Transitioning to Model Driven Software Development

Preparing for the Paradigm Shift

Author: Jorn Bettin

Version 1.0

April 2006

Copyright © 2006 SoftMetaWare Ltd.

SoftMetaWare is a trademark of SoftMetaWare Ltd.

All other trademarks are the property of their respective owners.

SoftMetaWare

Revision History

Version	Author	Description
1.0	Jorn Bettin	April 2006

REVISION HISTORY 2

1 THE MODEL DRIVEN & OPEN SOURCE SOFTWARE HYPE CURVE 4

2 DOMAIN SPECIFIC LANGUAGES 5

3 META MODELING - KEEPING IT SIMPLE 7

4 ARCHITECTURE-CENTRIC MDSD 8

5 OPEN SOURCE BASED MDSD 9

6 SOFTWARE FACTORIES 10

7 PLANNING THE TRANSITION TO A MODEL DRIVEN APPROACH 11

8 USING MDSD AS TOOL TO IMPROVE YOUR COMPETITIVE EDGE 13

9 REFERENCES 15

It is time for a major stock take of model driven software development approaches within software intensive industries. The speed of progress in the last few years in terms of interoperability standards for model driven tooling has not been spectacular. For example it took the OMG five years to develop an industry standard for expressing model transformations, and it remains to be seen if and when the OMG's Query, View, and Transformation (QVT) standard will become relevant. The term "Model Driven" has gone through the usual hype cycle, and the dust is in the process of settling. The outcome does not really come as a surprise - the assessment from the organizers at the OOPSLA'02 workshop "Generative Techniques in the Context of Model Driven Architecture" still holds:

> *The era of expensive software development tools is long gone. The market already discards MDA tool-hype, and does not readily buy into MDA. On the other hand, domain-specific approaches and generator technology have a huge potential. The main problem is not one of tools - as some vendors would like people to think - but one of culture. Unless this is recognized, MDA will become the CASE of this decade.*
> *…*
>
> *The only realistic way to gain widespread acceptance for the use of MDA-type approaches and generation tools is by addressing the related cultural issues, and by providing tools at a very low cost - so low that a CEO or CTO does not have to think twice. How about an open source-based project? While the tool vendors jockey for position and battle amongst themselves, an open source project may actually deliver results and become a de-facto MDA standard.*

Now, in 2006, there is still no market leading MDA tool vendor in sight, and in general the uptake of proprietary commercial tools has fallen significantly short of vendor expectations. In the mean time, Microsoft joined the MD* club in 2003 with their Software Factories initiative, and a large range of Open Source MD* projects have emerged and have had time to mature. For the most part decision-makers and software developers in the software industries have been watching from the sidelines. This is the result of lessons learnt from Computer Aided Software Engineering (CASE) tools and from the productivity drop in the 90s that followed the adoption of object oriented languages such as C++ and Java, and the hype about reuse.

This white paper analyzes the current situation from a tool vendor independent perspective, and it provides practical recommendations for the adoption of a Model Driven Software Development paradigm. The recommendations are consistent with SoftMetaWare's Industrialized Software Asset Development (ISAD) methodology, and are the result of guiding software development organizations in various industries through the paradigm shift to model driven approaches.

1 The Model Driven & Open Source Software Hype Curve

The following picture is a subjective - but possibly quite helpful - interpretation of the MD* adoption rate to date. The diagram intentionally superimposes MD* milestones and OSS milestones, so that it becomes clear how the MD* trend is able to surf on the wave of mainstream acceptance of OSS tools that currently rolls through software development organizations.

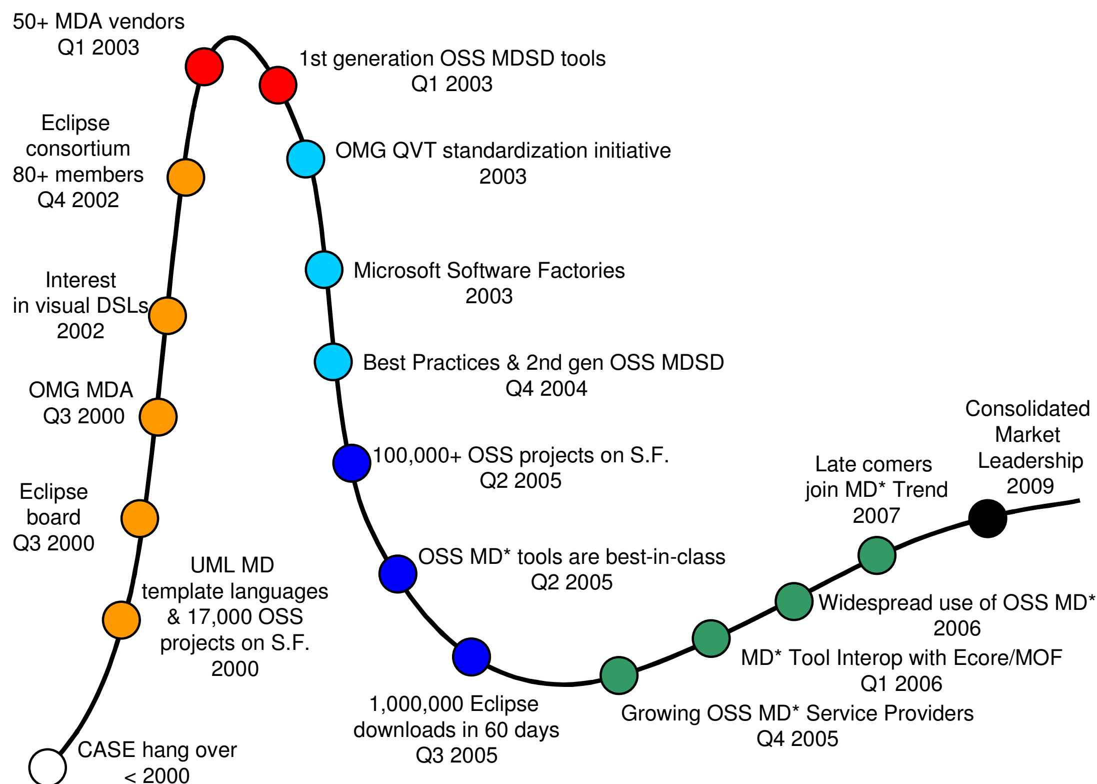

Figure 1 *MD* & OSS Hype Curve*

Without a reasonable understanding of how OSS shapes the development of software infrastructure [Bettin 2005], including MD* software development tools, it is easy to miss the significance of the illustrated OSS milestones.

From the OMG's perspective, the CASE hang over seemed to be over by 2000, and hence the Model Driven Architecture [OMG MDA] initiative was launched. At that point the market already included several vendors of UML model driven template languages, but customers were few and far between. At the same time the Open Source phenomenon was steaming ahead at grass roots level, but was only poorly understood by most software industry executives and within end user organizations.

Undoubtedly, one of the best decisions in the last five years by IBM was the decision to Open Source the Eclipse Java IDE [Eclipse] platform. Firstly this forced IBM staff at all levels within the organization to get familiar with the Open Source phenomenon. Secondly, it provided a very strong signal to the market, and paved the way for

incremental and gradual acceptance of OSS in conservative sectors such as banking and government.

A key element that simply does not fit on the timeline in figure 1 is the long track record of Software Product Line Engineering [SEI SPLE] approaches, which can be traced back to the mid 70s. Successful OSS MD* components such as OpenArchitectureWare [Eclipse GMT-OAW] mainly build on Software Product Line Engineering principles, and only to a lesser degree on emerging OMG MDA standards.

The one MDA standard that is really worth taking note of is the Meta Object Facility [OMG MOF], as this standard can be used as a the foundation for defining domain specific languages (DSLs). See section 2 for a detailed discussion of the relevance of DSLs. Since it is a result of the OMG consortium, MOF suffers from design by committee, and it contains a number of elements that are irrelevant in practice. The good news is that the Eclipse Modeling Facility OSS project includes Ecore, a pragmatic, simplified implementation along the lines of MOF. The practical applicability of Ecore is outlined in section 5.

It took the OMG until 2003 to launch an initiative to develop a standard for model transformations (Query, Views, Transformations or QVT). Without such a standard today's MDA vendors make use of various proprietary languages to express model transformations. Hence it still is somewhat premature to talk about MDA as an established "standard". The initial version of the QVT standard [OMG QVT] is in the process of being published in 2006.

Around the time when the OMG started to think about QVT, Microsoft joined the model driven bandwagon - or rather officially opted out of the QVT standard - by announcing a "Software Factories" strategy. The goal of Software Factories is the development of tools that enable software developers to define and subsequently use DSLs. See section 6 for a discussion of Software Factories and Microsoft's motivation.

2 Domain Specific Languages

The majority of software systems today are implemented in general-purpose languages that address a broad domain. Classical programming languages for business software development such as COBOL or RPG survive alongside modern object oriented languages such as Java or C#. Additionally, modeling languages such as UML provide a formal visual syntax to describe the structural aspect and part of the behavioral aspect of object oriented software systems.

In contrast, all advanced applications of model driven approaches involve domain specific languages (DSLs), i.e. formal specification languages that incorporate domain concepts as first class language elements. Familiar DSLs are end user programming tools such as spreadsheets and symbolic mathematics systems.

The limitations of the predominant one-fits-all philosophy became obvious during the attempts to replace languages such as COBOL with a new generation of general purpose tools for business software development that were marketed in the 80s and 90s under the heading of Computer Aided Software Engineering (CASE) tools. Although some of the better CASE tools definitely raised the level of abstraction - and thereby productivity - their output was always tied to a specific technology stack, which ran counter the trend towards more and more distributed and heterogeneous

systems. The tools simply could not keep up with the need to integrate with a growing number of implementation technologies.

The alphabet soup of CASE vendors and implementation technologies contributed to the rise of languages such as Java, and to standardization initiatives such as CORBA, UML, and later MDA. The main hope was to reduce infrastructure development costs and to enable interoperability through standards based approach to software development. In order to facilitate unproblematic adoption across hardware and operating system platforms, Java started off as a naked object oriented language without an extensive set of well-designed high-level APIs. This approach implied a big step backwards in terms of productivity, but the enterprise market bought into the standards based approach, hoping to shake off the technology stack lock-in imposed by traditional CASE tools. Ongoing development of the J2EE standard is a good example of the painfully slow and contorted process of reaching consensus on useful high level APIs.

To solve the software productivity problem the OMG tries to raise the level of abstraction of its standards, via i.e. the adoption of Model Driven Architecture standards such as MOF and QVT. Although there is a grain of truth in this line of thought, it still suffers from the limitations of a one-fits-all approach. In particular MDA products of the Executable UML flavor are limited in their applicability to problems that can be expressed as state-event machines. And in more general terms the MDA model transformation pattern from Computation Independent Model (CIM) to Platform Independent Model (PIM) to Platform Specific Model (PSM) is limited by the tendency of general-purpose notations for model transformations to be highly verbose. Even simple practical application scenarios lead to transformations that stretch over a whole series of formal transformation diagrams [Bettin 2003].

These limitations of general-purpose languages and of standards based software development raise questions about alternative paths to increase software quality and software development productivity. This is where domain specific (special purpose) languages come in. Whenever deep domain knowledge is available, it is usually possible to design formal specification languages that directly tap into domain specific terminology. Practical examples of DSLs include languages to specify electrical networks, languages to specify user interfaces for mobile phones, and languages to express pricing methodologies for specific types of products. A good DSL minimizes the notational gap to the problem domain. Hence a DSL is close to the natural language of a domain expert, and specifications expressed in a DSL tend to be very compact. Therefore DSLs can be a very effective antidote against the problems of the one-fits-all approaches described above.

The development of a DSL can be seen as the final step in building a domain specific framework [Bettin 2004b]. A useful analogy:

> A framework is like the engine in a car – it is essential, but it only adds real value when built into a usable product.
>
> A DSL provides the highly automated production facility that assembles engines and other parts into fully functional cars.

DSLs are at the heart of advanced Model Driven Software Development, and they are also the main pillar underpinning Microsoft's Software Factories approach. MD* approaches complement software product line engineering methodologies such as FAST [WL 1999] and KobrA [ABKL 2002]. The relationship between MDSD and software product line engineering can be compared to the relationship between

Component Based Development and Object Technology: one is building on the other. Hence the terminology of SoftMetaWare's ISAD methodology for MDSD is an extension of the terminology for software product line engineering. ISAD covers techniques for building domain-specific application engineering processes and tools, for addressing organizational aspects, and for managing fast-paced iterations in a distributed team environment.

The difficult aspect of development with DSLs is making appropriate DSLs available in the first place. The following factors all play a role:

- **Sufficiently deep domain knowledge.**
  Often a real problem, although not one that is easily acknowledged by a software development organization that supposedly specializes in developing specific types of applications.
- **Practical experience in designing and implementing formal languages.**
  A skill issue that can be overcome by hiring appropriate experts that can also provide knowledge transfer to internal resources. Language design is a skill that can be learned by talented software professionals.
- **Good tools that enable fast implementation of DSLs, i.e. the creation of high quality editors and the creation of high quality compilers and/or interpreters.**
  Availability of sufficiently good tools used to be an issue back in the 90s. Today there are several tools that are up to the job, but those tools are not part of the tool sets offered by vendors that target the mainstream enterprise market.
- **Practical experience in integrating the use of one or more DSLs into an effective iterative and agile software development process that may also include handcrafted code written in general purpose languages.**
  Introducing a DSL into a traditional software development organization is a non-trivial one-off transition activity. The necessary mind shift within the organization, and related streamlining and automation of the end-to-end production process usually requires expert assistance. This is particularly the case if the transition needs to be achieved within a matter of months rather than a matter of years.
- **Unjustified fears about not using a standards-based approach and mainstream programming languages.**
  Introduction of DSLs requires full management commitment within the organization. It needs to be understood in which areas standards and mainstream programming languages have their place, and in which areas DSLs represent a superior alternative. The best approach is to use domain analysis, and the initial design of appropriate DSLs, to clarify the question of scope before a rollout is considered. The success of using a DSL can be measured by the compactness of the resulting specifications: If it takes an inordinate amount of time to maintain specifications expressed in the DSL, it will not be possible to convince an application development team to use the new approach.

3 Meta Modeling - Keeping it Simple

Meta modeling, effectively just another name for language design, is an essential activity in any model driven approach. Meta models are used to specify the abstract syntax of a DSL, and they are also used in the context of MDA to describe the structural elements of Platform Independent Models, Platform Specific Models, and Model Transformations. Hence it does not surprise that MDA builds on the OMG's

Meta Object Facility (MOF) standard, which provides a standardized meta meta model - in other words a standard notation for expressing meta models.

Unfortunately the Unified Modeling Language (UML) has heavily influenced the MOF standard, and it contains much more than is actually needed in practice. MOF terminology draws heavily on object oriented terminology, which goes as far as including concepts such as operations and parameters in MOF. Depending on the types of meta models one needs to describe, this can either be advantageous, or it can be completely irrelevant and distracting.

When defining visual [domain specific] modeling languages, the key concepts that one always deals with are *boxes*, *lines*, *roles*, *properties*, and some structural concept that enables grouping or classification of model elements - let us call it *container*. Designing a DSL is nothing more and nothing less than identifying the names of the various types of *boxes*, *lines*, *roles*, *properties*, and *containers* that are intended to be allowable languages constructs, and subsequently precisely defining the semantics of these meta model elements. The best way to validate whether such a simple meta meta model is sufficient is to apply it to express itself. It is also possible to express the MOF meta model or for that matter the UML meta model using the "boxes & lines" meta meta model. In fact this is precisely the approach taken by tool vendors such as MetaCase, who were amongst the first to stress the need to keep meta modeling simple. The MetaCase MetaEdit+ tool uses a meta meta model consisting of *objects*, *relationships*, *roles*, *properties*, and *graphs* (GOPRR) [Tolvanen 2000]. In the remainder of this paper the term *boxes & lines language description language* is used to refer to meta meta models that are no more complex than absolutely necessary. This helps to keep the discussion grounded in reality - i.e. language design on a white board, and it should also appeal to those in the agile software development corner who are skeptical regarding the practical usefulness of DSLs.

4 Architecture-Centric MDSD

The term *domain* is typically used in conjunction with expert knowledge in specific industries, such as banking, manufacturing, etc. In MDSD the term domains is also used to refer to sub-domains of the software engineering domain.

In all those cases where there is a lack of deep business domain expertise, and also in those scenarios where 90% of the domain amounts to *CRUD*[1] *functionality for maintaining business entities realized with an implementation technology stack of* T_1, T_2, … T_n, there is no point in trying to develop a DSL motivated by the business domain. That does not mean that in such a scenario there is no place for Model Driven Software Development. It simply means that the main DSL could be a tailored type of "UML class diagram" that takes into account local rules of the game – that is local constraints and limitations within the enterprise architecture of an organization. Further DSLs may come into play to describe workflow, visual characteristics and behavior of business entities on the user interface [Bettin 2003b], mappings to structures from existing legacy systems, and so on. In this context DSLs provide an implementation technology free front-end to capture the majority of application requirements (say 90%) in a formal notation that can automatically be converted into executable application code. The remaining 10% of requirements that amount to *non-templatizable* business logic are often best implemented in a general purpose

[1] Create, Read, Update, Delete

programming language such as Java of C#, and MDSD tools can be used to integrate this handcrafted code with the rest of the application.

Typically one starts with creating a handcrafted reference implementation for CRUD functionality in the given technology stack, see [Bettin 2004] and [SV 2006] for a detailed description of relevant patterns. The reference implementation can be very thin, but it needs to cover all the layers of the implementation technology stack. It is used as a basis for the extraction of code templates[2], and it is also used for incremental refinement of the architecture. The name *architecture-centric MDSD* comes from the central role played by the reference implementation and its architecture.

In practice one of the biggest weaknesses in enterprise applications is a lack of modularization [LLBR 2005]. Architecture-centric MDSD is ideally suited to provide modularization constructs that can already be applied during the elaboration of requirements and that can automatically be enforced in the form of a strictly component based design within the implementation code. The prospect of being able to actively manage and control dependencies often is the key motivating factor to employ architecture-centric MDSD.

It is interesting to note that a DSL that equips class diagrams with direct support for component based development is quite easy to specify using MOF. Similarly all DSLs that primarily consist of variations of object oriented concepts are easily expressed in MOF. Of course such DSLs can also be defined using the *boxes & lines language description language*, but for software professionals who have been raised on objects and the UML, and who are not yet experienced in meta modeling, using MOF or a similar notation may feel more familiar.

5 Open Source Based MDSD

If one is looking for a practical implementation of MOF, one need not look further than the Open Source Eclipse platform. One of the tools used to build Eclipse is the Eclipse Modeling Framework [Eclipse EMF], which contains Ecore, a pragmatic interpretation of the MOF standard. Ecore is increasingly being used by various MDSD components from the Eclipse Generative Model Transformer [Eclipse GMT] project and the Eclipse MDDi project. Especially the Eclipse GMT project is heavily making use of Ecore:

- OpenArchitectureWare, a model driven template language now allows the use of Ecore as the meta meta model
- The Atlas Transformation Language relies on Ecore
- The MOFScript component for model to text transformations is based on Ecore

The above are just a few examples of Open Source components for MDSD. There are many more, and the list is growing continuously. The interesting observation to make here is that some of these components are considered "best-in-class" by a number of practitioners who have had the opportunity to apply both Open Source MD* tools and a range of proprietary MDA tools in industrial practice. This seems to confirm the assessment in [Bettin 2003b] and is also consistent with the success that the Open Source concept enjoys in the domain of "infrastructure software" in general.

---

[2] Extracting code templates from a reference implementation only increases the costs of developing a proof-of-concept prototype by about 15%-25%.

Within the sphere of influence of the Eclipse platform Ecore[3] is developing into a generally accepted common denominator for interoperability between MD* tools. This is a very positive development, and it demonstrates that the Ecore cousin of MOF is hitting the mark. EMF even provides the tools needed to bootstrap a *boxes & lines language description language*, which can be beneficial when building DSLs that have nothing to do with classes and other object oriented concepts.

6 Software Factories

As to be expected the Microsoft strategy does not fall in line with OMG standards and terminology, and Microsoft is busy attempting to establish credibility with its Software Factories [GS 2004] approach, which is a DSL approach for Architecture-Centric MDSD [MS SF]. The Microsoft strategy is designed to appeal to software professionals who are used to developing in general purpose programming languages such as Java and C#. The objective is to provide tools that enable this audience to experiment with DSLs and with code generation, and thereby to assist organizations in the paradigm shift towards model driven approaches.

In many ways the Software Factories concept is consistent with the recommendations for transitioning to Model Driven Software Development contained in this paper. But there are a number of points that are worthwhile to note:

- Microsoft DSL tools are strongly coupled to the Microsoft Visual Studio IDE. This is marketed as an advantage - strong integration between various tools. In reality it is probably more accurate to speak of classical Microsoft product bundling. No doubt that tool integration is a useful feature, but using a proprietary platform as the basis for infrastructure integration is increasingly anachronistic. Comparing the growth rates of Open Source infrastructure software and of Microsoft infrastructure software indicates where the market is heading.
- Software is increasingly developed by highly distributed teams that use the web as a collaboration platform. A good tool chain needs to minimize all non-essential coupling. In MDSD models and model elements are source code artifacts, and need to be treated as such by IDEs and tools such as source code control systems. Distributed teams need the ability to check out, work with, compare, merge, and check in model elements from various code repositories. The core integration points are source code artifacts (meta meta models, meta models, model transformation, models, and model elements) and open standards for exchanging these artifacts – and not tools. It is unrealistic to expect that in the future all participants in a global software supply chain will be working with exactly the same proprietary tool set.
- As a tool vendor Microsoft benefits from presenting a picture where tools play a dominant role. However, the transition to MDSD is akin to the paradigm shift from assembler coding to development using higher level languages. The real issues relate to training and knowledge transfer in DSL design, and to the necessary changes in traditional software development processes.
- The focus of Software Factories on architecture-centric MDSD distracts from the fact that some of the most successful and most widely used DSLs come in the form of end user programming tools such as Microsoft Excel or Wolfram

[3] http://download.eclipse.org/tools/emf/javadoc/?org/eclipse/emf/ecore/package-summary.html#details

Research Mathematica. Hence it is incorrect to assume that all DSLs are best packaged as part of an IDE, and to assume that the target audience for DSLs is limited to software professionals. Often building a good model driven application (i.e. an application that effectively is a powerful domain-specific modeling tool for end users) deserves as much or even more attention than the DSLs used in building the application.

- Compared to other players, including many of the MDA vendors, Microsoft is a new player in the MDSD space, and is still in catch-up mode. It is unrealistic to expect best in class DSL building tools from Microsoft in the short term. It is more realistic to expect DSLs and DSL building tools for Architecture-Centric MDSD that are closely coupled to Microsoft's implementation technology stacks.

7 Planning the Transition to a Model Driven Approach

Since most software development organizations are still within the early phases of the journey towards MDSD, it is worthwhile to map out a path that ensures that both the organization and its people safely reach the destination.

Observation: The suggestion of building and using domain-specific tools and processes lead some people to believe that MDSD is incompatible with Agile Software Development. This is one reason why some organizations continue to sit on the fence; in particular since the emerging MDA standards concentrate on interoperability and don't directly address agility of the software development process. The whole situation is not helped by some MDA vendors who use distorted numbers to promote their product. It is more instructive to compare concrete metrics from practical experiments and real projects [Bettin 2003b] than to engage in the emotional debate between the MDA camp and its detractors[4].

Recommendation: If you are still sitting on the fence, set up a small pilot project to explore MDSD and judge for yourself. There is no substitute for first-hand experience.

Observation: It is important to have realistic expectations regarding MDA standards. In particular the QVT standard is still in the process of being published, and the concerns about verbosity of visual notations for model transformation mentioned above apply. Today's proprietary and Open Source MDA/MDSD tools make use of non-industry-standard template languages to express model to text transformations. However, extracting code templates is a small effort compared to hand coding a reference implementation (which needs to be done anyway) from which templates can be extracted.

Recommendation: It is not worthwhile losing sleep about a non-standardized template language. The ergonomics of the template language do count, but an accepted standard is still some time away. In any case, vendor lock-in can be avoided by using one of the Open Source options.

[4] Agile Model Driven Development (www.agilemodeling.com/essays/amdd.htm) is one example of a marketing ploy aimed at catching the large number of organizations that have not yet done serious MDSD or MDA pilot projects.

Observation: Like with any major paradigm shift, it is a good idea to first apply a new approach such as MDSD in a pilot project that involves a team that is motivated to make the transition. The learning experience from a pilot project should be in place before rolling out MDSD to an entire organization.

Recommendation: Learn to walk before you run. Remember that a large part of MDSD is about knowledge management, and building domain-specific assets. If you are new to meta modeling, start your MDSD pilot project with Architecture-Centric MDSD. If your business is firmly rooted in the Microsoft technology stack, perhaps start with the Software Factories tools. Further business domain-specific languages can be added later if required.

Observation: Designing useful DSLs and making the most of deep domain knowledge depends on being able to pick up and articulate subtleties that may initially only be available in the form of tacit knowledge. If fundamentals such as a disciplined agile requirements management process are not in place, then the subtleties that hold the key to making significant progress may drown in the noise of daily emergencies.

Recommendation: Ensure that your team is capable of running a well oiled iterative and agile software development process before embarking on MDSD. Otherwise your MDSD project may fail due to reasons that have nothing to do with MDSD in specific.

Observation: Software development organizations and software professionals can be surprisingly conservative when it comes to adopting new processes. A pilot project is essential, and tangible results count and create confidence. Additionally the introduction of a new process requires visible and unwavering management support.

Recommendation: If you have been around long enough, try to remember the organizational learning curve and the time required to move away from assembly language programming to modern higher level languages. Don't assume that the transition to MDSD will be any easier, and prepare for the same types of detractors.

Observation: No software development process can replace essential knowledge about a business. The availability of domain experts with a deep understanding of your business is essential. The potential of MDSD is not delivered in a box that just needs to be unpacked.

Recommendation: Don't neglect the importance of a high quality reference implementation when embarking on MDSD. The reference implementation may entail the development of domain-specific frameworks. Framework development can be a very substantial part of MDSD. Plan for appropriate framework development, but take an iterative, incremental approach at all times [Bettin 2004]. Also keep the reference implementation up-to date at all times for the purpose of being able to test new increments of your MDSD tool kit.

8 Using MDSD as Tool to Improve Your Competitive Edge

Domain-specific assets usually need to be built, and only in some cases can they be bought off-the-shelf or are available as a public asset[5]. If software maintenance and enhancements are performed purely in the context of working towards short-term, i.e. quarterly results, then there is little incentive to build domain-specific assets, and the quality of the original software design degrades significantly over time. Unfortunately, treating software as a capital cost[6] encourages this trend [Henderson-Sellers].

The traditional accounting perspective is incompatible with the idea of incrementally building and nurturing a domain-specific application platform and reusing software assets across a large number of applications. High-quality software assets (models, components, frameworks, generators, languages, and techniques) need to be viewed in the same way as personnel assets that accrue over time. By leveraging domain-specific knowledge to refactor parts of the existing software into *strategic software assets*, the value of the software actually increases, rather than decreases. Thus a long-term investment strategy is required to maximize return on investment in strategic software assets.

Of course it is not worthwhile to develop all software used in an organization into a strategic asset. The SoftMetaWare *Industrialized Software Asset Development*[7] (ISAD) methodology [Bettin 2004] uses the following classification scheme as a tool for planning investments in software:

- **strategic software assets**—the heart of your business, assets that grow into an active human- and machine-usable knowledge base about your business and processes,
- **non-strategic software assets**—necessary infrastructure that is prone to technology churn and should be depreciated over two to three years, and
- **software liabilities**—legacy that is a cost burden.

The identification of strategic software assets is closely associated with having a clear business strategy, knowing which money-generating business processes are at the core of an organization, and being able to articulate the software requirements relating to these core processes. Strategic software assets are those that define your competitive edge. Off-the-shelf business software, even very expensive enterprise systems should only be considered strategic in terms of information about your business stored in the databases of such products, because the functionality provided is not unique and could be supplied from a number of vendors. In contrast, organizations often use highly unique packaging and pricing methodologies for their products and services, and off-the-shelf software can only cover this variability between organizations to a certain degree. Hence a software system that takes into

[5] This does not come as a surprise if one considers that domain-specific software assets are not commodities, they typically contain core intellectual property of an organization. In contrast, by definition, any software that can be bought off-the-shelf can't provide an organization with a sustainable competitive edge.

[6] I.e. treating software as an asset where maintenance only delays obsolescence of the asset value

[7] The term Industrialized Software Asset Development relates to the capability of the model driven approach of making tacit domain knowledge explicit, and of capturing strategic knowledge in a human and machine readable format

account the subtleties that differentiate you from the competition constitutes an example of a strategic asset that is worthwhile to refine and evolve. Strategic software assets should not be allowed to degenerate into a liability over time, i.e. a code base that is no longer fully understood by the software development team.

Recommendations: Strive to evolve your core business systems into strategic assets that increase in value over time through (re)use and refinement. Widely used and respectable Open Source software should also be treated as a strategic asset—in this case a public asset. Commercial 3rd party software largely falls into the category of non-strategic assets: usually the software is not built entirely on open standards, and you also cannot assume that the product will be supported indefinitely. Hence investments in commercial software should be treated as capital investments affected by depreciation.

- Be honest and don't portray all your legacy systems as strategic assets in the sense described above, identify the liabilities and strive to replace them by a combination of strategic assets and non-strategic assets. A healthy mix of strategic and non-strategic assets may sometimes only include a small core of strategic software assets.
- You need to differentiate between the 3rd party products and your organizations' information that builds up in the databases of such products. A subset of your information is a strategic software asset, and needs to be treated as such. The same can be said about legacy software that has become a liability: the applications have become a liability but some of the information managed by these applications constitutes a strategic asset.

9 References

[ABKL 2002] Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Laqua, R., Muthig, D., Paech, B., Wuest, J., Zettel, J., Component-based Product Line Engineering with UML, Addison-Wesley, 2002

[Bettin 2003] Jorn Bettin, OOPSLA'03 Workshop Generative Techniques in the Context of MDA, Ideas for a Concrete Visual Syntax for Model-to-Model Transformations, www.softmetaware.com/oopsla2003/bettin.pdf www.softmetaware.com/oopsla2003/bettin.ppt

[Bettin 2003b] Jorn Bettin, Model-Driven Architecture - Implementation & Metrics, www.softmetaware.com/mda-implementationandmetrics.pdf

[Bettin 2004] Jorn Bettin, Model-Driven Software Development: An emerging paradigm for Industrialized Software Asset Development, www.softmetaware.com/mdsd-and-isad.pdf

[Bettin 2004b] Jorn Bettin, Process Implications of Model Driven Software Development, www.softmetaware.com/process-implications-of-mdsd.pdf

[Bettin 2005] Jorn Bettin, Building Durable Enterprise Architectures: Extending Build versus Buying Decision Frameworks with Open Source Options, www.softmetaware.com/ea-and-oss.2.0.pdf

[Eclipse] The Eclipse IDE Platform, www.eclipse.org

[Eclipse EMF] Eclipse Modeling Framework project, www.eclipse.org/emf

[Eclipse GMT] Eclipse Generative Model Transformer project, www.eclipse.org/gmt

[Eclipse GMT-OAW] Eclipse Generative Model Transformer project, OpenArchitectureWare component, www.eclipse.org/gmt/oaw

[GS 2004] Jack Greenfield & Keith Short, Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools, Wiley, 2004

[Henderson-Sellers] Brian Henderson-Sellers, Object-Oriented Metrics, Measures of Complexity, Prentice Hall, 1996

[LLBR 2005] Editors: L. Liu & B. Roussev, Management of the Object-Oriented Software Development Process, chapter Managing Complexity with MDSD (Jorn Bettin), Idea Group Publishers, 2005

[MS SF] Microsoft, Visual Studio 2005 Team System Modeling Strategy and FAQ,

	http://msdn.microsoft.com/vstudio/teamsystem/workshop/sf/default.aspx?pull=/library/en-us/dnvs05/html/vstsmodel.asp
[OMG MDA]	Model Driven Architecture, www.omg.org/mda
[OMG MOF]	Object Management Group, Meta Object Facility standard www.omg.org/cgi-bin/doc?formal/2006-01-01
[OMG QVT]	Object Management Group, Query, Views, Transformation standard www.omg.org/cgi-bin/doc?ptc/2005-11-01
[OOPSLA02]	Workshop Generative Techniques in the Context of MDA, www.softmetaware.com/oopsla2002/mda-workshop.html
[SEI SPLE]	Carnegie Mellon Software Engineering Institute, Product Line Practice, www.sei.cmu.edu/plp/
[SV 2006]	Tom Stahl & Markus Voelter, Model-Driven Software Development: Technology, Engineering, Management, Wiley, 2006, to be published
[Tolvanen 2000]	Juha-Pekka Tolvanen, GOPRR Metamodeling Language www.cs.jyu.fi/~jpt/ME2000/Me07/sld001.htm
[WL 1999]	D. M. Weiss, C.T.R. Lai, Software Product Line Engineering, A Family-Based Software Development Process, Addison-Wesley, 1999