The State of Practice in Model-Driven Engineering

Jon Whittle, John Hutchinson and Mark Rouncefield
School of Computing and Communications,
Lancaster University
{j.n.whittle, m.rouncefield}@lancaster.ac.uk, johnhutchinsonuk@gmail.com

Abstract
Despite lively debate over the last decade on the benefits or drawbacks of model-driven engineering (MDE), there have been very few industry-wide studies of MDE in practice. We present a new study, covering a broad range of experiences and ways of applying MDE: we surveyed 450 MDE practitioners and carried out in-depth interviews with 22 more. Findings suggest that MDE may be more widespread than commonly believed, but developers rarely use it to generate whole systems; rather, they apply it to develop key parts of a system often using domain-specific modeling languages developed specifically for the purpose. Our findings also suggest reasons why some efforts to adopt MDE fail and some succeed. As is usually the case in software engineering, adoption largely depends on social and organizational factors, some of which we describe in this paper.

Keywords: Software Design Methodologies, Model Driven Engineering Practice

In 2001, the Object Management Group published the first version of its Model Driven Architecture (MDA) specification. MDA emphasized the role of models as primary artifacts in software development and, in particular, argued that models should be precise enough to support automated model transformations between lifecycle phases. This was not a new idea, of course, but it did lead to a resurgence of activity in the area as well as hotly contested debates between proponents and detractors of model-driven approaches [1].

Many years later, there remains a lack of clarity on whether or not model-driven engineering (MDE) is a good way to develop software [see sidebar on MDE]. Some companies have reported great success with it, whereas others have failed horribly. What is missing is an industry-wide, independent study of MDE in practice, which highlights factors that lead to success or failure. Although there have been a few prior surveys of modeling in industry, they have each focused on only one aspect of modeling, such as the use of UML [2] or the use of formal models [3].

In this article, we report on a new study of MDE practice. In contrast to prior surveys, we cover a broad range of MDE experiences. Furthermore, we focus on identifying success and failure factors of MDE. We surveyed 450 MDE practitioners and interviewed 22 more from 17 different companies representing 9

different industrial sectors. The study reflects a wide range of maturity levels with MDE: questionnaire respondents were equally split between those in early exploration phases, those carrying out their first MDE project, and those with many years' experience with MDE. Interviewees were mostly very experienced with MDE.

We discovered a number of surprises about the way that MDE is being used in industry. And we learned a lot about how companies can tip the odds in their favor when adopting MDE. Many of the lessons point to the fact that social and organizational factors are at least as important in determining success as technical ones. We describe elsewhere the gory details of the research approach [4, 5]. In this article, we focus on key take-home messages for those who have adopted or who are thinking of adopting MDE.

MDE Use is Widespread

Some claim that the application of MDE to software engineering is minimal. MDE, they argue, is only used by specialists in niche markets. Our data refutes such claims, however. We have found that *some form of MDE* is practised widely, across a diverse range of industries (including automotive, banking, printing, web applications etc.). The 450 questionnaire respondents, for example, were employed in a range of different roles (36% developers, 37% project managers) and represented a good spread of size of company with respect to the number of people involved in development (e.g. 52%<100 and 19%>1000). The interviews back up this finding and illustrate that MDE use is in fact widespread and, in particular, companies use MDE in many different ways – ranging from industry-wide efforts to define precise models for an entire application domain, to very restricted, limited uses of MDE in the generation of code for a single application family in one company.

Perhaps surprisingly, the majority of MDE examples in our study followed domain-specific modeling paradigms; that is, the companies who successfully applied MDE largely did so by creating or using languages specifically developed for their domain, rather than using general purpose languages such as UML. Interview data shows that it is common to develop small domain-specific languages (DSLs) for narrow, well-understood domains. In contrast to perceived wisdom – that significant effort should be employed in developing models that cover broad domains and capture knowledge in that domain – practical application of domain modeling is 'quick and dirty', where DSLs (and accompanying generators) can be developed sometimes in as little as two weeks. There is widespread use of mini-DSLs, often textual and there may be many such mini-DSLs used within a single project. A clear challenge is how to integrate multiple DSLs. Our participants tended to use DSLs in combination with UML; in some cases, the DSL was a UML profile. In any case, modeling languages require significant customization before they can be applied in practice.

Our findings also lead us to believe that most successful MDE practice is driven from the ground up. MDE efforts that are imposed by high-level management typically struggle; interviewees claimed that

top-down management mandates fail if they do not have the buy-in of developers first. As a result, there are fewer examples of the use of MDE to generate whole systems. Rather than following heavyweight top-down methodologies, successful MDE practitioners use MDE as and when it is appropriate and combine it with other methods in a very flexible way.

Code Generation not the Main Driver for MDE

Surprisingly, it appears from our data that code generation is *not* the key driver for adopting MDE. MDE is often considered to be synonymous with code generation (or at least MDD, see sidebar on MDE) and it is code generation that is perceived to bring the benefits, including productivity. However, although reports of productivity gains vary widely (from a 27% loss to an 800% gain [6]), most companies seem to experience productivity increases of between 20-30%.

Interestingly, our data suggests that such increases are not considered significant enough to drive an MDE adoption effort: MDE brings with it increased training costs and substantial organizational change that easily offset 20-30% productivity increases. This does not mean, however, that companies do not adopt MDE. Rather, the interview data illustrates time and again that, although companies use code generation, they find other benefits to MDE which are much more important than these relatively minor productivity gains. In this sense, therefore, code generation is a red herring when it comes to describing MDE and our results suggest a re-interpretation of how MDE is envisaged, marketed and understood.

The Real Benefits of MDE are Holistic

So, if the real benefits of MDE are not to be found in code generation, then where can they be found? It turns out that the main advantages are in the support that MDE provides in documenting a good software architecture.

Most would agree that a clearly described software architecture is one of the key ingredients for successful software development. However, software engineers lack the skills, know-how or time to invest in expensive architecture definition efforts and, as a result, although the value of architecture definition is accepted philosophically, it is often not practiced.

Unanimously, our interviewees argue that MDE makes it easier to define explicit architectures, especially when MDE is a ground-up effort. When precise modeling is gradually introduced into an organization, developers find themselves recognizing similar code fragments that they can then abstract into a DSL and write a generator for. In effect, they are incrementally building up an architecture description. The rigor that precise modeling imposes on developers forces them to develop an explicit

architecture description, but in a way that does not impose a heavyweight and lengthy architecture definition process.

The following example is illustrative. One company used a variety of XML-based DSLs to generate large parts of a major, complex system. Over time, the developers began to realize that they were building up an architecture by using a non-standard form of separation of concerns: they found themselves looking for parts of the system to automatically generate (the simpler parts) and parts that experienced software developers needed to write (the complex parts). This form of separation of concerns – a division of simple and complex – brought about a much deeper understanding of the system's architecture and arose not because of a managerial edict but because of the way that MDE evolved in practice.

Success Requires a Business Driver

Even in companies that recognize the benefits of MDE, adoption can take a long time, even when compared to adoption of other approaches such as agile methods. Our data illustrates that one of the main factors for this inertia is that MDE is usually marketed as a technology that can do the same things *faster* and *cheaper.* However, this is not usually enough motivation for companies to risk adopting MDE; rather, companies that adopt MDE do so because it can enable business that otherwise would not be possible.

An illustrative example is the experience of a well-known, global printer production company, which consciously started using MDE ten years ago. At that time, software was the bottleneck in the company – there was a widely-held perception that software was a limiting factor in getting a new generation of printers to market. However, after ten years of evolving their use of MDE, the company now reports that software is no longer the bottleneck in the company. In other words, MDE has enabled the printer company to be what it always should have been – a company focusing on printers not software. This finding suggests a re-thinking of the way we market MDE: not as a way to do things faster, but as a way to do new things.

The observations presented so far offer interesting insights as to why some companies adopt MDE successfully and others do not. In addition to this, our data sheds light on some psychological and organizational aspects of MDE for which the results show promise but would benefit from further investigation.

The Psychology of MDE

A phenomenon observed in other sub-fields of software engineering is that there can be significant individual differences between certain types of developers – e.g., between novice and expert programmers [7, 8]. We have observed similar effects with MDE.

Firstly, it appears that software architects generally react well to MDE. An MDE project uses code generators that encode architectural rules, constraints and patterns that software architects have formulated. MDE therefore puts more control into the hands of architects, who can now easily enforce their design decisions across a development team.

Secondly, certain types of developers can be very resistant to MDE. This applies both to 'code gurus', who are traditionally asked to solve hard technical challenges, and 'hobbyist developers', individuals that like to play with new coding technologies, even outside work hours. In the former case, the resistance to MDE is again an issue of control: these individuals see MDE as threatening to reduce their importance to the company. In the latter case, hobbyist developers perceive that MDE will constrain their creativity since it automates many tasks.

Managerially, we have observed similar findings as in other software engineering sub-fields. In particular, it appears that 'middle managers' can be a bottleneck in adopting MDE. These are managers who are subordinate to senior managers but are above operational staff. They typically have little strategic responsibility and therefore may not see the future vision that MDE can bring. Instead, their main responsibility is to track schedules and milestones, which makes them naturally risk-averse and resistant to new technologies.

MDE may offer a fundamental shift in global software development. Numerous companies reported that they reduced their offshoring activities as a result of MDE because they are now able to automate onshore tasks that were previously outsourced.

There appears to be some disagreement in industry whether everyone is capable of thinking abstractly. One company, for example, reported that *the* major bottleneck in their use of MDE is that they had to retrain hundreds of software coders, many of whom were unable to make the jump to abstract thinking. Other companies, in contrast, have reported that only a very small percentage of coders are unable to think abstractly (a figure of 3% was quoted, although, this is in no way scientific). Although this issue clearly relates to a company's level of MDE maturity, the results also suggest that we have only a very limited understanding of abstract thinking in software development – an observation also made by others [9].

Finally, there is some evidence that the 'MDE guru' needs to have software development (and abstraction) skills as well as an in-depth understanding of the domain (or domains). Since most MDE efforts are highly domain-specific, domain knowledge is crucial. However, although possible, success is

less likely when a team has a division of skills between domain experts and MDE experts. Chances of success increase if team members have both sets of skills – that is, individuals within the team are able to develop meta-models of the domain and code generators for the domain, and they are able to reason about the domain. This leads to fewer misunderstandings and can speed progress.

Organizational Factors

As with other software engineering methods, there are interesting relationships between the structure and business of an organization and the likelihood that MDE is appropriate or will be a success.

Our data resoundingly suggests that MDE is not appropriate for every type of organization (at least not yet). Interestingly, companies that target a particular domain – e.g., automotive, printer interfaces, financial applications – are more likely to use MDE than companies that develop generic software (e.g., software consultancies). The former already employ domain experts who are probably already creating models. Although they might create these models as sketches or, in some cases more detailed blueprints, they may only be doing this informally using (e.g.) Powerpoint. As one of our interviewees stated, it is easier to move from these informal models to precise, computer-readable models than to starting modeling from scratch.

In contrast, developers writing generic software may struggle to see the relevance of modeling and, in fact, modeling may not be appropriate for the kind of software they are developing. This point has been made no more forcefully than a large, global software consultancy noting that whilst they had used MDE successfully many times with clients working in specific domains, they considered it too unlikely to succeed in-house.

MDE seems to question some of the assumptions about how organizations evaluate individuals and teams of developers. For instance, architects have reported to us that they sometimes artificially increase the complexity of their models because their managers do not understand that a simple model is better; rather, their managers perceive simple models as not properly thought out.

The way in which organizations hire new staff also does not fit with the MDE way of thinking. Typically, developers are hired based on what technologies they are familiar with rather than what domains they have knowledge of. However, as seen above, the ‘MDE guru’ needs an in-depth understanding of one or more domains to make the technique succeed.

Tips of the Trade

Many of our results point to specific guidelines that practitioners should be aware of. A major goal of our study was to start to understand such guidelines. We offer here our top five tips for success with MDE, based on the empirical data we have gathered.

1. Keep Domains Tight and Narrow. In agreement with other sources (e.g., [10]), we have found that MDE works best when used to automate software engineering tasks in very narrow, tight domains. That is, rather than attempting to formalize a wide-ranging domain (such as financial applications), practitioners should write small, easy-to-maintain DSLs and code generators. In practice, however, multiple DSLs are usually required, which brings its own challenges in terms of integration.

2. Put MDE on the Critical Path. Perhaps counter-intuitively, successful MDE initiatives argue that MDE should be tried on projects that cannot fail – that is, avoid the temptation to try out MDE on side-projects which will not have sufficient resources or the best staff. MDE should still be introduced incrementally but each increment needs to add real value to the organization for it to succeed.

3. Be Careful About Gains Offset Elsewhere. Because of the rather modest gains in productivity, it is easy to offset these gains in other parts of the organization, and, many times, a company may not realize that gains in productivity achieved through code generation are in other branches of the company. A poignant example is when certifying code for use in government information systems: one case study showed that because of the lack of readability and inefficiency of code generated by commercial off-the-shelf generators, code certification costs rose by a factor of eight.

   A second example is a company that mandated the use of a commercial MDE tool. However, the developers could not get the tool to fit their processes, and, under pressure to 'make things work', they hacked it, messed with the generated code, and circumvented it when they had to.

4. Most Projects Fail at Scale-Up. As noted above, MDE may work best when driven from the ground-up. A natural point, of course, arises when an organization wishes to unite such grassroots efforts and effect organizational change. This is, not surprisingly, where problems start to arise and managers should be careful to allocate appropriate resources during this transition phase.

5. Don't Obsess About Code Generation. MDE is often sold as a code generation solution. As we have seen, however, the real benefits of MDE do not necessarily lie in code generation and companies would therefore be wise to consider the more holistic benefits that MDE can bring rather than focusing only on code generation.

Training in MDE

Our data also suggests implications for the way that modeling is taught. A typical university course in software engineering teaches in a top-down fashion in which requirements models are first developed and are then iteratively refined into architecture, design, code, tests etc. Students often have a great deal of difficulty proceeding in this manner because it requires them to formulate an abstract understanding of the system under development before the concrete details are understood.

However, in our study, we have observed that attempts to introduce MDE into a company in this kind of top-down, organizational-wide manner are fraught with difficulty. Those companies that do succeed invariably do so by driving MDE adoption from the grassroots: that is, small teams of developers try out aspects of MDE, which in turn lead them to recognize reusable assets, and eventually MDE propagates to the organisation as a whole. This way of working suggests that developers find it easier to get to grips with MDE when refactoring existing assets from the ground-up rather than in trying to abstract from above. So there is a mismatch between the way MDE works in practice, and the way we teach it.

In addition, it appears that MDE developers need both compiler development skills and abstraction skills. Unfortunately, these skill sets are usually taught in distinct parts of a computer science curriculum and it is often the case that there is little relationship between them. Based on our evidence, however, we would argue that abstraction and compilation/optimization techniques ought to be taught together in an integrated fashion. Such an idea would significantly alter the way that software engineering is taught and would skill-up a new generation of developers capable of both abstracting in a problem space and automating the transition to a solution space in an efficient manner [11].

Where Next?

Our study is the first wide-ranging industry study of MDE practice. It has uncovered many companies who have had great success with MDE and has identified some of the reasons why. It has also uncovered some companies who have tried to apply MDE but have given up. Many of our findings are general development lessons and are consistent with findings from other studies (e.g., on formal methods use [3]). Clearly, however, there are MDE-specific lessons too, such as those that deal with code generation or abstraction.

Perhaps the biggest eye-opener of the study, however, is the realization that state-of-the-art modeling techniques and tools do a poor job of supporting software development activities. We found no consensus on which modeling languages or tools developers use – they cited over 40 modeling languages and over 100 tools as ‘regularly used’ in our survey. A recent study by Petre [2] studied 50 software designers and found that these designers either did not use UML all or used it only very selectively and informally.

These studies highlight that the fundamentals of modeling – how designers 'do' abstraction, how engineers reason about a system in abstract terms, how organizations work with abstract concepts – are not well reflected in current modeling approaches. Indeed, the vast majority of modeling approaches – both industrial and academic – are developed without an appreciation for how people and organizations work. UML2.0, for example, a major revision of the UML standard, did not reflect the literature on empirical studies of software modeling or software design studies. As a result, current approaches force developers and organizations to operate in a way that fits the approach; rather than making the approach fit the people.

We end then by arguing for a concerted effort to develop modeling approaches that better reflect the way that developers and organizations handle abstraction and complex problem solving. We believe that the only way to achieve this is to unite three areas of study – software modeling, software design studies, and studies of organizations – which, to date, have yielded significant results within their own spheres of influence, but which have seen relatively little crossover and therefore rather few attempts to feed an understanding of developers' and organizations' practices into the tools and techniques that are supposed to support them.

[sidebar: What is MDE anyway?]

In software engineering, a model is an abstraction of a running system. Modeling is undoubtedly a core activity in software development. The precise form of modeling varies widely – from whiteboard sketches to precise models that support code generation – but modeling in some form is a fundamental part of understanding, communicating and analysing software-intensive systems.

A number of terms have been used to describe approaches that focus on models. We follow Ameller [12] and others in defining model-driven development (MDD) as a subset of MDE. That is, MDD focuses on the generation of implementations from models. In contrast, MDE includes other uses of precise models to support the development process, such as model-driven reverse engineering and model-driven evolution. In particular, MDA is a particular form of MDD that uses OMG standards. Figure 1 neatly sums this up.

Participants in our study used a variety of MDE approaches. The majority of our interviewees focused on code generation from models (i.e., MDD); however, a significant number used models in some other way consistent with the vision of MDE. Only two interviewees claimed to be using MDA.

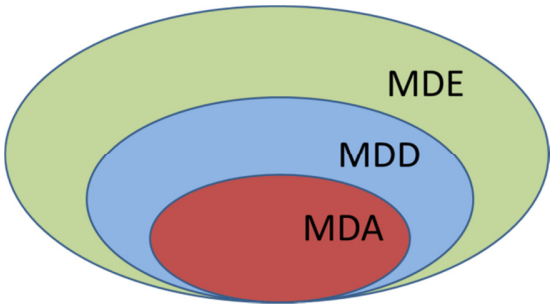

Figure 1: Different forms of model-driven engineering.

[sidebar: Methods]

We used an 'eclectic' set of research techniques, ranging from a widely disseminated questionnaire, to semi-structured interviews with industry professionals, to on-site observational studies of MDE practitioners at work.

The questionnaire was implemented online using Survey Monkey and comprised mostly closed questions, using both multiple choice and Likert scales for answers. In the questionnaire's preamble, we stressed that our target community was industrial practitioners with experience of using MDE in industry. The questionnaire was promoted through software engineering mailing lists and on the OMG's website.

We also carried out 22 semi-structured, in-depth interviews, mostly by telephone. The majority were generally positive about MDE, although we did identify a smaller number who had tried MDE but failed. The interviews lasted 45-60 minutes and included questions on the approach to MDE, the motivation for adopting it, the reasons for success/failure, and lessons learned. Interviews were recorded and transcribed; this produced over 150,000 words of written data describing MDE experiences.

More details about the study methodology are available [4].

References

[1] D. S. Frankel and J. Parodi, eds., *The MDA Journal: Model-Driven Architecture Straight From the Masters,* Meghan Kiffer, 2004.

[2] M. Petre, "UML in Practice," *Proc. 35th Int'l Conference on Software Engineering* (ICSE 2013), IEEE CS, *in press.*

[3] J. Woodcock, P. Larsen, J. Bicarregui and J. Fitzgerald, "Formal Methods: Practice and Experience," *ACM Computing Surveys,* 41(4), pp.1-36, 2009.

[4] J. Hutchinson, J. Whittle, M. Rouncefield and S. Kristoffersen, "Empirical Assessment of MDE in Industry," *Proc. 33rd Int'l Conference on Software Engineering.* (ICSE 2011), ACM, pp. 471-480.

[5] J. Hutchinson, M. Rouncefield, and J. Whittle, "Model Driven Engineering Practices in Industry," *Proc. 33rd Int'l Conference on Software Engineering.* (ICSE 2011), ACM, pp. 633-642.

[6] P. Mohagheghi and V. Dehlen, "Where is the Proof? – A Review of Experiences from Applying MDE in Industry," *Proc. 4th European Conference on Model Driven Architecture Foundations and Applications,* (ECMDA 2008), Springer LNCS 5095, pp. 432-443.

[7] E. Soloway and J.C. Spohrer, eds., *Studying the Novice Programmer*, Psychology Press, 1988.

[8] B. Curtis, "Fifteen years of psychology in software engineering: Individual differences and cognitive science," *Proc. 7th Int'l Conference on Software Engineering* (ICSE 1984), IEEE CS, pp. 97-106.

[9] J. Kramer, "Is Abstraction the Key to Computing?" *Communications of the ACM,* Apr 2007, pp. 36-42.

[10] S. Kelly and J-P. Tolvanen, *Domain-Specific Modeling: Enabling Full Code Generation*, Wiley, 2008.

[11] J. Whittle, J. Hutchinson and M. Rouncefield, "Mismatches between Industry Practice and Teaching of Model-Driven Software Development," *Proc. 14th Int'l Conference on Model Driven Engineering Languages and Systems, Workshops* (MODELS 2011), Springer LNCS 7167, pp. 40-47.

[12] D. Ameller, "SAD: Systematic Architecture Design, a semi-automatic method," Masters Thesis, Universitat Politècnica de Catalunya, 2010.

Jon Whittle is Chair of Software Engineering in the School of Computing and Communications at Lancaster University, UK. His research interests include software modeling, empirical software engineering, and social computing. He received a PhD in artificial intelligence from the University of Edinburgh. Contact him at j.n.whittle@lancaster.ac.uk.

John Hutchinson is a Senior Research Associate in the School of Computing and Communications at Lancaster University, UK. His research interests are in software modeling, service-oriented computing, and computational linguistics. He received his PhD in computer science from Lancaster University. Contact him at johnhutchinsonuk@gmail.com.

Mark Rouncefield is a Senior Research Fellow in the School of Computing and Communications at Lancaster University, UK, and is a Microsoft European Research Fellow. His research interests involve the empirical study of work, organization, human factors and interactive computer systems design. He received his PhD in sociology from Lancaster University. Contact him at m.rouncefield@lancaster.ac.uk.

Contact details of authors

Jon Whittle, Infolab21, School of Computing and Communications, South Drive, Lancaster University, Lancaster, LA1 4WA, United Kingdom. j.n.whittle@lancaster.ac.uk. +44 (0)1524 510307

Mark Rouncefield, Infolab21, School of Computing and Communications, South Drive, Lancaster University, Lancaster, LA1 4WA, United Kingdom. m.rouncefield@lancaster.ac.uk. +44 (0) 510305

John Hutchinson, Infolab21, School of Computing and Communications, South Drive, Lancaster University, Lancaster, LA1 4WA, United Kingdom. johnhutchinsonuk@gmail.com