Transitioning from code-centric to model-driven industrial projects – empirical studies in industry and academia

Miroslaw Staron, IT University of Göteborg, 412 96 Göteborg, Sweden, miroslaw.staron@ituniv.se

Abstract

Introducing Model Driven Software Development (MDSD) into industrial projects is rarely done as a "green field" development. The usual path is to make a transition from code-centric (CC) development in existing projects into MDSD in a step-wise manner. Similarly to all other software development activities, software quality assurance needs to be adjusted to meet the new challenges arising when using models instead of the code for the mainstream development. In this chapter we present a set of empirical data on the issues related to transitioning from CC to MDSD projects in industry. First, we present results from a set of experiments evaluating how a domain specific notation affects the effectiveness and efficiency of reading techniques used for inspecting models. Second, we present a comparison of productivity increase when changing to MDSD projects from one of the large Swedish companies. Finally we present a short survey on the prioritization of products, projects, and resource metrics in MDSD projects.

Keywords: UML, DSL, inspections, experiment, case study, industrial adoption, productivity

1. Introduction

Introduction of new development paradigms and technologies is never a simple task. It is even harder when we consider large software development organizations with a long history of using other methods and with a portfolio of long-lasting software products. The long-term nature of these projects coupled with their continual development requires stable and reliable development methods. In contrast, the global economy with its competition drive companies to seek out and adopt new methods and tools to improve productivity and enhance their competitive position with innovative products of higher quality and rapid development cycles. Using modeling in software development promises improved quality and productivity through increased automation of the software development process.

Model Driven Software Development (MDSD) comes in many flavors – starting from using general-purpose modeling languages such as UML (Unified Modeling Language, (Object Management Group, 2004)), and ending with a set of integrated Domain Specific Modeling Languages (DSLs). The main characteristic of MDSD projects, regardless of the modeling notation used is that models play the central role in the process. Models are used for code generation, but also for early quality assessment activities (e.g. software inspections, testing executable models), or for estimations.

This chapter addresses the problem of providing empirical evidence on how much improvements could be expected in the first projects conducted according to the principles of MDSD. It also addresses the issue of which aspects should a project manager consider when undertaking the first projects in MDSD, and which metrics should be customized for MDSD already for the first project.

In order to address the problem we analyze a set of empirical studies performed both in industry (case studies at Ericsson) and in academia (experiment with software inspections). By providing empirical evidences and experiences from industry we support managers of future software projects in making informed decisions concerning adoption of MDSD.

The chapter presents experiences of improvements brought by model-driven development in industrial projects and the expected increase of effectiveness of software inspection of models elicited through experiments.

The chapter is structured as follows. Section 2 presents the background for the claims presented in the chapter, outlines the existing problems in detail and overviews the existing literature in the area. Section 3 is the core of the chapter and presents the empirical studies, in the end discussing their validity. Section 4 presents a short meta-analysis of the series of studies presented in Section 3. Section 5 contains conclusions. The chapter concludes with a section on future research directions related to using reading techniques as a quality assurance technique for models, and research in productivity assessment in MDSD projects.

2. Background

Based on the roadmap for research on MDSD (France & Rumpe, 2007) it shows that MDSD is not yet a fully established technology and it will still evolve. Therefore, an issue could be raised whether it is mature enough to be adopted or whether it delivers on its promises. The main challenge in the industrial adoption of MDSD is that MDSD needs investments to be effective: the larger the investments, the larger the benefits. In large software projects and in large companies the adoption of MDSD is burdened with all the problems of immature technology (how to justify real expenses based on promises?) and organizational resistance (how do we know that the technology actually improves our way of working?). Herein lies a challenge – how to gradually build up the confidence that using models in a project can help to increase productivity (or quality, or ideally – both). As we are able to show in the case study at Ericsson in Section 3.3, in addition to investing in technology, the investments should also contain costs of coaching (making sure that modeling knowledge is in place), model migration, or gradual migration process.

Transitioning of software practices from document and code centric into model driven can take several years, which is shown in a recent study from Motorola (Baker, Loh, & Well, 2005). The length of time depends on the size of the

organization and the range of the products of the company. The long time span of the adoption activity needs to take into account the fact that technology changes during that time. This fact also means that the criteria for deciding whether to early adopt MDSD in industry are not the same as the decision criteria for the projects adopting MDSD a while later. The interpretation of the results from this study could indicate that there are several *flavors* of MDSD at large companies:

- using UML as the core modeling language of MDSD, and
- using Domain Specific Modeling Languages (DSLs) as the core languages of MDSD.

As it is currently observed, the first flavor is more popular. Therefore the "UML flavor" of MDSD forms the context of this chapter.

In this chapter we consider UML as the core modeling language in MDSD as all the presented studies use UML (both in the experiment and in industry). The studies presented here are based on the view of MDSD as a process of creating a sequence of models in a semi-automated way. The automation is achieved through the use of model transformations, which can be programs that transform one model into another or make updates to the same model. The process is semi-automated since not all model transformations can be automated at the current state of technology. Such a view of MDSD can be presented in Figure 1 and it is adopted from one of the pioneer companies introducing MDSD into their processes (Staron, Kuzniarz, & Wallin, 2004a).

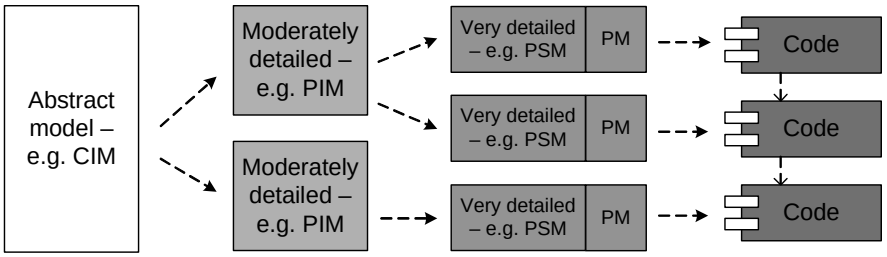

Figure 1. Models in MDSD in the studied organizations

The process of using models (which should be inherent in the product development process) is realized by Model Driven Architecture (Mellor, Kendall, Uhl, & Weise, 2002; Miller & Mukerji, 2003). MDA realization of MDSD recognizes four kinds of models: Computation Independent Models (CIM), Platform Independent Models (PIM), Platform Specific Models (PSM), and Platform Models (PM). The models, expressed in UML, are used sequentially, as shown in Figure 1. The models differ in the abstraction levels and purposes. The horizontal and diagonal lines represent transformations; the

transformations can be manual and automated[1]. The vertical lines in the right-hand side of the figure represent dependencies between code modules. This approach to MDSD can be referred to as the *generative approach* since new models are created from other, more abstract models, the models are used to generate the code and the code is then compiled. An alternative approach is the executable approach where the models are executed and verified – the code is embedded in the models (Mellor & Balcer, 2002; Starr, 2002). The transformations can themselves be expressed as models thus creating a set of interrelated models – called mega-models (Bézivin, 2005; Bézivin, Jouault, Rosenthal, & Valduriez, 2005).

Another flavor of MDSD can be seen as using DSLs as the core modeling notation. In the telecom domain, Jouault et al. (2006) show that this approach needs extra effort for integration of DSLs, which is required as the final product, is usually an embedded application. One of the major differences between DSLs and general-purpose languages (like UML) is the way of integrating models. In the UML case, the integration is easier, as the complete system can be expressed in one model, while in the previous case it is a set of models expressed in different DSLs. The practical problems with integrations of DSLs are that the extra effort is needed to create mechanisms for integration and the semantics of the integrations. Furthermore, Evans et al. (2003) show that creation of MDSD environments is usually a creation of a multitude of languages specific for dedicated purposes. Making these languages subsets of a single language like UML eases the integration and allows early verification and validation of the system (or its model).

The creation of modeling languages requires deep knowledge in the mechanism and techniques used for that purpose – the main one being metamodeling. As Atkinson and Kühne (2002) point out, metamodel creation is an essential part of MDSD and requires the competence of a language engineer. A way of simulating the creation of a brand-new modeling language is customization of an existing one. In the case of UML, stereotypes can be used for that purpose. The use of stereotypes has limitations, but it has also advantages – e.g. less strict requirements for knowledge from the creators of the customization (Staron & Wohlin, 2006).

France and Rumpe (2007) in their roadmap outline research needs in the area of MDSD and thus provide insight into the current challenges of MDSD. They identify 3 categories of challenges:

- Manipulating models – defining the challenges with automation of model transformations, e.g. the need for effective integration of models and increased research into mega-models (i.e. models of models and transformations between them).

[1] Although manual transformations should constitute the minority of all transformations.

- Supporting separation of design concerns – defining the challenges with creating separate views on the same phenomenon and integration of these views, e.g. Aspect Oriented Modeling.
- Modeling language – defining the challenges related to the use of high level modeling languages, e.g. managing language complexity and extensibility, domain specific modeling environments.

France and Rumpe also point out the need for executable models that can help to shrink the gap between problem domain and the solution space. They conclude that at the current stage, MDSD only contributes to the complexity of software and that the technologies of MDSD need more research into being effectively usable in industry.

A study at two Swedish companies willing to adopt MDSD (Staron, 2006) identifies additional challenges with large scale industrialization of model driven development. The outcome of that study indicated that the main challenges are:

- Maturity of modeling technology – indicating that the modeling environments are either restrictive (and simple, not well-suited for the problem at hand), or vast (and difficult, demanding large expertise in defining modeling languages and tool building).
- Maturity of modeling related methods – indicating that project need support in quality management based on models and improving the ways the models are used in the process.
- Process compatibility – indicating that the processes cannot be "revolutionized" by the introduction of models, but rather gradually improve efficiency.
- Core language engineering expertise – indicating that at the current state of the technology the project team needs to understand details behind the construction of a modeling language – e.g. to understand the constraints of the modeling technology.
- Goal-driven adoption process – indicating that MDSD should be adopted gradually aligned with elevating the competence of the team.

In this chapter we focus on providing empirical evidences on how much improvements one could expect from effective and efficient use of models. First we present a survey of a focus group, which results in identifying that process automation, modeling knowledge, and model based quality assurance are the most important elements which the group would see solved.

3. Issues and solutions in adopting MDSD in the initial projects

In this section we present a set of issues and controversies to address while transitioning to MDSD in large software organizations/projects. These issues are:

- How much can quality assurance benefit if domain-specific modeling notations are used?
- How much productivity improvement can we expect from the first project?
- Which are the most important investments in the first projects in MDSD?

These issues are addressed by proposing solutions which are in the form of results from several case studies and experiments both in academia and industry. Each study has a described background, motivation, outline of the design, and the results.

3.1. How much can QA benefit if domain-specific modeling notations are used?

From the perspective of quality assurance, MDSD promises increased quality of products, at the same time promising increased productivity. In order to verify these promises, we performed a series of experiments with domain specific notations and reading techniques. In the initial experiments we evaluated whether a domain specific notation, simulated by UML stereotypes, increases the level of understanding of models in comparison with the standard UML models (L. Kuzniarz, Staron, & Wohlin, 2004; Staron, Kuzniarz, & Wohlin, 2004, 2006). The outcome of the previous experiments was that the domain specific notation increased the understanding by up to 131% (the correctness of designs evaluated at Volvo IT). In the next experiment we evaluated whether a similar domain specific notation increased the effectiveness and/or efficiency of reading techniques, which are presented in this section. The experiment presented here is an extension of the experiment presented in (Staron, Kuzniarz, & Thurn, 2005).

The motivation behind this experiment was to evaluate whether a domain specific notation helps in increasing quality of models when structured reading techniques are used. We intended to check how much improvement in quality (correctness) one can expect when migrating from standard UML to domain specific notations. The characteristics of the study are as follows:

- Type: controlled experiment
- Treatments:
    - domain specific notation (simulated with UML stereotypes) and general purpose notation (UML)

- Sampling: Randomized Control Trial using blocking
- Analysis: Statistics: Shapiro-Wilk test for normality, paired t-test (or Wilcoxon depending on the results of Shapiro-Wilk)
- Results: Effectiveness is higher for a domain-specific notation than the general purpose notation; efficiency is the same

3.1.1. UML stereotypes and reading techniques

As defined in the UML specification documents (Object Management Group, 2003), the main idea behind using stereotypes is to introduce new semantics to the existing model elements. The UML definition of stereotypes involves the definitions of other extension mechanisms – tagged values and constraints (c.f. (Gogolla & Henderson-Sellers, 2002; Ludwik Kuzniarz & Staron, 2002)). Stereotypes allow extending the language in a way, which is consistent with the definition of the language and they are useful in automatic model transformations, like for example code generation for a specific purpose (e.g. (Uhl & Lichter, 2002)).

In addition to the above, there is also another way of perceiving stereotypes – the original intention of introducing the notion of stereotypes. The stereotypes can provide a secondary classification of model elements. This concept was initially introduced in (Rebecca Wirfs-Brock, Wilkerson, & Wiener, 1994). Such stereotypes provide a means of expressing some classification of the stereotyped model elements, adding properties, which cannot be defined for all model elements of the same kind, but only for some. These stereotypes can be classified as *transitive stereotypes* (according to the classification presented in (C. Atkinson, Kühne, & Henderson-Sellers, 2002)), because they are added to classifiers on the model level, but should also be recognized on the instance level. They are useful as a secondary classification mechanism (R. Wirfs-Brock, 1993) since they both brand the classifier and its instances with additional meaning. An example of a transitive stereotype is presented in Figure 2.

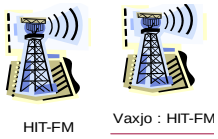

Figure 2. Example of a UML stereotype presented using a graphical notation

The figure presents two stereotyped elements – a class which is also a sender station and its instance – a particular sender in a city in Sweden.

Other important elements in the experiment design are the reading techniques. Different reading techniques are used to examine the artifacts during software inspections and to find errors. In the investigation presented in this paper, we

use two specific reading techniques – checklist-based reading (CBR, (Fagan, 1976)) and perspective-based reading (PBR, (Basili et al., 1996)) and an unstructured reading (further referred to as the ad-hoc technique).

In the context of software inspections, the reading techniques are only a part of the whole process. Usually, the complete process consists of planning, overview, preparation, meeting, rework and follow-up. The details of all steps in the inspection process can be found in (Fagan).

Checklist based reading (CBR) is a reading technique in which the reader is given a checklist with specific kind of faults to look for. The items in the checklist can be expressed as questions or as statements. In particular, the checklist contains items that help in finding logical errors in the inspected documents – errors that cannot be verified in an automatic way by a modeling tool (in the case of UML models).

Perspective based reading (PBR) is a reading technique in which artifacts are examined from certain perspectives. Each perspective is intended to provide a different way of examining the document. Using different perspectives allow focusing on various aspects of the document (for example user's or designer's perspective). One of the assumptions of PBR is that the reader can better identify faults if he/she works in a structured manner. The PBR is a special kind of scenario-based reading techniques (Porter, Votta, & Basili, 1995).

The third kind of reading can be characterized as ad hoc reading. It denotes a technique which provides no guidelines and implies that the readers use their personal experience to find faults. Only a general description of the task is provided as part of the instructions for this reading technique.

3.1.2. Outline of experiment design

The goal of the experiment was to evaluate the effect of domain specific notation on the effectiveness and efficiency of reading techniques in software inspections. The reading techniques used in the experiment were the most widely adopted techniques – checklist-based reading (CBR), perspective-based reading (PBR), and unstructured reading.

The hypotheses in the experiment were:

$H_{0\text{-effectiveness}}$: Introducing stereotypes does not influence the effectiveness of finding faults by subjects

$H_{1\text{-effectiveness}}$: Introducing stereotypes influences the effectiveness of finding faults by subjects

$H_{0\text{-efficiency}}$: Introducing stereotypes does not influence the efficiency of finding faults by subjects

$H_{1\text{-efficiency}}$: Introducing stereotypes influences the efficiency of finding faults by subjects

The derived variables, effectiveness and efficiency, are calculated from the direct variables – time (T), number of faults found (FF), and total number of faults in the design (TF), in the following way:

$$effectiveness = \frac{FF}{TF} \text{ and } efficiency = \frac{FF}{T}$$

The hypotheses are tested using the paired t-test and Wilcoxon (as efficiency was found non-normally distributed).

The experiment was done as a paired comparison design. The participants were divided into two groups (A and B). After the analysis between these two groups we observe the mean values for each reading technique, which are compared between the groups. However, due to the number of subjects (35) we did not use reading techniques as a factor level which would result in non-significant results caused not by the lack of effect, but by the insufficient number of subjects/data points.

3.1.3. Results

The basic descriptive statistics for the efficiency are presented in Table 1.

Factor level	Mean	Std. Deviation	Percentage
Domain specific (DS)	0.44	0.33	98%
General (G)	0.45	0.39	100%
Difference: DS-G	-0.01	0.45	2% = 0.01/0.45

Table 1. Descriptive statistics for efficiency

The descriptive statistics indicate that there is a small difference between the mean values of notations. The Shapiro-Wilk test for normality does not allow rejecting the assumption of the data being normally distributed with significance level of 0.322. Therefore the parametric paired t-test is used for testing of hypothesis $H_{0\text{-efficiency}}$. The paired t-test does not allow rejecting the null hypothesis as the significance level was 0.202. Thus the observed difference in efficiency is not statistically significant. This in consequence means that the introduction of stereotypes does not influence the efficiency of the reading techniques.

The basic descriptive statistics for the effectiveness are presented in Table 2.

Factor level	Mean	Std. Deviation	Percentage
Domain specific (DS)	0.63	0.20	129%
General (G)	0.49	0.20	100%
Difference: DS-G	0.14	0.20	29% = 0.14/0.49

Table 2. Descriptive statistics for effectiveness

The descriptive statistics shows that using stereotypes in models resulted in an increase of effectiveness by 0.14 (relatively by 29%). The Shapiro-Wilk test for normality does not allow rejecting the assumption of the data being normally distributed with the significance level of 0.020; Wilcoxon is used for testing of $H_{0\text{-effectiveness}}$. After running this test the null hypothesis can be rejected with the significance level of 0.0003. This means that the use of domain specific notation improves the effectiveness of reading techniques.

In order to investigate which of the studied reading techniques was affected most by introducing stereotypes, we perform an analysis of the effect of introducing stereotypes for each method. The analysis is done only with descriptive statistics due to the small number of data points for each reading technique. The mean values for the effectiveness by reading technique are presented in Figure 3.

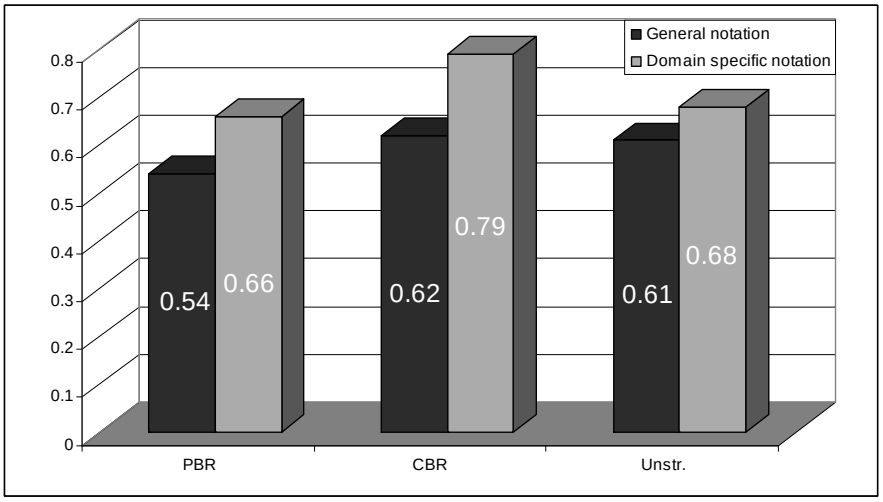

Figure 3. Summary of differences in effectiveness by reading techniques

The descriptive statistics indicate that the outcome of all reading techniques has been positively influenced, in terms of effectiveness, by the introduction of stereotypes. It seems that the most effective technique for using stereotypes is CBR which resulted in finding 79% of faults in design documents.

Since the checklists used in the experiment were general purpose checklists, we expect that using dedicated checklists would further improve these results – c.f. (Laitenberger, Atkinson, Schlich, & Emam, 2000). The fact that CBR was the most effective technique indicates that the checklists are a very useful help in the review process and provide the most structured reading when examining the documents.

The fact that the unstructured reading was better than PBR seems to be counter-intuitive. It could be caused by the fact that the perspectives might have actually mislead the subjects and let them focus on aspects which were not important in the experiment, while the unstructured reading stimulated the respondents to more active thinking and more thorough examining.

The observed improvements in the effectiveness of reading techniques show results from an academic experiment. Although the experiment was not replicated in industry, we still believe the results in industry would be stronger. This belief is based on our previous experiments where the use of a domain specific notation caused much stronger effect in industry than in academia (c.f. Figure 4) when it comes to correctness of understanding the design as presented by Staron et al. (2006).

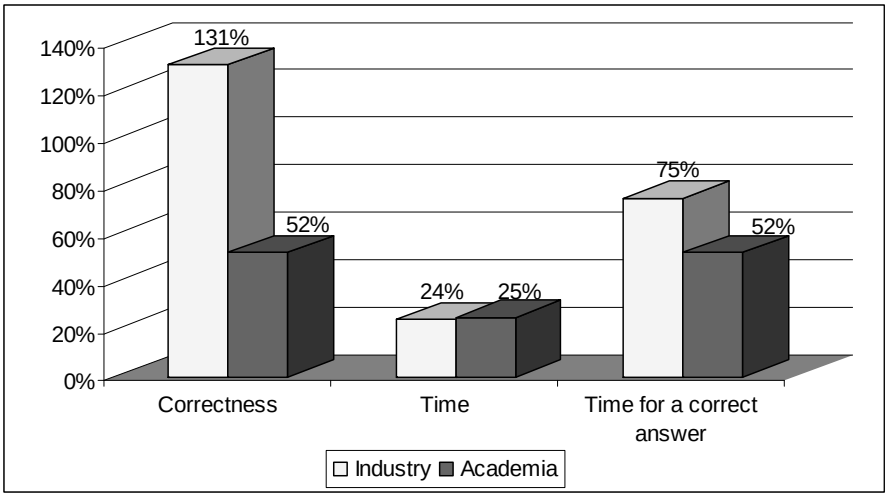

Figure 4. Differences between the improvements of industry professionals and university students in a series of experiments with a domain specific notation

These results show that the transitioning from standard modeling notations to more advanced notations, which are closer to the problem domain than the solution space leads to increased effectiveness of fault finding techniques. This in turn leads to increased quality of the products, as faults are found earlier in the development process. Although this is not an exhaustive study on quality, it shows what kind of improvements can be expected in an initial project adopting MDSD in terms of quality increase. The limitation of this study is its academic context, which was dictated by the need to obtain statistical power when it comes to results. The materials in the study were based on the materials from our industrial partners to ensure that the context of the experiments were as

close to reality as possible, at the same time retaining controllability over factors. The complementary aspect to quality – productivity – would have a limited use if studied in the same manner. Therefore, we studied an industrial project at another industrial partner – Ericsson – in order to address the issue of expected productivity increase from the first project.

3.2. How much productivity improvement can we expect from the first project?

One of the crucial aspects in adopting a new technology is the issue of productivity and quality improvement after adoption. The project and product managers are eager to observe the improvement already in the first project. The increase, however, comes with a price. The first project needs to be given a high degree of freedom in adjusting the company processes to achieve measurable improvement in productivity and quality. The project described in this section is not a special case, but rather a representative situation with respect to controllability and conformance to standard company process description. This was our assumption which we checked in the study presented in Section 3.3. The same situation was observed in the first advanced MDSD project at Volvo IT (Staron, Kuzniarz, & Wallin, 2004b).

This study can be characterized as follows:

- Type: case study
- Sampling: convenience sampling (we used the most suitable project at the studied organization)
- Data collection: artifacts analysis, interviews
- Analysis: descriptive statistics
- Results: show that the MDSD project was 39.5% more efficient than a sister CC project

3.2.1. Outline of the case study design

In order to assess the degree of initial productivity increase in the first MDSD project, we compared two similar projects run at Ericsson: the MDSD project and a sister code-centric (CC) project. The sister project used in the comparison was an old version of a similar technology[2]. The same platform was used, although a different approach was used to deploy the software in this platform. The positioning of the projects is shown in Figure 5.

[2] Naturally, due to the sensitivity of the data presented in this paper we cannot give details about the products.

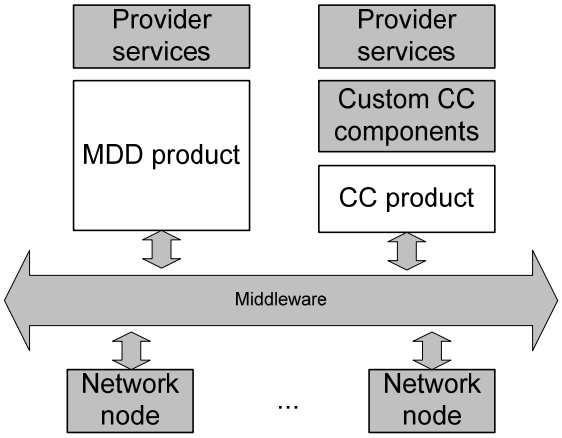

Figure 5. MDSD project and the sister project – position in the architecture of the telecom systems

Both products operate above middleware that mediate communication with network nodes. Both products are providing a way of configuring the nodes according to the specifications of provider services. The CC project requires more configuration and development of custom components which mediate between the CC product-specific messaging and the provider-specific messaging. The MDSD product is intended to improve that and provide more flexible and adaptable environment where the creation and deployment of new services for providers is more efficient, faster, and by that much cheaper. Both projects were done in an iterative way and in the comparison we used the data for the completed projects. However, since the MDSD project was in progress (it was just after the 1st iteration) for that project we used the actual data from the 1st iteration and the updated estimations for the coming iterations.

3.2.2. Results

The effort distribution per phase (the sum for all iterations) is presented in Figure 6. It should be noted that the effort for analysis and design could not be distinguished in the model-driven project. The term analysis did not mean the same thing in the CC and MDSD projects, what was called analysis in the CC project was included in the design part of it. This could have been caused by the fact that MDSD was adopted in this project.

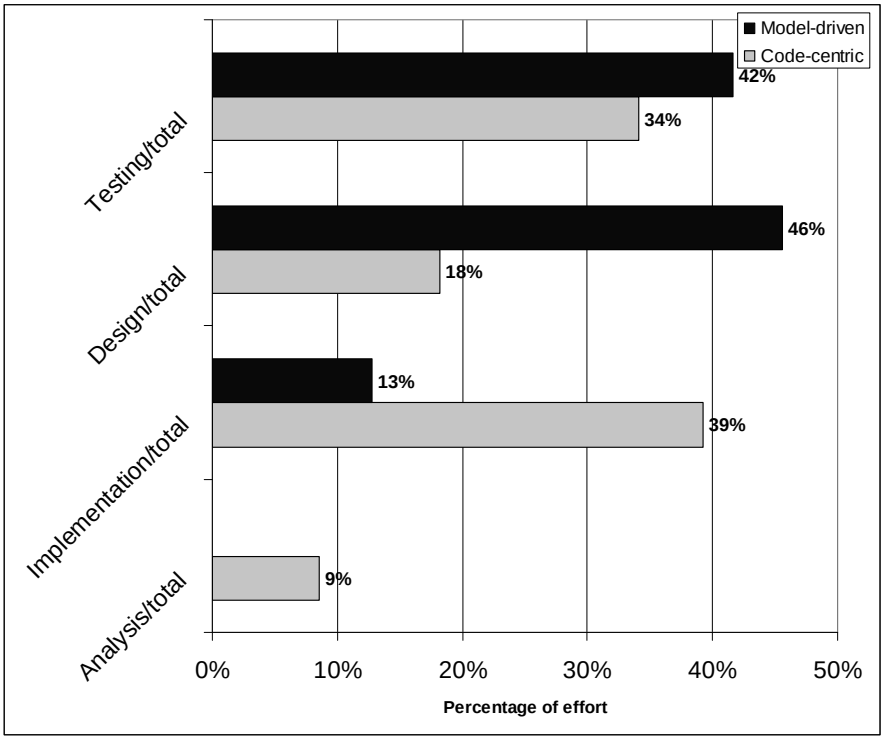

Figure 6. Effort distribution for code-centric and model-driven projects

The figure shows that there is a difference between the effort distribution between the MDSD and CC projects. The MDSD project spends almost twice as much effort for designing as the CC project. It should be noted that in the case of the CC project the design was done using textual specifications and code fragments illustrating important design decisions. It should also be noted that the implementation effort in the MDSD project was much smaller than for the CC project. In the MDSD project the implementation was intended to fill in the code which cannot be generated automatically from models. This is due to the fact that the standard UML with some basic profile support is used in the project. Nevertheless, the long-term goal for subsequent MDSD projects is to replace repetitive manual coding tasks by automated transformations. The resources released in this way could be used to develop new transformations and to focus on modeling of the core business functionality in the project.

Another important aspect was the effort per unit of size for both projects. For these two projects we chose the functionality of the product to be the determinant of size as the size cannot be measured uniformly in both projects (size of models vs. size of source code). We used an internal metric for the

functionality (which cannot be given together with the data on productivity due to the confidentiality agreement with the industrial partner). The results for both projects are presented in Figure 7. The data has been transformed as the real data is sensitive to the company, although after the transformation the proportions are still the same.

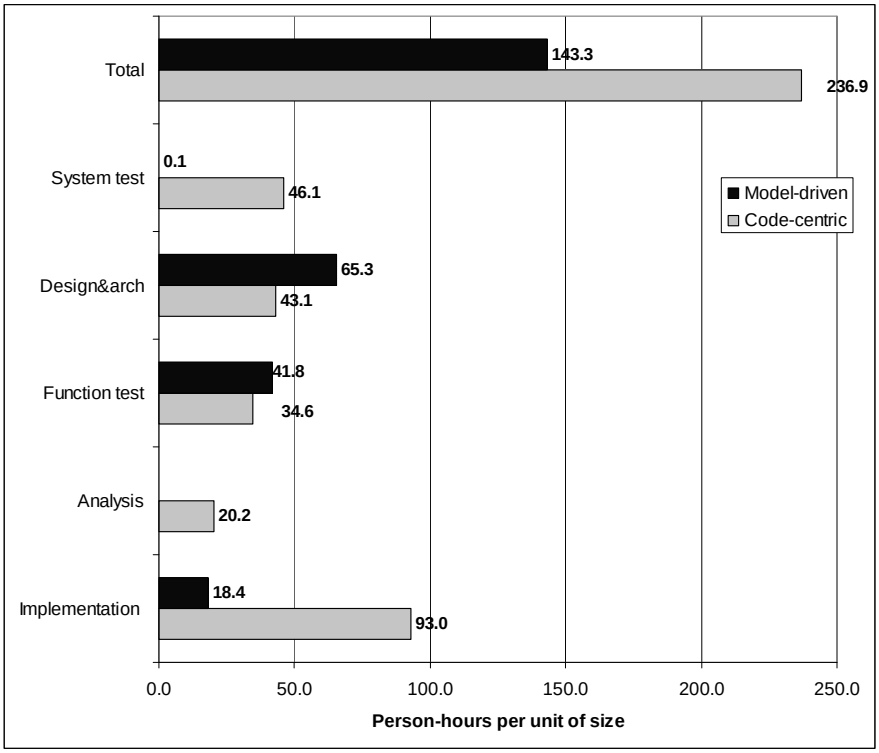

Figure 7. Effort per unit of size for code-centric and model-driven projects

The value of the total effort per unit of size shows that using models provides the means of decreasing the effort by 39.5%, which is a considerable value. Not surprisingly the most significant gains in efficiency are achieved in the implementation phase – 66.7% decrease in effort. Another interesting aspect is the reduction of effort for system testing and concurrent increase in the effort for function testing. This is caused by the fact that this first MDSD project expects to have problems with the software caused by the introduction of new paradigm and thus there is a need for compensating for that by increasing the effort for function testing (which is included in the planning). The initial productivity improvement seems promising and it does not require advanced tool or language customizations, which require significant effort (Staron & Wohlin, 2006). Larger benefits, however, require significant additional effort in

customization of the modeling environment, in particular to automate the process by the use of model transformations. The development of such model transformations needs to be carefully planned and introduced into the projects gradually.

3.3. Which are the most important investments in the first projects in MDSD?

One of the main issues in adopting MDSD is which elements of the project should be addressed in the first place when migrating to MDSD projects. In particular we were interested in the will to invest in developing (or customizing existing) metrics for MDSD projects and artifacts; ISO/IEC 9126 (described in section 3.1.1) was used as the reference standard for this purpose. To obtain an empirical data on the investments, we provide data from a survey among 20 experts in the focus group of researchers (4) and practitioners (16) working with the adoption of MDSD (or with MDSD that is already adopted) at their companies: Ericsson, Motorola, and others. The prioritization technique ($100 technique) was used to prioritize particular issues. We asked the experts a series of questions about:

- prioritization of measurements defined in the ISO/IEC 9126 standards,
- prioritization of quality characteristics of the ISO/IEC 9126 standards,
- prioritization of potential improvements in the first MDSD projects, and
- the use of models in their work.

Finally we interviewed a project manager while he was deciding whether to adopt MDSD in his project. Our goal was to obtain qualitative data and his perception of the investments. The manager had several years of experience as the project manager and engineer, including model driven software development.

The motivation behind this study was to investigate which measurements are most important for MDSD projects and to investigate the context of making the decision about migration to MDSD. The context consists of the decision criteria, as well as the required initial investments.

This study can be briefly characterized as follows:

- Type: case study
- Sampling:
    - Survey: Randomized Control Trial; population: project managers, quality managers, design engineers, and architects working with MDSD projects
    - Migration project: Convenience sampling; population: project managers of mid-size sub-projects that are migrating from code-centric to MDSD
- Analysis: descriptive statistics

- Results: show that process and resource metrics are the most important metrics in MDSD projects; modeling knowledge and process automation are key aspects in MDSD projects; and the most important quality characteristics are functionality and maintainability

3.1.1. ISO/IEC 9126 standard

One of the most widely adopted quality standards which includes the definition of software measurements meant to measure quality is the ISO/IEC 9126 standard (International Standard Organization & Commission, 2001). The standard defines the following quality perspectives (also called approaches to quality in the standard), with the associated types of metrics:

- Process quality: defines the quality of software processes followed during software development
- Internal quality: defines the details of software product quality that can be improved during code implementation, reviewing and testing,
- External quality: defines the quality when the software is executed, which is typically measured and evaluated while testing in a simulated environment,
- Quality in use: defines the quality of software product as perceived by the users

The perspectives are further divided into quality characteristics, which are further associated with specific metrics. Each quality characteristics has several metrics associated with it and the ISO/IEC 9126 has an example set of metrics. The metrics in the standard, however, are not dedicated for models, but for measuring code-based or document-based artifacts. Therefore, there is a need to develop (or customize the existing) metrics to reflect model driven software development.

The standard defines the following internal and external quality characteristics (the characteristics are defined for internal and external quality together – definitions are quoted after the standard):

- Functionality: the capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.
- Reliability: the capability of the software product to maintain a specified level of performance when used under specified conditions.
- Usability: the capability of the software product to be understood learned, used, and attractive to the used, when used under specified conditions.

- Efficiency: the capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions.
- Maintainability: the capability of the software product to be modified.
- Portability: the capability of the software product to be transferred from one environment to another.

These characteristics were used during the study presented in this section.

3.1.2. Outline of the case study design

The first part of the study (survey on measurements) presented in this chapter was performed during a focus group meeting at the workshop on quality in modeling at the MODELS conference and at Ericsson in Sweden. The focus group consisted of architects, researchers, managers, and design engineers, who have experience in the field. The sampling technique was Randomized Control Trial as we have randomly chosen participants and not the whole group of experts.

The second part of the study (migration issues) was performed at Ericsson, by interviewing a project manager who was involved in making the decision whether the project should adopt MDSD and how the adoption should be done. The sampling was a convenience sampling as we only looked for the appropriate managers at Ericsson, our industrial partner, and no other company in the region.

3.1.3. Prioritization of measurements in ISO/IEC 9126

The first question asked to the respondents was which of the measurements defined in the ISO/IEC 9126 they would see as most important – i.e. in which quality perspective (and the types of metrics associated with them) they were willing to invest and how much if they were to develop new measurements. Their rationale was that if the experts were to be part of the first MDSD project in their organization, which measurements they would need most to be able to ensure controllability of their work (which is different depending on the role – quality manager, project manager, architect, consultant, researcher, and designer). Figure 8 presents the average of the answers from the experts in the focus group.

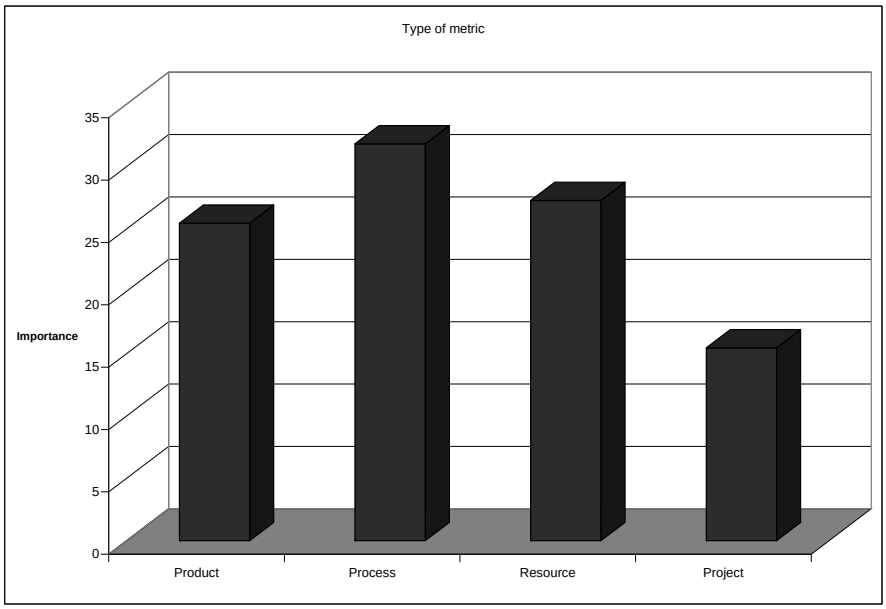

Figure 8. Prioritization of types of metrics from ISO/IEC 9126

The focus group prioritized the process metrics as the most important type of metrics although the product and resource metrics were not much less important. This indicates that in the first MDSD project, a strong focus should be put into having precise tools for collecting process metrics – e.g. efficiency of specific phase or effectiveness of the process of finding defects in models.

The resource metrics are prioritized quite high which shows that the results come from managers in a company who are very concerned by the costs of their project. This, in turn, is caused by the tight market in which the company has to operate, where the cost has a key role in success.

The project metrics are not highly prioritized as the way of working is potentially not altered to a large extent in the first project (since it is a transitioning from standard code-centric projects). Since MDSD changes the process of developing software, there is no doubt that the associated metrics must be changed as well. Productivity cannot be measured as size of the code produced per time unit, but rather as the size of model per time unit. The size of the model, however, needs to be specific for the phase (e.g. number of classes in high-level design, while the number of states in the detailed design phase). The size metrics, nevertheless, are specific for the modeling notation used and the process followed.

The process metrics are important for ensuring that MDSD actually delivers in terms of productivity or increased efficiency and effectiveness of software

processes. One should not, nevertheless, forget that the quality of the product that can be affected by adopting a new development technology.

3.1.4. Prioritization of quality characteristics

The experts in the focus group prioritized the quality characteristics of good software from ISO/IEC 9126. They were asked how much they would be willing to invest to improve each characteristic of the software. The results are presented in Figure 9.

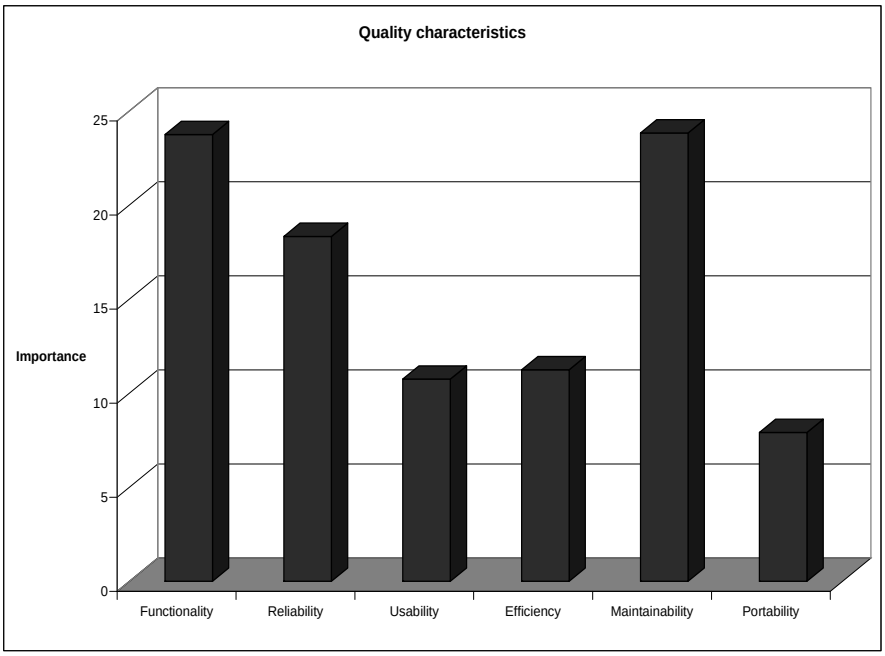

Figure 9. Prioritization of ISO/IEC 9126 quality characteristics

The results show that the experts were still willing to prioritize the functionality and the maintainability of the product as top quality characteristics. The least important characteristic was portability. This is rather surprising since MDSD promises increased portability through exchangeable code generators and pluggable platform models.

3.1.5. Prioritization of improvements in the first project

The experts were also asked which improvements they expect to see in the first MDSD project, caused by introducing MDSD. The results are presented in Figure 10.

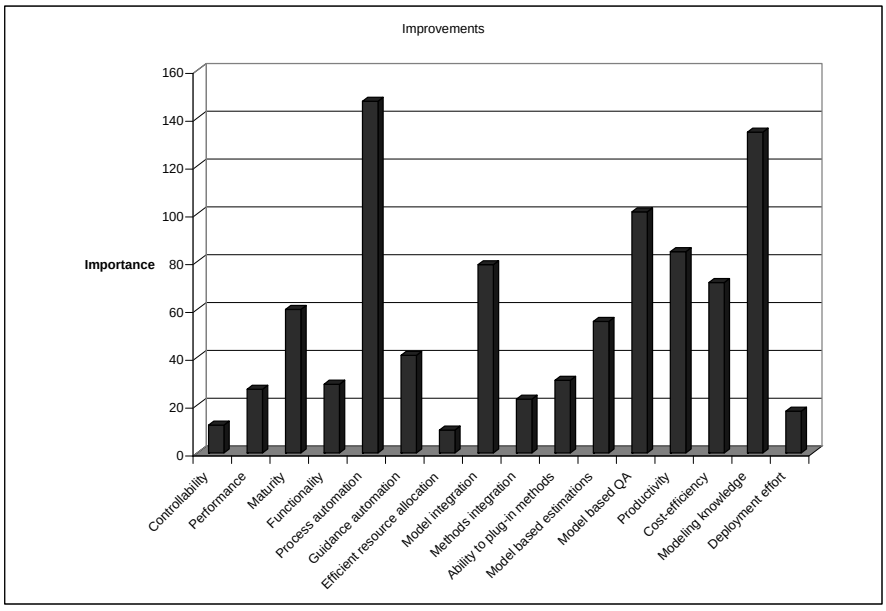

Figure 10. Prioritization of potential improvements in projects

The results show that the top three expected improvements are:

- Process automation – which includes automating tedious tasks – e.g. writing very similar code several times in the same project.
- Modeling knowledge – which includes the knowledge how to use abstractions effectively in software projects.
- Model based quality assurance – which includes using inspections of models rather than text documents to increase the effectiveness and efficiency of quality assurance of early stages of project artifacts.

The process automation should be considered in the context of productivity, namely how much productivity improvement we can expect in the first project by using automated code generation from design artifacts (as an example of process automation).

3.1.6. Presence of models in experts' work

The final question in the survey with the focus group was aimed to examine the presence of models in various phases of software development. The experts were asked what percentage of artifacts in a particular phase are models. The usage of models in the work of experts varies, and it is shown in Figure 11. The highest use of models is for architectural design – on average 42% of architectural design artifacts are models. The next highest usage is for detailed design with 35% of design artifacts being models.

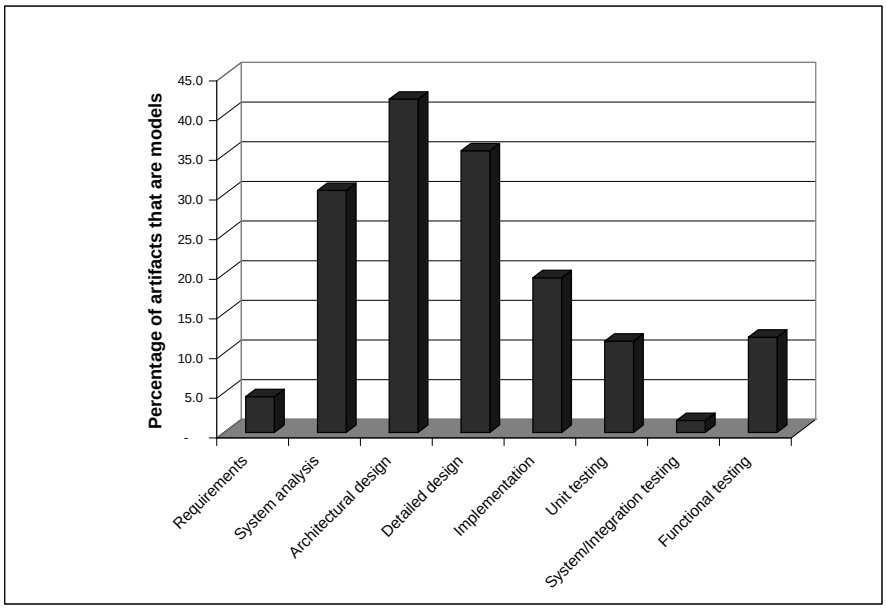

Figure 11. Use of models in the focus group work

The survey with the experts from the focus group provides an overview of the importance of metrics in the first projects. The survey, however, did not provide an insight on how the projects are chosen whether they can be migrated into MDSD projects.

3.1.7. Decision factors in adoption of MDSD

In order to establish such a set of decision criteria, we examined one small project at Ericsson. The project involved the developing of an algorithm used in a component in a mobile network. The size of the project is a few person months[3] and this project has been chosen to be the pilot project supporting the project management team in making a decision on how to proceed with the large project. In our study we identified the following decision criteria:

- **Structure migration**: It is possible to migrate the core model structure (e.g. class diagrams) to the new model in a very cost-effective way (i.e. with rather low effort).
- **Independent co-existence:** It is possible to model the new part/model of the software independently from the legacy part/model (e.g. by developing new sequence diagrams in the new model).
- **Migration effort:** The effort for changing to the new model is low.

[3] Due to the confidentiality agreement we cannot provide the exact numbers.

- **Controlled legacy changes:** It is possible to reference the legacy part, and there is no (or very limited/controlled) need for changing the legacy parts/models.
- **Model longevity:** The "new" *model* will be used for more than one project (e.g. to become product documentation).
- **Controlled initial change:** A limited group of people is going to be affected by the initial change.
- **Knowledge in place:** The modeling knowledge (in the new tool) is in place in the project and is not in the hands of one/two individuals.

Using the $100 technique the project manager prioritized these criteria, which resulted in identifying two levels of criteria as shown in Figure 12.

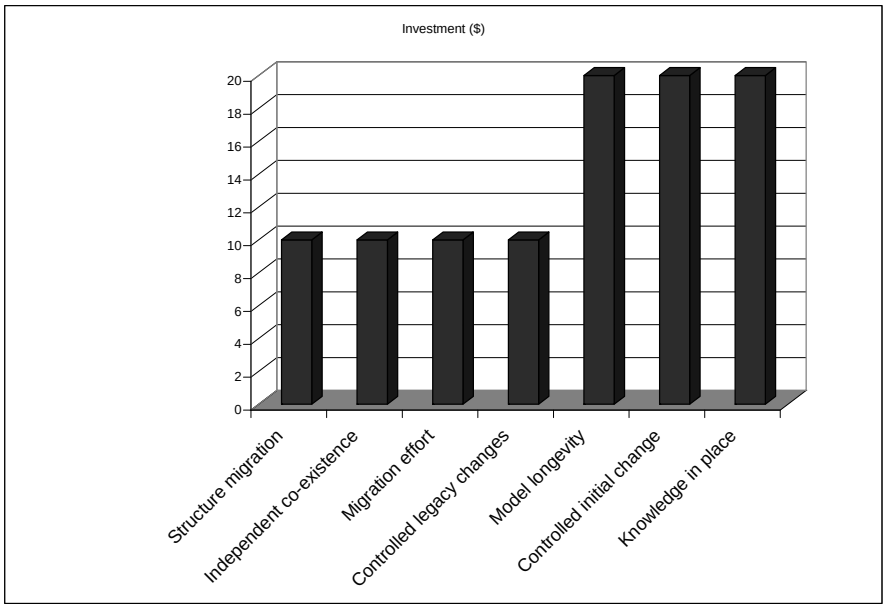

Figure 12. Prioritized criteria for migration to new models

The results show that there are two classes of criteria defined by their importance. The higher prioritized criteria are related to project management. They address the question of what the project manager needs to minimize the risk of failing the migration process already during the first project. The project manager identified also additional issues that are pre-requisites for adopting MDSD from his perspective:

(i) migration process should be longer than the span of a single project; in the studied organization, the migration process could not be automated due to the large legacy code base and the size of the products,

(ii) the initial knowledge gap should be small (unless large investments were envisioned), and

(iii) the old and the new documentation styles (code-centric and model-driven) can co-exist for some time since a lot of knowledge and documentation in large projects needs to be maintained and used in new projects (and there is no possibility of re-doing all documentation during the migration process).

Model migration was less important than model longevity, which could be seen as an unexpected situation. However, it is not the case in the studied organization. The effort of manual can be (and already) is spread over several projects and releases as no automated tools exist which would fulfil the migration purposes of the company.

The views of the project manager bring us to another issue – how to effectively adopt MDSD in industrial context. However, before diving into this issue, let us address another important aspect, directly related to quality assurance.

3.4. Validity evaluation

This chapter presents a series of empirical studies performed both in academia (section 3.1) and in industry (section 3.2 and 3.3). There are some threats to the validity of the results from the studies. In this chapter we use the validity evaluation framework by Wohlin et al (2000).

The main *external validity* threats are related to the case studies. The choice of projects was dictated by their availability. Only the projects which were already using (or just before using) MDSD were chosen in the study (section 3.2 and 3.3 respectively). As we only examined two projects, this poses the threat that the results are not representative. We believe, however, that the results are representative, as they are in line with our other studies, not related to the studies presented in this chapter (Staron, Kuzniarz et al., 2004a; Staron & Wohlin, 2006).

The main *construct validity* threat is related to meta-analysis. The studies presented in this chapter were performed separately, and combined afterwards. Although we designed and performed the studies in order of appearance and using the experiences from the previous studies when designing new ones, we did not initially mean to perform meta-analysis. Therefore, there is a threat that some aspects might have been missed when performing the separate studies. In order to validate this, we performed a workshop (during the presentation of results) at Ericsson during which we presented and discussed our results. We did not miss any points according to the company representatives present during the workshop.

The main *internal validity* threats are different for each study in the chapter:

- Experiment (3.1): the order of presenting the treatments to the subjects could bias the results; to minimize this we performed repeated-measures experiment design with each group having ABBA and BAAB design (Wohlin et al., 2000).
- Productivity case study (3.2): we measured the effort data using the measurements provided by the company; since there are no uniform size metrics for MDSD and CC projects, we had to resolve to high-level metrics in order to be able to compare the productivity. This threat, however, seems to be minimal for the company as the metrics we used are also used at the company to assess project progress and size.
- Survey and migration case study (3.3): we presented the quality characteristics which are used at the company, whereas we could have performed a workshop beforehand and let the respondents decide which quality model is best; we chose that as the company must adhere to adopted standards, which would render our results useless for the company if we did not adopt the ISO/IEC 9126 standard.

Finally, the main *conclusion validity* threat is related to the analysis of the results from case studies. Due to the small sample sizes, the results are very specific and might not reflect the trends in the general population. However, from the previously reported experiences, for example (De Miguel, Jourdan, & Salicki, 2002; Staron, 2006; Vokac & Glattetre, 2005), we find our results in line with the existing empirical evidence.

4. Meta-analysis

The studies presented in Section 3 show that introducing MDSD into software projects provides such benefits as increased quality (correctness) of artifacts and increased productivity. The increase in correctness was shown in the experiment, as this was the most adequate empirical method to provide evidence for this claim (due to sample size and controlled environment). The increase in productivity, however, cannot be assessed through an experiment since the productivity is best measured in a case study.

The above benefits can be considered in a context of costs of introducing MDSD in the first projects. Investments in adjusting methods, metrics, tools, and knowledge of engineers are unavoidable. The study presented in Section 3.3. shows that in the first projects, the most important metrics are process metrics and the most important investments should be put in elevating the knowledge of engineers as well as ensuring longevity of models.

The studies presented in this chapter provide evidence how much improvements MDSD can bring into an organization adopting it.

5. Conclusions

Transitioning from code-centric development into MDSD can be an effort and resource intensive process. In this chapter we outlined two main aspects that are important in the first projects that adopt MDSD in large organizations. The first was how much effectiveness and efficiency improvement we can expect when using a domain specific notation. The results showed that the effectiveness can be improved significantly with constant efficiency of the process. This leads to increased quality of the final product at a constant cost. The other main aspect is the productivity change in the first MDSD project. The industrial case presented in this chapter showed that the first project could improve the productivity by 39.5%. The other two supporting studies show that the group of experts prioritized quality assurance as one of the most important aspects in the first MDSD project.

Future Research Directions

The adoption of MDSD is moving from pilot projects and from small organizations into the phase where large organizations are adopting MDSD for their large, long-term projects. Aiming at the productivity increase, the large companies are pulling the technology forward, demanding advanced methods for working with models. Examples of needs that pull the development of MDSD project practices are configuration management techniques that are suited for models, supporting graphical identification of model differences and supporting model merging similar to code merging. Configuration management practices are necessary if the models are to increase the quality of software products. Ineffective configuration management will surely lead to delays in projects and inefficient verification and validation. This, in turn might lead to lower quality in the final product. Therefore, model-based CM is one of the future research trends within MDSD. The existing solutions, e.g. IBM/Rational Software Architect, support basic configuration management tasks, but fail to help developers in such situations as merging from several branched in a configuration tree. Although this problem also exists in code-based CM, it is easier to predict a result of merging more than two branches than it is when models are concerned.

Future trends in transitioning to MDSD lean towards adoption of DSLs as the core modeling languages. The use and integration of DSLs form the mainstream of the research in the field (France & Rumpe, 2007). Domain specific notations constitute a significant volume of research and several industry-quality tools have been released that support graphical DSLs – examples of these tools include Microsoft DSL toolkit for Visual Studio 2005, and MetaEdit. The interest of software development companies has risen significantly since the release of these tools as the DSL technology is no longer a research playground, but an industrial application. Defining the quality characteristics of domain

specific modeling languages is still an open issue. The standard quality characteristics of the ISO and IEEE standards need to be adapted, as the definition of languages is done at the meta-level (compared to the definition of models of systems).

Another strong trend in MDSD and especially in integration of quality assurance is the introduction of executable models in large software projects. Runtime models (as they are sometimes called) facilitate early verification and validation techniques, but at the same time require skills that are not common at the current software engineering education – working with abstract models and very refined action code. This working at two levels seems to be the main challenge to address in order to increase the quality of executable models.

One future research direction is the creation of methods for defining domain specific checklists when developing domain specific languages. The use of these checklists should further improve the effectiveness of reading techniques. The checklists used in our experiments were general checklist for designs. However, it could be expected that the domain specific checklist, which takes into account design guidelines of the organization, should increase the effectiveness and efficiency of the verification process considerably.

The second research direction is creating model-based project management practices to facilitate making the most out of software projects done in the MDSD way. Together with the research on model-based project metrics (e.g. productivity measurements), the results of research in this direction would be of a great value for project managers.

The third direction is research into effective introduction of MDSD into industrial projects. Industrial adoption needs to progress gradually and companies need support in the process of adopting modeling notations. Some of the challenges that this research should address are: increasing the level of abstraction, ensuring stability of modeling techniques in the company, or continuous professional development of software engineers who finished their education before graphical modeling languages were taught at the universities.

Finally, the most important aspect to address in the transitioning to MDSD is to create a roadmap how the transition should be done at a particular company. Based on the experiences from the current state-of-the-art in MDSD and the existing roadmaps for related areas, e.g. education in engineering roadmap (Shaw, 2000), this roadmap would be of a great value for industry.

References

Atkinson, C., & Kühne, T. (2002). *The Role of Metamodeling in MDA.* Paper presented at the Workshop in Software Model Engineering, Dresden, Germany.

Atkinson, C., Kühne, T., & Henderson-Sellers, B. (2002, 2002). *Stereotypical encounters of the third kind.* Paper presented at the 5th International Conference on the Unified Modeling Language «UML» 2002. Model Engineering, Concepts, and Tools., Dresden, Germany.

Baker, P., Loh, S., & Well, F. (2005). *Model-Driven Engineering in a Large Industrial Context - Motorola Case Study.* Paper presented at the Model Driven Engineering Languages and Systems - MoDELS, Montego Bay, Jamaica.

Basili, V. R., Green, S., Laitenberger, O., Shull, F., Sorumgard, S., & Zelkowitz, M. V. (1996). The empirical investigation of perspective-based reading. *Empirical Software Engineering, 1*(2), 133-164.

Bézivin, J. (2005). On the unification power of models. *Software and Systems Modeling, 4*(3), 171-188.

Bézivin, J., Jouault, F., Rosenthal, P., & Valduriez, P. (2005). *Modeling in the Large and Modeling in the Small.* Paper presented at the European MDA Workshops: Foundations and Applications, MDAFA 2003 and MDAFA 2004.

De Miguel, M., Jourdan, J., & Salicki, S. (2002). *Practical Experiences in the Application of MDA.* Paper presented at the The 6th International Conference on The Unified Modeling Language - «UML» 2002.

Evans, A., Maskeri, G., Sammut, P., & Willians, J. S. (2003). *Building Families of Languages for Model-Driven System Development.* Paper presented at the Workshop in Software Model Engineering, San Francisco, CA.

Fagan, M. E. (1976). Design and code inspections to reduce errors in program development. *IBM Systems Journal, 15*(3), 182-211.

France, R., & Rumpe, B. (2007). *Model-Driven Development of Complex Software: A Research Roadmap.* Paper presented at the 29th International Conference on Software Engineering, Minneapolis, MN, USA.

Gogolla, M., & Henderson-Sellers, B. (2002). *Analysis of UML Stereotypes in the UML Metamodel.* Paper presented at the UML 2002, Dresden.

International Standard Organization, & Commission, I. E. (2001). *Software engineering – Product quality Part: 1 Quality model*. Genevao. Document Number)

Jouault, F., Bézivin, J., Consel, C., Kurtev, I., & Latry, F. (2006). *Building DSLs with AMMA/ATL, a Case Study on SPL and CPL Telephony Languages*. Paper presented at the 1st ECOOP Workshop on Domain-Specific Program Development (DSPD).

Kuzniarz, L., & Staron, M. (2002). *On Practical Usage of Stereotypes in UML-Based Software Development.* Paper presented at the Forum on Design and Specification Languages, Marseille.

Kuzniarz, L., Staron, M., & Wohlin, C. (2004). *An Empirical Study on Using Stereotypes to Improve Understanding of UML Models.* Paper presented at the The 12th International Workshop on Program Comprehension, Bari, Italy.

Laitenberger, O., Atkinson, C., Schlich, M., & Emam, K. E. (2000). An experimental comparison of reading techniques for defect detection in UML design documents. *The Journal of Systems and Software, 53*(2), 183-204.

Mellor, S. J., & Balcer, M. J. (2002). *Executable UML : a foundation for model-driven architecture*. Boston ; San Francisco ; New York: Addison-Wesley.

Mellor, S. J., Kendall, S., Uhl, A., & Weise, D. (2002). *Model-Driven Architecture.* Paper presented at the Object-Oriented Information Systems, Montpellier.

Miller, J., & Mukerji, J. (2003). MDA Guide. 1.0.1. Retrieved 2004-01-10, 2004, from http://www.omg.org/mda/

Object Management Group. (2003). Unified Modeling Language Specification v. 1.5. Retrieved 2003-10-01, 2003, from www.omg.org

Object Management Group. (2004, December 2003). Unified Modeling Language Specification: Infrastructure version 2.0. Retrieved 2004-02-20, 2004, from www.omg.org

Porter, A. A., Votta, L. G., Jr., & Basili, V. R. (1995). Comparing detection methods for software requirements inspections: a replicated experiment. *Software Engineering, IEEE Transactions on, 21*(6), 563-575.

Shaw, M. (2000). *Software engineering education: a roadmap.* Paper presented at the International Conference on Software Engineering, Limerick, Ireland.

Staron, M. (2006). *Adopting MDD in Industry - A Case Study at Two Companies.* Paper presented at the ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems, Genova, Italy.

Staron, M., Kuzniarz, L., & Thurn, C. (2005). *An Empirical Assessment of Using Stereotypes to Improve Reading Techniques in Software Inspections.* Paper presented at the Third Workshop on Software Quality, St. Louis, MO.

Staron, M., Kuzniarz, L., & Wallin, L. (2004a). A Case Study on Industrial MDA Realization - Determinants of Effectiveness. *Nordic Journal of Computing, 11*(3), 254-278.

Staron, M., Kuzniarz, L., & Wallin, L. (2004b). *Factors Determining Effective Realization of MDA in Industry.* Paper presented at the 2nd Nordic Workshop on the Unified Modeling Language, Turku, Finland.

Staron, M., Kuzniarz, L., & Wohlin, C. (2004). *An Industrial Replication of an Empirical Study on Using Stereotypes To Improve Understanding of UML Models.* Paper presented at the Software Engineering Research and Practice in Sweden, Linköping, Sweden.

Staron, M., Kuzniarz, L., & Wohlin, C. (2006). Empirical assessment of using stereotypes to improve comprehension of UML models: A set of experiments. *Journal of Systems and Software, 79*(5), 727-742.

Staron, M., & Wohlin, C. (2006, June 12-14, 2006.). *An Industrial Case Study on the Choice between Language Customization Mechanisms.* Paper presented at the 7th International Conference, PROFES 2006, Amsterdam, The Netherlands.

Starr, L. (2002). *Executable UML: how to build class models*. Upper Saddle River, NJ: Prentice Hall.

Uhl, A., & Lichter, H. (2002). *A UML Variant for Modeling System Searchability.* Paper presented at the Object Oriented Information Systems, Monpellier.

Wirfs-Brock, R. (1993). Stereotyping: a technique for characterizing objects and their interactions. *Object Magazine, 3*(4), 50-53.

Wirfs-Brock, R., Wilkerson, B., & Wiener, L. (1994). Responsibility-driven design: Adding to your conceptual toolkit. *ROAD, 2*(1), 27-34.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslèn, A. (2000). *Experimentation in Software Engineering: An Introduction*. Boston MA: Kluwer Academic Publisher.

Vokac, M., & Glattetre, J. M. (2005). *Using a Domain-Specific Language and Custom Tools to Model a Multi-tier Service Oriented Application - Experiences and Challenges.* Paper presented at the Model Driven Engineering Languages and Systems, Montego Bay, Jamaica.

Additional reading

1. ATLAS research group website: http://www.sciences.univ-nantes.fr/lina/atl/

   The practitioners interested in the issues of automating the generation of model transformations should read the material on the ATLAS research group. The material describes how the notions of transformations and definitions of models can be unified. The materials include several case studies on industrial applications of these techniques.

2. Atkinson, C., & Kühne, T. (2000). Strict Profiles: Why and How. Paper presented at the ACM/IEEE 3rd International Conference on UML.

   Deeper understanding on the issues of defining profiles for theoreticians can be obtained by reading the material in the paper above. The paper explains the notion of instantiation which is important when defining model transformations. This material supports the reasoning in our experiment.

3. Atkinson, C., & Kühne, T. (2005). Concepts for Comparing Modeling Tool Architectures. Paper presented at the ACM/IEEE 7th International Conference on Model Driven Engineering Languages and Systems.

   A practitioner interested in details how UML model repositories are built should definitely read the above article. The article describes how meta-meta-models are related to models and meta-models in practice. It shows that modeling is usually done in multiple dimensions, which to a large extent can explain the limitations of the current UML tools.

4. Clark, T., Evans, A., Sammut, P., & Willans, J. (2004). Applied Metamodeling - A Foundation for Language Driven Development (1st ed.): Xactium.

   The above material is dedicated for practitioners interested in understanding the practical aspects of creating modeling languages. This book is an essential reading for language engineers who want to increase the productivity of modeling beyond the limitations of standard, UML-based modeling.

5. Bell, A. E. (2004, March 2004). Death by UML Fever. *ACM Queue, 2*, 72-80.

Skeptics in the adoption of MDSD should definitely read this article and its references. The author explicitly names the most common types of adopters of MDSD and reveals wholes in their reasoning. The material is a very good counterpart and a set of negative (or realistic – as some researchers put it) view of MDSD.

6. Glass, R. L. (2004). On modeling and discomfort. *Software, IEEE, 21*(2), 104-103.

   In the same tone as the previous article, Robert Glass presents a good debate on the use of domain specific modeling in industrial projects. The outcome of the debate is that the modeling community lacks empirical evidence that modeling indeed increases performance of software development.

7. Thomas, D. (2004). MDA: Revenge of the Modelers or UML Utopia? *IEEE Software, 21*(3), 15-18.

   The article above contains a discussion and explanation of how MDA is an evolution of the known UML-based software development. The authors explore the notions of model transformations and domain specific modeling as the next step in the evolution of UML.

8. Uhl, A. (2003). Model Driven Architecture Is Ready for Prime Time. *IEEE Software, 20*(5), 70-72.

   Practitioners interested in the discussion on whether MDA is mature enough to be used in industry should read the above article. In the article, the author explores the arguments for and against MDA being a viable alternative for industry in the time of its writing.

The readers interested in other industrial case studies can read:

9. Meservy, T. O., & Fenstermacher, K. D. (2005). Transforming software development: an MDA road map. *Computer, 38*(9), 52-58.

   In this article, the practitioners can find an example of appropriate use of MDA in the context of a web application. The authors discuss the levels of abstractions of CIM, PIM, and PSM and their relationships. They conclude that MDA stills needs to mature, even though it has been around for a while.

10. ModelWare project, “MDD maturity levels”, www.modelware-ist.org

    When working with MDSD in practice the issue of maturity of the use of MDSD often arises. The ModelWare project developed an initial version of MDSD maturity model. The model contains five stages which define how mature a use of MDSD is in an organization.

11. Vokac, M., & Glattetre, J. M. (2005). Using a Domain-Specific Language and Custom Tools to Model a Multi-tier Service Oriented Application - Experiences and Challenges. Paper presented at the Model Driven Engineering Languages and Systems, Montego Bay, Jamaica.

    In this article, the practitioners can find more evidence on effort required to develop an industry quality domain specific modeling language. The experiences of the authors show that the development of a good DSL require more than a few weeks of extra effort. This reading is a complementary to the evidence of the productivity increase presented in this chapter.

12. Knodel, J., Anastasopolous, M., Forster, T., & Muthig, D. (2005). An Efficient Migration to Model-driven Development (MDD). Electronic Notes in Theoretical Computer Science, 137(3), 17-27.

    In practice, migration from code-centric to model-driven software development is a multi-stage process. The authors of this article show a simple process of migrating existing projects into MDSD. This reading complements the material in this chapter when discussing the prioritization issues.

13. Zhang, Y. (2004). Test-driven modeling for model-driven development. Software, IEEE, 21(5), 80-86.

    In this case study, the author summarizes the process of modeling and executing test cases using TTCN-3 at Motorola. This material is an interesting reading for practitioners who want to have more than just code generated from their models.

Acknowledgements

The author would like to thank the experts participating in the studies described in this paper. I would also like to thank Ericsson Lindholmen, Ericsson Region South, Blekinge Engineering Software Qualities (BESQ) project, Software Architecture Quality Center (SAQC), and Ericsson SW Research for support in this study.