A Case Study on Model-Driven and Conventional Software Development: The Palladio Editor

Klaus Krogmann, Steffen Becker
University of Karlsruhe (TH)

{krogmann, sbecker}@ipd.uka.de

Abstract: The actual benefits of model-driven approaches compared to code-centric development have not been systematically investigated. This paper presents a case study in which functional identical software was once developed in a code-centric, conventional style and once using Eclipse-based model-driven development tools. In our specific case, the model-driven approach could be carried in 11% of the time of the conventional approach, while simultaneously improving code quality.

Keywords: MDD, DSL, model-driven software development, case study

1 Introduction

During the last decade, model-driven development has made enormous progress and supporting tools have improved significantly. Current tools form a second generation of instruments to support model-driven development. However, there is a lack of comparisons of new, model-driven development techniques with code-centric software development in terms of efficiency, code quality, and time effort.

The contribution of this paper is a case study on code-centric and model-driven software development. Software with the same functionality – an implementation of a software component meta-model and a graphical editor for that meta-model – has been created twice: "by hand" and with support of recent model-driven development tools. The case study has not been designed as a controlled experiment (i.e, we could not control some factors influencing the results of our study). Nonetheless, we found large differences among both approaches in terms of software development efficiency, time effort, and quality of code. Those differences are worth reporting despite the limitations of the case study (see section 4). This case study allows to draw conclusions for other software development projects of the same domain (graphical editors for a meta-modelled domain). By this, the paper tries to lower the lack of knowledge on the actual benefits and limitations of specific approaches in software engineering.

The paper is structured as follows: In section 2, we present requirements of the developed software, which includes a characterisation of the two compared software development projects. In section 3, we compare these two projects and clarify to which extent they are

comparable. Finally, we summarize the results of the comparison and present future work in section 6.

2 The developed software systems

Both software development projects that being compared developed an implementation of the Palladio Component Model (PCM, [Kro06], [BKR07]) and a corresponding graphical editor ("Palladio Editor").

The PCM is a research oriented meta-model to describe models of component-based software architectures with focus on Quality of Service (QoS) aspects. The PCM has a large variety of entities like components, interfaces, roles, and connectors. Requirements for an implementation of the PCM include persistence, model observers, query methods, and model validations.

The Palladio Editor is a graphical editor supporting multiple views of the model. Those views can be compared to the UML 2 component view (cf. [Obj05]), the UML 2 deployment view, and a view to illustrate the behaviour of components. Additionally, containment relations need to be supported to visualize composite component structures. Users of the PCM implementation and its editor are mostly researchers and students, no industrial maturity level is required.

There are several constraints complicating the development. The PCM is constantly changing. New research ideas shall be directly reflected by the PCM and its editor. As a result the PCM changed frequently during the development process. Furthermore, the required PCM implementation needed to be flexible to adapt for new requirements and extendible to integrate new views, support many analysis tools, and changing validation routines. These requirements had to be reflected by the architecture of the PCM implementation.

2.1 Conventional Development: Ride.NET

Overview The first software development project was performed by a student project group "Ride.NET" at the University of Oldenburg, Germany [PGR05]. In total 13 graduate students were involved in the project. The project started with a master thesis [Ufl05] that evaluated different extensible architectures for large scale editors. The output of this work was a prototype of an editor framework. Afterwards the main project group of 11 students started developing the editor for the PCM. In parallel, one student assistant developed a PCM implementation.

The development was done conventionally (i.e., through writing the code by hand). The software was developed in C# with the .NET framework 1.1. Besides the .NET framework only one important larger library was used: Netron, a free graphical library [Pro]. The remaining software was written from scratch.

The PCM and editor implementation finally were capable to support all requirements listed

in section 2. Additionally, the development included an import plugin that allowed to detect COM components [Cor] from given C# code and an export plugin able to generate C# code skeletons based on PCM specifications. During the development, often new requirements of the editor directly initiated development at the PCM implementation. A considerable effort was spent on debugging.

Time effort The project group developed one year in total. 11 students were involved in the development. For each student, we assume a weekly time effort of about 20 hours[1]. In total this sums up to 11 students * 20 h/week * 52 weeks = 11.440 man hours for the members of the project group. Additionally, one has to calculate the effort for the development of the PCM implementation by the student assistant. The student assistant had 30 hours per month, summing up to 30 h/month * 12 months = 360 man hours.

The project using conventional development took a total of about **11.800 man hours** to write the PCM implementation and the editor. The master thesis, which provided an initial prototype architecture is not considered in the estimation, as it had a different focus.

2.2 Model-Driven Development: GMF / EMF

Overview From November 2005 to April 2006, a second project started. The aim of this project was almost the same, but the chosen tools and frameworks were different. The project was a master thesis [Kro06] of one student that previously was involved in Ride.NET. Instead of hand-written code, the project completely focused on a model-driven approach to create an implementation of the PCM and a corresponding editor. At the end, all requirements specified at the beginning of section 2 could be fulfilled.

Development was done with IBM Rational Software Architect and the latest developer versions of the Eclipse Modeling Project [Ecl06], including Eclipse Modelling Framework (EMF), Graphical Editing Framework (GEF), and Graphical Modelling Framework (GMF).

Time effort If the same time measuring is used for the second project like for the first, one gets a total of 1 student * 26 weeks * 50 h/week = **1.300 man hours**.

3 Comparison

Just comparing the time effort of both projects reveals that Ride.NET took about ninefold the time of the master thesis project. One has to look into the project's details to grasp other quality attributes of the software like error rate, maintainability, extensibility and flexibility to make the comparison more serious. Table 1 gives an overview on the differences and commonalities of the two projects with respect to the attributes discussed in this section. Other assumptions and limitations are listed in section 4.

[1] The project group was treated as two lectures (24 ECTS points) à 4 hours per week; for one hour of lecture usually 2.5 hours of time have to be spend; this makes in total 20 hours per week per student

Criterion	Ride.NET	GMF-Editor
Programming language	C#	Java
Implementation	manual	generated
GUI-Library	Netron (“buggy”)	GEF (“mature”)
Base framework	.NET / Netron	Eclipse / EMF / GMF / GEF
Undo / Redo	implemented manually	given by framework
Event-mechanisms	implemented manually	given by framework
Extendability	given	given
Development effort	high	medium to high (depending on the degree of customizations)
Effort after model changes	high	medium (depending on the degree of customizations)
API of the component model used	by demand	capacious
Bug probability	high	low (systematic bugs)
Unit Tests	manual	generated test stubs
Effort for learning	high – frameworks have to be learned	medium – MDD-Tools abstract framework complexity
Time need	about 4 man year	about $\frac{1}{2}$ man year

Table 1: Comparison of Ride.NET and a GMF based editor for the component model

Development time was saved by the model-driven approach starting with the first development cycle, because good transformations existed. Usually for model-driven approaches in-front costs are assumed.

It could be stated that the quality of the generated code was much higher, than the one of the hand-written code. Especially, the most common types of failures changed: the faults of generated code usually are more systematic. Due to good templates the probability of failure was lower for the generated code.

One reason, why the development took more time in the Ride.NET project was the software infrastructure. Ride.NET had to develop the whole infrastructure and architecture “by hand”, while for the model-driven approach, the Eclipse framework plus the infrastructure of EMF, GMF and GEF could be used. Using the mature infrastructure of EMF, GMF and GEF (instead of the immature Netron library plus hand-written code for the editor) provided further advantages: The combination of Eclipse and GMF provides a generated redo / undo stack. The Eclipse plugin mechanism can be used for extensions – without coding the plugin architecture. For the PCM implementation (generated by EMF), an event based synchronisation of non-directed associations is generated and skeletons for unit tests can be created automatically. Overall, the generated API for the PCM was capacious instead of need-driven.

For both projects there was a high initial effort for learning the frameworks. In case of Ride.NET one had to learn how to *program* with those frameworks (i.e. Netron) – in the case of the model-driven approach, one had to learn how to *use* the frameworks for the model-driven process (EMF, GMF). In the latter case, there is less effort, as model-driven development provides an abstraction of framework complexity.

For the domain of graphical editors that follow a meta-model, model-driven software development provides substantial benefits. In contrast to the hand-written code, the generated PCM implementation as well as the graphical editor could be adapted to a new version of the meta-model (with moderate changes), that was caused by research requirements, within less than half a day by one person. Only a model-driven approach supports such fast update cycles.

4 Limitations

As already mentioned in the introduction, initially the projects were not designed to provide a controlled study. Hence, the differences among both projects should be pointed out clearly. In Ride.NET there was a requirements analysis phase that took about two months. Later, for the model-driven approach the focus of requirements analysis laid on the PCM instead. The group of 12 students in the Ride.NET project caused a lot of communication overhead. Some time is usually assumed to be spend on communication and coordination in large groups. In the Eclipse based model-driven project the architecture was given. Hence it would not have been possible to design the architecture according to very special needs. In particular, hand-written architecture and code tend to be more customisable. For the needs of the PCM / Palladio Editor, the customisations provided by EMF / GMF were mostly sufficient.

In the model-driven project, the focus was not on writing source code. Instead, the PCM meta-model had to be defined, useful tools had to be explored (EMF / GMF were not set), useful mappings and input models had to be defined – especially for GMF to generate the editor.

There was no time logging for the participants of both projects. Thus, all presented time efforts are rough estimations.

The comparison applies only to the narrow domain of graphical editors for software models – therefore the external validity of this comparison is limited to this domain. Although this section lists several limitations, one can assume that model-driven development would bring more efficiency to other graphical editor projects, too.

5 Related Work

Currently, there is a lack of case studies on model-driven development that compare the *same* software product that is once developed manually and once model-driven. Most

available studies focus on a migration from conventional to model-driven development. Such a study is available with [Sta06] where the migration processes of two companies that are about to introduce model-driven development are evaluated. Additionally, a previous study evaluating the applicability of MDD to different kinds of projects is included. A large case study in terms of size and duration is available in [BLW05]. This study points out the experience of 15 years of MDD at Motorola, systematically handling success and failure issues for MDD. Issues like abstraction levels, semantic problems, tool support, influence of personnel and scalability are investigated. Schmidt [Sch06] gives a broader overview on the latest MDD approaches, that emphasizes the improvements by new tool support.

6 Results

We have presented and compared two projects aiming at developing an implementation of the PCM and the Palladio Editor. It has been shown that developing functionally comparable software took about nine-fold more time if the source code was written by hand, compared to model-driven development. Although there are limitations of the presented comparison, the enormous win of efficiency and productivity in model-driven development of the graphical editor shows a reasonable domain where model-driven development is applicable.

The above description of both projects indicates that the successful deployment of model-driven approaches has to be supported by two pre-requisites: (a) the domain has to be well understood (which was the case due to the knowledge on the requirements and the architecture) and (b) the development has to be supported by powerful libraries and frameworks (in our case EMF, GEF, and GMF).

Depending on the tool support of a domain, the time proportions of writing code and doing meta-modelling change significantly. The existence of good templates is essential for productivity as shown in the comparison. Another factor that influences the ratio of hand-written code is the degree of customizations: Customizations that are not supported by templates still need hand-written code.

For further research, the empirical investigation on the role of the influencing factors on the success of model-driven development is required. In addition, the relation between generators and transformations on the one hand and libraries and frameworks on the other hand and their interplay has to be researched. There are some more challenges that come with the use of model-driven development, raised by this comparison. What happens to model instances after the evolution of a meta-model? How to manage the mix of generated and hand-written code after major refactorings of meta-models? So far, there is no support for (meta-)model maintenance. Future work will deal with these challenges.

References

[BKR07] Steffen Becker, Heiko Koziolek, and Ralf Reussner. Model-based Performance Prediction with the Palladio Component Model. In *Proceedings of the 6th International Workshop on Software and Performance (WOSP2007)*. ACM Sigsoft, February5–8 2007.

[BLW05] Paul Baker, Shiou Loh, and Frank Weil. Model-driven engineering in a large industrial context - Motorola case study. In *Model Driven Engineering Languages and Systems*, volume 3713/2005 of *Lecture Notes in Computer Science*, pages 476–491, Springer Berlin / Heidelberg, 2005.

[Cor] Microsoft Corp. The COM homepage. http://www.microsoft.com/com/, last retrieved 2006-10-30.

[Ecl06] Eclipse. The Eclipse Modelling Project, 2006. http://www.eclipse.org/modeling/, last retrieved 2006-10-30.

[Kro06] Klaus Krogmann. Entwicklung und Transformation eines EMF-Modells des Palladio Komponenten-Meta-Modells. Master's thesis, University of Oldenburg, Germany, May 2006.

[Obj05] Object Management Group (OMG). Unified Modeling Language Specification: Version 2, Revised Final Adopted Specification (ptc/05-07-04), 2005.

[PGR05] Project Group Ride.NET. Roundtrip Engineering Entwicklungsumgebung. Technical report, University of Oldenburg, Germany, http://ride.net.ms, 2005.

[Pro] The Netron Project. Netron Graph Library. http://netron.sourceforge.net/, last retrieved 2006-10-30.

[Sch06] Douglas C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *Computer*, 39(2):25–31, 2006.

[Sta06] Miroslaw Staron. Adopting Model Driven Software Development in Industry - A Case Study at Two Companies. In *MoDELS*, pages 57–72, 2006.

[Ufl05] Matthias Uflacker. Design of an Editor for the model-driven Construction of Component Based Software Architectures. Master's thesis, University of Oldenburg, Germany, January 2005.