VTT WORKING PAPERS 114

Päivi Parviainen, Juha Takalo, Susanna Teppola & Maarit Tihinen

Model-Driven Development

Processes and practices

ISBN 978-951-38-7175-8 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1459-7683 (URL: http://www.vtt.fi/publications/index.jsp)

JULKAISIJA – UTGIVARE – PUBLISHER

VTT, Vuorimiehentie 5, PL 1000, 02044 VTT
puh. vaihde 020 722 111, faksi 020 722 7001

VTT, Bergsmansvägen 5, PB 1000, 02044 VTT
tel. växel 020 722 111, fax 020 722 70014

VTT Technical Research Centre of Finland, Vuorimiehentie 5, P.O. Box 1000, FI-02044 VTT, Finland
phone internat. +358 20 722 111, fax +358 20 722 7001

Series title, number and
report code of publication

VTT Working Papers 114
VTT-WORK-114

Author(s)
Päivi Parviainen, Juha Takalo, Susanna Teppola & Maarit Tihinen

Title

Model-Driven Development
Processes and practices

Abstract

The purpose of this publication is to describe the state-of-the-art and results of two surveys about the model-driven development (MDD), particularly from the processes viewpoint. The study and surveys were carried out in the “Model-driven development of highly configurable embedded Software-intensive Systems (MoSiS)” ITEA project.

This publication will study the current state of MDD related processes and practices, as presented by literature. The literature study is based on the information found from publicly available documents, consisting of books, conference proceedings and websites and it gives an introduction to MDD and related processes. The results of two surveys about MDD processes and technologies are also presented.

ISBN
978-951-38-7175-8 (URL: http://www.vtt.fi/publications/index.jsp)

Series title and ISSN		Project number
VTT Working Papers 1459-7683 (URL: http://www.vtt.fi/publications/index.jsp)		7326
Date February 2009	Language English	Pages 102 p. + app. 4 p.
Name of project Model-driven development of highly configurable embedded Software-intensive Systems (MoSiS)	Commissioned by	
Keywords Model-Driven Development, MDD, Model-Driven Architecture, MDA, process, MoSiS, survey, results, challenges, literature study	Publisher VTT Technical Research Centre of Finland P.O. Box 1000, FI-02044 VTT, Finland Phone internat. +358 20 722 4520 Fax +358 20 722 4374	

Preface

Software products have become more complex with an increased number of product features and they have to adapt to market needs that vary with the geography, culture and environment in which they are used. At the same time, software development has faced the demand for shortened development times and higher expectations of product quality. Model-driven development (MDD) is an approach that aspires to tackle the challenge by taking software development into a higher level of abstraction, by using models as primary development artefacts.

MoSiS is an ITEA2 project (no 06035) that aims at improving the cost-efficiency and ability of companies to develop high quality products that are more flexible and adaptable. The MoSiS project plan summaries the project vision as follows:

"The MoSiS vision is that modelling technology, with its capacity for abstraction and for code generation, is a key enabling technology for the successful design, development and management of embedded software systems. Model-driven development (MDD) is gaining acceptance in a number of domains, with proven benefits regarding costs reduction and quality improvement for software development."

As part of the MoSiS project, this study documents the literature study of MDD related processes and practises, as well as results from two MoSiS surveys on real life experiences of using MDD in companies. The structure of the document is as follows:

Chapter one gives an introduction to model-driven development and model-driven architecture. This chapter also introduces the process viewpoint that is the main focus of this study.

Chapter two presents the MDD related processes, methods and practices as found from literature. It also summarises the elements of the surveyed MDD process models.

Chapter three introduced Model-Driven Development Maturity Model that was used as a framework for the first MoSiS survey.

Chapter four gives an overview of the MDD related tools and their characteristics.

Chapter five documents some experiences about MDD related technologies, as found in literature.

Chapter six describes some challenges presented in literature about the current state of the MDD technologies and their deployment.

Chapter seven explains the goals, set-up, and limitations of the 1st MoSiS survey.

Chapter eight reports the results of the 1st MoSiS survey, focusing on the MDD related processes, methods and tools.

Chapter nine explains the goals and set-up of the 2nd MoSiS survey.

Chapter ten reports on the results of the 2nd MoSiS survey, focusing on the influence of the MDD on product development activities and the challenges companies have faced with the MDD.

Finally, chapter eleven summarises findings from the literature study and MoSiS surveys.

Contents

List of acronyms

AMDD	Agile Model Driven Development
CIM	Computation Independent Model
CMMI	Capability Maturity Model Integration
DSL	Domain Specific Language
DSML	Domain-Specific Modelling Language
FRD	Functional Required Documentation
MBSE	Model-Based Systems Engineering
MDA	Model-Driven Architecture
MDD	Model-Driven Development
MDDP	Model-Driven Development Process
MDE	Model-Driven Engineering
MDSD	Model-Driven Systems Development
MVC	Model View Control
OMG	Object Management Group
OOA	Object-Oriented Analysis
OOP	Object-Oriented Process
PIM	Platform Independent Model
PSD	Platform-Specific Design
PSM	Platform Specific Model
RUP	Rational Unified Process
RUP SE	Rational Unified Process for Systems Engineering
SPEM	Software Process Engineer Metamodel
UML	Unified Modelling Language
UUID	Universally Unique Identifier
WIS	Web Information System

1. Introduction

Model-Driven Development (MDD) is an approach that uses models as a specification of software and transformations of those models to get the source code. Models are created before the source code is written or generated. MDD aims at speeding up the software development and making it more cost efficient, by using the models to visualise the code and, if used to raise the abstraction, the problem domain. MDD also separates implementation technology from the business logic of a program [MacDonald et al., 2005].

The idea behind MDD is that, it is possible to create models of a system that can then be transformed into a real thing [Mellor et al., 2003]. Systems are modelled at several levels of abstraction and from multiple perspectives and the created models are the primary artefacts of software development. These models can then be transformed into running systems by using generators or by executing the models at run-time. [France & Rumpe, 2007.] Sometimes, the term model-driven engineering (MDE) is used instead of MDD [France & Rumpe, 2007]. In this publication, we will use the term MDD.

Model-Driven Architecture (MDA) from the Object Management Group (OMG) is the most commonly know example of MDD. It has been referred to as a software systems modelling [Duby, 2003]. OMG itself is an industry-driven consortium for developing standards that enable the implementation of MDD. In MDA, the specification of system functionality is separated from the specification of its implementation on a specific technology platform. In MDA, Unified Modelling Language (UML) is used to visualise the code [Ambler, 2007]. When the processing and logic of software is separated from the implementation technology, developers are able to concentrate more on solving the problem than the implementation technology details [MacDonald et al., 2005].

MDA presents the modelling of systems at three viewpoints [France & Rumpe, 2007]. The computation independence viewpoint concerns the environment in which the system will operate, as well as the required features of the system and results in computation independent models (CIM). The platform independent viewpoint is interested in the features that are unlikely to change when the system is used on different platforms. This viewpoint produces platform independent models (PIM). PIMs

specify what the system does [Duby, 2003]. PIMs are integrated with the platform specific details in the platform specific viewpoint and result in the platform specific model (PSM). PSMs specify how the system is implemented [Duby, 2003].

Duby [2003] lists several advances of using MDA in embedded software development. She highlights that MDA enables quick reaction to changes in functional and technological platform requirements, as well as enabling a large scale reuse of platform independent models. The ease of documentation, lowered quality assurance costs, and improved system quality are also mentioned as advantages by Duby [2003].

France and Rumpe [2007] list some other non-UML modelling approaches, for instance, Alloy and B, which are formal specification languages. They also mention that high-level, math-based programming languages such as Mathlab, Simulink or Stateflow are used for the modelling and analysing of control system software. In general, France and Rumpe [2007] divide models in two broad classes: development and runtime models. With the development models, software is modelled at a level of abstraction that is above the code level. In turn, runtime models are abstractions of runtime phenomena and they depict some aspect of a running system. France and Rumpe [2007] claim that research has traditionally concentrated more on development models, but interest towards the runtime models has increased lately.

The model based approach has been studied from the systems engineering point of view by Estefan [2007]. In the report, he presents a survey of model-based systems engineering (MBSE) methodologies used in industry. Estefan [2007] characterises MBSE methodology as “a collection of related processes, methods, and tools used to support the discipline of systems engineering in a “model-based” or “model-driven” context. His report summaries follow MBSE methodologies: Telelogic Harmony-SE, INCOSE Object-Oriented Systems Engineering Method (OOSEM), IBM Rational Unifire Process for Systems Engineering (RUP SE) for Model-Driven Systems Development (MDSD), Vitec Model-Based System Engineering (MBSE) Methodology, and State Analysis (SA). He presents an overview consisting of the objectives and main elements for each methodology, as well as the status of available tool support. Most of these methods are briefly described in 2.1 based on Estefan [2007].

In the case of MDD, many advantages and disadvantages are recognised, for instance, in [Mellor et al., 2003], [Foustok, 2007], and [Duby, 2003]. Mellor et al. [2003] state that models can be used to increase productivity. As an example, they mention that “it’s cheaper to build a graphical model in UML, say, than to write in Java”. However, they mention that there are also arguments that claim that models offer more obstacles than help. According to Mellor et al. [2003], MDD offers potential for the automatic transformation of models into running systems. They continue that the challenge in companies adopting MDD is how they can find developers that “think at the level of abstraction, above the current programming languages and technology”.

Mellor et al. [2003] point out that MDD is not yet widely used in companies, but it offers great potential. They list that due to the following advantages of MDD, the using of models becomes an advantage instead of an expense:

- "MDD enables re-use at the domain level."
- "MDD increases quality since models are successively improved."
- "MDD reduces costs by using automated processes."
- "MDD increase software solutions' longevity."

1.1 Process viewpoint

In this publication, we focus on the process support for MDD. Process support is required for effective MDD, as it guides the development of the model and helps to manage the model relations. Chitforoush et al. [2007] reviewed several model-driven methodologies. According to their study, MDA does not provide a comprehensive and concrete process that would govern software development activities. Because MDA does not provide guidance to phases, activities or roles, each project that uses MDA has to define its own process or select it from existing MDA based methodologies.

Mansell et al. [2006] accent that only a few organisations have succeeded in maximising the benefits of MDD, but the ones that have succeeded, have established a clear system development process that covered both the activities and tools required to enable the adoption of MDD in the organisation. Vogel and Mantell [2005] state that MDD processes have to work with the existing methodology of the company. They also mention that MDD process differs from traditional software development process, because MDD uses several types of models to enhance the communication between project stakeholders, as well as to produce artefacts semi-automatically. According to Vogel and Mantell [2005], it is important to minimise the disturbance to existing processes when MDD is introduced in a company. Rios et al. [2006] mention that it is not easy to introduce the MDD methods and tools in a project and introducing those into a whole organisation will be even more challenging. The challenge is caused by the fact, that when an organisation adopts MDD, it results in changes in the organisation's culture and processes.

According to Schätz et al. [2004], a good model-based software development process fulfils the following properties:

- Adequate Models: During the development process, different aspects of the system under development are addressed (for the domain of embedded systems, e.g., overall functionality, time and resource limitation, partitioning and deployment, scheduling) during various phases. Specific models are available for the different phases (e.g., requirements analysis, design, implementation, integration) of the development process. Since software systems, and particularly embedded software

systems, are moving from monolithic single-functionality programs to distributed, interactive multi-functionality networks, a central aspect of these models is the treatment of interaction and communication, as well as time-related aspects. Furthermore, models must support the specific aspects required for the application domain (for the domain of embedded real-time systems, e.g., notions like messages/signals, tasks, schedules, controllers, bus). By supporting domain- and application specific modelling elements, model information on the system under development becomes available, leading to stronger analysis and generation techniques (e.g., in the domain of embedded systems, checking the worst case time bounds of a task, or generating a bus schedule from the communication description of a component model).

- Abstract Models: Models should only contain those aspects required to support the development phase they are applied for (e.g., modelling the interaction between components by messages, rather then method calls to the bus driver or the operating system). By abstraction, models reduce the complexity of the description as much as possible, as well as the possibility to produce faulty descriptions (e.g., ensuring type correctness between communication ports rather than using byte block messages at the level of bus communication). Furthermore, abstract models also support descriptions of the system under development in early analysis and design phases, making analysis and generation support available even at early stages of the development process (e.g., the completion of a state-based description of a component to introduce standard behaviour for undefined situations).
- Integration by Analysis: The relation between various models during the development is supported. This includes the analysis of different models used during the same phase (e.g., checking the consistency of a scenario and a complete behavioural description of a component) as well as different phases (e.g., checking a bus communication schedule against the abstract communication behaviour of the corresponding abstract component). This leads to a higher level of product quality, by especially supporting analysis of the system at earlier stages; additionally, it also increases process efficiency by supporting the earlier detection of defects.

Integration by Generation: The transition between different models in the development process is supported by generating models out of each other. This includes forward generation (i.e., from models of an earlier phase to a latter phase; e.g., the generation of test cases from a behavioural description) as well as backward generation (i.e., from a latter model to an earlier one; e.g., the generation of an abstract scenario-based description from an execution trace of the implementation level). This leads to increased process efficiency by a higher degree of automation, as well as increased product quality by eliminating the defects introduced by manual development steps.

2. MDD processes, methods and practices

There are several other processes for model based development published in the internet, conferences, journals and books. They (as far as was found) are also described in this chapter briefly. We also describe the elements of MDD that are normally found in the methods.

2.1 Example MDD processes and methods

2.1.1 MODA-TEL

MODA-TEL [MODA-TEL Web page, 2009] is a joint effort of European organisations that have business interests in component architectures. MODA-TEL aims at developing methodologies and tools to support the MDA among the telecommunications industry. Gavras et al. [2003] have documented the MODA-TEL Model-Driven Methodology that they adopted in the MODA-TEL project. Methodology provides definitions for concepts used in the methodology, as well as identifying phases and activities. Methodology depicts three groups of MDA users: experts on how to "build know-how repositories", experts on how to "assemble, combine, customise and deploy know-how" and experts on how to "apply know-how".

Gavras et al. [2003] depict that it is important to separate preparation from execution. The MODA-TEL methodology (see Figure 1) identifies five phases in which three of them ("Preliminary preparation", "Detailed preparation", and "Infrastructure set-up") are related to preparation and one ("Project execution") is related to the execution of the project. One phase is related to the management ("Project management") of model based development. The following is the list of activities in each phase of the methodology:

- Activities in the "Preliminary preparation" phase are: "Platform identification", Modelling language identification", "Transformation identification", and "Traceability strategy".

– Activities in the “Detailed preparation” phase are: “Specification of modelling languages” and “Specification of transformations”.
– Activities in the “Infrastructure set-up” phase are: “Tools selection” and “Meta-data management”.
– Activities in the “Project execution” phase are: “Requirements analysis”, “Modelling”, “Verification and validation”, “Transformations”, “Coding and testing”, “Integration deployment”, and “Operation maintenance”.
– Activities in the “Project management” phase are: “Software Development Process (SDP) Selection”, “Project organisation”, and “Quality management”.

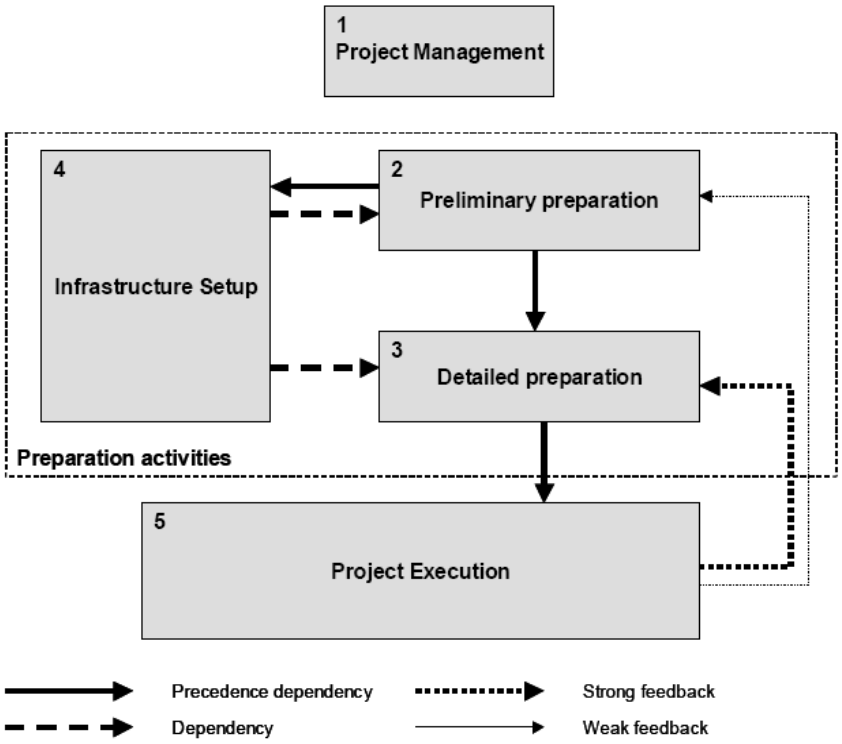

Figure 1. Phases of the MODA-TEL methodology as in [Gavras et al., 2003].

2.1.2 MASTER

Larrucea et al. [2004] present an MDD process and System Family Engineering concepts that can be used to adapt the MDD process according to user and functional requirements. They use notation from the Software Process Engineer Metamodel (SPEM) to describe the process. The methodology was developed during the MASTER project that was the European IST project (MASTER project IST-2001-34600).

The following are the main phases of the MDD process with their objectives as presented in [Larrucea et al., 2004].

- Capture user requirements: The objective is to identify, agree and document the functional and non-functional user requirements.
- PIM context definition: The objective is to define the scope for the software system that will be developed.
- PIM requirements specification: The objective is to create complete models of customer requirements, as well as to create unique requirement descriptions for all the subsequent models to use.
- PIM analysis: The objective is to model the system's internal view, in which the technological considerations are excluded and the separation between the functional and non-functional requirements is maintained.
- Design: The objective is to model the behaviour and structure of a solution, which fulfils both the functional and non-functional requirements.
- Coding and integration: The objective is to develop and verify the code that implements the design and fulfils the requirements.
- Testing: The objective is to show that requirements are satisfied by the developed system.
- Deployment: The objective is to ensure that the developed system is successfully taken into use among the final users.

2.1.3 MIDAS

MIDAS is an MDA based model driver methodology that is targeted for the development of Web information systems (WIS) [Cáceres et al., 2003]. Methodology utilises UML and proposes different platform independent (PIM) and platform specific (PSM) models. MIDAS defines mappings from PIM to PIM, from PIM to PSM, and from PSM to PSM.

The definition of the service architecture model is part of the MIDAS methodology, a framework for the development of information systems based on the MDA principles, in which the architecture guides the whole development process. The MIDAS model architecture, as shown in Figure 2, can be seen from 3 orthogonal dimensions:

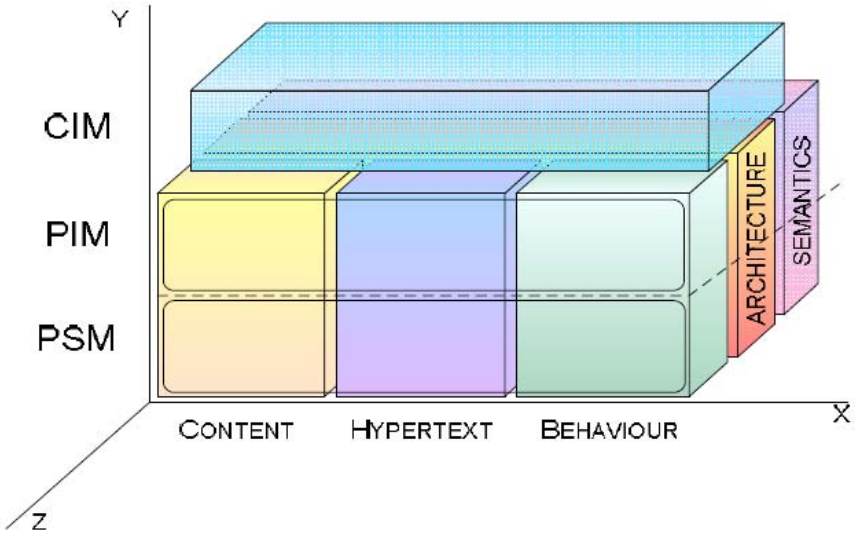

Figure 2. MIDAS Model Architecture.

- Vertical Axis (Y): reflects the three abstraction levels of the original MDA proposal.
- Horizontal Axis (X): Models are separated in the different concerns involved in the information system development.
- Traversal Axis (Z): Models in this axis are referred to aspects, which have an influence on other models of the cross-cutting axis.

The architecture is considered to be the driving aspect of the development process, as it allows the specification of which models, elements inside models or relations within models should be created during the entire software development process. With an architectural view of the system at a PIM level, it's ensured that there are no technology or implementation constraints in the model. In addition, it facilitates the establishment of different PSM-level models, according to the specific target platform and derived from one unique PIM model. [López-Sanz et al., 2008]

2.1.4 C^3

Hildenbrand and Korthaus [2004] present the C^3 approach for business application engineering that is composed of "Collaboration", "Concurrent Software Engineering", and "Component Orientation" elements. They have identified "Domain manager", "Project manager", "Application architect", "Component developers", and "Application deployer" as important roles in using the methodology during the development of business software.

According to Hildenbrand and Korthaus [2004], C^3 methodology consists of "standardisation" and "software development" phases. Standardisation refers to the actions of accessing and downloading the domain software assets from domain data repository to project data repository. Newly developed component models and domain-specific model elements can also be uploaded for future reuse. The software development phase consists of "Model design", "Code generation", and "Application deployment" steps. During the model design step, the architecture for the business application is selected and worked on by the developers. In the code generation phase, after the business application modelling is performed, the aim is to generate as much executable platform specific code as possible from the models. After the code is generated and possibly completed by the developers, the components will be deployed on a given application server based on the modelled architectural framework. [Hildenbrand & Korthaus, 2004.]

2.1.5 MDDP

The model-driven (software) development process – MDDP – can either be planned incrementally or as a waterfall project. Both processes are described on MDDP web pages [CRaG Systems, 2008] in great detail, including templates and detailed guidelines. The process is not reprinted here, as requested by the authors.

The model-driven (software) development process – MDDP – is a business process, requirements, use case, risk and model-driven. It provides traceability through all the stages from the business processes through system requirements, analysis and design models into test scripts and code. It conforms to the Object Management Group's Unified Modelling Language v 2.0, the Software Process Engineering Metamodel and the Model-Driven Architecture approach. It can be used to satisfy every level of the Capability Maturity Model Integration (CMMI) [CMMI Web page, 2009]. It also incorporates the broad principles of many current development trends, including component-based development, agile development, test-driven development, aspect oriented design and even extreme programming.

The model-driven (software) development process – MDDP – can be used 'as-is', configured or customised and the improvement of the process is part of the process.

2.1.6 Model Based Software Development Process for Production Applications

Chandrashekar et al. [2006] present a model based software development process for production applications, and experiences of using it in an automotive domain.

Model based software development starts with requirements being modelled to form the design. This design is basically graphical models such as Simulink® / Stateflow®

models incorporating process and control logic. This step involves analysing the requirements to decide the input and output interfaces and eventually implementing the algorithm. Most importantly, some constraints are imposed on the modelling by way of style guidelines. This is done to make the models more consistent in style and structured and also to make the model suitable for autocode. Along with the model, the design also consists of a data dictionary, containing the list of calibrations and the module's interfaces. The data dictionary provides the range and resolution of the calibrations and the interfaces, which are used during the implementation.

In the next step, the models created above are validated against the requirements. This validation can be done on the host by way of simulation. The host validation can be performed in an interactive way using tools such as Altia® or GUIDE. A more vigorous way of model validation is by using a rapid prototyping environment. For example, running and testing the models in the vehicle using an emulator like dSpace's Autobox.

The validated model is then used to generate the C – code, using autocoding tools such as dSpace's Targetlink®, Mathworks's E-Coder. The first step is to make the model compatible for autocoding. This can be partially taken care of during the modelling with proper style guidelines. Further, the design model is modified to satisfy the target controller constraints and operating system requirements. Software information is then entered into this model to form the autocode model. Finally, an autocode is generated from this model. It must be noted that autocoding is not completely automatic coding. Rather, it is a graphical way of coding as it involves the manual effort of software partitioning and entering appropriate software information to obtain the optimised code.

Even though autocode is automatically generated, testing is still required because their software properties are entered manually. During the autocode testing, the autocode is tested against the model. In the model-based approach, errors in the autocode due to a wrong interpretation of the logic tend to be zero as the autocode is obtained directly from the requirement model. However, errors do exist in the autocode and are mainly due to manual activities such as range-resolution analysis and software partitioning. Errors in the autocode will be typically overflows, underflows and resolution deficiencies. Autocode testing strategies are developed to detect the above errors. In short, the testing is performed to ensure that the autocode behaves as the design expects it to do.

The process described in this paper is based on the authors' experiences using the combination of MATLAB/Simulink/Stateflow from The Mathworks Inc and TargetLink® from dSpace. However, the same result could be applied to the code, which is generated with E-coder from The Mathworks Inc instead.

Several production programs have followed this process over the past few years. The same process has been used for both fixed-point and floating-point autocoding. Based

on the experience described in this paper, the Model Based Software Development process scores higher than the conventional hand-code process, with respect to development time and software quality. The engineering effort in fixed-point autocoding is around 50% less compared to conventional hand coding.

2.1.7 General framework for model-driven development based on Model-Driven Architecture

Chitforoush et al. [2007] surveyed several model-driven methodologies and based on the survey, they proposed their own MDA based framework for the MDD. According to Chitforoush et al. [2007], their framework can be used in assessing, comparing, and adapting existing methodologies, as well as engineering new methods that meet the requirements of the MDD. The proposed framework consists of four main phases. These phases are “Project Initiation”, “Software Development Process Analysis and Selection Phase”, “MDA Support Phase”, and “Software Development Process Execution Phase”. Moreover, a number of monitoring and management activities are needed in parallel with the four main phases. Each main phase consists of a number of stages, which in turn, consists of activities.

2.1.8 MDD Process Framework

Vogel and Mantell [2005] present the Process Framework for implementing the MDD and they document an example case, in which the framework is adopted by SME. The case company used a traditional waterfall software development process and during the study they adapted the existing process to use MDD tools, techniques and roles. In their case example, Vogel and Mantell [2005] list the following phases in the MDD process implementation:

- A requirements phase in which the requirements related to the system to be developed were being identified, captured and validated.
- An analysis phase, in which functionally-independent and platform-independent specifications were produced for the system.
- A design phase, in which the technical specification of the system was produced.
- A development phase, in which the code of the system was generated, checked and manually completed.
- A quality assurance phase, in which the defects of the software were identified and fixed.

2.1.9 Agile Model Driven Development

Agile Model Driven Development (AMDD) is an agile version of MDD. Basically, in traditional MDD, extensive modelling is performed before writing or generating a source code. In the AMDD, extensive models are not made such as in the traditional MDD. Instead, during the MDD, agile models, which are "just barely good enough", are produced. [Ambler, 2007.]

Lahman [2006] examines an alternative to the OOP-based processes that is based upon model-based development; specifically object-oriented analysis (OOA) models. The primary goal is to demonstrate that model-based development can actually be more agile with respect to the primary goals of agile software development than the OOP-based processes that are most commonly associated with agile development.

2.1.10 Process, product, conceptual and system models

Schätz et al. [2002] and Schätz and Spies [2002] describe process, product, conceptual, and system models and their relations: The first two models are used to describe the development process from the engineer's and thus the domain model point of view. Together, process and product models form the domain model:

- Process model: The process model consists of a description of activities of a development process and their relations. In the domain of embedded reactive systems, e.g., the process model typically contains modelling activities ("define system interface", "refine behaviour") as well as activities ("generate scenarios or test cases", "check the refinement relation", "compute the upper bound for worst case execution time"). The activities are related by their dependency between them for defining a possible course of activities throughout the development process. By relating them to a product model, process patterns can be formalised as activities and thus integrated into the process model.
- Product model: The product model consists of the description of those aspects of a system under development, explicitly dealt with during the development process and handled by the development tool. For embedded systems, a product model typically contains domain concepts such as "component", "state" or "message", as well as relations between these concepts such as "is a port a component", etc. In addition to these more conceptual elements, used for the description of the product, more semantically oriented concepts like "execution trace" are defined to support, for example, the simulation of specification during the development process. Finally, it contains process oriented product concepts such as "scenario" or "test case", supporting the definition of process activities.

The product model describes the aspects, concepts and their relations required to construct a product during the development process. While the ‘abstract syntax’ is sufficient to describe the conceptual relations of abstract views or the functionality of modelling activities during the process, a semantical relation is required to define or verify the more complex semantical dependencies of views, as well as the properties of activities (such as refining activities or activities not changing the behaviour as, e.g., refactoring). Since the semantical and the conceptual part of the product model are used differently in the model-based approach, the product model is broken up into two sub-models:

- Conceptual model: The conceptual model consists of the modelling concepts and their relations used by the engineer during the development process. The conceptual model is independent of its concrete syntactic representation used during the development process. Typical domain elements for embedded systems are concepts such as “component”, “port”, “channel”, “state”, “transition”, etc. Typical relations are “is a port of”, “is behaviour of”, etc.
- System model: The system, or semantical, model consists of semantical concepts needed to describe the system under development. A typical element is “execution sequence”. Typical relations are “behavioural refinement” or “temporal refinement”.

To illustrate how process, conceptual and system models interact, Schätz et al. [2002] use the example of the “elimination of dead states” refactoring step that reduces the code size. In this example, a control state, including all of its adjacent transitions, is removed from an automaton provided that this state is marked as unreachable.

- Process: On the level of the process model, an activity “Show the unreachability of a state” must be introduced. This step may either be a single atomic operation (and can be carried out, e.g., by some form of model checking algorithm) or a more complex operation requiring user interaction. These operations correspond to the operational relations of the system model. Furthermore, the activity “Remove dead state” removes all transitions leading to a state marked unreachable as well as the state.
- Conceptual: On the level of the conceptual model, the concept of “unreachability” of a state, e.g. as a state annotation, is introduced. If no user interaction is required for the proof of unreachability, this extension is sufficient. Otherwise, the conceptual elements of proof steps must be added, as well as additional concepts such as state/assignment of variables, or preconditions of a transition.
- System: On the level of the system model, the semantics have a direct effect: in case of an atomic operation, there is an operational notion of the semantical predicate “unreachability”, e.g., in the form of a model-checking algorithm. This

operation adds the conceptual annotation “unreachable” to a state unreachable, according to the semantics of the system model. In case of an operation requiring user interaction, the system model is used via atomic operations corresponding to the operational relations of the system model (e.g. combining parts of a proof, applying modus ponens). Besides this application of the system model “at run-time of the CASE tool”, the system model is also applied “at build-time” to prove the correctness of the refactoring step.

Since an activity, as the atom of a process, describes how a product is changed, an activity can be understood as a small process pattern. Additionally, each activity is described in an operational manner. Furthermore, by the use of the system model, properties such as “soundness considering behavioural equivalence” or “executability of the specification” can be established for activities and phases. This combination of user guidance by consistency conditions, of executable activities, and of the possibility for both arbitrary and provably sound process activities and states of a product, is directed at improving the efficiency of the CASE based development process.

2.1.11 SDL-MDD

Kuhn et al. [2006] present SDL-MDD, a model-driven development process that is based on the ITU-T design language SDL (see Figure 3). They also present a semantically integrated tool suite, consisting of several commercial tools, including a graphical SDL editor, an SDL model debugger, and an SDL-to-C compiler, especially supporting model-driven code generation and model-driven simulation. Both the production and simulation code are entirely generated from SDL models and automatically instrumented to interface with different operating systems and communication technologies. The use of SDL-MDD and of the tool suite is also illustrated by an extensive case study from the ubiquitous computing domain, in the article.

SDL-MDD decomposes the development of a software system into a number of stages. Since it is an iterative process, the development of a software system may go through some stages repeatedly, while certain stages may be skipped over during the early cycles.

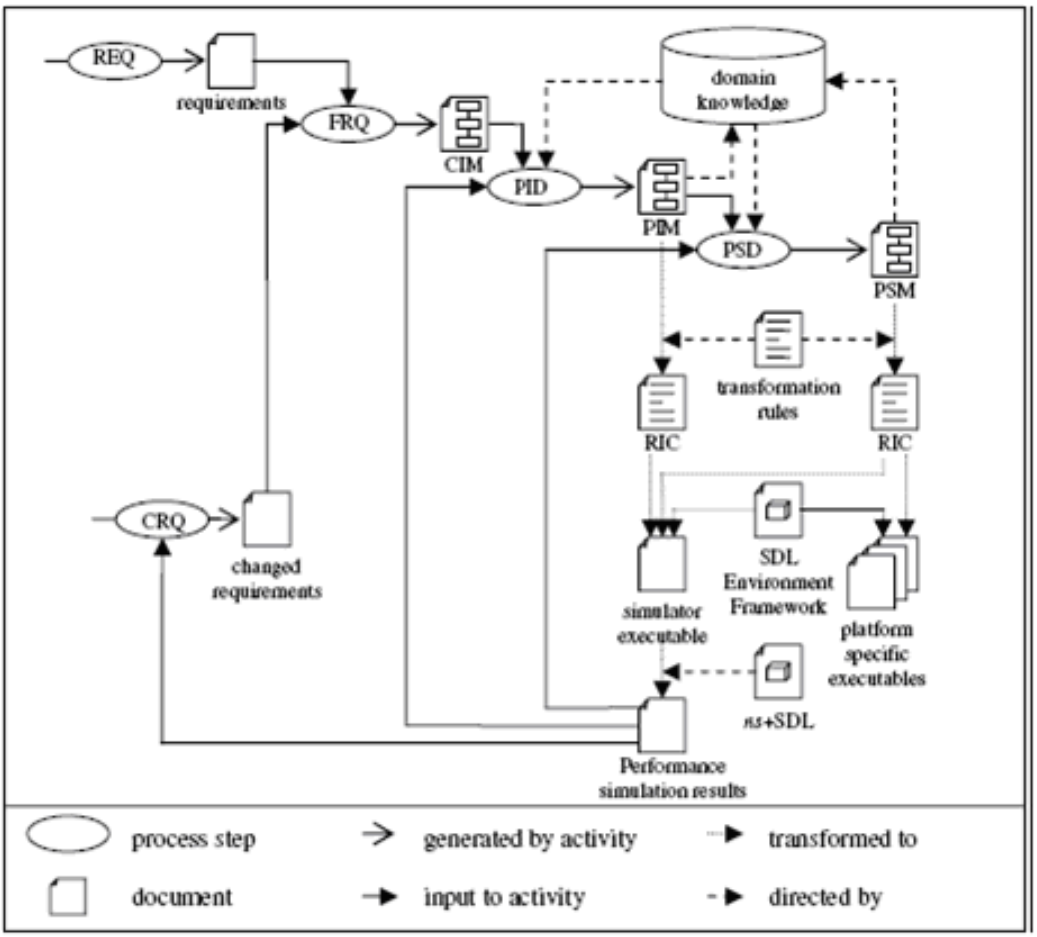

Figure 3. SDL-MDD process model [Kuhn et al., 2006].

The following stages are distinguished in the SDL-MDD process:

- In the requirements stage (REQ), the requirements are elicited and described in an informal way.
- In the formalise requirements stage (FRQ), the requirements document is partially formalised, yielding a computation-independent model (CIM). Similar to the MDA, the CIM is a system specification from the viewpoint of the domain expert. Since the SDL-MDD is directed towards the ubiquitous computing domain, we specify message scenarios with MSC on different levels of granularity. This specification can be traced throughout all the models of subsequent development stages, and be validated against scenarios that are generated by the SDL model debugger.
- In the platform-independent design (PID) stage, the platform-independent model (PIM) is specified, using SDL as the design language. It contains both the platform independent structure of the software system, and the platform-independent

behaviour. The result of this stage is a functionally complete, closed SDL design model, which can be analysed using existing tools for the debugging and validation of the functional system behaviour.

- In the platform-specific design (PSD) stage, the platform-specific model (PSM) is specified, again using SDL as the design language. The fact that the PSM incorporates the PIM, and that the same design language is used as for the PIM, makes transformations between the PIM and corresponding parts of the PSM obsolete. However, design decisions leading to platform-specific PSM components are to be made. The resulting SDL design model can be analysed using existing tools for the debugging and validation of the functional system behaviour. In addition, the design model forms the basis for model-driven performance simulations.
- In the changed requirements (CRQ) stage, the initial requirements may be modified, based on feedback from the performance assessment.

It should be noted that the SDL-MDD process does not contain an explicit implementation stage. The reason is that implementations including the code for interfacing with system environments are generated entirely from SDL models.

2.1.12 MDA effects on the RUP

Nikulsins and Nikiforova [2008] present the changes caused by using MDA to RUP (rational unified process):

- At the Inception phase, requirements for the system are being elicited, resulting in a CIM. A CIM model covers the Business Modelling and Requirements disciplines. At the same time, during the Inception phase, metamodel planning for the PIM development and initial implementation are carried out.
- Elaboration is the main phase impacted by an MDA project. It is important to look at the Elaboration activities and briefly describe the MDA modifications. The Architect role is the main role in the Elaboration phase that requires additional consideration. As a specialisation of “Architect”, the MDA Architect role is appropriate for many MDA projects. Essentially, this role defines specific MDA activities and artefacts, creates the transformations, and so on. The primary MDA artefacts of this role are mapping documents, transformations, and UML profiles.
- PIM also covers some parts of the Construction – the models should be enriched with mapping functions and appropriately marked. Models are iteratively tested for conformance. At the Construction phase, model transformation into different PSM or codes takes place.

– PSM covers the Transition phase and Deployment discipline as well – transition to some production environment can be performed with the help of a separate PSM as well.

– The Configuration and Change Management discipline is also affected by the MDA – these disciplines cover metamodel and metadata repository maintenance.

Typically, the MDA automates activities within the RUP. Rather than changing RUP activities, the MDA enhances them with additional tasks aimed at supporting automation with a number of the primary RUP activities. The primary changes to the RUP involve a more subtle change of perspective in the development process. MDA encourages architects and developers to work at higher levels of abstraction than typically expected in non-MDA projects. This is most apparent during construction, where the code automation aspects of MDA significantly change the emphasis of the implementation tasks. They work less with actual implementation models and source codes themselves, and more with designs for the appropriate business-focused workflow of the solution. A smaller subset of developers will implement the model-to-code transformations themselves.

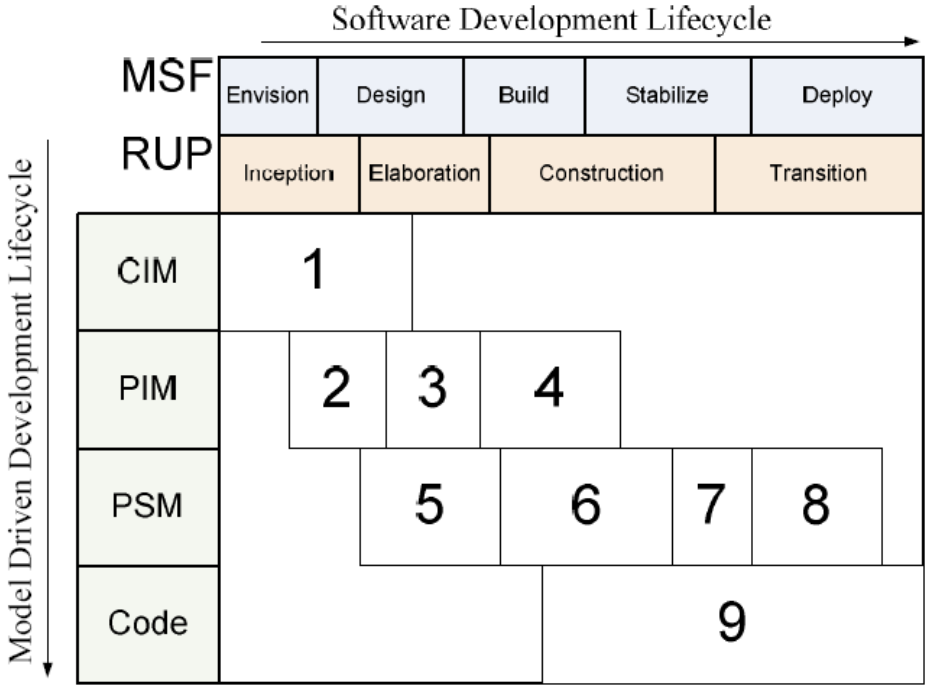

Figure 4. Software development lifecycle and MDA mappings [Nikulsins & Nikiforova, 2008].

According to Figure 4, appropriate activities in the phase crossings are marked with digits. Their description is provided in more detail in Table 1.

Table 1. MDA specific activities in the phase crossings.

1.	Define the possible domain specific metamodel Traceability convention from requirements to code Choose the modelling tool Technology selection Requirements specification development (ontology) Information structuring Initial PIM development
2.	Transformations strategy selection Business process modelling role evaluation Risk assessment
3.	Refine the PIM context Define verification strategies PIM design
4.	Finalise metamodels Create model-capturing support (model instances, which are not easy to illustrate in diagrams) Annotate the PIM with platform specific information Verify the models
5.	PSM identification Define the verification strategies Define the user-interface options (with model aware aspects) PSM metamodel design
6.	Verify the models Mark the PIM and PSM Create additional transformations and code generation scripts for target platforms Create test scripts PIM to PSM automatic or semi-automatic transformation
7.	Verify the established meta-models Code files generation from PSM
8.	Automatic deployment (deployment profiles or PSMs can be used)
9.	Fine-tuning and repeated use of the automated testing process Executable code verification to models

2.1.13 Telelogic Harmony-SE

This description is based on [Estefan, 2007], where a more detailed description and references are also given.

The Harmony process was designed to be tool- and vendor-neutral, although elements of the process are supported by the Telelogic Rhapsody model-driven development environment (formerly, I-Logix Rhapsody) and by the Telelogic Tau offering. Note that the Harmony process somewhat mirrors the classical "V" lifecycle development model of system design. The process assumes that model and requirements artefacts are maintained in a centralised model/requirements repository. The task flow and work products (artefacts) in the Harmony-SE process include the following three top-level process elements:

- Requirements analysis
- System functional analysis
- Architectural design.

Figure 5 illustrates these process elements in more detail, along with the flow of some of the primary work products.

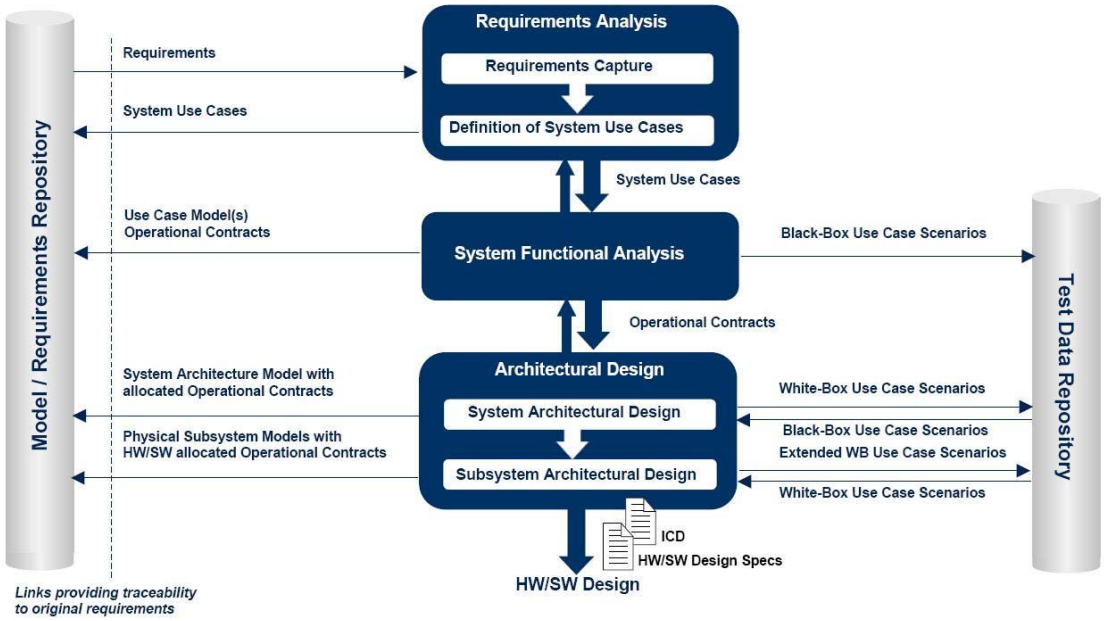

Figure 5. Harmony process elements.

2.1.14 RUP SE

This description is based on [Estefan, 2007], where a more detailed description plus references are also given.

The Rational Unified Process for Systems Engineering (RUP SE) is a derivative of the Rational Unified Process® (RUP®). The RUP SE was created to specifically address the needs of systems engineering projects. The objective for its creation was to apply the discipline and best practices of the RUP for software development to the challenges of system specification, analysis, design, and development.

Since the RUP SE is derived from the RUP, it retains the RUP's cornerstone principles, which have been refined and extended to enhance their utility for systems engineering efforts. The RUP SE brings the RUP style of concurrent design and iterative development to systems engineering (as illustrated in Figure 6). In addition, it provides the highly configurable discipline (workflow) templates required to identify the hardware, software, and worker role components, which comprise a systems engineering project.

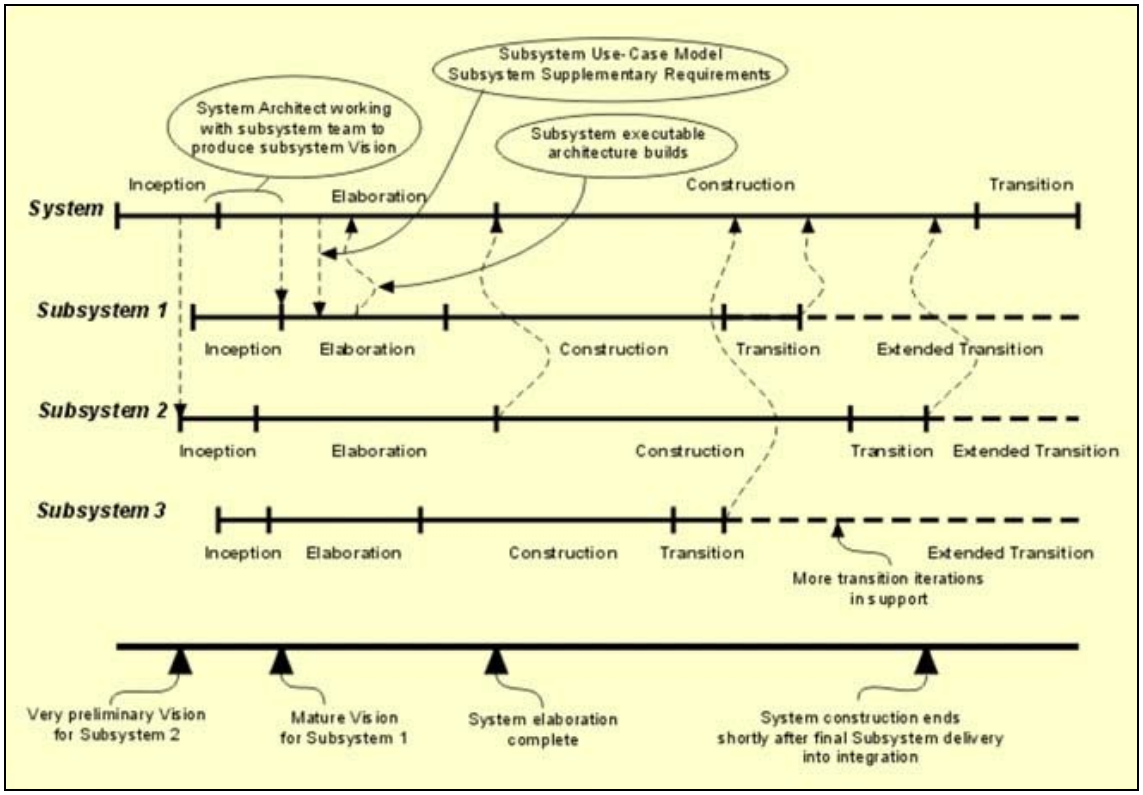

Figure 6. RUP SE lifecycle.

2.1.15 Vitech Model-Based System Engineering (MBSE) methodology

This description is based on [Estefan, 2007], where a more detailed description plus references are also given.

Although the Vitech MBSE methodology is considered to be "tool-independent," there is a strong tie of the tutorial materials to the CORE tool set. The Vitech MBSE methodology is based on four primary concurrent SE activities, which are linked and maintained through a common System Design Repository (see Figure 7). Each of these primary SE activities is linked within the context of associated "domains", where the SE activities are considered elements of a particular kind of domain, known as the Process Domain.

Four core tenets help drive the Vitech MBSE methodology:

1. Model via modelling "language" the problem and the solution space; include semantically-meaningful graphics to stay explicit and consistent. This helps facilitate model traceability, consistent graphics, automatic documentation and artefacts, dynamic validation and simulation, and promotes more precise communication.
2. Utilise a MBSE system design repository.
3. Engineer the system horizontally first before vertically, i.e., do it in complete, converging layers.
4. Use tools to do the "perspiration stuff" and your brain to do the "inspirational stuff".

To support tenet #3 above, the Vitech MBSE utilises an incremental SE process known as the "Onion Model," which allows complete interim solutions at increasing levels of detail during the system specification process.

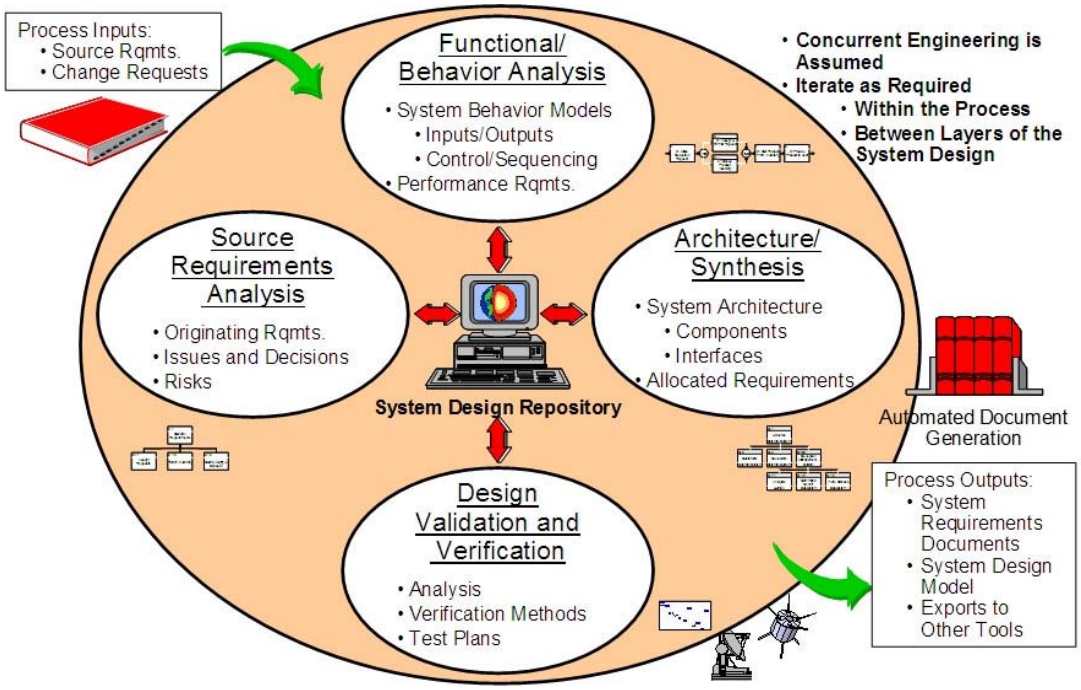

Figure 7. Vitech MBSE primary SE activities.

2.1.16 State Analysis (SA)

This description is based on [Estefan, 2007], where a more detailed description plus references are also given.

State Analysis (SA) is a JPL-developed MBSE methodology that leverages a model- and state-based control architecture (see Figure 8), where the state is defined to be "a representation of the momentary condition of an evolving system," and models describe how the state evolves.

SA provides a process for capturing system and software requirements in the form of explicit models, thereby helping reduce the gap between the requirements on software specified by systems engineers and the implementation of these requirements by software engineers. Traditionally, software engineers must perform the translation of requirements into system behaviour, hoping to accurately capture the system engineer's understanding of the system behaviour, which is not always explicitly specified. In SA, model-based requirements are mapped directly to software.

In SA, it is important to distinguish between the "state" of a system and the "knowledge" of that state. The real state may be arbitrarily complex, but ones knowledge of it is generally captured in simpler abstractions that one finds useful and sufficient to characterise the system state. These abstractions are called state variables. The known state of the system is the value of its state variables at the time of interest.

Together, the state and models supply what is required to operate a system, predict a future state, control towards a desired state, and assess performance.

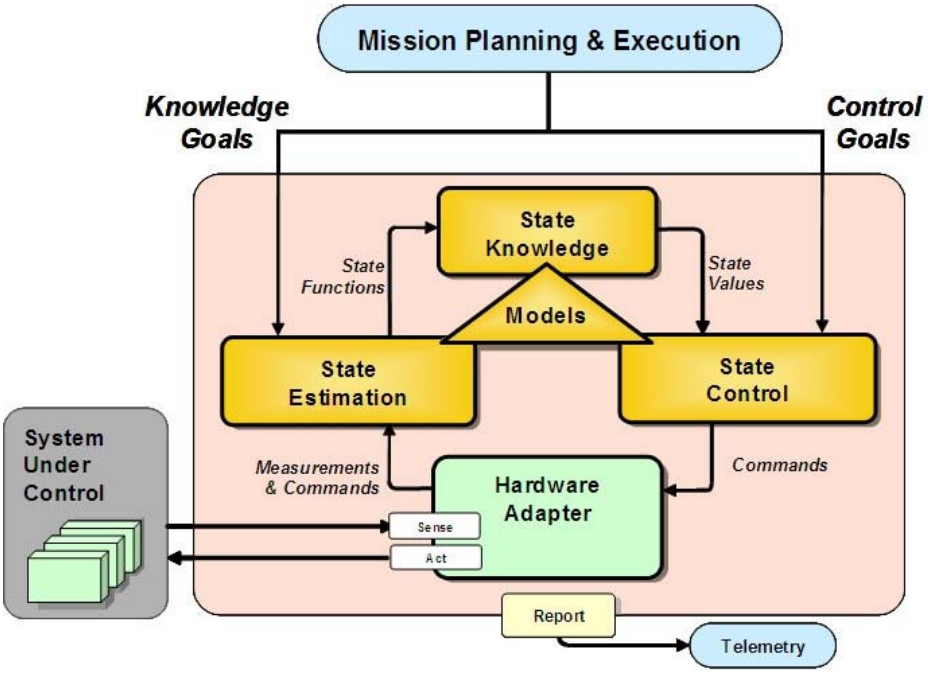

Figure 8. Model- and State-Based Control Architecture (“Control Diamond”).

2.2 Elements in the MDD processes

In this section, we will summarise the elements of the surveyed MDD process models. The process models use various names for the activities, but at a high level the same activities are commonly done as in non-model-driven development. Thus, the elements are presented using a commonly known structure of product development activities. The focus is on development work, the supporting activities such as project management are mostly out of the scope of this section, since there is not much difference in them due to the MDD, except for planning and managing the MDD activities. This section is based on the models presented earlier in this publication, as well as the examples from [Afonso et al., 2006].

In general, the process models don’t specify in detail what should be done during the development work and when. This makes their application in practice difficult. The reason for this could be that the main source of the models was conference and journal articles with limited space to describe the process models or methods in detail.

Context definition: The objective is to define the scope for the software system that will be developed.

- In the MDDP, this is often called developing the context for the PIM, platform independent model.

Requirements elicitation and analysis: The objective is to identify, capture, agree and validate the functional and non-functional requirements related to the system to be developed.

- The Model-driven view is that complete models of customer requirements, as well as unique requirement descriptions for all the subsequent models to use, are created.
- Example: The requirements gathering phase begins by capturing the business/ project concepts and entities. UML class diagrams were created to specify more details of the concepts and entities. The next step was to focus on the development of the use-cases model using the UML 2.0 modelling tool (RSA). The use-cases model was created following a top-button approach: from finding use-cases and its interaction with the system actors to identifying the use-cases' main/alternate flows and detailing its behaviour using activity diagrams. Each textual requirement has been associated to correspondent to use-cases, establishing traceability relationships. The Computation Independent Model (CIM) is produced as a result of this phase. From the CIM, one can extract (and deliver to a customer) a readable documentation generated completely from the model (e.g. glossary, use-cases, etc.).

Early analysis: The objective is to model the system's internal view, in which the technological considerations are excluded and the separation between the functional and non-functional requirements is maintained.

- The model-driven view means that the functionally-independent and platform-independent specifications for the system are produced. This step involves analysing the requirements to decide the input and output interfaces and eventually implementation of the algorithms.
- Example: In the analysis phase, the PIM model has been created using UML 2.0 models in the RSA. Having the use-cases model as the main input, the use-cases of the participant classes have been identified and their behaviour have been analysed creating the analysis use-cases realisations. Class diagrams were created to model the system boundaries, controls and entities and sequence diagrams were created to capture the behaviour and the responsibilities of those classes. Use-cases have been transformed in analysis use-case realisations, generating the traceability relationships between the requirements and the PIM model. The PIM is the main input model for Enablers functional required documentation (FRD) and system user interfaces, artefacts that are usually delivered to the customers.

Design: The objective is to produce the technical specification of the system, i.e., to model the behaviour and structure of a solution that fulfils both the functional and non-functional requirements.

- In model-driven development, the design is graphical models such as Simulink® / Stateflow® models, incorporating process and control logic. Created models are validated against the requirements. This validation can be done on the host by way of a simulation. The host validation can be performed in an interactive way using tools. A more vigorous way of model validation is by using a rapid prototyping environment. For example, running and testing the models in a vehicle, using an emulator such as dSpace's Autobox.
- Example: The PSM model considers several platform specific details that reflect the choice of deployment technology: operating systems, databases, programming languages, platform runtimes, high level protocols, etc. To map the PIM model to a J2EE model, the JAVA and EJB UML profiles (built-in in the RSA) were used. Additionally, profiles have been developed for the RSA: for instance, the Model View Controller UML profile was created to be able to use the open-source library Model View Control (MVC) scope. Some RSA pattern mechanisms were used; most of the mapping was made by applying to the models elements, the stereotypes and tagged values provided by the available UML profiles and then transforming our models in J2EE architecture related models.

Implementation (or transformations): The objective is to develop and verify the code that implements the design and fulfils the requirements.

- The model-driven view is that the code of the system is generated, checked and manually completed. The validated model from the design is used to generate the C – code using autocoding tools such as dSpace's Targetlink®, Mathworks's E-Coder. The first step is to make the model compatible for autocoding. This can be partially taken care of during modelling with proper style guidelines. Further, the design model is modified to satisfy the target controller constraints and operating system requirements. Software information is then entered into this model to form the autocode model. Finally, the autocode is generated from this model.
- Example: The MDD is all about Models and transformations, Model to Model but also Model to Text, to perform those tasks with quality and power we used two different transformation tools:
    - ATL is more sophisticated and complex; it is one model to model transformation tool. To generate a code, one needs to define the input metamodel (like in the MOFscript) and also to define the output metamodel (like in JAVA) following the MDA principles of executable models. To use

ATL, the models have been exported to the EMF and declarative transformation rules have been developed to map the models. This is a potentially powerful tool; however, some difficulties have been experienced when using it with RSA models.

- MOFscript is a model to text transformation tool that provides an easier way to parse the model structure outputting the desired code. To work with this tool, the models have been exported to the EMF format, and using the MOFscript specific language, the desired text outputs have been produced. However, as with the ATL, some difficulties have been experienced, such as accessing meta information. The research object system contains features that are not represented in the PSM, however a good code structure can be generated and later augmented by the developers. It is expected that the creation of an hibernate profile will improve the expressiveness and, therefore, potentially increase the quantity of the generated code.

Integration and testing: the objective is to show that requirements are satisfied by the developed system.

- In a model-driven development, even though code is generated, testing is still required because the software properties are entered manually. During the autocode testing, the autocode is tested against the model. In the model-based approach, errors in the autocode, due to the wrong interpretation of logic, tend to be zero as the autocode is obtained directly from the requirement model. However, errors do exist in the autocode and are mainly due to manual activities like range-resolution analysis and software partitioning. Errors in the autocode will be typically overflows, underflows and resolution deficiencies. Autocode testing strategies are developed to detect the above errors. In short, the testing is performed to ensure that the autocode behaves as the design expects it to do.

Release: The objective is to ensure that the developed system is successfully taken into use among the final users.

- No specific things for model-driven view in this phase.

3. Model-Driven Development Maturity Model

The Model-Driven Development Maturity Model was developed within the MODELWARE project. Rios et al. [2006] state that the role of the MDD Maturity Model is to assist in estimating the level of MDD technology implementation in an organisation. The MDD Maturity Model consists of five maturity levels that provide the characterisation of the degree of adoption and implementation of the MDD in an organisation. Each level has goals relating to the MDD practises and MDD elements.

According to Rios et al. [2006], the MDD Maturity Model defines three categories of MDD practices:

- **Engineering practices** cover the development activities in model-driven SW engineering.
- **Project management practices** cover such activities that relate to management decisions required to set up and manage an MDD project.
- **Support practices** cover the support activities related to engineering and project management practices.

According to Rios et al. [2006], the MDD Maturity Model identifies the following elements, which are the basic artefacts used in MDD technology:

- **Models** are abstractions of something in the real world and they capture the essential characteristics. A model can be a “Domain metamodel”, “Architecture-centric metamodel”, “Domain model”, “Business model”, “Technical model”, or “Code”.
- **Model transformations** are mechanisms for converting a model into another model.
- **Modelling tools** are tools that are used in activities related to modelling.
- **Documentation** is the set of text documents, in which the development process and generated assets are described.

Rios et al. [2006] depicts, that the MDD Maturity Model is a staged model and consists of five maturity levels. Each level of maturity characterises the MDD elements that are

expected to be seen. Table 2 lists the maturity levels and their descriptions as in [Modelware, 2006]:

Table 2. Levels of the MDD Maturity Model [adopted from Modelware, 2006].

Level of Maturity	**Description**
Level 1 – Ad-Hoc Modelling	Engineering is not model-driven.
Level 2 – Basic MDD: Implementation level models	Basic usage of models in an organisation.
Level 3 – Initial MDD: Abstraction away from the implementation level	The organisation starts developing systems more according to model model-driven approach.
Level 4 – Integrated MDD: Integrated chain of models	Models at different abstraction levels are built and they are fully integrated within a comprehensive modelling framework.
Level 5 – Ultimate MDD: Software factories	The role of coding will disappear and the transformations between models are automated.

4. MDD tools

The MDD uses models to represent software artefacts in various aspects and produces both platform independent and platform specific models. Tools play an important role in supporting the practical MDD and at its best they will reduce development costs and man-made errors. Currently, there is no universal tool that could satisfy all the needs in the MDD and, in practice, the MDD tool environment is a collection of separate tools. The available tools are plentiful, and we will not list them here. Instead, we will focus on the role of the tools in the MDD and the requirements for them. We also present some experiences from using the tools. One example categorisation of the MDA tools can be found from Wikipedia (http://en.wikipedia.org/wiki/Model-driven_architecture) that mentions that an MDA tool can be one or more of the following types:

- **Creation tool** that is used to elicit initial models and/or edit derived models.
- **Analysis tool** that is used to check models for completeness, inconsistencies, or error and warning conditions, as well as to calculate metrics for the model.
- **Transformation tool** that is used to transform models into other models or into code and documentation.
- **Composition tool** that is used to compose (i.e. to merge, according to a given composition semantics) several source models.
- **Test tool** that is used to test models.
- **Simulation tool** that is used to simulate the execution of a system represented by a given model.
- **Metadata management tool** that is used to handle the general relations between different models, metadata on each model and the mutual relations between these models.
- **Reverse engineering tool** that is used to transform particular legacy or information artefact portfolios into full-fledged models.

Link et al. [2008] claim that a continuous tool supported transformation process from model to source code is not yet possible. Vogel and Mantell [2005] mention that during their MDD implementation, immature tools caused the “debugging” of tools itself and there were also some challenges with the interoperability of tools and the standards they used.

According to France and Rumpe [2007], there is a need for tools which support the tracking of relationships between various model elements. A lack of tracking support makes the MDD more complex. There is also a need for a repository that is able to store the models produced by various development tools. France and Rumpe [2007] point out that developing such a repository will face difficult technical problems. They also mention that the current code generator tools do not provide enough support for the integration of foreign code.

France and Rumpe [2007] highlight the importance of quality of domain specific language (DSL) tools and state it to be the primary concern for them. Because DSL tool developers need a profound knowledge of the domain, France and Rumpe [2007] see that DSL tool developers should closely interact with application developers. They propose that the quality assurance programs of these two parties should be more integrated.

Kleppe et al. [2003] list the requirements for the MDD tools. These tools should:

- Allow developers to choose between different development platforms and within one platform, allow them to choose different implementation strategies, architectures and coding standards.
- Allow the integration of domain specific information into the models.
- Be able to support new modelling languages and target language transformations.
- Integrate with the version control and requirement management tools, as well as, with testing suites etc.

MacDonald et al. [2005] completes the previous list by adding the following requirements:

- The tool should fully support the target language.
- The tool should offer rich platform independent libraries.
- The tool should support an intermediate language for text diagrams.
- The tool should make the integration of the model with legacy systems easier.

In general, MDA based development needs tool support for success. According to Aagedal and Solheim [2004], the vision of the MDA is to provide a set of integrated tools to support the development of models and executable code. They highlight, that tools should support the synchronisation of codes and models, different model views and levels of abstraction, as well as the transformation of models and generation of codes. Aagedal and Solheim [2004] mention a model editor, model repository, model

transformers, model analysis tools and model simulator as tools that are required to support the activities of the roles that they identified as important for the MDE.

Schätz et al. [2003] compare the results of applying eight different tools for the development of embedded software to a common problem: the specification of a software module controlling comfort electronic functionality. The applied tools are:

- ARTiSAN RealTime Studio by Artisan Software
- ASCET-SD by ETAS GmbH & Co.KG
- AutoFOCUS by Technische Universität München
- MATLAB/StateFlow by The MathWorks Inc.
- Rose RealTime by Rational (now owned by IBM)
- Rhapsody in MicroC by I-Logix Inc. (now owned by IBM)
- Telelogic Tau by Telelogic Inc. (now owned by IBM)
- Trice Tool by Protos Software GmbH.

With each tool, a model of a controller software module was developed, based on a given textual requirement specification. The requirement specification of the controller was taken from a revised version of a controller specification provided by F. Houdek, Daimler Chrysler AG. These state-of-the art CASE tools for embedded systems offer support for:

- the use of graphical representations for the system under development
- the possibility to describe the system (or software) with a certain degree of abstraction from the actual implementation platform
- the possibility to generate an executable system out of the model
- describing the system using different hierarchic views
- using message- or signal-based communication in the operational model
- including timing aspects in the description of the system
- checking the model for inconsistencies, mainly on a syntactic level (e.g., undefined identifiers, type mismatches)
- simulating the system at the level of the description
- deploying the generated code to a host and a target system.

5. Experiences on MDD technologies

MacDonald et al. [2005] evaluated some of the claimed MDD advantages against their experiences in using the MDD in an industrial setting. According to their findings, there is no evidence about the improved development speed in the case where they had to integrate workarounds with legacy systems. They also experienced that there were no significant differences in the amount of bugs found in the generated code. The amount of bugs compared in the generated code was about the same as what would be expected with traditional development. According to MacDonald et al. [2005], the bugs were more difficult to find and fix in the model. They also reported that they succeeded in the automatic generation of code, as well as improving the communication by using the Tau Developer tool that allows different views of the models. MacDonald et al. [2005] found out that the time that is needed to study a new system was reduced because the design was more easily communicated to people.

Aagedal and Solheim [2004] emphasise that when MDD related processes and tools are introduced in a company, they quite often realise that new ways of working require some new skills in a company. They list a group of skills that can also be considered as roles that form a meta team in a company and that are used to define modelling languages, domain and platform concepts, as well as to customise tools. They identified the following roles for the meta team:

- domain and platform experts,
- language and method engineers, and
- transformation specifier.

In addition to a meta team, Aagedal and Solheim [2004] define the roles for a project team. This team performs the application development and its work is based on the foundations of the meta team. The following roles are defined for the project team:

- application designer
- system analyser, and
- system tester.

Staron [2006] presents a case study of two companies willing to adopt the principles of the MDD. One of the companies is in the process of adopting the MDD, while the other withdrew from its initial intentions. The conditions that were identified in the case study are as follows (prioritising according to importance):

1. **Maturity of modelling technology:** The technology used in the company should:
   - provide advanced features for developing and executing model transformations,
   - develop and execute model analysis methods (e.g. model measurements, test case generation, and the identification of problem areas), and
   - develop and introduce domain-specific modelling (e.g. using UML profiles, if the base language is UML).
2. **Maturity of modelling related methods:** In addition to the modelling tools, the methods for using models should be mature. These methods include activities such as:
   - early model-based verification and validation,
   - model-based quality assurance,
   - model-based project planning and management (development efforts are different when using modelling, thus project managers need to know how to structure their projects, taking that into consideration), etc.

MDD should be the next step in using models based on the evolution of software development processes.

3. **Process compatibility:** The process used in the company should be “compatible” with MDD principles. The compatibility means that
   - it should be possible to use the models effectively in the process without a complete redefinition of the process
   - there should be room in the process to actually use models as primary artefacts in the process – i.e. quality assessment should be performed on models, estimations should be performed on models, testing should be performed on models.

This condition is especially important for large companies, as in such companies the software process improvement activities require a great effort.

4. **Core language-engineering expertise:** Since the advanced model usage scenarios require either customising a modelling language or engineering (at least to some extent) a new language, there should be staff available in the company with the required expertise. The required expertise is a combination of: domain knowledge (to the most extent), tool building/extending methods, and language engineering (to some extent, related to the expertise in tool building/extending).

5. **Goal-driven adoption process:** A set of precisely defined goals for introducing the MDD should be in place in the company – e.g. to improve the quality of data modelling or improve the productivity of development of a given part of software. The scope and the methods for introducing the MDD should be defined before the adoption, in order to decrease the risk of changing the scope of the MDD framework and thus providing better control of the costs in the projects.

Foustok [2007] mentions that using the UML in a global environment may improve the human communication and lower the language barriers that the different parties may face during the project. He reported that the use of the MDD together with product-line engineering and an asset repository indicate some significance in quality and productivity. During a two years period, they have not found any defects that had been caused by the common components that they produced. Foustok [2007] states that when the amount of binary generated through a model-driven was compared to traditional development, the productivity was increased 70%.

Vogel and Mantell [2006] introduce a set of metrics they defined as a potential for measuring the success of MDA implementation in a company. As a primary measure, they mention "**Effort**", since all the others may be derived from it. The list of measures they mentioned is as follows:

- **Effort**: Man days spent in software development.
- **Cost**: The function of effort multiplied the resource rate.
- **Duration**: A measure that can be manipulated by introducing more resources into the scene.
- **Tools maturity**: The time that the user was idle, because of bugs in the tool.
- **Learning curve**: How fast people have learned the model-driven tools and methods. This was measured by surveying the attendees after each course.
- **Resistance to change**: By surveying attitudes, it was measured how willing people were to start using the MDD.
- **Perceived value of using the MDA**: By surveying attitudes, the perceived advantages, constraints of using the MDA were also measured.

Vogel and Mantell [2006] also collected information on "**Job function**", "**Job tenure**", "**Professional tenure**", and "**Highest completed education**" to collect the demographic data.

Afonso et al. [2006] presents a case study that they conducted in a systems integration company. Their study consisted of three complementary studies, namely "Software Development", "Software Maintenance", and "MDD Education" studies. The goal of their research was to understand the impacts and added value of infusing the MDD in the software development process of a systems integration company. Their finding was that the MDD infusion had a positive impact on the performance of software development.

They reported an improvement in the communication of specifications, supporting a focus on business knowledge, and reducing the number of defects, as well as the effort required to develop a given feature. When compared to traditional development methods, Afonso et al. [2006] highlight that the MDD means a transfer of effort from development to analysis and specification. One specific finding from their research was that the learning curve of the people that were involved with the study reduced the potential return of investment of introducing the MDD in the development process.

Afonso et al. [2006] also mention some constraints to the successful MDD infusion in a company. The cost of the tools may have a large impact on their adoption. It also influences the price of the final solution to the customer. They also mention organisational experience with modelling technique as a constraint to successful infusion. A longer learning curve may have an impact that delays achieving the advantages of MDD introduction in a company. Based on the findings from the studies, Afonso et al. [2006] propose strategies for successful infusion. These strategies are: “Provide clear guidance from the organisations leaders”, “Use an incremental approach starting in areas where success is easier and visible for the organisation”, “Select the appropriate tools”, and “Begin involving people with some modelling experience and in parallel, train other organisational actors”.

Middleware [2003] conducted a case study to prove or disprove the productivity claims related to MDA adoption. Their study consisted of two teams that developed the same J2EE application. One team used the MDA-based tool and other team used traditional integrated development environment. The study revealed that the team that used the MDA-based tool produced their application 30% faster than the team with a traditional development environment. Therefore, their claim is that the MDA improves the productivity and they encourage companies to evaluate the MDA based tools, if they are seeking improvements in productivity. Middleware [2003] mentions that the MDA is best utilised with capable and experienced architects, but there is still a need for developers that know J2EE patterns and best practises, object-oriented development and architectural tradeoffs. One observation was that in traditional development, quite a lot of time was spent in testing and finding bugs, but in the MDA-based development, there was no need for such extra activities as the code was automatically generated via patterns.

Mattson et al. [2007] report their experiences from architectural work practices using the MDD in a large industrial project. They state that the current methods and tools for MDD do not support the formal representation of architectural rules. Therefore, manual interpretations and reviews are needed and due to the manual routines, it is error prone and requires lots of effort. Another observation is that because architectural rules are introduced manually through the models, it is difficult to make late changes to them. Mattson et al. [2007] summarise that from their experience there is a “need for the support of formal modelling of architectural rules and automatic enforcement of these rules on the generated models of the system.”

6. Challenges and success factors for MDD adoption

Uhl [2008] presents challenges related to model-driven approaches.

- Life-cycle issues: Modelling often implies the managing of partly redundant artefacts for concrete and abstract syntax. Tool users must store and control versions of a model diagram's layout information, just as they do for the model's elements. Understanding the interrelations among the artefacts can sometimes be tricky, and some of these relations can cause surprises – particularly for casual users.
- Model comparison and merging: All projects, not just enterprise set-ups, keep versions of development artefacts. Versioning makes the capability to compare models essential. Comparing texts is well understood. However, models can include graphical views, forms, dialogs, and property sheets, as well as text. They are much more difficult to compare, mostly because visualising the differences in a usable way is difficult.
- Model transformations: Transforming models into code or other models once is well understood, and all the MDD tools and frameworks support it. Tools vary in how much support they offer for specifying transformations – for example, in debuggers and smart editors. The difficulties with transformations arise in three areas: Refining the results (e.g., by adding code in protected regions), referencing results from within other models and only transforming small pieces of the model quickly after small incremental changes.

  Only a few tools and frameworks address these basic difficulties. Tool and transformation designers can work around the challenge of preserving manual refinements across transformation runs by attaching the refinements to the input model. Such attachments are called marks in the OMG's Model-Driven Architecture. Marks preserve manual refinements and also solve several life-cycle issues as well, but refactoring the marks' content remains difficult. Furthermore, editors typically only have a restricted support for marks, such as generic text

fields, which don't compare to any reasonable integrated development environment's usability standards. Referenceable output models require additional measures that let the references use alternative keys instead of or in addition to those based on the Universally Unique Identifier (UUID) standard. Alternatively, they can use a repository that lets the model transformation framework produce stable UUIDs for the model elements that it produces. SAP has implemented prototypes indicating that such stable UUIDs solve the problem. However, the long-term ramifications of such an approach aren't well understood yet.

- Model-level debugging: When a tool user expresses an application's behavioural aspects in models and transforms those models into runtime artefacts, debugging becomes an issue. Only tools that are very specialised, support model-level debugging, typically only for a specific modelling language. A model transformation can keep a traceability record of which model elements mapped to which elements in the runtime. Debuggers can then use this information to suspend, introspect, and continue the application execution on the basis of locations in the model.
- Models and text-based syntaxes: Text syntaxes are a valid way of viewing and editing models. Many developers prefer them to graphical editors. For example, parsing Object Constraint Language text or syntaxes derived using the Human Usable Text Notation into model repositories gives some great querying, browsing, and refactoring capabilities. Sometimes, a tool or framework manages the underlying models by regarding them as cached, derived forms rather than development artefacts. Eclipse's Java Development Tools work this way. This approach preserves all the benefits of regular text editors. For example, the tool keeps all of the lexical information that the user entered (indentation, comments, and so on) and saves inconsistent or unparsable texts. Users can copy and paste arbitrary text sections and not just the valid sub-trees of the concrete syntax tree. The Text Editing Framework (TEF) and openArchitectureWare's Xtext follow a similar approach. Problems occur when trying to put such a parser-based approach on top of a UUID-based repository. The parser can't easily identify UUID changes, particularly if the element names have changed in the text. Parsing the text again might create elements with the new UUIDs, thereby breaking old references.

  At the other end of the spectrum, there are approaches such as Intentional Software (www.intentsoft.com), where the tools can combine a variety of syntaxes, including text-based ones, even in a single editor. The editor applies modifications directly to the underlying model, and the modifications immediately affect all the other views. Intelligent parsing technology ensures that users can still syntactically save incorrect stretches of text and that editing the model will feel like editing a text document. This approach lets you combine text-based syntaxes

with repositories that use UUIDs. However, these capabilities aren't yet widely available. Since tool customers are reluctant to get locked into proprietary solutions, the powerful combination of graphical and forms-based syntaxes with text views hasn't yet seen widespread adoption.

Hailpern and Tarr [2006] also discuss several challenges in model-driven development:

- Redundancy: There are multiple representations of software development artefacts representing different views or levels of abstraction of the same concepts. To the extent that these are manually created duplicate work and consistency management are required.
- Rampant round-trip[1] problems: The more models and levels of abstraction that are associated with any given software system, the more relationships will exist among those models. Many of these interrelationships are complex. The round-trip problem occurs whenever an interrelated artefact changes in ways that affect some or all of its related artefacts, as the mutual consistency cannot always be automatically assured.
- Moving complexity rather than reducing it: As with any development technique or technology, one must determine whether a given MDD approach reduces the complexity visible to the developer or whether it simply moves the complexity elsewhere in the development process. As the number of artefacts increases, the number – and potentially the complexity- of artefact relationships increases, as does the complexity of the tools that manipulate and visualise them. It remains to be seen if people have an easier time managing a relatively small number of large artefacts with fewer relationships or if they manage better with a large number of more specialised artefacts with a correspondingly greater number of relationships.
- More expertise required: The interrelationships between multiple types of models, and potentially, different modelling formalisms, suggests that it will be difficult for any given stakeholder (e.g., use case developer, architect, tester), to understand the impact of a proposed change on all of the related artefacts that could be described in different notations. Different MDD models can aid in communication between distributed sub-teams, but it also implies that the different sub-teams cannot be experts in only their own development genre, but knowledge of different model technologies and terminologies must exist at each site.

[1] Round-trip engineering is a functionality of software development tools, which provides a generation of models from source code and a generation of source code from models; this way, existing source code can be converted into a model, be subjected to software engineering methods and then be converted back.

Six requirements were identified as critical for a successful MDD infusion, within the MDD toolset provided [Afonso et al., 2006]:

- Traceability ("Whole lifecycle support"): tracking the relationships among all the artefacts. This allows, for instance, impact analysis when a change occurs, ensuring the traceability of the changes, guaranteeing consistency among the artefacts and improving reactivity.
- Round trip: required to trace back from code to model, which can be useful, to reverse engineering or to synchronise the model with code changes.
- Automatic code generation: should support automatic generation, from the model, of readable documented platform specific code, allowing the early identification of requirements inconsistencies, missing system components or unexpected development needs.
- Automatic document generation: should provide automatic generation of the documentation in standard formats, meeting the quality required for formal sign-off processes. This contributes to easier/faster consistency among project artefacts.
- Automatic quality assurance: should support simulation and automatic generation of multilevel testing (unit, system and integration testing) for market standard frameworks, such as TPTP for instance.
- Interoperability among the tools: the tools should follow OMG standards (e.g. UML 2.0, OCL, XMI) and be able to be interoperate.

Table 3 summaries of the challenges of the MDD adoption.

Table 3. Summary of the challenges.

Challenge	**Description**
Understanding and managing the interrelations among partly redundant artefacts.	There are multiple representations of software development artefacts representing different views or levels of abstraction of the same concepts. To the extent that these are manually created, duplicate work and consistency management are required.
Comparing and merging different versions of models.	Models are much more difficult to compare, mostly because visualising the differences in a usable way is difficult.
Difficulties with the transformations of models (to code or other models). Rampant round-trip problems.	The more models and levels of abstraction that are associated with any given software system, the more relationships will exist among those models. Many of these interrelationships are complex. The round-trip problem occurs whenever an interrelated artefact changes in ways that affect some or all of its related artefacts, as mutual consistency cannot always be automatically assured.
Model-level debugging is not supported by tools.	When a tool user expresses an application's behavioural aspects in models and transforms those models into runtime artefacts, debugging becomes an issue.
A combination of graphical and form-based syntaxes with text views is not well supported.	Text syntaxes are a valid way to view and edit models. Many developers prefer them to graphical editors.
Moving complexity rather than reducing it.	As with any development technique or technology, one must determine whether a given MDD approach reduces the complexity visible to the developer or whether it simply moves the complexity elsewhere in the development process. It remains to be seen if people have an easier time managing a relatively small number of large artefacts with fewer relationships or if they manage better with a large number of more specialised artefacts with a correspondingly greater number of relationships.
More expertise required.	The interrelationships between multiple types of models, and potentially, different modelling formalisms, suggests that it will be difficult for any given stakeholder (e.g., use case developer, architect, tester), to understand the impact of a proposed change in all of the related artefacts that could be described in different notations.

7. 1st ITEA MoSiS survey: MDD state of the art and practise

This chapter presents the results of a survey that was conducted in the work package 3 of ITEA MoSiS project. The focus in WP3 was on disclosing and evaluating the effects of the model-driven development (MDD) on the existing processes and product lifecycle. Another focus of the WP3 was on providing mechanisms to support a smooth evolution from code-centric approaches towards more model-driven approaches and on exploring the problem from process and organisational points of view. To discover the state-of-practise of model-driven development and processes in organisations, the survey was conducted. The survey was carried out during February, 2008 and it was organised by VTT Technical Research Centre of Finland.

7.1 Goal of the survey

The goal of the survey was to find out the current state of modelling practices and processes in industry and academia. It was also an intention to find out what kind of aims, needs and possible problem areas companies have when applying the MDD.

The research framework for the survey is the Model-Driven Development Maturity Model that is described in Section 3. This framework was selected because the model has been developed to complement the existing models such as Capability Maturity Model® Integration [CMMI Web page, 2009], ISO requirements for Quality Management system [ISO9001:2000 Web page, 2009] and European Foundation for Quality Management [EFQM Web page, 2009] by putting the focus on specific model-driven engineering activities. The model consists of three dimensions: technological capability, organisational maturity and capitalisation capability. Each of which consists of 5 maturity levels. A maturity level represents the level of adoption of the MDD in organisations. Each maturity level has particular goals associated to it, which are used to determine whether it is completely achieved.

Based on the Maturity Model, the questions in the survey were categorised in three dimensions: technological capability, organisational maturity and capitalisation capability.

By asking questions about the technological capability, the intention was to find out a basis for understanding and improving an organisation's range of "MDD technical means for engineering the system/software". Questions concerning organisational maturity were focused on the practices that have to be put in place in order to determine whether an organisation has achieved a certain level of MDD-maturity. Domain capitalisation questions address the level of technological or domain capitalisation achieved in the organisation.

In addition, experiences and background related questions were also asked. The following is the list of question groups in the survey:

- Group A: The background of the respondent
- Group B: Personal experiences of the MDD
- Group C: Modelling processes and practices of the MDD (organisational maturity)
- Group D: Software reuse and core assets in organisations (capitalisation capability)
- Group E: MDD tools and methods in the organisations (technological capability)
- Group F: Opinions about the future of modelling activities.

This publication summarises Group A: The background of the respondents' questions as well as the questions related to the MDD processes and technologies from Group B: Personal experiences of the MDD, Group C: Modelling processes and practices of the MDD, and Group E: MDD tools and methods in organisations. The complete list of questions of each of the groups can be found on Appendix A.

7.2 Set-up of the survey

The survey was prepared as an electronic questionnaire that was available on the web. The questionnaire contained 41 questions, altogether. However, groups of the questions were always filtered out, depending on how the respondent had answered the earlier questions. If the respondent did not have any experience of the MDD, the questionnaire was very short, just a few questions were asked from the respondent. This publication presents the results from 19 questions that are most relevant from the MDD process and technology point of views.

The survey was available to respondents during 25th of February – 5th of March. The link to the survey was emailed to the MoSiS partners, which were from Spain, Sweden, Norway and Finland. The link was also emailed to country co-ordinators of MoSiS, who were asked to distribute the link forward to suitable mailing lists within their country. The link was also available to people who had been in close contact with VTT; they had participated in seminars at VTT or they were customers of VTT. All the respondents of the survey were recorded as anonymous.

7.3 Limitations of the survey

This survey offered qualitative results for studying the state-of-practise of the MDD in organisations. Respondents were not named beforehand and the link to the research was distributed freely in different organisations.

The limitation of the survey is that the respondents might be unbalance in a way which people having a lot of experience of the MDD answered the survey more actively than people who were not familiar with the MDD. In that sense, the questionnaire did not give a realistic picture of how much or how extensively organisations are really using the MDD in their product development. The number of the respondents was also relatively small, thus the results of this survey are not scientifically significant but they still indicate where the world is with the MDD today.

8. Results of the 1st MoSiS survey

This chapter documents the results from the MoSiS survey. Altogether, the number of respondents totalled 69, the respondents gave their responses only to those questions that they found relevant in their position and organisation. Therefore, the number of responses varies between the questions.

8.1 Group A: Background of the respondent

Questions in this group collected the basic information of the respondents and their organisations.

8.1.1 The profile of the respondents (age and role in organisation)

Participants of the survey were from both industry and academia; however, most of the respondents were researchers (see Figure 9). Architects/designers and project managers were also represented well There were also other roles mentioned in the results of the survey. These were Ph.D. students and teachers. The age of most of the respondents was between 30–39 years.

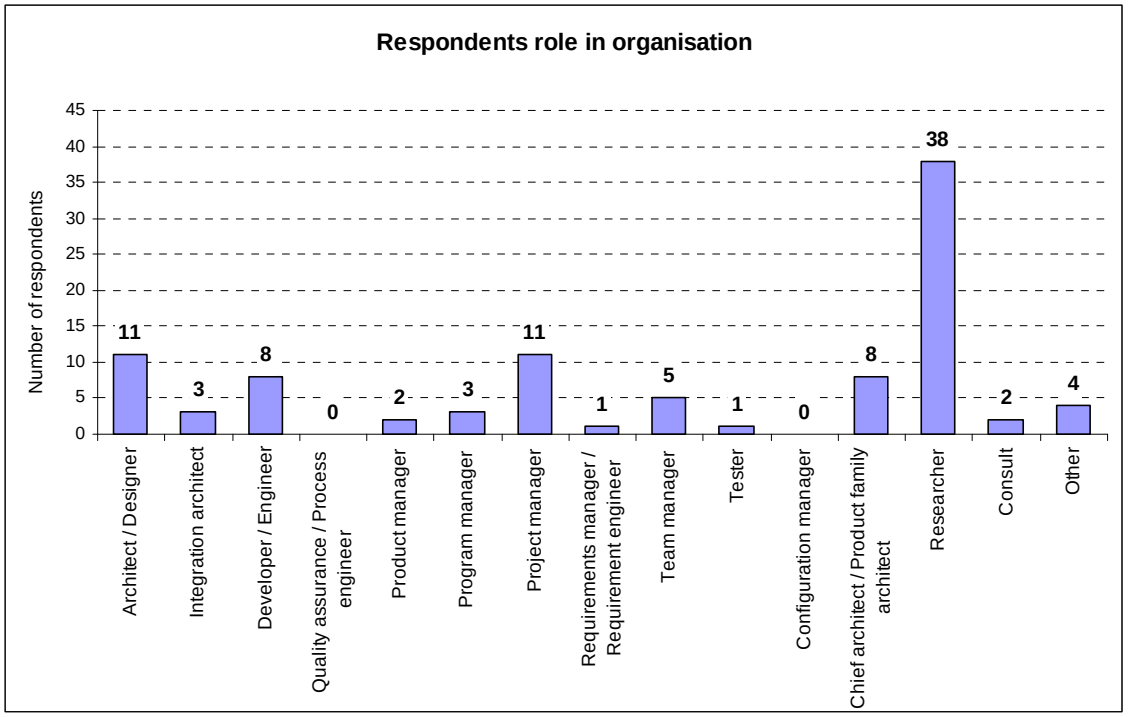

Figure 9. The roles of the respondents in their organisations.

8.1.2 Respondents and domains

Most of the respondents (see Figure 10) represented research organisation domain (29). Software products (10), embedded software (7) and information systems domains (7) were also represented quite well. 4 persons were from the telecommunication domain and one person represented both industrial automation and the other domain (research on embedded software and hardware).

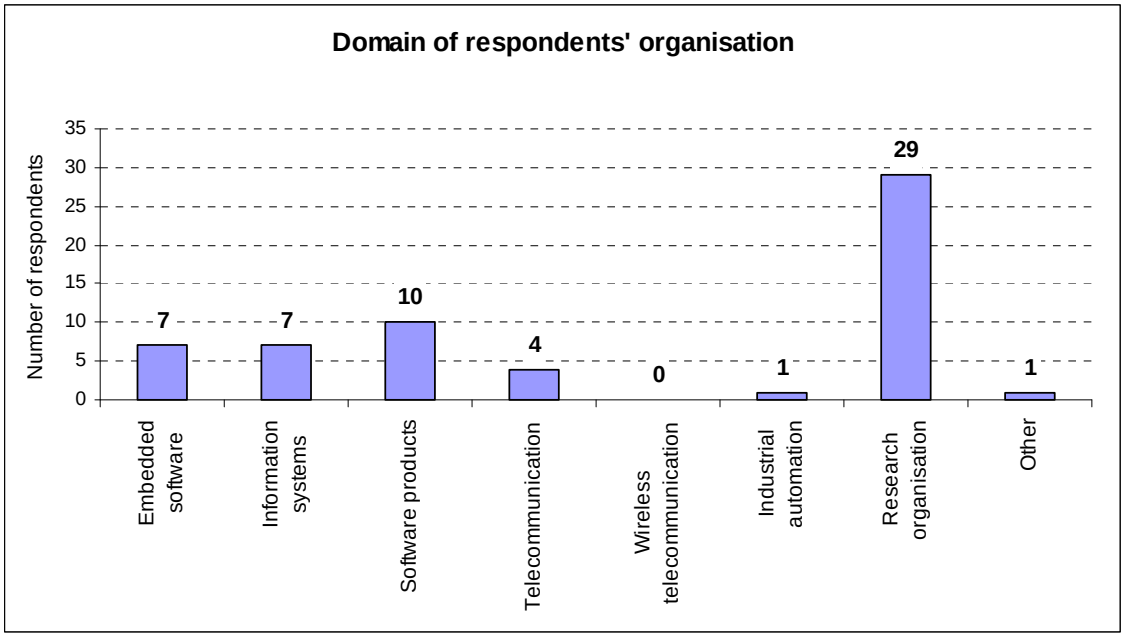

Figure 10. The domain of the organisations.

8.1.3 Size of the organisation

The number of personnel in the organisations varies a lot between the respondents (see Figure 11). There were small, middle sized and large companies involved in the survey. Most respondents (14) were from companies having 50–249 employees.

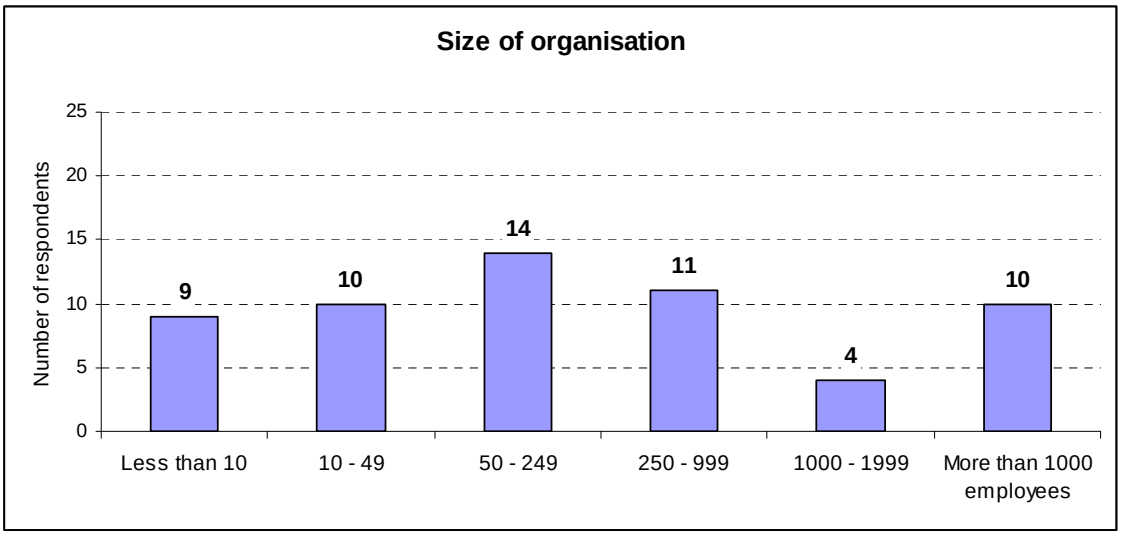

Figure 11. The number of personnel in the organisations.

8.1.4 Products and services

Typically, the products (see Figure 12) of the organisation that respondents were from were SW products (26), or something other than SW, HW products, or consulting services, (15). 13 respondents mentioned that their products were consultation services, 7 persons represented organisations which produced HW-SW products.

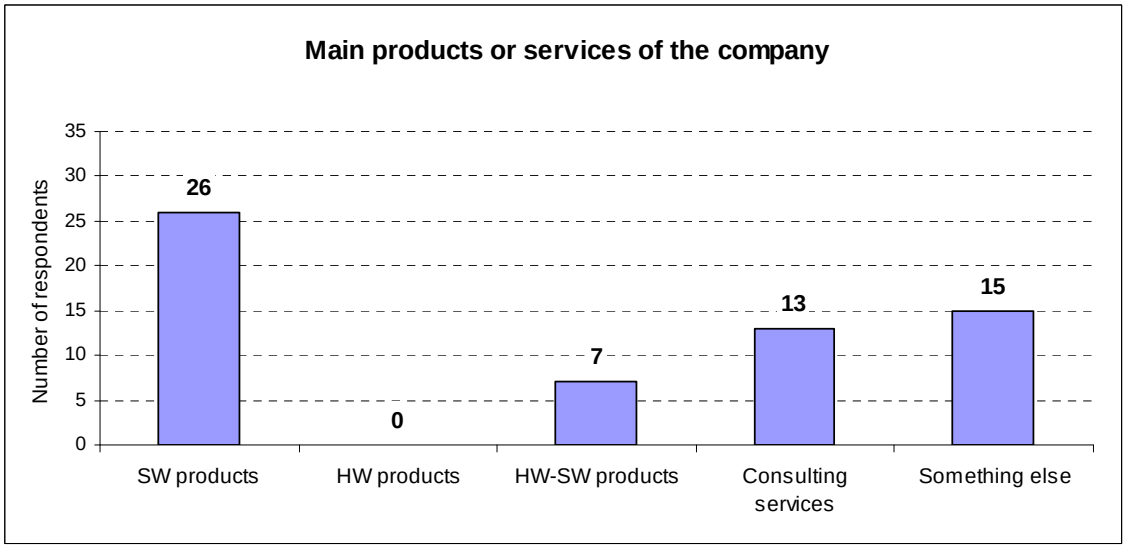

Figure 12. Main products or services of the company.

8.2 Group B: Personal experience of the MDD

Questions in this group clarified the persons experience level related to the MDD.

In the beginning of the survey, two questions were asked about the respondent's familiarity with the MDD. This question classified the respondents, so that if the respondent were not familiar with the MDD at all, then most of the following questions were filtered out. If the respondent had knowledge of the MDD, more specific questions were then asked within the questionnaire.

8.2.1 Familiarity with the MDD

As illustrated in Figure 13, most of the respondents used the MDD in their work (31), 16 respondents were familiar with the MDD artefacts, 10 respondents used the UML models in their work, 4 persons considered their knowledge of the MDD as the same level that they know the UML. Only 1 person was not familiar with software modelling at all.

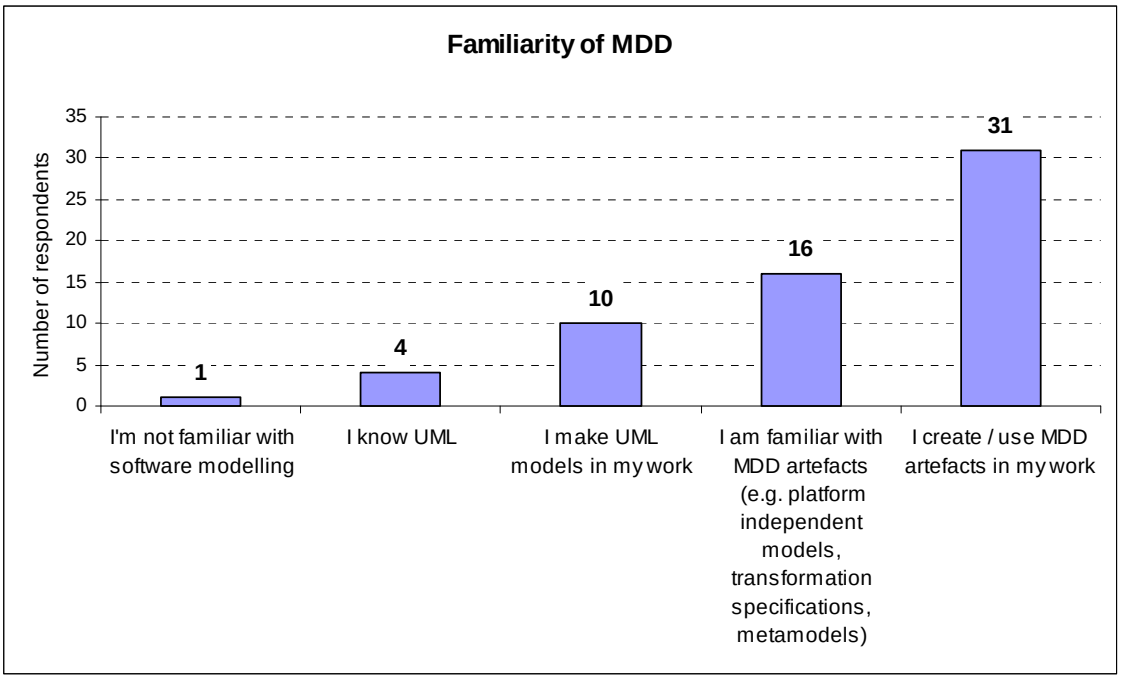

Figure 13. Familiarity of the MDD.

8.2.2 Usefulness of the MDD

Respondents were asked to specify how useful they considered the MDD to be. In the scale from 0 to 100, the higher number represented the higher level of usefulness. Responses varied between 30 and 100 and the average for all the responses was 81 (see Figure 14). According to the responses, the potential of the MDD is recognised and is considered to be useful.

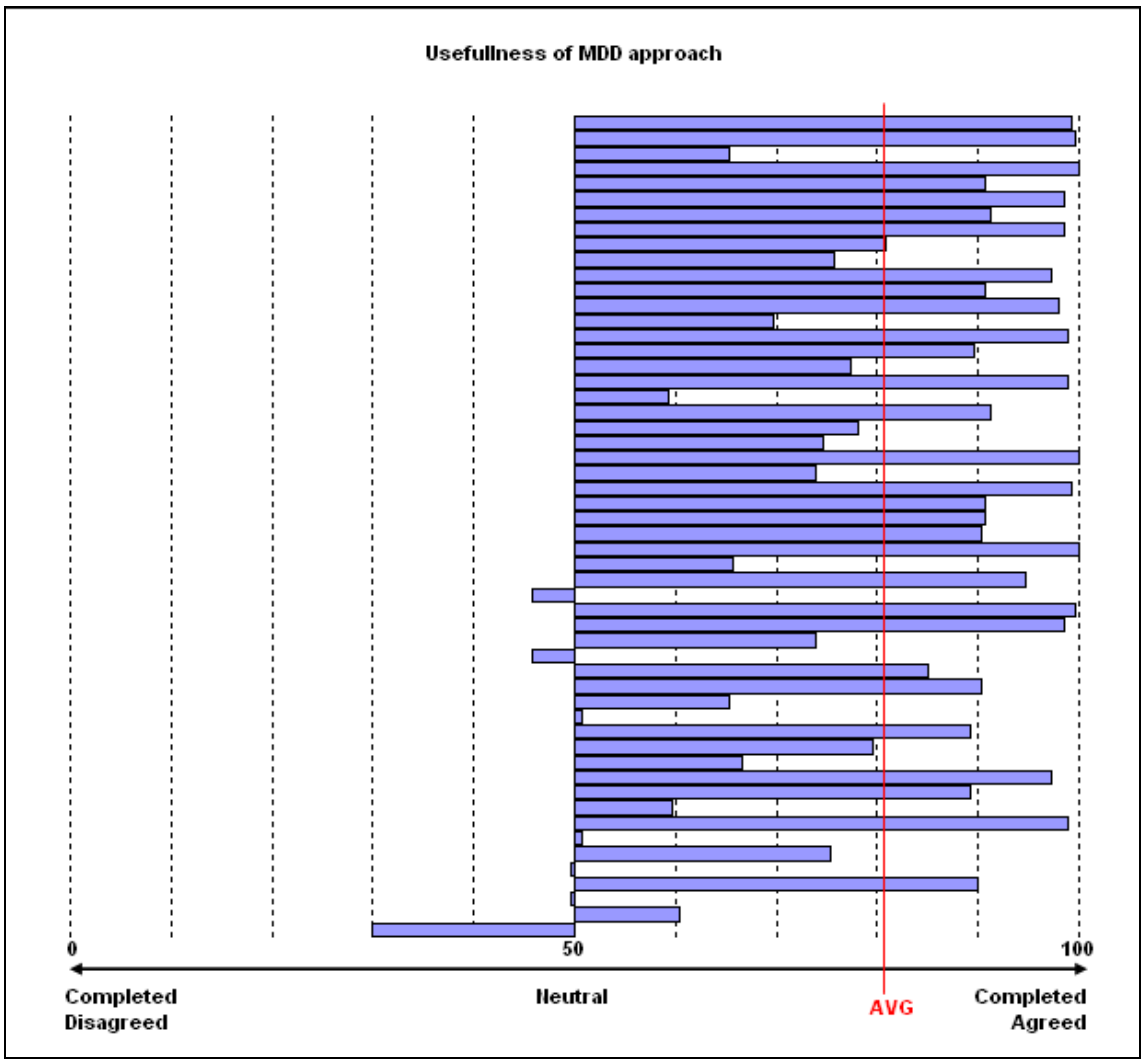

Figure 14. The usefulness of the MDD.

8.3 Group C: Modelling processes and practices of the MDD

This group of questions in the survey was to study the MDD related processes and practices in organisations. The questions will clarify how software processes were defined in organisations and also, if the MDD was not applied, it was asked what the main reasons were for that. The respondent's opinion about the challenges, benefits and experiences of MDD related issues were also requested.

8.3.1 Extent of software process definitions

Respondents were asked about the extent of software development process definitions in their organisations. Most of the respondents (see Figure 15) consider that the software development processes are defined largely in their organisation, meaning that most of the phases of development have been defined. There were eight respondents who answered that there were no process definitions or guidelines to support software development in their organisations. According to seven respondents, the processes are defined as well as some metrics are collected to evaluate the performance and quality in their organisation.

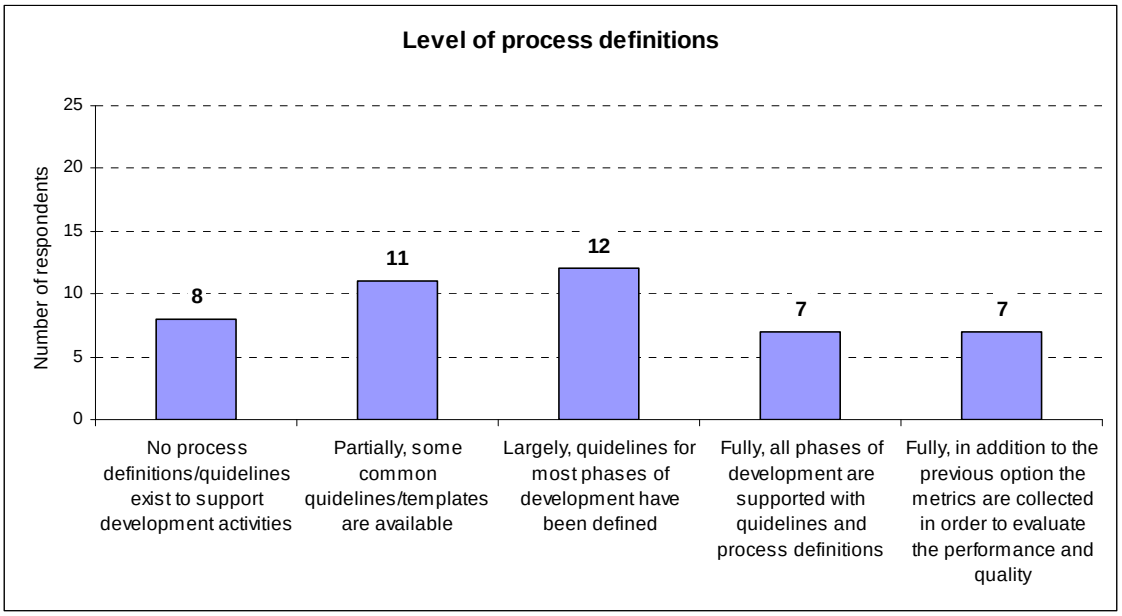

Figure 15. The extent of software development process definitions in a company.

8.3.2 Usage of the MDD

Respondents were asked how systematically the MDD related activities are applied in software projects (see Figure 16). According to the responses, the application of the MDD related activities depends largely on the interests and experiences of the project's personnel. 11 responses selected the option that all the projects apply some MDD activities, but they are not necessarily the same in every project. Three respondents answered that their organisations are not using the MDD. According to two respondents, their organisation applies the same and well defined MDD activities and provides metrics to support the evaluation of the performance and predictability of the MDD outcomes.

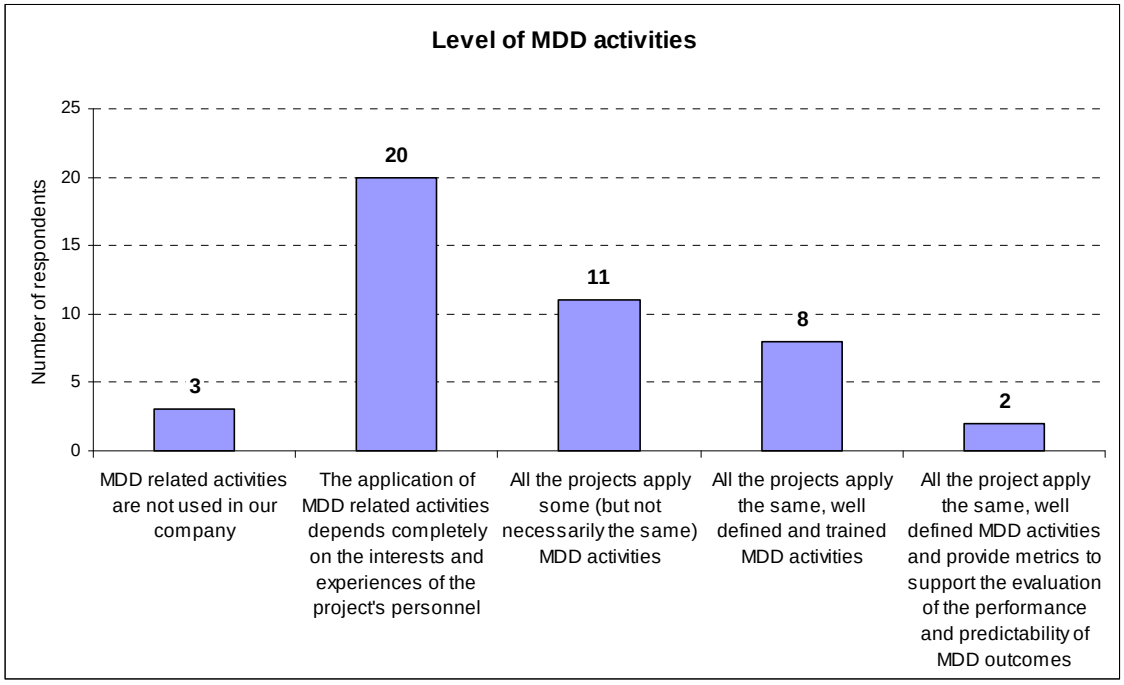

Figure 16. The usage of MDD in companies.

8.3.3 Reasons for not applying the MDD

Nine of the respondents (see Figure 17) consider that the reason for not applying the MDD in their organisation is that only a few persons are interested in applying MDD related practices. In five responses, respondents see that the MDD has not been considered as a part of the development process in their organisation and there are no proper tools and techniques that support the MDD. Four respondents identified the lack of training in MDD tools and techniques as a reason not to apply the MDD in their

organisation. Other reasons not to apply the MDD are listed here: “Not familiar enough with the MDD”, “The applicability of the MDD for modelling is highly heterogeneous, dynamic, distributed and middleware oriented systems are not fully clear yet. Particular support for modelling distribution related problems is unclear”, “Critical real-time performance & footprint goals”, and “Still evaluating”.

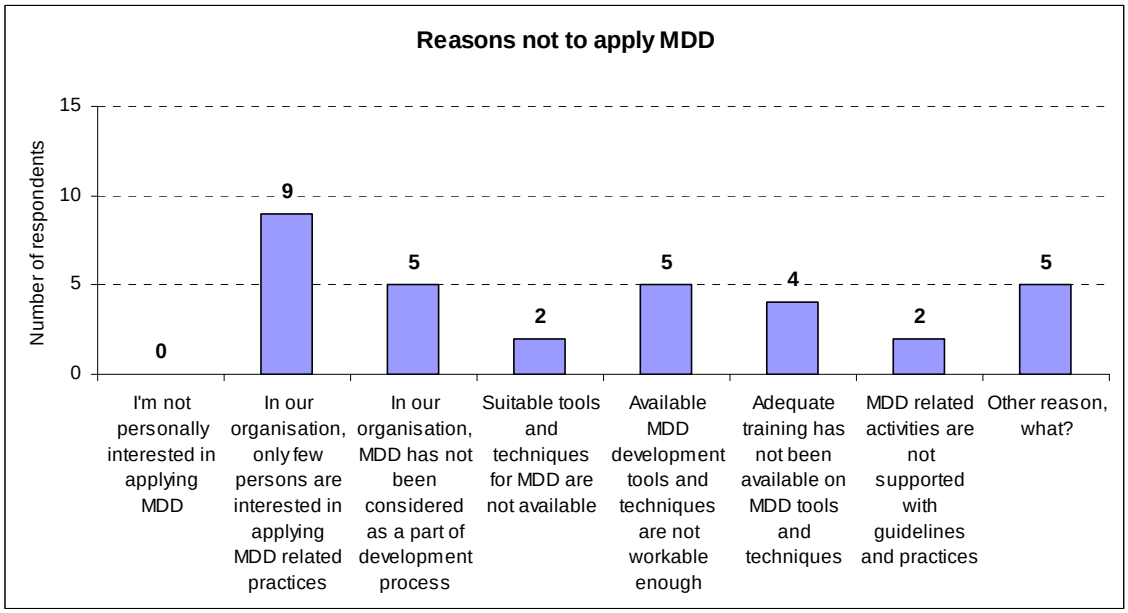

Figure 17. Reasons why the MDD is not applied in organisation.

8.3.4 MDD related challenges

The answers to this question were given in free form. Most responses mentioned that there are a lack of modelling experts in organisations, as well as it being difficult to select the correct languages and tools and make them collaborate. There were also difficulties in synchronising models and metamodels.

8.3.5 MDD related benefits

The answers to this question were given in free form. Most often, it was emphasised that the MDD improves productivity by improving SW reuse and by decreasing errors in the code. It was also mentioned that the MDD made it easier to read designs and improve communication. The MDD approach moves the development focus on the problem domain away from the actual coding work.

8.3.6 MDD improvement targets

Most of the respondents (see Figure 18) agreed that the tools and techniques related to the MDD need to be improved. 22 respondents consider that there is a need to develop MDD related processes, practices and guidelines. Training was mentioned by 21 respondents and standardisation by 15 respondents. As other development target following issues were mentioned: Requirement language, Integration of multiple models- Not one model provides an adequate view of the entire problem, Common Action Language as part of UML standard (not Profile), Incremental approaches are in most cases the way to proceed, Case studies and empirical results, and Support for architectural work practices.

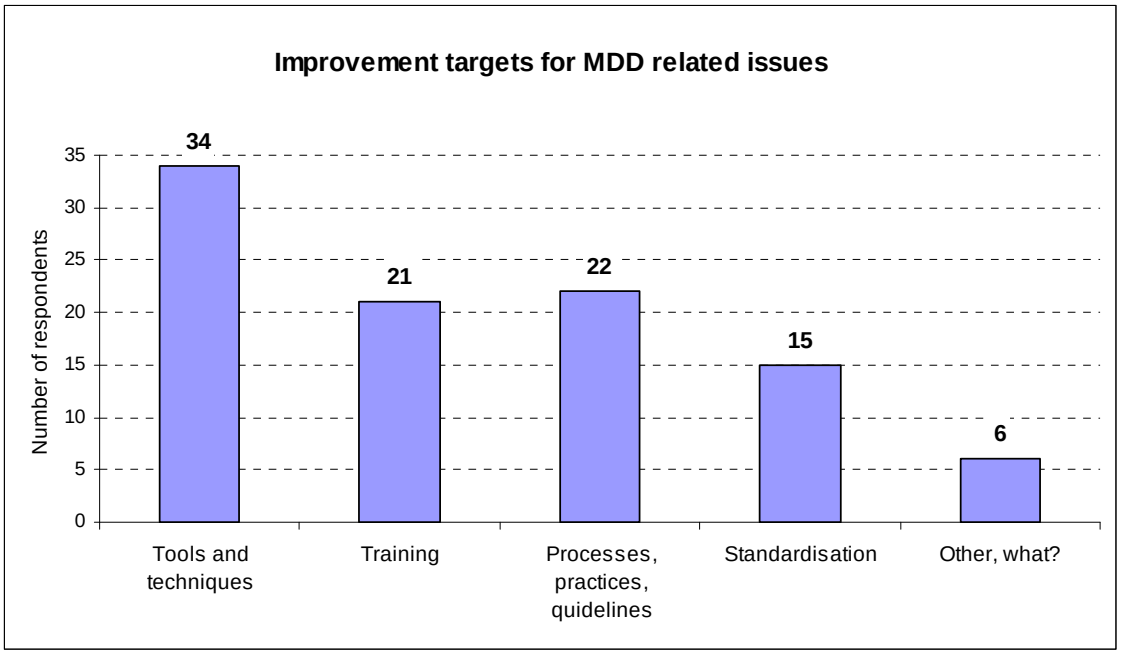

Figure 18. Improvement targets for MDD related issues.

8.4 Group E: MDD tools and methods in organisations

Questions in this group studied the MDD related tools and methods used in organisations.

8.4.1 Software modelling languages

According to the survey (see Figure 19), the Unified Modelling Language (UML) seems to be the most popular modelling language. In addition to UML, other general purpose languages besides the UML, such as XML, MOF and architecture description languages

are also used. The boundary between a general purpose (modelling) language and a domain-specific (modelling) language seems to be quite vague based on the survey.

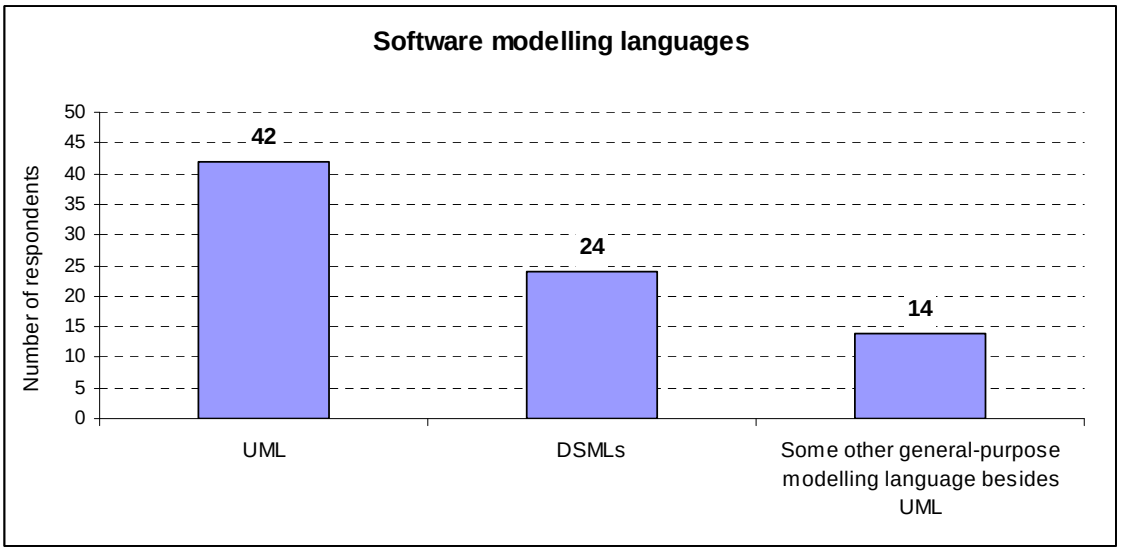

Figure 19. Software modelling languages and their usage.

Furthermore, in Figure 20, we can see the usage of code generation with the UML, DSML or other languages. According to the respondent, code generation is often used regardless of the language that is in use.

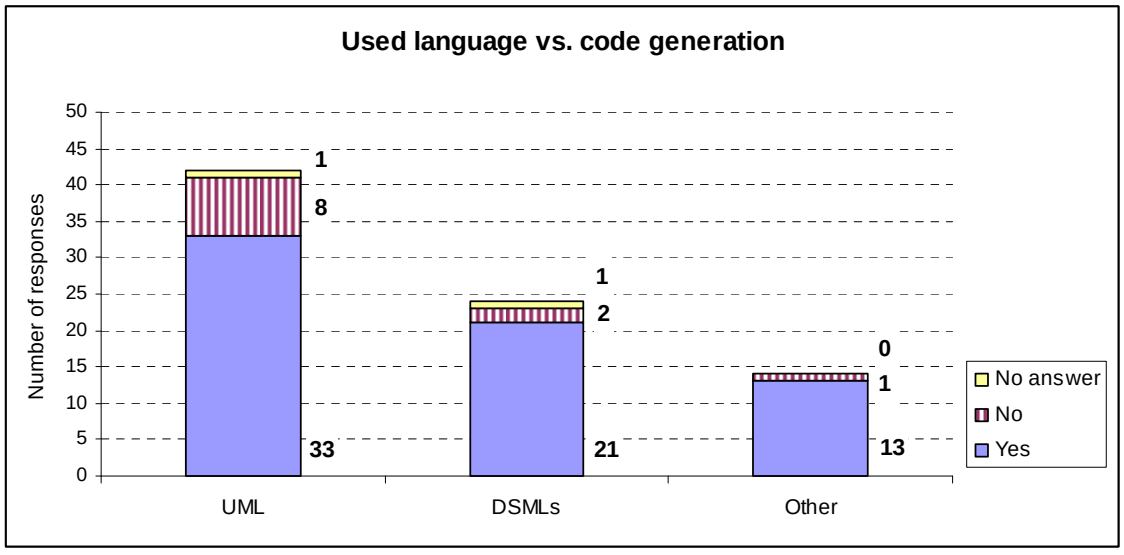

Figure 20. Usage of code generation vs. modelling languages.

8.4.2 Usage of code generation

44 respondents (see Figure 21) answered that they use code generation from software models and ten respondents answered that they do not generate code from models.

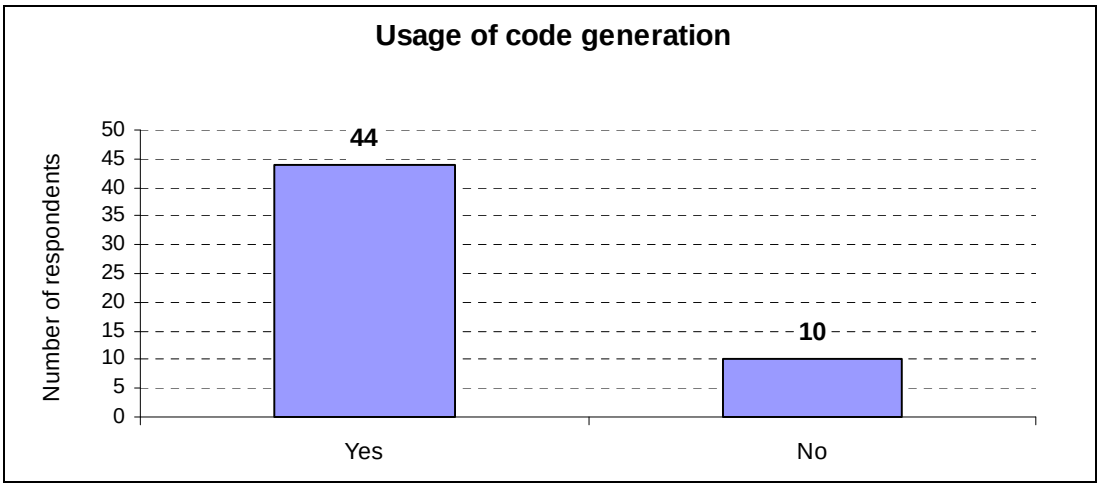

Figure 21. Usage of code generation.

8.4.3 The purpose of using UML

In addition to documentation and code generation from implementation models, UML is used for many purposes e.g. designs, analysis, reverse engineering, and transformations (see Figure 22).

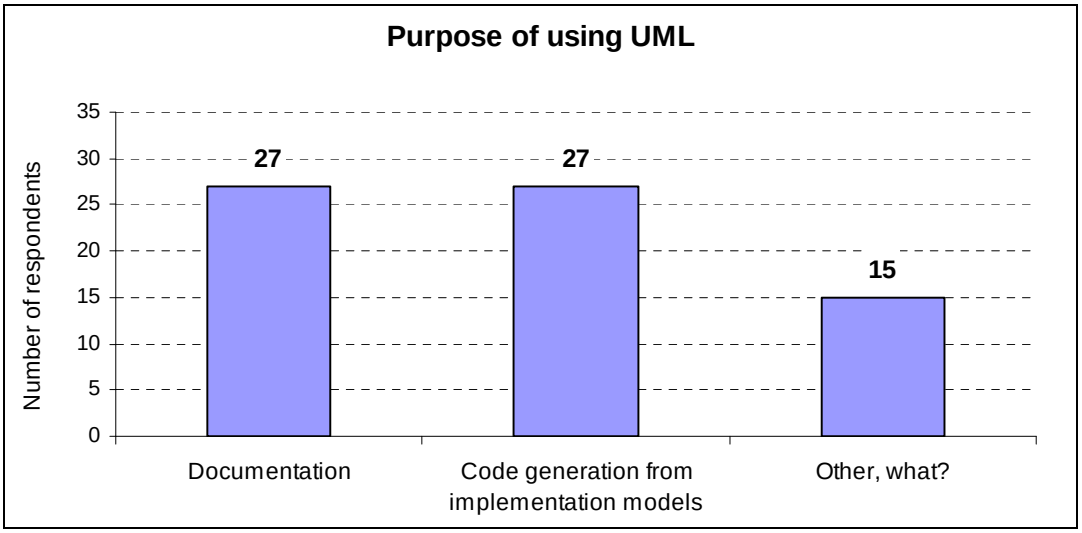

Figure 22. Purpose of using UML.

8.4.4 UML tools

According to the survey, Ratinal Rose is the most highly used UML tool (see Figure 23) and was mentioned by 17 respondents. Second was Microsoft Visio with ten responses and the third most common UML tool was Rhapsody with nine responses. In the other category, several tools were mentioned: MagicDraw, PoseidonUML, MetaEdit+, Papyrus, Omundo / Eclipse, Bouml, IBM Rational Software Modeler, Topcased, Eclipse UML2, Rational Software Architect, Eclipse UML2 Tools, GMF, RSA/RSM, ADONIS, Objecteering, Telelogic Tau, Netbeans, Enterprise Architect, and Sparx's Enterprise Architect.

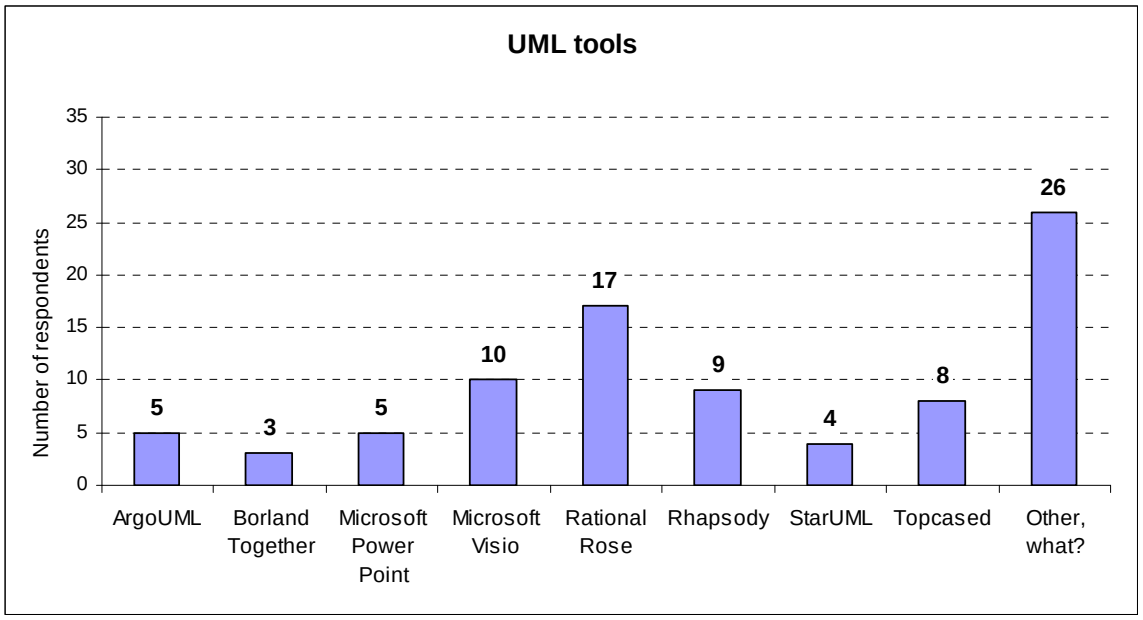

Figure 23. UML tools used in organisations.

8.4.5 Usage of other languages

Respondents were asked to mention what other general purpose language addition than UML they are using. As a result, we received a list of the following tools: BPMN, use case, BPDM, Feature Diagrams, Entity-Relationship Diagrams, XML, FOCUS (a formal specification language for verification purposes) Ordinary programming languages such as Java, C#, C, Ecore based languages, V3Studio, MOF (EMF Ecore) -> DSLs, Aspect J, OCL, XMI, ER, Petri Nets, ADL, and own UML-like languages.

8.4.6 DSM tools

From a given set of answering options, respondents mostly selected the “Other” option and the list of the other tools are as follows: Kermeta, Eclipse and GMF based editors, Visio, GMF, ADONIS, ADOxx, Generic editors, EMF, and OCL.

From a named list of tools, the most commonly used tool were the Microsoft DSL tools with ten responses (see Figure 24). Second were the MetaEdit+ and Generic Eclipse Modelling System with five responses each.

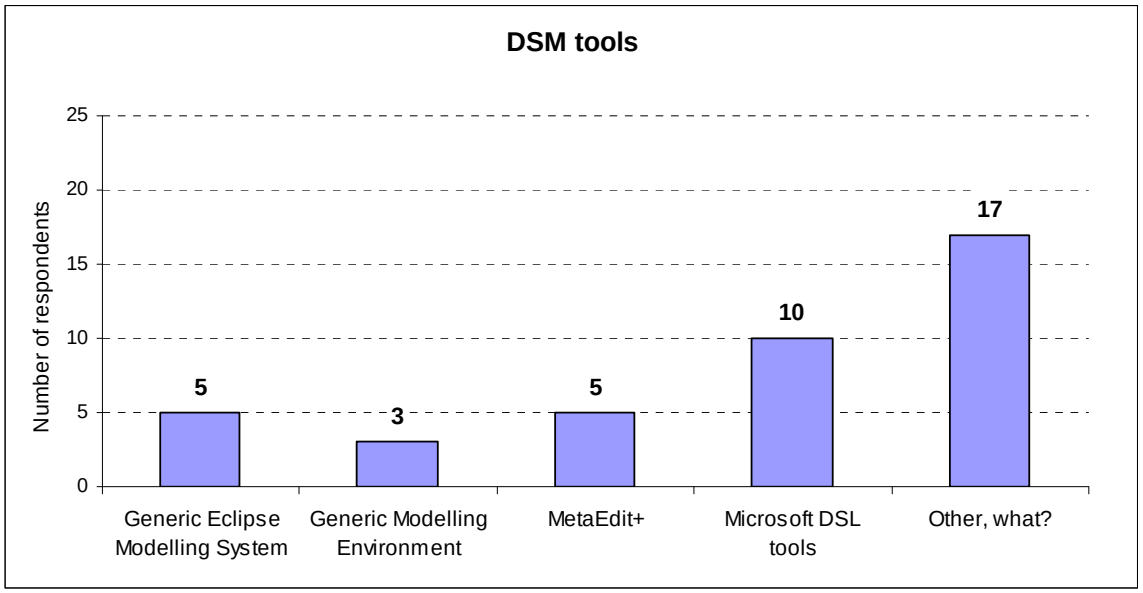

Figure 24. Usage of DSM tools.

9. 2nd ITEA MoSiS survey: MDD usage and challenges in companies

This chapter presents the results of the 2nd survey that was conducted in work package 3 of the ITEA MoSiS project. The survey was conducted to discover the usage of the MDD and related challenges in companies. The survey was carried out during October, 2008 and it was organised by VTT Technical Research Centre of Finland.

9.1 Goal of the survey

The goal of the 2nd survey was to find out:

- How largely the MDD was used in companies?
- How has the MDD technologies affected product development process activities?
- What kind of challenges have companies faced with the MDD technologies?

9.2 Set-up of the survey

The survey was prepared as an electronic questionnaire that was available on the web. The questionnaire contained 14 questions, altogether. The survey was available to respondents during over the period 21st of October – 31st of October 2008. The link to the survey was emailed to MoSiS partners, which were from Spain, Sweden, Norway and Finland. The link was also emailed to country co-ordinators of the MoSiS project, who were asked to distribute the link forward to suitable mailing lists within their country. As with the first survey, the link was also available to people that had been in close contact with VTT; they had participated in seminars at VTT or they were customers of VTT. All of the respondents of the survey were recorded as anonymous.

Questions were grouped as follows:

Group A: The background of the respondent

Group B: Utilisation of the model-driven development (MDD) in the company

Group C: Model-driven development and the product development process activities

Group D: The maturity of modelling technology and related methods

Group E: The benefits with the model-driven development technologies

Group F: Challenges with the model-driven development technologies

Groups from G to R: The importance and solutions of specific model-driven development challenges

Group S: Free comments on the model-driven development

The complete list of questions in each of the groups can be found in Appendix B.

10. Results of the 2nd MoSiS survey

This chapter documents the results from the MoSiS survey. Altogether, the number of respondents totalled 30, in which 16 respondents answered the whole questionnaire and 14 stopped answering during some point of the questionnaire. Therefore, the number of responses varies between the questions.

10.1 Group A: Background of the respondent

Questions in this group collected the basic information of the respondents and their organisations.

10.1.1 The respondents' software development experience

According to the responses, respondents were experienced software developers. 19 respondents out of 27 had more that 10 years experience with software development (see Figure 25). Six of the respondents' experience in software development was somewhere between five to ten years. One respondent had three to five years experience in software development. Only one respondent had less than one year of experience in software development.

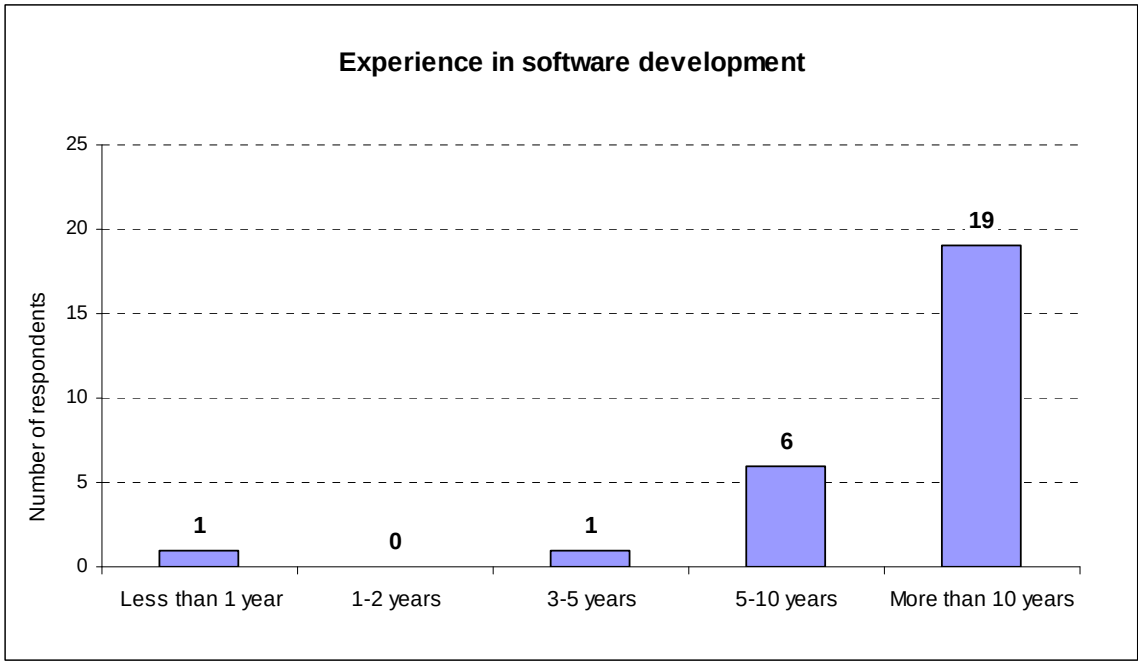

Figure 25. The respondents' experience in software development.

10.1.2 The respondent's role in the organisation

Most of the respondents were either architects/designers or developers/engineers (see Figure 26). Both of these categories had six respondents. The second most common group was chief architects with four respondents. Three researchers and consultants were also among the respondents. The categories of quality assurance/process engineer, program manager, and project manager had two respondents in each category. One requirements manager/requirements engineer, team manager and tester also responded to the questionnaire. In the category of others, a methods developer, VP of engineering, and SW Specialist were mentioned as a role of the respondents'.

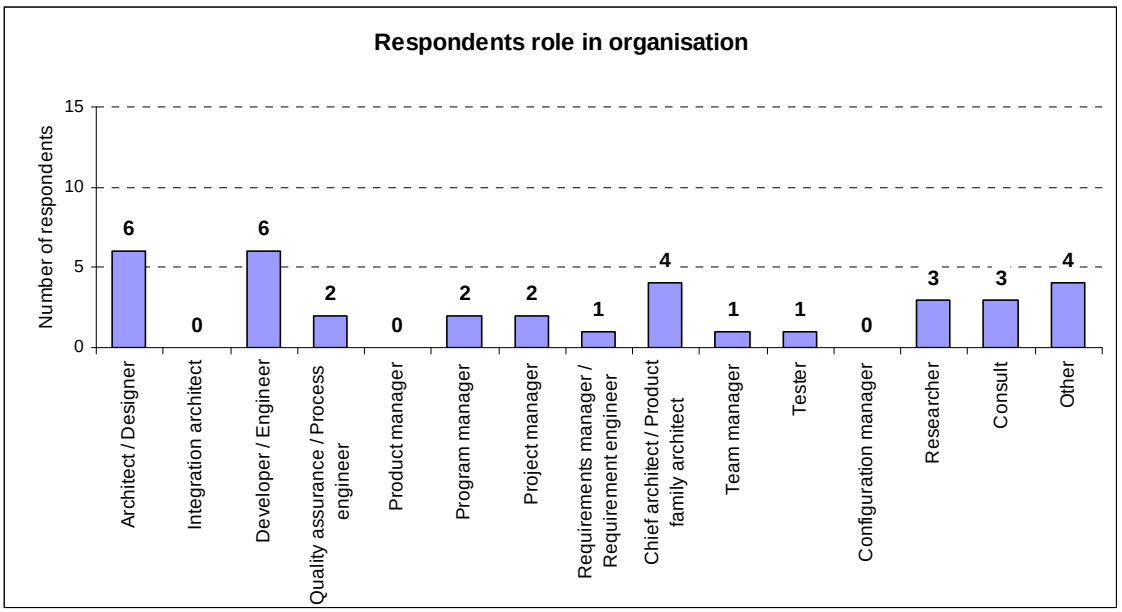

Figure 26. The respondents' role in the organisation.

10.1.3 Domain of respondents' organisation

In this question, some respondents categorised their organisation into several domains. Most of the respondents (14) represented embedded software organisations (see Figure 27). Telecommunication (7), industrial automation (6) and software product (6) domains were also represented well in the survey. There were three respondents in the information systems and research domain categories. Two respondents were from the wireless telecommunication domain. In the others category, avionics, military systems, and aerospace/defence were mentioned with one respondent in each domain.

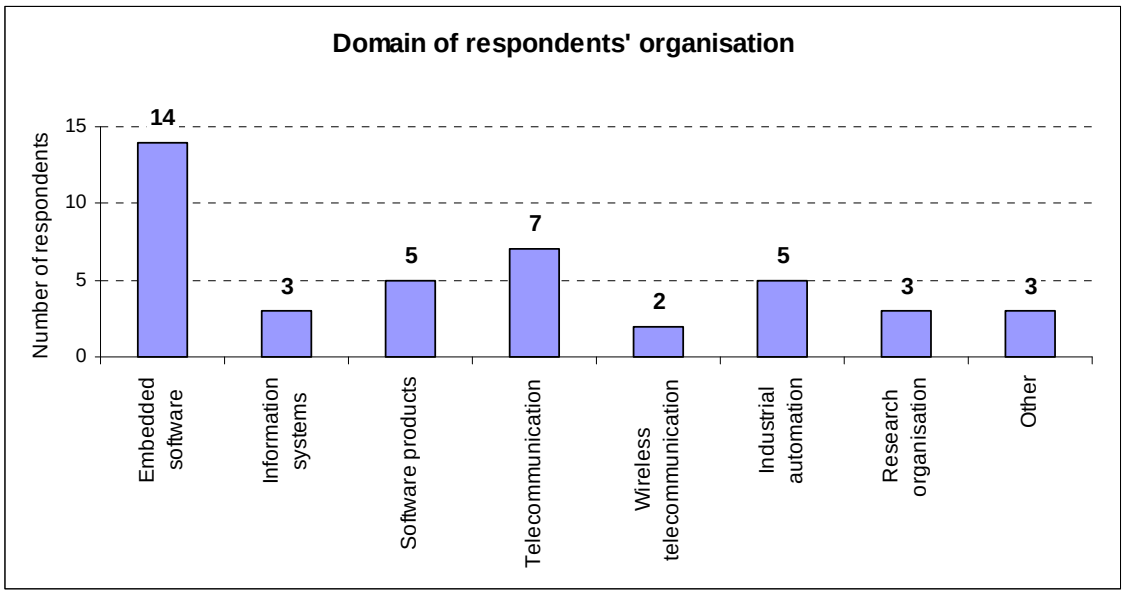

Figure 27. The domain of the respondents' organisation.

10.1.4 The size of the organisation

There were small, middle size and large organisations involved in the survey (see Figure 28). Most respondents (10) came from organisations having 250–999 employees. Nine respondents represented organisations with more that 1000 employees. Two respondents came from organisations having less than 10 employees.

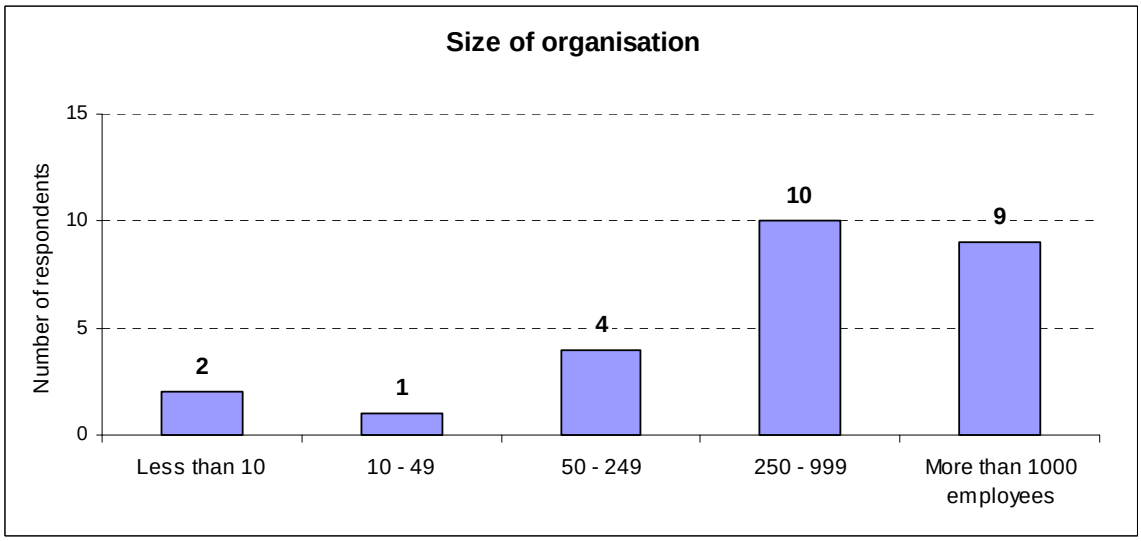

Figure 28. Size of the respondent's organisation.

10.1.5 Products and services

13 respondents mentioned that their company makes HW-SW products (see Figure 29). These products varied all the way from process automation to specific military systems. There were six respondents coming from organisations that make software products related e.g. to security, communication and sw development. The consulting services category was selected by four respondents and they offered services for development, training and mentoring, as well as for process improvement. One respondent in the others category mentioned that they are performing research.

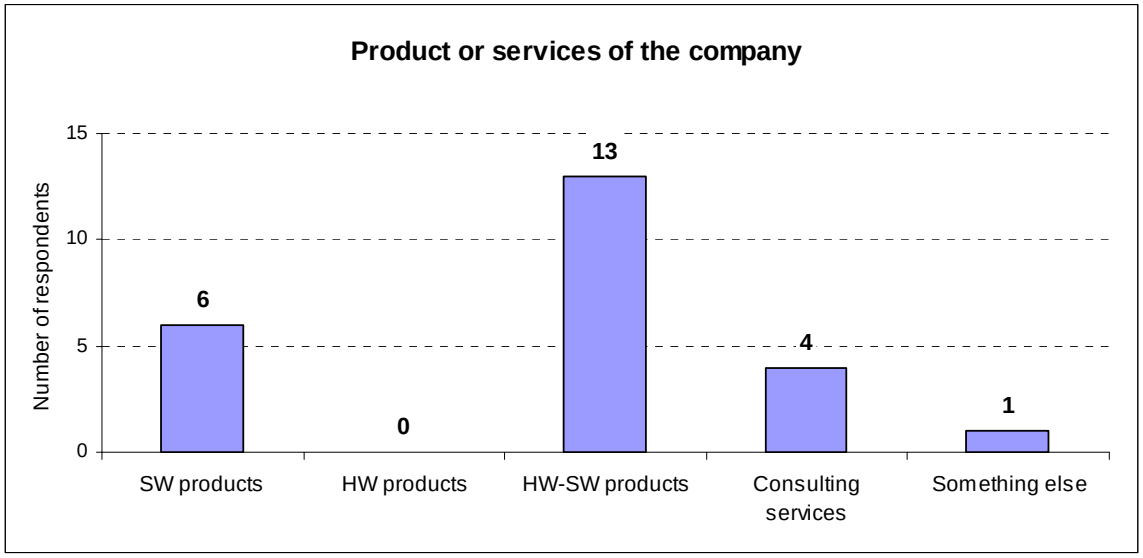

Figure 29. Products or services of the respondent's company.

10.2 Group B: Utilisation of the model-driven development (MDD) in the company

Questions in this group were to study the utilisation of the MDD in the respondents' organisations. Questions collected information on the history related to applying the MDD in the organisations.

10.2.1 Level of MDD process definitions

Respondents were asked to evaluate how exactly the MDD process was defined in their organisation. According to the responses, most of the organisations (11) had partially defined their the MDD processes or guidelines, meaning that some common MDD guidelines or templates were available but they were not integrated with the general

product development process (see Figure 30). Ten respondents answered that there were no process definitions or guidelines to support the MDD activities. Three respondents evaluated that all the phases of the development were supported with guidelines and process definitions were integrated into the general product development process in their organisation.

Answers from five respondents reveal that organisations, having more than 1000 employees, have no process definitions or guidelines to support MDD activities. Three respondents representing organisations with more than 1000 employees responded that they have some common MDD guidelines or templates available, but they are not integrated with the general product development process. There were no respondents from large companies which have full process support.

Three respondents from organisations with 250–999 employees responded that all the development phases are supported with guidelines and process definitions are integrated into the general product development process. According to four respondents from medium size organisations, some common MDD guidelines or templates are available, but they are not integrated with the general product development process. Two respondents shared the view that they have no process definitions or guidelines to support MDD activities.

When it comes to smaller organisations (less than 249 employees), three respondents answered that their organisation has some common MDD guidelines or templates available, but they are not integrated with the general product development process. Two respondents stated that they have no process definitions or guidelines to support MDD activities. According to one respondent, their organisation has guidelines for most phases of development.

One of the respondents did not give information about the size of their organisation. That respondent is calculated in Figure 30, but the information could not be used in analysing the correlation between the size of the organisation and the level of MDD process definitions.

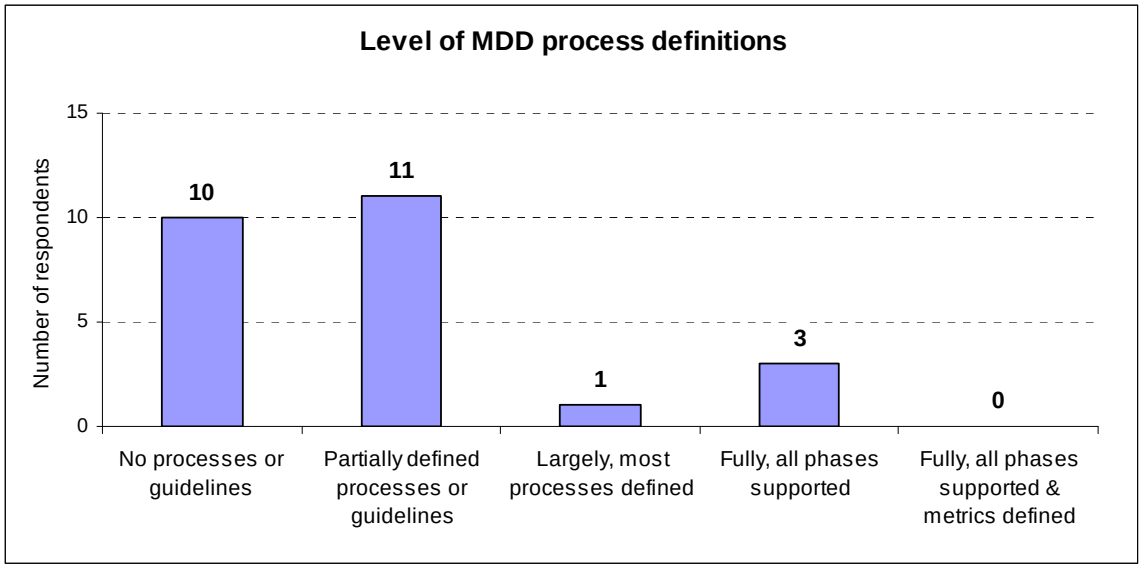

Figure 30. Level of MDD process definitions in the organisation.

10.2.2 Level of applying MDD technologies

Among the respondents, there were no organisations that would apply the MDD in all of their projects, but according to six respondents they apply the MDD in most of their projects. In turn, most of the respondents (8) answered that they already have piloted the MDD in their projects (see Figure 31). Organisations represented by four respondents had plans for applying the MDD soon. There were also three respondents that chose the category, in which organisations were interested in applying the MDD but had not yet planned it.

Two respondents from large companies, with more than 1000 employees, answered that they were interested in the MDD technologies, but had not yet planned to apply them. According to two respondents, they have plans to apply the MDD technologies soon and according to three respondents from large companies, they have already piloted the MDD technologies in their projects. One response claimed to apply the MDD technologies in most of their projects.

One respondent from a medium size organisation with 250–999 employees answered that they are interested in MDD technologies, but they are not yet planned to apply them. According to two respondents, they have plans to apply the MDD technologies soon. Two respondents stated that they already have piloted MDD technologies in their projects. Three respondents are from organisations in which they apply MDD technologies in most of their projects.

Three respondents from smaller organisations, with less than 249 employees, responded that they have already piloted the MDD technologies in their projects and according to two respondents, they apply MDD technologies in most of their projects.

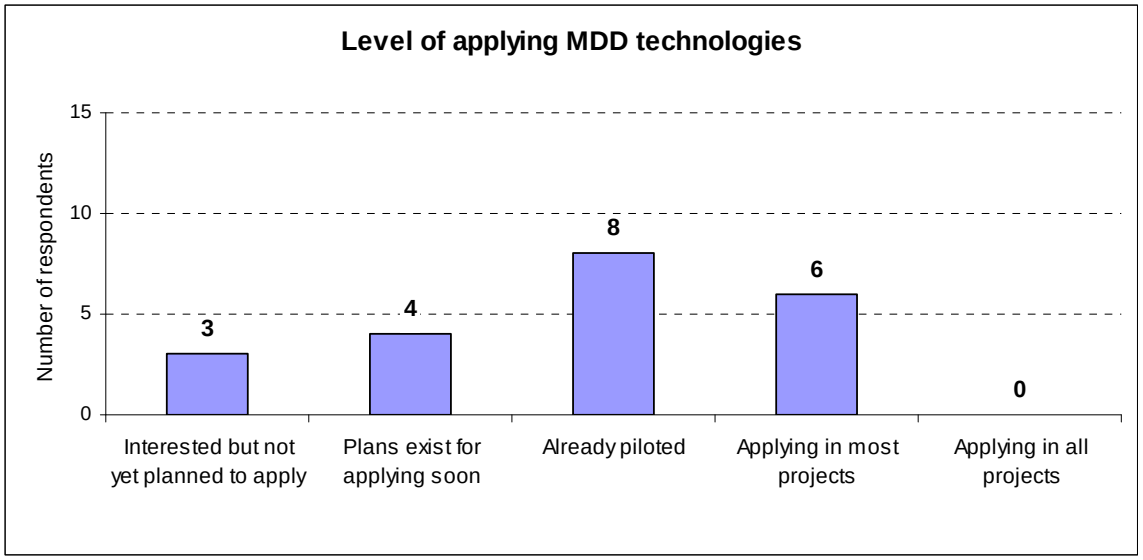

Figure 31. Level of applying MDD technologies.

10.2.3 Time how long MDD has been applied

Most of the respondents answered that the MDD had been applied for more than 2 years in their companies (see Figure 32). According to four respondents, they had not applied it at all and according to three respondents they had applied it for a maximum of 0.5 years. Two respondents answered that their organisations had applied the MDD for one to two years.

According to two respondents from large companies (more than 1000 employees), they have not applied MDD technologies in their companies. Two respondents stated that they had applied it for 0.5 years. One respondent answered having applied the MDD for 1–2 years and according to two respondents, they have applied it for more than 2 years.

Six respondents from medium size companies answered that they have applied the MDD for more than 2 years in their companies. One respondent stated not having applied it at all and one responded that they have applied the MDD for 1–2 years.

Four respondents from small companies have applied the MDD for more than 2 years and one respondent 0.5 years.

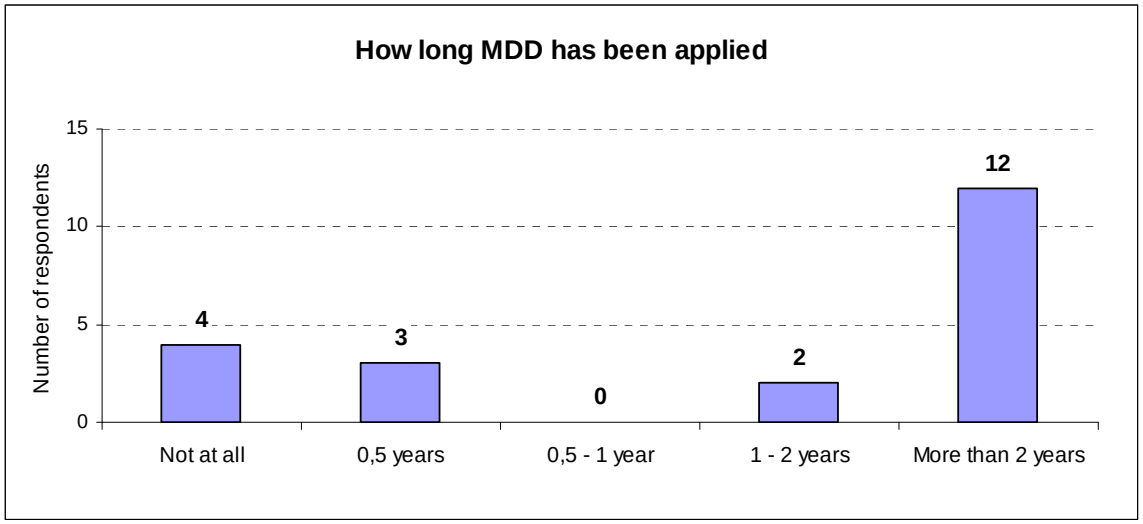

Figure 32. Time how long MDD has been applied in the organisation.

10.2.4 MDD and how systematically it is applied

In most cases (11 responses), the application of MDD related activities depends completely on the interests and experiences of the project's personnel (see 3). There were three responses that reveal that all projects in their organisation apply some, but not necessarily the same, MDD activities. We can see from the answers that there were no organisations that would apply the MDD in all of their projects, and at the same time, would provide metrics to support the evaluation of the performance and predictability of the MDD outcomes. Anyway, according to five respondents, all the projects in their organisations apply the same, well defined and trained MDD activities.

Answers from five respondents reveal that the application of MDD related activities in organisations having more than 1000 employees depends completely on the interests and experiences of the project's personnel. One respondent answered that all their projects apply the same, well defined and trained MDD activities. According to one respondent, all the projects apply some, but not necessarily the same, MDD activities in organisation.

Also five respondents from middle size organisations answered that the application of MDD related activities depends on the interests and experiences of the project's personnel. According to three respondents, al the projects apply the same, well defined and trained MDD activities.

In case of smaller organisations, one respondent said that in their organisation the application of MDD related activities depends completely on the interests and experiences of the project's personnel. Two respondents answered that all the projects

apply some, but not necessarily the same, MDD activities. According to one respondent, all the projects apply the same, well defined and trained MDD activities.

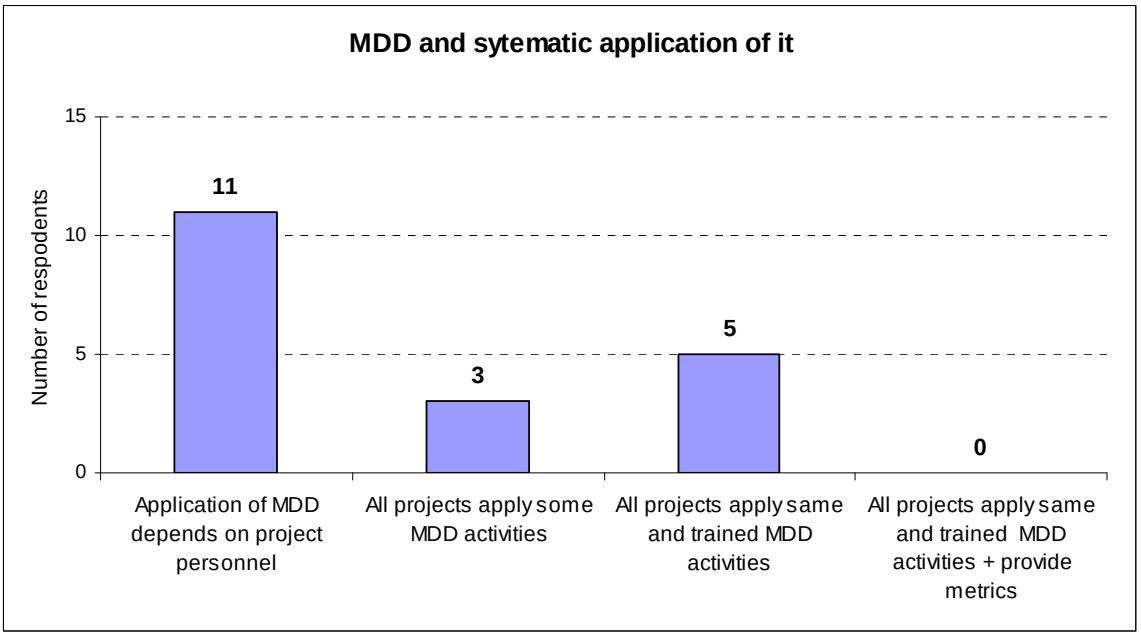

Figure 33. How systematically is the MDD applied in organisation.

10.2.5 The initiation of the MDD in the organisation

The initiation of the MDD in the organisation has typically (9 respondents) began when some individuals started to use the technologies in their work, based on their own initiative (see Figure 34). Organisations, in which three respondents were coming from, had a systematically planned and monitored pilot with several MDD technologies at the same time. According to two respondents, their organisations began using the MDD by systematically piloting certain MDD technology in a project. Two respondents answered that their organisations had precisely defined goals, e.g. to improve the quality of data modelling or improve the productivity of development of a given part of software.

According to five respondents from large organisations, the use of the MDD began because some individuals started to use the technologies in their work, based on their own initiative. In one case, they had a systematically planned and monitored pilot with several MDD technologies at the same time.

Two respondents from medium size organisations answered that some individuals began to use the technologies in their work, based on their own initiative. According to two respondents, they had systematically planned and monitored a pilot of a certain

MDD technology in a project. Two respondents also answered that they had systematically planned and monitored a pilot with several MDD technologies at the same time.

In two cases in smaller organisations, some individuals began to use the technologies in their work, based on their own initiative. According to two respondents, they had precisely defined goals for introducing the MDD.

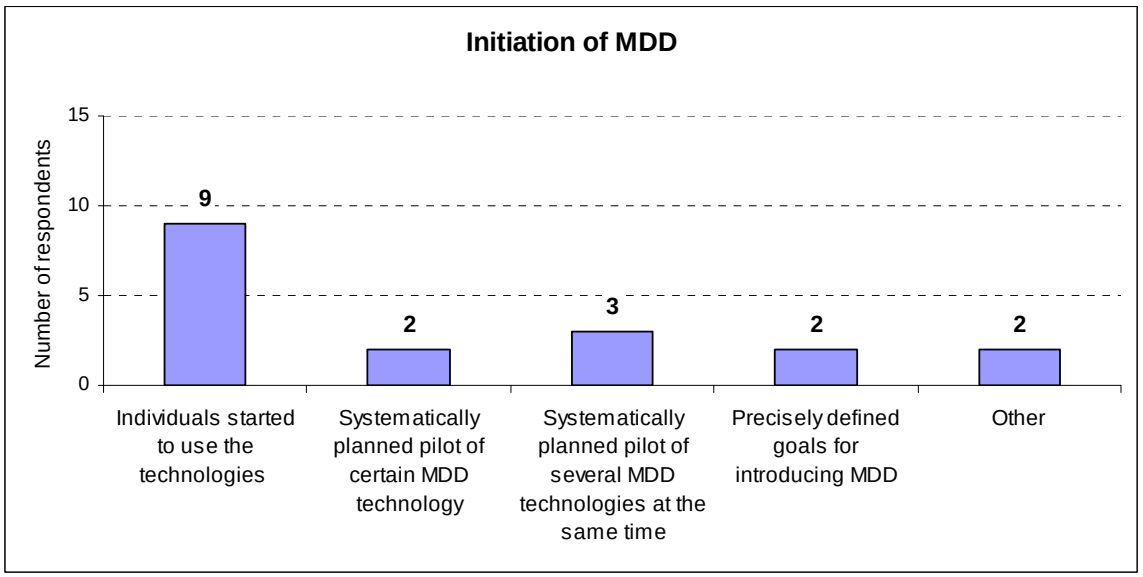

Figure 34. The initiation of the MDD in the organisation.

10.3 Group C: MDD and the product development activities

Questions in this group clarified the influence of MDD to the following product development process activities: context definition, requirements elicitation and analysis, analysis, design, implementation, integration and testing, release. Respondents evaluated each activity and selected the level of influence from a scale ranging from 1 to 5, where 1 had no influence, 3 had moderate influence and 5 had a strong influence. For each activity, the average influence and median of all the answers is reported.

10.3.1 Influence on the context definition

In the questionnaire, the context definition was described as follows: “Context definition – the objective is to define the scope for the software system that will be developed.”

According to the respondents, MDD has generally had a mild or moderated influence on context definition activity (see Figure 35). Responses were varied from no and minor

influence (8 respondents) to high influence (2 respondents), when the average of all the responses was about 2.5 and the median of all the responses was about 2.3 (see Figure 36).

Figure 35. Influence on context definition.

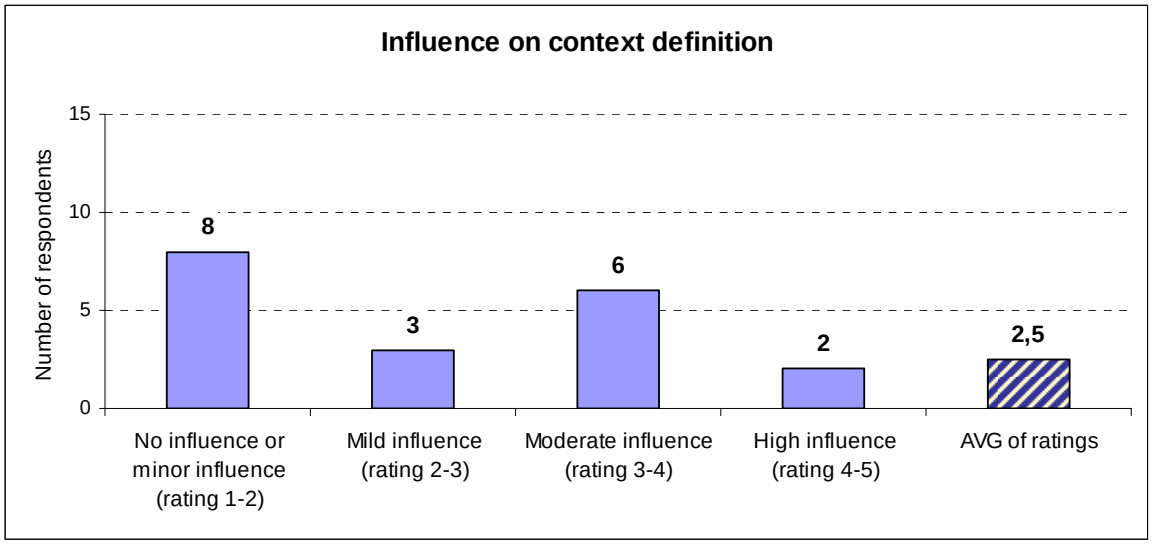

Figure 36. Influence on context definition: the variation of individual answers.

Respondents were asked to more specifically describe the influence of the MDD on the context definition activity. Some of the respondents highlighted that they have used MDD for rapid prototyping and according to their observations, it has made prototyping faster. MDD has affected the timing and resourcing of feature development, but not the features themselves. It has enabled the scope to be set at an earlier stage and also in a better way. According to some responses, the context is defined at the system specification level and the MDD is not used at the system level. It was also noticed that a use case analysis is often used when defining new systems. The MDD has made communication easier, allowed comparison and evaluation, as well as discussions on firm bases.

10.3.2 The influence on requirements elicitation and analysis

In the questionnaire, requirements elicitation and analysis were described as follows: "Requirements elicitation and analysis – the objective is to identify, capture, agree and validate the functional and non-functional requirements related to the system to be developed."

According to the respondents, MDD has generally had a mild or moderated influence on the requirements elicitation and analysis activity (see Figure 37). Responses varied from no and minor influence (5 respondents) to high influence (2 respondents), when the average of all the responses was about 2.7 and the median of all the responses was about 2.8 (see Figure 38).

Figure 37. The influence on requirements elicitation and analysis.

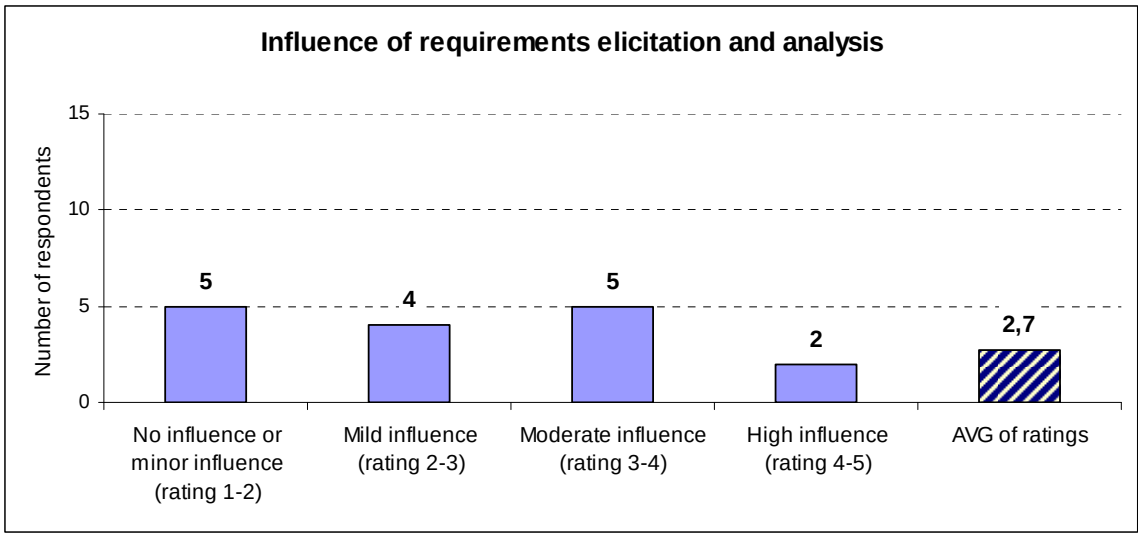

Figure 38. The influence on requirements elicitation and analysis: variation of individual answers.

Respondents were asked to more specifically describe the influence of the MDD on requirements elicitation and analysis activity. According to responses, use case modelling was used for requirements capturing and analysis, modelling of functional requirements, and prototyping for the investigation of qualities, such as performance. MDD was also used for defining and analysing interactions and deriving requirements from the interactions.

10.3.3 The influence on analysis

In the questionnaire, analysis was described as follows: "Analysis – the objective is to model system's internal view in which the technological considerations are excluded and the separation between the functional and non-functional requirements is maintained."

According to the respondents, the MDD has generally had a mild or moderated influence on analysis activity (see Figure 39). Responses scattered from no and minor influence (3 respondents) to high influence (2 respondents), when the average of all the responses was about 2.8 and the median of all the responses was about 2.9 (see Figure 40).

Figure 39. The influence on analysis.

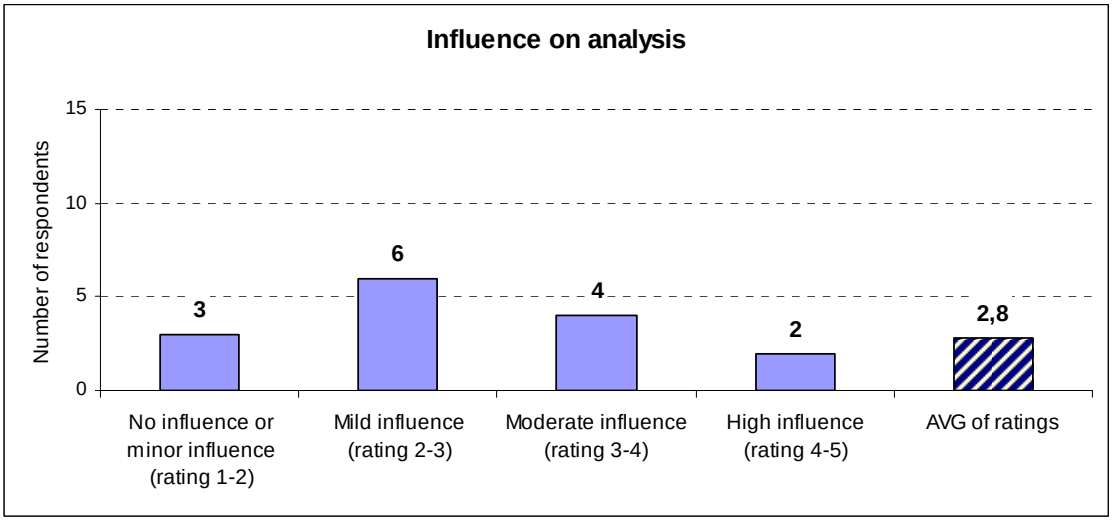

Figure 40. The influence on analysis: variation of individual answers.

Respondents were asked to more specifically describe the influence of the MDD on analysis activity. According to respondents, internal and low-level design is performed on a model level as well as they are using modelling and code generation. Some respondents stated that for new projects, they try to use models as the main tool for analysis. In some cases, platform independent design is fully modelled and also tested and executed to verify correctness.

10.3.4 The influence on design

In the questionnaire, design was described as follows: “Design – the objective is to produce the technical specification of the system, i.e., to model the behaviour and structure of a solution that fulfils both the functional and non-functional requirements.”

According to the respondents, the MDD had a slightly above moderate influence on the design activity (see Figure 41). Responses varied from no and minor influence (3 respondents) to high influence (7 respondents), when the average of all the responses was about 3.2 and the median of all the responses was about 3.6 (see Figure 42).

Figure 41. The influence on design.

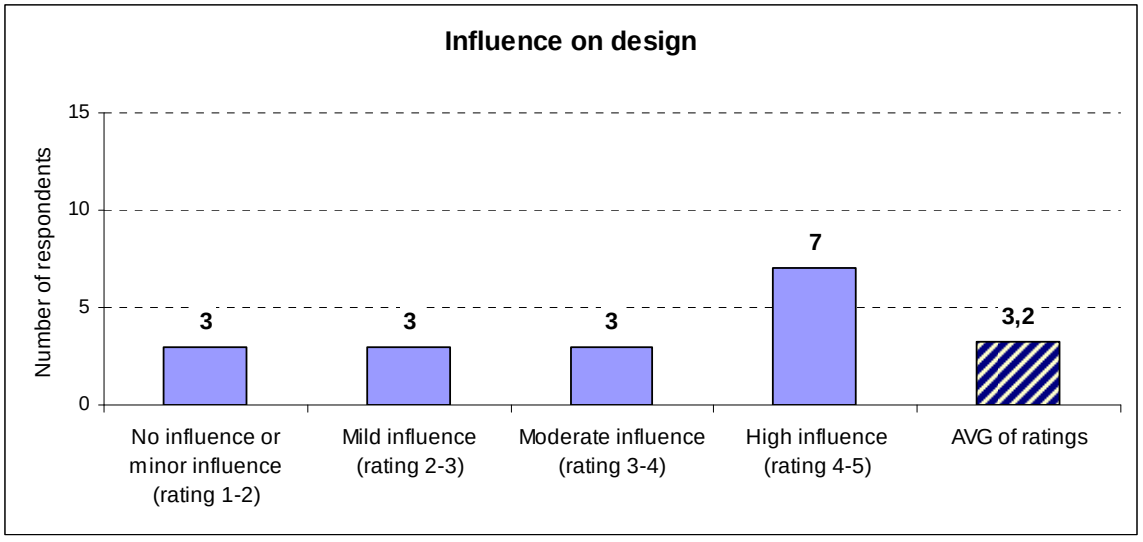

Figure 42. The influence on design: variation of individual answers.

The respondents were asked to more specifically describe the influence of the MDD on design activity. According to the answers, modelling has been deployed in almost all software design. Hard real-time and embedded software has been excluded, but it is used in high availability software. Use of the MDD had also supported communication and understanding and it has allowed other stakeholders e.g. testing to participate in design activity. Some respondents stated that they use the MDD only at a software level and only for some products. On the other hand, in some organisations, all behaviour is

modelled and used in code generation. One respondent stated that the possibility to reuse solutions had increased and with a complete code generation, it was also easier to separate the impact of the functional requirements from the non-functional in the solution.

10.3.5 The influence on implementation

In the questionnaire, implementation was described as follows: “Implementation – the objective is to develop and verify the code that implements the design and fulfils the requirements.”

According to the respondents, the MDD typically had a moderate influence on the implementation activity (see Figure 43). Responses scattered from no and minor influence (5 respondents) to high influence (6 respondents), when the average of all the responses was about 3.1 and the median of all the responses was about 2.9 (see Figure 44).

Figure 43. The influence on implementation.

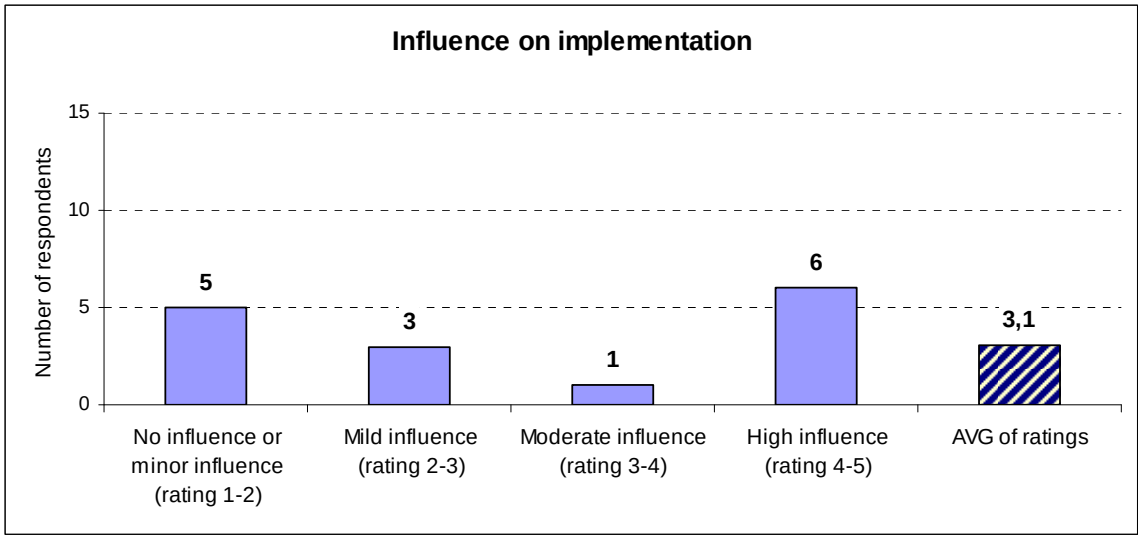

Figure 44. The influence on implementation: variation of individual answers.

Respondents were asked to more specifically describe the influence of the MDD on the implementation activity. The majority of the respondents stated that the MDD influenced the implementation through generation of code from models. According to some responses, they generate all codes from the models. Some respondents highlighted that the code generation from design models provided a fundamental increase in productivity. In the area where respondents use fully automatic code generation, the iterations have increased and the possibility to analyse and modify the solution has increased, resulting in better, more robust design and stable software. The possibility to reuse solutions has also increased, according to a respondent. With a complete code generation, it is also easier to separate the impact of functional requirements from the non-functional in the solution. On the other hand, some scepticism against the code generation was present among respondents.

10.3.6 The influence on integration and testing

In the questionnaire, integration and testing was described as follows: “Integration and testing – the objective is to show that the requirements are satisfied by the developed system.”

According to the respondents, the MDD had less than a moderate influence on integration and testing activity (see Figure 45). Responses varied from no and minor influence (5 respondents) to high influence (6 respondents), when the average of all the responses was about 2.7 and the median of all the responses was about 2.5 (see Figure 46).

Figure 45. The influence on integration and testing.

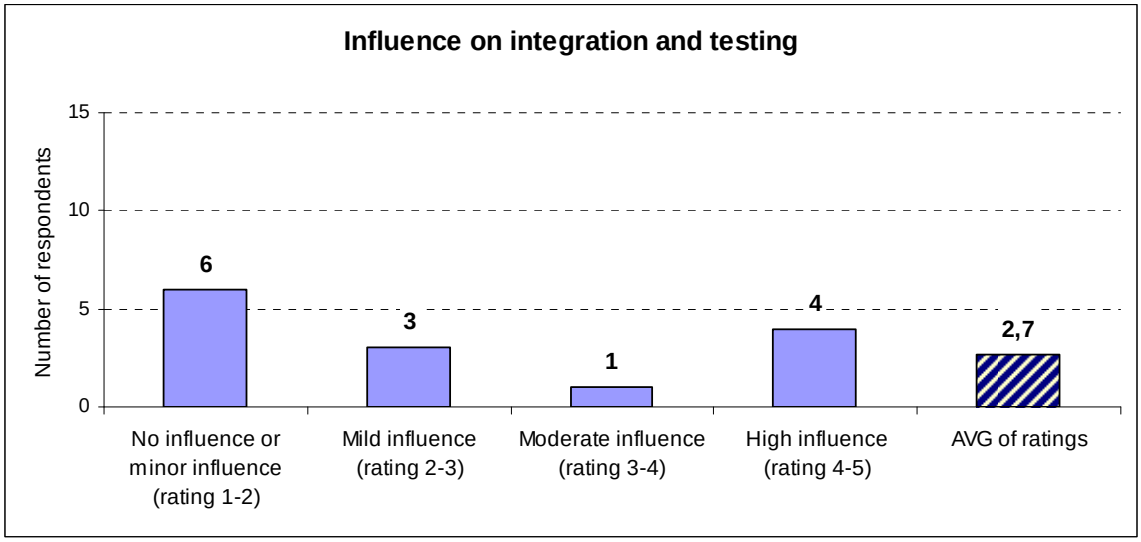

Figure 46. The influence on integration and testing: variation of individual answers.

Respondents were asked to more specifically describe the influence of the MDD on the integration and testing activity. Some respondents stated that integration is done on a model level and also tested at that level. According to the respondents, e.g. early phase (close to the design) testing uses more modelling than system integration testing and system level sequence diagrams are used in the system integration testing. Among the answers, there was also a notice that the influence depends if the languages were already applied, so that they are supporting the integration or are the models separate, e.g. code files, so that there is an extra effort in the integration. In some cases, generators have been applied to support the integration, such as build script, test case generation. In the projects, where the respondents applied the fully translated approach, the integration was much easier, since all the developers were able to generate a complete system very early. The integration activities were hence solved during the design time in most cases. Testing was still not that affected, but the error frequency tending to go down, since all of the developers can run its module in its environment very early and do not need to wait for the integration activities.

10.3.7 The influence on the release

In the questionnaire, the release was described as follows: “Release – the objective is to ensure that the developed system is successfully taken into use among the final users.”

Among all the activities, according to the respondents, the MDD had the lowest influence on the release activity (see Figure 47). Responses varied from no and minor influence (8 respondents) to high influence (1 respondent), when the average of all the responses was about 2.0 and the median of all the responses was about 1.7 (see Figure 48).

Figure 47. The influence on the release.

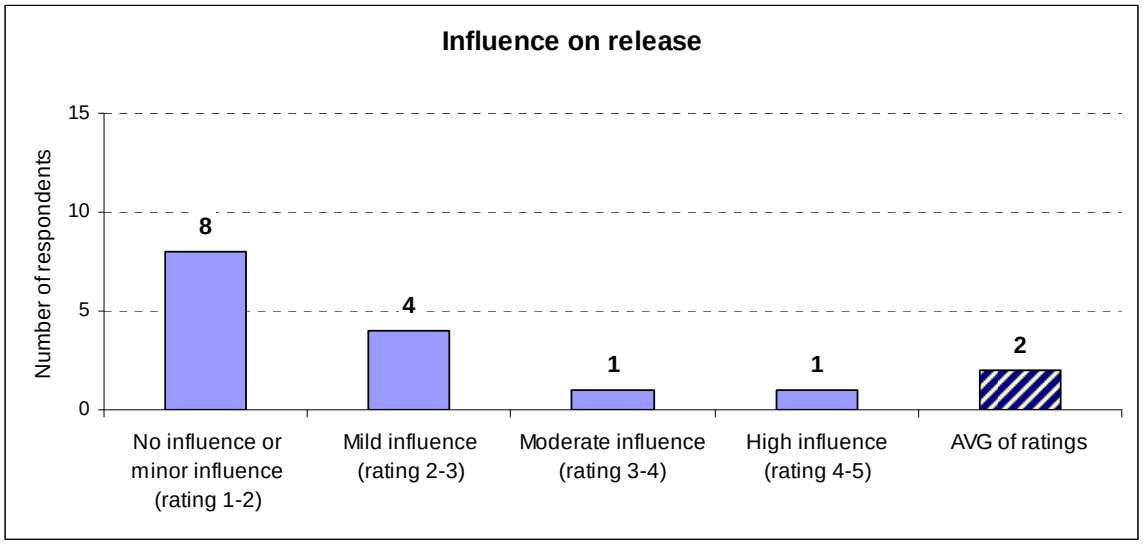

Figure 48. The influence on the release: variation of individual answers.

Respondents were asked to more specifically describe the influence of the MDD on the release activity. Some of these answers stated that the MDD did not have any influence at all or did not have much influence on the release activity as they are using the same release process as before the MDD. It was also mentioned that the influence depends on the project and at best, they can generate the installer/configurator at the same time when they are generating the code. The use of one source makes sure of the consistency between the functionality of the product and its installation procedure.

10.3.8 Summary of the Influences

The following Figure 49 shows the individual respondents answers (each line is a respondent).

As the figure shows, there is quite some variation in the evaluated influence of the MDD practices in phases between the respondents. That causes the medians, for most of the phases, to be between 2 and 3, even though there are many respondents that have evaluated e.g., the influence on design and implementation is very high. The results also show that if a respondent has evaluated the impact as high in some phase, (s)he has

often evaluated the impact as high in other phases as well (see e.g., evaluator 10, 11, 17). Similarly, many of the respondents that have evaluated the impact as low at some phase, have also evaluated the impact as low in many other phases (see e.g., evaluator 12, 15, 16).

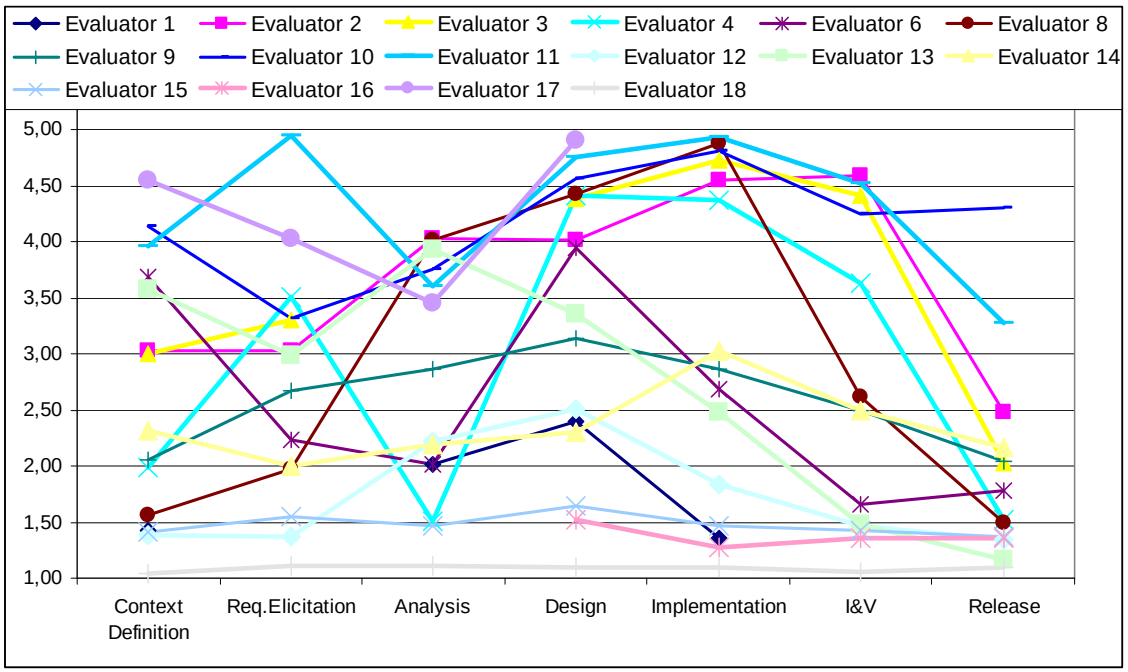

Figure 49. Individual respondent's answers to the influence of the MDD to process phases.

10.4 Group D: The maturity of modelling technology and related methods

The questions in this group were to study the maturity of the modelling technology and related methods.

10.4.1 Modelling technology and functionality

According to the respondents (see Figure 50), most of them value the modelling technology that provides mechanisms to develop and execute model analysis methods as well as mechanisms to develop and introduce domain specific modelling. Both groups of mechanisms were marked as important by nine respondents. Seven respondents valued the advanced features for developing and executing model transformations. In the something else category, the following functionalities was mentioned as important modelling technology features:

- Support for planning projects integration steps and change management activities, with analysis of the change impact in the solution as well as the time schedule.
- Fully allow design and debugging at the UML model level and allow specification of the modelling architectural design rules on the meta-model level in the UML.
- Allow design and debugging at the model level, there should not be any need to view the source code level.

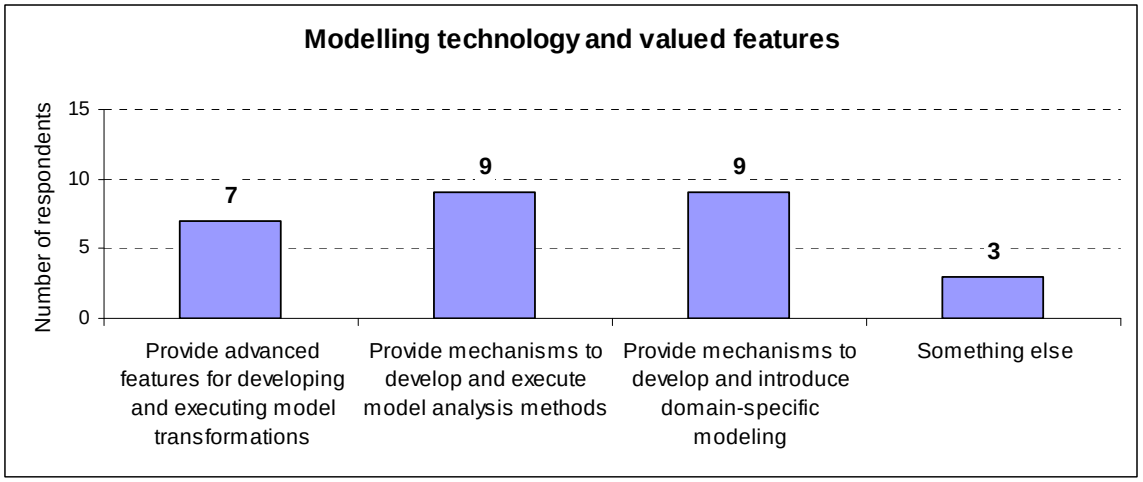

Figure 50. The modelling technology and valued features.

10.4.2 Modelling methods and valued activities

Most of the respondents (12) valued a modelling method that includes early model-based verification and validation (see Figure 51). The second most valued activity (9 respondents) was model-based project planning and management. The model-based quality assurance was selected by seven respondents. The modelling of architectural design rules at the meta-model level, configuration aid, metrics, documentation and autobuild were mentioned in the something else category as important activities in the modelling method.

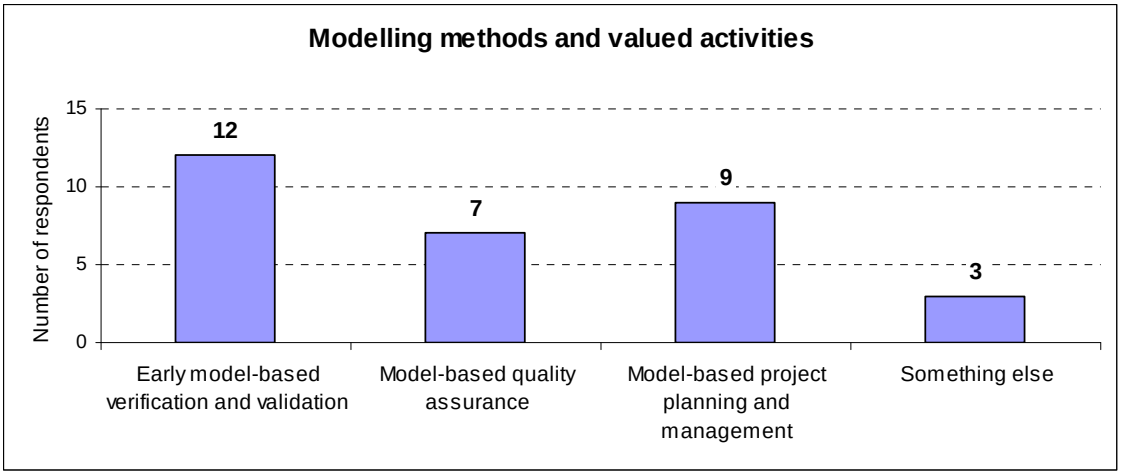

Figure 51. The modelling methods and valued activities.

10.5 Group E: The benefits of the MDD technologies

The questions in this group were to clarify the benefits that the respondents had faced with the MDD technologies.

10.5.1 The MDD technologies and their benefits

Most of the respondents (10) agreed that the MDD technologies reduced the time that was required to study a new system, because the design was more easily communicated to people (see Figure 52). Eight respondents stated that the MDD technologies improved human communication and lowered language barriers between the different parties during the project. Improvements in quality, e.g. a smaller amount of bugs in the generated code than manually implemented code, were seen by seven respondents and improvements in development speed by six respondents. Respondents stated in the something else category that the MDD technologies improved satisfaction among developers and made it easier to introduce new developers.

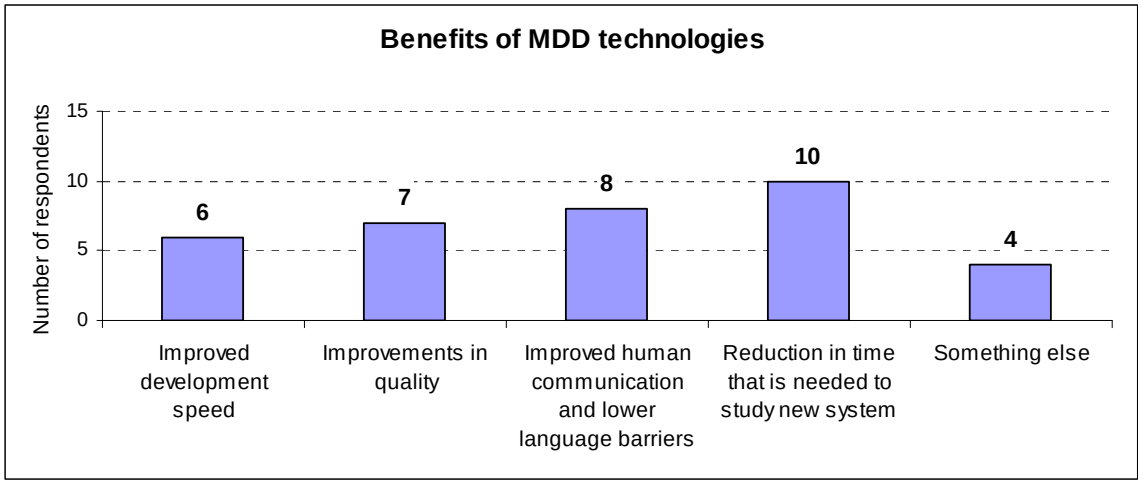

Figure 52. Benefits of MDD technologies.

10.5.2 Comments on the MDD technologies and their benefits

In this question, respondents were asked to give their free comments on the benefits of the MDD technologies. According to these comments, MDD technologies decreased required effort by about 50 to 70 percent and made it possible to implement more features with fewer resources. It was stated that the MDD technologies made it easier to understand and review each other's designs. For instance, model based interfaces were easier to communicate and understand and one common picture of the system was provided. Other remarks highlighted that piloting was faster because less documentation was required, the number of errors were decreased, code was more efficient and its quality was better and documentation was always up-to-date. Due to the tool support, code generation and build configuration generation resources were able to focus more on the actual design.

10.6 Group F: The Challenges of MDD technologies

The questions in this group were to clarify the challenges that respondents faced with the MDD technologies.

10.6.1 Challenges of the MDD technologies

Most of the respondents (10) recognised the lack of modelling experts in organisations as a challenge (see Figure 53). Another challenge that was recognised by several respondents (6) was about comparing and merging different versions of models, for

example, visualising the differences in a usable way. According to five respondents, model-level debugging was not sufficiently supported. The rest of the options got responses from two to four respondents.

In the other challenges category, respondents mentioned that architectural design rules were enforced with manual reviews, so the architects become bottlenecks in the process. The high learning curve in modelling was also mentioned to be challenging. It was also noticed that some of the problems were caused by the use of immature tools causing, for example, the explosion of models if the metamodel is changed. One respondent stated that software designers were more experienced with the MDE tools via the UML, but a larger challenge was to spread methods into system and electronics engineering. The inheritance between the UML and SysML was seen as problematic, since the UML and hence the SysML consist of so many stereotypes and notations that it is a large threshold to overcome, before it can be used and that every one can interpret the model in the same way. The importance of limiting the modelling freedom to get a common model interpretation was also mentioned.

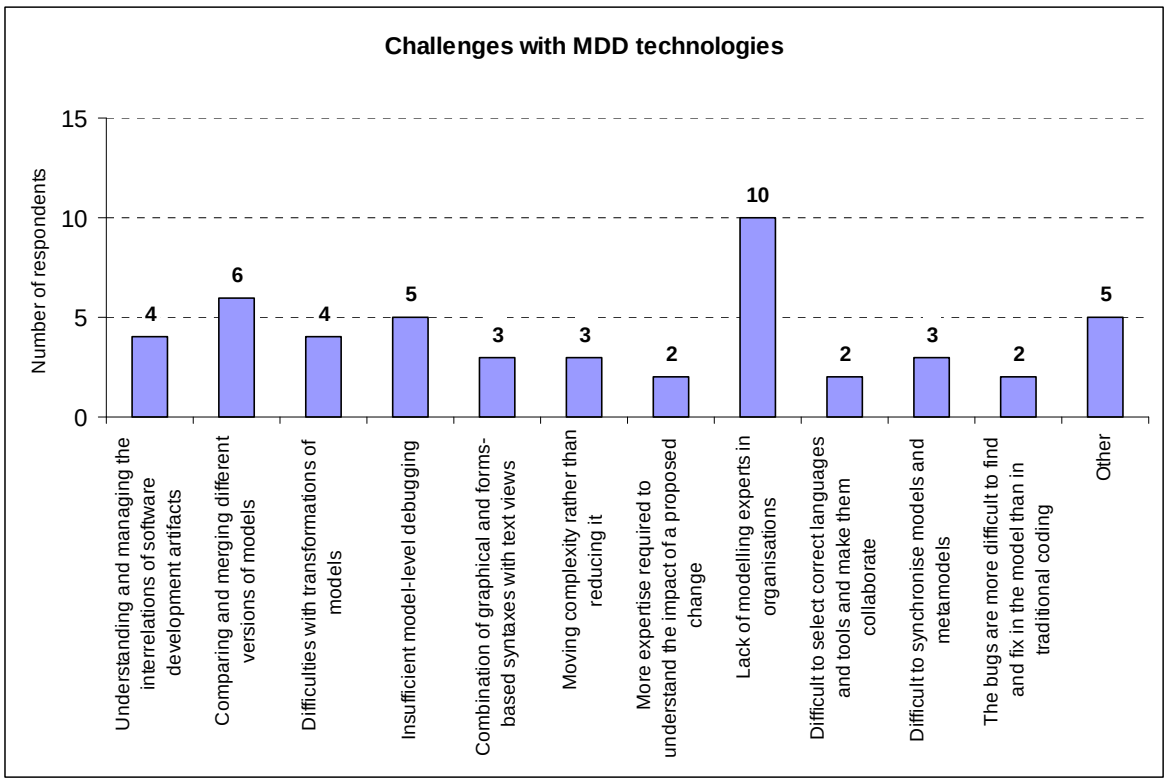

Figure 53. Challenges with MDD technologies.

10.6.2 The rating of challenges related to the MDD technologies

In this question, respondents were asked to choose the three most important challenges from the list. According to the responses (9 respondents), the lack of modelling experts in organisations was recognised as the most important one (see Figure 54). The second most important challenge with four respondents was the understanding and managing of interrelations among the multiple representations of software development artefacts representing different views or levels of abstraction of the same concepts. The remainder of the challenges were seen as almost equally important with respondents varying from one to three respondents.

Respondents were allowed to list other challenges in this question. As a challenge, it was mentioned that architectural rules are manually enforced. Getting the organisation to understand that the model is the specification and high learning curve in modelling were also seen as challenges.

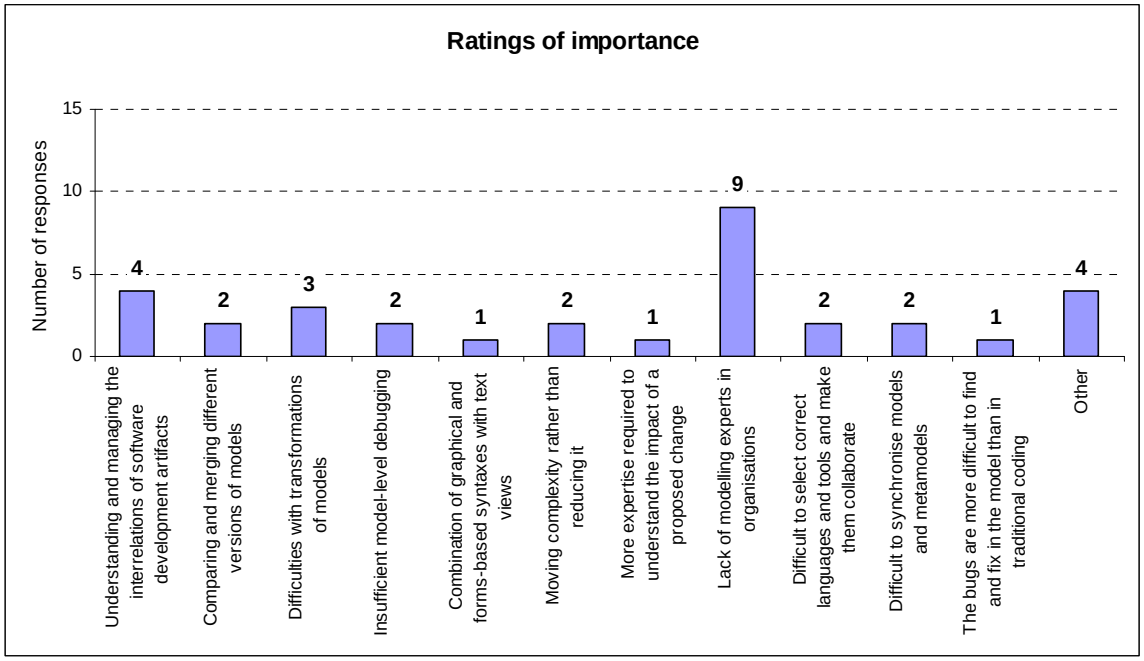

Figure 54. Ratings of the MDD technology challenges.

10.7 Groups G–R: The importance of the MDD technology challenges and solutions for these challenges

The questions in these categories clarified the opinions of respondents related to the importance of each MDD technology challenge. Respondents were also asked to describe the solutions that they might have for each challenge.

10.7.1 The challenges and solutions of the MDD technologies

We gained the following answers when respondents were asked how important a particular challenge was for them, and if they had a solution or some support for handling the challenge:

Challenge 1: "Understanding and managing the interrelations among the multiple representations of software development artefacts representing different views or levels of abstraction of the same concepts":

- This is not only an SW problem. With the SysML, the problem is also an SE problem and is the main threshold to overcome before the MDD can be applied and accepted fully.

Challenge 2: "Comparing and merging different versions of models, e.g., visualising the differences in a usable way":

- There is no tool support, extra manual work is required.

Challenge 3: "Difficulties with the transformations of models (to code or other models): whenever an interrelated artefact changes in ways that affect some or all of its related artefacts, as the mutual consistency cannot always be automatically assured":

- No solutions or support mentioned by the respondents for this challenge.

Challenge 4: "Model-level debugging is not sufficiently supported":

- In the Rhapsody modelling tool, you can execute and debug at the model level but only in a limited way, you are still heavily dependent on co-debugging in the target language, which in turn means that you need to understand how the tool generates code. It would be very good if this level could be abstracted out.

Challenge 5: "A combination of graphical and forms-based syntaxes with text views":

- No solutions or support mentioned by the respondents for this challenge.

Challenge 6: “Moving complexity rather than reducing it (the MDD approach does not reduce the complexity visible to the developer but simply moves the complexity elsewhere in the development process)”:

- Divide the work between the language, generator, additional domain code and existing libraries.

Challenge 7: “More expertise required to understand the impact of a proposed change in all of the related artefacts that could be described in different notations”:

- No solutions or support mentioned by the respondents for this challenge.

Challenge 8: “A lack of modelling experts in organisations”:

- It is of course very important to have good “modellers”. Our solution to this is internal courses and mentoring.
- We have very few “experts” and they are often not just experts in this particular area, so they are often very hard to reach.

Challenge 9: “It is difficult to select correct languages and tools and make them collaborate”:

- No solutions or support mentioned by the respondents for this challenge.

Challenge 10: “It id difficult to synchronise models and metamodels”:

- No tool support.

Challenge 11: “The bugs are more difficult to find and fix in the model than in the traditional coding”:

- No solutions or support mentioned by the respondents for this challenge.

Challenge 12: “Other challenges”:

- Currently, there is no support for the modelling and automation of the enforcement of architectural design rules, this is currently a quite heavy burden for the architects, especially in large distributed projects.
- Training and mentors exist.
- Without tool support, it is almost impossible to deploy the MDE fully in our organisation.

10.8 Groups S: Other comments on the MDD

10.8.1 Free comments on the MDD

In this question, respondents were asked to give their free comments on the MDD. The following answers were submitted:

- Helps in development, but there is still room for improvement e.g. in tools. Tools from different vendors are not compatible.
- It is the future of Systems and Software Engineering, but it is a great challenge to apply and adapt it in large organisations with a tradition of “old school” methodology.
- The MDD, with full code generation, is way more efficient than the traditional development. There is however still a lot to improve in both methods and tools.
- Would be nice to know more about the MDD.
- Not usable in many areas. Does not improve the SW development at all.

11. Summary

The literature study shows that there are many MDD related process models. Some of these models are about how to implement the MDD in an organisation, whereas some models describe the actual MDD process that guides the model based development activities in the organisation. The process models use various names for the activities, but commonly at a high level, the same activities are performed as in non-model-driven development.

Based on the MDD survey, it may be concluded that there are many challenges in the area of model-driven development processes and practices. It was mentioned that in most cases, the use of the MDD in organisations depends only on the interest of people to use it. It was also mentioned that it is difficult to select suitable tools and techniques and adapt them in different environments.

Moreover, it is challenging to get the tools and transformation languages to collaborate. It was also mentioned that there is a lack of MDD experts in organisations and more training would be required. Many of the respondents still evaluate the approach and there is still much confusion about the suitability of the MDD tools and techniques against their needs for product development.

However, despite the challenges of the MDD that the survey pointed out, most of respondents feel that in the next five years the product development processes, methods and tools will be developed towards model-driven development in their organisations, and they also found the approach useful.

Acknowledgements

The authors of the publication acknowledge every organisation and respondent that gave their valuable time and contribution to answering the survey. In addition, the authors would like to thank the support from ITEA and Tekes – the Finnish Funding Agency for Technology and Innovation.

References

[Aagedal & Solheim, 2004] Aagedal, J. Ø., Solheim, I., New Roles in Model-Driven Development, Second European Workshop on Model Driven Architecture (MDA) with an emphasis on Methodologies and Transformations (EWMDA-2), Canterbury, England, 2004

[Afonso et al., 2006] Afonso, M., Vogel, R., Teixeira, J., From Code Centric to Model Software Engineering: Practical case study of MDD infusion in a Systems Integration Company, in Proceedings of the Fourth Workshop on Model-Based Development of Computer-Based Systems and Third International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MBD/MOMPES'06), Potsdam, Germany, 2006

[Ambler, 2007] Ambler, S. W., Examining the Model Driven Architecture (MDA), http://www.agilemodeling.com/essays/mda.htm (referenced 30.1.2009)

[Cáceres et al., 2003] Cáceres, P., Marcos, E., Vela, B., A MDA-Based Approach for Web Information System Development, Workshop in Software Model Engineering (WISME), San Francisco. USA, 2003

[Chandrashekar et al., 2006] Chandrashekar, M. S., Dong, S., Alwandi, N., Model Based Software Development Process For Production Applications, in Proceedings of the 2006 International Conference on Embedded Systems & Applications, Arabnia, H. (ed.), Las Vegas, Nevada, June 26–29, 2006, pp. 38–45

[Chitforoush et al., 2007] Chitforoush, F., Yazdandoods, M., Ramsin, R., Methodology Support for the Model Driven Architecture, 14th Asia-Pasicific Software Engineering Conference (APSEC 2007), Nagoya, Japan, 2007

[CMMI Web page, 2009] Http://www.sei.cmu.edu/cmmi/ (referenced 30.1.2009)

[CRaG Systems, 2008] The Web pages of Model-Driven Development Process http://www.cragsystems.co.uk/development_process/ (referenced 30.1.2009)

[Duby, 2003] Duby, C., Accelerating Embedded Software Development with a Model Driven Architecture, Pahtfinder Solutions, September, 2003, http://www.omg.org/mda/mda_files/MDA_overview.pdf (referenced 30.1.2009)

References

[EFQM Web page, 2009] Http://www.efqm.org (referenced 30.1.2009)

[Estefan, 2007] Estefan, J. A., Survey of Model-Based Systems Engineering (MBSE) Methodologies, Rev A, Incose MBSE Focus Group, 2007, http://syseng.omg.org/MBSE_Methodology_Survey_RevA.pdf (referenced 30.1.2009)

[Foustok, 2007] Foustok, M., Experiences in large-scale, component based, model-driven software development, 1st Annual IEEE Systems Conference, Hawaii, USA, 2007

[France & Rumpe, 2007] France, R., Rumpe, B., Model-driven Development of Complex Software: A research Roadmap, Future of Software Engineering (FOSE '07), IEEE Computer Society, 2007

[Gavras et al., 2003] Gavras, A., Belaunde, M., Steinhau, R., Almeida, J. P. (eds.), MODA-TEL Model-Driven Methodology, MODA-TEL IST-2001-37785, http://www.modatel.org/public/deliverables/D3.add1.htm (referenced 30.1.2009)

[Hailpern & Tarr, 2006] Hailpern, B., Tarr, P., Model-driven development: the good, the bad and the ugly, IBM Systems Journal, Vol. 45, No. 3, pp. 451–461

[Hildenbrand & Korthaus, 2004] Hildenbrand, T., Korthaus, A., A Model-Driven Approach to Business Software Engineering, Proceedings of the 8th World Multi-Conference on Systemic, Cybernetics and Informatics (SCI), Volume IV Information Systems, Technologies and Applications: I, IIIS, Orlando, Florida, USA, 2004

[ISO9001:2000 Web page, 2009] http://www.iso.org/iso/catalogue_detail?csnumber=21823 (referenced 30.1.2009)

[Kleppe et al., 2003] Kleppe, A., Warmer, J., Bast, W., MDA Explained The Model Driven Architecture: Practice and Promise, Pearson Education, 2003

[Kuhn et al., 2006] Kuhn, T., Gotzhein, R., Webel, C., Model-Driven Development with SDL – Process, Tools, and Experiences, in Model Driven Engineering Languages and Systems, in Proceedings of 9th International Conference, MoDELS 2006, Genova, Italy, October 1–6, 2006, Springer LNCS, 2006

[Lahman, 2006] Lahman, H. S., 2006, Model-Based Translation: The Next Step in Agile Development, Pathfinder solutions (white paper), http://www.PathfinderMDA.com (referenced 30.1.2009)

[Larrucea et al., 2004] Larrucea, X., Díez, A. B. G., Mansell, J. X., Practical Model Driven Development Process, European Software Institute, February 2004, http://www.cs.kent.ac.uk/projects/kmf/mdaworkshop/submissions/Larrucea.pdf (referenced 30.1.2009)

[Link et al., 2008] Link, S., Schuster, T., Hoyer, P., Abeck, S., Focusing Graphical User Interfaces in Model-Driven Software Development, in Proceeding of the First International Conference on Advances in Computer-Human Interaction (ACHI 2008), 10–15 February 2008, Saint Luce, Martinoque, IEEE Computer Society Press

[López-Sanz et al., 2008] López-Sanz, M., Acuña, C. J., Cuesta, C. E., Marcos, E., Defining Service-Oriented Software Architecture Models for a MDA-based Development Process at the PIM level, Seventh Working IEEE/IFIP Conference on Software Architecture, 2008, pp. 309–312

[MacDonald et al., 2005] MacDonald, A., Russel, D., Atchison, B., Model-driven Development within a Legacy System: An industry experience report, in Proceedings of the Australian Software Engineering Conference (ASWEC'05), 2005

[Mansell et al., 2006] Mansell, J., Bediaga, A., Vogel, R., Mantell, K., A Process Framework for the Successful Adoption of Model Driven Development, in Proceedings of the Model Driven Architecture – Foundations and Applications, Second European Conference, ECMDA-FA 2006, Bilbao, Spain, July 10–13, 2006

[Mattson et al., 2007] Mattson, A., Lundell, B., Lings, B., Fitzgerald, B., Experiences from representing software architecture in a large industrial project using model driven development, in Proceedings of the 29th international Conference on Software Engineering Workshops (ICSEW), IEEE Computer Society, Washington, DC, USA, 2007

[Mellor et al., 2003] Mellor, S. J., Clark, A. N., Futagami, T., Model-Driven Development, IEEE Software, IEEE Computer Society, 2003

[Middleware, 2003] The Middleware Company, Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach, Productivity Analysis, June 2003, http://www.omg.org/mda/mda_files/MDA_Comparison-TMC_final.pdf (referenced 30.1.2009)

[MODA-TEL Web page, 2009] Http://www.modatel.org/ (referenced 30.1.2009)

[Modelware, 2006] ModelWare FP6-IP 511731, D2.3. MDD Maturity Levels Definition, revision 2.1.,10/08/06, http://www.modelware-ist.org/index.php?option=com_remository&Itemid=79&func=fileinfo&id=108 (referenced 30.1.2009)

[Nikulsins & Nikiforova, 2008] Nikulsins, V., Nikiforova, O., Adapting Software Development Process towards the Model Driven, in Proceedings of the Third International Conference on Software Engineering Advances (ICSEA '08), 26–31 Oct. 2008

[Rios et al., 2006] Rios, E., Bozheva, T., Bediaga, A., Guilloreau, N., MDD Maturity Model: A Roadmap for Introducing Model-Driven Development, in Proceedings of the Model Driven Architecture – Foundations and Applications, Second European Conference, ECMDA-FA 2006, Bilbao, Spain, July 10–13, 2006, Springer-Verlag Berlin Heidelberg

[Schätz & Spies, 2002] Schätz, B., Spies, K., Model-Based Software Engineering – Linking more and less formal product models to a structured process, in Proceedings of ICSSEA 2002 15th International Conference on Software and Systems Engineering and their Applications, 2002

References

[Schätz et al., 2002] Schätz, B., Pretschner, A., Huber, F., Philipps, J., Model-based development of Embedded systems, in Bruel, J.-M., Bellahsene, Z. (eds.), Advances in Object-Oriented Information Systems – OOIS 2002 Workshops, Montpellier, France, Springer LNCS, 2002

[Schätz et al., 2003] Schätz, B., Romberg, J., Strecker, M., Slotosch, O., Spies, K., 2003, Modeling Embedded Software: State of the Art and Beyond, in Proceedings of ICSSEA 2003 16th International Conference on Software and Systems Engineering and their Applications, 2003

[Schätz et al., 2004] Schätz, B., Broy, M., Huber, F., Philipps, J., Prenninger, W., Pretschner, A., Rumpe, B., Model-Based Software and Systems Development – A White Paper. http://www4.informatik.tu-muenchen.de/~schaetz/papers/ModelBased.pdf (referenced 30.1.2009)

[Staron, 2006] Staron, M., Adopting Model Driven Software Development in Industry – A Case Study at Two Companies, Miroslaw Staron, in Model Driven Engineering Languages and Systems, in Proceedings of 9th International Conference, MoDELS 2006, Genova, Italy, October 1–6, 2006, Springer LNCS, 2006

[Uhl, 2008] Uhl, A., Model-Driven Development in the Enterprise, IEEE Software, January/ February 2008

[Vogel & Mantell, 2005] Vogel, R., Mantell, K., MDD process for an SME: evolution, not revolution – Phase 1, The First Workshop From code centric to model centric software engineering: Practices, Implementation and ROI, ECMDA-FA 2005, Nuremberg, Germany, 2005

[Vogel & Mantell, 2006] Vogel, R., Mantell, K., MDA adoption for s SME: evolution, not revolution – Phase II, The Second Workshop From code centric to model centric software engineering: Practices, Implications and ROI, EMCDA-FA 2006, Bilbao, Spain, 2006 http://www.esi.es/modelware/c2m/docum/papers/PAPER7-ECMDA2006-EnablerAdoption.pdf (referenced 30.1.2009)

Appendix A: Summary of the questions of the 1st MoSiS survey

Group A: "Background of the respondent" consisted of the following questions:

A.1 What is your age?
A.2 What is your role in the organisation?
A.3 What is the domain of your organisation?
A.4 What is the number of employees of your organisation?
A.5 What are the main products or services of your company?

Group B: "Personal experience of MDD" consisted of the following questions:

B.1 How extensively do you use model-driven development (MDD) in your work?
B.2 In my opinion, the MDD approach is useful

Group C: "Modeling processes and practices of MDD" consisted of the following questions:

C.1 How exactly is the software development process defined in your company?
C.2 How systematically are MDD related activities applied in your software projects?
C.3 What are the main reasons for NOT applying MDD?
C.4 In your opinion, what are the main challenges in your current application of MDD related activities?
C.5 In your opinion, what are the main benefits gained by the application of MDD related activities?
C.6 Based on your experiences, what MDD related issues should be improved or supported?

Group E: "MDD tools and methods in organisations" consisted of the following questions:

E.1 What is the software modelling language that you typically use?
E.2 Do you use code generation from your software models?
E.3 For what purposes do you use UML?
E.4 Which UML tools are in use in your organisation?
E.5 Which general-purpose languages do you use besides UML?
E.6 What DSM tool do you typically use?

Appendix B: Summary of the questions of the 2nd MoSiS survey

Group A: "Background of the respondent" consisted of the following questions:

A.1 How long is your experience with software development?
A.2 What is your role in the organisation?
A.3 What is the domain of your organisation?
A.4 What is the number of employees of your organisation?
A.5 What are the main products or services of your company?

Group B: "Utilisation of model-driven development (MDD) in company" consisted of the following questions:

B.1 How exactly is the Model-Driven Development (MDD) process defined in your company?
B.2 How far are you in applying the MDD technologies?
B.3 How long have you applied MDD technologies in your company?
B.4 How systematically are MDD related activities applied in your software projects?
B.5 How did you start the use of MDD?

Group C: "MDD and the product development process activities" consisted of the following questions:

C.1 What is your opinion about how much MDD technologies have influenced "Context definition" -process activity in companies. Please, describe how MDD has influenced on context definition -activity in companies.
C.2 What is your opinion about how much MDD technologies have influenced "Requirements elicitation and analysis" -process activity in companies. Please, describe how MDD has influenced on requirements elicitation and analysis -activity in companies.
C.3 What is your opinion about how much MDD technologies have influenced "Analysis" -process activity in companies. Please, describe how MDD has influenced on analysis -activity in companies.
C.4 What is your opinion about how much MDD technologies have influenced "Design" -process activity in companies. Please, describe how MDD has influenced on design -activity in companies.

C.5 What is your opinion about how much MDD technologies have influenced “Implementation” -process activity in companies. Please, describe how MDD has influenced on implementation -activity in companies.

C.6 What is your opinion about how much MDD technologies have influenced “Integration and testing” -process activity in companies. Please, describe how MDD has influenced on integration and testing -activity in companies.

C.7 What is your opinion about how much MDD technologies have influenced “Release” -process activity in companies. Please, describe how MDD has influenced on release -activity in companies.

Group D: “Maturity of modelling technology and related methods” consisted of the following questions:

D.1 In my opinion, the modelling technology used in the company should...

D.2 In addition to the modelling tools, the methods for using models should be mature. In my opinion, these methods include activities like:…

Group E: “Benefits with the model-driven development (MDD) technologies” consisted of the following questions:

E.1 What kind of benefits have you faced with the MDD technologies?

E.2 Please give your comments about those benefits.

Group F: “Challenges with the model-driven development (MDD) technologies” consisted of the following questions:

F.1 What kind of challenges have you faced with the MDD technologies?

F.2 Which of the challenges you would rate as the most important? Please tick the THREE most important.

Groups G: - R: “How important is this challenge for you, and do you have a solution / support for handling the challenge?” consisted of the following challenges:

G.1 Understanding and managing the interrelations among the multiple representations of software development artefacts representing different views or levels of abstraction of the same concepts.

H.1 Comparing and merging different versions of models, e.g., visualising the differences in a usable way.

I.1 Difficulties with transformations of models (to code or other models): whenever an interrelated artefact changes in ways that affect some or all of its related artefacts as the mutual consistency cannot always be automatically assured.

J.1 Model-level debugging is not sufficiently supported.

K.1 Combination of graphical and forms-based syntaxes with text views.

L.1 Moving complexity rather than reducing it (the MDD approach does not reduce the complexity visible to the developer or but simply moves complexity elsewhere in the development process).

M.1 More expertise required to understand the impact of a proposed change on all of the related artefacts that could be described in different notations.

N.1 Lack of modelling experts in organisations.

O.1 Difficult to select correct languages and tools and make them collaborate.

P.1 Difficult to synchronise models and metamodels.

Q.1 The bugs are more difficult to find and fix in the model than in traditional coding.

R.1 Other challenges.

Groups S: “Other comments about MDD”

S.1 Please, give your free comments about MDD.

VTT Working Papers

100 Bernd Ebersberger & Olavi Lehtoranta. Effects of Public R&D Funding. 2008. 27 p. + app. 1 p.

101 Stephen Fox, Patrick Ehlen, Matthew Purver, Elizabeth Bratt, Matthew Frampton, Ichiro Kobayashi, Bevan Jones, Rob Monroe & Stanley Peters. Applying computational semantics to the real-time communication of skill knowledge. 2008. 85 p.

102 Stephen Fox. Ontological uncertainty and semantic uncertainty in global network organizations. 2008. 122 p.

103 Kati Tillander, Helena Järnström, Tuula Hakkarainen, Juha Laitinen, Mauri Mäkelä, & Panu Oksa. Palokohteiden savu-, noki- ja kemikaalijäämät ja niiden vaikutukset työturvallisuuteen. Polttokokeet ja altistumisen arviointi. 2008. 67 s.

104 Eija Kupi, Sanna-Kaisa Ilomäki, Virpi Sillanpää, Heli Talja & Antti Lönnqvist. Aineettoman pääoman riskienhallinta. Riskit ja riskienhallinnan käytännöt yrityksissä. 2008. 44 s.

105 Teemu Mutanen, Joni Niemi, Sami Nousiainen, Lauri Seitsonen & Teppo Veijonen. Cultural Event Recommendations. A Case Study. 2008. 17 p.

106 Hannele Holttinen. Tuulivoiman tuotantotilastot. Vuosiraportti 2007. 2008. 44 s. + liitt. 8 s.

107 Kari Keinänen, Jarkko Leino & Jani Suomalainen. Developing Keyboard Service for NoTA. 2008. 17 p. + app. 2 p.

108 Hannele Antikainen, Asta Bäck & Pirjo Näkki. Sosiaalisen median hyödyntäminen paikallisissa mediapalveluissa. 2008. 64 s.

109 Raine Hautala, Pekka Leviäkangas, Jukka Räsänen, Risto Öörni, Sanna Sonninen, Pasi Vahanne, Martti Hekkanen, Mikael Ohlström, Bengt Tammelin, Seppo Saku & Ari Venäläinen. Benefits of meteorological services in South Eastern Europe. An assessment of potential benefits in Albania, Bosnia-Herzegovina, FYR Macedonia, Moldova and Montenegro. 2008. 63 p. + app. 35 p.

110 Jaana Leikas. Ikääntyvät, teknologia ja etiikka. Näkökulmia ihmisen ja teknologian vuorovaikutustutkimukseen ja -suunnitteluun. 2008. 155 s.

111 Tomi J. Lindroos. Sectoral Approaches in the Case of the Iron and Steel Industry. 2008. 58 p. + app. 11 p.

112 Johan Mangs. A new apparatus for flame spread experiments. 2009. 50 p. + app. 27 p.

113 Stephen Fox & Brent Stucker. digiproneurship. New types of physical products and sustainable employment from digital product entrepreneurship. 2009. 29 p. + app. 7 p.

114 Päivi Parviainen, Juha Takalo, Susanna Teppola & Maarit Tihinen. Model-Driven Development. Processes and practices. 2009. 102 p. + app. 4 p.

ISBN 978-951-38-7175-8 (URL: http://www.vtt.fi/publications/index.jsp)
ISSN 1459-7683 (URL: http://www.vtt.fi/publications/index.jsp)