On the Industrial Adoption of Model Driven Engineering. Is your company ready for MDE?

Sobre la adopción industrial del Model Driven Engineering (MDE) ¿Está su empresa preparada para MDE?

Antonio Vallecillo[1]

[1] Universidad de Málaga, Spain

av@lcc.uma.es

ABSTRACT. Model Driven Engineering (MDE) is an approach to software development where models play a central role in all software engineering processes. Conceived to provide significant gains in productivity, portability, maintainability and interoperability, MDE is now starting to be effectively used in industry. Thus, companies are beginning to evaluate their possibilities for adopting it. This paper examines the current state of MDE in industry, summarizes the current obstacles for adoption, and discusses the advantages that it should bring to businesses and its limitations. Finally, some ideas for a smoother transition towards a wider adoption of MDE are outlined.

KEYWORDS: Model Driven Engineering, MDE, Software development, Software engineering processes, Adopting MDE, Software Engineering.

1. Introduction

1.1. Context

The ever-increasing complexity of software applications and the rapid changes in the technologies are posing serious challenges to our traditional software development methods and tools, which are having problems to cope with the new requirements. At the same time, most companies are now facing serious difficulties with their IT systems, in particular in the way in which they are developed, acquired and perceived by users:

– Customers are increasingly demanding better functionality, improved performance, more usable interfaces and enhanced integration facilities with their heterogeneous systems. These needs are normally very hard to incorporate into current applications and software production environments.

– Investments in software applications and IT systems are seldom recovered because of rapidly changing and volatile technologies – many enterprise applications become obsolete even before their costs are amortized.

– Migration and evolution of systems are costly and painful processes. This is why many companies decide to build sets of new layers atop their existing applications, fearing that a change in their core systems will quite simply ruin their business.

– Developing large applications by manual coding is still a very expensive and rather unpredictable process: very often software projects are, for the most part, overdue, error-prone, and blast all budget previsions.

– When software development is outsourced, the contracting company ends up becoming over-reliant on the application developer, losing most of its independence and control. In-house software development has several drawbacks too, because maintaining an IT department up-to-date with the latest tools and able to keep up with the ever-evolving technologies and experience represents a high cost – and does not necessarily solve the aforementioned problems, either (e.g., late, inefficient and over-budget projects).

– Once a system has been developed, most companies are only left with the code of the application, where their business knowledge and rules remain embedded or hard-wired. Thus, migration to a new technology normally means starting from scratch, because companies have no reusable and technology-independent blueprints of their business models.

Software Engineering is the discipline whose goal is the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software [IEEE90]. Thus, Software Engineering is currently trying to tame the inherent complexity of such processes and of the new requirements in different ways, which range from the creation of new programming languages (e.g. multi-paradigm ones) and development methods (e.g. agile) to new abstraction mechanisms and tools.

One of these new approaches is called Model-Driven Engineering (MDE) [Sch06, BCW12]. MDE is an approach to software development where the central focus of attention shifts from writing code by hand to dealing with higher levels of abstractions (models). Moreover, MDE broadens the scope of models beyond code generation, lifting them to first-class citizens of all software engineering processes: analysis, design, development, maintenance, modernization, etc.

MDE is based on three sound and time-proven principles: higher levels of abstraction, higher levels of automation, and standardization [B+04, BCW12, Sel03]. MDE raises the level of abstraction by enabling specifications that use different models to focus on different concerns and by automating the production of such specifications and the software that meets them; the use of standards aims to ensure the required interoperability between all MDE artefacts and tools.

MDE represents a big leap in Software Engineering, especially for developing and maintaining information systems, and implies some radical changes to the way software is constructed using traditional programming

methods. MDE has already proved to bring along significant gains to companies implementing it. But there are of course many companies still reluctant to adopt these changes to their proven internal processes and to their businesses in general: for many, MDE only represents yet another fad in software engineering. However, this time there is sufficient evidence that MDE is mature enough to provide significant benefits and value to companies, and to consolidate as an established approach for the development of industrial software. This paper discusses such evidences, and analyses the state of MDE adoption by the industry, the current difficulties for implementing it, as well as the advantages that it should bring to companies and its limitations.

1.2. A bit of history

It all started in the 90s with the advent of the Unified Modeling Language (UML), which provided designers with a standard set of graphical notations for representing software systems at a higher level of abstraction than code permitted, hence removing part of the accidental complexity of the problems being addressed. Models proved to be very useful for understanding the interesting characteristics of a system (either existing or to be built), predicting some of its properties, communicating with stakeholders, or guiding the implementation of the system (i.e., models as blueprints). Bran Selic perfectly defined the characteristics that software models have to satisfy to be useful [Sel03]: they have to be purposeful (i.e., address a specific set of concerns); abstract (emphasize important aspects while removing irrelevant ones); understandable (expressed in a form that is readily understood by observers); accurate (faithfully represent the system modeled); predictive (can be used to answer questions about the system modeled) and cost-effective (much cheaper and faster to develop and maintain than the actual system).

But it was soon clear that having models was not enough. They had to be manipulated and maintained in an automated way, and be an integral part of the software development processes. To address this issue, the Model-Driven Architecture (MDA) initiative was launched by the OMG in late 2000 to propose a new way to consider the development and maintenance of information systems [Wat08]. In the MDA approach, models are the key elements used to direct the course of understanding, design, analysis, construction, testing, deployment, operation, administration, maintenance, evolution, integration, acquisition and modernization of systems. MDA differentiates between platform-independent models (PIM) and platform-specific models (PSM). Thus, the basic functionality of the system can be separated from its final implementation and the business logic can be separated from the underlying platform technology. Models are described using metamodels that define the concepts of the languages in which models are written, as well as the well-formed rules that correct models should fulfil. MDA also introduced model transformations, as the key elements that enable the automated implementation of a system from the different models defined for it or, alternatively, enable reconstructing abstract models from legacy code to realize software modernization or migration.

We can look at MDA as one way of realizing MDE. More precisely, MDA is the approach proposed by the Object Management Group (OMG) to achieve MDE, using OMG standards (e.g., UML for modeling, MOF for metamodeling, QVT for model transformation, XMI for model interchange, etc.). The Eclipse ecosystem of programming and modelling tools (Eclipse Modeling Framework) is another MDE approach, where EMF provides metamodeling facilities; ATL is used as transformation language, etc.

MDA also permits defining further models of the system, each one focusing on a specific concern, and at the right level of abstraction. These specific models are described using Domain Specific Languages (DSLs) [FP10, Tol11, Vol13, KLRT13] and are related by model transformations that drive the automated code generation from the models.

Since the emergence of MDA much has happened in the field of system and software model engineering. A variety of new acronyms (MDD, MDE, MBE, MIC, ADM, SBA, MDI, etc.) are appearing to delimit the constantly extending scope of application of the core modeling techniques. In addition, the evolution towards modeling practices has joined with the Open Source Software movement in environments like Eclipse to give more strength to this paradigm shift.

For example, Model-Driven Development (MDD) principally focuses on the generation of implementations from models [Sel03, BVGR08], and can be considered as part of MDE, which is a wider discipline that covers all software engineering activities, beyond automated code generation, using models, metamodels and model transformations. In MBE, in turn, software models play an important role although they are not necessarily the key artifacts of the development – i.e. they do not "drive" the process [Cab14].

1.3. Some misconceptions about modeling and MDE

Before going any further, it would be useful to identify some typical misconceptions that many software engineers and developers still have about models and MDE.

a) Programs are not models, models are not programs. Many people think that programs are text-based sets of instructions that can be executed by a computer, and models are graphical, non-executable representations of a system. But programs can also be written in a visual form, and many models are in fact executable. There are also models described using textual notations (such as HUTN, for instance). Probably the only difference between programs and models lies on the level of abstraction, and the fact that a program must always contain enough information to be executed in a given platform (models may not). Interestingly, this has led to the term "prodel" (or "mogram" [Kle08]) to refer to either a program or a model, in cases in which they can be used interchangeably.

b) MDE = Modeling (or, similarly, MDA = Using UML). As mentioned above, MDA not only entails higher levels of abstraction using models as representations of a system, but also implies automation and standardization. The UML only provides notations for representing abstractions of a system, being just one part of MDA. In fact, models are not sufficient; they only leverage all their potential when used inside MDE processes. Similarly, MDE not only involves modeling the system (or some aspects of it), but also considers these models as software artefacts that need to be manipulated using model transformations or other model operations (difference, merge, etc.) for various purposes: generating implementations (i.e., code and configuration files), deriving analysis models to analyze the system properties, inferring models from existing code, or maintaining all related artefacts in synch, e.g., to realize round-trip engineering. Depending on the technological spaces involved in a model transformation, it can be classified as model-to-model (M2M), model-to-text (M2T) or text-to-model (T2M) transformation.

c) Modeling = Using UML. The UML is just one modeling language, but it is not the only one. In fact, Domain Specific Languages (DSLs) are now becoming commonplace for modeling the structure and behavior of systems [FP10, Tol11, BCW12, Vol13, KLRT13, BF14]. In addition, other modeling notations are commonly used in practice, including Flowcharts, the Entity-Relationship (E/R) notation, IDEF, Simulink, Modelica, the Goal Structuring Notation (GSN), BPMN, i*, System Context Diagrams (SCD), etc. Normally, software engineers use more than one modeling notation in their projects. A recent study [Tow14] showed that UML, Flowcharts, SCD and SysML were those more heavily used, and that, on average, software engineers use 5.5 different modeling languages at work. The complete list of modeling notations used goes up now beyond 35, which clearly confirms that UML is not the only modeling language in town.

d) MDE has nothing to do with business processes. MDE is not only about modeling the structure of systems or their behavioral and extra-functional aspects. Models can also be successfully used to represent business processes (using any existing notation apt for this purpose, such as BPMN, SPEM or UML), which can then be designed, analyzed, simulated, implemented, deployed, managed, re-engineered and maintained using existing MDE practices and tools. This discipline (called Business Process Modeling) is gaining acceptance in the industry as more usable and powerful supporting tools are becoming commonplace and readily available – and not only to specify and reason about business processes, but to generate code (see, e.g., [W14]).

e) MDE implies an "all-or-nothing" approach. Adopting the use of models and MDE practices needs not

be a binary decision across the company, or even within a project. Every company should decide "the right amount of modeling" for every project [Cab13], depending on its particular characteristics and on the project requirements.

2. Current state of MDE adoption

2.1. Empirical studies about the adoption and use of MDE in the industry

MDE was originally conceived to provide significant gains in the development of software in various aspects – chiefly in productivity, portability, maintainability and interoperability [KW03, CK08]. However, the industrial adoption and use of MDE still seems to be an exception rather than the norm [Sel12].

There are numerous verifiable examples of success stories with MDE, which are clear proofs of its feasibility and value. There are of course cases of failure, too. Thus, companies deciding whether to adopt MDE or not are often faced with confusion: successful cases tend to paint things too brightly, whereas cases of failure do not seem to have applied MDE properly [HWRK11]. In general, there has always been a lack of precise information about the level of adoption of MDE in industry, how it is used, and how (and when) it has been adopted. Besides, it is difficult to provide absolute measures of the real benefits of MDE and of its impact in current development projects.

To respond to the lack of accurate and contrasted information, the MDE community has been very actively working over the past few years on the compilation of empirical data and evidences about the use of design models in industry [A+6, BBB11, DP06, FBL10, G+11, GTA14, LCM06, Pet13, Pet14] and about the adoption of MDE practices [A+07, Avi14, AVT06, B+14, BB14, BC10, BF14, BLW05, CGR14, DGHC14, DGWC14, DMBD14, DPP14, DV10, FL08, HRW11, HWRK11, HWR14, KR06, KR08, KRK10, KTM12, LRTR13, MCMT14, MD08, MGS13, M+06, N+14, OMG09, Pez14, PPL13, R+06, SCG14, Sel12, Tow14, T+11, T+12, T+13, T+13b, WW06]. The goals of these studies have been to measure the penetration of MDE in industry, to analyze the benefits and problems found, the hurdles for adoption, and the actual value that MDE is able to deliver to companies. The following subsections summarize the findings of these papers, focusing on the aspects of interest to this study.

2.2. Models are not quite here yet

Regarding the use of design models in industry, all systematic studies seem to agree that the knowledge about modeling and its use are not widespread yet. In the main, developers do not make the best use of models and tend to perceive little or no value added in modeling. More precisely:

- Design models are not used very extensively, and where they are used, the use is informal and mostly without tool support; the notation is often not UML but many others.
- Models are used primarily as a communication and collaboration mechanism where there is a need to solve problems and/or get a joint understanding of the overall design in a group.
- Many software practitioners are still completely unaware of modeling notations and of MDE.
- Many software designers, who use UML, are using models only to illustrate and document the architecture of a software solution.
- The use of models decreases with an increase in developers' experience, and increases with higher levels of qualifications.
- Many software engineers currently use diagrams and simulations in their work but do not consider they are modeling.
- Models are seldom updated after initial creation, and are usually drawn on a white board or on paper.
- Complexity, coordination, and cost are ongoing issues.
- OCL [WK03] is rarely used in the modeling community. Only a few software engineers agree

with the importance of constraints, and the majority of modelers never define pre and post-conditions or invariants as part of their models. Constraints are either completely ignored or incorporated later at the code level, using programming languages.

– Tool support is still insufficient, in particular for model validation, simulation and interchange; for specifying constraints and correspondences between models; for compiling OCL or ALF statements to robust code libraries that can be used in production environments, etc.

– As long as there is no common agreement about a usable action language for specifying the behavior of models and executing them (e.g., ALF), developers are not really interested in another language which is restricted to constraint expressions. The benefits of models are not seen in comparison to programming languages by many current software developers.

– Modeling also requires a mindset change. The use of (the right) abstraction techniques is more difficult and less common than expected [SZ13], and modeling requires different skills other than programming.

As summarized by M. Petre [Pet14], *"One of the major reasons professional practitioners give for declining to use UML is that, after due consideration, they have concluded that it doesn't add sufficient value to warrant the cost of transition. Many practitioners already have a repertoire of tools and representations that has been thoughtfully developed and evolved over time to fit their effective practices."*

2.3. The value of MDE

The previous statement by Petre is a real and common criticism, but it normally applies to the inadequate introduction of models in the development processes of companies. As discussed before, models leverage all their potential benefits when used as part of MDE practices, and not in isolation. This is reinforced by most conducted studies, which show that models, when integrated within MDE processes and used in the right domains, can significantly improve both the quality of software and the productivity of the teams that develop it [A+07, Avi14, AVT06, B+14, BB14, BC10, BF14, BLW05, DGHC14, DGWC14, DPP14, DV10, MCMT14, MGS13, SCG14, WW06]. In particular, experiences of adoption of MDE in industrial settings report about significant gains:

– On average, projects achieved between 60% and 100% code-generation, contributing to significant productivity and quality improvement. In particular, productivity improvements ranged between 2X and 8X, when measured in terms of equivalent source lines of code or in function points.

– Models proved to be successful in obtaining an approximate 2.3X reduction in effort through the use of co-simulation, automatic code generation, and model testing.

– The use of scenario-based test generation tools yielded approximately 33% reduction in the effort required to develop test cases.

– Overall reduction in defects ranged between 1.2X and 4X, and detection happens much earlier in the development process, when they are less expensive to fix.

– The overall cost of quality decreased due to a decrease in inspection and testing times.

– Up to 80% of maintenance costs can be saved by working directly with models.

For example, in one case study at Motorola [BLW05], a 30X to 70X reduction was reported in the time needed to correctly fix a defect detected during system integration testing. This reduction was due to the ability to add a model test that illustrates the problem, fix the problem at the model level, test the fix by running a full regression test suite on the model itself, regenerate the code, and run the same regression test suite on the generated code – the time to do this was 24 hours or less, while achieving the same quality with several hundred thousand lines of hand code could have easily taken them one to two months.

Web engineering is another domain where MDE has demonstrated many of its potential benefits for building complex Web-based information systems (leading what is now called *"Model-Driven Web Engineering"*, MDWE). MDWE has proven to be economically profitable and sustainable in many real cases, with peak pro-

ductivity rates reaching five times the number of delivered function points per staff/month of traditional programming languages like Java [A+07]. Empirical studies in large companies (e.g., ACER) also show that between 2 and 3 months are required to train developers and make them fully productive with the MDWE tools; but once they master them, gains in productivity can reach 60% of development time, and up to 80% in maintenance and evolution costs [BB14] – due to the automatic generation of code from models (it can be fully automated in many MDWE applications), and the possibility of working at the model level for maintaining the systems [BF14].

In general, MDE has been successfully applied in a broad range of companies, and in a number of different domains including telecommunications, business and finance, defense and aerospace, manufacturing, health, and web information systems. The size of companies that adopt MDE for their own developments is normally large. Very small firms, in turn, are successful in using MDE for conducting consultancy and software development for others: there is a growing market for companies providing Domain-Specific Modeling environments, tools, consultancy and support. Example of such successful companies include MetaCase (https://www.metacase.com/), Obeo (http://www.obeo.fr/), Atego (http://www.atego.com), Sodious (http://sodius.com/), WebRatio (http://www.webratio.com/), Icinetic (http://www.icinetic.com/), Itemis (http://www.itemis.com/) and Mia-Software (http://www.mia-software.com/), among many others.[1]

Apart from gains in quality and productivity, significant improvements in reuse and maintainability have also been reported. The degree of modeling maturity has evolved from informal whiteboard or napkin-modeling to formal modeling with simulation, code generation, test-case reuse, automated marshaling code generation, etc., which are the processes where the real value and benefits of models and MDE can be perceived.

Moreover, large companies that have successfully adopted MDE clearly recognize that models, in their own right, have become very valuable assets. Models permit capturing the company business rules, processes and knowledge, independently of the technologies used to implement them, and at the right level of abstraction. These high-level models now allow companies to analyze and reason about their processes and business rules using specialized tools, much more efficiently than before.

3. Adopting MDE

3.1. Hurdles to MDE adoption

In light of these results, one wonders why companies are not adopting MDE yet. Far from this, MDE is not a dominant software development paradigm and is still seen by many practitioners as "either unproven or, even worse, as yet another waning fad in a discipline that already suffers from an excess of silver bullets" [Sel12].

The fact is that although the benefits of MDE are frequently cited and are often considered to be obvious, there are reasons why MDE may have detrimental effects. One of them is that MDE involves dependent activities that have both positive and negative effects. For example, automatic code generation can have a positive effect on productivity. But the extra effort required to develop the models, along with the possible need to make manual modifications, may have a negative effect. The balance between these two effects is related to context, and needs to be carefully considered [HRW11, HWR14].

Furthermore, there are significant hurdles that may hinder the adoption of MDE in practice. Although many of these are technical in nature, the main ones come about for organizational, cultural and economic reasons [Sel08, HRW11, Sel12, HWR14, MCMT14].

[1] The list shows just a few companies as examples; it is by no means intended to be complete or exhaustive.

Examples of Technical hurdles include:

– Lack of mature tools (scalable, robust, usable, efficient and interoperable). As occurs with programs, if no good compilers/debuggers/IDEs are available for a programming language, nobody beside academics will ever use it (especially in production environments).
– Lack of well-defined semantics (of models, metamodels, and transformations) and of a sound theory of MDE.
– Lack of documented and proven MDE processes.
– No "MDE Good Practices" manuals available. Only success stories of various companies (mostly large companies) have been documented. However, these reports normally give no indication on whether the changes implemented in one company will work for others, how to apply them, or when.
– Lack of model validation tools.
– It is by no means guaranteed that higher abstraction levels lead to better software [Kra07].
– Different flavors of MDE exist and choosing the right variant for a company's business is critical (and not obvious) to a project's success [HRW11].

Examples or Cultural hurdles include:

– Lack of education, team experience and skills sets in most developers and software practitioners. MDE is not only a change in technology, but a complete paradigm shift.
– Too much emphasis on technology and not enough on technology users and their needs.
– Inadequate or flawed information about MDE concepts, goals, tools and real achievements – for many companies it is not clear whether MDE is just an academic theory, the tool vendor's sales pitch or if there are indeed many organizations successfully using MDE to realize measurable benefits on real software engineering projects.
– Lack of "systems" perspective and lack of abstraction skills.
– Conservative mindset of many software practitioners; resistance to technological change, even if the new technology can lead to better results.
– Conservative mindset of managers, normally reluctant to drastic changes in development processes, methods and tools.

Economic hurdles include:

– Fear of excessive over-costs.
– Current business climate heavily focused on short-term gain discourages investment in new methods and tools [Sel08].
– Upfront investments difficult to quantify in the mid-term.
– Development teams' re-education and training can indeed be expensive (since it may imply changing their mindsets, not only their methods and tools).

3.2. Pros and Cons: Technical Issues

Something critical for a company before deciding whether or not to adopt MDE (either in full, in a project, or in a set of related projects), is to understand the pros and cons that such an adoption would imply. We have previously mentioned (Section 3.1) that MDE may also have detrimental effects, and that each decision can entail advantages and disadvantages, as well as savings and associated costs, which need to be carefully evaluated.

These tradeoffs can have a significant impact on different aspects of the development process and on the final product quality (see, e.g. [BLW05, DV10, HWRK11]):

– Requirements elicitation time is reduced by the use of models and the right domain specific lan-

guages, which use notations closer to domain experts and to the system stakeholders; but these efforts can be completely wasted without the right tools that make effective use of them in the whole process – not to mention the time needed to master these new languages.

– Design time is reduced by the use of the right languages and tools, which work at the appropriate level of abstraction; but the learning curve needed to master these notations and tools is not negligible and it may require several projects before it starts to pay off (see, e.g., [Cab11, DV10]).

– Development time is reduced by automatic code generation; but it is also increased by the need of developing models and implementing model transformations. Here we also need to mention that higher level artefacts are easier to reuse in subsequent projects (for instance, the models or even the model transformations).

– Protection of investment. Depending on the domain, developing models could be a more difficult, expensive and time consuming task than programming; however for a company, the value of models can be much higher than programs when used within MDE environments: models become platform-independent; they can be used to generate code into different platforms; have a longer lifespan than code; are more reusable, and can facilitate the implementation of novel applications (such as Simulation-based Acquisition [Nav13] or Model-driven Interoperability, for example). In addition, high-level models allow companies to become more independent from software vendors and technology provides, hence protecting their investment in modeling.

– Testing time is reduced by reducing defects in code and by the use of model-based techniques; but it can be significantly increased by the effort needed to test model transformations and validate models (mature debugging and validation tools for models and model transformations are not commonplace). Besides, testing times can also be significantly reduced by the use of models and the application of Model-based Testing (MBT) techniques. Again, the introduction of MBT requires the implementation of changes in the testing processes of the company, which cannot be neglected.

– Maintenance time is reduced since it is achieved at the model level; but new efforts are required to automatically keep models and code in synch.

– Integration time is reduced because it is done at model level, and all gluing and mediation artefacts can be automatically generated; but increased by the need to define correspondences between the models, and implementing the transformations that generate and deploy all artefacts.

By looking at these tradeoffs, we can synthesize from them three major factors that do have a significant impact on the success of an MDE project:

– First, the level of knowledge and expertise of the company in the problem domain (this only depends on the particular project), and the value that the company sees in solving the problem.

– Second, the level of training (and education) of the development team on the solution domain (Modeling and MDE, in this case), and their familiarity with it.

– Third, the availability of the right languages and tools for solving the problem and implementing the solution. It may be the case that the tools are not yet ready for industrial use, being just mere prototypes or toy-ish applications that do not scale, are not robust enough, or are unmaintainable.

Interestingly, these three aspects are related to the determinant technical factors that Mohagheghi et al. identify in [MGS13] for the adoption of MDE in industrial settings: perceived usefulness, ease of use and maturity of the tools.

Therefore, each company, depending on the particular project, on its perceived value and ROI, and on its capability to be resolved using MDE, should decide whether to adopt MDE or not, for which projects, and with which development team. The criteria to use, and the measures needed to assess the success of the project should be clearly established since the beginning, and continuously evaluated throughout the overall process.

Note that outsourcing part of these projects could be an option in some early projects, in the case of a lack

of in-house resources or expertise. For example, external MDE experts can be contracted to introduce the company development team to this new paradigm, helping to train them in MDE; or simply subcontracted for developing part of one project using MDE, so that our company could evaluate its use and its results in our particular application domain.

3.3. Pros and Cons: Beyond Technical Issues

Apart from the technical issues mentioned above, companies should take into account other factors that also affect the adoption of MDE practices. In particular, these factors include a number of cultural, social, economic and organizational issues, which also influence the relative success or failure of MDE practices. Most of these issues have already been mentioned in Section 3.1, when listing the general hurdles to MDE adoption. In contrast to the technical issues, they are normally the more critical ones to overcome [Sel12, HWR13].

Some of them could be solved by counting on better educated software engineers: not only in technical questions but also in cultural, human and business matters – allowing them to view technology more as a means rather than as an end in itself.

In the long term, education should be facilitated through changes in academic curricula and investment in foundational research: the competencies needed for dealing with the new concepts should be part of the education provided by universities and other teaching institutions. This is similar to the introduction 25 years ago of object-oriented programming (OOP) in the curricula of computer scientists and software engineers. Current managers no longer question the value of OOP, because they know it well and it forms part of their culture. More importantly, in MDE this knowledge accounts not only for modeling theories, concepts and tools, but also for the business aspects and the workings of the marketplace. At the same time, more research on a theory of MDE is required, to ensure that the corresponding technology and methods are well understood, useful, and dependable [Sel12].

In the meantime, companies have two choices: they can either invest in educating their software engineering teams, or subcontract external experts that introduce these concepts and expertise into the company know-how and practices (by, e.g., jointly working with local teams on MDE projects). Alternatively, they can outsource the MDE development to third parties, while they continue to use their previous development methods and tools. It is therefore up the company to estimate the cost of training and re-educating their development teams, versus outsourcing the services they need.

Although better education can tackle most of the social and cultural issues mentioned in Section 3.1, economic issues also need to be addressed. Normally, they can be mitigated by a gradual introduction of MDE practices in the company. In this way, pilot projects can serve to assess the actual costs of the projects in a controlled way, hence alleviating possible risks and unwelcome surprises. The pilot projects could also allow managers to estimate the costs of MDE adoption in other environments, the upfront investments needed, and their expected ROI.

Finally, we also have to consider the managerial and organizational aspects that also influence the successful adoption and deployment of any new paradigm, in this case MDE. Several authors [Sel08, HRW11, Sel12, HWR14, MCMT14] have systematically analyzed various industrial case studies, from different domains, and they have all identified the same essential organizational aspects of any successful MDE deployment.

– Motivation: The company should have a clear reason (not only technical) that drives the change, and such a reason should be shared by all employees.

– Commitment: the organization must be fully committed to making the project a success; otherwise the doubts can ruin the adoption process when initial problems appear.

– Innovation: the organization must be willing to adopt new development processes, integrate them with their own processes, and adapt the existing ones if required.

– Gradual adoption: MDE should be adopted in a progressive and iterative approach; pilots play a key role here, versus all-or-nothing strategies.

– Conscious adoption: Adoption of MDE requires a real understanding of the necessary process and technology changes, the consequences of the adoption, and the associated costs and benefits.

– Involvement: The decision to adopt MDE cannot be made by a single individual, group or department. All principal players in the company should feel part of the change.

– Business focus: ultimately, successful MDE has to have a clear business focus. MDE should not only be adopted for technical reasons, but as a solution to new commercial and organizational challenges.

3.4. And now?

So far we have discussed the current state of the adoption of MDE in industry, the current hurdles for that adoption, and the pros and cons that companies considering MDE should weigh up. It is now up to each individual company to decide whether or not to adopt MDE, how, and when.

Although there are no universal answers to these issues, the following list of questions could be helpful to companies in order to make decisions on the adoption of MDE in their organizations.

1. Does my company understand the implications of the necessary process change to adopt MDE in the project? [What are those changes? What are the implications?]
2. Is the project I'm considering able to be carried out using MDE? [Why? How?]
3. Does my company have development teams trained in MDE concepts, tools and processes that can carry out the project? [Who are they?]
4. Do I have the MDE tools that I need for carrying out the project? [Which ones?]
5. Are the MDE tools that I need mature enough for the job? [Why?]
6. Is the MDE approach that I am considering easy to use by my development teams, and to integrate with the rest of my company's notations, processes and tools? [Why?]
7. Is the company management seriously committed to the adoption of MDE? [How much?]
8. Do I know how to estimate the costs and duration of the project, and the expected ROI? [How?]
9. Have I established the criteria to use and the metrics needed to measure the success of the project? [What are the criteria and the metrics?]
10. Is the project technically feasible and economically viable? [Why?]
11. Have I considered outsourcing the project to a company with expertise in MDE? [Why?]
12. Do I have any real need to adopt MDE for the project? [Which one? Is it urgent?]

4. A new paradigm shift

By looking from a high-level point of view at all the issues and tradeoffs outlined above, they represent something that we have seen before. They resemble the situation we went through in the late 80s when Object-Oriented Programming (OOP) emerged and started replacing the existing and well-established Structured Programming: the strong acceptance by Academia, the initial doubts by large companies, the slow emergence of useful tools, and then the wide adoption by industry. Basically, we are just facing a new paradigm shift, in which models are the new first-class citizens of our theories, methods, tools and techniques – the same role objects took when OOP was adopted. We should learn from that experience.

OOP was originally created in the late 1960s at the Norwegian Computing Center in Oslo, when Ole-Johan Dahl and Kristen Nygaard developed the Simula I and Simula 67 programming languages. It took Objects almost 30 years to be start to become mainstream: almost 14 years to be known in the industry (in 1981, with the publication in the Byte magazine of a special issue dedicated to Smalltalk [Byt81]), and then another 14 years to become the dominant programming methodology in the mid-90s, as acknowledged by the Gartner Group in 1995 [Gar05].

To analyze the level of technology adoption of MDE, let's examine Figure 1, which plots together the Gartner "Hype Cycle" (HC) model [Gar14] and the "Technology Adoption Lifecycle" (TALC) model popularized by Everett Rogers and Geoffrey Moore [Moo14].

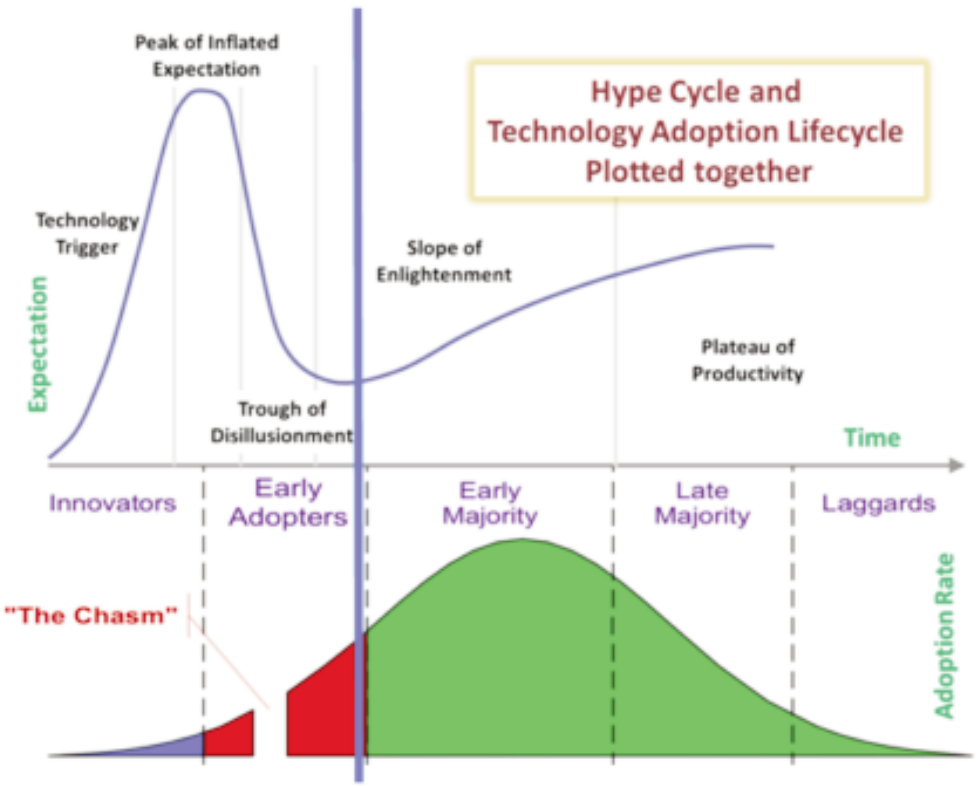

Figure 1. The "Hype Cycle" (HC) and the "Technology Adoption Lifecycle" (TALC) models plotted together (from: http://setandbma.wordpress.com/2012/05/28/technology-adoption-shift/), with the current position of Model Driven Engineering indicated by a thick blue line.

Most studies (e.g., [CGR14, Tow14]) tend to coincide that MDE has just passed the Trough of Disillusionment of the HC model (where it was in 2012 [Gar13]), and is now starting the second period of growth. This is corroborated by several facts, which are precisely the ones that characterize the start of the Slope of Enlightenment of the Hype Cycle model [Gar14]:

– the market has already started to mature, and it has become more realistic about how the new technology can be useful to organizations;

– many companies have already commenced to use the technology, and more organizations fund pilot projects;

– there are more instances and real case studies of how the new technology can benefit the industry starting to materialize and becoming more widely understood (see, e.g., the list of references in the bibliography at the end of the paper);

– second and third-generation products are appearing from technology providers: tools are starting to become more usable, scalable and robust;

– conservative companies still remain cautious.

Considering the TALC model, presently MDE has already passed the Chasm, surviving the most critical

phase, and is now at the beginning of what is called "The Bowling Alley". At this stage, where we are now, market momentum picks up as early pragmatists overcome their reluctance and decide to adopt the new technology to solve niche-specific problems. These companies decide to leave their comfort zone to find solutions for particular applications in which they want to stand out from the competition, or for which they find no other solution.

Knowing where we are, we can benefit from previous experiences of technology adoptions. In particular, the key to success at this stage is to use a gradual and iterative approach to adoption. First, to provide a complete solution for one application (or domain) while identifying closely aligned applications that could benefit from a similar solution [Moo14]. When the momentum from successfully delivering solutions for the first application (the lead bowling pin) is felt, this momentum is leveraged into adjacent applications. By dominating several applications, the company is now able and ready to decide when to use MDE and when not, estimate the benefits and costs for the adoption in each case, and make an optimal use of this new paradigm.

To estimate the adoption of MDE in the coming years, we can easily draw a parallel between OOP and MDE, assuming that there is a difference of 25 years between them (although OOP was used in academia since the 70s, it was not known by the industry until 1981, and started to be adopted in the 90s; in the case of MDE, OMG launched MDA in 2001 but MDE was not fully recognized by the industry until 2005 [Gar05, Sch2006]) and that the pace of adoption by industry will be similar, both being comparable technologies. In this respect, the situation of OOP in 1991 was very similar to the situation of MDE now, ending the period of Early Adoption and about to start climbing the second slope of the Hype Cycle. Thus, we estimate that the situation of MDE in 5 years' time (around 2020) will coincide with the position of OOP in the chart in 1995. That is, if MDE technologies keep maturing and developing at the same pace, we can expect MDE to reach the peak of adoption by the early majority of the industry in 2020.

As it happened with OOP, we will know that MDE has been successfully adopted when it becomes transparent and we no longer discuss about its application: it becomes an integral part of our curricula, and fully integrated into our software engineering processes and practices. This does not mean that we apply it in all projects, but that we know how, and when, to apply it.

In this sense, all indicators show that MDE has already passed the turning point, and its adoption by industry is progressively expanding. Thus, companies should now start to evaluate their options and opportunities for adopting MDE, and the strategies to make this happen.

5. How can we help?

Regardless of what individual companies decide, there are also collective actions to be carried out by different groups to advance MDE and to exploit its full potential.

In the first place, the competencies needed for dealing with MDE and with its new concepts and mechanisms, should be part of the education provided by Universities and other teaching institutions. Most universities already incorporate UML and modeling courses in their curricula. MDD and MDE are becoming popular subjects in many master courses too, with many masters entirely devoted to MDE, and several universities already offering subjects on model-driven matters at graduate level. Moreover, the use of Models and Model-driven techniques has been incorporated in the new edition of the Software Engineering Body of Knowledge (SWEBOK) [PF14]. The presence of MDE topics in most Software Engineering curricula today is one of the definitive signs of the relevance of MDE and the best guarantee that the adoption of MDE by a company is worth the investment in the medium and long term.

But universities should incorporate not only MDE theories, concepts and tools into their Software Engineering curricula, but also business aspects and the workings of the software marketplace [Sel12, Pai14].

As we have seen, technology should not be an end in itself, but just a means, and should always be considered within the business context it is developed for and applied in.

At the same time, more research is required on MDE. New methods and tools to cope with the growing demands imposed by new technologies and increasingly complex systems are needed. More importantly, there is an urgent need for research on a sound theory on MDE to ensure that the corresponding technologies and methods are well understood and reliable [Sel12].

Finally, companies should take the initiative now and become the drivers of the change. Of course, MDE technologies are still not perfect; they need to be significantly improved to be more useful in industry. New problems will also appear as soon as MDE is more widely used in new projects and in different domains. New challenges will need to be addressed too, to meet new market demands. In this sense, companies must definitely work together with universities and research centers to help improve the usability, usefulness and applicability of MDE in industry.

6. Conclusions

As occurs with any major technology, the evolution of MDE has reached the critical point where industry needs to decide whether to adopt it or not. On the one hand, it seems an unavoidable process because the benefits and advantages seem to outweigh the costs and limitations, it is now part of our software engineers' education, and the industry is starting to successfully embrace it. On the other hand, many large companies are still reluctant to consider these changes of technology paradigms because they represent a revolution to their proven internal processes and to their businesses in general.

In this paper we have summarized the state of MDE adoption by the industry, the current difficulties for implementing it, as well as the advantages that it should bring to companies and its limitations.

Whether or not the industry eventually accepts MDE as a major technology solution for the engineering of its software systems is always difficult to predict, despite the recognizable signs currently provided by all indicators. Experience is showing that MDE is able to successfully address many of the traditional shortcomings in software development and maintenance of software systems, and to provide significant benefits and value to businesses. The key to progress now lies in all parties working together to make these new technologies available so that software applications can be more systematically, quantifiably, and predictably developed, resulting in more robust, reliable and useful software systems.

Acknowledgements

Thanks to the reviewers of this paper, and especially to David Ameller, Loli Burgueño, Jordi Cabot, Jesús García-Molina and Manuel Wimmer for their valuable comments and suggestions on earlier drafts of this work.

Cómo citar este artículo / How to cite this paper

Vallecillo, A. (2014). On the Industrial Adoption of Model Driven Engineering. Is your company ready for MDE?. International Journal of Information Systems and Software Engineering for Big Companies (IJISEBC), Vol. 1, Num. 1, pp. 52-68. Consultado el [dd/mm/aaaa] en www.ijisebc.com

References

a) On the use of MDE in industrial settings

[A+07] Acerbis, R., Bongio, A., Brambilla, M., Tisi, M., Ceri, S., Tosetti, E. Developing eBusiness Solutions with a Model Driven Approach: The Case of Acer EMEA. In Proc. of ICWE 2007, LNCS 4607, pp. 539–544, Springer, 2007.

[Avi14] Ávila-García, O. Optimización del rendimiento de aplicaciones ABAP. Novática, 228: 29-35, 2014.

[AVT06] Afonso, M., Vogel, R., and Teixeira, J. From code-centric to model-centric software engineering: practical case study of MDD infusion in a systems integration company, in Workshop on MBD/MOMPES, 2006.

[B+14] Büttner, F., Bartels, U., Hamann, L., Hofrichter, O., Kuhlmann, M., Gogolla, M., Rabe, L., Steimke, F., Rabenstein, Y., Stosiek, A. Model-driven standardization of public authority data interchange. Sci. Comput. Program. 89:162-175, 2014.

[BB14] Brambilla, M., Butti, S. Quince años de Desarrollo Industrial Dirigido por Modelos de aplicaciones Front-End: desde WebML hasta WebRatio e IFML Novática, 228: 26-43, 2014

[BC10] Bone, M., Cloutier, R. The current state of model based system engineering: results from the OMG SysML request for information 2009. In: Proc. of the 8th Conference on Systems Engineering Research (CSER), March 2010.

[BF14] Brambilla, M., Fraternali, P. Large-scale Model-Driven Engineering of web user interaction: The WebML and WebRatio experience. Sci. Comput. Program. 89:71-87, 2014.

[BLW05] Baker, P., Loh, S., and Weil,F. Model-Driven Engineering in a Large Industrial Context — Motorola Case Study. In Proc. of MoDELS 2005, LNCS 3713, pp. 476–491, Springer, 2005.

[Cab11] Cabot, J. MDD pays of in the mid-term: an industrial experiment. Post in the Modeling Languages blog, http://modeling-languages.com/mdd-pays-mid-term-industrial-experiment/, 2011

[Cab13] Cabot, J. UML adoption in practice: has anything changed in the last decade? Post in the Modeling Languages blog, http://modeling-languages.com/uml-adoption-in-practice-has-anything-changed-in-the-last-decade/, 2013.

[CGR14] Cabot, J., García-Molina, J., Rossi, G. Adopción industrial de la ingeniería del software dirigida por modelos. Novática, Núm. 228. Abril-junio 2014.

[DGHC14] Davies, J., Gibbons,J., Harris, S., Crichton, C. The CancerGrid experience: Metadata-based model-driven engineering for clinical trials. Sci. Comput. Program. 89:126-143, 2014.

[DGWC14] Davies, J., Gibbons, J., Welch, J., Crichton, E. Model-driven engineering of information systems: 10 years and 1000 versions. Sci. Comput. Program. 89:88-104, 2014.

[DMBD14] Drapeau, S., Madiot, F., Brazeau, JF., Dugré, PL. SmartEA: Una herramienta de Arquitectura Empresarial basada en las técnicas MDE. Novática, 228:21-28, 2014.

[DPP14] Di Ruscio, D., Paige, R.F., Pierantonio, A. Guest editorial to the special issue on Success Stories in Model Driven Engineering. Sci. Comput. Program. 89:69-70, 2014.

[DV10] Diaz, O., Villoria, F.M. Generating blogs out of product catalogues: An MDE approach. Journal of Systems and Software, 83(10):1970-1982, 2010.

[FL08] Forward, A., Lethbridge, T.C., 2008. Problems and opportunities for model-centric versus code-centric software development: a survey of software professionals. In: International Workshop on Models in Software Engineering (MiSE 2008), pp.27–32, ACM Press, 2008.

[HRW11] Hutchinson, J., Rouncefield, M., Whittle, J. Model-driven engineering practices in industry. In Proc. of ICSE 2011, pp. 633-642, ACM, 2011.

[HWR14] Hutchinson, J., Whittle, J., Rouncefield, M. Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure. Sci. Comput. Program. 89:144-161, 2014.

[HWRK11] Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S. Empirical assessment of MDE in industry. In Proc. of ICSE 2011, pp. 471-480, ACM, 2011

[KR06] Kulkarni, V., Reddy, S. Introducing MDA in a large IT consultancy organization. In Proc. of ASPEC 2006, pp. 419-426, Dec. 2006.

[KR08] Kulkarni, V., Reddy, S. A model-driven approach for developing business applications – experience, lessons learnt and a way forward. In Proc. of 1st India Software Engineering Conference, pp. 21-28, Feb. 2008.

[KRR10] Kulkarni, V., Reddy, S., Rajbhoj, A. Scaling up model-driven engineering – experience and lessons learnt. In Proc. of MoDELS 2010, LNCS 6395, pp. 331-345, 2010.

[KTM12] Kuhn, A., Thompson, A. and Murphy, G. An Exploratory Study of Forces and Frictions Affecting Large-Scale Model-Driven Development. LNCS 7590, p. 352-367, Springer, 2012.

[LRTR13] Leotta, M., Ricca, F., Torchiano, M., Reggio, G. Empirical evaluation of UML-based model-driven techniques: Poster paper. RCIS 2013, pp.1-2, 2013

[R+06] Rios, E., Bozheva, T., Bediaga, A., Guilloreau, N. MDD Maturity Model: A Roadmap for Introducing Model-Driven Development. In Proc. of ECMDA-FA 2006, pp. 78-89, 2006.

[M+6] Mansell, J.X., Bediaga, A., Vogel, R., Mantel, K. A Process Framework for the Successful Adoption of Model Driven Development. In Proc. of ECMDA-FA 2006, pp. 90-100, 2006.

[MCMT14] Murguzur, A., De Carlos, X., Mendialdua, X., Trujillo, S. Ingeniería del Software Dirigida por Modelos: Adopción industrial para software empotrado. Novática, 228:44-50, 2014.

[MD08] Mohagheghi, P., Dehlen, V. Where is the proof? A review of experiences from applying MDE in industry. In Proc. of MDA-FA 2008, LNCS 5095, pp. 432-443, Springer, 2008.

[MGS13] Mohagheghi, P., Gilani, W., Stefanescu, A., Fernandez, M., An empirical study of the state of the practice and acceptance of

model-driven engineering in four industrial cases. Empirical Software Engineering 18(1):89-116, 2013.
[N+14] András Nádas, Tihamer Levendovszky, Ethan K. Jackson, István Madari, Janos Sztipanovits: A model-integrated authoring environment for privacy policies. Sci. Comput. Program. 89:105-125, 2014.
[OMG09] The Object Management Group. Compilation of SysML RFI - Final Report. OMG Document syseng/2009-06-01 (2009)
[Pai14] Paige, R. Ingeniería de Software con modelos: Panorama actual y futuros retos. Novática, 228:11-15, 2014
[Pez14] Pezuela, C. ARTIST: Una solución global para la modernización de software hacia el cloud. Novática, 228:16-20, 2014
[PPL13] Papotti, P.E., do Prado, A.F., Lopes de Souza, W., Cirilo, C.E. and Pires, L.F. A Quantitative Analysis of Model-Driven Code Generation through Software Experimentation. In Proc. of CAiSE 2013, LNCS 7908, pp. 321-337, Springer, 2013.
[SCG14] Sánchez Cuadrado, J., Cánovas Izquierdo, J.L., García Molina, J. Applying model-driven engineering in small software enterprises. Sci. Comput. Program. 89: 176-198 (2014)
[Sel12] Bran Selic. What will it take? A view on adoption of model-based methods in practice. Software and System Modeling 11(4):513-526, 2012.
[T+11] Torchiano, M., Tomassetti, F., Ricca, F., Tiso, A., Reggio, G. Preliminary findings from a survey on the MD state of the practice. In Proc. of ESEM 2011, pp. 372-375, 2011.
[T+12] Tomassetti, F., Torchiano, M., Tiso, A., Ricca, F., Reggio, G. Maturity of software modelling and model driven engineering: A survey in the Italian industry. In Proc. of EASE 2012, pp. 91-100, 2012
[T+13] Telinski, L, Agner, W., Soares, I.W., Stadzisz, P.C., Simão, J.M. A Brazilian survey on UML and model-driven practices for embedded software development, Journal of Systems and Software, 86(4): 997-1005, 2013.
[T+13b] Torchiano, M., Tomassetti, F., Ricca, F., Tiso, A., Reggio, G. Relevance, benefits, and problems of software modelling and model driven techniques - A survey in the Italian industry. Journal of Systems and Software 86(8): 2110-2126, 2013.
[Tow14] Towers, J. Model Based Systems Engineering 'The State of the Nation'. Presentation at the Atego seminar "Realising the Benefits of Model-Based Systems Engineering", 2014. http://www.atego.com/downloads/slides/1403/1JT_The-State-of-the-Nation.pdf
[WW06] Weigert, T., Weil, F. Practical experience in using model-driven engineering to develop trustworthy systems. In Proc. of IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC'06), pp. 208–217, 2006.

b) On the use of UML and design models in industrial settings

[A+06] Anda, B., Hansen, K., Gullesen, I., and Thorsen, H. Experiences from Introducing UML-based Development in a Large Safety-Critical Project, Emiprical Software Engineering 11:555-581, 2006.
[BBB11] Budgen, D., Burn, A.J., Brereton, O.P., Kitchenham, B.A., Pretorius, R. Empirical evidence about the UML: a systematic literature review. Software – Practice and Experience 41(4):363–392, 2011.
[DP06] Brian Dobing and Jeffrey Parsons. How UML is used. Commun. ACM 49(5):109-113, 2006.
[FBL10] Forward, A., Badreddin, O., Lethbridge, T.C. Perceptions of software modeling: a survey of software practitioners. In: 5th Workshop from Code Centric to Model Centric: Evaluating the Effectiveness of MDD, pp. 12–24, 2010.
[G+11] Genero, M., Fernández-Sáez, A.M., Nelson, H.J., Poels,G., Piattini, M. Research Review: A Systematic Literature Review on the Quality of UML Models. J. Database Manag. 22(3): 46-70, 2011
[GTA14] Tony Gorschek, Ewan Tempero, Lefteris Angelis, On the use of software design models in software development practice: An empirical investigation, Journal of Systems and Software, 95:176-193, 2014
[LCM06] CFJ Lange, MRV Chaudron, J. Muskens. In practice: UML software architecture and design description IEEE Software 23(2):40-46, March-April 2006
[Pet13] Marian Petre. UML in practice. In Proc. of ICSE 2013, pp. 722–731, IEEE, 2013.
[Pet14] Marian Petre. "No shit" or "Oh, shit!": responses to observations on the use of UML in professional practice. Software and Systems Modelling 13:1225–1235, 2014

c) On MDA, MDD, MDE and DSLs

[B+04] Booch, G., et al. An MDA Manifesto, in Frankel, D. and Parodi, J. (eds.) The MDA Journal: Model Driven Architecture Straight from the Masters, pp. 133-143, Meghan-Kiffer Press, 2004.
[BCW12] Brambilla, M., Cabot, J., Wimmer, M. Model-Driven Software Engineering in Practice. Morgan & Claypool, 2012.
[BVGR08] Bezivin, J., Vallecillo, A., García-Molina, J., Rossi, G. Model Driven Software Development . Special issue Novática/UPGRADE Vol. IX, No.2, 2008.
[Cab14] Cabot, J. Clarifying concepts: MBE vs MDE vs MDD vs MDA. http://modeling-languages.com/clarifying-concepts-mbe-vs-mde-vs-mdd-vs-mda/ Last accessed: Dec 2014.
[CK08] Cook, S., Kent, S. The Domain Specific IDE. Novática/Upgrade IX(2):17-21, 2088
[FP10] Fowler, M., Parsons, R. Domain-Specific Languages. Addison-Wesley, 2010
[FR07] France, R and Rumpe, B. Model driven development of complex software: A Research Roadmap. Future of Software Engineering, IEEE, 2007.
[GMR04] García-Molina, J., Moreira, A., Rossi, G. UML and Model Engineering. Special issue Novatica/UPGRADE Vol. 5, No.2, 2004.
[Kle08] Kleppe, A. Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Pearson, 2008.
[KLRT13] Kelly, S., Lyytinen, K., Rossi, M., Tolvanen, J.P: MetaEdit+ at the Age of 20. In: Seminal Contributions to Information Systems Engineering, pp. 131-137. Springer, 2013.

[KW03] Kleppe, A. G., Warmer, J. MDA Explained: The Model Driven Architecture: Practice and Promise, Addison-Wesley Longman Publishing Co., 2003.
[Nav13] Naval Air Warfare Centre, DoD. Simulation-based Acquisition (SBA). http://www.navair.navy.mil/nawctsd/Resources/Library/Acqguide/sba.htm, 2013. Last accessed, Dec 2014.
[Sch06] Schmidt, D.C. Model-Driven Engineering. IEEE Computer 39 (2):25-31, 2006.
[Sel03] Selic, B. The Pragmatics of Model-Driven Development. IEEE Softw. 20(5):19-25, 2003.
[Sel08] Selic, B. MDA Manifestations. Upgrade. IX(2):12-16, April 2008.
[Tho04] Thomas, D. MDA: Revenge of the modelers or UML utopia? IEEE Software 21(3):22–24, 2004.
[Tol11] Tolvanen, J.P. Creating Domain-Specific Modelling Languages That Work: Hands-On. In Proc. of ECMFA 2011, LNCS 6698, pp. 393-394, Springer, 2011.
[Vol11] Voelter, M. MD*/DSL Best Practices (Update March 2011) http://voelter.de/data/pub/DSLBestPractices-2011Update.pdf Last accessed, Dec 2014.
[Vol13] Voelter, M. DSL Engineering - Designing, Implementing and Using Domain-Specific Languages. dslbook.org, 2013.
[Wat08] Watson, A. A Brief History of MDA. Upgrade IX(2):7-11, April 2008.
[WK03] Warmer, J., Kleppe, A. The Object Constraint Language: Getting Your Models Ready for MDA. Addison-Wesley Longman, 2003.

d) General references

[BF14] Bourque, P., Fairley, R.E. (eds.) Guide to the Software Engineering Body of Knowledge, Version 3.0, IEEE Computer Society, 2014; www.swebok.org.
[Byt81] Byte Magazine. Special Issue on SmallTalk. August 1981.
[Dav89] Davis, F.D. Perceived usefulness, perceived ease of use, and user acceptance of information technology, MIS Quarterly, 13(3): 319–340, 1989.
[Gar05] Gartner, Inc. Gartner Hype Cycle of Emerging Technologies. https://www.gartner.com/doc/484424/gartners-hype-cycle-special-report, 2005. Last accessed: Dec 2014.
[Gar13] Gartner, Inc. Gartner Hype Cycle of Application Architecture, 2013. https://www.gartner.com/doc/2569522/hype-cycle-application-architecture-, 2013. Last accessed: Dec 2014.
[Gar14] Gartner, Inc. Gartner Methodologies: The Gartner Hype Cycle. http://www.gartner.com/technology/research/methodologies/hype-cycle.jsp. Last accessed: Dec 2014.
[IEEE90] IEEE Computer Society. IEEE Standard Glossary of Software Engineering Terminology. IEEE Std 610.12-1990, 1990.
[Kra07] Kramer, J. Is Abstraction the Key to Computing? Comms. of the ACM, 50:37-42, 2007.
[Moo14] Moore, G.A. Crossing the Chasm: Marketing and Selling High-Tech Products to Mainstream Customers. Harper Business Essentials, 1991 (revised 1999 and 2014).
[SZ13] Saitta, L., Zucker, J.D. Abstraction in Artificial Intelligence and Complex Systems. Springer, 2013.
[W14] Wikipedia. Comparison of Business Process Modeling Notation tools. http://en.wikipedia.org/wiki/Comparison_of_Business_Process_Modeling_Notation_tools Last accessed: Dec 2014.