Terry Bailey
Regis Vogel
Jason Mansell (Eds.)

From code centric to model centric software engineering: Practices, Implications and ROI

4th European Workshop on "From code centric to model centric software engineering: Practices, Implications and ROI" (C2M)

University of Twente, Enschede, The Netherlands. June 24th 2009

Enschede, the Netherlands, 2009
CTIT Workshop Proceedings Series WP09-07
ISSN 0929-0672

Preface

This volume contains the proceedings of the 4th From code centric to model centric software engineering: Practices, Implications and ROI (C2M) held on June 24 in University of Twente, Enschede, The Netherlands in conjunction with the European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA 2009).

Model Driven Development (MDD) technology needs the change of paradigm, from code centric to model centric engineering, which implies an evolution in the processes, tools and methods used by IT providers and IT divisions. This evolution is strategic and needs to be defined and planned carefully to achieve a successful adoption. Also, MDD brings to industry new opportunities such as: having faster requirement to prototype cycles; improving cost efficiency of requirements review or new releases; significantly reducing effort to deploy to new platforms; speed up development using domain specific extensions; using models to describe all relevant aspects (functional and non functional requirements, behavior, etc.) so that these models can be used to generate artifacts (software code, test scripts, documentation, etc.);...

Industry plays a critical role in the development of MDD by providing the scientific and vendor communities with a both real life requirements and real life testing of their solutions. This workshop's aim is to bring together people from academia and industry who can provide their experiences and highlight new challenges in their current practice and how they envision adopting MDD.

In this edition of the workshop we have accepted seven papers representing a wide range of approaches and techniques explaining how these respective companies have adopted, or are in the process of MDD. Once again the focus is on explaining how companies face up to the change of paradigm through explaining both their successes and lessons learned.

We would like to take this opportunity to thank the contributors, all authors and reviewers for their valuable help. We wish all the participants a successful continuation of their work in this area. Finally, we would like to thank the organization of the ECMDA-FA 2009 conference in which this workshop has been embedded.

June 2009

Terry Bailey
Regis Vogel
Jason Mansell

Organisation

Workshop Chairs

Terry Bailey	VICINAY Cadenas S.A.
Regis Vogel	The Information Highway Group
Jason Mansell	Fundación European Software Institute

Programme Committee

Terry Bailey, VICINAY Cadenas S.A

Jason Mansell, Fundación European Software Institute - TECNALIA

Regis Vogel, IHG

Alan Hartman, IBM India Research Lab.

Jon Oldevik, SINTEF

Tom Ritter, Fraunhofer FOKUS

Supporting Organisations

VICINAY Cadenas S.A.
Fundación European Software Institute
The Information Highway Group
MODELPLEX project https://www.modelplex.org
Centre of Telematics and Information Technology, University of Twente

Table of Contents

Assessing usability of model driven development in industrial projects

Florian Fieber[1,3], Nikolaus Regnat[2], and Bernhard Rumpe[1]

[1] RWTH Aachen University, Department of Computer Science 3,
Ahornstraße 55, 52074 Aachen, Germany
rumpe@se-rwth.de

[2] Siemens AG, Corporate Technology CT SE 1,
Otto-Hahn-Ring 6, 81739 München, Germany
nikolaus.regnat@siemens.com

[3] qme Software GmbH
Gustav-Meyer-Allee 25, 13355 Berlin, Germany
florian.fieber@qme-software.de

Abstract. An integral use of the model driven development paradigm influences and changes an organization's software development division rather heavily. Such a paradigm reduces some tasks in complexity and costs, but also introduces new tasks and, if introduced seriously, has severe affects on activities and roles in the software development process. As the model becomes the most important development artifact, there are new challenges to the development team, e. g. assessing the model's quality, model partitioning and configuration management for distributed teams, setup of build management, tool chaining and tracing of information through the various artifacts. Organizations coping with model driven development need to successfully introduce new tools and new ways of thinking, they are challenged in adopting their processes and training their staff. This paper presents an ongoing research project on the assessment of the usability of modeling and model driven development at a global industrial organization with its headquarters in Germany. The matter of interest is the analysis of the usability of modeling (especially with the UML) and model driven development by accomplishing an empirical, quantitative survey.

Keywords: Modeling, Model Driven Development, UML, Software Engineering, Empirical Study

1 Introduction

The Unified Modeling Language (UML) is a loosely coupled set of object oriented modeling notations, standardized by the OMG [20,21]. It has become a de-facto standard and been successfully adopted by industry for the specification and documentation of object oriented systems [8,9]. The UML is a modeling notation but

does not provide a method on how to model respectively adopt modeling in the software development process. However, most software process models consider modeling activities (e. g. design of the software architecture), roles (e. g. architect/designer) and outcomes (e. g. software architecture/design), but in a variety of ways. Albeit there is an intense use of software models and large parts of the system are modeled, the process is still code centric as the code has almost always to be developed manually.

With the advent of the model driven development paradigm (e. g. OMG's Model-Driven Architecture approach [18,19]) the model becomes the central development artifact and the process becomes model driven as code and documentation are (ideally) being generated from the model. While in code centric processes the model is a sole means for documentation and specification, in model driven development approaches the model becomes a "real" development artifact and modeling activities have to be considered in the development process, e. g. model reviews in quality assurance activities. However, the MDA approach does itself not provide very much methodology for the adoption of the model driven development paradigm in the software development process. Companies applying the model driven development paradigm are challenged to adopt their existing processes in respect to these new and changed activities.

Those challenges are various, starting with extended technological issues, such as a necessity for homogeneous, much more elaborated development environments, a wider variety of development artifacts that want to be managed, stored, versioned, generated or have other kinds of dependencies. This leads to a larger set of "roles" in the development process and it has to be clarified, whether and how these roles are assigned to people (including individual or common ownership of artifacts). Siemens is a company with one of the largest software development companies and we can expect to learn a variety of solutions as well as current challenges that occur when adopting the model driven development paradigm.

The remainder of the paper is organized as follows. Section 2 presents the status quo of modeling and model driven development at Siemens. Section 3 presents our research project, setup of interviews and first findings. Section 4 summarizes and concludes the paper.

2 Software Development at Siemens

Siemens is one of the world's largest software companies but typically is not recognized as one, as most of the software is part of embedded systems. Nevertheless, Siemens has more than 20.000 software engineers worldwide and around 60% of the Siemens business is based on software.

2.1 Organizational Structure

In 2007 Siemens changed its internal organization and now has three sectors: Industry, Energy and Healthcare. Each sector has a broad range of products and/or

provides solutions where software is an integral part – from industry automation systems to power distribution and medical products. Some software development departments within these sectors are developing software since decades but increasing complexity and introduction of globally distributed teams create new challenges that have to be solved. Improving the software development processes and searching for solutions to these new challenges has been an important topic over the past decade. In 1995 Siemens established a best practice sharing forum, the Siemens Software Initiative, which addresses strategic software topics and shares technical best practices regarding processes and architecture.

2.2 Model Driven Development at Siemens

During the last five years the model driven development paradigm began to influence the software developing departments within the Siemens sectors. Although languages like the UML are available for more than a decade, it took years until the industry picked them up. Currently the languages and tools are finally mature enough to be introduced in an industry environment but the basic principles on how to handle models or introduce model based approaches in industry scope projects are not well known or documented. As a result, at Siemens there's currently no common approach regarding model driven development – one reason may be the huge diversity of produced products and provided solutions within Siemens. Currently each software development department has to define its own way of working with models (e. g. [11]). As most model driven approaches however have similarities and, even more important, share common challenges, a goal at Siemens and especially at the Siemens Software Initiative is to analyze and consolidate the topic of model driven development. Building up knowledge on the basic principles when introducing a model based approach to a software department and sharing this knowledge is a major goal during the next years. Understanding and documenting the key challenges and success factors will be the first step to reach this goal.

2.3 Key Challenges and Success Factors

When introducing a model driven development approach a number of key challenges have to be solved. Choosing the appropriate modeling language is the first and most important step. Using e. g. the UML out of the box is rarely possible as most software developers working on embedded systems are electrical engineers and are seldom familiar with the UML. In several projects it turned out that customization of the UML ensures that the language is accepted by the developers – this can be done by restricting the elements and diagrams to be used and by creating domain specific profiles.

As the UML provides no method it is crucial to define one during introduction of an MDSE approach. Typically there are already existing software development methods and processes within the Siemens departments – therefore the challenge is to integrate a model-based approach with minimal impact. Especially the interface to other departments is critical – it rarely happens that all departments introduce such

radical changes simultaneously. Using customized document generators is one possibility to ease the transition between traditional and model based development approaches.

Another important step is to decide on how to organize (i. e. partition and structure) models. Once a tool has been selected that supports the defined approach and also takes the existing tool infrastructure into account, the organization of models is critical. Distributed world-wide development and large teams are common within Siemens software departments and without a proper organization for models an MDSE approach is bound to fail.

The given examples are just a few of the typical challenges one has to deal with when introducing model driven software development at industry scope.

3 Empirical Study

There are a number of supposed key benefits of model driven software development which are constantly reported, e. g. increasing productivity, improving quality, standardization and formalization in [17]. However, there are only few investigations on these statements in large projects and almost no valid empirical data available. Besides, only some approaches discussing the integration with light- or heavy-weight process models exist [3,4]. Furthermore, as described above, modeling as well as model driven development activities are executed but rarely handled in formal software process models in industry.

3.1 Related Work

There is not very much work that relates to empirical investigations of modeling or model driven development in a larger scale, i.e. beyond reports on parts of individual projects. Most work relate to modeling conventions or the quality aspects of certain UML diagram types.

Asadi and Ramsin [16] provide a review of several model driven methodologies and present a criteria-based evaluation and a framework for assessing, comparing, selecting, and adapting model driven methodologies. They compared and evaluated different MDA-based methodologies and concluded:

- The methodologies are not mature enough, especially in regard to supporting standard software engineering activities. Besides, definitions of the methodologies are not complete.
- Most of the methodologies do not offer any guidelines on the usage of MDA tools in coherence with the methodology. All tool-related issues are left to the tool vendors.
- Most methodologies do hardly provide other important MDA features like support for extension of rules, round-trip engineering, model synchronization. and model verification/validation.

Mohagheghi and Dehlen [17] reviewed 25 empirical studies by evaluating reasons for and effects on applying the model driven development paradigm in industrial projects. Among others, they present the following findings:

- Increasing productivity (and shortening development time) and improving quality may be regarded as the ultimate reasons for applying the model driven development paradigm.
- Most known processes are not tailored for the model driven development paradigm. Besides, the paradigm does provide any support for the software development process or the design methodology. It may be unrealistic to use a pure model driven process as "software engineering methods are not fitted to use models as main artifacts, i. e. activities such as analysis and evaluation is still largely done at the code level" and "software engineering environments are not mature enough."
- Some projects suffer from productivity loss due to immature tools and high start up costs, and that modeling can be at least as complex as programming. Also, reasons for not adopting the paradigm are high initial investment and unsure benefits.
- However, "most papers evaluate models as useful for improving understandability and communication among stakeholders".

In any case they identify a demand for more empirical studies and detailed data, as only few reports are on larger projects. Return-On-Investment (ROI) aspects should be evaluated in future as well..

Lange [12] presents a comprehensive set of 21 papers on modeling quality in his dissertation. The work covers papers on model quality attributes, modeling conventions, metrics and quality assessment.

Wang and Brooks [13] describe three empirical studies on conceptual modeling and the modeling process. The studies analyze the efforts of modeling and effects on the modeling process.

Dzidek, Arisholm and Briand [14] accomplish a controlled experiment that investigates the costs and benefits of using UML documentation in an industrial project.

Anda and Hansen [15] describe a case study on the application of UML in legacy development and depict a need for better methodological support on applying UML in legacy development.

In previous works, especially [7,8,10], we describe possible processes and methods to deal with models integrated into a code-centric agile development process, but do not yet provide empirical evidence on its usefulness. Many other works focus only on special issues, e.g. when models are massively used, they need a quality management process. See e.g, references in [1] and [5].

3.2 Outline and Approach of the Study

Our objective is to investigate the conditions and challenges for a successful adoption of the model driven development paradigm at Siemens by running a quantitative survey among a high number of software projects. To our knowledge, typical papers and reports on the application of the model driven development paradigm only focus

on single projects to gather data, e. g. [17]. By contrast, we want to take a high number of projects into account and assess them regarding to modeling and model driven development. From the results of the survey and analysis of the projects we expect to get hints on the key factors for successful modeling in a large, software developing company. We want to derive possible improvements of the software development processes at Siemens.

For the purpose of preparing the survey we are currently running several guided interviews with project managers from different organizational units of Siemens, located at different sites and concerned with different domains. The outcome of these interviews are used to detect the status quo in modeling and model driven development at Siemens and to derive assumptions and hypotheses that shall be proved in the survey.

3.3 Interviews as integral Part of the Study

The qualitative data of the interviews is used to develop the subsequent survey. The interview is set up as a guided interview and covers 54 questions in eight groups:

- Personal data (6 questions): skills, experiences, etc.
- Project information (8 questions): scope, effort, etc.
- Information on the organization and process (5 questions): process model, etc.
- Modeling (14 questions): goals, extent, problems, etc.
- Model Driven Development (7 questions): goals, extent, problems, etc.
- Tools and Technologies (5 questions): usage, problems, etc.
- Team Qualifications (5 questions): education, training, etc.
- Others (4 questions): planning, improvements, etc.

The questions relate to one concrete project chosen by the project manager. The projects should have used the UML as a modeling language and, ideally, used the model driven development paradigm.

In Mid March 2009, six interviews have already been conducted. The interviewees were project managers of (embedded) software projects in the domains of industry and railway automation, automotive and enterprise software. They were guided by the interviewer through the set of open questions, a single interview took up to two hours.

3.4 A first Set of Hypotheses

Although there have only been conducted six interviews so far, the qualitative setup of the interviews allows to postulate a first set of hypotheses that shall be validated in the subsequent quantitative survey:

- Models are hardly used for communication or documentation purposes but mostly for generating purposes. Most teams did not model before they introduced model driven development. The introduction of modeling activities is regarded as a necessity of the introduction of the model driven development paradigm. The terms *modeling* and *generating* are often seen as synonyms as the teams use models only for generating artifacts.

- The model driven development paradigm is often used informal, i. e. there are no formal (organizational) processes or methods related to modeling or the model driven development paradigm. The same applies for the modeling tasks. As mentioned above, modeling is regarded just as a supporting activity but not as a full-scale phase with task, role and outcome definitions.
- Projects that successfully adopt the model driven development paradigm are small and agile. The paradigm is not predetermined from the organization or driven by economical considerations but is started from within the teams ("grass roots movement"). There is often one key team member pushing the adoption of the paradigm.
- The most successful adoption of modeling and the model driven development paradigm is achieved when the team members have formal qualifications (e. g. a computer science degree) and are systematically trained in (UML) modeling.
- The UML is often regarded as a too powerful and complex language. Teams that are insufficiently trained in UML modeling, assume that for successful modeling almost all UML elements have to be used.
- Typical and important reasons for adopting the model driven development paradigm are raising the software quality and enforcement of consistent structures and architectures.

Besides these hypotheses we also want to consider some of the results and hypotheses from the related work (especially from [16,17]).

3.5 Survey

Based on the insights we gained from the interviews we are developing a quantitative survey to validate respectively falsify the hypotheses we postulated as well as possibly further hypotheses from related work. As most questions in the interview are open and hardly to quantify, we have to adopt and transform them to multiple choice questions.

Target group of the survey are software projects which used (UML) modeling or even used model driven development techniques. The problem is that we will not cover conventional projects this way, but we are interested in targeting successful as well as challenged ones, small and large ones and a variety of domains. We want to assert the constraints for successful model driven projects and identify process improvements from a foundational perspective. We want to address project managers from all three sectors of Siemens and estimate a return of at least 100 results which hopefully will be good enough for quantitative conclusions.

We are aware that a pre-selection of the target group could be a threat to the survey, as well as running the survey in a single company. But we assume that Siemens is big enough to be representative for at least other big software developing companies.

4 Conclusion and Outlook

We do know, that model driven development today by far does not yet deliver its promises. Various variants of the paradigm, assisted by various, but often in their functionality very similar tools are not that easy and effective to use, such that quality increases, time-to-market and costs at the same time can be reduced. A number of papers have already discussed that, but we still do not profoundly know what the actual obstacles are. Therefore, we have started this effort and reported on its current status. We will hope that the results help us understanding how to really improve model driven development and will have impact to larger parts of Siemens as well as possibly other German and European industrial companies with heavy parts of software.

Acknowledgements

We are thankful to Prof. Lutz Prechelt from FU Berlin for discussions and critically reviewing our approach from an empirical perspective.

References

1. Fieber, F., Huhn, M., Rumpe, B.: Modellqualität als Indikator für Softwarequalität: eine Taxonomie. Informatik Spektrum /2008, Springer, Berlin (2008)
2. France, R., Rumpe, B.: Model-Driven Development of Complex Software: A Research Roadmap. In: Future of Software Engineering 2007 at ICSE. Minneapolis, pg. 37-54, IEEE, May 2007.
3. Petrasch, R., Fieber, F.: Model-Driven Architectur (MDA) in Verbindung mit dem V-Modell XT. In: Petrasch, R., Fieber, F., Macos, D., Bohlen, M. (eds.): Schriften zum Software-Qualitätsmanagement, Band 4, pp 51-60, Logos, Berlin (2008)
4. Petrasch, R., Fieber, F., Meimberg, O., Morlok, S.: Model-Driven Architectur (MDA) im Kontext von Vorgehensmodellen. In: Cremers, A.B., Manthey, R., Martini, P., Steinhage, V. (eds.): INFORMATIK 2005 - Informatik LIVE!, Band 2 (2005)
5. Frank, U., Rumpe, B.: Qualität konzeptueller Modelle. Workshop Modellierung 2006, 21.–24. März 2006, Innsbruck (2006)
7. Grönniger, H., Krahn, H., Rumpe, B., Schindler, M.: Integration von Modellen in einen codebasierten Softwareentwicklungsprozess. In: Mayr HC, Breu R (Hrsg) Modellierung 2006, 22.–24. März 2006, Innsbruck. Lecture Notes in Informatics (LNI), GI-Edition (2006)
8. Rumpe, B.: Agile Modellierung mit UML – Codegenerierung, Testfälle, Refactoring. Springer, Heidelberg (2004)
9. Rumpe, B. :Modellierung mit UML – Sprache, Konzepte und Methodik. Springer, Heidelberg (2004)
10. Rumpe, B.: Agile modeling with the UML. In: Wirsing, M., Knapp, A., Balsamo, S. (eds.): Radical innovations of software and systems engineering in the future. 9th International Workshop, RISSEF 2002, 7.–11. Oktober 2002, Venice, Italy. LNCS 2941 (2004)
11. Margull, U., Kersten, M. and Regnat, N.: A Model-Based Development Method for Device Drivers. In: Hegering, H.-G., Lehmann, A., Ohlbach, H.J. and Scheideler, C. (eds.): INFORMATIK 2008, Beherrschbare Systeme - dank Informatik, Band 2, Beiträge der 38.

Jahrestagung der Gesellschaft für Informatik e.V. (GI), 8. - 13. September, in München, pp. 656-661, GI (2008)
12. Lange, C.: Assessing and Improving the Quality of Modeling - A Series of Empirical Studies about the UML. Dissertation, Technische Universiteit Eindhoven, 2007, IPA dissertation series 2007-14 (2007)
13. Wang, W., Brooks, R.J.: Empirical Investigations of Conceptual Modeling and the Modeling Process. In: Henderson, S.G., Biller, B.,Hsieh, M.-H.,Shortle, J., Tew, J.D. Barton,R.R. (eds.): Proceedings of the 2007 Winter Simulation Conference. IEEE (2007)
14. Dzidek, W.J., Arisholm, E., Briand, L.C.: A Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance. IEEE Transactions on Software Engineering, vol. 34, no. 3, pp. 407-432 (2008)
15. Anda, B., Hansen, K.: A Case Study on the Application of UML in Legacy Development. ROA’06 May 21, 2006, Shanghai (2006)
16. Asadi, M., Ramsin, R.: MDA-Based Methodologies: An Analytical Survey. In: I. Schieferdecker and A. Hartman (Eds.): ECMDA-FA 2008, LNCS 5095, pp. 419-431 (2008)
17. Mohagheghi, P., Dehlen, V.: Where Is the Proof? - A Review of Experiences from Applying MDE in Industry. In: I. Schieferdecker and A. Hartman (Eds.): ECMDA-FA 2008, LNCS 5095,, pp. 432-443 (2008)
18. OMG: MDA Guide Version 1.0.1. http://www.omg.org/docs/omg/03-06-01.pdf, May 9th 2009
19. Petrasch, R., Meimberg, O.: Model Driven Architecture. dpunkt.verlag, Heidelberg (2006)
20. OMG: UML Infrastructure. http://www.omg.org/spec/UML/2.2/, May 9th 2009
21. OMG: UML Superstructure. http://www.omg.org/spec/UML/2.2/, May 9th 2009

Model-Driven Documentation
The art of authoring model-driven documentation

Philippe Desfray – SOFTEAM

Abstract: MDA can provide great services to document model driven applications. However, we need to define how to use it and what the nature of a good documentation is. Producing documentation is a specific matter that differs from code generation issues: its nature is flexible, not « formatted » as code generation and there are no clear criteria to determine what a good documentation is and to check it. Here is the essence of the problem: it is very hard within existing software developments to find good documentations and to define what the quality criteria exactly are. This paper describes the documentation problems, provides rules for producing good documentations and details mechanisms that MDA technologies shall provide in order to support effective documentation generation.

Key words: Model Driven Documentation. MDA. Methodologies and MDA. Model to Text. Document template.

1. Introduction

MDA technologies are increasingly successful and more and more widely used. The main application of MDA approaches is code generation, or the automation of technical design work, through the application of systematized design and coding principles to a model [1] [2] [3] [9]. Another domain where MDA technology is often used is in the support, assistance and automation of methodological approaches [5] [8]. This type of use is also on the increase, although it does suffer from a certain lack of maturity in terms of documentation production.

MDA literature often considers that the model is the documentation [3] [4]. However, models alone do not provide exhaustive enough information on intention and use, and cannot be understood by all project participants. If model-driven documentation production is a problem handled in many cases, such as in [7], documentation poses a problem in itself. By nature more variable and less "formatted" than code, documentation is consequently far more difficult to define and control. If MDA can indeed provide significant help in the field of application documentation, it is unclear as to how to use it and how to define the nature of "good documentation". And therein lies one of the main difficulties, when one considers how difficult it is to find good documentation and to determine what criteria define good documentation.

This article describes the problem in more detail, provides advice on producing quality documentation and determines the detailed mechanisms that the MDA approach must provide, if good documentation production is to be achieved.

2. Application technical documentation: a rarely satisfactory reality

Documenting an application is all too often a badly understood, badly undertaken activity, which meets neither its objectives nor its intended readership. The result varies between no documentation whatsoever and overabundant, mechanical documentation, that is illegible and does not provide the expected service. It is rare to find well-produced documentation, which provides the reader with exploitable information on, for example, the motivation for the work, the intention of the author, the internal design, the functioning principles, an inventory of the constituents and their role.

The reasons for this are multiple, but are often the result of incorrect motivation (or lack of motivation) for the actual production of the document in the first place:

- **Situation 1:** The documentation is of no use, and authoring it is a complete waste of time. This situation can result from existing documentation that is so wordy and overabundant that new documentation can only add to the general confusion. It can also stem from the reticence felt by many developers when faced with the task of producing documentation. Some approaches like "extreme programming" actually encourage developers to limit their documentary efforts, which suit their natural tendency to avoid documenting their work.
- **Situation 2:** Developers must respect a certain approach (such as ISO 9001, SPICE, CMMI, DOD, and so on) and author pre-defined types of documentation. This type of situation was implemented with MDA automated documentation production in [6]. The initial intention of these documents and their usefulness are forgotten – sometimes they were produced only to follow the certification procedure to the letter – and what remains is the procedural constraint of mechanically producing the deliverables required either contractually or by the standard itself. The author or designer no longer has free rein, and the result is the production of often insufficiently thought-out and frequently useless documentation.

A typical example of documentation dysfunction is the mechanical production of documents on class models, where descriptions of the following type are often found behind systematized chaptering:

The Foo class represents the Foo notion. It contains the Att1, Att2, … attributes.
Or:
Operation getSize ()
Pre-condition
None
Processing

Get the size of the element.
Post Condition
None.

This type of description is useless: no information is provided to the intrepid reader who was brave enough to read it.

MDA technology is a powerful tool for assisting documentation production. However, in all its blind power, MDA technology is capable of the best and the worst.

MDA technology can lead to the automatic production of voluminous, mechanical documents, where cumbersome and mechanical chapter and sentence repetition can render the text illegible. In this case, MDA technology takes on a tedious task, but results in an incomprehensible and difficult to use deliverable.

On the other hand, MDA can also automatically take models and their related diagrams and descriptions, and automatically apply documentation structuring rules. This leads to the production of relevant documents, all the while alleviating authors of the tedious task of copying names, symbols, diagrams and chaptering. The use of an operational documentation approach within an organization is then assisted and institutionalized, thereby guaranteeing consistent documentation quality.

Associated with code production, MDA also guarantees permanent documentation/code/model consistency. MDA can automate tedious and fastidious tasks: for example, the construction of traceability matrices, or the documentation of links (parent classes, child classes, associated elements, and so on). MDA can also introduce calculated, relevant information, such as the measure of the use of a class, its quality metrics, the impact of evolutions, which enrich the otherwise all-too-static information presented.

Documentation production is a skillful task, far more difficult and delicate than code production: the targets are multiple, the rules are neither mechanical nor automatically checkable (no compilation), and the proof of good generation is essentially down to subjective human judgment.

Last but not least, presentation is vital. It must be possible to adapt document presentation each user's individual preferences.

3. Principles for quality documentation production

Without pretending to provide an exhaustive list of all the principles of quality documentation production, the following principles must be respected to obtain professional standard documentation:

- **Principle 1**
  Documentation is an assembly of widely varied elements: for example, an introduction is generally free text with no strong link to the model, graphics and tables can complete a document, appendices are included referencing other existing documents, often not resulting from associated modeling, and so on. Great flexibility is needed when assembling the different parts of a model: it must be possible to have document parts operating different model extractions and uses in any order you like. Thus, if you want a chapter

dedicated to Use Cases, another dedicated to the design model of an application, and finally a chapter for the glossary, listing requirements or other. Documentation production must therefore be flexible.

- **Principle 2**
  Documentation authoring can be model-focused or documentation-focused. Here we distinguish between the documentation construction phase and the documentation presentation phase. According to what type of documentation is being created, it can be more practical to author targeted documentation by model element, providing some sentences to describe it, or else to opt for general authoring taking into account the final presentation of the document and its general articulation.
- **Principle 3**
  Presentation is a highly important issue. However, it must be based on the features of text editors. The notion of style sheet has proved itself. MDA technology must use this knowledge and let the text editor manage this aspect.
- **Principle 4**
  Document template definition requires an easy-to-use, flexible tool, which, as well as handling notions related to the metamodel and to model transformation into text, can also be used by non-technical users, such as quality engineers. Where MDA has traditionally provided highly technical tools for code generation, such as QVT or MOF2Text, destined for true MDA specialists, intuitive, graphical, presentation-focused tools must now be provided for the definition of document templates.
- **Principle 5**
  The responsibility of document template authors and document authors (model designer and author) remains vital. The quality of the end result always primarily depends on the quality of the work carried out. The documentation production system must therefore help authors, enabling them to author in the best, most straightforward conditions.
- **Principle 6**
  There are two types of documentation. On the one hand, documentation that is intended to be read from start to finish, with a logical progression from introduction to overview to detailed view and by theme. This type of documentation requires numbered chapters, indicating the progression of reading. On the other hand, documentation that constitutes part of a repository, for example, the functions of a library or the terms of a glossary, which can be read in any order, dictated purely and simply by the reader's entry point. The reader searches for the operations of a certain class, or for information on the use of these operations, their exact parameters or their meaning, and so on. In this case, chapters must not be numbered, as this would have no sense.
- **Principle 7**
  The authoring and content of documentation must always be oriented towards its targeted reader. For example, some documentation describes the functioning or use of a system or component. An external view is therefore required, and examples of use are helpful. Other documentation is oriented

towards readers in charge of maintenance or those participating in development activities. In this case, an internal view is required: How does it work? What are the functioning principles? What internal choices are made?

What is clearly shown by this is that even if production is automated by MDA technology, the author is still ultimately responsible for it. However, it is also clear that documentation generation is built by assembling fragments, each of which can be automatically generated. Therefore, production of an entire document using one single document template should be avoided, in favor of combining several document templates, each one focusing on a particular theme or chapter, such as the glossary, the Use Cases chapter, and so on.

4. Documentation best practices

Canvas template at the start of the document.

It is common practice to provide a title, the nature of the document (specifications, and so on.), the author, the date and the current version, as well as a history of document versions, the reasons behind these versions and the nature of the updates carried out.

Essential information appears at the top of the page (title and version, for example), with the bottom of the page being traditionally reserved for copyright information, for example.

Summary and detail.

Documenting a model is quite distinct from constructing the model: a progressively educational form of presentation must be implemented, where an overview precedes a detailed view, and where principles must be outlined before going into detail. Thus, a general overview, a kind of first sweep through the model, will enable the reader to get to grips with the scope and structure of the model before looking at each individual element. An initial presentation of the package structure, together with a list of the classes belonging to each package and a summary (one line) of their role, will help the reader to gradually get into the subject. In this way, models are ordered depending on their generality and their phasing during model elaboration. For example, the targeted application can be positioned in a wider environment, or Use Cases can be presented before the description of the system's classes, and so on. The degree of generality of one model with regard to another is a question of context and how to approach the problem to be dealt with. Although Use Cases are often considered as being the first and most general of all models, a conceptual model of an application's domain (class diagram) can, for example, be more general than a Use Case diagram concerning this application.

Glossary.

Inserting a glossary into a document is a practice that significantly improves document quality. We systematically recommend this. The glossary can be situated at the start of the document, if a domain is to be introduced to the reader, or else at the end of the document if it is rather intended to serve as a reference that will be consulted from time to time.

Traceability.

Traceability management is a sign of very high quality. This traceability will define which dictionary term is used to name which model element, which requirement is satisfied by which model element, which business rule applies to which element, and in general, which source initiated the transformation into what.

Graphics and titles.

Graphics in a document must always have a clear title and a number. In explanatory text, the number clarifies references and notes.

For easy and comfortable reading, make sure graphics and text are balanced.

Information mapping.

Documentation must focus on real information and limit "padding". Thus, the attributes of a class, or any type of repetitive property, would generally be better expressed in a table. Sentences like "The Account class contains the following attributes:" should be outlawed, as this repetition serves no purpose other than to weigh down the document, without providing the user with any useful information.

Use variability depending on element importance.

Elements of the same nature do not necessarily have the same importance. Homogeneous and invariable generation would drown important elements in a sea of insignificant elements. For example, let's imagine we have the "Parameters", "Pre-conditions", "Content" and "Post-condition" sections for all operations. This would significantly weigh down insignificant operations with, for example, accessors, while certain operation algorithms would be worth developing through an activity diagram, for example. Different types of situation should be well thought out. Generation behavior in the case of an empty section should be studied in detail: an "Attributes of the class" section should almost certainly be omitted where no attributes are present. However, if a Use Case has no pre-conditions, the documentation must include: "Pre-conditions: none".

Internal links: hypertext.

Hypertext links are a major advance provided by modern textual presentation technologies such as internet/HTML, and also by modern version of word processing applications. Here, MDA provides a considerable advantage, by systematizing links, using traceability, model links (inheritance, association, and so on) and owner/content links. By systematizing their use, documentation provides the reader with a high level of comfort. An example of this is the case of hypertext links from diagrams to represented elements, in other words to the document section that develops their description. However, for documents destined to be printed after distribution, this hypertext feature must be replaced by an explicit link that can be used with paper format documentation, for example: "See diagram on page # or §x.y".

Position detail that can potentially detract from linear reading in appendices

Depending on element complexity, it is often the case that some developers construct detailed models, while other do not. The linear production of detailed models over several pages would perturb the reading of the document as a whole. For example, certain Use Case scenarii are short and textual, while others are expressed using sequence diagrams. For the latter, it can be useful to transfer detailed models into an appendix and to create a link to them from the scenario itself.

Chapter numbering

As seen earlier, chapter numbering should only be used in certain specific cases. Thus, any enumeration of elements of the same nature in a given context (classes in a

package, for example) should avoid numbering, as this would have no particular meaning.

The different themes in a document (Overview, Use Case, Detailed description of the model, sub-systems, and so on) require chapter numbering.

Document versus HTML pages

The characteristics of documents produced for web access differ significantly from the one of traditional paper/text editors. If it is to be read on the internet, a document is no longer read from start to finish, and for this reason a generalized system of links is provided to assist browsing. User browsing and situating must always be made as easy as possible, through the addition of specific features such as a document browser, the possibility of jumping to the next page, the possibility of jumping to a higher-level element (a package from a class, for example), and so on. In this form of documentation, chapter numbering has no point and should be avoided.

Document plan and structure

Generally speaking, models are based on the hierarchical organization of information whose depth can be relatively great in the case of complex or large-scale models. The difficulty with an automated approach is to make this hierarchical organization correspond to the plan of the document. This inevitably leads to a high-depth plan echoing that of the model, one that could in certain cases be difficult to control: the number of title levels in a Word document is limited to 9, whereas the hierarchical structure of model packages is unlimited. This means that it is very often necessary to transform a hierarchical representation of a model to a flat structure in a document. This poses a problem: the document template imposes particular structuring semantics and predetermines the importance of packages according to their depth, which does not correspond to UML or other model semantics.

As a general rule, too many title levels in a document make it difficult to read and understand. The classic example is of a document including six title levels but whose table of contents is limited to the first three levels, so as not to drown the reader in excessive detail. We recommend against exceeding three title levels in most cases.

Document consistency

Those in charge of defining document templates often want to impose a strict framework on document structure, with the intention of helping authors by guiding them and putting them "on the right track". This generally leads to a rich document structure with relatively fine section granularity. The combination of these two factors often produces documents with numerous section titles and yet little informative section content, rendering them highly inconsistent.

Document templates must not therefore impose excessive structuring on a document. A pragmatic rule is to start by limiting the maximum number of section levels.

5. Implementing an MDA strategy

We used the Modelio UML tool (www.modeliosoft.com) and its MDA capacities to define and realize a new strategy for generating documentation from models. This tool includes the support of dictionary, requirements, business rules and goals with the

UML and BPMN model support, and provides traceability management support. One key aspect of this tool is its wide modeling coverage on the complete scope of system modeling: The implemented metamodel supports dictionary (glossary) analysis, business rules analysis, goal analysis, requirements analysis, BPM, Enterprise Architecture and UML2. This enables the generation of complete documentations. For example, the documentation can provide a glossary (from the dictionary), or traceability matrices to requirements.

The key generation principle element lies in the notion of the *document template*. The document template determines the way in which a model will be browsed and transformed to produce documentation. In essence, the document template is a kind of "Model2Text" transformation module [10], but its presentation and functioning are specialized to provide a graphical (rather than a programming) interface for non-technical users, focused on documentation production needs (text, styles, graphics, tables, hypertext links, and so on). The document template enables styles to be referenced specifically by name, which will be defined in the target documentation editor through their specific style sheet capacities. In our case, production targeted MS-Word or HTML.

The innovation here consists in leaving the responsibility of the actual assembly of the document up to the end-user: he designates the model elements from which he wants to produce his documentation, and for each model element selects the document template he wants to apply. He also determines in which order these document parts (document template application on model elements) should be assembled. For example, he can designate a "Use Cases" package, apply a document template dedicated to Use Cases, and then take an "Analysis" package and a document template dedicated to class documentation and assemble the two. Documentation generation will then produce one (or several) chapter(s) on Use Cases with dedicated chapters and presentation, and one (or several) chapter(s) documenting the class design model.

The document produced is itself considered to be a specialized UML artifact. In fact, the principle of assembly briefly outlined here results in a documentation assembly model (that end-users do not necessarily see) with specific logic added to artifacts.

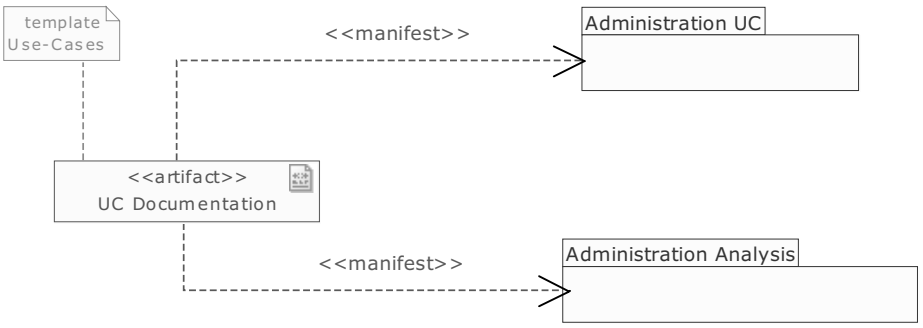

Figure 1 – “UC Documentation” assembly

Figure 1 shows that the "UC Documentation" will be based on the "Use-Case" template, and will be applied to two packages specified by the UML "manifest" link. The order of production can be specified by the "manifest" order in the model explorer.

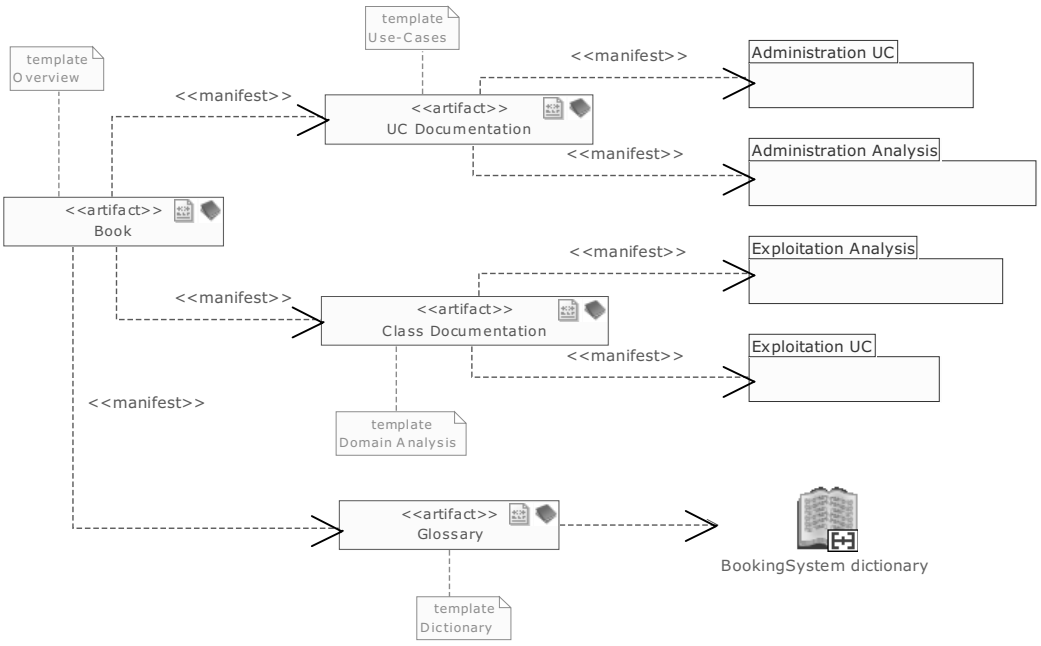

Figure 2 – Assembly of several parts within a main document

In Figure 2, a document (Book) manifests other documents, in this case specifying the assembly of several document parts into one container document. Each assembled document has its own template and its own selection of model elements to be documented. We see here that a glossary is incorporated, by manifesting a dictionary (sort of package of "Term"). This is an extension to UML.

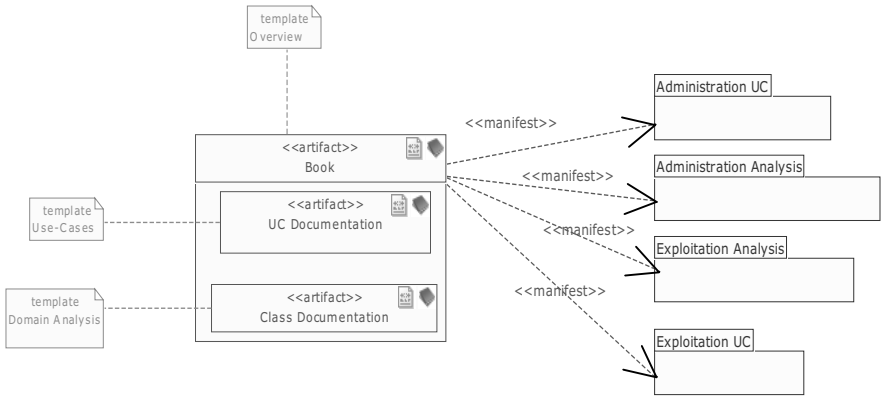

Figure 3 – Factorizing manifestation links through embedded artifacts

Figure 3 is an added sophistication to Figure 2. The use of embedded artifacts specifies that each embedded artifact will be applied to the elements manifested by the container artifact. This is simply a factorization mechanism, useful when several document templates will process the same model elements with a different purpose.

These principles have helped us to cover a large variety of documentation needs through a limited set of document templates (Figure 4). Instead of having templates like "analysis", "requirement analysis", "detailed analysis", "design" and so on, templates have been designed on two axes: the horizontal axis is related to lifecycle (e.g. Use-Case -> Integration), while the vertical axis refers to the intended use of the document (as a reference guide, or as a specification – see the "Principle 6" dichotomy).

By assembling document parts based on these finer grain templates, the end-user will specify his intentions in a context-specific way.

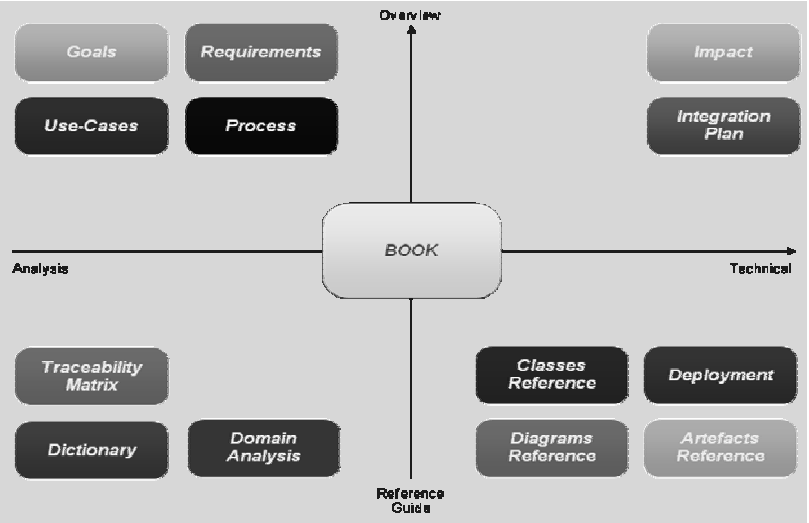

Figure 4 – Overview of the predefined document templates included in the Modelio solution

Predefined templates assembling these basic sets can also be provided.

6. Conclusion

Our experience shows that we have increased documentation generation capacity through these techniques. Documentation is a delicate issue that requires a mixture of MDA technology and specific tools that adapt generic "model to text" capacities.

However, the end-user still has a lot of responsibility over the resulting documentation, which is not a black box generation feature. Authoring documentation remains an art, with MDA simply being a tool that empowers its production.

Special thanks:

Bruno Lheritier, Christophe Malgouyres and Philippe Vlaemynck from SOFTEAM have brought decisive ideas and work to the elaboration of these techniques, and to the production of high quality documentation.

7. References

1. OMG. Unified Modeling Language Specification. V2.1. 2006.
2. OMG. Model Driven Architecture (MDA). Technical Report OMG Document ormsc/2001-07-01, Object Management Group, 2001
3. Anneke Kleppe, Jos Warmer, Wim Bast: MDA Explained – The model Driven Architecture : Practice and Promise,; Addison-Wesley 2003 ISBN:032119442X

4. Thijs Reus , Hans Geers and Arie van Deursen: Harvesting Software Systems for MDA-Based Reengineering Springer Berlin / Heidelberg – ISBN 978-3-540-35909-8
5. A Estévez, JD García, J Padrón, C López, M …: System Integration Methodology Based on MDA - European Conference on Model Driven Architecture-Foundations …, 2006 - Springer
6. Cristina Vicente-Chicote, Universidad Politäcnica de Cartagena, Begoña Moros and Ambrosio Toval, Universidad de Murcia: REMM-Studio: an Integrated Model-Driven Environment for Requirements Specification, Validation and Formatting ;– JOT Vol. 6, No. 9, Special Issue: TOOLS EUROPE 2007, October 2007
7. , Feng Yang, Hongji: Model Oriented Evolutionary Redocumentation ; Chen; Computer Software and Applications Conference, 2007. COMPSAC 2007 - Vol. 1. 31st Annual International ; July 2007
8. P. Desfray: Broadening the use of MDA.– 2005. White paper – www.objecteering.com
9. P. Desfray: "Object Engineering - The fourth dimension – Addison Wesley - 1994."
10. MOF Models to Text Transformation Language version 1.0 ; OMG document formal/08-01-16
11. P.Desfray: UML – Making a success of preliminary analysis using UML - "Pre-modeling" techniques: a bridge to the model. 2004. White paper – www.objecteering.com

A Comparative Case Study of Model Driven Development vs Traditional Development: The Tortoise or the Hare

Tim Kapteijns[1], Slinger Jansen[1], Sjaak Brinkkemper[1],Henry Houët[2], Rick Barendse[2]

[1] Utrecht University, Dept. of Information and Computing Sciences,
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
{s.jansen, tkapteij,s.brinkkemper}@cs.uu.nl

[2] Response B.V,
P.O. Box 270, 5240 AG Rosmalen, The Netherlands
{rick, henry}@response.nl

Abstract. At present, model driven development is showcased for large software development projects while simultaneously in industry, model driven development is used extensively for small-scale projects as well. The application of model driven development for small-scale development projects has received little attention, reducing the ability to theorize or develop it. This paper presents a case study of the development of a small middleware application, which is further contrasted with a number of leading studies on model driven development. The case study shows that model driven development is well applicable to small-scale development projects under easily satisfiable conditions.

Keywords: Model Driven Development, small middleware development, Model Driven Engineering.

1 Introduction

Model Driven Development (MDD) is maturing and promises to solve problems with large, complex system development [1, 2, 3]. MDD promises to decrease development times and reduce maintenance effort while product quality increases. Interest in MDD is growing; many companies are taking the first steps to adopt MDD.

The largest challenges in software development are the lack of abstraction and high complexity; software does not have a representation in reality, like floor plans, leading to abstraction difficulties [4, 5]. The software industry itself adds to this challenge by expecting more reliable software and faster delivery [6].

High application complexity and high abstraction makes communication between team members difficult, in turn leading to delays, flaws, and missed deadlines [5], and even a single flaw in software can cause major problems [4]. When applying MDD and

using models as primary development artifacts, the abstraction level is raised making software more visual and thereby reducing complexity [4].

This study compares an MDD implementation with regular third generation programming, helping in the analysis of the impact that MDD has on small system development. The MDD framework used is a proprietary framework aimed at developing simple applications, called XuWare. Simply said, XuWare generates "create-remove-update-delete" functionality for web applications from UML models.

In the next section we will elaborate on related work. Section 3 continues with the research description. The case study results are discussed in section 5. The conclusions are presented in Section 6.

2 Related Work

There are already several case studies on MDD available, a summary can be found in table 1; the results of these differ. Both development performance increase and decrease have been observed. Two case studies, performed by Anda and Baker [7], and Baker et al. [8] display contrasting results. Both case studies are performed in large-scale projects applying different MDD frameworks. The first project concerned a legacy development project, being a case study based on questionnaires and interviews, the second concerned the development of a new system identifying particularities when implementing MDD at Motorola. The use of MDD to develop and enhance legacy software created implementation difficulties when replacing the old application code with new; interfacing with the old code was more time consuming when using MDD than without MDD. New development projects seem to benefit the most of MDD.

Table 1. Relevant literature

Case Study	*Issues Encountered*	*Conclusion*
[8]	Difficult to set up a parallel project for large developments. MDD tools scalability is poor and Semantics have to be extended.	Successful implementation and increases in productivity and decrease of defects.
[9]	Some workarounds needed that decreased performance. Integration of the legacy software was more difficult with MDA.	There was no proof of increased development speed.
[7]	Introducing models for existing code was expensive. Identifying and using use cases was the most difficult for legacy development	Some positive results were measured with the use of sequence and class diagrams.
[10]	Decomposition of rules is still difficult.	A magnitude in code reduction can be achieved.

[11]	Part of the requirements had to be interpreted from the code, and part in models. Debugging was only available on the code level, making debugging a more complex task.	MDD has helped in quickly retargeting the application. Increase in production a defect removal.
[12]	Defining the requirements in a format that fits in the toolset needs further investigation	Developing an adequate tool chain will increase performance.

There are different forms of MDD: Model Driven Architecture (MDA), Agile Model Driven Development (AMDD) focused on modeling just enough and Feature Oriented Model Driven Development (FOMDD), which focuses on creating models with predefined parts. These three techniques differ in the application of MDD, MDA uses extensive models for application development while AMDD only uses basic models and model parts are used in FOMDD to compose the application, but overlap in that model parts or complete models are used the base of the application.

The ability of MDD to improve productivity has been proven in many large projects [13]. MDD has proven itself for large enterprise application development, but few studies on smaller software projects were found; this gap is an opportunity for researching the application of MDD in small-scale development projects.

3 Research Approach

Based on scientific literature and previous case studies, the following research question is formulated: *"What is the effect of Model Driven Development on the productivity of software development in a small scale middleware development project?"* MDD should reduce development time, small software development projects should benefit similarly from MDD as larger projects do.

This research is conducted using action research and case study methods. First, a legacy middleware application is rebuilt using MDD. The development time needed to redevelop the legacy application is then compared to the development time of the legacy middleware application. The new application will be tested with a number of test cases created specifically for the legacy application to compare whether the applications are *functionally equivalent*.

Three measures are used to test for functional equivalence. To quantitatively compare development productivity of both applications the *Function Point Analysis* (FPA) measure is used [14], using the International Function Point Users' Group (IFPUG) Function Points. The second comparison test is a user test, where users are asked to evaluate whether the new application could be a substitute for the legacy application. Thirdly, experts are asked to assess the framework

and the application development process. Through a questionnaire co-developers are asked to report on the quality and maintainability of the generated code.

The case study is performed as action research. *Action research* is defined as: "a recognized form of experimental research that focuses on the effects of the researcher's direct actions of practice within a participatory community with the goal of improving the performance quality of the community or an area of concern" [15, 16]. Essentially action research is research that involves all relevant parties in actively examining current tools and techniques in order to change and improve them [17]. The researcher is the primary developer, studying this case while actively participating in the development will give valuable insights in how to improve MDD when used for small application development.

This section contains a description of the case study validity threats. These threats need to be addressed to ensure a valid case study. The first validity threat is the possibility of incomplete information on the legacy development project. To perform a proper comparison first hand development information about the development time of the legacy application is required. The original developer and management are interviewed and reports have been studied to determine project length and cost.

A fellow researcher monitors the research process to prevent a positive bias to the research.

4 Case Study: DMS Case

Response BV is a small company focused on Enterprise Application Integration (EAI). MDD can play a vital role in the development of middleware for the EAI sector. With faster and easier development of middleware large productivity increases can be gained, if successful MDD can help to achieve shorter integration projects. MDD is an upcoming paradigm, this MDD study will give insight in the use of MDD in general and the potential it can offer to Response B.V.

MDM satellite is an application of medium size and was developed in two months; an application that fit the resources of the researchers. Data Management Satellite (DMS) is a master data management application. DMS contains extensive product information that is used by retailers to order products and to estimate shelf space. Retailers or warehouses subscribe to the data; changes in the data are then pushed to the subscriber at the moment of change. The chosen application matches the application type that would be generated with MDD in this company. One of the customers currently using this application has agreed to cooperate in the application tests.

4.1 Application development

A single model does not contain sufficient information to generate a full application at once; full software generation is not the goal of this study. For the creation of a small application MDD is combined with a middleware framework "XuWare", leading to faster application development. The more complex requirements such as business logic and graphical user interfaces have been built with manual code, extending the framework.

The framework contains generic Graphical User Interface (GUI) elements, such as text boxes, data access components for database access, XML reading and writing, etc. The generic components of the framework create the basic functionality of a middleware application. The framework is built up in a modular fashion, created in ASP.net with VB.net. Most of the domain parts of the application are generated through UML class diagrams. The generator creates application code, default user interfaces and an application database. The translation rules used for generation are programmed into the generator. The generated code (approximately 70% of the required functionality) is tailored to the framework. The remaining parts, such as complex graphical user interfaces, will be created manually after generation.

Generating the application code from the application model starts with loading model elements from the database. The complete generation process consists of creating an application database, framework code and framework configuration code. After code is generated, the files are placed into the framework directory of the new application and the application becomes functional. To finalize the process the application has to be configured.

Using manual code for the more complex requirements has some drawbacks; after customizing the code, the models are reduced to documentation, and with time outdated documentation. When the code is changed and the models are not changed accordingly they lose their value. Round trip engineering [4] could potentially solve this issue, updating models based on the code provides the possibility to visualize the existing code [18]. In this study MDD is used without round trip engineering.

5 Case Study Results

The total application development time was 16 days; including generation and customization. The development of the custom code took 13 out of the total 16 days. The development of the first models and the generation was performed in 3 days of the development time and created about half of the application functionality.

Both projects are developed in VB.net;. The generation combined with the use of an application framework has increased the performance of the development with an inexperienced developer. Compared to the legacy application developed in 42 days the new development improves productivity for this small application.

The generated application contains 8 of the 16 identified application functions; approximately half of the application functionality. The other eight functions contain or consist solely of complex business logic, which cannot be generated or modeled using the XuWare framework.

Eight versions of the model were created before the model was finalized. The base model was not fully compatible with the model needed for generation; the syntax was different. Parts of the model had to be changed to create a model suited for generation

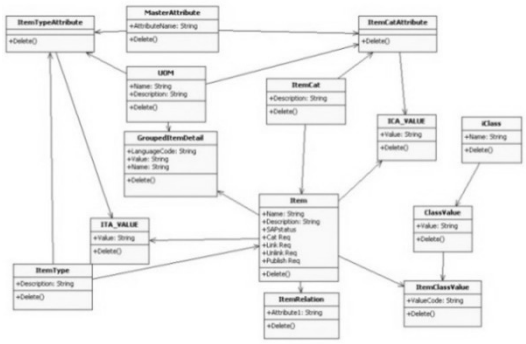

Fig. 1. Class Diagram of DMS Application

5.1 Evaluation and Comparison

The complete test of the application was performed in 1.5 hours, at a location specified by the user. The user test was started after a small introduction of the test and the goal of the test. The testing user was given the test form and the test cases and could perform the actions specified. To allow the user to add missing functionality on the list, several empty lines were added to the form. During the test, the user specified one missing test case; this functionality was present in the application so it posed no problem. The results are summarized in Table 2. Most remarks from the user relate to missing functionality, but this functionality is also missing in the legacy application. The lack of this functionality indicates that the new application has the same flaws. If the application was developed and extra functions were available it would not be functionally equivalent.

5.2 Function Point Comparison

When looking at the Function Point (FP) count of both the applications a difference of 2 function points is seen (Table 2); the new application has less FP than the legacy application. This is due to the XuWare framework. Some functionality has been replaced or reordered resulting in an DMS application with less function points.

Table 2. Productivity Comparison

	FP Count	*VAF*	*Unadjusted FP*
Legacy Application	51	0.86	59
DMS Application	49	0.84	58

According to the value adjustment factor (VAF), measuring application complexity, the legacy application is more complex. The unadjusted FP count is adjusted on application complexity with the VAF to calculate the final amount of FP. The difference in FP does not complicate the calculation. A slight difference in the applications FP count can be explained with the use of MDD and the perception on functionality. The difference in FP of approximately 4% is due to perception, the development method differs, and so does the presentation of the application.

The VAF only differs in reusability, the code generated can be reused, and manually created code is less re-useable. Less re-usable code is no issue for the application functionality. The specific code is mostly generated code thus re-use is no issue, the code will be regenerated for new applications.

The development productivity is measured in FP per hour. With the FP count comparison the time spend on realizing a single FP is calculated (Table 3). The time spend on a single FP is used to compare development performance of both applications. The productivity is based on the measure FP and the measure development time in hours.

Table 3. Development Productivity

	FP Total	*Development time (h)*	*Productivity (FP/h)*	*Improvement*
Legacy Application	51	400	0.13 (7.8h/FP)	Base
DMS Application	49	128	0.38 (2.6h/FP)	192% (2.9x)
DMS Application, after bug fixes	49	144	0.34 (2.9h/FP)	162% (2.6x)

The legacy application developer spent 400 hours on application development, realizing 51 FPs, resulting in a productivity of one FP per 7.8 hours. The application was rebuilt using MDD, the framework, and

manual coding in 144 man-hours. The productivity for the old application was approximately an FP per workday, in the MDD project a productivity of one FP per 2.9 hours was achieved; an increase of 162%. The function point analysis displays an increase in development performance. Both the user test and the FP count prove a successful development that resulted in a functional equivalent application.. The use of MDD resulted in a 2.6x increase in development productivity. The results from the user test did not provide problems and resulted in minor bug fixes.

5.3 Solution Comparison

There are several other application and code generators, the XuWare framework will be compared to three other generation frameworks (table 4). The other generators are Ruby on Rails (RoR), 42 Windmills (42W) and the technique used by Schilt [19]. All of these techniques allow development of small applications and require manual code to some extend.

RoR is an open source framework written in the Ruby language. RoR focuses on developing application with less and easier to understand code, using the model-view-controller principle. To achieve this ease of use RoR uses textual models to create the application. Simple commands are used to generated functionality throughout the RoR framework, creating the necessary views and controllers,

The generator technique used at 42 Windmills is based on and interface that allows users to define an application. 42W does not use any graphical models to defined the application. Based on the entities and attributes provided by the user through the graphical interface the 42W generator creates an entire application. The 42W generator creates a 4-tier application and has potential to add models into the development.

Schilt has developed and MDD solution based on a database model. This model is than used to generate source code for the application. The generated code contains basic user interfaces and default CRUD methods. Additional meta data is than used to create a basic but working application.

When comparing these three methods with XuWare it becomes clear that Schilt uses a similar solution. Both MDD implementations use a single model to generate an application or application parts. In the next paragraphs these techniques used for fast development of small applications are compared.

Information was gathered on all techniques using software and document study. In the case of 42 windmills interviews were held with developers and the product manager of 42 Windmills.

Not all of these techniques are Model Driven Development as defined in this study, RoR and 42 Windmills do not use visual models to develop applications. RoR uses an textual modeling language and 42 windmills use a graphical user interface. The MDD method used in this study and the method of Schilt use graphical models to develop applications. The base for comparison is the ability of all methods to create small applications in a short period of time.

All generation methods differ, the method used by Schilt is closest to the technique used in this study; using a single graphical model to develop a working, basic application. The generation technique of 42 windmills creates a more complex application, without the use of models. Textual models as used in RoR speed up the development although do not really help developers in understanding the system, there is still the need for manual coding.

Comparing XuWare to 42 windmills provides interesting results; 42 windmills is capable of generating more complex applications using SOA. Compared to the XuWare generator the 42 windmills generator generates more code. The entire application is generated after input via a graphical user interface, with an empty logic layer. When adding models to the development interface of 42 windmills and techniques like Business Process Management (BPM) and SOA can help in developing complete applications through models. Development and executing of business rules with models is already possible.

Table 4. MDD comparison

Characteristics **MDD Technique**	**XuWare**	**RoR**	**42 W**	**Schilt**
Type of application generated	*Business*	*Basic*	*Business*	*Business*
Use of Graphical Models	X	-	-	X
Model Type	UML	DSL	None	Database
Graphical Model	X	-	None	X
Amount of models	1	0	None	1
Model Name(s)	Class	Ruby	None	Data Model
Productivity increase proven	X	-	X	X
Generated applications in use	-	X	X	-
SOA applications	-	-	X	X
Manual coding necessary:				
Yes, for missing functionality	X	-	X	X
Yes, always	-	X	-	-
No	-	-	-	-
What can be generated:				
Database tables	X	-	X	X
Data Creation (C)	X	X	X	-

Characteristics	MDD Technique	XuWare	RoR	42 W	Schilt
Data Lookups	(R)	X	X	X	X
Data Editing	(U)	X	X	X	-
Data Deletion	(D)	X		X	-
GUI:					
Generated through the framework			X		X
Generated through the code generator				X	
Generic parts in the framework		X			X
Specific parts in the generated code		X			

5.4 Discussion

Approximately 50% of the application functionality was generated. A productivity increase of 2.6 times faster compared to regular development was found. The results of this study are encouraging, and display that applying MDD in small application development is useful. There is little information on the effect of MDD on the long-term maintainability of software in the case of small application development. More information on the effect of using models on the maintainability of systems is required to determine long-term cost implications.

Based on the case study great potential for MDD in small application development is discovered. MDD has potential for even higher performance gains with the use of additional simple models. Performing further research in the application of MDD and the implementation of fully automated MDD [4] can aid the adoption for smaller projects.

Based on the results from the case study it is discovered that MDD has potential for small application development, using a basic form of MDD. There are several improvements imaginable. The first improvement is extending the application generator to generate more of the complex and yet repetitive tasks in software development, such as framework configuration for the application specifics. The configuration of the application is a suitable task for the generator but is performed manually in this study, taking a full day to complete in this study; a performance increase can be gained by extending the generator.

The models can be extended further to achieve a higher amount of code generation. Class diagrams and sequence diagrams are chosen as viable models for the model subset. Extending the MDD method used in this study with sequence diagrams will result in extra code generation due to more detail in the model; this could however result in

redundancy issues. Creating relationships between these models will be important success factors.

An improvement for the method but not related to development effort is the adoption of round trip development in the process. After the first application parts are generated, the models are abandoned and development reverts to manual coding. The models become obsolete and do not offer their potential value. Adding round trip will increase the value of the application models by keeping them up to date with the development. Then the models can be used to maintain the application, providing knowledge about the current structure and state.

6 Conclusions

The question of how MDD affects the development on a small middleware application is answered with the results of the case study. The case study concerns the redevelopment of a middleware application through MDD, using class diagrams as the source for code generation.

For the presented case, it appears unfeasible to generate a complete application from a single model. Manual code had to be added for complex business logic. Manual development took thirteen days; almost eighty percent of the development was spent on creating extra functionality. The productivity increase is largely dependent on the amount of functionality created after generation. The legacy application has been developed in 42 days. With the use of MDD, a similar application is created in 16 days resulting in a considerable short-term performance increase.

Concluding, the answer to the main research question is: "The application of Model Driven Development in this project resulted in a performance increase of 2.6 times compared with the legacy project, thus proving MDD successful in increasing development productivity. Development productivity in this project is improved with the use of Model Driven Development."

7 References

1. Balasubramanian, K, Gokhale, A, Karsai, G, Sztipanovits, J, Neema, S: Developing Applications Using Model-Driven Design Environments. (2006)
2. Hailpern, B, Tarr, P: Model-driven Development : The good, The bad and the ugly. IBM systems journal., Vol 45., No 3, 451-461 (2006)
3. Picek, Ruben, Strahonja, Vjeran: Model Driven Development Future of Failure of Software development., Varazdin (2007)
4. Selic, B: Model-Driven Development: Its Essence and Opportunities. Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing

(ISORC'06), 313-319 (2006)

5. Brooks, F: No Silver Bullet: Essence and Accidents of Software Engineering. Computer 20(4), 10-19 (April 1987)
6. Bendraou, Reda, Desfray, Philippe, Gervais, Marie-Pierre: MDA Components: A Flexible Way for Implementing the MDA Approach. LNCS 3748, 59-73 (2005)
7. Anda, B, Hansen, K: A case study on the application of UML in legacy development. Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering , 124-133 (2006)
8. Baker, P, Loh, S, Weil, F: Model-Driven Engineering in a Large Industrial Context — Motorola Case Study. LNCS 2005(3713), 476-491 (2005)
9. MacDonald, A, Russell, D, Atchison, B: Model-driven development within a legacy system: an industriy experience report. In : Australian Software Engineering Conference , pp.14-22 (2005)
10. Hemel, Z, Kats, L, Visser, E: Code Generation by Model Transformation. Springer, Delft (2008)
11. Kulkarni, V, Reddy, S: Model Driven Development of Enterprise Applications. LNCS 2005(3297), 118-128 (2005)
12. Bozheva, T, Bailey, T, Ritter, T: Applying MDA in the avionics : A Practical Approach. The Second Workshop" From code centric to model centric" (2006)
13. Gally, M: What is MDD/MDA and where will it lead the software development in the future?, Zurich (2007)
14. Jones, C: Applied Software Measurement 3rd edn. McGraw-Hill, USA (2008)
15. Dick, B: Action research: action and research. In : Doing good action research, Southern Cross University (2002) http://www.scu.edu.au/schools/gcm/ar/arp/aandr.html.
16. Hughes, I, Seymour-Rolls, K.: Participatory Action Research : Getting the Job Done. Action Research E-Reports, 4 (2000)
17. Wadsworth, Y: What is Participatory Action Research? In: Action research international. (Accessed 1998) Available at: http://www.scu.edu.au/schools/gcm/ar/ari/p-ywadsworth98.html
18. Portier, Bertrand, Ackerman, Lee: InfoQ : Model Driven Development Misperceptions and Challenges. In: InfoQ. (Accessed Jan 21, 2009) Available at: http://www.infoq.com/articles/mdd-misperceptions-challenges
19. Schilt, Maarten: Applying Model-Driven Development to Reduce Programming Efforts for Small Application Development., TU Delft (2007)