# A Selection Method for Model-Driven Development Tools

*Master's Thesis*

Remco Luitwieler

# A Selection Method for Model-Driven Development Tools

MASTER THESIS

in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Remco Luitwieler
born in Middelburg, the Netherlands

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Logica
George Hintzenweg 89
3086 AX Rotterdam, the Netherlands
www.logica.com

© 2010 Remco Luitwieler.

# A Selection Method for Model-Driven Development Tools

Author: Remco Luitwieler
Student id: 1159828
Email: R.Luitwieler@student.TUDelft.nl

**Abstract**

This thesis project helps Logica adopt Model-Driven Development in an efficient way. A Modeling Tool Selection Method is constructed that helps selecting a Model-Driven Development tool at the start of an embedded system engineering project. This Modeling Tool Selection Method helps software engineers to decide when to use which Model-Driven Development tool. During a case study, three Model-Driven Development tools are used to develop software for a charge point for electric vehicles. The use of the Modeling Tool Selection Method to select a tool for a project is demonstrated and compared to the results of the case study. With a validation of the Modeling Tool Selection Method and a validation of the conducted case study, we show that the Modeling Tool Selection Method is constructed accurately.

Thesis Committee:

| | |
|---|---|
| Chair: | Dr. E. Visser, Faculty EEMCS, TU Delft |
| University supervisor: | Dr. Phil. H.G. Gross, Faculty EEMCS, TU Delft |
| Committee member: | Dr. K. van der Meer, Faculty EEMCS, TU Delft |
| Company supervisors: | Mr. A.J. de Neef, Logica |
| | Ir. E.C. Essenius, Logica |
| Company mentor: | Ir. N.T. Quach, Logica |

# Preface

First of all, I like to thank all the people from Logica who helped me during my graduation period. There was always time for questions and some of their work helped the progress of this research. I would also like to thank Hans-Gerhard Gross, my supervisor at the TU Delft, for his feedback and the advice he gave during the meetings we had.

Remco Luitwieler
Delft, the Netherlands
June 12, 2010

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

This Chapter introduces all aspects forming the context and motivations of the thesis project. We will describe Model-Driven Development, the scope of the thesis, the research questions, the research approach that was taken, and the contributions of this thesis to science and Logica. Finally we will describe what the reader can expect throughout the Chapters of this document.

**Model-Driven Development (MDD)** is a software engineering approach consisting of the application of models and model technologies. MDD raises the level of abstraction at which developers create and evolve software, with the goal of both simplifying and formalizing the various activities and tasks that comprise the software life cycle [23]. Since the use of models is common good in traditional engineering disciplines as civil and mechanical engineering [44] many people think that software engineering can also benefit. With the higher level of abstraction that the models offer, it is easier to understand how complex systems work when using Model-Driven Development. Introduction of a Model-Driven Development process forces organizations to transform their software development process [29] and this thesis includes the evaluation of the impact of using Model-Driven Development tools in the context of Logica.

Logica is a large IT and management consultancy company that employs 39.000 people across 36 countries. Logica wants to know how Model-Driven Development can be used within their company. The use of Model-Driven Development involves some major challenges that have to be tackled in order to realize it [18]. Logica is interested in a matrix that classifies Model-Driven Development tools and their usability for software engineering processes of Logica.

The **goal** of the thesis project is to help Logica adopt MDD in an efficient way. This thesis will describe the construction of a selection method and a matrix that decides which MDD tools best fit a particular type of software engineering project. To validate the constructed selection method and the matrix a case study is conducted. The construction of the case study allows extensive use of the MDD tools that are under selection and checks the correctness of the selection method and matrix for the MDD tools.

The **research questions** and their sub questions investigated in this thesis are:

1. To which extent can Model-Driven Development be used in the context of Logica?
   a) What changes are needed in the current development process?
   b) How can these tools help in delivering higher quality code for a low price?
   c) When to use which tool?
2. How can the selection method be improved with knowledge from future projects?
   a) To which extent can we add knowledge gained during projects?
   b) How to add tools to the method/matrix?

The **research approach** used during this thesis project is the construction of a selection method and a comparison matrix for MDD modeling tools. For evaluation a case study will be conducted that evaluates the selection method and matrix by using the tools in a practical environment. The case study will be validated to qualify its quality. The results from this research will be discussed in the conclusion and can be used by other companies or for future research. Developing a case study [51] is useful when investigating a contemporary phenomenon within its real-life context [46]. As described in [16] it is possible to generalize from a specific case. The specific case that will be looked into during this research is the development of a charge point for electric vehicles. This charge point or load pole is equipped with an embedded Linux controller [47] that communicates with peripherals such as a cardreader and a ledring. The software that runs on the controller will be developed during this case study using Model-Driven Development tools. These tools are:

- Rhapsody from IBM (v7.5.1)
- Analytical Software Design from Verum (v5.0.0)
- Bouml from Bruno Pagés (v4.19.1)

Logica is interested in these tools since both IBM and Verum are business partners of Logica and this gives advantages when licenses are needed, and in some cases training of developers can be arranged for a reduced rate. With ASD a pilot project is already started and with Rhapsody positive experiences are gained during previous research and pilot projects. Bouml is added during this thesis project, because it is open source and is frequently updated. Bouml is one of the few open source tools that is capable of design and code generation in the same tool. Bouml is multi-platform and capable of code generation in a wide range of programming languages. The development process will be monitored and the tools will be compared on criteria as ease of use, development time, generated code size and capabilities of the tools.

This thesis introduces and evaluates Model-Driven Development in the context of a company. A Modeling Tool Selection Method is constructed that helps selecting a MDD tool at the start of an embedded system engineering project. The MTSM currently decides over three tools, but there can be added more tools when a case study is conducted. That case study allows the researcher to evaluate the tool and adjust his mental model and the MTSM according to it. Most valuable **contribution** for Logica is the matrix that quickly decides on the use of the tools. The contribution for science is the introduction of the, during this thesis constructed, Modeling Tool Selection Method (MTSM) and the reflection of it during the case study. The validation of the MTSM in Section 4.7 and the validation of the case study in Section 3.6 show that the MTSM is constructed accurately.

The next Chapter introduces the reader with the concepts of Model-Driven Development, the tools and the Unified Modeling Language (UML) which is extensively used during this research and describes the related work that has been done in the field of Model-Driven Development. Chapter 3 introduces the Modeling Tool Selection Method that helps selecting a tool to use during a software engineering project. In Chapter 4, a case study is conducted and the results are translated to the method that was introduced in Chapter 3. In the final Chapter, a summary is given, the conclusions are drawn, and recommendations for future research are given.

# Chapter 2

# Background and Related Work

This Chapter describes the background of the thesis and work that is related to this thesis project. Concerning the related literature, we made a distinction between Related and Essential work and Related but Borderline work to clarify the scope of the project.

First, Model-Driven Development and Model-Driven Engineering are described and then the Model-Driven Architecture, Platform-Independent Models (PIMs) and Platform-Specific Models (PSMs) are introduced. The description of a model is given and UML-diagrams are described and discussed. After that, the tools that are being used are introduced and discussed, and there is a section about knowledge management. In the summary, the relation of the subjects with the thesis project is described.

## 2.1 Model-Driven

**Model-Driven Development** (MDD) raises the level of abstraction by using models of a system. Model-Driven Development is described in [7] as a paradigm which promotes the use of models at different levels of abstraction, and transformation between them, in order to drive a concrete application implementation. MDD promotes the construction of models at a high level which can then be transformed to the final implementation platform. The modeling language in MDD is often the commonly known Unified Modeling Language [14][32]. The combined use of MDD and UML2.0 is described in [19]. They discuss UML2.0 as being a first-generation modeling language and evaluate what are the positive and negative points in the standard. Both UML and MDD are extensively used during this project and therefore this is related and essential work.

**Model-Driven Engineering** (MDE) is described by [43] as a technology that combines domain-specific modeling languages, transformation engines and generators. MDE is seen as an evolution of the Computer Aided Software Engineering (CASE) tools that were introduced in the 1980s. These tools focus on developing software methods and tools that enable developers to express their designs in terms of general purpose graphical programming representations, such as state machines, structure diagrams, and data-flow diagrams. MDE is wider in scope then MDD and MDA [2][31]. The evaluation of MDE by using a case study in a large industrial context is described in [5]. This paper [5] describes in short the experiences of using MDE and the use of tools for certain parts of the MDE process and the lack of tools for the whole MDE process. MDE and the use of it in an industrial context is related but borderline work.

**Model-Driven Architecture** (MDA) is a Model-Driven Development approach defined in 2001 [33] by the Object Management Group (OMG) [32] that defines an information technology system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform. This MDA approach and the standards that support it allow the same model specifying system functionality to be realized on multiple platforms through auxiliary mapping standards, or through point mappings to specific platforms, and allows different applications to be integrated by explicitly relating their models, enabling integration and interoperability and supporting system evolution as platform technologies come and go [33]. The use of MDA in MDE is described in [31]. In MDA there is made a separation between a Platform-Independent Model (PIM) and a Platform-Specific Model (PSM). The PIMs provide a formal specification of the structure and function of the system that abstracts away technical details. How the functionality specified in a PIM is realized, is specified in a platform-specific way in the PSM, which is derived from the PIM via some transformation [33][31]. Since OMG is the founder of MDA, that provides a framework which integrates the existing OMG standards, it is no surprise that it is intended that all PIMs and PSMs are being expressed in the Unified Modeling Language (UML) which is also described by the OMG [32][31]. Figure 2.1 shows that models of different systems can be structured explicitly into PIMs and PSMs.

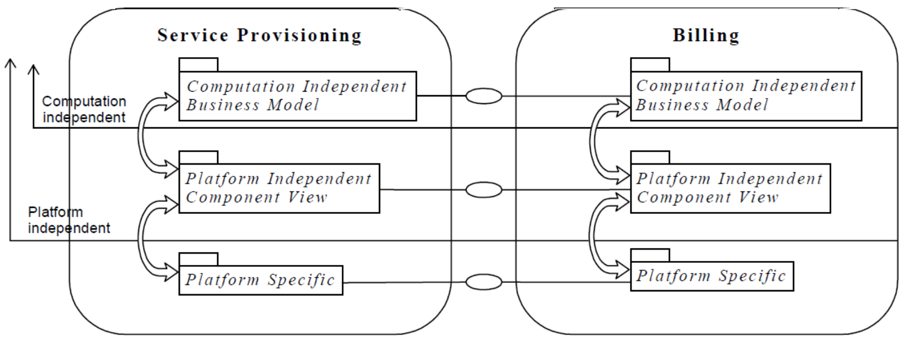

Service Provisioning
Billing
Computation independent
Platform independent
Computation Independent Business Model
Platform Independent Component View
Platform Specific
Computation Independent Business Model
Platform Independent Component View
Platform Specific

Figure 2.1: Consistent model separations and relationships in MDA [33].

## 2.2 Model

In [6] we learn that a model is defined as a simplified version of something complex used. The consensual definition of modeling was given by [41] as:

> ”Modeling, in the broadest sense, is the cost-effective use of something in place of something else for some cognitive purpose. It allows us to use something that is simpler, safer or cheaper than reality instead of reality for some purpose. A model represents reality for the given purpose; the model is an abstraction of reality in the sense that it cannot represent all aspects of reality. This allows us to deal with the world in a simplified manner, avoiding the complexity, danger and irreversibility of reality.”

From [31] and [33] we learn that the Object Management Group describes a model in its Model Driven Architecture as a representation of a part of the function, structure and/or behavior of a system.

### 2.2.1 UML Diagrams

UML offers the use of several types of diagrams to display the system model with. The diagrams UML2.0 offers:

- Use-Case Diagram: *Connects external actors to the descriptions of functionality that the system offers*
- Class Diagram: *Shows the static structure of classes in the system*
- Object Diagram: *Shows the object instances of classes at a certain moment in time (a snapshot during execution)*
- State Machine Diagram: *Shows all the possible states that objects of a class can have during its life-cycle, modeling complex interactive behavior*
- Activity Diagram: *Shows a sequential flow of actions*
- Interaction Diagrams: *Diagrams that show interaction between objects during the execution of the software*
- Component Diagram: *Shows the physical structure of the code in terms of code components*
- Deployment Diagram: *Shows the physical architecture of the hardware and software in the system*
- Composite Structure Diagram: *Shows the participating elements and their relationships in the context of a specifier such as a use, object, collaboration, class or activity*

The most important UML diagrams are the Class Diagram to model the structure of a system, and the State Diagrams and Activity Diagrams to model the behavior of a system. For a complete description of the diagrams see [14].
UML distinguishes between two types of state machines:

- Behavioral state machines that describe all the details of a class its life cycle
- Protocol state machines that focus on the transitions of states and the rules governing the execution order of operations.

Behavioral state machines can become quite complex as they try to model concurrent behavior, show nested states and allow for redefinition. This is also the strength of the state machines because they are able to describe the behavior of the possibly complex system with these diagrams.

Protocol state machines can provide rules for implementation by interfaces or ports. These rules provide the guidelines that other systems must comply with in order to work with the associated class. The protocol state machine helps UML support component-based development by providing clear interfaces and rules for the communication between different objects.

## 2.3 Tools

During the case study, whose goal is to investigate the use of the tools and validate the Modeling Tool Selection Method by reflecting the results of the case study, three software tools are compared that are designed for use in a MDD environment. This section introduces the tools and their capabilities are discussed.

In [4] modeling tools for model validation are compared and modeling languages are described and used.

### 2.3.1 ASD

Analytical Software Design (ASD) is designed by Verum and combines formal methods, Sequence-Based Specification (SBS), Communicating Sequential Processes (CSP) and Failure Divergence Refinement (FDR) to design and mathematically verify the correctness and completeness of complex software systems [9].

Model-Driven Development with ASD is also researched by other large companies as Sioux [45] and the R&D department of Philips: Philips Applied Technologies [36]. The results from the research by Philips in cooperation with the vendor of the company is published in [10] and seems promising towards the use of ASD. The results that Sioux measured, are published in a white paper on the website of Verum [49] and are also in favor of choosing to use ASD above other MDD tools.

The following description of how ASD works is related but borderline. Formal methods used in developing computer systems are mathematically based techniques for describing system properties [50]. Formal methods provide the means of proving that a specification is realizable, proving that a system has been implemented correctly, and proving properties of a system without necessarily running it to determine its behavior [50].

Sequence-Based Specification is a collection of techniques for implementing rigorous, practical software specification. The central techniques, sequence enumeration and sequence abstraction, provide a systematic way to explore and discover intended system behavior in a complete, consistent and traceable correct way [37].

Communicating Sequential Processes [24] is a formal language for describing patterns of interaction. CSP uses mathematical models and reasoning methods to describe concurrent systems whose component processes interact with each other by communication [40]. Concurrent systems are systems where multiple processes can exist at the same time.

Failure-Divergence Refinement is a model-checking tool for state-machines. FDR has foundations in the theory of concurrency based around CSP [15]. FDR checks whether properties defined in CSP hold, and is able to check determinism of a state machine.

ASD uses Sequence-Based Specification to specify functional requirements and designs as total black box functions. Sequence-Based Specification is also used to map every possible sequence of input stimuli to a response. The ASD Model Generator generates CSP models automatically from these specifications and designs. These CSP-models can be analyzed and verified by FDR. ASD also has a code-generator that can generate a significant amount of program source code in C++, C or other similar languages according to the former specifications [9]. Size of the code that can be generated differs per project, but according to experiences from Verum [9] it generates about 70 to 90 percent of the total amount of programming code for the project.

In Figure 2.2 we see how ASD fits in the design process.

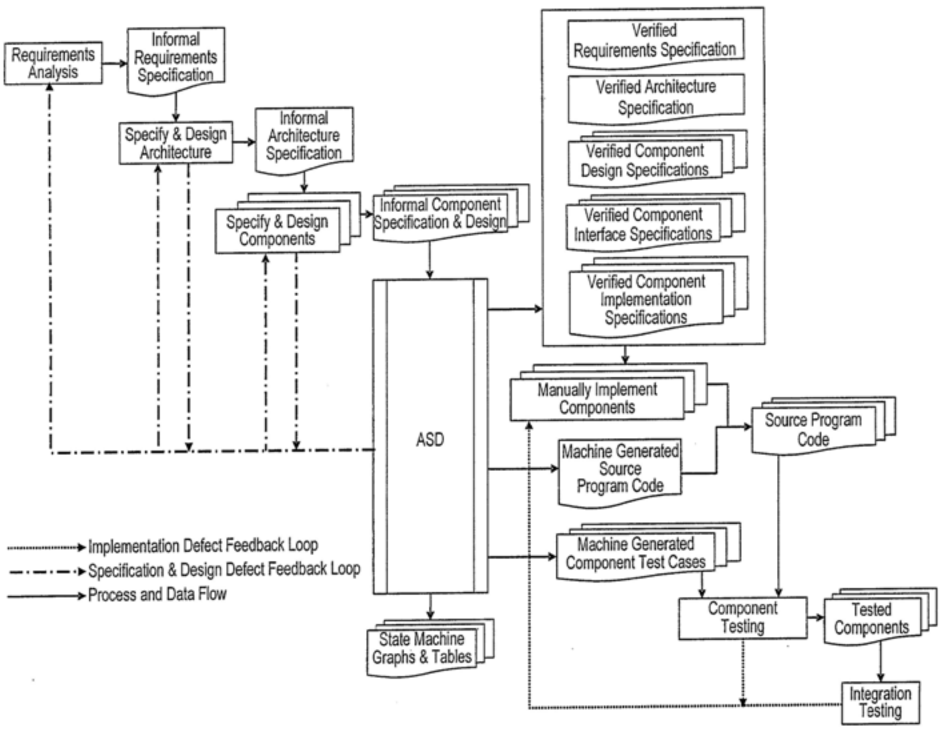

Requirements Analysis
Informal Requirements Specification
Specify & Design Architecture
Informal Architecture Specification
Specify & Design Components
Informal Component Specification & Design
ASD
Verified Requirements Specification
Verified Architecture Specification
Verified Component Design Specifications
Verified Component Interface Specifications
Verified Component Implementation Specifications
Manually Implement Components
Machine Generated Source Program Code
Source Program Code
Machine Generated Component Test Cases
Component Testing
Tested Components
Integration Testing
State Machine Graphs & Tables
Implementation Defect Feedback Loop
Specification & Design Defect Feedback Loop
Process and Data Flow

Figure 2.2: Methodology of ASD from [11].

### 2.3.2 Rhapsody

Rhapsody is a UML based software development tool, designed to support complete model-based iterative life-cycle [21]. How MDA relates to Rhapsody is described in [13]. Rhapsody is designed for real-time and embedded systems engineering, software development and testing based on UML/SysML. With Rhapsody the programmer gets a clear view of how the requirements are implemented and is able to generate programming source code and there is a possibility to test the designed models using a simulation. With this simulation the programmer sees if everything is working as he expected it to work and he can keep track of the communication in the models. Rhapsody offers the programmer the ability to design the system by using different UML-models for different parts of the system. When all the requirements are modeled in the system the programmer can generate code. The programmer can now change the model or the code, but whatever he chooses, the other also adapts to the changes. This makes Rhapsody-projects flexible for changes when the requirements change.

### 2.3.3 Bouml

Bouml works in a similar way as Rhapsody. UML2.0 models are created and from these models programming code is generated. Bouml is developed by one developer and is considered fast and careful with its memory usage compared to Rhapsody and other UML-modeling tools [8]. Bouml can also reverse engineer code and since it is open source it is possible to integrate extra functionality called plug-outs. Some useful plug-outs can be found on the website of Bouml and are easy to integrate with the tool. Bouml does not offer a simulation like Rhapsody, but there is automatically added debugging code to monitor the behavior during run-time. Bouml offers simultaneous programming in C++, Java, Php, Python and IDL.

### 2.3.4 Differences between Rhapsody, Bouml and ASD

ASD comparing to Rhapsody (or Bouml) is difficult because Rhapsody is a model-driven development tool that helps the programmer to create models and to give a higher level of abstraction about the system that is being designed. ASD helps the programmer checking the requirements in a way that the requirements need to be complete and correct and ASD also checks the model that is created from these requirements. So, ASD checks the model whereas Rhapsody does not check the model, but Rhapsody gives the programmer the ability to design by using different types of models. This way the programmer can get a clear view of how the system operates and therefore should be able to check the correct functioning according to the models by hand. Off course this is not as robust as letting ASD check, but it is more robust then not checking at all. Rhapsody and Bouml are typical MDA-compliant tools [13] because MDA persists the use of UML-models to develop a Platform-Independent Model of a Platform-Dependent Application (PDA). ASD is not MDA-compliant because it does not uses UML models to create a PIM, but since it uses the Sequence Based Specification to construct a model ASD is MDD-compliant.

### 2.3.5 Relation with MDD

In Model-Driven Development it holds that models are used as primary artifacts for which efficient implementations are generated by the application. Rhapsody is a typical MDD tool, whereas ASD is a MDD tool, but in a smaller subset of the field. Rhapsody uses the complete range of UML models, introduced in Section 2.2.1, to describe the behavior of the system and generate code according to that models and behavior. ASD generates a model according to the specifications that are entered with respect to the requirements. The model that ASD generates from the specifications is a Communicating Sequential Processes model that will be checked for completeness and correctness. To get a global view, ASD is also able to generate a State Diagram. From the CSP-model it is possible to generate programming code automatically in ASD. Bouml also uses UML models to describe the architecture and behavior of a system and generates code according to the specification of these models.

## 2.4 Knowledge Management

Knowledge management is an emerging discipline that promises to capitalize on organizations intellectual capital [42]. Software development is a quickly changing, knowledge-intensive business involving many people working in different phases and activities. In software development, every person involved constantly makes technical or managerial decisions. Most of the time, team members make decisions based on personal knowledge and experience or knowledge gained using informal contacts [42]. Knowledge management is also an issue for companies such as Logica, because the knowledge gained during the projects is often not documented or reflected. The selection method introduced in the next Chapter transfers specific knowledge into a selection method, and allows the adaption of knowledge from future projects in the selection method. By using the selection method the knowledge is equal for all persons using the selection method. Knowledge management is related and essential since the introduced method will become part of the knowledge management of Logica.

## 2.5 Summary

This Chapter introduced the subjects and researches that are related to this thesis project. There are several researches that are related to this thesis project and during this Chapter some interesting researches are shortly described and a reference is given. The researches are also used to describe several subjects that are related to this thesis project.

We first described MDD, MDE and MDA, these concepts are often used during this thesis and the distinction should be clear. The description of PIMs and PSMs is important in the context of this thesis because it is one of the essential concepts of MDA. The generic concept of MDD is very closely related to the usage of PIMs and PSMs. The PIM shows the model without all the details and the PSM is the realization of this model for a specific platform. The models of the PIM are thus not platform specific and this is one of the advantages of using a Model-Driven approach, because this allows the developer to model on a higher level of abstraction. We then give the definition of a model since this explains why we want to use models, because they describe things in a simplified manner. Then the different UML diagrams are described because we use the UML to model in during the case study.

The Section about the tools introduces the modeling tools that we are using and some work that is related to the tools and this thesis project. A description of how the tools work, and some of the main differences are given. The relation of the tools with MDD and MDA is also given.

Finally, there is a Section about knowledge management which is also related to this thesis project because the, in the next Chapter, introduced Modeling Tool Selection Method holds important knowledge for Logica. And the MTSM is capable of adopting knowledge which makes it part of the knowledge management of Logica.

# Chapter 3

# Modeling Tool Selection Method

This Chapter describes the selection method that is constructed during this thesis project. The selection method uses questions to determine the processes and products of a software engineering project and concludes on the tool that supports the development process and products best. Also a selection matrix will be introduced that quickly decides over the use of an MDD tool. The tools were introduced in Chapter 2. In the first Section the meta-process model is described which supports the creation of the method. The Modeling Tool Selection Method (MTSM) is described in the second Section. In the third and fourth Section the actual method is constructed and then the selection matrix is introduced. Finally, the method will be validated by using results from other companies and cross-checking the constructed method with developers that have experience with the use of the particular tools.

## 3.1 Meta-Process Model

The abstraction levels in process modeling are displayed in Figure 3.1 from [39]. There are three levels in which the Process Meta-Level is the highest level. The Process Meta-Level describes the generic concepts of the second level, the Process Model. The Process Model corresponds to the way of working prescribed by the methodology in use in the lowest level, the Development Runs, which is the actual process that outputs a product. The three levels

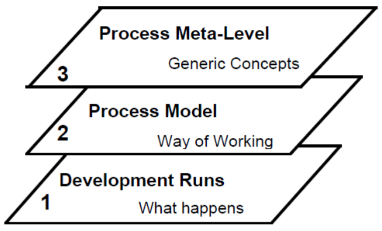


Figure 3.1: Abstraction levels in process modeling from [39]

of Figure 3.1 can be mapped to the Meta-Process Model of the Modeling Tool Selection Method as described in Table 3.1.

| Abstraction level | Meta-Process Model of MTSM |
|---|---|
| 3 Process Meta-Level | The Meta-Process Model displayed in Figure 3.2 |
| 2 Process Model | Asking questions about Processes, Products and Tools in Sections 3.3, 3.4 and Figure 3.3 |
| 1 Development Runs | Actual use of the MTSM to select a tool, Figure 3.4 |

Table 3.1: Mapping of abstraction levels

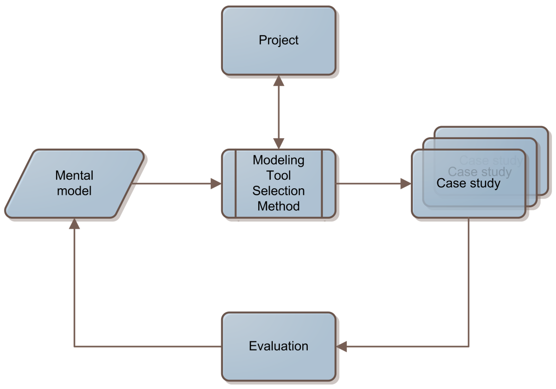


Figure 3.2: The Meta-Process Model

In the Meta-Process Model of Figure 3.2 we see that the Modeling Tool Selection Method is constructed from the mental model of a Software Engineer. A mental model is the way a person understands a particular domain of knowledge [20]. The domain of knowledge in Figure 3.2 is the use of MDD tools in a software engineering project. The main construction of the MTSM is done during this thesis and in the future changes are allowed to be performed. The created MTSM can be used to select a MDD tool for a project, or the other way around, as indicated with the double ended arrow in Figure 3.2. From project to MTSM is the selection of important products and processes when the selection of a tool has already been made. For instance, when the principal suggests the use of a particular MDD tool. It is also possible to change the MTSM. Changing the MTSM is done in the following way:

1. Use MTSM to select a tool or multiple tools
2. Use the tool in a case study or use multiple tools in the same case study
3. Evaluate the results of the case study
4. Adjust mental model according to the findings during evaluation
5. Adjust MTSM to cope with the mental model

We have described how the MTSM can be used at the highest level. The next Section describes how the MTSM is constructed (2 Process Model) and how the MTSM is used (1 Development Run).

## 3.2 Description of the Method

The Modeling Tool Selection Method (MTSM) is used to gather information of the development process at the start of a software engineering project. The gathered information is used to determine which MDD tool fits the project best. Not all software engineering projects are suitable for using a MDD-approach and therefore we will restrict the MTSM to Embedded Systems that do not require real-time computing during this research. An overview of the process of the model is displayed in Figure 3.3.

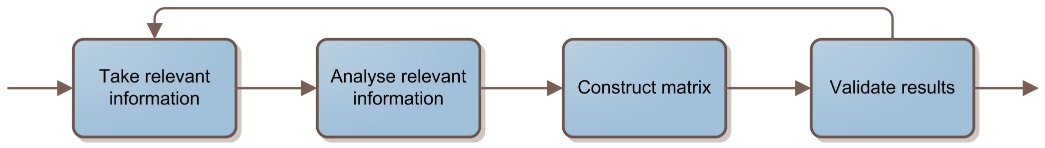


Figure 3.3: The process model of the Modeling Tool Selection Method

Eventually, when the MTSM is actually used, it takes the Business Processes and Products, and the Tools, see Figure 3.4, and asks questions to analyze them. The answers to these questions are used to select one out of three modeling tools that ensures an effective process along that particular project or it concludes that the use of one of the three tools is not appropriate. Eventually the MTSM determines which tool is the best to use.

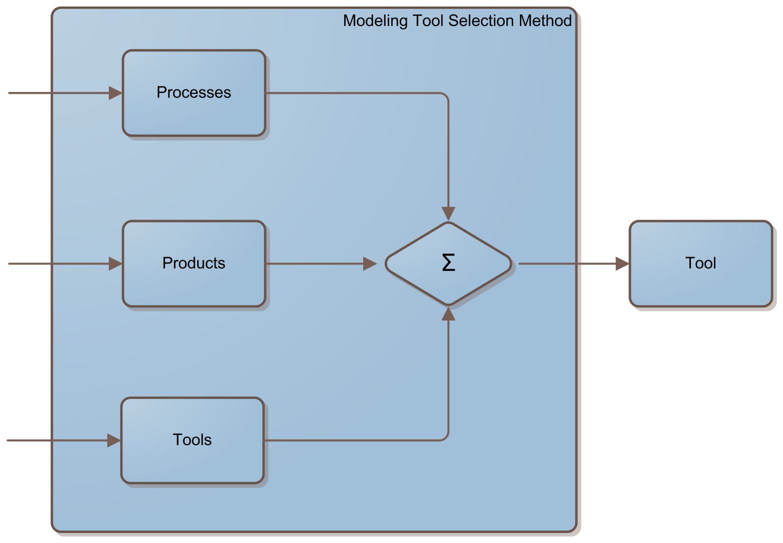


Figure 3.4: Overview of input and output during a development run of the MTSM

The questions in the Business Processes and Products Section are used to find out how the developers are going to work, are they developing a prototype or a high availability system? Are the requirements complicated and possibly incomplete? Is the system a component that uses other components from foreign developers and needs to fit in exactly? During the case study in Chapter 4 this Modeling Tool Selection Method will be validated by using a practical case that shows how this method works and shows the advantages and disadvantages per tool.

## 3.3 Business Processes and Products

This Section describes the Processes (determining what information is available) and Products (the documents that are available) that are available when choosing the right approach for developing a project. The Modeling Tool Selection Method uses the following common activities and for each activity there need to be answered certain questions. It is not mandatory to follow these activities during development since these activities are only used to get a view of what kind of project the software is developed for.

**Requirements** What quality are the requirements?

**Analysis** What platform and language are demanded by the customer? What is the budget of the project?

**Design** What diagrams will be used? What is the quality of the models?

**Implementation** How many components are involved?

**Testing** Are there tests available? Is there a testing environment? What is the importance of quality?

**Maintenance** Are future extensions needed?

Defining the quality of **requirements** is very hard [12]. The MTSM uses sub-questions to determine the quality of the requirements:

- Distinguish between common natural language, structured natural language and formalized language

From [12] we learn that natural language is imprecise and ambiguous. In case of using ASD the requirements should not be ambiguous and imprecise since the tool forces the developer to implement the requirements precisely. Rhapsody and Bouml are less strict, but they also force the developer to think about how the system interacts.

During the **analysis** of the project the programming languages are considered. Here we use Table 3.2 to determine the applicability of the tools.

| | ADA | C | C++ | C# | Idl | Java | Php | Python |
|---|---|---|---|---|---|---|---|---|
| ASD | | X | X | X | | X | | |
| Bouml | | | X | | X | X | X | X |
| Rhapsody | X | X | X | | | X | | |

Table 3.2: Tools and programming languages support

Other considerations during analysis are the platform that is developed for and this raises the following sub-questions:

- Can the generated code run on the target platform or is cross-compilation necessary?
- What is needed to cross-compile for the target platform?

If the tool generates code directly for the platform this saves time since cross-compilation also takes time and raises extra dependencies. Rhapsody uses a run-time framework that needs to be linked to. So when using Rhapsody on a new target platform (Rhapsody was not used with this target platform before) a static link to this framework needs to be created. ASD uses the Boost libraries when using C++ and these need to be installed on the host that cross-compiles for the target. Bouml has (initially) no specific dependencies and only needs to be cross-compiled. When Rhapsody is configured correctly it creates a platform specific makefile and this makes adding files or dependencies easier among ASD and Bouml where the makefile needs to be changed manually.

The budget of the project is also interesting to consider since the costs of the tools are displayed in Table 3.3. In the Table the license costs of each tool is given, for ASD the average cost per Executable Line of Code (ELOC) is also given in the Table. The actual cost per ELOC depends on the number of times the model is verified. License costs for Rhapsody and ASD are very expensive compared to Bouml. With ASD every verification run is billed and this makes the costs hard to predict at the start of a project. With Rhapsody there are additions possible for integration with other tools from IBM, such as Doors and Testconductor for additional costs [25]. Only comparing on costs is not fair and therefore we consider the return on investment of the tools which refers to the costs made during the usage of the tool and the gain by using the tool. The return on investment is a combination of the costs and the development time needed when the tools are used. For instance if Bouml is free, but requires weeks of extra work compared to Rhapsody it is more expensive than Rhapsody that has a higher initial cost, but saves these extra weeks of work. Another cost that is important is that of learning the tool. For ASD and Rhapsody training courses at the vendors are available. For Bouml there are no trainings available. According to estimations of Software Developers and Software Architects at Logica, the effort of using ASD and Rhapsody is equal for a particular project, for Bouml there are no estimations made. This results in higher return on investment when the development team is equal and the project size is expanding. For smaller projects Bouml has the highest return on investment since there are no additional costs for using the tool. To make harder statements on the return on investment, results of future projects should be analyzed.

| Tool | Estimation of costs |
|---|---|
| ASD | 2,50 euro per Executable Lines of Code [49] and 2500 euro seat/year |
| Bouml | Free |
| Rhapsody | 12000 euro per seat (floating) |

Table 3.3: Approximation of costs

The **design** phase of the project involves the actual use of models that describe the system. Determining the quality of UML models can be done in different ways according to [38]. Most important aspects that pertain the quality of UML models are: the proportion and completeness of UML designs and the design–code correspondence [38]. Important here are the differences between the use of models of the tools. Where ASD uses the models to summarize the formal description, Bouml and Rhapsody use models to describe the behavior of the system. ASD asks more effort from the developer than the other tools. The overview can be lost during system design, due to the view that ASD offers. Rhapsody and Bouml offer more levels of detail, and therefore it is easier to understand the behavior of the system.

The **implementation** of the software is important since this phase often connects the software to other parts of the system. The strictness that ASD asks from the developer during the development helps him connect to the other parts, since the interfaces are thoroughly defined and fit immediately. Rhapsody and Bouml are less strict in defining the components and certain situations (such as exceptional situations) might not be in the model. This might not be a problem until the system is implemented and needs to connect with other parts of the system.

**Testing** activity is where the software is tested for correct functionality. For Rhapsody there are test suites that can be used to automatically generate tests. With ASD the system is guaranteed to be the same as the ASD model, and therefore it is complete and correct and testing is not necessary. Although testing is not necessary Verum recommends to test as usual since defects (misinterpretation of requirements) in the model are also guaranteed to be in the generated code. With Bouml the generated code can be tested by using for example unit tests. With all three tools the conventional testing phase is kept as is.

The final important activity is the **maintenance** of the project. When using MDD to adopt changes, the model changes, but changes in one model could affect other models. The learning time of the tool and ease of use of the tool are important since the developer needs to have a good view of how the system works. Changing the model and not only the code keeps the design the same as the code which is a great advantage of using MDD [38].

## 3.4 Tools

This Section classifies the tools by following the activities from the previous Section. The classification was constructed during the extensive research of the tools and uses a scale (-, 0, +) to distinguish between the tools. Where - means a negative score, 0 medium and + positive. The tools are compared to each other and are not rated as a whole since this is impossible when looking to only three MDD tools. The classification is the result of using the tools during the case study and evaluating afterwards to construct a mental model. In the next Section the classification that resulted from this mental model has been cross-checked by architects and developers of Logica and other companies to verify them. Classification will be done on the following criteria:

**Usability** How easy is the tool to use for developers? (- not easy, 0 medium, + easy)

**Understandability** Are the models well formatted and easy to understand? (- not easy, 0 medium, + easy)

**Correctness** Is the generated code working? Is the implementation of UML correct? (- incorrect, 0 medium, + correct)

| | ASD | Bouml | Rhapsody | Activity involved |
|---|---|---|---|---|
| Usability | 0 | 0 | + | Design and implementation |
| Understandability | - | 0 | + | Design |
| Correctness | + | + | + | Verification/testing |
| Development time | - | 0 | 0 | Design and implementation |
| Code size | 0 | + | 0 | Analysis |
| Executable size | 0 | + | 0 | Analysis |
| Pre-release defects | + | 0 | 0 | Testing |
| Maintainability | - | 0 | + | Maintenance |
| Strictness | + | - | - | Requirements and Implementation |

Table 3.4: Rating of the tools

**Development time** How long does the process take from modeling to generated code? (- long, 0 medium, + short)

**Code size** What is the size of the automatically generated code? (- large, 0 medium, + small)

**Executable size** How large is the resulting executable? (- large, 0 medium, + small)

**Pre-release defects** Is the number of defects before release lower than conventional software engineering methods? (- higher, 0 equal, + lower)

**Maintainability** How easy is it to change models/variables? (- not easy, 0 medium, + easy)

**Strictness** Does the tool require strict/closed requirements? (- not strict, 0 medium, + strict)

Some of these criteria like usability and maintainability are known from the ISO/IEC 9126 standard that classifies software quality and is described in [26]. In [30] there is a set of metrics proposed to support the selection of tools for software quality management. The other criteria are selected on their ability to help forming the mental model and these criteria can be related to the activities mentioned in the previous Section (Section 3.3). Closely related are for instance the usability, understandability and maintainability since they give an idea of the level of knowledge a developer should have for each tool. The ease of use is important to give flexibility among the development team in such a large company as Logica.

Table 3.4 gives an indication of the positive and negative points of the tools when compared to each-other. Not all points are equally important and should only be used to give an indication on the tools. What we can read from the Table is that Rhapsody is the most mature and broad MDD-tool. We can also see that ASD is very strict in the Requirements and Design. Bouml is the tool that scores best on code and executable size since it has less dependencies, described in Section 3.3, in a relatively small project. During the case study the code and executable size will be discussed more thoroughly. This Table is validated by a Software Architect of Logica.

## 3.5 Matrix

Finally the MTSM leads to a **matrix** that helps deciding which tool is most useful for developing a particular software engineering project. Information gathered during the two phases of the MTSM are combined, see Figure 3.5. We read this matrix from left to right and check for every activity which tool is recommended. The colors are only used to clarify the boundaries. The tool that is checked most is selected to use during the project. The user can prioritize the activities on importance as he likes. We will now describe the columns of the matrix:

**Requirements** Here a distinction is made between the type of requirements. When there are formal requirements given, or there is need for them, the use of ASD is recommended. Rhapsody can be used with any type of requirements given. Bouml can best be used when the requirements are given in common natural language or structured natural language.

**Budget** This refers to the budget available for conducting the project. This is very difficult since there are more factors involved and this is typically a column that is prioritized when using the matrix. The choice here depends mainly on the initial costs of the tool and the development time with the tools that was experienced.

**Interfaces** Whether the design involves many interfaces with other components is used because ASD offers the design of interfaces that are easy to connect with other components because it is very strict. The other tools allow a wider interpretation and this can cause extra work when connecting the components.

**Design** There can be need for a more formal design, then ASD is the tool of choice. In Rhapsody and Bouml the design is made in UML diagrams and this is easier to model and easier to understand.

**Implementation** Here the number of deployment steps is the criterion. Which is the number of steps we have to take from generated code to executable. With much steps involved the less dependencies that Bouml needs are a asset. When the framework of Rhapsody is integrated in the programming environment the use of Rhapsody is recommended.

**Testing** When there are legacy tests available for validation (in case of redesign of a system) Rhapsody and Bouml are recommended since they have no verification built-in like ASD. Because of the verification in ASD it is likely that the validation tests are accepted quicker than with the other tools. The distinction in quality is made on the fact that verification is done on the models of ASD and it is possible to simulate the system in Rhapsody to see how it interacts.

**Maintenance** When changes are needed in the model this is most easy in Rhapsody. More detailed changes are easier in ASD since this requires only changing the code. The code of for example the hardware that is used is in Bouml and Rhapsody integrated in the model and needs to be changed within the tool to keep the model conform. In

ASD the code for the hardware is manually added to the generated code and needs only be changed there.

**Language experience** This is the experience of the developers that use the tool with the programming language they are developing in. In case of Rhapsody low experience is required since almost everything can be done with the tool. In Bouml there are much more language specific options that have to be filled in manually and in ASD there are sometimes changes in the generated code necessary.

**Learning curve** Using ASD without knowledge of the tool is very hard. Bouml requires more knowledge of programming, but it is quite easy to model and generate code from the model. Rhapsody is most easy to model in and generate code from compared to the others.

We now created the Modeling Tool Selection Method according to our mental model. The next Section introduces results from other researches to check the MTSM and in the next Chapter a case study will be conducted to cross check the MTSM with a practical application of the tools.

## 3.6 Validation

This Section offers results from researches of other companies to show similarities with the MTSM. Also the MTSM is reviewed by Software Architects and Software Designers at Logica. In the presentation of [17] that discusses the use of ASD during a period of two years the main conclusions are:

+ Used because of the guaranteed correct interfaces
+ Smooth integration
+ Less effort in testing and problem solving
+ Using ASD enforces the overthinking of every possible scenario

- Design and implementation require greater effort than traditional design
- Tools are not mature (gets better every release)
- High conceptual knowledge required of designer
- High ASD knowledge required
- Adding one function requires lots of model files to be changed

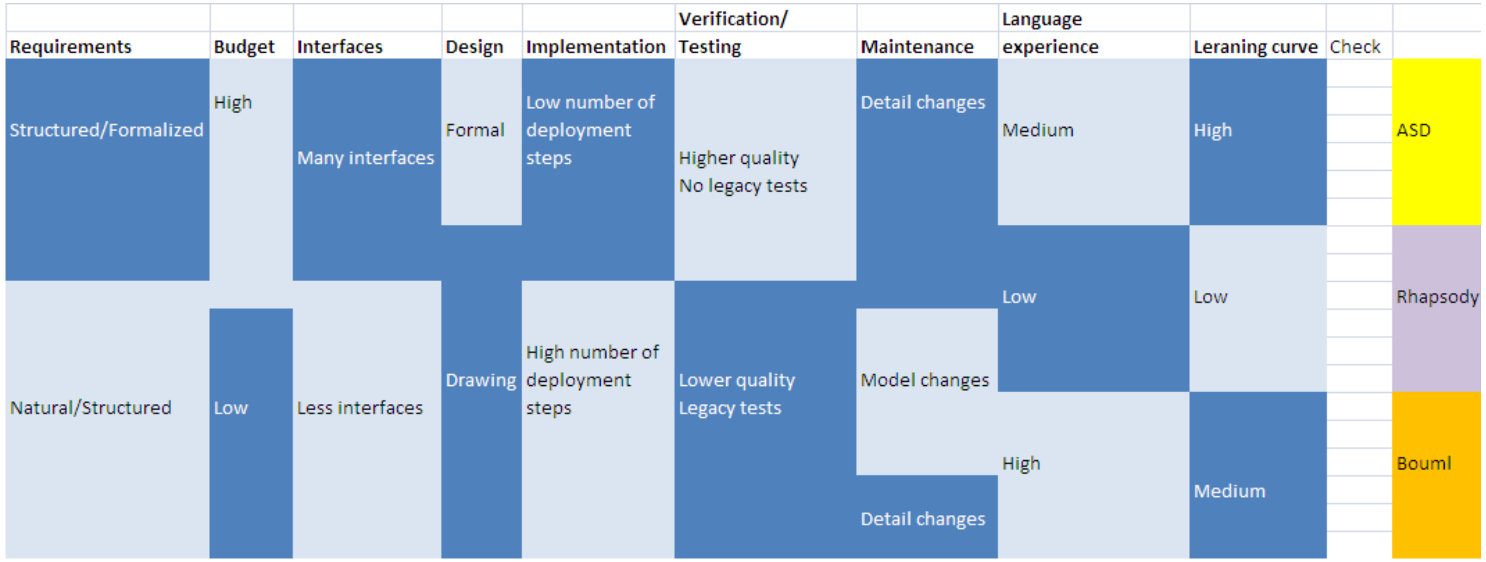
These findings agree with the MTSM for the ASD part. In the search for a Free/Libre Open Source Software (FLOSS) tool for Analysis and Design the authors of [1] consider Bouml, and they conclude that Bouml is good on functionality, usability and maintainability among seven other open source tools. Again, these [1] results agree with the findings before, although they used version 2.30.2 (July 2007), and this study uses version 4.19.1 (March 2010) of the software. At Saab Aerosystems they selected Rhapsody among four others as a UML/SysML-tool because it has a wide market penetration and is standards-based [3].

Their decision was also based on best practices and benchmarks of other state-of-the-art software engineering support tools and methods.

The previously mentioned researches from other businesses agree with the MTSM and show therefore that the MTSM displays a mental model that is shared by other researchers. The next Chapter describes the conducted case study and this will validate the MTSM by reflecting the results from the case study.

## 3.7 Summary

This Chapter introduced the Modeling Tool Selection Method, a method that decides which MDD tool fits a software engineering project best. First, the Meta-Process Model is introduced that describes where the MTSM is situated and how it can be used. The Meta-Process Model also describes how the MTSM can adopt knowledge from future case studies. Then a description of the method is given and the important business processes and products are discussed. We describe what is important and we reflect this on the three MDD tools. So the method is introduced and applied on the tools immediately. Then there is a Section about the tools, which gives a better view of the tools on a higher level. To allow accessible use of the method there is constructed a matrix that quickly decides on the use of a MDD tool, a description of the columns of the matrix and a usage description is given. Finally, the method is validated by using other researches and it was reviewed by Software Architects and Software Designers of Logica.



| Requirements | Budget | Interfaces | Design | Implementation | Verification/ Testing | Maintenance | Language experience | Leraning curve | Check | |
|---|---|---|---|---|---|---|---|---|---|---|
| Structured/Formalized | High | Many interfaces | Formal | Low number of deployment steps | Higher quality No legacy tests | Detail changes | Medium | High | | ASD |
| | | | | | | | Low | Low | | Rhapsody |
| Natural/Structured | Low | Less interfaces | Drawing | High number of deployment steps | Lower quality Legacy tests | Model changes | High | Medium | | Bouml |
| | | | | | | Detail changes | | | | |

Figure 3.5: Decision matrix of the MTSM

# Chapter 4

# Case Study

This Chapter describes the case study that is conducted during this thesis project. This case study describes a software engineering project that was performed at Logica. The objective of this case study is the evaluation of the Modeling Tool Selection Method from Chapter 3. The case study uses a project that was performed at Logica. During this project a prototype of a charge point for electric vehicles was developed. A basic implementation of this software is also constructed with the three tools, using a Model-Driven Development approach, and the findings are evaluated.

The DESMET methodology described in [27] is used to determine which evaluation methodology can be used to evaluate a method or tool within a particular organization [28]. We used the DESMET methodology to determine which evaluation method can be used to evaluate the tools. We evaluate the tools because we construct the mental model from this evaluation, and the MTSM from the mental model, see Figure 3.2.

First, the charge point project is described, then the hardware is introduced and the Section about software describes the software that runs on the board and how the programming code is made ready for the board. After this context of the case study, the design of the case study is described. The design is followed by the results, the implementation of the software and the relation with the Modeling Tool Selection Method from Chapter 3. The results are reflected on the MTSM and finally the considerations that formed the basis of the MTSM are validated.

## 4.1 Introduction

The goal of the charge point project at Logica was creating a prototype of a charge point for electric vehicles. During this project the question arose whether the use of MDD tools would have been useful, and this thesis project is the result of that question. During the original assignment Logica developed a prototype. The hardware for the charge point was delivered by the principal and the task of Logica was the interconnection of the hardware, by means of software development. The developed software was written in the C programming language. Since the assignment was the development of a prototype, the requirements were given in common natural language and were incomplete. The development methodology used was evolutionary prototyping [48] to keep the costs low and add additional requirements while developing. Eventually this project lead to a working charge point. During this project the requirements and design became clear, and we are using these findings during the case study.

The charge point project is also used during this case study to investigate the advantages and disadvantages of ASD, Bouml and Rhapsody. Difference with the first project is that we are no longer developing a prototype and therefore the charge point software is developed with the three tools using a iterative development methodology [48]. We do not use evolutionary prototyping since the requirements are clearer than during the development of the prototype. Also this allows us to develop the components of the system separately which enables us to monitor small pieces of the system.

## 4.2 Hardware

The hardware that is used for the charge point:

- Embedded Linux board (FOX Board LX832 [47])
- NFC Reader (MiFare Easy 5513 [22])
- Ledring
- GPRS Module
- Door lock
- Resistance meter
- Power meter

The diagram in Figure 4.1 gives a graphical representation of the hardware used and how it is connected.

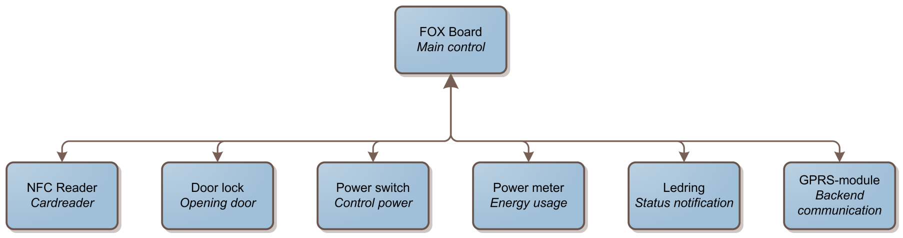


Figure 4.1: Overview of the hardware

## 4.3 Software

The operating system that runs on the board is Linux with kernel 2.6.19 and uses an Axis ETRAX 100LX CRIS CPU. For this CPU an Axis SDK for Linux is available that can be used to cross-compile the source code for the FOX Board. A graphical representation of how the SDK is used to create an image from the source code with the cross-compiler can be seen in Figure 4.2. How the SDK looks like is shown in Figure 4.3.

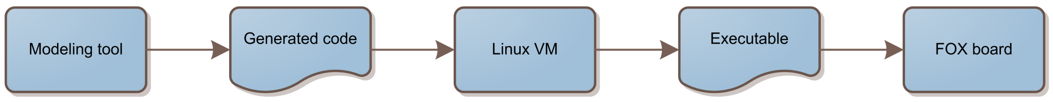


Figure 4.2: Sequence from tool to FOX board

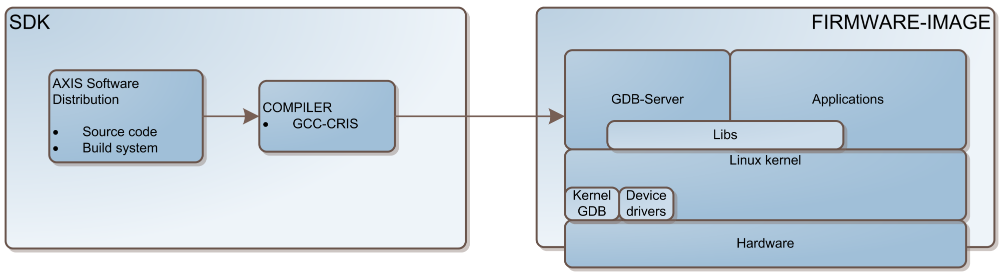


Figure 4.3: Software Development Kit block diagram from [11]

## 4.4 Design

The previous Sections described the context of the case study, this Section first describes why we use a case study and than it describes the goals for this case study.

As mentioned before, we used the DESMET methodology [27] to select a evaluation method. The DESMET methodology identifies nine methods of evaluation and a set of conditions that favor each evaluation method. We want to investigate the appropriateness of the method/tools, and we want to measure effects of using the method/tools. The method/tools we mention here, is the MDD development methodology and the tools are ASD, Bouml, and Rhapsody. The DESMET methodology concludes that a case study can be used to evaluate our method/tools, and it makes a separation between qualitative and quantitative case studies. This separation is not exactly applicable to our research and a combination is also possible, we therefore speak, of a case study.

The **goal** of this case study is to investigate the use of the tools and validate the MTSM by reflecting the results of the case study. We do this in almost the same way as described in Section 3.1 on how to change the MTSM.

1. Select multiple tools
2. Use multiple tools in the same case study
3. Evaluate the results of the case study

4. Construct mental model according to the findings during evaluation
5. Check MTSM to cope with the mental model

During this case study we aim to answer the following questions:

- What are the important business processes and products during this case study?
- Are the tools compatible with the used development process?
- To which extent can the MTSM be used as intended in Figure 3.2?
- Which tool fits this project best according to the results? And which tool fits this project best according to the MTSM?

The case study will be divided in the activities that were introduced in Section 3.3 so the results can be compared with the MTSM. The experiences gained with each tool will be discussed and the main focus will be on **quality** and **costs**. Quality of the software and costs of developing the software are main issues for Logica and other commercial businesses as well. Logica wants to deliver high quality software for a reasonable price to its customers.

The charge point application will be designed and implemented with ASD, Bouml and Rhapsody and tested on a real charge point that is available at Logica for testing purposes. The use of ASD is investigated in a pilot project at Logica which we were members of, and monitored closely. This allowed us to use the results during this case study.

## 4.5 Results

During this case study the software is modeled using the tools in an iterative development process. Because of this iterative process, the results can be divided in a implementation that only involves the board and the ledring and displays a sequence of three colors and a implementation of an complete working charge point application. The ledring application is constructed because the model is relatively easy and this way the cross-compilation and execution on the board can be tested with the different tools as source code generators. The results discussed in this Section will be linked to the activities used by the MTSM of Chapter 3.

### 4.5.1 The Ledring application

The ledring application as described here is only used to investigate the deployment of the models on the board and to measure code and executable size. The models of the ledring are not representative for the models of the ledring in the next Section of the complete charge point implementation.

**Requirements** For the Ledring application the following requirement in common natural language was constructed:

- Generate a green-blue-red sequence that shows every color 2 seconds.

**Analysis** Since the prototype was written in C, the choice for programming language C++ was made according to Table 3.2 and because C++ is an object-oriented language which fits this project, with multiple components, best. Fragments of the C code can be reused in C++ since most C code can be compiled in C++ without rewriting it. Such a fragment is for example the initialization of the card reader in the next Section.

**Design** Since the requirements are very short and not complicated the application was easy to design in all three tools. In Figure 4.4 we see a model of the ledring. The design in Figure 4.4 is extracted from Rhapsody. The state diagram in Bouml is displayed in Figure 4.5 and we see here that the state transitions are triggered by events. In Bouml the programming code that delays the programming (for 2 seconds) is in the states. The state diagram of ASD is displayed in Figure 4.6 and we see that there are responses, the ILedRing.NullRet is such a response. These responses are fired to the process that initiates the transition. This way the initiator knows whether a transition request is received (a synchronous return). In ASD this is used for verification of the model. In Bouml and Rhapsody the programming code to alter the coloring of the ledring is in the model. In ASD it is not possible to add foreign code directly in the model and this is added, during implementation, in the generated code.

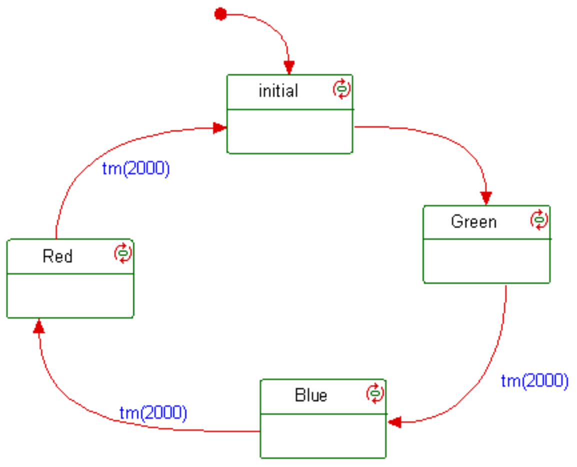


Figure 4.4: State Diagram of the Ledring from Rhapsody

**Implementation** With all three tools there is automatically generated programming code. The generated code for ASD needs to be manually edited to actually turn the leds in the right color. Bouml and Rhapsody allow the code to be added in the editor and generate code that is complete. The code is then for all three tools compiled and linked with the cross-compiler for the FOX Board. The executable size is determined using different options/configurations. The code is statically linked and/or the unneeded code was stripped by using a special command that removes the debugging information. The resulting sizes of the executables are displayed in Table 4.1. In the Table we see that the executable generated with Bouml is the smallest, while showing the exact same behavior to the outside world. Static and stripped is the option that is used during this case study and the size of the executable of ASD and Rhapsody is almost similar using that option.

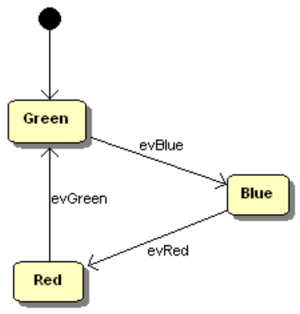


Figure 4.5: State Diagram of the Ledring from Bouml

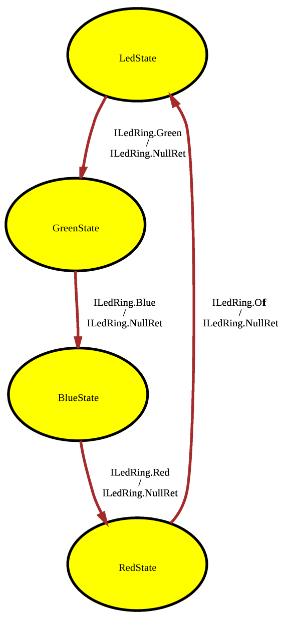

LedState
ILedRing.Green / ILedRing.NullRet
GreenState
ILedRing.Blue / ILedRing.NullRet
ILedRing.Of / ILedRing.NullRet
BlueState
ILedRing.Red / ILedRing.NullRet
RedState

Figure 4.6: State Diagram of the Ledring from ASD

Table 4.1: Different sizes of executables (in bytes)

| Tool | No options | Static | Stripped | Static and Stripped |
|---|---|---|---|---|
| ASD | 215460 | 7311084 | 113048 | 775036 |
| Bouml | 144306 | 5710907 | 7040 | 558820 |
| Rhapsody | 1841285 | 8808747 | 108280 | 753652 |

**Testing** All implementations showed the same behavior when ran on the FOX board. Rhapsody offers a simulation that shows the state sequences, this way we can check the behavior of the model before it is compiled and deployed on the board. ASD offers the possibility to check the model for deadlocks and live-locks before generating code. Bouml can output debugging information that shows the flow through the model while running on the board.

**Maintenance** Changing the sequence of the colors is easy with all three tools, but most easy is ASD in case of this Ledring application, because in ASD only the generated code needs to be changed while in the others the model has to be changed. When the model needs to be changed in ASD this is more work than in Bouml and Rhapsody because than we have to check the model again, and manually add the foreign code in the generated code. Bouml and Rhapsody generate the code according to the changed model and no manual editing is necessary.

### 4.5.2 Complete Chargepoint

The complete charge point involves the interconnection of all the hardware from Figure 4.1.

**Requirements** The requirements for the charge point project are given in common natural language, e.g. when a user wants to charge its car it stops nearby a charge point and swipes his NFC-card along the card reader and when he is authenticated the door opens and the ledring is green (red when authentication fails). He connects the plug of the power cord and closes the door, the charge point starts charging and the ledring becomes blue. When the user swipes his card again, the charging stops and the door opens. The user disconnects his power cord and closes the door again.

The quality of this requirement is poor since it is given in common natural language and can be interpreted in different ways and is not closed [12]. For instance the behavior of the ledring is partly described and the case where the door is supposed to open, but it does not open, is not described.

**Analysis** For this case study, as in the previous Section mentioned the C++ programming language is used. The programming environment for the tools is Microsoft Windows XP. The generated C++ source files are copied to a Virtual Machine that runs Linux and compiles the sources for the FOX board. From the VM the files will be copied to the FOX board.

**Design** The design in ASD takes a bottom-up approach by designing the components first. The design in Rhapsody and Bouml first describes the class view and then adds more detail by using state diagrams to design the behavior in the components. In Table 4.2 we see the time it took for developing the parts by a novice developer with all three tools. The time displayed by charge point is the time needed for connecting the parts together. In Figure 4.7 we see the state diagram from the charge point modeled in Rhapsody.

Table 4.2: Modeling time (in hours)

| Tool/Component | ASD | Bouml | Rhapsody |
|---|---|---|---|
| Chargepoint | 8 | 24 | 16 |
| Cardreader | 24 | 20 | 16 |
| Ledring | 30 | 16 | 10 |
| Door | 20 | 10 | 10 |

The state diagram from Figure 4.7 shows how the system works on the highest level. Init initializes the card reader and waits for the card reader to generate an evNFCDetected, in the state nfc_detected the NFC sequence that was read is authorized. When the event evAuthorized is fired the system will transfer to the authorized state in which the door is opened, it then waits for the door to close before it transfers to the charging state. Then the system is again waiting for a evNFCDetected and when a NFC sequence is read, it checks whether the user is the user that is currently charging. If it is indeed the same user, the door is opened again and the system waits for the door being closed, before it transfers to the init state again. We also see in the state diagram that there are states not_authorized, door_error and sign_off_refused. As the names do suspect, the system only transfers to these states when: the user is not authorized to use the charge point, the door did not open or the user that tries to end the charging process is not the same as the one who initiated the charging process.

The chargepoint is easy to realize in ASD since the parts that are connected (cardreader, ledring, door) were carefully designed before. The other components were partly reused so the modeling time is not from scratch, but the starting point was the same for all three tools. In Bouml and Rhapsody the components are designed faster since the flow is not verified like in ASD and need less rework/rethinking of the design. With Bouml a certain level of knowledge about C++ is necessary because connecting the parts requires knowledge about pointers and inheritance.

**Implementation** The automatic generation of code is the best and easy to use in Rhapsody since it is possible to compile and execute in the tool. The overhead of copying to the Virtual Machine (VM) and compiling for the board and copying to the board, before running on the board is then not necessary and that saves time. Also with ASD there were troubles when compiling in the VM. Some parts of the generated code needed to be edited before it successfully compiled. The framework offered by Rhapsody (described in Section 3.3 allows that the process of generating code and running it on the board goes very easy compared to ASD and Bouml.

**Testing** In ASD every component is verified for correctness and completeness. In Rhapsody and Bouml this is not verified and deadlocks and live-locks can occur. In Rhapsody it is possible to simulate the system. Rhapsody allows the designer to step through the diagrams and see what happens. It is also possible to fire triggers and this way the designer can test what happens when for instance the door is opened when the system is not expecting it to open and so on. In Bouml there is no form of verification included and verification should be done separately.

**Maintenance** The way the cardreader reads a card was changed during the case study. At first the program just had to open the reader and read the buffer and check for a card-ID. During a later iteration the program had to initiate a read on the cardreader before opening it. This change was in the foreign code that was added to the models and therefore the models did not change. Such detailed changes are easy to process in ASD, because in ASD only the manually added code on the Linux VM from Figure 4.2 needs to be changed. In Bouml and Rhapsody the code that is in the models needs to be changed and this requires more effort.

Another change that was made during the case study was the possibility to check if the card is on a white-list that is locally stored on the board. This changes the models and was therefore more interesting than only a detailed change in the foreign code. Changing the model in ASD took more time then changing the model in Bouml and Rhapsody. In ASD the model that changes needs to be verified, and the component (chargepoint) that uses the model of the, in this case, cardreader also needs to be changed and verified again. So adding changes requires changes in related components, the verification of all these changes is time consuming. In this particular case there was another issue with ASD, since storing values from a foreign component in ASD, can not be done in the model (the tool does not support that), and needed to be solved by manually changing the generated code.

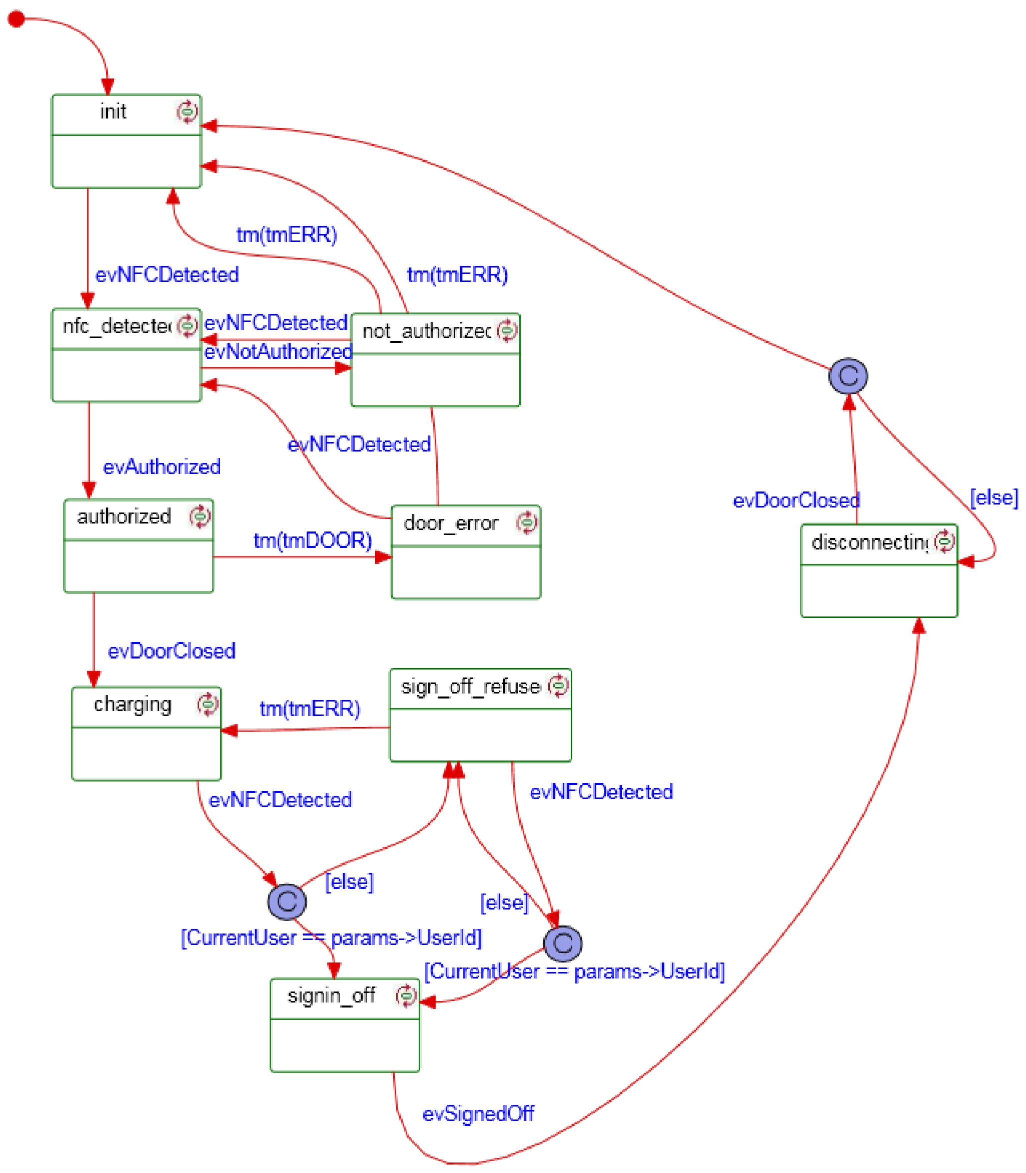


Figure 4.7: State diagram of the charge point from Rhapsody

## 4.6 Discussion of the Case Study Results

This Section reflects the findings from the case study with the MTSM. First we use the MTSM to select a tool for the charge point project that was used during the case study and then we check which tool was the best option for this project according to the results of the conducted case study. The result of this case study can be described as the experiences by actually using the tools in a project environment, the actual formation of the mental model. We use the matrix from Figure 3.5 to select a tool for the charge point project. The charge point project that was also used during the case study. The goal of this project is delivering software that interconnects the hardware of the charge point.

We use the matrix and discuss the project per column of the matrix:

**Requirements** The requirements were given in common natural language, so we put a check-mark by Bouml and Rhapsody.

**Budget** For this project the budget is low, because it is not clear whether the principals charge point will be commonly used in the future. Therefore we check Bouml and Rhapsody.

**Interfaces** There are not many interfaces and the software designed does not need to be fitted with other parts so we check Rhapsody and Bouml

**Design** No particular design is needed, we check all three tools.

**Implementation** The number of steps is considered high, see Figure 4.2, but we think we need to implement changes and need to compile multiple times, so we choose Rhapsody because of its framework.

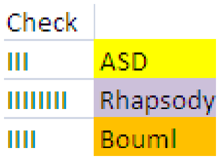
**Testing** There are no tests available so we choose ASD. Since we consider this important we check ASD twice.

**Maintenance** The charge point for electric vehicles is new software and hardware and the expectation is that the software evolves over time, so we choose Rhapsody.

**Language experience** The experience of the developers is low and therefore Rhapsody is checked.

**Learning curve** The developers have no experience with the tools so Rhapsody is checked.

We can conclude from the matrix that according to the MTSM Rhapsody is the tool that is recommended to be used, see Figure 4.8. One reason to choose another tool could be the programming language that needs to be used, whether the language is supported by the tool can be found in Table 3.2.



| Check | |
|---|---|
| III | ASD |
| IIIIIIII | Rhapsody |
| IIII | Bouml |

Figure 4.8: Results of using the matrix

According to the findings during the case study the recommendation of tool to use for this project would be Rhapsody. This agrees with the MTSM for this particular project. Below we describe some of the experiences of the performers of the case study:

- Rhapsody is easy to use and the deployment is fast

- Rhapsody simulation is easy to use and the models are clear
- The verification of ASD is the strongest point
- Use of ASD is not easy and the way of formal designing requires a special type of developer
- Bouml is fast and code generation is clean

The next Section describes the threats to validity of this case study.

## 4.7 Threats to Validity

This Section evaluates the applicability of using a case study to evaluate the MTSM and the quality of the conducted case study. The evaluation of the quality is done by using four tests from [53] that are used to validate case study research in social science. These tests are also applicable for validating case studies in the Software Engineering field as described in [27][34][52]. The four tests that will be discussed in the subsections of this Section are: Construct validity, Internal validity, External validity and Reliability. Goal of the case study is to investigate the use of the tools and validate the MTSM by reflecting the results of the case study.

During the design in Section 4.4 we used the DESMET methodology to select a evaluation method, an case study. The DESMET methodology is able to evaluate methods and tools. Often only the evaluation of methods and processes is described, as in [51], and therefore we used DESMET [27] [28] [35] to select an appropriate evaluation method. According to the DESMET Technical Report, using a case study, as we did, holds a relatively high risk because it could give a misleading result. We think we covered this by reflecting the results with Software Architects and Software Engineers at Logica and by using the results of ASD from the pilot project. This way the evaluation is not based on one person's experience of using the tool, as DESMET indicates to be a risk.

### 4.7.1 Construct Validity

This test is used to check for the right operational measures during the case study. For the MTSM in this case study this test shows that the used criteria are correct for selecting tools. The important criteria are divided into the same activities as used in the MTSM and therefore the case study could be monitored in small controllable parts. We also check whether we could answer the questions that we aimed to answer during this case study from the design in Section 4.4:

- What are the important business processes and products during this case study? *During the design we stated that quality and costs are most important. Therefore the activities that are most important, are the design and the implementation. The design took much effort, because not all requirements were clear, which also influences the quality. The implementation took much effort, because the generated code needed to be cross-compiled for the board.*

- Are the tools compatible with the used development process? *The tools can be used in an iterative development process.*
- To which extent can the MTSM be used as intended in Figure 3.2? *From the evaluation of the results of this case study a mental model was formed and this was reflected with the MTSM correctly. Also the selection of a tool according to the MTSM worked as intended.*
- Which tool fits this project best according to the results? And which tool fits this project best according to the MTSM? *For this project the MTSM concluded on the same tool as the gained knowledge during the case study concluded on this project. The case study was conducted in the same way that the actual project would be developed.*

### 4.7.2 Internal Validity

This test is used to determine causal relationships in the results of the case study. The relationship between the activities is very important since a longer modeling time could cause a shorter implementation time which was monitored when using ASD. Another threat for the internal validity is the following: the monitored time when constructing a model in the first tool could be higher than in the second and third tool because the idea of how to construct this model could be reused. To avoid this, one could select a group of equally experienced users which could then be divided in three and each model in one tool. During this case study this effect was minimal since the modeling in ASD is very different from Bouml and Rhapsody. The modeling in Bouml and Rhapsody differs in the sense that the way transitions work are different and therefore the model needs to be reinvented on the level of transitions and triggers.

### 4.7.3 External Validity

Tests whether the results of this case study are generalizable to other case studies. The results of this case study can be used by other companies or researchers for future research or for selecting a MDD tool. Although certain parts such as the initial selection of the tools is initiated by Logica this did not involve the results of the case study or the construction of the MTSM. When future case studies are performed the results can be used to supplement the constructed MTSM from Chapter 3.

### 4.7.4 Reliability

Objective of this test is to find out if a later investigator that follows the same procedures as described by an earlier investigator arrives at the same findings and conclusions. The results found during this case study were used to check the MTSM which was checked by software architects and developers from Logica. This cross-reference shows that the mental model found during this case study was correct for the construction of the MTSM. Reproduction of this case study would result in the same findings during the case study when using the same software versions of the tools.

## 4.8 Summary

This Chapter introduced the case study, a project that was performed at Logica and was used to evaluate the MDD tools with. First the charge point project was described and the hardware and software that this project used. Then the design of the case study was given and we showed, by using the DESMET methodology, that a case study gives the ability to evaluate the results. The results of the case study were discussed by using a small application for the evaluation of the code generation and implementation on the hardware, and a complete charge point application for the evaluation of the whole development process. During the discussion of the case study results we used the MTSM to select a MDD tool and compared this selection with the results of the conducted case study. Finally the threats to validity were discussed and we used four tests to evaluate the quality of the conducted case study. We showed that the case study was thoroughly validated by experts from Logica and future case studies can refine the MTSM.

# Chapter 5

# Summary, Conclusions and Future Work

This chapter gives an overview of the project's contributions. After this overview, we will reflect on the results and draw some conclusions. Finally, some ideas for future research will be discussed.

## 5.1 Summary

The goal of this thesis project was to help Logica adopt Model-Driven Development (MDD) in an efficient way. Logica is interested in two MDD tools (closed source) and we added a third tool, which is open source. During this thesis a unique Modeling Tool Selection Method (MTSM) was constructed that helps Logica to select a MDD tool at the start of a software engineering project. This MTSM was reviewed by Software Architects and Software Developers of Logica, and validated with other researches. We designed a case study to validate the MTSM, and to investigate the use of the tools. The project we used for our case study was already developed at Logica and was the development of software for a charge point for electric vehicles. During the case study the main functions of the software were developed by using all three MDD tools separately. For ASD there was a pilot project, for the charge point software, at Logica which we participated in and monitored closely for our case study. The software was also developed with Bouml and Rhapsody during the case study. The discussion of the results of the case study, in Section 4.6, showed that the experiences of the developers that developed the software during the case study cope with the MTSM.

## 5.2 Conclusions

The Modeling Tool Selection Method (MTSM) introduced in this thesis helps Logica selecting one of their favorite Model-Driven Development (MDD) tools. The MTSM uses the available information of an upcoming project to select a tool for that project. The MTSM asks questions about the available information and it classifies the tools on criteria as usability, understandability, maintainability and others. Eventually all this information is put into a matrix that helps in quickly choosing a tool. Using the MTSM saves valuable time for Logica since the selection of a tool is now guided by this method.

The MTSM can also be used when a tool is already selected for a project. It then gives the people involved in the project the important processes and products for that particular tool. Validation of the MTSM is done by comparing results from other companies and reviews of the method by Software Architects and Software Designers of Logica.

The case study involves a project that was programmed on a conventional way at Logica and is build with all MDD tools during the case study. The evaluation of this case study and the mental model that was formed during this extensive research are the root for the MTSM.

The **research questions** and their sub questions investigated in this thesis are:

1. To which extent can Model-Driven Development be used in the context of Logica?
    a) What changes are needed in the current development process?
    b) How can these tools help in delivering higher quality code for a low price?
    c) When to use which tool?
2. How can the selection method be improved with knowledge from future projects?
    a) To which extent can we add knowledge gained during projects?
    b) How to add tools to the method/matrix?

The conclusions on the research questions are formed throughout this thesis and here we describe how the conclusion is formed per research question. The conclusion on research question 1a is that the development process is not specific per tool or for a MDD approach. The current development process needs not to be changed or formed in a specific way, see Section 3.3. At least the development process used during the case study in Chapter 4, iterative development, can be used by all three MDD tools. A case study can be considered to investigate the difference between several development processes and the use with an MDD tool. The answer to research question 1b can be found in Section 2.1. The use of MDD offers a higher level of abstraction and the tools force the user to design a model. The designing of models during the case study resulted in questions that the former designers gave no thought. So we might conclude that the use of MDD resulted in higher quality of the design, but whether this was at a lower price is hard to conclude. The development time of the charge point application with the conventional way of programming could not be compared to the design and programming time of MDD, since the first programming time (conventional) was not measured, and parts of the programming code of the conventional approach were reused in the MDD approach. When to use one of the tools that were researched is asked in research question 1c and is described in the method of Chapter 3. The method is especially constructed to make this selection at the start of a project. We can now answer research question 1 that asks to which extent MDD can be used within Logica. We introduced a method that selects a MDD tool at the start of a project. We narrowed the use of this method, and thus MDD, to embedded software engineering projects and according to our results the MTSM selects the right tool for the project.

The second research question has two sub questions and are more specific for the use of the MTSM. The answer to research question 2a is that it is not recommended to add knowledge gained during projects. It is better to perform a case study since this results in a more thorough investigation and better evaluation and validation of the findings. The other sub question is 2b and is about the addition of a tool to the method. This is possible as described in Section 3.1 and displayed in Figure 3.2 where a description is given of the addition of a MDD tool. These sub questions answered the main research question 2 of how the MTSM can be improved with future knowledge, by adding results from case studies and by adding more MDD tools.

This research is useful for Logica because it resulted in the MTSM and a matrix that can be used to select a MDD tool. This helps Logica adopting MDD in a efficient way, since the best tool for the project is used and this increases the efficiency, the goal of this thesis project. The introduction of the MTSM is also useful for science since it is a expandable method that can also be used by other companies or for future research. The introduction of the Meta-Process model allows the addition of criteria in the future, and the ability to add other MDD tools. Useful extensions for future research are introduced in the next section.

## 5.3 Future Research

Since the MTSM is only evaluated with one case study useful future work is the evaluation with multiple cases as already illustrated in Figure 3.2. This probably makes the MTSM more accurate and gives a broader platform for acceptation of the MTSM. Also the use of other development processes can be researched within a case study. When conducting more case studies another useful extension would be a more detailed measurement of the costs. Whether the costs are made during design or during implementation is interesting since there can be significant differences in wages of Software Architects and Software Developers. Another interesting research would be to investigate to which extent it is possible to measure the quality of the generated code.

Another useful extension is the addition of more MDD tools that Logica might be interested in. This shows the ability to recover the mental model from this study and adds another tool to the selection method.

As already discussed in Chapter 3 the MTSM is now only capable of use with embedded systems that are not real-time. Future research could perhaps extend the use of the MTSM by researching the applicability of the selection method in other types of projects.

# Bibliography

[1] O. Alfonzo, K. Domínguez, L. Rivas, M. Pérez, L. Mendoza, and M. Ortega. Quality Measurement Model for Analysis and Design Tools based on FLOSS. In *Proceedings of the 19th Australian Conference on Software Engineering*, pages 258–268. IEEE Computer Society, 2008.

[2] D. Ameller. Considering non-functional requirements in model-driven engineering. Master's thesis, Universitat Politcnica de Catalunya, june 2009.

[3] H. Andersson, E. Herzog, G. Johansson, and O. Johansson. Experience from introducing unified modeling language/systems modeling language at Saab Aerosystems. *Systems Engineering*, 2009.

[4] E.G. Aydal, M. Utting, and J. Woodcock. A comparison of state-based modelling tools for model validation. *Objects, Components, Models and Patterns*, pages 278–296.

[5] P. Baker, S. Loh, and F. Weil. Model-driven engineering in a large industrial context-Motorola case study. *Lecture notes in computer science*, 3713:476, 2005.

[6] J. Bézivin. On the unification power of models. *Software and Systems Modeling*, 4(2):171–188, 2005.

[7] K. Bjerge. Model-driven development for embedded systems. 2008.

[8] Bouml. Bouml, Last accessed 02-02-2010. `http://bouml.free.fr/historic.html`.

[9] G.H. Broadfoot. Asd case notes: Costs and benefits of applying formal methods to industrial control software. *Lecture Notes in Computer Science*, 3582:548, 2005.

[10] G.H. Broadfoot and G. Kielty. Analytical Software Design Case MagLev Stage Software Project for Philips Applied Technologies, 2005.

[11] AXIS Communications. Source development kit (sdk), Last accessed 06-06-2010. `http://www.axis.com/products/dev_sdk`.

[12] J.C.S. do Prado Leite and J.H. Doorn. *Perspectives on Software Requirements*. Kluwer Academic, Boston, Mass., 2004.

[13] B. P. Douglass. Model driven architecture and rhapsody.

[14] H.E. Eriksson, M. Penker, B. Lyons, and D. Fado. *UML 2 toolkit*. Wiley, 2004.

[15] FDR2. Failures-divergence refinement, Last accessed 23-05-2010. `http://www.fsel.com`.

[16] B. Flyvbjerg. Five misunderstandings about case-study research. *Qualitative inquiry*, 12(2):219, 2006.

[17] Bert Folmer. Philips healthcare interventional x-ray project management, 11-05-2010. `Presentation Philips`.

[18] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *2007 Future of Software Engineering*, pages 37–54. IEEE Computer Society, 2007.

[19] R.B. France, S. Ghosh, T. Dinh-Trong, and A. Solberg. Model-driven development using uml 2.0: promises and pitfalls. *Computer*, pages 59–66, 2006.

[20] D. Gentner and A.L. Stevens. *Mental models*. Lawrence Erlbaum, 1983.

[21] E. Gery, D. Harel, and E. Palachi. Rhapsody: A complete life-cycle model-based development system. *Lecture Notes in Computer Science*, pages 1–10, 2002.

[22] ACG Identification Technologies GmbH. Hf mifare easy module, Last accessed 16-03-2010. `http://www.acg.de`.

[23] B. Hailpern and P. Tarr. Model-driven development: The good, the bad, and the ugly. *IBM Systems Journal*, 45(3):451–461, 2006.

[24] C.A.R. Hoare. Communicating sequential processes. *Communications of the ACM*, 1985.

[25] IBM. Ibm, Last accessed 10-06-2010. `http://www.ibm.com`.

[26] H.W. Jung, S.G. Kim, and C.S. Chung. Measuring software product quality: A survey of ISO/IEC 9126. *IEEE software*, pages 88–92, 2004.

[27] B. Kitchenham, S. Linkman, and D. Law. DESMET: a methodology for evaluating software engineering methods and tools. *Computing and Control Engineering Journal*, 8(3):120–6, 1997.

[28] Barbara Ann Kitchenham. Evaluating software engineering methods and tool part 1: The evaluation context and evaluation methods. *SIGSOFT Softw. Eng. Notes*, 21(1):11–14, 1996.

[29] X. Larrucea, A.B.G. Díez, and J.X. Mansell. Practical Model Driven Development process. In *Second European Workshop on Model Driven Architecture (MDA) with an emphasis on Methodologies and Transformations*. Citeseer, 2004.

[30] L.E. MENDOZA, M.A. PÉREZ, T. ROJAS, A. GRIMÁN, and L.A. DE LUCA. Selecting tools for software quality management. *Software Quality Professional*, 4(4):18–27, 2002.

[31] OMG. Mda guide version 1.0.1. 2003. OMG document number omg/2003-06-01, available from www.omg.org.

[32] OMG. Object management group, Last accessed 23-05-2010. `http://www.omg.org/`.

[33] OMG Architecture Board ORMSC. Model driven architecture. 2001. OMG document number ormsc/2001-07-01, available from www.omg.org.

[34] D.E. Perry, A.A. Porter, and L.G. Votta. Empirical studies of software engineering: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 345–355. ACM, 2000.

[35] Shari Lawrence Pfleeger. Design and analysis in software engineering: the language of case studies and formal experiments. *SIGSOFT Softw. Eng. Notes*, 19(4):16–20, 1994.

[36] Philips. Philips applied technologies, Last accessed 07-06-2010. `http://www.apptech.philips.com`.

[37] S.J. Prowell and J.H. Poore. Foundations of sequence-based software specification. *IEEE Transactions on Software Engineering*, pages 417–429, 2003.

[38] J. Rech and C. Bunse. *Model-Driven Software Development: Integrating Quality Assurance*. Information Science Publishing, 2008.

[39] C. Rolland. Modeling the requirements engineering process. In *3rd European-Japanese Seminar on Information Modeling and Knowledge Bases, Budapest, Hungary*. Citeseer, 1993.

[40] A.W. Roscoe, C.A.R. Hoare, and R. Bird. *The theory and practice of concurrency*. Citeseer, 1998.

[41] J. Rothenberg. The nature of modeling. *Artificial Intelligence, Simulation, and Modeling*, pages 75–92, 1989.

[42] I. Rus and M. Lindvall. Knowledge management in software engineering. *IEEE software*, pages 26–38, 2002.

[43] D.C. Schmidt. Guest editor's introduction: Model-driven engineering. *IEEE Computer*, 39(2):25–31, 2006.

[44] B. Selic. The pragmatics of model-driven development. *IEEE software*, 20(5):19–25, 2003.

[45] Sioux. Sioux, Last accessed 07-06-2010. `http://www.sioux.eu`.

[46] D.I.K. Sjoberg, T. Dyba, and M. Jorgensen. The future of empirical methods in software engineering research. In *International Conference on Software Engineering*, pages 358–378. IEEE Computer Society Washington, DC, USA, 2007.

[47] ACME systems. Fox board, Last accessed 11-05-2010. `http://foxlx.acmesystems.it`.

[48] A.B. Tucker. *The computer science and engineering handbook*. CRC press New York, 1997.

[49] Verum. Tools for building mathematically verified software, Last accessed 10-05-2010. `http://www.verum.com`.

[50] J.M. Wing. A specifier's introduction to formal methods. *Computer*, 23(9):8–10, 1990.

[51] C. Wohlin, M. Höst, P. Runeson, M.C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering: an introduction*. Kluwer Academic Pub, 2000.

[52] C. Wohlin, M. Höst, P. Runeson, M.C. Ohlsson, B. Regnell, and A. Wesslén. *Experimentation in software engineering: an introduction*. Kluwer Academic Pub, 2000.

[53] R. Yin. *Case study research: Design and methods*. Sage Publications, 2008.

# Appendix A

# Glossary

In this appendix we give an overview of frequently used terms and abbreviations.

**ASD:** Analytical Software Design

**CASE:** Computer Aided Software Engineering

**ELOC:** Executable Lines of Code

**FDR:** Failure-Divergence Refinement

**MDA:** Model-Driven Architecture

**MDD:** Model-Driven Development

**MDE:** Model-Driven Engineering

**MTSM:** Modeling Tool Selection Method

**OMG:** Object Management Group

**PDA:** Platform-Dependent Application

**PIM:** Platform-Independent Model

**PSM:** Platform-Specific Model

**SBS:** Sequence-Based Specification

**UML:** Unified Modeling Language

**VM:** Virtual Machine