10th Workshop on Models and Evolution, 2016
Published by CEUR-WS

Software Evolution Management: Industrial Practices

Antonio Cicchetti
Mälardalen University, IDT
72123, Västerås, Sweden
antonio.cicchetti@mdh.se

Federico Ciccozzi
Mälardalen University, IDT
72123, Västerås, Sweden
federico.ciccozzi@mdh.se

Jan Carlson
Mälardalen University, IDT
72123, Västerås, Sweden
jan.carlson@mdh.se

ABSTRACT

The complexity of modern software systems and the global competition make the adoption of model-based techniques unavoidable. A higher level of abstraction not only allows to mitigate the intricacy of the development, e.g., through separation of concerns, but it is also expected to permit shorter round-trip cycles to add new system functionalities, fix bugs, and refine existing features.

This paper reports practical experiences in the management of industrial software evolution collected by means of semi-structured interviews with software development experts. All the interviewed companies develop embedded real-time safety-critical systems and aim at reaching more agile processes. Interestingly, while model-based methodologies appear to be widely accepted, shortening round-trip cycles due to changes appears still to be a major issue towards a more efficient development process.

CCS Concepts

•Software and its engineering → Software development methods; Software development techniques; Software post-development issues;

Keywords

Industrial software systems; Model-driven engineering; Model-based development; System evolution; Round-trip engineering

1. INTRODUCTION

Nowadays software can be practically considered as part of systems in any application domain. It is typically exploited to enhance or even substitute electrical and mechanical portions of systems in order to make the final product more efficient, durable, and appealing from a user's perspective. In other words, software is used to *smartify* systems by adding complex features thanks to its malleability. This wide adoption does not exclude mission-critical systems, that is systems for which a failure can have huge impacts in terms of money loss and harm to human lives. In this respect, the development of modern industrial systems is an endeavour that has to trade off development costs, time-to-market, system reliability, and customers' satisfaction, just to mention a few.

Model-Driven Engineering (MDE) [14] aims at alleviating the complexity of building modern software systems by advocating a model-centric development paradigm. Models permit to abstract away the unnecessary details and to focus on the aspects that matter in a particular development stage and/or in a specific-domain [3]. Moreover, models disclose unprecedented opportunities for automation, as they can be exploited to perform early system analysis and validation, as well as contribute to the creation of the final production code [9]. However, empirical studies have stressed that one of the major obstacles to the adoption of MDE in industry is the lack of adequate tool support [17, 5, 9], which is often intended as the need for custom modelling languages and notations, analysis and validation facilities, code generation transformations, tools able to deal with complex systems, and so forth [13].

This paper describes the preliminary exploration of industrial practices related to the evolution of software-related artefacts, meant as models, documents and production code, throughout the development process. The aim is to identify the most relevant factors hampering flawless evolution and thereby an efficient iterative approach. The general common development traits emerged from this investigation are:

- all interviewed companies use a centralised document repository which keeps track of development versions;
- the development flow is top-down, meaning that very little information on changes is propagated from lower levels of abstraction up to higher levels;
- the development process is horizontally iterative but vertically waterfall, meaning that once artefacts at a certain level of abstraction are considered successfully completed, they are frozen until the whole process is iterated again.

The core contribution of the paper is meant to be a set of findings concerning the main issues with current industrial practices when it comes to evolution of software-related artefacts. In addition, we provide what we believe are interesting research directions towards tackling the identified challenges. Despite the collected results not constituting statistical evidence, the interviews remark the need of a proper

DOI: 10.1145/1235

tool chain able to guarantee the consistency across development stages and artefacts as the precondition to achieve MDE promised gains. Otherwise, separation of concerns ends up in creating several *discontinuities* [15] that jeopardise the consistency of the integrated system and slows down the development process. In fact, not only closing those discontinuities is a time-consuming and error-prone task, but the existing gaps hamper any opportunity of automating change impact analysis, thus limiting the chances of reasoning about system evolution and its side-effects.

The paper is organised as follows: the next section provides background details about the context in which the investigation has been performed, while Section 3 discusses how the interviews have been structured and done. Section 4 provides a deep analysis about the findings that we could extract from the data gathered via interviews and Section 5 describes our view on how to proceed further for mitigating identified challenges. Eventually, Section 7 discusses related investigations on the application of MDE in industrial contexts while Section 8 draws conclusive remarks about the current work.

2. CONTEXT

The development of software shows a trend of increasing complexity under ever tighter time and budget constraints. In order to face this scenario, the development process has necessarily to be as swift as possible and avoid possible stalls due to inefficiency. Only in this way, product development can promptly adapt to the continuously changing market demands. In this respect, agile techniques promise to improve development by reducing the time spent on requirements gathering and analysis, and in general by reducing documentation efforts to a minimum [10].

Software Center (SwC)[1] is a joint effort of Swedish industry and academia which aims at rapidly introducing innovation in industry with the support of academic partners.

The results illustrated in this work have been collected in the context of a SwC project[2], which addresses the support of architectural artefacts evolution; the authors of this paper represent the research team in the project. The aim is to shorten system development iterations by means of an adequate change impact analysis and tracing of the propagation of system evolution effects. The first sprint was intended to investigate the current state-of-practice inside the participating companies, in order to better understand their development processes, evolution pressures, solutions, and needs. This paper describes the outcome of this first sprint in terms of industrial practices related of evolution of software-related artefacts.

3. SEMI-STRUCTURED INTERVIEWS

The project started with a preliminary workshop in which the participating companies and the researchers had an open discussion about the topic in order to align the different points of view and interest. In the workshop, the companies gave a high-level description of their current situation in terms of architecture/design activities and the tool chains supporting them with particular focus on how evolution is handled and on migration of information between different tools, formalisms and storages. The outcome of the workshop was that evolution management and migration of various software-related artefacts was the common point of interest. To gather more specific data on the practices and problems at each company, we decided to carry out a set of semi-structured interviews with companies staff. This data was analysed and represents the ground for the next project iterations. In this section we describe how we set up and carried out the semi-structured interviews.

3.1 Interviews preparation

Data was gathered through semi-structured interviews [8] with staff. The difference between structured and semi-structured interviews is that in the latter the interviewee is allowed to divert from an initial set of open questions, which are considered as a minimal set of well-thought topics, related to software evolution in our case, to be discussed. Such topics (and questions) were used by the interviewers as interview guide, and were sent in advance to the interviewees for preparation. During the interviews, questions were asked in different ways and at different moments, depending on the interviewee and the flow of discussion. Doing so, we were able to customise our questions to the specific interview and interviewee.

The set of topics and questions was the following:

T1 Tools ecosystem – What formalisms, languages and tools do you use for development and documentation of design/architecture/implementation?

T2 Artefacts storage – How are artefacts (models/code/documents) stored?

T3 Evolution strategies – How, and how often, do you exchange/migrate/synchronise/version information between different languages/tools/formats/storages?

T4 Diff/merge and conflict resolution – When exchanging/migrating/synchronising/versioning artefacts, how is differencing and merging as well as resolution of conflicts among artefacts handled?

T5 Concurrent development – How common is it that multiple persons work on the same artefact (concurrently or interleaved) and how do you currently organise concurrent work?

T6 Envisioned improvements – What improvements would you like to see in relation to your current evolution management practices?

Semi-structured interview was also chosen as data collection method since, being at an initial phase of investigation, we did not want to steer the interviews in a predefined direction through a strict questionnaire. On the contrary, with semi-structured interviews, we wanted to gather a broader set of information about practices, problems, ideas about software evolution in the companies.

3.2 Running the interviews

We interviewed a total of 9 individuals across the 3 companies (3 persons per company); interviewees were chosen so to represent a reasonable range of different roles involved in software development. The three companies are large Swedish enterprises with focus on embedded real-time and

[1]http://www.software-center.se/.

[2]http://www.software-center.se/research-themes/technology-themes/continuous-architecture/Evolution+Support+for+Architectural+Artefacts.

safety-critical systems in different application domains. We interviewed software engineers, testers, product managers, and modellers. All interviewees had experience with MDE. Interviews were conducted on-site by at least two of the authors and in the interviewees' native language, Swedish. The set of questions was sent to the interviewees in advance in order to allow them to raise possible issues and prepare for the interview by gathering additional information if needed.

The interviews started with a short summary by the interviewers about the SwC project and the interview itself. This was followed by the interviewee describing shortly her current role in the company and experience with similar topics in general. Thanks to the fact that we exploited a semi-structured format, we could notice that the various interviews took different, and interesting, routes, highlighting expected and unexpected problems in the industrial software evolution. In case of unclear statements by the interviewee, a clarification was always sought and given before going forward to the next topic.

The interviews lasted between 30 and 90 minutes and were recorded in order for resulting data to be analysed and synthesised. For analysis purposes, the recorded audio of each interview was listened to by one researcher and transcribed by question; additional details not related to a specific question were annotated separately. The transcribed answers were discussed extensively within the research team in order to identify the most relevant findings to document and that would represent the basis for the next project iterations.

4. RESULTS

The interviews gave interesting answers to the posed questions but also provided unexpected insights thanks to the semi-structured format. In this section we provide a summary of the most relevant (shared) findings that the research team identified by discussing the transcribed interviews.

4.1 Tools ecosystem and artefacts storage

The answers gathered during the interviews clearly show that at companies there usually exists a plethora of tools, languages and formalisms used more or less together for requirements specification, software modelling and production code. Even if standard languages (e.g. ADLs or UML) are used, the observed trend is working on company-tailored versions of existing commercial tools. Moreover, different groups in the same company use different toolchains, even in joint projects. When it comes to requirements, the common line is represented by textual descriptions that are maintained in various tools. These textual descriptions are manually broken down to a so called "software architecture" defined in terms of graphical models. The architecture usually defines parts of the software and hardware nodes composing the entire system as well as allocation of software to hardware, and it is used for early assessment of modelled functionalities. In some cases, dynamic parts of the software system are modelled using tools and languages different from the ones used for architectures. Moreover, in many cases different parts of the software system are modelled and developed with different tools and languages. When it comes to production code, in some cases it is generated from models, while in others models are just used as blueprint for the programmers to implement. The common trait is that consistency models-code is regarded as important but currently hard to achieve since it is not automated (many tools in the ecosystem do not have automatic bridges): *"It happens that, after a number of years, models are not updated anymore, thus losing their consistency with up-to-date code.. There is simply no time to do it.. It feels like the lack of agile ways to evolve models and related artefacts leads to neglect models and update only the code."*[3]

Models, documentation and production code are managed with different tools but are commonly stored in a central database. Anyhow, having them in a central database does not provide tangible advantages when it comes to evolution since there are still serious issues in bridging the various tools: *"Exchanging information among tools using XMI creates many problems due to the format misalignment in the various tools.."*.

In synthesis, many heterogeneous tools and languages are used without effective and automated mechanisms for univocally going from one to the other and for ensuring that changes to one artefact are correctly propagated to related and dependent artefacts. This makes evolution of the various artefacts (documents, models, code) a manual, error-prone, tedious and in some cases a shabby task.

4.2 Evolution strategies, diff/merge and conflict resolution

Evolution strategies, especially among related artefacts of different types, are shallow. Changes to code are not always reproduced in models and documentation since it is a manual effort up to a single individual and there does not seem to be a strong focus on "consistency maintenance" of less crucial artefacts (e.g., models and documentations) in the companies. Running versions of code on products with very long lifetime are kept as they are as much as possible; only small fixes are made when unavoidable. Moreover, evolution happens at different levels and in different tools although affecting the same artefact; while some automation for propagating change from one level and/or one tool to the other exists, much is still left to manual, unstructured activities. Instead of improving evolution strategies, all companies try to minimise the need of evolution by going for a conservative and add-only strategy as much as possible.

There is usually no efficient support for round-tripping from code to models. More specifically, companies would find it beneficial to propagate changes done on code back to models and documentation (across different tools), for instance in terms of warnings. The impossibility to efficiently do this, as well as the aforementioned intrinsic difficulties in bridging the many tools composing the ecosystem, *".. make often software development start from scratch in new projects rather than reusing models, documents and code (or parts of them) from previous successful projects."*. Instead, new projects should be intended as evolutions of previous ones, with artefacts that evolve from one project to the other in a unbroken manner.

The common trait with diff/merge is that all companies try to avoid merging as much as they can. Differencing is, on the other hand, considered a very important aspect but, being often a manual task, it is error-prone and tedious. Implicit diff/merge is done by versioning tools used by the companies, such as Subversion; also in this case, several issues arise when versioning graphical models exploiting their textual representation. Moreover, the heterogeneity of for-

[3]Note that reported extracts from interviews have been translated by the authors from Swedish to English.

mats among the many different tools and shaky links among them clearly does not simplify diff/merge and conflict resolution.

4.3 Concurrent development

Concurrent modelling is commonly achieved by partitioning models in sub-portions (either physical files or logical sub-systems) in order to avoid concurrent changes on the same model portion. While developers do not work with the same model portion, they usually work on the same database concurrently and the data is continuously synchronised through automatic database-specific mechanisms so that everyone has the same overall picture of the up-to-date product version. There does not seem to be any particular problems with this practice; this is explained by the fact that in reality no concurrent modelling ever takes place since sub-portions are not meant to be concurrently accessed by multiple persons. Nevertheless, the possibility of disruptive changes to dependent software modules is still possible both at model and code level due to (i) human misunderstandings and (ii) because no dependency analysis is done among different software modules. *"These kind of problems do not come up until testing production code.. for instance by getting compilation errors."*.

4.4 Envisioned improvements

An aspect which seems to be of high priority for the interviewed companies is the need for improving traceability, that is to say back-tracing of changes done at lower levels of abstraction (e.g., code) to related artefacts at higher abstraction levels (e.g., models and documents). This is seen as crucial for being able to enhance traceability as well as improving versioning and boosting reuse. Particularly important is considered to achieve much more efficient (and automated) ways to ensure consistency between models and code in the long run. From our interviews it became clear that practitioners are starting to perceive the current fixed tool-centric idea as somewhat obsolete within a model-driven or model-based development process. Instead, they put emphasis on the need to move the attention towards more flexible (meta)model-centric approaches. Functionalities get more in number and more complex; additionally, there is a much higher dependability among different parts of the software system than before. This complicates evolution, also considering the huge variations among product versions; legacy running systems must be supported and updated in a smart, non-breaking way. Having a (meta)model-centric approach would permit to *".. define variation points in the metamodel that would simplify evolution tasks"..*

5. DISCUSSION

So far we described the feedbacks collected during the interviews with companies, grouped by topic. Based on those feedback, this section highlights a set of relevant issues that we believe hamper a better management of evolution, together with a set of possible corresponding research investigation directions in order to improve the current practice.

Challenge 1: Heterogeneity of tools and languages in the toolchain

Issue Heterogeneity of the many (modelling) tools and languages typically exploited in an industrial development process is a major hinder for effective evolution due to the lack of appropriate format exchange and change propagation support.

Investigation directions Introduce more powerful and automated links between interconnected artefacts across different tools and languages, and in the long run create bidirectional bridges (through e.g. model transformations) supporting uncertainty.

A critical aspect that seems to be underestimated by current empirical investigations on the adoption of MDE in industry is the transitional nature of such a process [2]. In fact, usually software is developed by companies whose main products are not software (e.g., cars, satellites). Therefore, software development gets intertwined with other engineering activities and very often imposing a one-step adoption of a completely (even if fully featured) new development platform is not realistic.

A more realistic scenario is a step-wise adoption of domain- or task-specific tools that contribute to the development process. The process is orchestrated by means of a centralised storage support, which is format-agnostic, and where tool interconnections are kept through links. Since in general the tools cannot communicate with each other, change propagation can only be supported in its minimal terms, that is raising warnings for artefacts linked to entities involved in evolution activities. Moreover, consistency management becomes necessarily a manual task.

The companies solve consistency and change propagation issues by constraining the development in a waterfall process. More precisely, all the artefacts related to a certain development step or abstraction level (that is, horizontally) are iteratively developed until they satisfy a planned goal. After that, those artefacts are *frozen*, i.e. kept as read-only until the next development process iteration. In this way, consistency between abstraction levels is enforced by the subsequent steps, since each higher abstraction level is input for lower ones. Besides, whenever there is a need to make changes outside the specification coming from a higher level of abstraction, this would be recorded in terms of a change request to be dealt with in the next iteration.

Given this scenario, we propose two possible investigation directions: introducing more powerful links in the central storage and/or creating transformations between tools. The former can be conceived as a short term solution, since it does not require many changes in an existing development process. However, already by adding semantic information to the links (notably, the degree of dependence between linked artefacts) it would improve change impact analysis. This solution would require a company-specific investigation of the evolution characteristics that artefacts undergo during a typical development process.

In the long run, tools should be connected by means of model transformations. These transformations should be bidirectional to allow the synchronisation on both sides, and supporting uncertainty to be able to effectively deal with the intrinsic heterogeneity of the mappings [6, 7]. In this

case, the degree of interconnection between the various tools should be carefully evaluated, in order to avoid the need of writing dozens of model transformations.

Challenge 2: Diff/merge at the appropriate abstraction level

Issue Differencing and merging are mostly manually performed (when they cannot be avoided) since currently employed versioning tools are text-based and not able to effectively operate diff/merge on complex artefacts, such as graphical models. The XMI format does not seem to help in this matter.

Investigation directions Introduce diff/merge operations at the modelling level of abstraction.

Differencing and merging models at an appropriate level of abstraction is a known problem for the MDE research community. The proliferation of tools illustrated in the previous challenge stresses this need since manipulations might have different interpretations/impacts depending on the consumer of the modified artefact. Moreover, in general modelling tools are not equipped with diff/merge features.

Companies tackle this issue by avoiding diff/merge, i.e. by *serialising* concurrent manipulations through the exploitation of disjoint sub-portions of models and/or the adoption of artefact locks. Even more, in some cases code is edited by hand instead of re-generated after an appropriate modification of the source models it originates from, especially in case of small refinements. This relieves companies from making an additional effort in understanding how model changes would propagate to interconnected artefacts at higher abstraction levels.

For this challenge we propose the study of appropriate diff/merge mechanisms. The solutions could be gradually developed and introduced firstly based on the improved links discussed in the previous challenge, and hence trying to detect at least the changes relevant for tool interconnections. In the long run, change detection should provide enough information to enable a reliable change impact analysis and possibly automated propagation through synchronisation transformations.

Challenge 3: Explicit traceability between related artefacts

Issue The lack of effective support for explicitly keeping track [1] of dependencies among different artefacts in the many tools makes round-trip engineering very difficult and error-prone.

Investigation directions Analyse the changes at various levels of abstraction, at different development stages, with respect to the used tools, etc., and classify them in terms of their impact on other artefacts. In the long run, produce estimates of propagation effort due to changes.

The central storage used to coordinate the development process can be considered as a very basic support for traceability. However, the lack of adequate interconnection information makes this tracking support not effective. Notably, the dependencies among different artefacts in the many tools can make cautious concurrent modelling still cause disruptive changes that can go unnoticed until production code is tested. In turn, this slows down the whole process due to the efforts required to understand the origins of the problem.

Another important issue caused by the lack of traceability is the poor support for round-trip engineering. Since the link between the various artefacts gets typically blurry when going from an abstraction level to another, it is very difficult to understand how modifications at lower abstraction levels should be propagated back to higher levels. The companies mitigate this issue by adopting the waterfall-like process described before, which however slows down the development process and limits the opportunities for concurrent development.

Our proposal is to empirically retrieve historical evolution information and propose a characterisation of possible system evolutions. The improved interconnections between the tools proposed for *challenge 1* would support a more effective synchronisation of artefacts at different levels of abstraction. Together, these two investigations can create the potentials for an enhanced round-trip engineering process, in which code and other artefacts can be consciously modified, and the effects of the modifications can be better analysed and propagated to interconnected artefacts.

6. THREATS TO VALIDITY

When it comes to internal validity, we adopted a systematic approach in preparing the study, gathering, analysing and synthesising data. On the one hand, since we did not opt for structured interviews with a strict set of questions, but rather went for semi-structured interviews, the rigour of the study and its results could be questioned. On the other hand, less structured methods [8] have already proven very useful when the interviewer seeks a broad spectrum of information. This is possible since the interviewee is given more freedom and somehow participates actively to steer the discourse even towards unforeseen (but not less interesting) directions [12]. In order to avoid misinterpretations of answers given by interviewees, interviews were always carried out by at least two members of the research team. Moreover, analysis and synthesis of the gathered data as well as elicitation of the findings were performed by the entire research team.

Regarding external validity, the interviewed companies develop large-scale software to run on safety-critical embedded real-time systems with long lifetime. This means that our results can happen to not being applicable to very small business cases or short lifetime products. Moreover, due to the fact that we did not adopt a strictly structured interview method and given the rather limited number of interviewees we do not claim statistical relevance of our results. Nevertheless, we believe that the results coming out from our study, especially considering the fact that they were agreed by all interviewed companies, represent a first step towards understanding the practical issues of evolving software artefacts when adopting model-driven or model-based development.

7. RELATED WORKS

The last decade has seen an increasing amount of publications devoted to empirically assess the industrial practices in adopting MDE and both its good and bad effects [11, 9]. Usually the existing literature surveys the problem from an overall software development process perspective, analysing the effects across different applicative domains [5, 17, 4] and/or target systems [9]. On the contrary, this work specifically targets evolutionary scenarios and the impact of MDE in their management. In this respect, as mentioned in Section 5, even if our data cannot be considered as statistically relevant, it is possible to notice some trends confirming other existing results, and also to deduce interesting explanations about some current challenges in adopting MDE in concrete industrial settings.

Burden et al. [5] present a study of MDE adoption at three large companies, two of which are also involved in this work. It is therefore not surprising that we share several observations with the cited paper, especially related to the development process and the need of freezing artefacts/specifications pertaining to an abstraction level (or development stage) before proceeding further. More generally, studies like [5, 4, 16] or the one described in this work remark the distinction between companies producing software as main business and companies using software in their products: in particular, the latter typically face more problems due to the need for integration of the software development with the remaining part of the system realisation process.

It is worth noting that most of the empirical investigations seem to assume the adoption of MDE as a one step process, or they observe the effects once the adoption process is considered as satisfying/completed. However, MDE adoption usually happens as a transition [2] in which MDE methods are incrementally *plugged in* in the development process. This also emerges in our interviews, where there exist several degrees of adoption even when considering different departments in the same company. The consequence is that issues due to the partial adoption of MDE are revealed through other side-effects and troubles, notably the lack of modelling competence, the inadequacy of the tools and their integration, the extra efforts required to boot-strap MDE-based development processes.

8. CONCLUSIONS

This paper discussed the issues faced in the management of evolution in industrial software development processes adopting MDE. These issues have been identified by analysing the data gathered from semi-structured interviews with practitioners in the context of an industry-academia joint research project aiming at enhancing development processes.

The problems described by the interviewees remarkably affect the development performances and hamper effective round-trip engineering. Starting from those problems, in this paper we list a number of relevant challenges together with their effects on the development process. Moreover, we propose a set of corresponding research directions that we plan to investigate as next steps in the joint research project mentioned above.

9. ACKNOWLEDGMENTS

The authors would like to thank all the interviewees that accepted to collaborate in our project and provided useful and honest insights about the internal development practices at their companies. The project is supported by Software Center.

10. REFERENCES

[1] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni. Model traceability. *IBM Systems Journal*, 45(3):515–526, 2006.

[2] J. Aranda, D. Damian, and A. Borici. *Transition to Model-Driven Engineering*, pages 692–708. Springer, Berlin, Heidelberg, 2012.

[3] J. Bezivin. On the Unification Power of Models. *SoSym*, 4(2):171–188, 2005.

[4] M. Broy, S. Kirstan, H. Krcmar, and B. Schätz. *What is the Benefit of a Model-Based Design of Embedded Software Systems in the Car Industry?*, pages 343–369. IGI Global, 2012.

[5] H. Burden, R. Heldal, and J. Whittle. Comparing and contrasting model-driven engineering at three large companies. In *Procs. of the 8th ACM/IEEE Int. Symposium on Empirical Software Engineering and Measurement*, ESEM '14, pages 14:1–14:10, New York, NY, USA, 2014. ACM.

[6] R. Eramo, A. Pierantonio, and G. Rosa. Managing uncertainty in bidirectional model transformations. In *Procs. of the 2015 ACM SIGPLAN Int. Conf. on Software Language Engineering*, SLE 2015, pages 49–58, New York, NY, USA, 2015. ACM.

[7] C. Hardebolle and F. Boulanger. Exploring multi-paradigm modeling techniques. *SIMULATION*, 85(11-12):688–708, 2009.

[8] S. E. Hove and B. Anda. Experiences from conducting semi-structured interviews in empirical software engineering research. In *11th IEEE International Software Metrics Symposium (METRICS'05)*, pages 10–23. IEEE, 2005.

[9] G. Liebel, N. Marko, M. Tichy, A. Leitner, and J. Hansson. Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice. *Software & Systems Modeling*, pages 1–23, 2016.

[10] R. C. Martin. *Agile software development: principles, patterns, and practices*. Prentice Hall PTR, 2003.

[11] P. Mohagheghi and V. Dehlen. *Where Is the Proof? - A Review of Experiences from Applying MDE in Industry*, pages 432–443. Springer, Berlin, Heidelberg, 2008.

[12] K. Musante and B. R. DeWalt. *Participant observation: A guide for fieldworkers*. Rowman Altamira, 2010.

[13] G. Mussbacher, D. Amyot, R. Breu, J.-M. Bruel, B. H. C. Cheng, P. Collet, B. Combemale, R. B. France, R. Heldal, J. Hill, J. Kienzle, M. Schöttle, F. Steimann, D. Stikkolorum, and J. Whittle. *The Relevance of Model-Driven Engineering Thirty Years from Now*, pages 183–200. Springer International Publishing, Cham, 2014.

[14] D. C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, 2006.

[15] B. Selic. The Pragmatics of Model-driven Development. *IEEE Software*, (5):19–25, 2003.

[16] M. Staron. *Adopting Model Driven Software Development in Industry – A Case Study at Two Companies*, pages 57–72. Springer, Berlin, Heidelberg, 2006.

[17] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden, and R. Heldal. *Industrial Adoption of Model-Driven Engineering: Are the Tools Really the Problem?*, pages 1–17. Springer, Berlin, Heidelberg, 2013.