraw-Hill, 2004.
24. J. Rech and C. Bunse. *Model-Driven Software Development: Integrating Quality Assurance*. Idea Group Inc., 2009.
25. D. C. Schmidt. Model driven engineering. *IEEE Computer*, 39(2), 2006.
26. R. Scott and G. F. Davis. *Organizations and Organizing: Rational, Natural, and Open System Perspectives*. Prentice Hall, 2007.

27. W. R. Scott. *Institutions and Organizations: Ideas and Interests*. Sage, 2008.
28. M. Staron. Adopting model driven software development in industry - a case study at two companies. In *Proceedings of MoDELS 2006*, 2006.
29. A. L. Stinchcombe. Social structure and organizations. In J. G. March, editor, *Handbook of Organizations*, pages 142–193. Rand McNally, 1965.
30. R. K. Yin. *Case Study Research: Design and Methods (3rd Edition)*. Sage, 2003.