A Field Study of Modellers at Work

Eirini Kalliamvakou
University of Victoria
Victoria, BC, Canada
ikaliam@uvic.ca

Marc Palyart
University of British Columbia
Vancouver, BC, Canada
mpalyart@cs.ubc.ca

Gail C. Murphy
University of British Columbia
Vancouver, BC, Canada
murphy@cs.ubc.ca

Daniela Damian
University of Victoria
Victoria, BC, Canada
daniela@cs.uvic.ca

***Abstract*—Knowing the impact in real settings of a software development approach is beneficial to both the industry and the research community. In this paper, we report on a field study we conducted at General Motors with the intent of understanding whether the organization was achieving some of the intended benefits of introducing a model-driven approach to software engineering. This study involved both observations and interviews. We found that several factors are still limiting the productivity increase claimed by MDE.**

I. Introduction

All too often, software technologies are deployed into organizations with an intent to improve some aspect of development, such as to improve the speed of software development, but without any systematic follow-up as to whether the intended benefits are realized. Performing appropriate systematic investigations as to whether intended improvements are realized is challenging. For instance, if a systematic investigation is attempted too early after the technology is deployed, the organization may still be going through adoption pains that mask whether the technology might indeed satisfy the intention of introducing the technology.

In this paper, we report on a field study we conducted at General Motors, a large automotive company, with the intent of understanding whether the organization was achieving some of the intended benefits of introducing a model-oriented approach to software engineering. By model-oriented approach, we mean a software development process in which the creators of software components use modelling tools that generate traditional code rather than having the creators (named modellers in this paper) write code directly.

The field study involved both observations and interviews of members of a single group within the large organization. As part of the study, we observed four modellers at work between two and four hours each and we interviewed seven members of the team for an average of 50 minutes each. This study builds on earlier work we performed at the same organization in which we conducted interviews about how modelling was performed across a broader set of teams [6]. Our intent with this field study was to understand in more detail how model-oriented software engineering was performed and to help elaborate on issues that were raised in the earlier broad interviews. Through this study we found:

- a surprisingly high amount of time, approximately 20%, is spent in non-modelling, code oriented activities,
- many points of friction remain in the tooling, such as a lack of support for partial models and cumbersome testing and debugging approaches, and
- the organization structure which follows the product architecture create awareness challenges.

By identifying these detailed aspects of how modelling work is performed—ways in which modelling is achieving intended benefits and ways in which modelling is hampered—an organization can focus efforts on further improving the approach to gain desired intended benefits.

We begin by describing the context of the study and how it relates to our earlier work at this organization (Section II). We then discuss details of the study method (Section III), before describing results (Section IV), and threats to the validity of the results (Section V). We relate our findings to previous efforts (Section VI) before summarizing (Section VII).

II. Background

There are many advantages to studying software development teams at work in industrial settings, such as understanding what non-technical forces might affect technical decisions [1]. However, interpreting any findings requires an understanding of the context in which the work occurs. We provide a brief overview of the context in which the modellers we study work (Section II-A) before summarizing our earlier interview study conducted within the same organization (Section II-B).

A. Study Context

The field study was conducted in one thirty-person team that builds software for a controller that is deployed on an automobile. The team is split between two highly distributed geographical locations with twelve members in one location and eighteen in the other location. Our study involved members in the former location. As described by the team members, the part of the team we studied included:

- an Engineering Group Manager who is responsible for overseeing all of the technical deliverables of the team and delivering error-free, reusable software and who also provides HR and career management support for the team.
- a Software Design Lead who provides technical leadership for the team members and who serves as the point of contact with other teams in the organization. The design lead also had responsibility for two software components himself.

- Software Engineers who implement features and changes to software components. We refer to these software engineers as "modelers". They express the structure and behaviour of the software components they build using a combination of IBM Rhapsody's tool and MatLab's Simulink tool.
- a Software Integration Lead who integrates software components.

Testers are also part of the team but were located in the second location and were not part of the study. The modellers are guided in their work by requirements written by Strategists, who use IBM DOORS.

A modeller begins work by receiving a notification through email that there is a ticket for them on the change management tool, IBM RTC. The ticket starts as a work ticket and points to DOORS for the relevant requirements description. After the modellers have acquired and read the requirement description they communicate with the relevant strategist to do analysis and get clarifications. Once there is a commonly agreed understanding of the requirement then an implementation ticket is issued for the modeller. Subsequently the modeller uses either the Simulink or IBM Rhapsody tool to produce their model, implementing the logic that corresponds to the feature to which they are assigned. At this stage, the modellers also use signals coming from other components as inputs to the component on which they are working. The Rhapsody model is updated after modelling is done, followed by code generation and unit testing, which is designed to check the fulfilment of the strategy intent, not code coverage. The generated code is bench tested on control units by a Readiness team and then on the vehicle by a Verification team. At any point when there are errors indicated as a result of the testing, the ticket is referred back to the modeller for resolving.

B. Earlier Interviews

In 2011, interviews were conducted with 20 individuals at General Motors: 12 engineers and 8 managers. Each interview was broad, ranging from descriptions of the work performed by those interviewed to organizational factors that affect the performance of the work.

On the technical side, four forces and five points of friction were identified that affected the use of model-driven engineering. For instance, it was identified that one force driving the process was the exploratory nature of inventing new vehicle control algorithms, yet a point of friction in the process was a lack of tool support for testing model changes at runtime when testing on actual vehicles [6]. In the study we report on in this paper, we are able to delve into more detailed aspects of the toolchain used by a modeller and where time is spent.

On the organizational side, the previous interviews identified a number of points, including that model-driven engineering does not alter many of the processes and challenges of existing organizational structures [1]. In the study we report on in this paper, we are able to identify more specific outcomes of existing structures, such as silos effects that occur based on how the system itself is structured.

III. Study Design

The study was conducted during a one week period in June of 2014 with two researchers—the first two authors on this paper—on site. The study consisted of two parts: an interview segment and an observational segment. Team members were invited to sign up for either or both segments.

Each interview segment lasted on average 50 minutes. We followed an interview script of open-ended questions where we asked the interviewees questions regarding their roles and responsibilities, communication and work practices, as well as problems they have encountered and suggestions for improvement they may have. The full set of interview questions is available.[1] Five of the seven interviews were recorded with consent by the interviewees. After the study was conducted, we listened through the recordings and revisited our notes, assigned labels to the subject under discussion and the comments made by the interviewees. For the two interviews that were not recorded we relied on our notes. We iteratively aggregated the subjects and comments and present the areas that we got input on below.

Four team members signed up for observation sessions not to exceed two hours at a time, at multiple times during the one-week visit. Two of the modellers were observed for two sessions; the other two for one session.

During an observation session, a researcher sat behind the modeller and observed them at work without interacting with the modeller or interrupting the modeller in any way. Although we told modellers that they could speak during the sessions to describe what they were doing, most did not take that option. The researcher recorded the modeller's activity using: a timestamp for the start and end of every activity,the application that they were using, the artefact they were working on, their goal for performing the activity, and any problem they encountered. For activities that were interrupted by the need to do something else first, the log entry was left open-ended and then was subsequently closed after the nested task was completed. After each observation session, there was a debriefing between the researcher and the modeller when they looked through the entries to confirm the accuracy of the recorded activities and tasks, as well as the tools used. The modeller was given the opportunity to remove entries that they considered sensitive but none of them felt the need to do so. They were also given the opportunity to provide any additional information about problems they encountered during the session and provide input on what the researcher told them was their inference from the observation session. After the study was conducted the observation data was analysed in two phases. First we looked back at each session individually and gathered the problems encountered by the modellers with the tooling. Second we merged the logs from all the sessions and assigned manually each task to an activity type. Overall we witnessed 253 tasks that span over 646 minutes of work.

[1] www.cs.ubc.ca/~murphy/studydata/GM_FieldData_InterviewQuestions2014.pdf

Category		% of time over all Observ.
Development		
Spec	analyzing the specification	12.1%
Model	reading/editing/navigating model	33.1%
Debug	debugging	13.6%
Test	performing unit testing	5.1%
CodeGen	generating code from model	5.6%
CodeBuild	building generated code	3.6%
VC	reading/accepting/submitting changes	12.8%
DevOther	other related to development	2.6%
Email	reading/writing emails	1.2%
Planning	editing work items/tasks/todos; creating/changing calendar entries	6%
MeetInformal	ad-hoc, informal communication; e.g. unscheduled phone call / IM, or colleague asks a question	3.8%
Other	anything else, such as break	0.6%

TABLE I
ACTIVITY CATEGORIES FOR OBSERVATION.

IV. RESULTS

Analysis of the interview and observation data led us to three high-level findings.

A. Workflow and Time Spent

We analysed the logs collected from the observations to determine the workflow actually used by the modellers. Through this analysis, we inferred the process shown in Figure 1. This process covers the development or improvement of a functionality, ranging from the reception of the specification through the change request system to the publishing of the functionality to the team repository. Depending on the tooling used for modelling and their habits developers might perform some debugging directly on the model; Debugging directly on the model was possible for Simulink models but not for Rhapsody models. As most of the unit testing is performed at the code level, most of the debugging also occurs at the code level.

An overview of the percentage of time spent on each type of activity over all the observations is available in Table I. The prevailing activity is modelling representing one third of the total time. However, a surprisingly high amount of time, approximately 20%, is spent in generating code, building code and working with the version control system. This long cycle time is problematic as it reduces the timeframe where modellers are productive.

The higher level workflow presented agrees with the development process described by engineers and managers in the earlier interviews [1], [6]. In this study, we were able to investigate some parts of the workflow more thoroughly by observing the tasks on which modellers worked.

B. Tool Friction

We found many points in the workflow of the team where tools created more friction than necessary to complete work.

a) Modelling Phase: A lot of the points of friction observed during this phase were about usability. First when a modeller wants to link a model element to another model element (e.g. referencing a signal in a port) the proposed list of model elements is too long to be usable. Most of the the time a modeller needs to first explore the model to find the exact name of the model element she wants to link. This point of friction could be eased through recommendations or a context-aware list of possible model elements to use. Second a modeller is blocked if she needs a model element that she is not in charge of and that have not yet been created. It would be helpful to support partial modelling with a temporary model element and providing a merge mechanism when the model element is actually created. Third when writing code in the model, the lack of static checking in the source code is time consuming. Even for a small mistake, the developer will learn it after the generation and build phases. She might have to redo a long process only for a missing comma. In addition the fact that source code editor is not aware of the model elements (model elements referenced in the code are not checked) produces the same problem. Finally it is worth noting that some modellers rely heavily on the copy/paste feature to model.

b) Code Generation Phase: Code generation is time consuming, requiring 5.6% of a modeller's working time (see Table I) because the modellers perform the activity often. A lack of traceability between model and generated code increase the debugging time needed for modellers and causes them to generate code more often than if better warnings at the model level, such as for missing dependencies, were available.

c) Testing Phase: Unit tests are written and run at the code level. The difference between the two mental representations (model and code level) that the modeller needs to handle requires a lot of concentration since they are at different level of abstraction. The generation of tests from the model by using constraints on the model could be a step forward. Of course from a modeller point of view the ultimate goal would be to perform the complete validation at the model level.

d) Debug Phase: When a test fails the modeller needs to debug its model. As testing occurs at the code level, most of the debugging also happens there. The remark regarding the difference between mental representations is still valid in this step. As the generation process is long, modellers tend to fix the bug on the generated code by modifying it and then try to find how they should modify the model to produce the changes they made on the generated code. Overall they are forced to model/generate/test/debug by small steps if they want to manage the system complexity.

C. Communication follows Architecture

As part of the interviews, we asked each interviewee to discuss with whom they communicate to perform their work. The interviewees described cases of vertical communication with other teams and horizontal communication within their team.

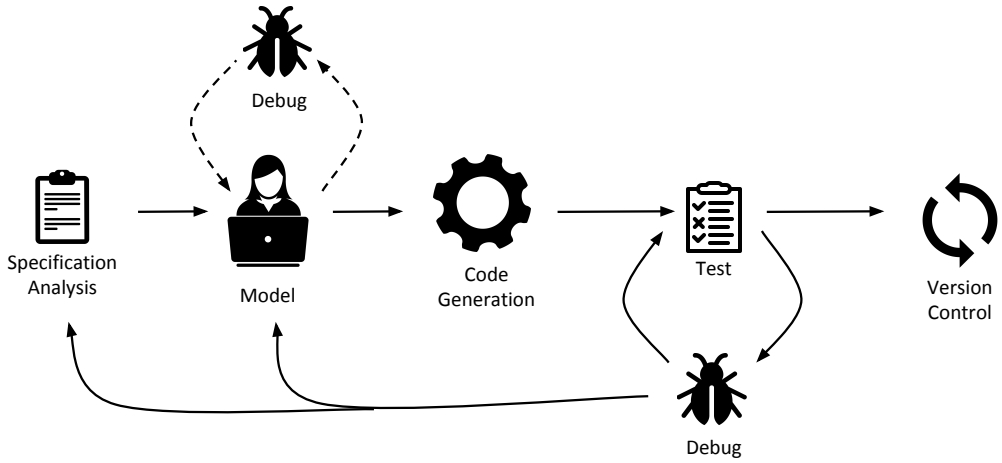

Fig. 1. Feature development process inferred from observations

e) Vertical Communication: Modellers described how information flows to them from strategists in the form of requirements, written documentation of a required feature, change or fix. Communication between modellers and strategist is frequent. Formal communication between these two parties is through change requests in IBM RTC. Since in most cases the modeller and strategists have been working together for an extended period of time, there is an established and comfortable informal communication channel between them. For quick questions that require short answers, the modellers communicate with a strategist using instant messaging. For longer questions or possible discussions that involve more individuals, the usual form of communication is email. Face-to-face meetings are also arranged between modellers, strategists, and testers, especially when a new feature is going to be implemented. Overall, modellers estimated that 75% of the time the issues that arise are minor and are resolved quickly. Once an answer, an explanation or a solution has been reached through informal communication, the relevant documentation is written as record of the decision, and finds its way into the formal process.

Vertical communication also occurs emanating out from the modellers when they flow information in the form of generated code to other teams that proceed to bench testing and testing in vehicles.

f) Horizontal Communication: Horizontal communication refers to intra-team communication. Communication between the members of the team is infrequent due to the modularization of the software architecture.The modellers reported that they seldom need to communicate with members of their team, and mostly that communication has to do with occasional questions that they may have when they are are working on a component and can benefit from the experience and insight of fellow modellers.

Communication thus follows the architecture of the system. Vertical communication flows from strategists to modellers to testers based on the component on which they all are working to define, implement and test. Communication within a team is limited as each modeller works on different components. This structure of communication can potentially create communication silos (within a component) and potentially can reduce awareness. Interviewees mentioned cases in which where they are in possession of changes that they have difficulty propagating to other teams or co-workers as this would have required communication outside their silo which is not facilitated. There were also cases reported of when knowledge of upcoming changes in the architecture would have helped teams prepare better for the changes.

V. Threats to Validity

By conducting observations of modellers at work, we were able to gain deeper insights into aspects of the work, such as where time was spent in different modelling activities. However, the observational approach was inherently limited in the number and breadth of different modellers that could be observed. Although these limitations affect the generalizability of the results both within the organization and across organizations, identifying issues where intended benefits are hampered provides value within and beyond the organization as it may indicate where to concentrate initial efforts spent investigating model-oriented technology and its use. For instance, these early findings may help direct the development of tools to allow tracking of detailed observations automatically across more of an organization or organizations.

Our interview findings are also limited to focusing on one group within a large organization. Several of the individuals interviewed had experience across the organization which may help generalize the findings beyond one particular group. The results we report are also threatened by biases in questions asked and the particular work chosen to be observed. The results should be interpreted carefully in light of these threats.

VI. Related Work

Several reports [2], [9], [7] on the adoption of model-driven engineering (MDE) in industrial context have been produced

in the last 15 years but few of them provide quantitative data showing the impact of the use of MDE [9]. More recent work by Hutchinson et al. [5], [11] attempts—especially through a large online survey—to fill that gap by collecting empirical data showing which factors lead to successful adoption of MDE and cataloguing exactly what works in MDE projects. Other recent work from Burden et al. [3] extends Hutchinson's work by applying grounded theory on interviews performed in three large companies. Our study approach in this paper is similar, relying on interviews. However, the findings of our study are to our knowledge, the first to provide quantitative data on the use of MDE based on observations collected on site.

Hebig et al. [4] reviewed literature to study the impact of MDE on software processes. They found that even if in some cases it is possible to reuse standard processes, the adoption of MDE can also result in heavyweight changes to a process. Our study complements this work by reporting an additional case where software process has already been tailored to MDE and should be tailored further in order to solve awareness challenges.

In [10] Mussbacher et al. report the results of a week-long design thinking experiment carried out with 15 MDE experts to identify the biggest problems with current MDE technologies. Our observations confirm several of the problems they ranked as major. First obstacles for tool usability and adoption and second the fact that models are still not valued as much as code. However, it is not clear if the first problem encompasses the fact that modellers spent 20% of their time doing unproductive activity (generation, building, version control) during our observations.

Existing work such as [8] observed non-modeller developers in their day to day routine to get a better understanding of their productivity. Our work differs by focusing on modellers and by identifying problems in their specific activities. It is interesting to note that Table 5 from Meyer et al. study [8] reports that the time spent on coding represented 32.3% of their observations which is close to the percentage of modelling that we observed (33.1%).

VII. Summary

Detailed observations and interviews of specific parts of how a new technology has been deployed can be helpful in understanding whether desired benefits are being achieved and where points of friction in the technology adoption may be occurring. Through detailed observations and interviews of GM modellers, we discovered where time was being spent, where tools were introducing friction into the process and how new approaches may still require attention to how communication occurs within and between teams.

Acknowledgement

This research has been funded by the Canadian Network on Engineering Complex Software Intensive Systems for Automotive Applications (NECSIS). We are grateful to General Motors and to all our study participants for their time and collaboration.

References

[1] J. Aranda, D. Damian, and A. Borici. Transition to model-driven engineering: What is revolutionary, what remains the same? In *Proceedings of the 15th International Conference on Model Driven Engineering Languages and Systems*, MODELS'12, 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, 2005.

[3] H. Burden, R. Heldal, and J. Whittle. Comparing and contrasting model-driven engineering at three large companies. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '14, 2014.

[4] R. Hebig and R. Bendraou. On the need to study the impact of model driven engineering on software processes. In *Proceedings of the 2014 International Conference on Software and System Process*, ICSSP 2014, 2014.

[5] 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*, 89, Part B, 2014.

[6] 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, 2012.

[7] D. Lugato, M. Palyart, and C. Engelvin. Domain specific modeling for operations research simulation in a large industrial context. In *Proceedings of the 2012 Workshop on Domain-specific Modeling*, DSM '12, 2012.

[8] A. N. Meyer, T. Fritz, G. C. Murphy, and T. Zimmermann. Software developers' perceptions of productivity. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '14, 2014.

[9] P. Mohagheghi and V. Dehlen. Where is the proof? - a review of experiences from applying mde in industry. In *Proceedings of the 4th European Conference on Model Driven Architecture: Foundations and Applications*, ECMDA-FA '08, 2008.

[10] G. Mussbacher, D. Amyot, R. Breu, J.-M. Bruel, B. Cheng, P. Collet, B. Combemale, R. France, R. Heldal, J. Hill, J. Kienzle, M. Schottle, F. Steimann, D. Stikkolorum, and J. Whittle. The relevance of model-driven engineering thirty years from now. In *Proceedings of the 17th International Conference on Model Driven Engineering Languages and Systems*, MODELS '14, 2014.

[11] J. Whittle, J. Hutchinson, and M. Rouncefield. The state of practice in model-driven engineering. *IEEE Software*, 31(3), 2014.