The State of Practice in Model-Driven Engineering

Jon Whittle, John Hutchinson, and Mark Rouncefield, Lancaster University

// Despite lively debate over the past decade on the benefits and drawbacks of model-driven engineering (MDE), there have been few industry-wide studies of MDE in practice. A new study that surveyed 450 MDE practitioners and performed in-depth interviews with 22 more suggests that although MDE might be more widespread than commonly believed, developers rarely use it to generate whole systems. Rather, they apply MDE to develop key parts of a system. //

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 life-cycle phases. This wasn't 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 model-driven engineering (MDE) is a good way to develop software (see the "What Is MDE, Anyway?" sidebar). Some companies have reported great success with it, whereas others have failed horribly. What's missing is an industry-wide, independent study of MDE in practice, highlighting the factors that lead to success or failure. Although there have been a few prior surveys of modeling in industry, they've focused on only one aspect of modeling, such as the use of UML[2] or formal models.[3]

In this article, we report on a new study of MDE practice that covers a broad range of experiences. In particular, we focus on identifying MDE's success and failure factors. We surveyed 450 MDE practitioners and interviewed 22 more from 17 different companies representing 9 different industrial sectors (see the "Methods" sidebar for more information on the particulars). The study reflects a wide range of maturity levels with MDE: questionnaire respondents were equally split among those in early exploration phases, those carrying out their first MDE project, and those with many years' experience with MDE. Interviewees were typically very experienced with MDE.

We discovered several 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 it. Many of the lessons point to the fact that social and organizational

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 analyzing software-intensive systems.

Several terms have been used to describe approaches that focus on models. We follow David Ameller[1] and others in defining model-driven development (MDD) as a subset of MDE: MDD focuses on the generation of implementations from models. In contrast, model-driven engineering (MDE) includes other uses of precise models to support the development process, such as model-driven reverse engineering and model-driven evolution. In particular, model-driven architecture (MDA) is a particular form of MDD that uses the Object Management Group's (OMG) standards.

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

Reference

1. D. Ameller, "SAD: Systematic Architecture Design, A Semi-Automatic Method," master's thesis, Universitat Politècnica de Catalunya, 2010.

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 MDE or who are thinking of adopting it.

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 found that some form of MDE is practiced widely, across a diverse range of industries (including automotive, banking, printing, Web applications, and so on). The 450 questionnaire respondents, for example, were employed in a range of different roles (36 percent developers, 37 percent project managers) and represented a good spread of companies with respect to the number of people involved in development (52 percent at fewer than 100, and 19 percent at more than 1,000). Our interviews back up this finding and illustrate that MDE use is in fact widespread and used 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 a single company.

Perhaps surprisingly, the majority of MDE examples in our study followed domain-specific modeling paradigms: 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's 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's also widespread use of mini-DSLs, even within a single project. A clear challenge, then, is how to integrate multiple DSLs. Our participants tended to use them in combination with UML—in some cases, the DSL was a UML profile. Whatever the context, however, modeling languages requires significant customization before the languages 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 imposed by high-level management typically struggle; interviewees claimed that top-down management mandates fail if they don't have the buy-in of developers first. Consequently, 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's appropriate and combine it with other methods in a very flexible way.

Code Generation Doesn't Drive MDE

Surprisingly, it appears from our data that code generation isn't the key driver for adopting MDE. Although MDE is often considered to be synonymous with code generation (or at least model-driven development) and code generation itself is perceived to bring benefits such as productivity, reports of productivity gains vary widely (from a 27 percent loss to an 800 percent gain[6]). Most companies seem to experience productivity increases of between 20 to 30 percent.

Interestingly, our data suggests that such increases aren't considered significant enough to drive an MDE adoption effort: MDE brings with it increased training costs and substantial organizational change that easily offset 20 to 30 percent of productivity increases. This doesn't mean, however, that companies shouldn't adopt MDE. Rather, the interview data illustrates time and again that, although companies use code generation, they find other benefits to MDE that are much more important than 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.

METHODS

We used an eclectic set of research techniques, ranging from a widely disseminated questionnaire to semistructured interviews with industry professionals to on-site observational studies of model-driven engineering (MDE) practitioners at work.

The questionnaire was implemented online using SurveyMonkey 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 in using MDE in industry. The questionnaire was promoted through software engineering mailing lists and on the OMG website (www.omg.org).

We also carried out 22 semistructured, 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 to 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 more than 150,000 words of written data describing MDE experiences.

More details about the study methodology appear elsewhere.[1]

Reference

1. J. Hutchinson et al., "Empirical Assessment of MDE in Industry," *Proc. 33rd Int'l Conf. Software Eng.* (ICSE 2011), 2011, pp. 471–480.

The Real Benefits of MDE Are Holistic

So, if the real benefits of MDE aren't to be found in code generation, then where are they? 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 usually accepted philosophically, it often isn't 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're 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 doesn't impose a heavyweight and lengthy architecture definition process.

One company in our study 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 nonstandard 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 the adoption of other approaches such as agile. 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 isn't usually enough motivation for companies to risk adopting it; rather, companies that adopt MDE do so because it can enable business that otherwise wouldn't be possible.

An illustrative example is the experience of a well-known, global printer production company that consciously started using MDE 10 years ago. At that time, software was its bottleneck: a widely held perception promoted software as a limiting factor in getting a new generation of printers to market. However, after 10 years of evolving its use of MDE, the company now reports that software is no longer the problem. In other words, MDE enabled the company to be what it always should have been—a company focusing on printers, not software. This finding suggests a rethinking of the way we market MDE: not as a way to do things faster, but as a way to do new things.

The Psychology of MDE

In addition to offering these kinds of interesting insights into why some companies adopt MDE successfully and others don't, our data sheds light on the psychological and organizational aspects of MDE.

A phenomenon observed in other subfields of software engineering is that there can be significant individual differences between certain types of developers—for example, between novice and expert programmers.[7,8] We've observed similar effects with MDE.

Companies that target a particular domain are more likely to use MDE than companies that develop generic software.

First, 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.

Second, 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, the individuals who like to play with new coding technologies even outside of 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 because it automates many tasks.

We've observed similar findings in management. In particular, it appears that middle managers can be a bottleneck in adopting MDE. These managers are subordinate to senior managers but above operational staff. They typically have little strategic responsibility and therefore might 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 can offer a fundamental shift in global software development. Numerous companies reported that they reduced their offshoring activities as a result of MDE because they're now able to automate onshore tasks that were previously outsourced.

There appears to be some disagreement in industry as to whether everyone is capable of thinking abstractly. One company, for example, reported that the major bottleneck in its use of MDE is that it had to retrain hundreds of software coders, many of whom were unable to make the jump to abstract thinking. Other companies, in contrast, reported that only a very small percentage of coders are unable to think abstractly (a figure of 3 percent was quoted, but 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 made by others as well.[9]

There's 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). Because most MDE efforts are highly domain-specific, domain knowledge is crucial. However, success is less likely when a team has a division of skills between domain 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 metamodels of and code generators for the domain, as well as have the ability 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 isn't appropriate for every type of organization (at least not yet). Interestingly, companies that target a particular domain—automotive, printer interfaces, financial applications—are more likely to use MDE than companies that develop generic software, such as 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 might only do this informally using something like PowerPoint. As one of our interviewees stated, it's easier to move from these informal models to precise, computer-readable models than starting modeling from scratch.

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

MDE seems to question some of the assumptions about how organizations evaluate individuals and teams. For instance, architects have reported to us that they sometimes artificially increase the complexity of their models because their managers don't 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 doesn't fit with the MDE way of thinking. Typically, developers are hired based on what technologies they're familiar with rather than what domains they have knowledge of. But 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. We offer here our top five tips for success with MDE, based on the empirical data we gathered:

- *Keep domains tight and narrow.* In agreement with other sources,[10] we found that MDE works best when used to automate software engineering tasks in very narrow, tight domains. 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.
- *Put MDE on the critical path.* Perhaps counterintuitively, successful MDE practitioners argue that MDE should be tried on projects that can't fail—avoid the temptation to try out MDE on side projects that won't 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.
- *Be careful about gains offset elsewhere.* A company might not realize that gains in productivity achieved through code generation are lost 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

Typically, developers are hired based on what technologies they're familiar with rather than what domains they know.

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 couldn't get the tool to fit their processes, so they hacked it, messed with the generated code, and circumvented it when they had to.

- *Most projects fail at scale-up.* MDE works best when driven from the ground-up, but there comes a point when an organization needs to unite such grassroots efforts and effect organizational change. Not surprisingly, this is where problems start to arise, so managers should be careful to allocate appropriate resources during this transition phase.
- *Don't obsess about code generation.* MDE is often sold as a code generation solution. As we've seen, however, its real benefits don't necessarily lie in code generation. Companies would therefore be wise to consider the more holistic benefits that MDE can bring rather than focusing only on code generation.

As you can see, these top tips require some careful judgments to be made about the application of MDE; plenty of evaluation into how things are progressing seems to be another important part of successful adoption.

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 then iteratively refined into architecture, design, code, tests, and so on. 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 observed that attempts to introduce MDE into a company in this kind of top-down, organization-wide manner are fraught with difficulty. Those companies that do succeed invariably do so by driving MDE adoption from the grassroots: small teams of developers try out aspects of MDE, leading them to recognize reusable assets, and eventually MDE propagates to the organization as a whole. This way of working suggests that developers find it easier to come to grips with MDE when refactoring existing assets from the ground-up rather than in trying to abstract from above. This highlights 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 skillsets are usually taught in distinct parts of a computer science curriculum with little connection 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.[11]

These studies highlight that the fundamentals of modeling aren't well reflected in current modeling approaches.

Our study is the first wide-ranging industry study of MDE practice. It uncovered many companies who have had great success with MDE and some of the reasons why. It has also uncovered companies who have tried to apply MDE but gave up. Many of our findings are general development lessons and consistent with findings from other studies (for example, those 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 was 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 modeling languages or tools—developers cited more than 40 modeling languages and 100 tools as "regularly used" in our survey. A recent study[2] surveyed 50 software designers and found that these designers either didn't use UML at all or used it only 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—aren't 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. UML 2.0, for example, a major revision of the UML standard, didn't reflect the literature on empirical studies of software modeling or software design studies. Consequently, current approaches force developers and organizations to operate in a way that fits the approach instead of 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 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 that have seen relatively little crossover. To date, there have been too few attempts to feed an understanding of developers' and organizations' practices into the tools and techniques that are supposed to support them—addressing this gap could solve all kinds of problems and make modeling even more widely applicable than it currently is.

ABOUT THE AUTHORS

JON WHITTLE is Chair of Software Engineering in the School of Computing and Communications at Lancaster University. His research interests include software modeling, empirical software engineering, and social computing. Whittle 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. His research interests are in software modeling, software process evaluation, and computational linguistics. Hutchinson received a PhD in computer science from Lancaster University. Contact him at j.hutchinson@lancaster.ac.uk.

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

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 Conf. Software Eng.* (ICSE 13), 2013, pp. 722–731.
3. J. Woodcock et al., "Formal Methods: Practice and Experience," *ACM Computing Surveys*, vol. 41, no. 4, 2009, pp. 1–36.
4. J. Hutchinson et al., "Empirical Assessment of MDE in Industry," *Proc. 33rd Int'l Conf. Software Eng.* (ICSE 11), 2011, pp. 471–480.
5. J. Hutchinson, M. Rouncefield, and J. Whittle, "Model Driven Engineering Practices in Industry," *Proc. 33rd Int'l Conf. Software Eng.* (ICSE 11), 2011, 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 Conf. Model Driven Architecture Foundations and Applications*, (ECMDA 08), 2008, 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 Conf. Software Eng.* (ICSE 84), 1984, pp. 97–106.
9. J. Kramer, "Is Abstraction the Key to Computing?," *Comm. 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 and J. Hutchinson, "Mismatches between Industry Practice and Teaching of Model-Driven Software Development," *Models in Software Eng.*, LNCS 7167, Springer, 2012, pp. 40–47.