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

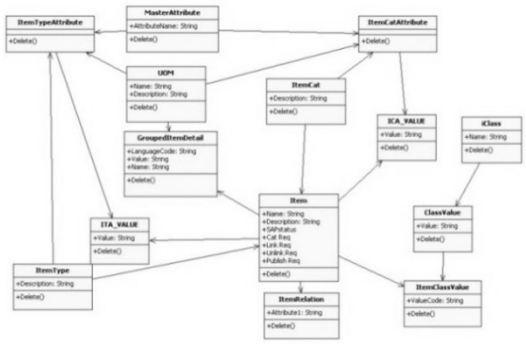

Fig. 1. Class Diagram of DMS Application

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)