MODELING PATTERNS AND CULTURES OF EMBEDDED SOFTWARE DEVELOPMENT PROJECTS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS OF
THE MIDDLE EAST TECHNICAL UNIVERSITY
BY

DENİZ AKDUR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

FEBRUARY 2018

MODELING PATTERNS AND CULTURES OF EMBEDDED SOFTWARE DEVELOPMENT PROJECTS

Submitted by **DENİZ AKDUR** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy in the Department of Information Systems, Middle East Technical University** by,

Prof. Dr. Deniz Zeyrek Bozşahin
Dean, **Graduate School of Informatics** _____

Prof. Dr. Yasemin Yardımcı Çetin
Head of Department, **Information Systems** _____

Prof. Dr. Onur Demirörs
Supervisor, **Information Systems, METU & IZTECH** _____

Examining Committee Members:

Assoc. Prof. Dr. Altan Koçyiğit
Information Systems, METU _____

Prof. Dr. Onur Demirörs
Information Systems, METU & IZTECH _____

Assoc. Prof. Dr. Erhan Eren
Information Systems, METU _____

Asst. Prof. Dr. Sadık Eşmelioğlu
Computer Engineering, Çankaya University _____

Asst. Prof. Dr. Ömer Özgür Tanrıöver
Computer Engineering, Ankara University _____

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this wok.

Name, Last name : Deniz AKDUR

Signature : _____

ABSTRACT

MODELING PATTERNS AND CULTURES OF EMBEDDED SOFTWARE DEVELOPMENT PROJECTS

Akdur, Deniz

Ph.D., Department of Information Systems

Supervisor: Prof. Dr. Onur Demirörs

February 2018, 126 pages

Due to their multiple constraints across different dimensions of performance and quality, the analysis, design, implementation and testing of software-intensive embedded systems are not trivial, which makes their development more challenging. To cope with these growing complexities, modeling is a widely used approach in this industry. However, the modeling approaches in embedded software vary since the characteristics of diagram development and usage (e.g., purpose, modeling rigor, medium type used, modeling stakeholder profile, target sector, etc.) differ among systems as well as among sectors. At one extreme, some stakeholders use software modeling informally, where they sketch the diagrams on a paper in order to communicate with other stakeholders. At the other extreme, modeling turns into programming with automated generation of some software development life cycle (SDLC) artifacts (i.e., code, documentation or test driver). Moreover, different stakeholders in the same software development project can use diagrams for different purposes within different SDLC phases. This PhD dissertation identifies and defines the modeling patterns and cultures of embedded software development projects. To achieve this, it firstly figures out the current state-of-practice of modeling to investigate the relations between the characteristics of diagram development and usage and also the significant parameters to identify modeling patterns. After identifying the modeling patterns and cultures, this study proposes a characterization model. This model not only identifies and defines modeling patterns and cultures of the modeling stakeholder in embedded software development projects, but also gives recommendations for commonsense modeling practices. Finally, this proposed model is validated by multiple case studies.

Keywords: Software Modeling, Embedded Software, Model Driven Engineering (MDE), Modeling Patterns and Cultures, Characteristics of a Diagram

ÖZ

GÖMÜLÜ YAZILIM GELİŞTİRME PROJELERİNDE GÖZLEMLENEN MODELLEME YAKLAŞIMI KALIP VE KÜLTÜRLERİ

Akdur, Deniz

Doktora, Bilişim Sistemleri Bölümü

Tez Yöneticisi: Prof. Dr. Onur Demirörs

Şubat 2018, 126 sayfa

Tasarım, geliştirme ve sınanması diğer yazılım sistemlerine göre daha karmaşık olan yazılım-yoğun gömülü sistemlerde, artan karmaşıklıkla başa çıkabilmek için kullanılan en etkin yöntemlerden biri yazılım modellemesidir. Ancak, gömülü yazılım endüstrisinde kullanılan diyagramların geliştirilmesi ve kullanımı sırasındaki öz niteliklerinin (örneğin, amaç, modelleme katılığı, kullanılan medya, modelleme paydaşlarının profilleri, hedef sektör, vb.) farklılaşması, modelleme yaklaşımlarının da hem sektörler hem de sistemler arasında değişiklik göstermesine neden olmaktadır. Uç bir örnek olarak, bir modelleme paydaşı kâğıt üstünde kabataslak diyagram çizip sadece fikir alışverişi yapmak isteyebilir. Diğer uç bir örnekte ise, yazılım modellemesi programlama diline dönüştüğünden yazılım geliştirme yaşam döngüsü (YGYD) çıktılarını (örneğin, kod, doküman, test simülatörü gibi) bu modeller aracılığıyla oluşturabilir. Dahası, aynı şirketteki farklı bölümlerki paydaşlar bile yazılım modelleme yaklaşımlarını farklı amaç ve YGYD evrelerinde kullanabilirler. Gömülü yazılım geliştirme projelerinde gözlemlenen modelleme yaklaşımı kalıpları ve kültürlerini belirleyen bu doktora savunması, öncelikle endüstrideki en son modelleme kullanımlarını ortaya çıkartarak modelleme sırasında kullanılan diyagramların öz niteliklerini ve birbirleriyle olan ilişkilerini karakterize etmiştir. Elde ettiği bu bilgiler ışığında, gömülü yazılım geliştirme projelerinde gözlemlenen modelleme yaklaşımı kalıp ve kültürlerini ortaya çıkaran ve tanımlayan bu çalışma, sonrasında bir model önermiştir. Bu model, gömülü yazılım geliştirme projelerinde kullanılan modelleme yaklaşım kalıpları ve kültürlerini ortaya çıkarmakla kalmamış, modelleme paydaşına etkin bir modelleme yaklaşımı için öneriler de vermiştir. Son olarak, önerilen model yapılan çoklu vaka çalışmaları ile doğrulanmıştır.

Anahtar Sözcükler: Yazılım Modellemesi, Gömülü Yazılım, Gömülü Sistem, Model Güdümlü Mühendislik (MGM), Modelleme Kalıpları ve Kültürleri, Diyagram Öz Nitelikleri

to my beloved son, Ege

ACKNOWLEDGMENTS

First of all, I would like to express my sincere gratitude to my supervisor Prof. Dr. Onur Demirörs for his continuous guidance, insightful suggestions and comments throughout my study. I also deeply appreciate his never ending patience and encouragements.

I would like to thank the rest of my thesis committee: Assoc. Prof. Dr. Altan Koçyiğit, Asst. Prof. Dr. Sadık Eşmelioğlu, Assoc. Prof. Dr. Erhan Eren and Asst. Prof. Dr. Ömer Özgür Tanrıöver for their valuable advices.

I would also like to thank Asst. Prof. Dr. Bilge Say, who gave me great suggestions and feedbacks while finalizing and writing of this thesis. I am also grateful to Assoc. Prof. Dr. Vahid Garousi, who contributed to related research during this PhD study.

The case studies reported in this dissertation would not have been possible without the collaboration of my colleagues. I would like to thank all software professionals and companies, who contributed to this study.

Last but not the least, I would like to thank my family for supporting me spiritually throughout my life. Special thanks go to my parent: my father Mahmut and my mother Kadriye; to my wife Tuğba and also my brother Barış for their encouragement and support throughout this academic journey. I am really happy and fortunate to be a part of this wonderful family.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AR	Action Research
AUTOSAR	AUTomotive Open System ARchitecture
BSP	Board Support Package
CMM	Capability Maturity Model
DSL	Domain Specific Language
DSML	Domain Specific Modeling Language
DSP	Digital Signal Processing
EBSE	Evidence Based Software Engineering
EMF	Eclipse Modeling Framework
GQM	Goal, Question, Metrics
ICD	Interface Control Document
IT	Information Technology
MAPforES	Modeling Approach Patterns for Embedded Software
MBE	Model Based Engineering
MBT	Model Based Testing
MDA	Model Driven Architecture
MDD	Model Driven Development
MDE	Model Driven Engineering
MML	Modeling Maturity Level
M2M	Model to Model
M2T	Model to Text
PIM	Platform Independent Model
R&D	Research and Development
RQ	Research Question
SCI	Software Configuration Item
SDLC	Software Development Life Cycle
SE	Software Engineering
SLR	Systematic Literature Review
SM	Systematic Mapping
T2T	Text to Text
UML	Unified Modeling Language

CHAPTER 1

INTRODUCTION

Software-intensive embedded systems shape our world by becoming an essential aspect of our lives [1]. They can be found in many devices such as cars [2], TVs [3], smart phones [4] and defense systems [5]. As consumers acquire more such devices, the volume of embedded software on different domains is increasing at 10% to 20% per year. Moreover, embedded microprocessors account for more than 98% of all produced microprocessors, thus vastly surpassing computing power in the IT industry [1]. The growth rate in software–intensive embedded systems is more than 14% per annum and it is forecasted there will be over 40 billion devices (5–10 embedded devices per person) worldwide by 2020 [6].

Design, implementation and testing of software for modern embedded systems are not trivial [7, 8] due to their multiple constraints across different dimensions of performance and quality [9, 10]. Moreover, the increasing amount of components in these systems and having distinct functionalities incorporated into a single system, which require seamless integration of many hardware and software systems, make the embedded software development more challenging [11]. Software modeling plays a crucial role in the embedded software industry by becoming a tool to manage complexity of these systems. However, there is a large variety of modeling practices used in this domain. Therefore, it is important to empirically analyze and investigate all different approaches by understanding its state-of-the-practice while identifying the relations between the characteristics of modeling (e.g., modeling rigor[1], purpose, code correspondence, stakeholder, medium type used, SDLC phase, benefits, challenges, etc.) in order to help different modeling stakeholders to increase their awareness of these modeling practices.

The rest of this chapter represents the context and problem, research goal and the approaches used as a research strategy, the contribution and significance of this study, and finally, the structure of the overall thesis.

1.1 Context and Problem

Software modeling helps engineers to work at higher levels of abstraction [12] and facilitates communication [13]. However, the modeling approaches in embedded software vary since the characteristics of modeling differ among systems as well as among sectors, e.g., consumer electronics, defense or automotive. At one extreme, some stakeholders (e.g., some project managers or systems engineers) use software modeling informally, where diagrams are sketched on a paper or on a white board in order to communicate with colleagues. In such

[1] Modeling rigor is the formality of modeling language (e.g., informal or formalized), which affects software modeling usage in varying degrees.

cases, the emphasis is on communication rather than comprehensive formal specifications and these diagrams might be either soon discarded or quickly become inaccurate since they are not kept updated along with the source code [14]. At the other extreme, for some stakeholders (e.g., some software developers), modeling turns into programming with automated generation of code from models and the corresponding diagrams have more lifespan and archivability. Furthermore, different units within the same company might use different modeling approaches for different purposes in different phases of software development life cycle (SDLC) [15].

To better analyze these different approaches, it is necessary to understand the terminology used in the context of software modeling. According to Brambilla, Cabot and Wimmer, model-driven development (MDD) treats models as the primary artifact of the development process [16]. Usually, in MDD, there is an automatic code generation from the models. In addition to just development, model-driven engineering (MDE) encompasses all the other tasks of the software engineering (SE) process such as testing and maintenance, and thus, MDE is considered a superset of MDD. On the other hand, model-based engineering (MBE) is a process, where models have still important roles although they are not necessarily the key artifacts of the development. For example, software designers specify the diagrams on paper or by using modeling tools, but then these diagrams are directly handed out to the software developers to manually write the code (i.e., no automated code generation). In that sense, all model-driven processes are model-based but not the other way round [17]. Note that Brambilla et al. differentiate between "model-based" and "model-driven" approaches for prescriptive modeling [15] in their terminology [16]; however there is also descriptive modeling [18], where MBE might be used. Therefore, the terminology used in this study is enriched and synthesized by "sketching" with informal diagrams, which also plays an important role in descriptive modeling. Figure 1 visually depicts all these concepts as discussed above.

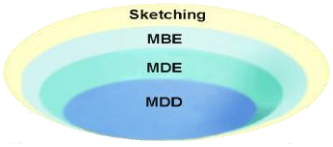


Figure 1: The relationship among MDD, MDE, MBE and sketching

In this study, software modeling usage has been decomposed into two main categories (i.e., descriptive and prescriptive modeling) and four main patterns (i.e., no modeling, sketching, model-based and model-driven) as depicted in Figure 2.

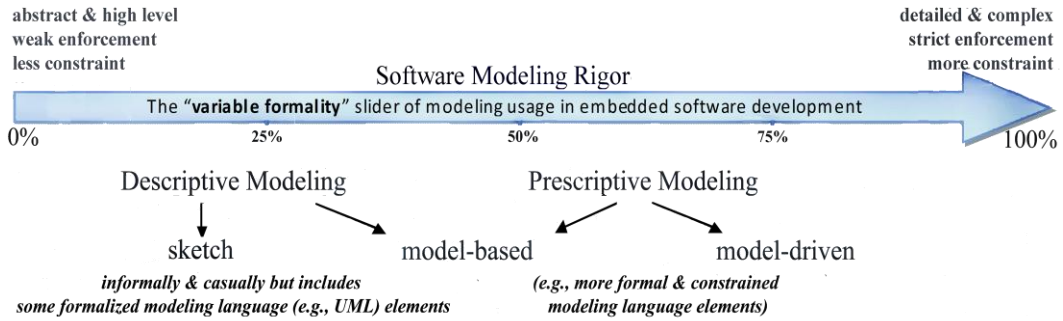


Figure 2: The "variable formality" slider for software modeling usage approaches

Throughout this study, "*diagram*", which means a drawing that visually represents a thing to explain how it works by showing the relation between its components [19], might be either informal (e.g., sketch) or formalized (e.g., model); so any drawing (e.g., descriptive or

prescriptive) is considered as a "diagram"[2] . In descriptive modeling, the diagrams classify actual objects, events, and processes into categories; whereas prescriptive ones specify what is expected of systems' components and how to develop them [15]. Descriptive diagrams are created from observations for a specific intent and once the intent is satisfied, they might lose its importance. On the other hand, in the context of prescriptive modeling, the subject does not exist yet and the diagram is derived from the information available at that time. The primary purposes of descriptive modeling are understanding and communication, while the primary purpose of prescriptive modeling is development [15]. That distinction provides a formal justification between the diagrams developed and used for analysis and design, which depends on the purpose of modeling and affects the modeling rigor (formality). As seen from Figure 2, "*the variable formality*" slider of modeling usage (i.e., modeling rigor) depends on these categories. For example, when you have an abstract and high level modeling approach, which has weak enforcement, you are at the beginning of the variable formality slider (e.g., sketch); on the other hand, when you have more detailed and complex modeling approach, which has strict enforcement, you are closer to 100% of modeling rigor (e.g., model-driven). In other words, depending on the characteristics of a diagram, software modeling usage degree varies.

According to various sources, MDE is considered as the most popular approach and the state-of-the-art in software abstraction while modeling [16, 20, 21]. MDE helps software engineers manage the embedded systems' challenges without accidental complexities [22] by automating SDLC artifacts not only in implementation but also in testing and maintenance [23]. Moreover, several studies point out the necessity of MDE in the embedded domain to minimize the effects of platform heterogeneity (e.g., [24]) besides validation and verification (e.g., [25]). On the other hand, there are also some criticisms on MDE since it might create significant hurdles to demonstrate sufficient value to satisfy the needs of all different modeling stakeholders (e.g., for sketch users[3], depending on their purposes, there is no need for MDE deployment and also related costs) and hence, stakeholders should not be forced to adopt MDE. Moreover, some other studies claim that MDD, which is the subset of MDE, creates other problems during automatic generation of software artifacts and moves complexity rather than reducing it. Redundancy, which is caused by representing different views or levels of abstraction on the same model; round-trip engineering during model transformations; and need for more expertize are the main challenges in MDD [26].

Since there is a danger that resources are being wasted, deciding when to model or in what degree and with how much modeling rigor (e.g., as a sketch without modeling language formality or by automating software artifact generation as in MDE with strict enforcement) are frequently asked and challenging questions for software teams. Therefore, there is a need to identify the relations between the characteristics of modeling (e.g., modeling rigor, purpose, code correspondence, stakeholder, medium type used, SDLC phase, benefits, costs, etc.) to respond to these challenges. A potential approach to resolve the challenge would be to identify, define and use "*modeling patterns and cultures*", which might be analogous to the characterization models, that is defined and tailored for software process improvement (SPI) (e.g., Software Sub-Cultures [27]). In other words, a model, which defines software modeling

---

[2] See Section 4.1 for the conceptual model of development and usage of software modeling, where "*diagram*" is the backbone of all modeling entities.

[3] In this study, as being one of the main modeling patterns, "*sketch users*" are the stakeholders, whose modeling purposes are communication or understanding. They either use no formal modeling language or some elements of a formalized modeling language selectively. They mainly use analog media (e.g., paper or white/blackboard) while modeling.

characteristics in embedded software development project might assist modeling stakeholders to realize an effective modeling approach with respect to these characteristics.

There are different definitions for "*pattern*" in the literature. It is defined as *"a particular way in which something is done, is organized, or happens"* [28] or *"consistent and recurring characteristic or trait that helps in the identification of a phenomenon or problem"* [29]. In SE literature, there is the "pattern" concept to rely on proven solutions to recurrent design challenges like "software design pattern" [30, 31]. In this study, a "*modeling pattern*" consists of specific characteristics of modeling (e.g., purpose, medium type used, modeling language type, SDLC phase, etc.), which helps to identify stakeholder's modeling practices; whereas a "*culture*", which is seen as a particular group of "*modeling patterns*", consists of different combinations of these characteristics for an effective modeling approach to better guide stakeholders with necessary and sufficient process and tool improvements.

In the literature, there are two research studies related to the modeling patterns and categories (See Section 2.1 for the details of these studies). Kleppe, Warmer and Bast classified the modeling usage as maturity levels by taking only one of the characteristics of modeling (i.e., "modeling formality") [32]. According to Kleppe et al., there are six (0 through 5) Modeling Maturity Levels (MMLs) in software development projects, in which there are different types of modeling usage based on "modeling rigor". They claimed that the awareness of the different MMLs enables modeling stakeholder to make an assessment of her/his own modeling practice and may trigger her/him to try to reach for a higher level [32]. They also thought that the goal should be in MML-5, which is "models only" [32]. However, there are other significant characteristics (e.g., "purpose", "medium type used", "SDLC phase", etc.), which characterize and affect the modeling usage patterns and cultures. Moreover, different characteristics of modeling process need not necessarily force modeling stakeholders up the maturity level (i.e., into level MML-5) with respect to a single dimension such as rigor. The variety of modeling characteristics are related with different purposes, notations, tasks and roles.

The second study focused on Unified Modeling Language (UML) usage categories and revealed that there are different categories of what 'using UML' means in practice [33]. In her study, Petre interviewed with only software developers and did not focus on the other modeling stakeholders like software testers or software project managers. The majority of those interviewed simply do not use UML, and those who do use UML tend to do so selectively and often informally (See Section 2.1). The different patterns imply different purposes and needs – and hence different implications for tool support [33]. However, software modeling, which also includes Domain Specific Languages (DSL), is not restricted with just UML usage. Although UML has been publicized as "*defacto standard*" of modeling, there are others, who think UML is problematic due to the complexity of its semantics and difficulty while learning it [13, 34]. Moreover, some other studies claim that MDE has more potential when using formal languages especially DSLs as opposed to graphical languages. Greenfield et al. argued that although UML is useful, it is not appropriate for MDD since it is designed for documenting -not for programming- and they promoted DSLs instead [35]. As seen, there is a gap between what constitutes "modeling" (e.g., including DSL usage) and the classification reported by [33], which focused on only one graphical modeling language (i.e., UML).

Moreover, different modeling stakeholders might spend time, budget and effort to investigate different modeling practices during feasibility analysis for relevant modeling languages, diagram types, modeling tools, etc. with respect to their profiles (e.g., according to their SE roles, project team size, the target sector of the products, etc.). Hence, there is also a danger to waste the resources while trying out unusable or not yet experienced modeling practices, which potentially increases initial entrance cost. Therefore, the modeling stakeholders would benefit from a characterization model, which will decrease these feasibility costs.

There are not any studies in embedded software industry in particular that define the characteristics of modeling and identify the modeling patterns and cultures, which helps to improve modeling practices that can be used in different phases of SDLC by wide audiences of SE roles (e.g., from software developer to tester and systems engineer to project manager).

In this study, we focused to fill all these gaps in the existing literature by identifying and defining modeling patterns and cultures in embedded software industry with a characterization model called MAPforES standing for Modeling Approach Patterns for Embedded Software. The model enables modeling stakeholder to identify a commonsense approach of modeling for her needs, by utilizing the modeling community's prior experiences. This model, not only identifies patterns and cultures of the modeling stakeholder, but also guides process and tool improvements for modeling by referencing to a set of commonsense industrial practices in embedded software development projects.

1.2 Research Goal and Strategy

The main goal of this PhD study is to identify and define different modeling patterns and cultures of embedded software development projects for enabling the modeling stakeholders with a characterization model to improve current modeling practices by getting strategically important questions for commonsense industrial modeling practices to achieve an effective modeling approach with respect to the characteristics of modeling (e.g., purpose, medium type used, benefits, costs, etc.).

The main goal is decomposed into three sub-goals. The first sub-goal is to understand the latest state-of-practice of software modeling and MDE together with the benefits and challenges. The second sub-goal is to identify the characteristics of modeling in the embedded software development projects in different contexts. We assume that there are different modeling practices used for different needs and we can identify and group significant characteristics to establish modeling patterns and cultures. In doing so, to utilize these patterns and cultures, this study, as a third sub-goal, aims to construct a characterization model to find out modeling patterns and cultures in embedded software industry to improve current practices.

Based on the above goals, this PhD has the following research questions (RQs):

RQ1: What are the modeling usage patterns and cultures in the embedded software development projects?

RQ1.1. What is the current state-of-the-art and practice of software modeling and MDE in embedded software industry?

RQ1.1.1. What is the current state of modeling in the embedded software industry?

RQ1.1.2. What is the current state of MDE adoption in the embedded software industry?

RQ1.1.3. What are the achievements, challenges and consequences of using MDE in the embedded software industry?

RQ1.2. What are the significant characteristics of modeling in the embedded software development projects?

RQ2: How can a modeling stakeholder be guided to adopt commonsense industrial modeling practices?

RQ2.1. What are the commonse industrial practices (with similar demographics) doing while modeling?

RQ2.2. What are the recommendations for the corresponding modeling pattern?

RQ3: Is the proposed model useful and generalizable?

RQ3.1. Does the model reflect stakeholder's current modeling pattern and culture?

RQ3.2. Is the model useful and conceptually insightful?

The research strategy includes two phases; "building the existing knowledge" and "iterative model building" as depicted in Figure 3. It can be categorized as constructive since it builds an innovation based on the existing knowledge [36].

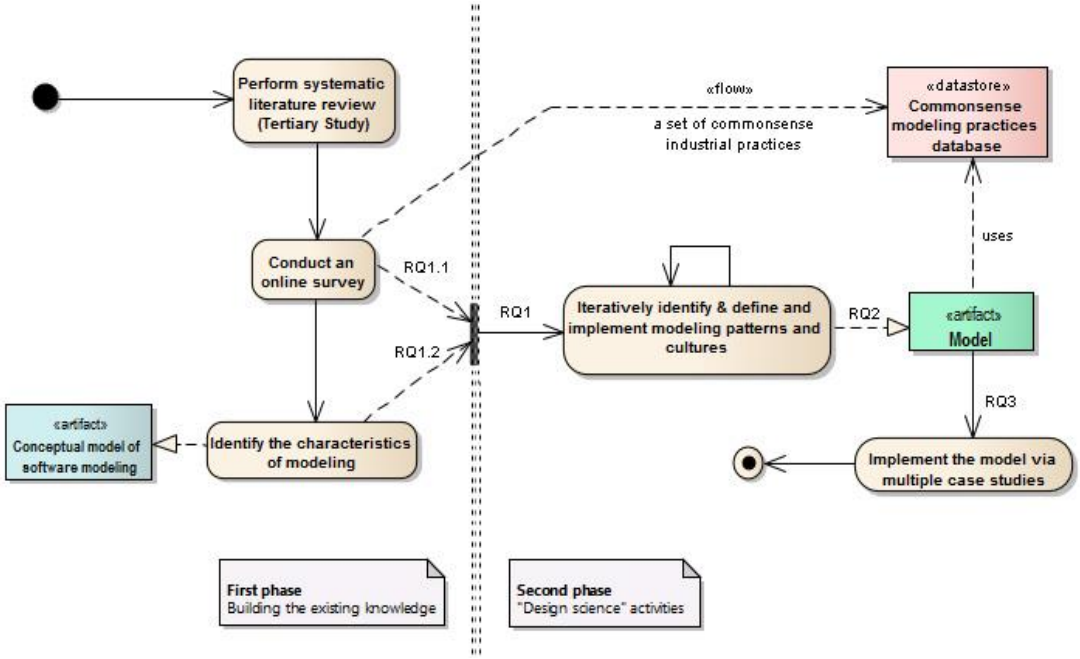


Figure 3: Research process

In order to build up this "*existing knowledge*", at the beginning of the research, a systematic literature review is performed to understand the related work on this domain as suggested in [37]. The results of this baseline are planned to be used in later phases (e.g., during designing survey questions as identifying the purpose(s), motivation(s) and challenge(s) for modeling and while creating the conceptual model for software modeling). We then conduct a survey to determine the state-of the-practices. The survey results are used to establish a commonsense practices database (i.e., a set of common industrial practices on software modeling) and to identify the significant characteristics of modeling. A conceptual model for the development and usage for software modeling, which is enriched by expert opinions via semi-structured interviews [38], is also presented to better characterize these significant characteristics.

After investigating the relations between these characteristics, in order to finalize RQ1, the modeling patterns are identified and categorized in two iterations. During this iterative process, firstly, as an attempt to create things that serve human purposes, a preliminary model is created to achieve one of the main activities of design science (i.e., "build") as described in [39]. Then, this preliminary version is validated with case studies via semi-structured interviews and improved accordingly. After grouping resultant patterns according to their characteristics, the modeling cultures in the embedded software development projects are defined and further refined by expert opinions. In order to address the need for software modeling practices' improvements, which might help modeling stakeholders to know beforehand what similar profiles (e.g., similar SE positions, target sector of products, etc.) are doing while modeling, a commonsense modeling practices database is added to the model (i.e., the model now includes the set of common industrial practices, which is constructed by survey data). Hereby, the model becomes an artifact, which not only identifies patterns and cultures of the modeling

stakeholder in embedded software development projects, but also gives suggestions based on commonsense modeling practices, which addresses RQ2. In that sense, the model guides process and tool improvements for modeling by referencing a set of industrial practices, which are formed by modeling community's prior experiences. By this way, constructing an innovation for a specific purpose is completed as a "build activity" of design science [36].

Finally, to address RQ3, "evaluate activity" (i.e., the other main activity) in design science, which is the process of determining how well the artifact performs [36] is achieved by implementing the model via multiple case study strategy as described in [38, 40, 41].

1.3 Contribution and Significance of the Study

This study provides a new insight for both conceptual and operational issues of software modeling in embedded software development; therefore both practitioners and also academia (e.g., researchers and also educators) will benefit from the research contributions of this study.

The main contribution of this dissertation is the identification of modeling patterns and cultures by investigating the significant characteristics of modeling in embedded software development projects. In doing so, to utilize these identifications, a characterization model (MAPforES) to identify and define modeling patterns and cultures in embedded software development projects with a set of commonsense industrial modeling practices is defined and implemented for software organizations. In that sense, its theoretical contribution is lying in identifying the current state-of-the-practices of modeling and MDE (e.g., with systematic review, survey results) and in characterizing software modeling (e.g., conceptual model of development and usage of software modeling, and the characteristics of modeling) in embedded software development projects. With the help of these theoretical contributions, both researchers and educators would benefit from their implications and this will also encourage more academia-industry collaborations in this domain. Note that all case study stages in this dissertation (e.g., survey, interviews, etc.) were conducted in both local and global scale with high number of practitioner participants by focusing all aspects of software modeling usage and practices in the world-wide embedded software industry, which is also important with respect to the novelty and validity of this research. On the other hand, for its practical contribution, the resulting artifacts of this thesis can be used by any modeling stakeholder in the embedded software industry, with a variety of different SE roles from software developer/programmer to tester, who would benefit from commonsense modeling practices depending on their profiles to achieve an effective modeling approach.

The significance of this study is to being the first research in the literature, which defines and characterizes the modeling patterns and cultures in the embedded software development project by focusing on all significant characteristics of modeling (e.g., not only "modeling rigor" but also "purpose", "medium type used", "stakeholder profile", etc.) and filling the gap of what constitutes "software modeling" (e.g., including DSLs and other formal languages beyond UML usage). Additionally, the model presented here, MAPforES, is also known to be the first wide-coverage model, which not only identifies patterns and cultures of the modeling stakeholder, but also enables process and tool improvements for modeling by referencing to a set of commonsense industrial practices in embedded software development projects.

1.4 Structure

The rest of the thesis is divided into six chapters. Chapter 2 gives the background of this study, in which literature review is presented. Chapter 3 presents online survey, which depicts the state-of-the-practice in software modeling in the embedded software industry. Chapter 4, presents the modeling patterns and cultures by first presenting a conceptual model and the

characteristics of diagram development and usage in embedded software domain; and then defining the modeling patterns and cultures. The results of the implementations of the characterization model in various organizations by utilizing a multiple case study strategy is presented in Chapter 5. Finally, Chapter 6 provides an overall conclusion of this study and discusses some of the future research directions.

CHAPTER 2

EXISTING LITERATURE ON SOFTWARE MODELING PRACTICES

This chapter consists of two sections. The first section gives the related studies for modeling levels, patterns and categories in software development projects. The second section analyzes existing systematic review studies on software modeling and MDE to generate inputs for the following studies (e.g., survey design).

2.1 Modeling Patterns and Categories

There are two research studies, which have investigated the modeling patterns and categories in the literature. According to Kleppe et al., modeling usage patterns are classified as maturity levels and there are six (0 through 5) levels of MMLs in software development projects [32]. This concept is very similar to Capability Maturity Model (CMM) used for software process improvement. Each level of MML is defined as presented in Table 1:

Table 1 Modeling Maturity Level (MML) according to Kleppe et al. [32]

Level-0 "No Specification"	*"The specification of software is not written down. It is kept in the minds of the developers"*
Level-1 "Textual Specification"	*"The software is specified by a natural language text, written down in one or more documents"*
Level-2 "Text with Models"	*"A textual specification is enhanced with several models to show some of the main structures of the system. The models often take the form of diagrams"*
Level-3 "Models with Text"	*"The specification of software is written down in one or more models. In addition to these models, natural language text is used to explain details, the background, and the motivation of the models, but the core of the specifications lies in the models.* *The transition of model to code is done mostly manually. Keeping the models up-to-date is often considered to be unimportant and to time consuming.* *The code is the thing that is polished until the customer is satisfied.* *The code is the thing that is changed when requirements change or bugs must be fixed. The code is the product."*
Level-4 "Precise Models"	*"The specification of the software is written down in one or more models. Natural language can still be used to explain the background and motivation of the models, but it takes on the same role as comments in source code.* *At this level the models are precise enough to have a direct link with the actual code. Because of this direct link between models and code, it is possible to generate large portions of the code automatically. Changes to the system are done in the models, after which the code is regenerated. In effect the models become part of the source code. This means that it is easy to keep models and code up-to-date"*
Level-5 "Models only"	*"The models are precise and detailed enough to allow complete code generation. The code generators at this level have become as trustworthy as compilers, therefore no developer needs to even look at the generated code. It is as invisible as assembler code is today. In other words, the MML 5 is the modeling Valhalla.* *Unfortunately, there are no modeling languages in which we can write MML 5 models"*

Kleppe et al. claimed that the awareness of the different MMLs enables modeling stakeholder to make an assessment of her/his own modeling practice and may trigger her/him to try to reach for a higher [32]. They also thought that software engineers' goal should be in MML-5 while using software modeling [32]. However, in 2003, they expressed that they cannot work at Level-5 at that time. They claimed that *"only within specific and limited application domains there are languages and tools that could achieve this"* (e.g., "Executable UML") and the challenge for all, who work in this field is to reach this level [32].

On the other hand, Petre, who focused on only modeling language usage (i.e., UML), reported a series of interviews with 50 software developers in her empirical study [33]. The results showed that there are *"very different models of what 'using UML' means in practice",* with different implications. Accordingly, five categories of UML use were identified (Table 2):

Table 2 UML usage categories according to Petre [33]

Category of UML use	**Characteristics**
No UML (70%)	*"This category doesn't use UML with some specific criticisms (e.g., 'Lack of context', 'overheads of understanding the notation', and 'Synchronization/ Consistency issues'"*
Retrofit (2%)	*'Retrofit' UML use means, by and large, documenting things after-the-fact "This group doesn't really use UML, but retrofit UML in order to satisfy management or comply with customer requirements"*
Selective (22%)	*"UML is used in design in a personal, selective, and informal way, for as long as it is considered useful, after which it is discarded.* *There are different aspects of selective use as 'UML as a 'Thought Tool', 'Communicating with Stakeholders', 'Collaborative Dialogues', 'Adaptation', and 'Keeping It Small – Selective Traction'"*
Automated code generation (6%)	*"UML is not used in design, but is used to capture the design when it stabilizes, in order to generate code automatically (typically in the context of product lines)"*
Wholehearted (0%)	*"This usage should be organizational, top-down introduction of UML, with investment in champions, tools and culture change, so that UML use is deeply embedded."*

Petre interviewed with only software developers and she did not analyze any other modeling stakeholders. According to the results, the majority of those interviewed in this study simply do not use UML, and those who do use UML tend to do so selectively and often informally. According to this study, using UML did not necessarily lead to success with respect to project budget and time [33]. The interviewees, who used to be wholehearted use, which means that they were wholehearted but later retracted to a different usage category, reported that they did not achieve market requirements, despite the investment, or projects did not satisfy clients, who found the UML representations complex and difficult. So, Petre's study also shows that there might not be upper "level" concept (e.g., wholehearted is the most intensive UML usage category) since it cannot guarantee the optimal cost-effective solution. Moreover, the results showed that companies, which use MDE, tend to use multiple modeling languages such as DSL in addition to UML since DSL notion is very product/implementation focused [33].

The model presented in this dissertation, which identifies and defines modeling patterns and cultures of embedded software development projects, proposes that the modeling approaches vary since the characteristics of modeling differ among systems as well as among sectors (See Section 4.2 for these characteristics). Kleppe et al. said: "*your modeling maturity level indicates how* ***complete***, ***precise***, *and* ***consistent*** *your model should be"* [32], but in fact, this is just ONE of the significant characteristics of diagram (i.e., modeling rigor) and according

to other characteristics (e.g., purpose, media type, benefit, cost, etc.), there is no "*maturity level*" concept, but "*pattern and culture*" in the real industrial context.

In the categorization of patterns and cultures reported in this study (despite MML), more rigorous ones (the ones who has more modeling rigor on the "*variable formality*" slider, see Figure 2) can use other modeling approaches, which is less rigorous. A "higher" culture can use the characteristics of the "lower" cultures and the modeling stakeholder might apply the modeling stakeholders' lower level patterns' modeling practices, if necessary; but not vice versa (e.g., model-driven users can also use both sketch and model-based approaches according to the characteristics of modeling such as purpose; hence MML might vary). For example, during "Analysis" phase of SDLC, since the purpose might be just "Communication" and "Understanding", sketch on an analog media with no formal modeling language is enough to achieve a cost-effective approach; but for "Code generation" purpose in "Implementation" phase of SDLC, more precise models are needed in digital media with a formal modeling language. Therefore, a "higher" culture does not necessarily entail a more "correct" or "mature" use of modeling with respect to job/task requirements of the stakeholder although a change into a "higher" pattern might allow the stakeholder to better use software modeling with possibly some extra costs and challenges.

Although Petre's study has some similarities with respect to some similar characteristics such as purpose and selective and informal usage of modeling [33], the study presented here fills the gap what constitutes "modeling" since it also includes DSLs not just UML usage. Although UML has been publicized as "*defacto standard*" of modeling, it is also problematic due to the complexity of its semantics and difficulty while learning it [13, 34]. Furthermore, MDE has more potential when using formal languages especially DSLs in opposed to graphical languages (e.g., UML) [35]. According to Petre, the different categories (i.e., "patterns" in her terminology) imply *"different purposes and needs – and hence different implications for tool support"* [33]. In that sense, she also mentioned about some (but not all) characteristics of modeling (Note that Petre's terminology on "*pattern*" is similar to the "*culture*" in this research, which is formed by a combination of patterns).

This dissertation is to being the first research in the literature, which focuses on all significant characteristics of modeling (e.g., not only "modeling rigor" but also "purpose", "medium type used", "stakeholder profile", etc.) and fills the gap of what constitutes "software modeling" (e.g., including DSLs and other formal languages beyond UML usage) in the embedded software development project.

2.2 Empirical Evidence in Software Modeling and MDE

At ICSE04, Kitchenham, Dybå and Jørgensen suggested SE researchers should adopt ''Evidence-based Software Engineering", which aims to apply an evidence-based approach [42]. In this context, evidence is defined as a synthesis of best quality scientific studies on a specific topic. Systematic literature reviews (SLR) are one of the main method of this synthesis [37]. They are referred to as "*secondary studies"* and the studies they analyze are referred to as "*primary studies"*. Secondary studies play an important role both in supporting further research efforts and also in providing information about the impact of methods to assist SE practitioners [43].

In order to investigate and understand the latest trends and practices in software modeling and MDE, existing Systematic Mapping (SM) and SLR studies were analyzed. This investigation was crucial since RQs of these referenced studies empirically present inputs for the rest of this study (e.g., methods, languages, diagrams or techniques used to carry out the modeling

approach with different purpose(s), benefit(s), and challenge(s)) to identify and characterize the different modeling patterns).

In that sense, this section gives a brief information about a sub-part of a *tertiary study* (as SLR of secondary studies), whose resulting data sets (i.e., purposes, motivations and challenges) were used during designing survey questions, which is presented in Chapter 3. This survey forms one of the main inputs while identifying and defining modeling patterns and cultures. Please refer APPENDIX A – Systematic Literature Review – Tertiary study for the RQs, search strategy and process of this empirical evidence study. After the data extraction, the following results were found: (Note that since there were different terminologies to indicate the same purpose, benefit or challenge in different secondary studies, to get a common language and get a catalog, similar items were combined in a single item. See [44] for all extracted data in a specific paper).

- The "modeling purpose set" is given in Table 3.

Table 3 Modeling purposes derived from the tertiary study

Understanding a problem at an abstract level	Code generation
Communication	Test case generation/Model-based testing (MBT)
Business process automation	Documentation generation
Documenting design	Model to model (M2M) transformation
Model simulation	Prototyping

- The "modeling benefit set" is given in Table 4.

Table 4 Modeling benefits derived from the tertiary study

Cost savings	Shorter development time
Ensuring source code & design model compatibility	Time and effort reduction
Quality improvement	Reliability
Portability	Traceability
Productivity	Reusability
Maintainability	Team collaboration
Extensibility	Test effectiveness
Expandability	Interoperability
Modularity	Expressiveness
Guaranteeing the verification of important properties of a system in the early development stage	

- The "modeling challenge set" is given in Table 5.

Table 5 Modeling challenges derived from the tertiary study

Tool support
Model quality (i.e. how to define, assure, predict, measure, improve and manage it?)
Model verification/validation techniques
Modeling expertize in the company
Modeling languages (i.e. domain specific modeling language (DSML) needs)
Optimization and performance issues with automatic code generation
Software certification (i.e. for safety-critical systems) with automatic code generation
Training
Transformation/merging of models (i.e. how to integrate/merge models in different projects?)
Understanding and acceptance of the model driven concept / Organizational resistance to change

While mentioning about tool support problems, these secondary studies pointed out specific challenges related to these modeling tools (e.g., Back/Forward compatibility issues between

tool versions, difficulties in taking technical support from the tool supplier, difficulties with code generation capabilities, and many usability issues in their editor, etc.).

All these results on modeling purposes, benefits and challenges are systematically used within the survey, which will be presented in Chapter 3.

2.3 Surveys on State-of-the-Practices in MDE

There are only three survey studies [45-48], which have investigated the-state-of-the-practice of model-driven techniques via opinion surveys. Some of the surveys have focused on the embedded systems domain, while others are generic in terms of the domain. Table 6 summarizes those three surveys, which have been conducted in this topic. Apart from these embedded systems-related surveys, there are also several studies, which investigate mainly UML-based modeling [20, 49-57], which we also briefly review in Table 6.

The study in [45] was a 2011 world-wide survey of 67 participants which investigated the reasons of introducing model-based development in only one domain of embedded systems (i.e., automotive & transportation) with its costs and benefits. It focused on only "development" phase (MBD) of the entire "engineering" (MBE) process. The main findings from this study were: (1) The top three motivations of model-based development are: "improvement of the product quality", "development of functions with high complexity", and "shorter development times"; (2) Positive experiences of MBD are "communication with other colleagues", "possibility of early simulation of the functional model", "easier maintenance if the generated code is not changed manually"; whereas "high process of redesign costs" and "tool costs" are the negative experiences; and (3) MBD can bring significant cost savings, but only with a "well-chosen" approach and an established development process with defined interfaces and role allocations. Otherwise MBD can be much more expensive than a hand-coded manual software development.

The study in [46] investigated the use of UML and MDE in the Brazilian embedded software development industry. According to the results: (1) 45% of the participants use UML either completely or partially; (2) The participants report increases in productivity and improvements in quality (e.g., maintainability and portability) as key benefits of model-driven techniques; (3) Models are mainly used for documentation and design with a little of code generation; (4) Class, sequence, use case, and state machine diagrams are the most popularly used diagram types. One of the interesting results is that experienced users (i.e., the ones with more than 10-year experience) can better assess the benefits of UML during the embedded software development. On the other hand, the major problems encountered in UML adoption refer to the lack of modeling skills, the lack of appropriate tools, and the strict time requirements.

The recent study in [47] was a 2014 European survey, which investigated the current state of MBE in embedded domain by analyzing its positive & negative effects and its shortcomings. Its target projects were applying model-based approaches, where its participants had already used model-based techniques (93%), therefore, it lacks of general embedded software professionals contribution. The results confirmed that MBE is widespread in the embedded industry. The main finding from this study was that models are not only used for communication and documentation purposes; they are "key artifacts of the development processes", and they are also used for simulation and code generation. Moreover, while participants reported mostly positive effects of MBE, the results showed some major challenges (i.e., adoption, tool support and its interoperation). The same group of authors presented a very recent study [48] in 2016 in which they analyzed the results of [47] in more depth, and offered insights into the current industrial practice.

The survey in [49] was a 2005 world-wide survey of 131 participants, which investigated the adoption and usage of UML by analyzing its perceptiveness and perceived ease of use. The results of this survey showed: (1) The majority viewed UML as "accurate, consistent, and flexible enough" to use on development projects; (2) Developers seemed eager to use UML, which was spreading across the world; and (3) Use case, class, and sequence diagrams are the most popularly used diagrams types.

In [50], how and why using UML were investigated. According to their results, UML may be too complex supported by phrases such as "Not well understood by analysts" or "insufficient value to justify the cost". Respondents of [50] reported that class, use case, and sequence diagrams were the most popularly used diagrams; whereas collaboration diagrams were used the least. The other interesting result was that class, sequence and state machine diagrams were considered as the most useful for capturing technical details; whereas use-case narratives, activity and use case diagrams were the preferred means with respect to customer involvement.

The study in [51] investigated UML usage and its quality in actual projects. The results addressed UML's problems, where the main problems were synthesized as: "scattered information", "incompleteness", "disproportion" and "inconsistency". The results in [51] showed that UML practices should be improved in some areas (e.g., modeling uniformity and standards, development of project-specific reference architectures and patterns).

The survey in [53] was a 2008 European survey of 80 participants, which investigated the impact of UML modeling styles. The findings focused only on the improvement in software development quality and productivity. One of the results showed that the benefits of UML on productivity was perceived mostly in the design, analysis, and implementation phases of SDLC.

On the other hand, there were also some national surveys on UML. The results of survey in [52], which investigated the use of UML in Bulgaria, showed that UML was not properly used in the industry and more training was needed.

A Greek survey [54] with 91 participants, which mentioned "model-driven" concept but only with UML, investigated the role of UML in different types of applications (e.g., web, windows, or embedded). The findings indicated that UML was successfully used by the majority of participants. Among the results: (1) The most popular diagrams were class, use cases and activity, whereas the least used diagrams were package and state machines; (2) Even though UML was extensively used, its extensions and profiles (e.g., SysML) were not well known and majority was not familiar with them. The main conclusion was that despite its limitations and further extensions needed, UML is a general-purpose modeling language that is supported by various tool vendors [54].

There are also surveys on MDE in general [20, 55-57], which do not explicitly address embedded software industry as their target. The study in [55] was a 2008 survey with two thirds of the respondents from Canada and the United States, which investigated software modeling experiences. According to its results, UML was the dominant notation. Participants reported that "the biggest problem of model-centric approaches is keeping the model up-to-date with the code". Moreover, another interesting result is that participants working on real-time systems agree that their organizational culture does not like software modeling.

The study in [56] was a 2011 survey of 250 participants which investigated the adoption and application of MDE. According to the results: (1) MDE represented a need for new skills, including modeling expertise (in which significant additional training is needed); (2) Code generation was very important as one of MDE benefits, but integrating this code into existing

projects could be problematic; and (3) Class, activity and use case were the most popularly diagrams. The same authors presented another study [20] by identifying the importance of organizational, managerial and social factors, as opposed to only technical factors, which affect the relative success, or failure, of MDD.

Another study [57] was a 2011 Italian survey which investigated the modeling languages, processes and tools in the Italian software industry with MDE. According to its results: (1) 68% of participants reported to always or sometimes use models, and among them, 44% reported generating codes from models; (2) The participants who do not use models commonly stated that modeling requires too much effort and time investment (50%) or was not useful enough (46%); (3) Models were used mainly in larger companies; and (4) a majority of all the participants using models (76%) apply UML although DSLs are used as well. The same authors presented another study [58] in which they analyzed the results in more depth.

The novelty of our survey in comparison to these studies is that, our study is not limited to neither a sub-domain of the embedded systems (i.e., automotive & transportation), nor a subset of engineering phase (i.e., development), nor a specific region. This survey focused on all aspects of modeling usage in the world-wide embedded software industry. In other words, our survey intents to be the first world-wide survey, which focuses on embedded software industry by investigating a wide range of software modeling and MDE practices.

Table 6 Existing surveys explicitly on MDE

Citation	**Scale/ region**	**Number of participants**	**Goal/Focus area**	**MBD/MBE / MDD/MDE**	**Domain**
[45]	World-wide	67	Investigated the reasons of introducing model-based development in a single domain of embedded systems (i.e., automotive & transportation) with its costs and benefits. Focused on only "development" phase (MBD) of the entire "engineering" (MBE) process.	MBD	Embedded systems
[46]	Brazil	209	Investigated the use of UML and MDD in the embedded software development industry	MDD	Embedded systems
[47]	Europe	112	Investigated the positive & negative effects of MBE. It did not address categorization between model-based and model-driven techniques. Same authors presented a very recent study [48] in 2016 in which they analyzed the results in more depth.	MBE	Embedded systems
The survey reported in Chapter 3	World-wide	627	Investigates the degree to which, why and how software modeling and it challenges, shortcomings and consequences.	MDE	Embedded systems

Table 6 (continued)					
[49]	World-wide	131	Investigated the adoption and usage of UML by analyzing its perceptiveness and perceived ease of use.	MBD	In general
[50]	No information given	182	Investigated how and why using UML.	MBD	In general
[51]	No information given	80	Investigated UML usage and its quality in actual projects.	MDD (only with UML)	In general
[52]	Bulgaria	100+	Investigated the utilization of UML	MDE (only with UML)	In general
[53]	Europe	80	Investigated the impact of UML modeling styles.	MDD (but only with UML)	In general
[54]	Greece	91	Investigated the role of UML.	MDD (but only with UML)	In general
[55]	World-wide	113	Investigated software modeling experiences.	MDE	In general
[56]	World-wide	250	Investigated the adoption of model-driven software development in industry. Same authors presented another study [20] by identifying the importance of complex organizational, managerial and social factors, as opposed to only technical factors, that appear to influence the success or failure of MDD.	MDD	In general
[57]	Italy	155	Investigates the modeling languages, processes and tools with MDE. Same authors presented another study [58] in 2013 in which they analyzed the results in more depth.	MDE	In general

CHAPTER 3

STATE-OF-THE-PRACTICES IN SOFTWARE MODELING

The goal of this chapter is to understand the state-of-the-practices in software modeling and MDE practices in the embedded software industry by addressing RQ1.1. Getting such an overview benefits to understand different modeling approaches by being aware of the trends, successes and challenges in these areas as providing one of the main channel of evidence-based inputs to identify and define the modeling patterns and cultures. To address that need, a survey was designed and conducted in spring of 2015. 642 engineers with 627 acceptable responses from 27 different countries working in different subsectors of embedded software industry participated in this survey.

Although there have been a few prior surveys related to modeling in the embedded software industry (e.g., [45-47]), they have either focused on only one aspect of modeling, (i.e., the use of UML or the use of formal models), or modeling in regional contexts (e.g., UML and model-driven approaches in Brazil or in Greece). There are also some surveys, whose participants were involved with model-based/driven techniques on a single target sector of embedded systems (i.e., automotive & transportation [45]). However, the survey reported here takes a larger scope with a global higher scale (from world-wide) after reviewing all existing surveys (See Section 2.3) and benefitting from SLR (i.e., tertiary study presented in previous section).

3.1 Research Methodology

Survey methodology is a well-established technique for obtaining broad characterization of a particular issue by enabling collection of different information such as opinions, perceptions, attitudes and behaviors [59]. In contrast to experiments and case studies, surveys only collect and investigate information; hence, they are suitable for collecting empirical data from large populations. Although there are different surveying methods [60], in this study, the online survey method was chosen to obtain information from a relatively large number of practitioners in a quick manner so that categorizing and analyzing these data would be easier (Note that the other conventional approach is to conduct interviews, which is usually more effort intensive – and this approach is used in the other parts of this research, see Chapter 4). However, since there is no interviewer, poorly-worded questions might be problematic and the opinion surveys approach may have drawbacks [59]. In order to cope with this challenge, a pilot study was applied before the execution of this online survey (See Appendix B.1 – Survey design and execution for pilot study).

Although it is relatively easy for software engineers to fill out questionnaires, *"they still must do so on their own and may not find the time"* [59]. In that sense, the organization of survey questions are crucial and require special considerations [61]. In order to get a survey with a high quality and reduce the time taken to complete this survey, questions were carefully designed. Individual item based design and organization of the survey to satisfy design criteria is presented in Appendix B.1 – Survey design and execution for designing survey questions.

The research approach used in this survey is the Goal, Question, Metric (GQM)[4] methodology [62]. By using its template [62], the goal is to understand the current state-of-the-art and practice of modeling and MDE in the embedded software domain by identifying to what degree, why and how modeling is conducted with its challenges. Based on this goal, the following RQs were raised, which were previously presented in Section 1.2:

RQ1.1.1: What is the current state of modeling in the embedded software industry?
RQ1.1.2: What is the current state of MDE adoption in the embedded software industry?
RQ1.1.3: What are the achievements, challenges and consequences of using MDE in the embedded software industry?

Note that by answering these RQs, RQ1.1, which enlightens the current state-of-the-art and practice of software modeling and MDE in embedded software industry is achieved; and these findings helps to investigate RQ1.2, which also helps to characterize the modeling patterns.

3.2 Survey Design and Execution

In designing the survey, it was made sure that the survey questions are relevant to the embedded software industry and also capture the most useful information based on the goal and RQ's. During the design, several survey guidelines (e.g., [61, 63, 64]), the systematic literature review (i.e., tertiary study) and also previous experience of executing industrial survey studies, (e.g., [65]) were utilized and benefitted.

The identified target audience in this survey is anyone working in the embedded software development projects, with a variety of different SE roles from requirement engineer to business analyst and from software developer/programmer to tester. This study established a sampling frame composed by a large set of embedded software professionals working in different locations around the world and in different industrial sectors. Please refer Appendix B.1 – Survey design and execution, which also includes sampling method used in this research, the details of designing survey questions, survey piloting & execution and pre-analysis considerations with data validation.

3.3 Results

The survey received 627 acceptable responses from 27 different countries in five continents and different industrial sectors related to embedded software. There was a good mixture of different profiles (both participants and companies), which helps to provide unbiased results from certain types of demographics such as SE roles and target sectors of the companies. The survey showed latest trends and interesting results in the embedded software, which help to characterize the modeling patterns and cultures.

In this section, the findings, which are directly related to the rest of the study (e.g., the results, which help to investigate the characteristics of a diagram, hence the modeling patterns and cultures) are presented. Note that the survey answers given here are the main inputs for the conceptual model and the characteristics of a diagram, which will be presented in Section 4.1 and Section 4.2. All other remaining results, which are not directly related with research question RQ1 of this dissertation, are reported in Appendix B.2 – Results.

---

[4] Goal, Question, Metric (GQM) is a methodology to identify meaningful metrics for measurement process. In this methodology, questions are formulated based on a more abstract goal and metrics are chosen to answer each question.

3.3.1 *Demographics*

Note that the results given here are related with the significant characteristics of a diagram (i.e., the modeling stakeholder's characteristics and the target sector of the product, in which this stakeholder is working), which will be presented in Section 4.2.

In order to understand modeling stakeholder's characteristics, the participants' educational skill-set (their university degree) were asked as a multiple-response question (Q3). (Figure 4). Note that depending on the country, *(since some universities have started to offer new computing disciplines degrees in recent years)*, there might not be such a department and it is better to analyze the underlying discipline in a single item as Computing Disciplines (e.g., computer engineering, computer science, software engineering, information systems) since their "software modeling" curriculum might be similar.

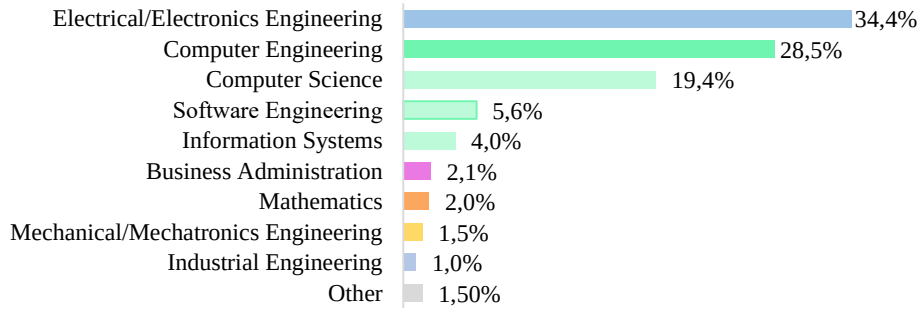


Figure 4. Survey - University degrees

The current positions of respondents (Q4) was also a multiple-response question, so multiple positions could be selected (e.g., a person can be a software developer/programmer and software designer at the same time). The results are shown in Figure 5. Note that, people in different positions, have different viewpoints on SE and related processes [66]. As seen, the survey has a wide range of embedded professionals including from developer to tester and project manager to quality assurance engineer, which supports different viewpoints.

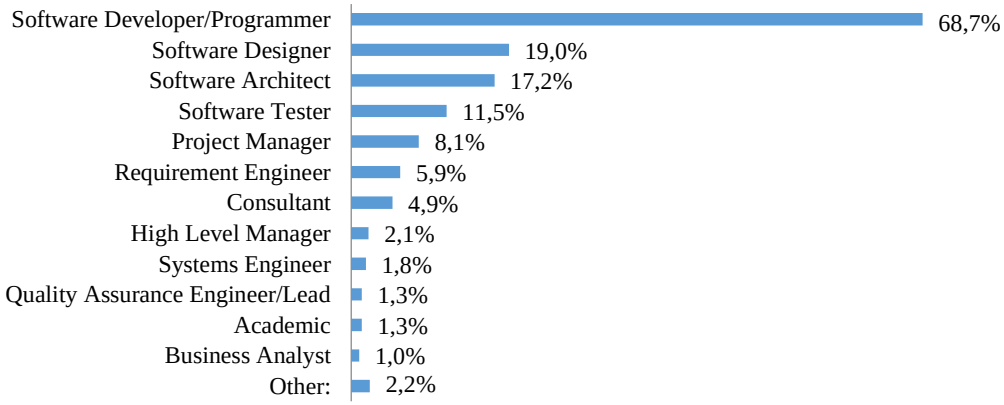


Figure 5: Survey - Current positions

When work experience of the participants in software development was asked (Q5), the majority of respondents have 10+ years (52%) and 6-10 years (40%) work experience. This indicates that the participants are generally experienced industry professionals in embedded systems (*assuming that their work experience is on embedded systems*). The participants were then asked to report their modeling experience (Q11) in software development. The interesting point here is that, although the majority of respondents have 10+ years (52%), which is

followed by 6-10 years (40%) of work experience, in this question the majority is in 6-10 years (46%), followed by 10+ years of modeling experience (40%) (Figure 6). This might have occurred by some possible reasons. Firstly, some respondents might have learned software modeling after getting the job or employment (i.e., after graduation, during the job or with some training). Secondly, modeling in embedded domain might require some initial work experience to understand embedded requirements.

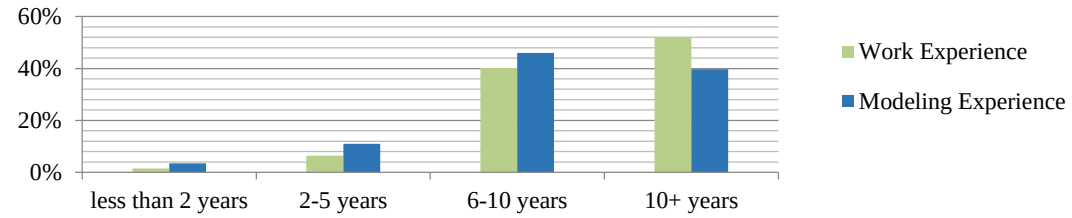


Figure 6: Survey - Work vs. modeling experience of participants, who use modeling

Q12 was again a multiple-response question, in which where/how the participant learned software modeling was asked. (e.g., participants might learn modeling in university and from formal corporate trainings). As expected, "University" is the majority, followed by "On your own" and "Formal corporate training" as shown in Figure 7. The answers are compatible with the previous question, which investigates the modeling experience and explains why 6-10 years modeling experience is the majority. For example, some participants, who graduated from Electrical/Electronics Engineering (EE) department, have learned software modeling after getting the job (after graduation, on his/her own or with formal corporate training). Therefore, his/her work experience is more than modeling experience since he/she did not take any software engineering or computer science courses on modeling during university. However, any computing discipline graduate's work experience and modeling experience are most probably the same. Note that this issue is addressed while investigating the relation between the characteristics of diagram development and usage in Section 4.2.

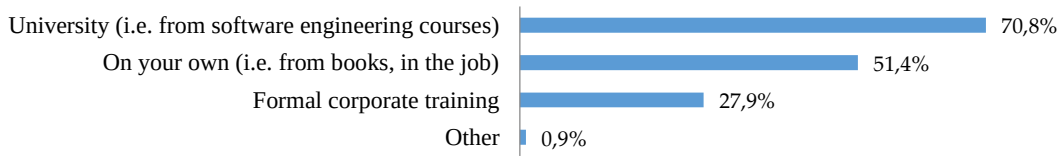


Figure 7: Survey - Where/how software modeling was learned

Notice that in this section, only Q3, Q4, Q5 and the relation between these questions with Q5, Q11 and Q12 were presented since they are directly correlated to one of the characteristics of a diagram (i.e., modeling stakeholder's profile). Please refer [67] and Appendix B.2 – Results for the details of other demographics data.

Q7 was about the target sectors of the products developed by the company, in which the respondent is working (Figure 8). Seven possible choices were pre-given in the questionnaire, after the discussions with embedded software industry partners during survey design. As seen, there is a good mixture of participants from various embedded software industry sectors *(Please see Section 3.3.3 for cross-factor analysis on modeling practices and the target sector(s) of the products developed by the company, which also affects the characteristics of a diagram).*

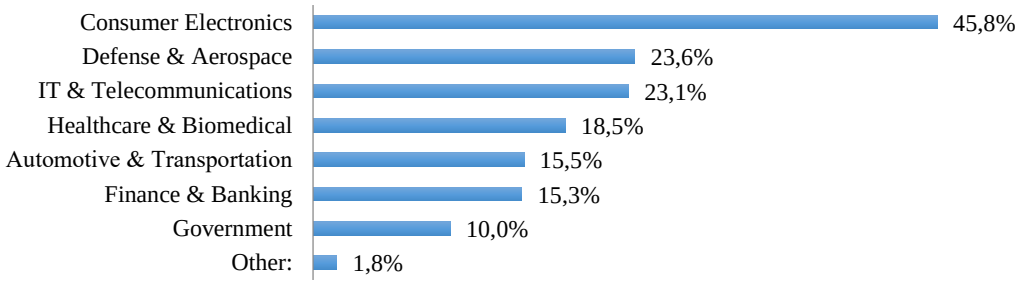


Figure 8: Survey - Target sectors of products

3.3.2 *Software Modeling and MDE-related Questions*

Note that all the questions presented in this section are directly related with the identification of the characteristics of a diagram and the relations between them (which will be presented in Chapter 4) and all other remaining answers, which are not directly related with this investigation (but related to the corresponding RQs) are presented in Appendix B.2 – Results.

Degree of using software modeling in SDLC (Q10)

Q10 investigated how often the participants use software modeling in the SDLC by including both formal and informal usage (i.e., models or sketches) using a 5-point Likert-scale. The results are shown in Figure 9. As it is seen, the "often" choice is the most reported one.

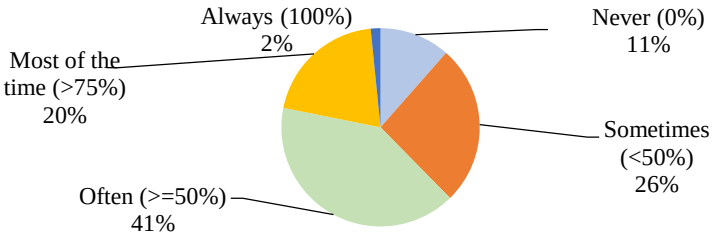


Figure 9: Survey - Degree of software modeling usage

Media used to create sketch or model (Q13)

In this multiple-response question, respondents were asked to report the media they use to create (draw) the diagrams. A 5-point Likert-scale was utilized for the answers and results are depicted in Figure 10. Accordingly, using modeling software/tool on PCs is the most used medium for modeling. Modeling using pen and paper is the next common approach.

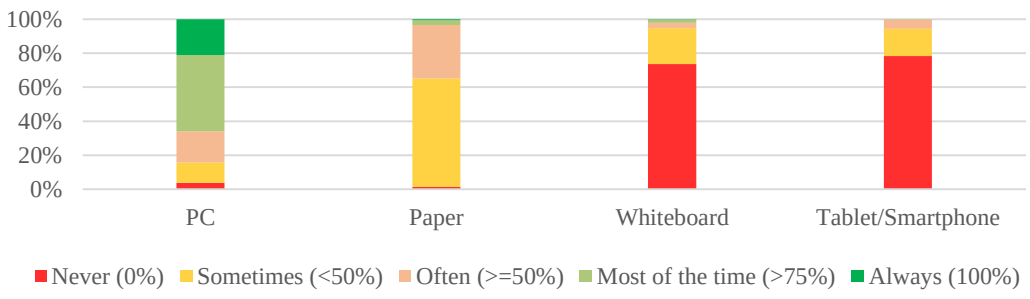


Figure 10: Survey - Mediums to create sketch or model and their usage frequency

The purpose of the modeling and the main software modeling category (e.g., sketching, model-based or model-driven) are strongly related with the medium used and the industrial need. If there is no auto-generation of some software artifacts (i.e., code, document or test scripts, etc. – which means “model-based” usage), analog media like paper or whiteboard are enough for communication or understanding a problem. It does not mean that model-driven users do not use paper or whiteboard; indeed, such analog mediums might be a quick solution for better communication and faster idea sharing in some situations. However, the lifespan of these sketches or diagrams are less than the ones created digitally via PC or tablet/smartphone. In that sense, the digital mediums like PC or tablet/smartphone are advantageous on archiving and have longer lifespan. Therefore, PC is the most used medium since it provides modeling tools (for both model-based and model-driven users) and easier archiving of diagrams (both sketches and more formal models) as being digital.

It is possible that some of the respondents were referring to descriptive and others to prescriptive modeling while answering this question and this is directly related with what software modeling is used for (i.e., the purpose(s), see Q20). As the primary purpose of descriptive modeling is communication and understanding [15], paper is enough to achieve this. Therefore, there is a strong relation between the purpose and the medium used besides the lifespan and archivability of this diagram. This issue is also addressed while investigating the relations between the characteristics of a diagram.

Cross-factor analysis of the above data with Q14 (Modeling languages) showed that the participants, who do not use any formal software modeling (i.e., the ones who draw some sketches), use just paper or whiteboard. On the other hand, the participants, who use any formal modeling language (e.g., the ones, who use UML), usually use modeling tools on PCs. Note that there is a specific question that asks about the modeling tools (Q16).

Modeling languages (Q14)

Notice that any informal usage of modeling (e.g., sketch with no formalized modeling language) is seen as "modeling usage" at the survey and this question is aimed to understand the modeling language that participant use, if any. Multiple modeling languages could be chosen (i.e., participants might use both UML and DSL) in the answers since this was again a multiple-response question. The responses are given in Figure 11. The majority of participants (77%) use UML (not surprisingly), but it is interesting that the second most frequently selected response is “Sketch/No formal modeling language” (65%). Another interesting result is that some respondents chose both UML and also “Sketch/No formal modeling language”, which show that these participants use modeling both formally and informally as in [33] depending on their purposes.

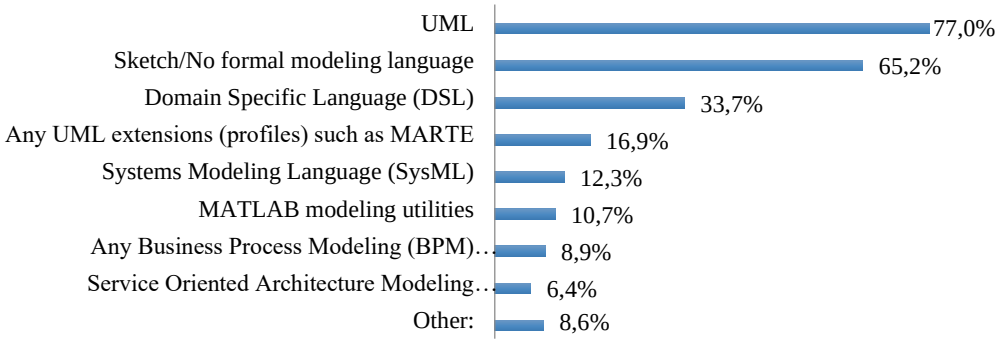


Figure 11: Survey - Modeling languages

Apart from the pre-given choices, many "Other" modeling languages were reported, such as: (1) models in the AUTOSAR (AUTomotive Open System ARchitecture) notation (2) models compliant with the Eclipse Modeling Framework (EMF), (3) Markov Chain Markup Language, (4) models compliant with the Architecture Analysis & Design Language (AADL), (5) Modelica, and (6) EAST-ADL, an Architecture Description Language (ADL) for automotive embedded systems. This denoted that there exists a wide spectrum of modeling languages in this domain and engineers select the modeling languages suitable for their needs in their projects (See Section 3.3.3 for cross-factor analysis of these modeling languages)

Since UML is a general-purpose modeling language, *"its usage is not only restricted to modeling software, but it is also used for system engineering, for business process modeling and for representing the organizational structures"* [54] although there are some specific modeling languages for these disciplines (e.g., SysML for system engineering, BPML for business process). Moreover, although UML is built upon object-oriented concepts such as classes and operation, non-object oriented systems may also be modeled using it. Furthermore, during university (i.e., from SE courses), mostly UML is taught as modeling language. Therefore, UML's popularity is not a surprise [68]. On the other hand, a very recent study on the usage of UML in practice shows that *"although UML is viewed as the 'de facto' standard, it is by no means universally adopted"* [33]. The majority of those interviewed in [33] who do use UML tend to do so selectively and often informally. This finding also supports the ratio of our second most selected response as "Sketch/No formal modeling language".

Programming languages (Q15)

The responses given for this multiple-response question is given in Figure 12. The C language is the first, followed by C++ and then Java. Notice that, although C is the most popular programming language in the embedded world, the total responses for C++ and Java combined, which are object-oriented programming languages are much more than C. MATLAB, C#, BPEL, Ada, Delphi and Smalltalk received some responses, which were in the pre-given answer set. Apart from these pre-given choices, Python, Objective-C, JavaScript, and Scala were among the "Other" answers for this question. Please refer [67] for the details of the answer set.

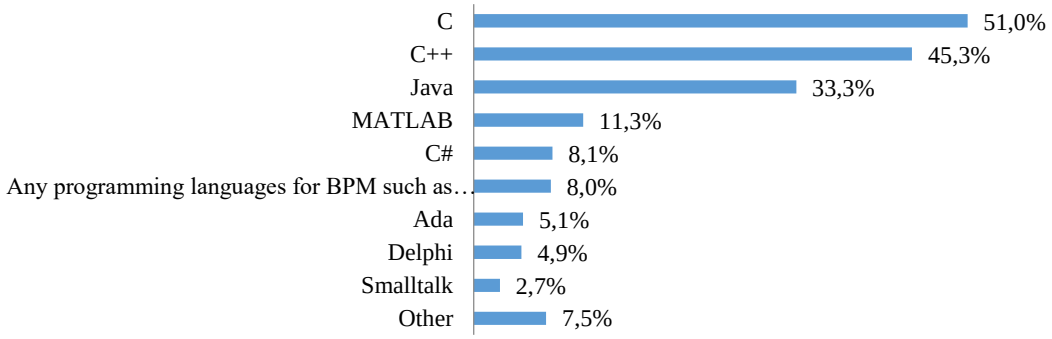


Figure 12: Survey - Programming languages

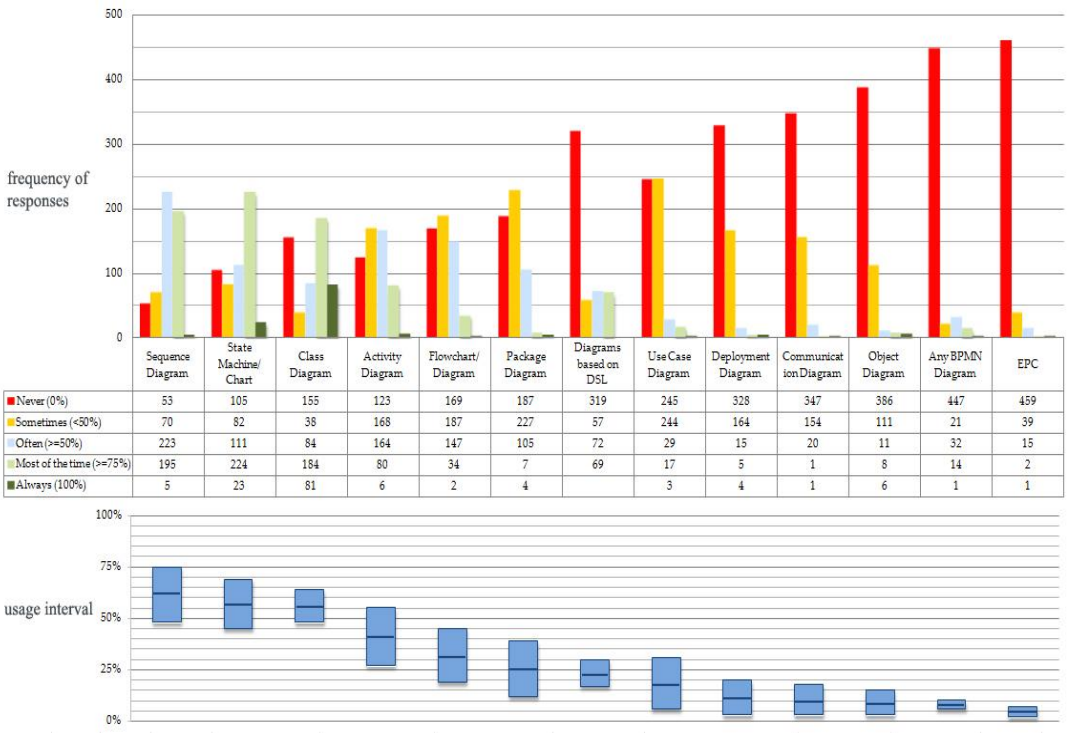
Diagram types (Q17)

Participants were asked about the diagram types that they use while modeling via 5-point Likert-scale. Notice that, it was not mandatory to select a frequency answer on each item, therefore, total responses for each diagram types might vary (i.e., total response for Class Diagram is 542, whereas this number is 516 for Deployment Diagram). Note that the respondents, who state that they were doing informal modeling, make the sketches, which include some essences of UML (e.g., some elements of state machine/charts, but not dependent on strict UML rules) as reported in [33]. Therefore, these participants, who do informal

modeling, answered this question by selecting some model (diagram) types (e.g., some participants, who use “Sketch/No formal modeling language”, draw a use case diagram or sequence diagram informally). All responses are given in Figure 13.

According to the responses, sequence diagrams and state -machines/-charts are the most popular diagram types in the embedded software by analyzing their usage interval values [67]. It was a surprise that sequence diagrams were more popular than state machines/-charts, since the latter are discussed more commonly in the embedded-software-focused research venues and also in industry meetings. By an in-depth look at the data, most people use sequence diagrams informally to convey the communication among the entities in a given system.

Notice that although class diagram is relevant for object-oriented programming languages (i.e., C++ or Java) and is not used in C, which is the most used programming language according to the survey result, this diagram is in third place. In other words, where applicable (i.e., if relevant diagram for the programming language used), Class Diagram is widely used. The reason for a large usage of class diagram might be just due to the fact that it is a fundamental part of any well-formed UML diagram (i.e., if you draw a sequence diagram you need some classes to type the lifelines).



	Sequence Diagram	State Machine/ Chart	Class Diagram	Activity Diagram	Flowchart/ Diagram	Package Diagram	Diagrams based on DSL	Use Case Diagram	Deployment Diagram	Communication Diagram	Object Diagram	Any BPMN Diagram	EPC
Never (0%)	53	105	155	123	169	187	319	245	328	347	386	447	459
Sometimes (<50%)	70	82	38	168	187	227	57	244	164	154	111	21	39
Often (>=50%)	223	111	84	164	147	105	72	29	15	20	11	32	15
Most of the time (>=75%)	195	224	184	80	34	7	69	17	5	1	8	14	2
Always (100%)	5	23	81	6	2	4		3	4	1	6	1	1

Figure 13: Survey - Usage frequency and interval of different diagram types

In [46], since Agner et al. focused only on UML, the four most used UML diagrams were class, sequence, use-case and state machines, which were also reported so in [49] and [50]. Class diagrams were the most frequently used in these three surveys [46, 49, 50]. One of the most interesting result is that, although previous surveys on modeling indicates that use-case diagram usage was at one of the first places, the frequency of use case diagram usage is relatively low in our survey. Perhaps, since use-case diagram has a specific role for the analysis phase rather than design or implementation of SDLC and our pool of participants might use different types of diagrams for analysis, if needed. Moreover, use cases might not be the best way to present the requirements for an embedded system.

SDLC phases in which software modeling is used (Q18)

This multiple-response question was about SDLC phases, where software modeling is used. The majority of respondents use modeling in the "systems/software design", "implementation" and "preliminary/systems analysis (requirements)". "Integration" is the SDLC phase, in which modeling is used at least. The results are presented in Figure 14. Notice that there is no categorization on modeling approach (i.e., for sketches, model-based or model-driven) while answering this question; therefore there is no any distinction for either descriptive or prescriptive modeling. These findings are as expected since modeling (UML for example) is mainly for design and requirements phases.

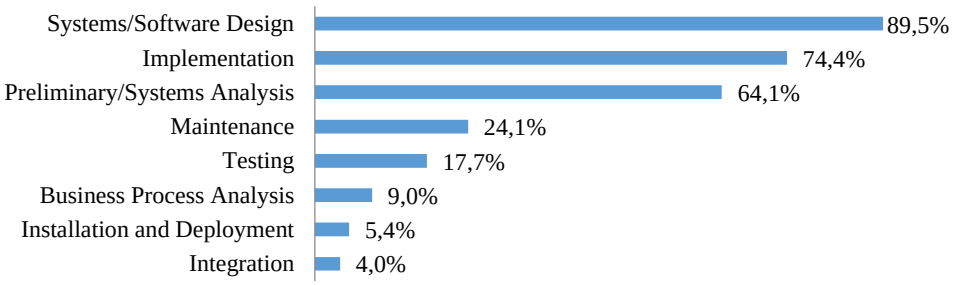


Figure 14: Survey - SDLC phases where software modeling is used

What software modeling and MDE are used for (Q20)

The reasons for software modeling and MDE usage was asked in this multiple-response question; therefore multiple purposes could be chosen. Note that this set of purposes was synthesized from the tertiary study and also related surveys (as discussed in Section 2.2 and Section 2.3). Results are shown in Figure 15. Documentation and code generation were reported to be the most popular reasons for using MDE. Notice that there is no distinction between descriptive and prescriptive models in that question (e.g., as in previous studies such as [15], remember the terminology in Figure 2). However, as indicated, the purpose of the modeling and the category of software modeling (and also the media used, the lifespan and the archivability) are strongly related (See Q13). Note that all these relations are presented while investigating the characteristics of software modeling.

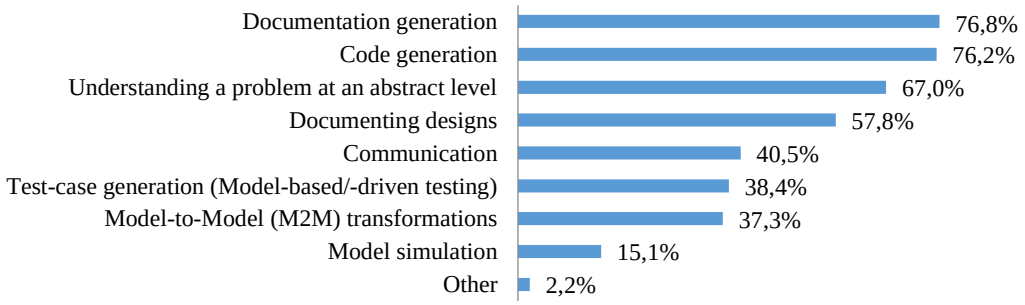


Figure 15: Survey - What software modeling and MDE are used for

In [45], communication and early simulation of the functional model were reported as the main usage reasons of MBE. According to [46], communication, understanding and documenting designs are the most important reasons of using MDE. The survey [47] reported that models are mainly used for model simulation, code generation, test-case generation and information/documentation; hence, using models for assisting activities in the SDLC seems to be an important function as also confirmed by our survey results. On the other hand, most participants in the survey in [46] reported no use of model-based automatic code and document generation. The authors in [46] argued that the lack of skilled professionals in MDE and also

the lack of powerful and user-friendly MDE tool support are the main reasons of such a situation. They also claimed that these findings differ from results of [56], which reported that activities such as code generation, transformation models, and executable models are more used in practice. We assumed that "documentation generation", "code generation" and "test-case generation" include some Model-to-Text (M2T) transformation; therefore we just gave "M2M" transformations[5] in the answer set in order to get rid of any possible duplication. By focusing on the embedded software industry, the survey reported here differs from [46]'s results since automatic artifact generation (e.g., document or code) seems to be quite popular in the embedded world for those who employ MDE.

Motivations for adopting MDE (Q23) and Achievement/Benefits of MDE (Q24)

Participants were asked about the motivations that they and/or their companies considered for adopting MDE and results are shown in Figure 16. Since using MDE provides different types of benefits for different users, the survey provided 12 motivations to be selected according to the degree of importance. As the set of purposes, this set of motivations was also synthesized from the tertiary study and also related works. According to results, cost savings and shorter development time were generally ranked of the highest importance. In [45], quality improvement, development of functions with high complexity and shorter development time were reported as the top three motivations for MDE. On the other hand, according to [47], shorter development time, reusability and quality improvements were the most three popular motivations to introduce MBE; whereas cost savings is at sixth place in popularity while adopting MBE.

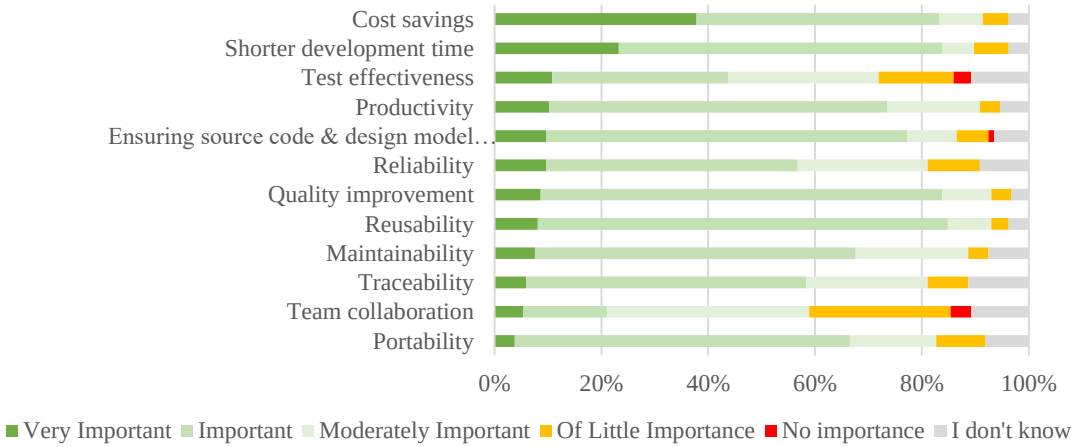


Figure 16: Survey - Motivations for adopting MDE

Following the above sets of questions, since it is important to understand the impact of the MDE, participants were asked about the degree to which their motivations were actually achieved (i.e., the degree to which their expectations were met). Note that the list of possible answers for question Q23 (motivations) is the same as for that question, where "importance" and "achievement" ranges are different. Results are shown in Figure 17.

[5] Model transformation, in MDE, is an automated way of modifying and creating models. This might be occurred as Model-to-Model (M2M), Model-to-Text (M2T) or Text-to-Model (T2M).

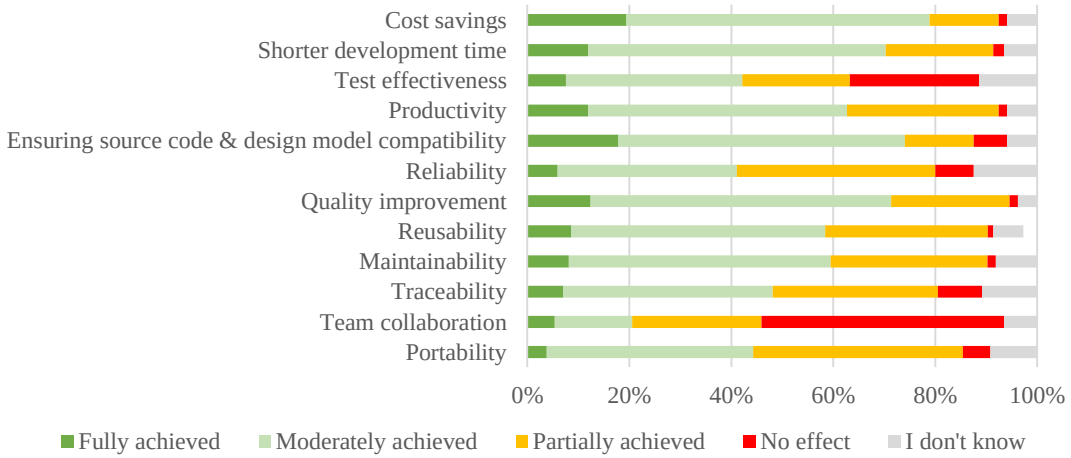


Figure 17: Survey - Achievements of MDE

According to respondents, cost savings, ensuring compatibility between source code and models, shorter development time and quality improvement are the top four achievements. Generally, all the achievements are below the importance levels, denoting that expectations are not fully met. If motivations versus achievements of MDE are depicted in a single graph to see what expected and gotten from MDE, Figure 18 is achieved.

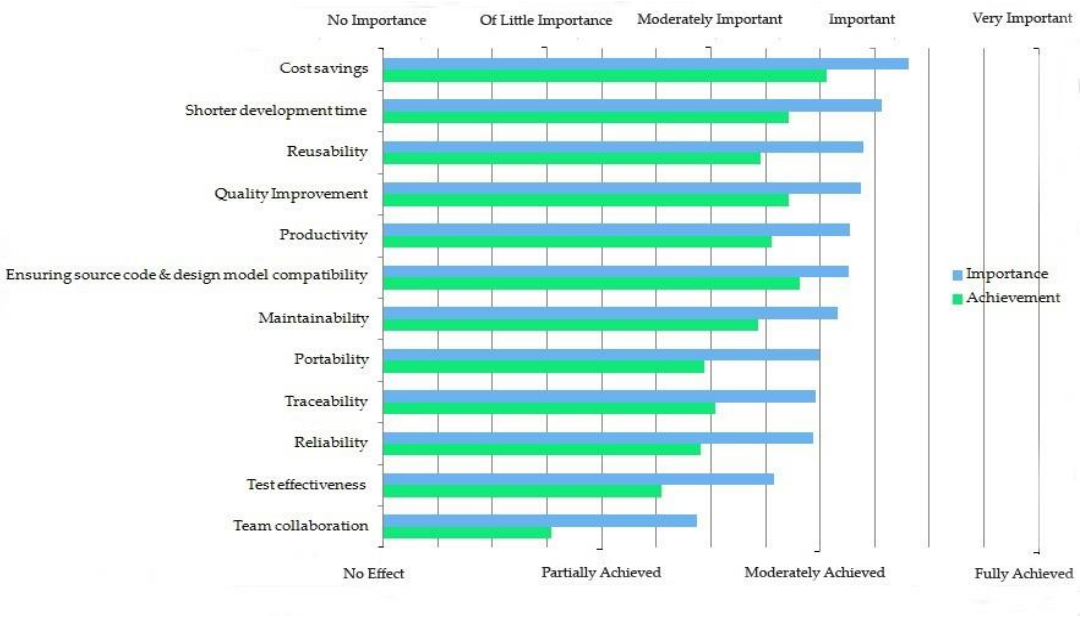


Figure 18: Survey - Motivations versus achievements of MDE

As in any engineering activity, embedded software projects should also be completed within anticipated budget (cost), within anticipated schedule (time) with conformance to requirements (quality) [69]. All individual quality factors (e.g., reusability, maintainability, portability, etc.) and shorter development time have significant effect on project budget, which is related with cost. Our participants experienced different achievement degrees on some specific quality attributes (e.g., moderately achieved reusability, but partially achieved productivity or vice versa) with a direct or an indirect effect on cost savings. Similarly, some of our participants achieved shorter development time, which also affects cost savings. In other words, although there might be some variations in the degree of achievement for quality

attributes, improvements and shorter development time; all these resulted cost savings. This viewpoint might explain why "Cost savings" is the only achievement, which is between "Fully Achieved" and "Moderately Achieved" range according to the findings.

MDE challenges (Q25)

Participants were asked about the MDE challenges in their company as multiple-response answers. According to responses, tool support and modeling expertise in the company are the most encountered challenges. All pre-given challenges (which were synthesized from the tertiary study and also related works) and "Other" answers are presented in Figure 19.

Although there was no explicit question on MDE challenges in [46], the reasons of not using UML diagrams was asked and the top three results were: "short lead-time for the software development", "lack of understanding or knowledge of UML models" and "existence of few people in the company who have deep knowledge of UML". Furthermore, according to [46], in MDE, *"the users must have access to appropriate tools, in a way that integrating a tool suite that meets requirements such as modeling, transformations, and code generation".* This supports our finding about tool support challenges in order to guarantee synchronization between software artifacts; i.e., code, document and test driver. In addition, although it is not directly related with embedded systems, the study reported in [56] pointed out the need of a longer training period in order to cope with the lack of UML expertise. According to [47], "high effort for training" and "modeling tool challenges" were also mentioned, which are similar to our findings. There was no explicit MDE-challenge question in [45], however "tool costs" and "training" were seen as a negative aspect of MDE.

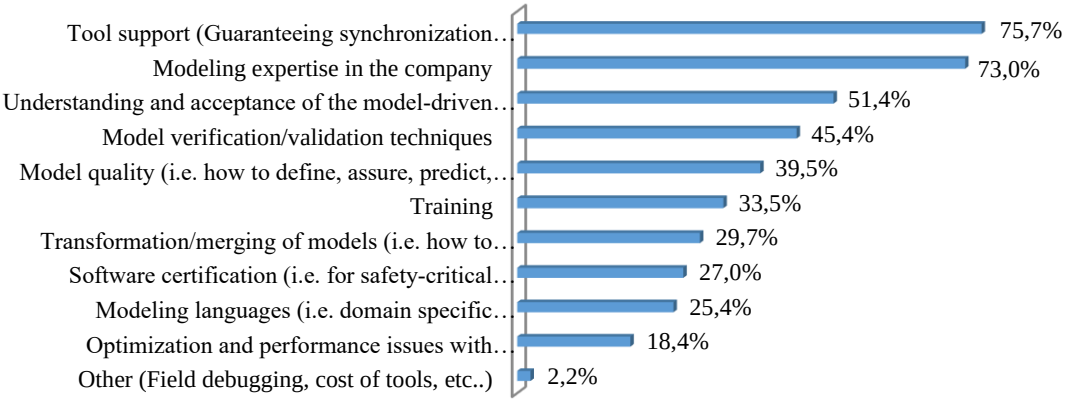


Figure 19: Survey - MDE challenges

3.3.3 *Cross-factor Analysis*

One of the opportunities the survey data provided as a further topic of study was to analyze relations among software modeling practices and practitioner demographics. To understand the effect of target sector of product(s) on the relations between the characteristics of a diagram (See Section 4.2 for this terminology), this cross-factor analysis was conducted. Please refer [70] for the details. According to the results:

- "Healthcare & Biomedical" sector is using software modeling the least (at "Sometimes" level (<50%), the other sectors is at "Often" level as seen in Figure 20. However, according to MDE usage, all sectors is at "Sometimes" level as in Figure 21:
    - "Finance & Banking" is the least model-driven user sector.
    - Although "Consumer Electronics" might be probably considered as one of the sectors where innovation and time to market drives the business, MDE usage ratio is between 9%-17%. MDE is a technique established to support these values at

most; but it might be important to analyze what and where is the problem in this sector although its software modeling usage ratio (but not MDE usage) is high (e.g., the participants in this sector use model-based or sketch/no formal modeling approaches, but what are the specific consumer electronics challenges or bad/poor experiences on MDE, which resulted such a situation?)

- "Defense & Aerospace" sector is the one, which uses MDE at most, whose MDE usage ratio is between 24%-43%. Perhaps, the project length and necessary investigation on MDE (its corresponding costs, i.e., tool, training, etc.) might be suitable for this sector.

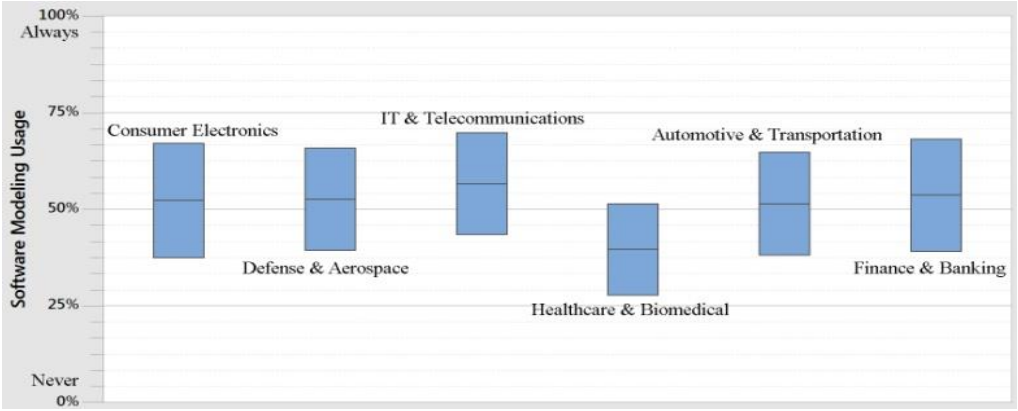


Figure 20: Survey - Software modeling usage ratio based on sectors

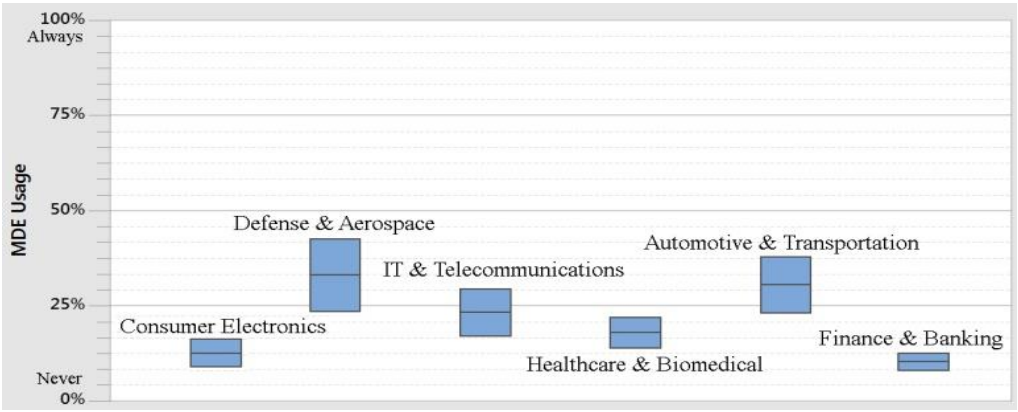


Figure 21: Survey - MDE usage ratio versus sectors

- The dominant modeling language is UML in all sectors (Table 7); however, there are interesting results based on sectors.
  - Specific modeling language for target sectors (i.e. AADL (Architecture Analysis & Design Language) for "Defense & Aerospace", EAST-ADL for "Automotive & Transportation" and Markov Chain Markup Language for "Consumer Electronics") are interesting results.
  - DSL is mostly used in "Automotive & Transportation", where AUTOSAR usage is ~15% although it was not in the pre-given answer set.
  - The usage of "Sketch/No formal modeling language" is very similar to UML usage in "Finance & Banking".

Table 7 Choices of modeling languages versus sectors

	Consumer Electronics	Defense & Aerospace	IT & Telecommunications	Healthcare & Biomedical	Automotive & Transportation	Finance & Banking
UML	84%	77%	69%	67%	71%	79%
Sketch/No formal modeling language	76%	55%	63%	57%	59%	76%
DSL	33%	36%	32%	34%	47%	40%
UML extensions (profiles)	13%	28%	17%	13%	21%	15%
SysML	10%	25%	19%	15%	12%	15%
MATLAB	7%	23%	14%	5%	15%	8%
BPML	9%	7%	7%	8%	9%	7%
SoaML	7%	6%	10%	9%	7%	9%
AUTOSAR	0%	1%	0%	0%	15%	0%
EMF	2%	1%	1%	2%	0%	2%
EAST-ADL	0%	0%	0%	0%	3%	0%
Markov Chain Markup Language	2%	0%	0%	0%	0%	0%
AADL	0%	4%	0%	0%	0%	0%
Modelica	0%	1%	1%	0%	0%	0%

- The most used diagram type according to the overall survey result (i.e., Sequence Diagram, which is left-most side) is also the most used diagram for only two sectors (i.e., "IT & Telecommunications" and "Healthcare & Biomedical); the other sectors have different most frequently used diagram types (e.g., for "Consumer Electronics" is "Flowchart/Diagram" or for "Defense & Aerospace" is "State Machine/Chart") as shown in Figure 22 *(Note that green boxes indicate the most used; whereas the red ones indicate the least used diagram types in this figure).*

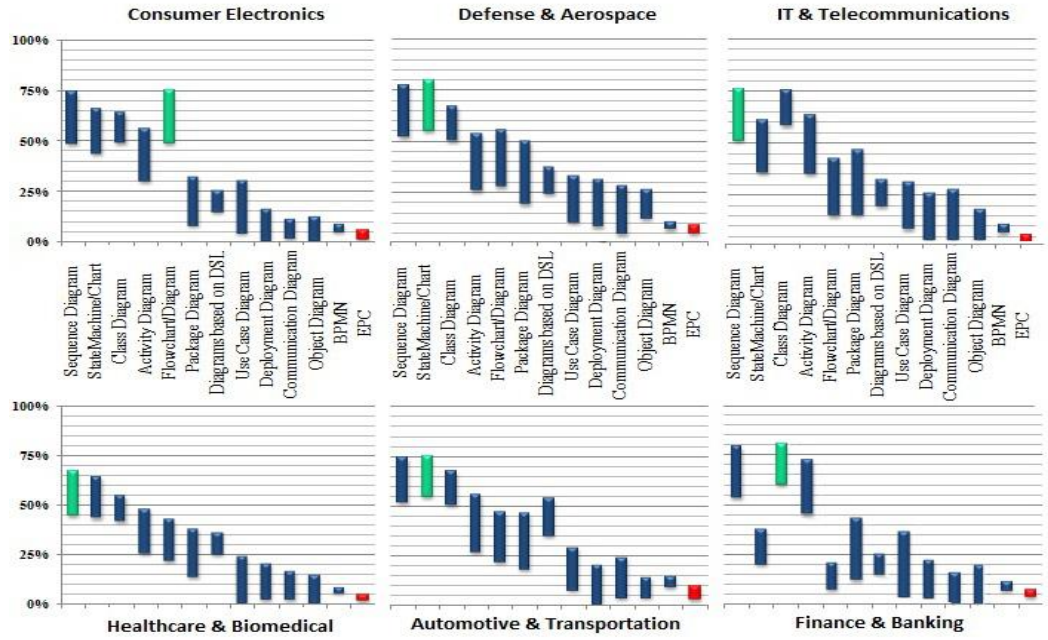


Figure 22: Survey - Diagram types usage versus sectors

The state-of-the-practice of software modeling practices in different industrial sectors was better understood with the help of this cross-factor analysis. It is interesting to see how embedded software professionals within different embedded target sectors have different software modeling usage and practices. Some modeling languages or diagrams are specific to some sectors or their usage ratio is different depending on their needs and challenges.

3.4 Summary

Note that an overall summary of RQ1.1 is also presented in Appendix B.2 – Results. The survey results have shed light on the state of software modeling and MDE practices in embedded systems and would provide practical benefits. The implications of this survey findings for practitioners, researchers and educators are presented in Appendix B.3 – Implications for Practitioners, Researchers and Educators. The limitations of this survey, based on a standard checklist [71], are presented in terms of construct, internal, external and conclusion validity concerns in Appendix B.4 – Limitations and Threats to Validity. Moreover, the steps to minimize or mitigate them are also discussed in this part.

With the help of this survey, the state-of-the-practice of software modeling and MDE were better understood by identifying to what degree, why and how it is used with its possible challenges and benefits. Notice that, all this survey data will be used in the next chapter to identify and define modeling patterns and cultures in embedded software development projects. However, survey data was insufficient to answer some qualitative questions (e.g., why they do not use MDE or what are their specific modeling challenges) and there is a need to conduct in-depth interviewing to capture some detailed, rich contextual analysis concerning the everyday practical realities of software modeling in embedded industry to better characterize modeling patterns and cultures (See Chapter 4).

CHAPTER 4

IDENTIFICATION OF MODELING PATTERNS AND CULTURES IN EMBEDDED SOFTWARE DEVELOPMENT PROJECTS

The goal of this chapter is to identify and define modeling patterns and cultures in embedded software development projects after understanding the current state-of-the art and practices of software modeling, which answers research question RQ1.1.

The survey presented in previous section showed that the embedded software professionals use modeling approaches in varying degrees (e.g., either as informal sketch or more formal model) with different constrainment and enforcement levels depending on their needs as depicted in Figure 2, "*the variable formality*" slider of modeling usage. All of the usages could be effective depending on the characteristics of modeling in embedded software industry, but what are these significant characteristics? Based on the results of the survey and the findings of Action Research [72] (AR) project's interviews[6] [5] as well as others incorporating different classifications [17, 73, 74] about software modeling, the first section systematically presents a conceptual model of development and usage for software modeling. The conceptual model, which is enriched by expert opinions (via one-to-one interviews), clarifies the meaning of ambiguous modeling terms (mainly related with modeling rigor) and ensure that different interpretations of the concepts do not occur. The second section, which addresses research question RQ1.2, investigates the significant characteristics of software modeling (i.e., the characteristics of diagram development and usage) by referencing this conceptual model. Then, the relations between these characteristics are also identified with the help of survey data analysis to complete research question RQ1.2. Notice that these correlations are also inputs to identify and define different modeling patterns and cultures of the embedded software development industry. The third section presents the identification for modeling patterns based on prior findings as a preliminary model. The fourth section, provides results on deeper and more personalized modeling experiences via semi-structured in-depth interviews on top of the results taken from the survey to improve and validate different modeling patterns. In this section, the research methodology for this case study besides its main findings are presented. During this process, hidden patterns (i.e., "unawares"), which could not be identified in the survey, are also found with direct observations or informal question & answer sessions. After identifying the final set for the patterns and cultures in the embedded software development projects in fifth section, the sixth section proposes a characterization model, MAPforES, which is also enriched by expert opinions.

---

[6] This study uses an industrial evidence to ensure the cost effectiveness and benefits of a MDE tool, which is based on AR. While reporting the impacts, challenges and lessons learned of this tool, besides presenting quantitative data, informal interview session results were also presented. Not directly related with the context of this dissertation, these interviews provided real-life MDE practices, benefits and also challenges, which are used as one of the inputs while identifying and defining modeling patterns.

4.1 Conceptual Model of Development and Usage of Software Modeling

In order to investigate the best real industrial context for modeling in embedded software industry after the surprising results of the survey (e.g., “Sketch/No formal modeling” is the second most frequently reported response), a conceptual model on software modeling usage was created. Since it was crucial to have a complete & correct conceptual model for the rest of the study, it was also important to take feedback and suggestions on this model from experienced embedded software professionals. As a qualitative approach, to take feedback and suggestions about the conceptual model, one-to-one interviews in different sectors were conducted over four months with 20 embedded software professionals, whose total work experience is 358 years. During the planning phase, it was necessary to decide from whom to take feedback. Since it is recommended to select these professionals based on differences instead of similarities [38], it is good to try to involve different industrial sectors, different roles, different experiences and different practices in embedded software. These one-to-one interviews were conducted mostly in face-to-face meetings, but if it was inconvenient, on Skype as in the case of intercontinental interviews (i.e., USA and Taiwanian Companies’ interview were conducted via Skype; all other were face-to-face) [75].

Given their feedbacks and suggestions, the model was refined and updated for full industrial coverage. In this way, the conceptual model used information was obtained from the survey results, AR Project, similar related works [20, 45-47, 58, 73, 76] and finally feedbacks during one-to-one interviews with expert opinion strategy. The conceptual model is given in Figure 23. Note that this conceptual model is also descriptive diagram (i.e., for understanding and communication), in which there are some UML elements (e.g., some class diagram or use case diagram elements as inheritance or actors, but selectively and informally as in [33]).

The model is decomposed into five conceptual areas, where "Diagram", which is developed and used in different SDLC phases, is the backbone of this conceptual model. According to the conceptual model, there are "*Influencing Factors*" (e.g., "Purpose", "Stakeholder Profile", "Target Sector" and “Programming Language”), which affect software modeling usage hence modeling rigor. These factors are derived from the survey results (e.g., for purpose Q20, for stakeholder profile Q3, Q4, Q5, Q9, Q11, Q12, for target sector Q7, for programming language Q15, etc.). In order to understand and easily follow the model, these five areas are explained next.

- Area 1 in Figure 23: Modeling rigor and modeling categories *(On the upper middle part)*

The “Diagram”, which is the backbone of the model, has "*Modeling Rigor*", which is either "*Informal"* (i.e., sketch) or *"Formalized"* (i.e., model) modeling category. The survey results showed that this formality affects the usage of modeling in varying degrees (i.e., Q10 and Q19).Therefore, the terminology used in this study plays a critical role (i.e., descriptive modeling versus prescriptive modeling) since “*the variable formality*” slider of modeling usage (i.e., modeling rigor) depends on these categories of software modeling.

The “Diagram” has "*Code Correspondence*", which shows the compatibility between design model and source code. Our survey results showed that ensuring source code & design model compatibility is one of the most reported benefits of MDE (i.e., Q23 and Q24), which can be achieved by maximum code correspondence (i.e., Q27). As mentioned in [76], the rigor and styles of modeling affect the model-code correspondence; “*the higher the similarity between models and the code, the higher the correspondence is*”.

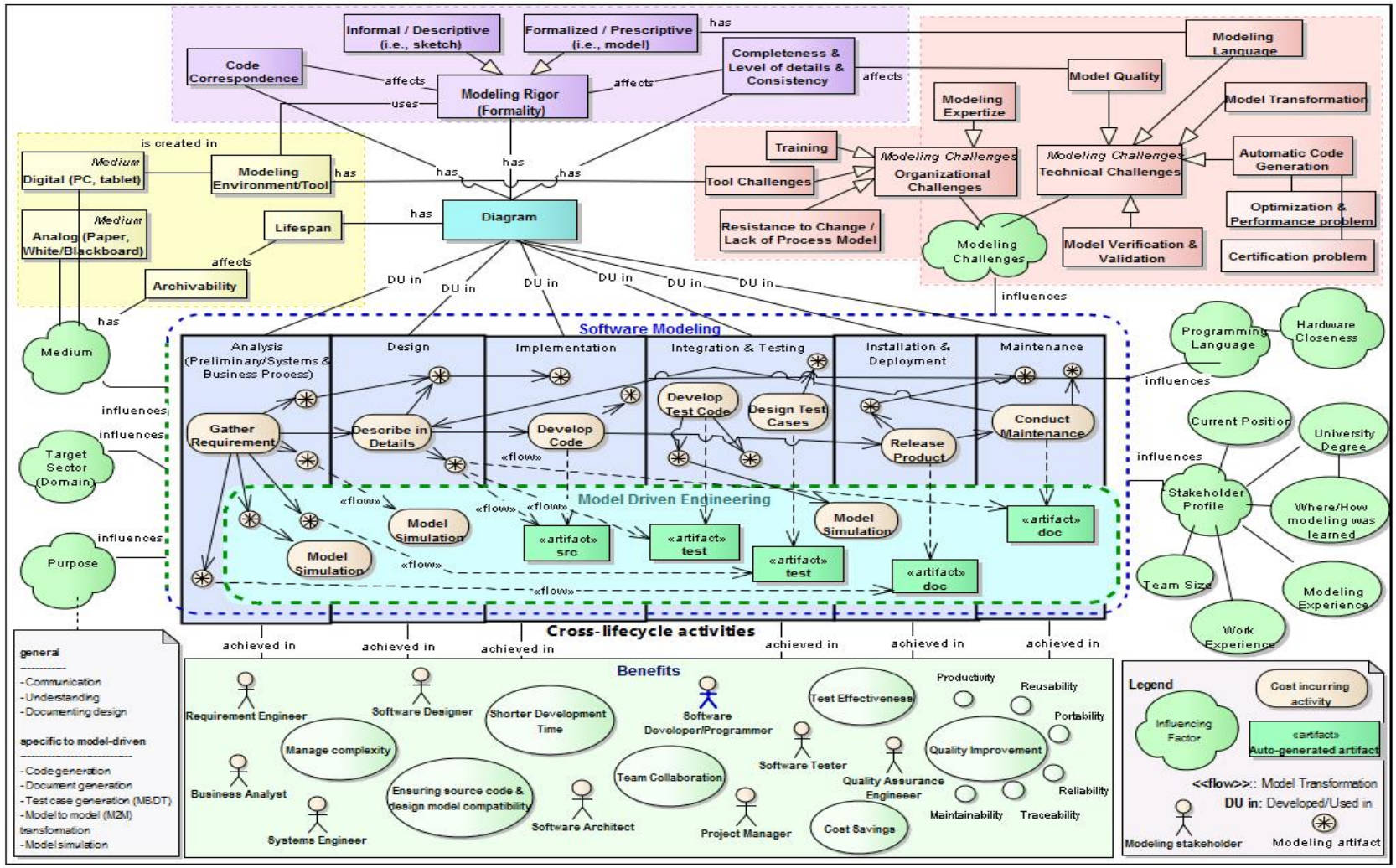


Figure 23: Conceptual model of development and usage for modeling

Moreover, our AR Project [5] showed that whenever the code is synchronized with the other artifacts (i.e., test driver and documentation), which means that the correspondence is high (e.g., as in MDE), the benefits of software modeling is fully achieved. These indicators showed that the correspondence affects modeling rigor (e.g., more rigor guarantees more correspondence). As reported in [76], "*Completeness & Level of details & Consistency*" of diagram, which affects other modeling entities, is also affected by modeling rigor (i.e., in sketch, there is an abstract and high level modeling approach as depicted in Figure 2).

- Area 2 in Figure 23: Benefits for modeling stakeholders (*On the lower part*)

Since using software modeling provides different types of *"Benefits*" for different modeling stakeholders (e.g., "Software Developer/Programmer" or "Software Tester"), who has different purposes (e.g., either for general or specific to model-driven), our survey provided pre-given motivation set to be selected according to the degree of importance. All the benefits in the conceptual model, which are achieved in different SDLC phases, received some responses (i.e., at least one participant chose it) in our survey; however, according to respondents, "*Cost Savings*", "*Ensuring source code & design model compatibility*", "*Shorter Development Time*" and *"Quality Improvement*" are the top four achievements. When we analyzed the related researches in embedded software development, the most significant benefits in [46] were associated with *"Quality Improvement", "Portability", "Maintainability"* and *"Productivity"*. On the other hand, according to [47], the effect of introducing MBE are "*Reusability*", *"Reliability", "Traceability", "Maintainability"* and "*Shorter development time"*, respectively (according to highly positive answers). Note that modeling stakeholders depicted in the conceptual model are derived from survey's demographics of participants.

- Area 3 in Figure 23: Medium type used and its effects *(In the left upper part)*

As the survey (Q13) investigated, different stakeholders use different media to create (draw) models. Different diagrams, which might have different purposes, are drawn on a different "*Medium*", which is either "*Digital*" (e.g., PC or tablet) or "*Analog*" (e.g., paper or whiteboard). The results showed that using modeling software on PCs for modeling is the most used medium; whereas modeling using pen and paper is the next common approach.

The semi-structured interviews and also related works (e.g., [73]) showed that the medium type has a "*Archivability*" and this directly affects the "*Lifespan*" of this diagram. As reported in [73], sketches created on analog media had an estimated lifespan of several work days, whereas sketches created digitally had an estimated lifespan of several months.

Different from analog media, digital media is created in a "*Modeling Environment/Tool*". Our survey results showed that a variety of modeling tools are used by embedded software professionals from different SE roles with different motivations and challenges (i.e., Q16). This also showed that "influencing factors" (e.g., stakeholder's profile, their purposes, tool challenge etc.) affects modeling tool choice; hence modeling rigor.

- Area 4 in Figure 23: Modeling Challenges *(In the right upper part)*

The survey results showed that there are different *"Organizational"* and *"Technical"* challenges while modeling [67]. Participants were asked about the modeling challenges (i.e., Q25) in their company as multiple-response answers. All modeling entities related to modeling challenges in the conceptual model received some responses [67]; however, *"Tool Challenges", "Modeling Expertize"* in the company and *"Resistance to Change*" are the most encountered challenges. Related works also mentioned such challenges. In [45], "Tool costs" and *"Training"* were seen as a negative aspect of MDE. In [46], *the existence of few people in the company who have deep knowledge of UML* (which maps to "modeling expertize") and appropriate modeling tools were the reasons of not using UML diagrams. In addition, although it is not directly related with embedded systems, the study in [20] pointed out the need of a longer training period to cope with the lack of UML expertise, which is also in parallel with the

"*Modeling Expertize*" challenge in our survey. According to [47], "high effort for training" and "tool challenges" were also mentioned. As seen, these top challenges are mainly organizational; however there are also technical modeling challenges, which are due to the nature of modeling (e.g., "*Modeling Language*" itself (e.g., DSML needs), "*Model Transformation*", or "*Model Verification & Validation"*). While investigating the other technical challenges, during semi-structured interviews, we also investigated that "*Model Quality*" is affected by a diagram's "*Completeness & Level of details & Consistency*" characteristic, which directly depends on modeling rigor. Moreover, as our survey (Q25) and the AR interviews showed that embedded software professionals suffered from "*Optimization and Performance problem*" besides "*Certification problem*" (e.g., for safety-critical software) with *"Automatic Code Generation"*[7] challenge.

- Area 5 in Figure 23: Cross lifecycle activities during modeling

During cross-lifecycle activities (i.e., Q18, where SDLC phases in which software modeling is used was asked), there are "*Cost incurring activities*", which are related with purpose, hence modeling rigor (e.g., for communication or understanding, *"Gather Requirement"* is valid for descriptive modeling as a sketch; however, *"Develop Test Code"* for test case generation is only valid for prescriptive model-driven usage). These activities create "*Modeling artifacts"*, which might be also an "*Auto-generated artifact*" (e.g., "*src*", "*test*", "*doc*") as in the case of MDE via "*Model Transformation*" flow. The critical question is that depending on the modeling purpose, whether this modeling cost is affordable or not with respect to its potential benefits. Therefore, it is important to find out the optimal degree of modeling rigor for a cost-effective approach. At that point, the characteristics of software modeling and their relations between each other plays a crucial role to find the best solution.

4.2 Characteristics of Diagram Development and Usage

With the help of the conceptual model, the characteristics of modeling based on diagram development and usage in embedded software development were identified. Accordingly, there are 11 main characteristics, where some sub-characteristics affect its main characteristic as seen in Figure 24. Based on previous results, the relations between these characteristics are also presented in this section.

As survey results showed that RIGOR (i.e., modeling rigor) is affected by all other characteristics (i.e., Q10 and Q19 results are correlated with these characteristics), therefore it is crucial to analyze other characteristics based on this. In other words, PURPS (i.e., the purpose of modeling), CORRS (i.e., the correspondence/compatibility between design model and source code), COST (i.e., cost of modeling), STAKH (i.e., stakeholder profile), SDLC (i.e., SDLC phases where modeling is used), BENFT (i.e., the benefits of modeling), CHALL (i.e., the challenges of modeling), PL (i.e., programming language used), DOMN (i.e., embedded target sector of the company, where stakeholder works) and MEDM (i.e., the media used while modeling) have a correlation with modeling formality. These characteristics – somehow- influence RIGOR based on "*the variable formality*" slider, which explains the

[7] In embedded software development, although automatic code generation has benefits to manage the embedded systems' challenges by decreasing accidental complexities, some embedded software professionals claimed that it introduces new challenges like performance problems and certification problems. For example, in AR project interviews, some embedded engineers reported that due to automatic code generation, they could not get certification from DO-178B/C standards for their safety-critical airborne systems. These developers also thought that with automatic code generation, they could not guarantee the optimization and the performance of the software.

difference and the notions between descriptive (e.g., sketch) and prescriptive (e.g., model-based or model-driven) modeling.

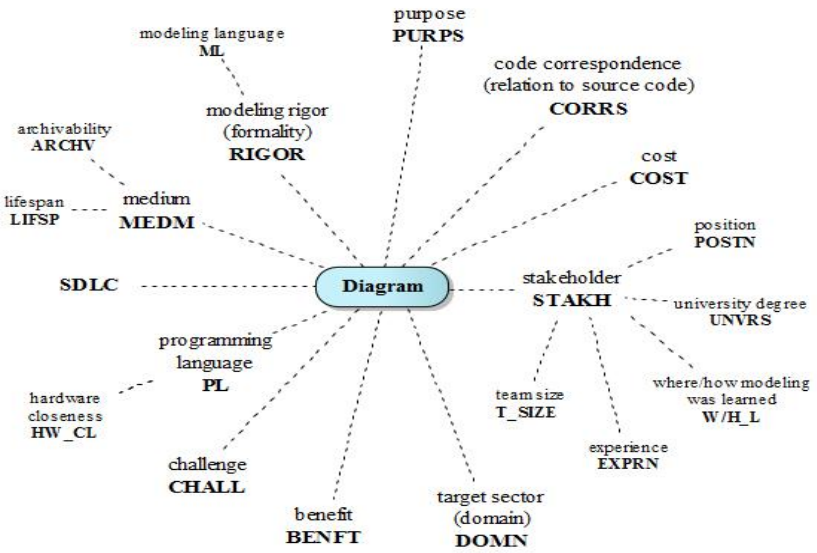


Figure 24: Characteristics of diagram development and usage while modeling

The chart in Figure 25 explains the relations of these characteristics based on modeling approach, hence RIGOR (i.e., first column for sketch, second column for model-based and third column for model-driven, if there is). RIGOR has a degree between 0% and 100% on "*the variable formality*" slider. According to this column-based category, all common and different characteristics of modeling usage (either descriptive or prescriptive) are mapped, if applicable (e.g., there is no PL or DOMN on this chart since it is not easy to put these characteristics on this column-based category).

For PURPS, which is depicted as an influencing factor in the conceptual model, "Communication, Understanding and Documenting design" are all common purposes; whereas "Code generation, Document generation, Test case generation (MB/DT), Model transformation and Model simulation" are specific to the model-driven usage. (Note that in the chart, "Communication" is closer to descriptive; whereas "Documenting design" is closer to prescriptive modeling; but both of them are valid for all three categories).

For CORRS, whenever your modeling rigor is high, your source code and design model compatibility is high. (e.g., your model-driven code correspondence is higher than to the ones in the sketch). This code correspondence check is achieved by manual review, reverse engineering or roundtrip depending on your modeling approach [76].

PURPS of the modeling and the category of software modeling are strongly related with the MEDM used, which was also derived from Q13 of the survey. In that sense, digital media are usable for all main modeling approaches (e.g., you can use PC for all three columns); however analog media can be used for only sketch and model-based (e.g., for the first two columns). The survey cross-factor analysis of medium type data with modeling languages showed that the participants, who do not use any formal modeling (i.e., the ones who draw sketches), use just paper or whiteboard. On the other hand, the participants, who use any formal modeling language (e.g., UML), usually use modeling tools on PCs besides using paper also.

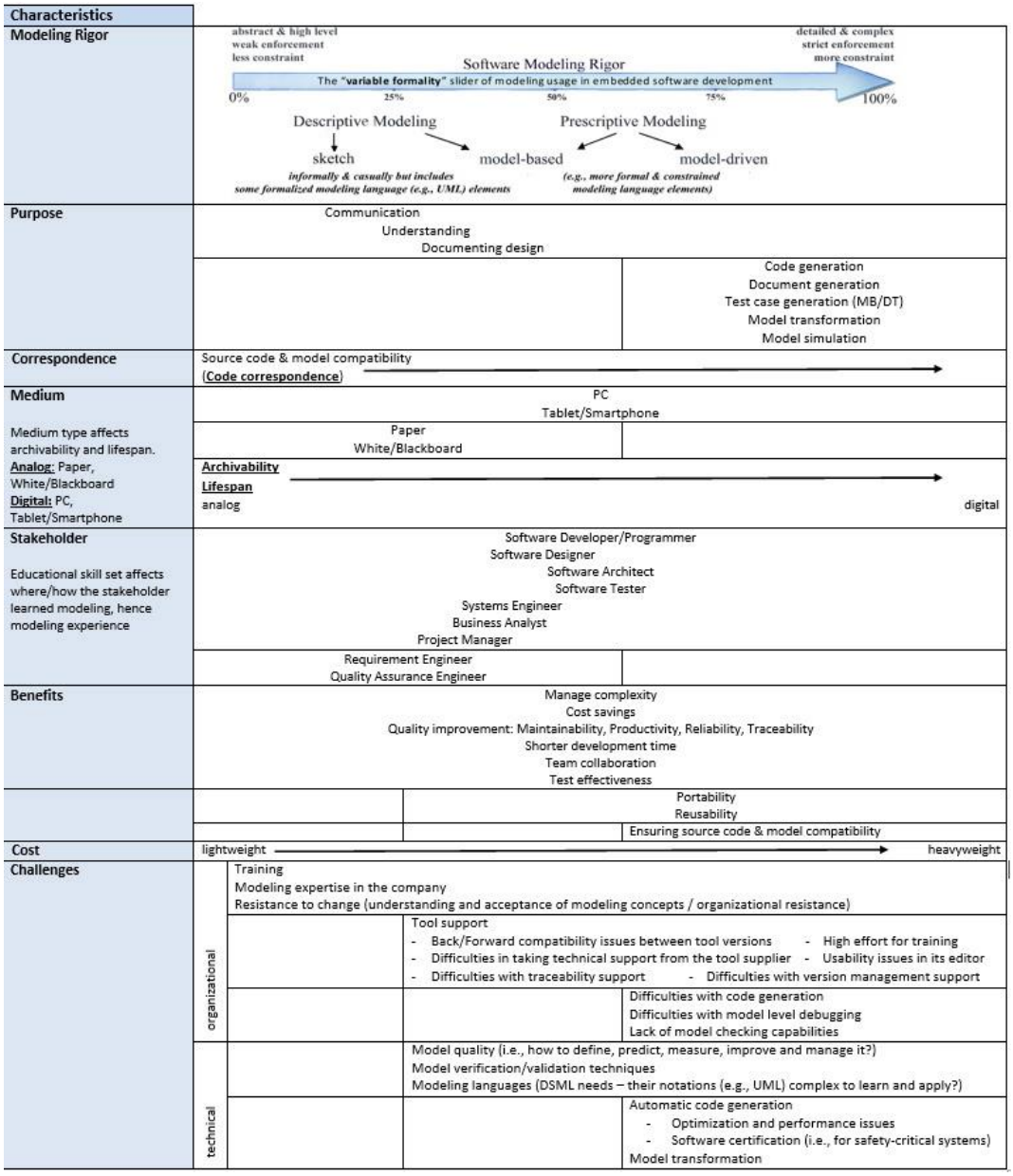


Figure 25: Chart showing the relations between characteristics of a diagram

MEDM type affects ARCHV and LIFSP. The lifespan of the sketches or model-based diagrams created on analog media are less than the ones created digitally via PC or tablet/smartphone. Therefore, the digital mediums like PC or tablet/smartphone have advantageous on archiving and have longer lifespan (Q13). During the semi-structured interviews, it was also observed some transitions from one medium to another to achieve more ARCHV and hence LIFSP. For example, some analog models (e.g., either in paper or in white/blackboard) are archived by saving a digital picture or by redrawing them digitally. In that sense, archived models are more formal; hence more RIGOR.

Descriptive modeling is lightweight and has low cost since it may benefit from lack of precision (e.g., no extra cost for a modeling tool/environment as in prescriptive modeling). However, prescriptive modeling is heavyweight and requires more precision. Therefore, COST increases whenever you have more RIGOR on "*the variable formality*" slider. (e.g., it is important to balance the cost according to your purpose and you do not need to use an

expensive modeling tool if your purpose is just selective communication, which might be modeled with pen and paper). The survey showed that model-driven users have specific MDE problems, which increases modeling costs (Q25 and Q26).

Modeling STAKH profile strongly affects modeling usage and the characteristics of a diagram. Note that SE roles in Figure 23 are identified by the survey, which has a wide range of embedded professionals including from developer to tester and project manager to quality assurance engineer. Accordingly, the survey results (the correlation between current position and modeling category; i.e., Q4 and Q19) showed that except some roles (i.e., requirement engineer and quality assurance engineer), all given stakeholders might selectively use all modeling types (i.e., sketch, model-based, model-driven). It was also observed during semi-structured interviews that educational skill set affects where/how the stakeholder learned software modeling, hence modeling experience. For example, user, who graduated from Electrical/Electronics Engineering (EE), have learned software modeling after getting the job (after graduation, on his/her own or with formal corporate training); however any stakeholder who graduated from a Computing Discipline (e.g., Computer Science (CS), Computer Engineering (CENG), Software Engineering (SE), and Information Systems (IS)) has learned software modeling at the university (i.e., from SE courses). Therefore, there is a distinction between work and modeling experience of STAKH (See Figure 6) and this affects the degree of modeling and its relevant practices. Moreover, team size of the stakeholder (i.e., Q9) also affects modeling practices with respect to PURPS and MEDM (e.g., for large team, communication is very important to get the same understanding on a problem in the early SDLC phases).

BENFT and CHALL are also mappable to this column-based chart. Software modeling category (i.e., sketch, model-based, model-driven approaches) has common BENFT (e.g., "Managing complexities", "Cost savings", "Team collaboration"). However, "Portability" and "Reusability" are achieved mainly in prescriptive modeling (e.g., model-based and model-driven). On the other hand, since there is an automatic generation of artifact (e.g., code), "Ensuring source code and model compatibility" is only achieved in model-driven approach. This is also strongly related with CORRS, which affects RIGOR. On the other hand, as our conceptual model revealed that there are mainly two modeling CHALL: organizational and technical. These challenges – based on RIGOR – might increase COST. (e.g., if RIGOR on "*the variable formality*" slider is low (i.e., sketch), you do not need to concern about difficulties/costs with code generation.)

One of the opportunities the survey data provided as a further study was to analyze relations among software modeling practices and the target sector of the products (i.e., DOMN) as a cross-factor analysis (See Section 3.3.3). The results of this cross-factor analysis of software modeling practices versus DOMN showed that software modeling usage degree (i.e., RIGOR) varies among embedded sectors.

Depending on PURPS, SDLC phases, where software modeling is used are affected. For example, if PURPS requires only descriptive modeling (e.g., communication or understanding) "preliminary/systems analysis" might be sufficient; however if there is code generation, perhaps all SDLC phases use software modeling (e.g., use case diagram in analysis phase, sequence diagrams in design phase, state machine and class diagrams in implementation phase)

PL choice affects the diagram type used while modeling. The survey consisted two questions on both programming languages (Q15) and diagram types used while modeling (Q17). The C language is the first, followed by C++ and then Java. Notice that, although C is the most popular programming language in the embedded world, the total responses for C++ and Java combined, which are object-oriented programming languages are much more than C. Notice that although class diagram is only relevant for object-oriented programming languages (e.g., C++ or Java)

and is not used in C, which is the most used PL, this diagram is in third place. In other words, where applicable (i.e., if relevant diagram for the programming language used), Class Diagram is widely used. The reason for a large usage of class diagram might be just due to the fact that it is a fundamental part of any well-formed UML diagram (i.e., if you draw a sequence diagram you need some classes to type the lifelines), which is directly related to RIGOR. Furthermore, it was also observed in survey results that STAKH position affects PL for a specific PURPS on "*the variable formality*" slider during SDLC. For example, some systems engineers use MATLAB for model simulation purpose in systems analysis and design phases of SDLC. Moreover, by an in-depth look at the data, the respondents, who state that they were doing informal modeling, make the sketches, which include some essences of UML (i.e., some elements of state machine/charts, but not dependent on strict UML rules) as stated in [33]. Therefore, these participants, who do informal modeling, answered Q17 by selecting some diagram types (i.e., some participants, who use "Sketch/No formal modeling language", draw a use case diagram or sequence diagram informally). Similar cases were also observed during interviews that most people use sequence diagrams informally to convey the communication among the entities in a given system.

During the interviews, it was also observed that software's closeness to hardware affects RIGOR and its corresponding modeling practices (e.g., modeling languages, diagram types, etc.) via PL selection What meant by HW_CL (i.e., hardware closeness) is that firmware or digital signal processing (DSP) software is closer to hardware than User Interface (UI) or middleware software. This characteristic indirectly affects RIGOR, but the real industrial context (via semi-structured interviews and our industrial experience) showed us that whenever the software is close to hardware, the PL selection is critical. As AR project [5] showed that even in the same project, DSP team uses a PL (i.e., C), middleware team uses a different PL (i.e., C++) and UI team uses another PL (i.e., Java); and their modeling practices are different.

With the help of this section, which addresses research question RQ1.2, the characteristics of diagram development and usage in embedded software development was better understood in the best real industrial context. By this way, one of the inputs to investigate modeling patterns and cultures of the embedded software development industry has been obtained by understanding the relations between these characteristics.

4.3 Pre-investigated[8] Modeling Patterns

After investigating RQ1.1 and RQ1.2, rough clustered groups and classification on survey data is very important to identify the possible modeling patterns.

As reported, 11 main characteristics were investigated and there would be lots of logical combinations while grouping these characteristics on survey results data, which includes ~80 attributes/features [77]. Therefore, it is crucial to reduce the complexity by selecting the right subset with the accuracy. As a preprocessing step, by using all our previous results and observations on modeling usage patterns, the number of attributes/features in dataset were reduced via a process, which is similar to any feature selection filter methods [78]. The correlation here was based on Section 4.2 results. By this way, by eliminating the possible dependent features, the most critical ones would be determined as most relevant starting points

[8] These modeling patterns are derived from quantitative survey data. However, there might have been would be other patterns, which might be found out with more qualitative strategies (e.g., in-depth interviewing, which will be presented next section). Therefore, at that stage, they were pre-investigated.

instead of trying all combinations (e.g., heuristically, modeling purpose would be more potential candidate than modeling challenge since the latter might be the result of the former).

As survey results showed that *rigor*, hence *modeling language* is affected by all other characteristics. Furthermore, while investigating the relations between these characteristics, it was found that purpose, which is one of the most important influencing factor to determine modeling usage category, is strongly related with the medium type used while modeling; hence *purpose* and *medium type* combinations would also be good candidates as a starting point. Notice that this process is based on both our previous results and also experience.

Another problem is that, in some cases, one characteristic might include more than one survey data item (attribute/feature) since it becomes more meaningful with the combinations of these attributes. Therefore, it is also crucial to eliminate the unnecessary combinations by creating a derived attribute on survey data while trying out the possible alternatives. For example, for *medium type*, there are four medium type choices in the survey data (i.e., PC, paper, white/blackboard, tablet/smartphone) with 5-point Likert-scale (i.e., from never (0%) to always (100%); which leads us to derive "medium type(s) set" on these data. By this way, the survey data includes a new single derived data, in which related survey responses are logically grouped in this set. For example, if the data includes "*The participant's PC and tablet/smartphone usage is never while modeling*" (i.e., both PC usage and Table/Smartphone usage is Never (0%)), this means that the medium type(s) set is "only analog media usage". Similarly, based on the other 5-point Likert-scale attributes depending on the medium type (i.e., either digital or analog), the other two items are derived for the medium type(s) set: "analog media usage is equal or greater than digital media usage" and "digital media usage is greater than analog media usage".

Since *purpose* and *modeling language* are also multiple-response questions, similar derived attributes set were generated by applying the same technique on the necessary characteristics (e.g., purpose(s) set, modeling language(s) set, etc.). As reported in Figure 25, which shows the relations between the characteristics of software modeling, the modeling purposes might be grouped whether it includes any model-driven purpose or not. Further grouping would be done among model-driven and no-model-driven purposes. In model-driven purposes, "Code generation" and "Test case generation (MB/DT)" are significant since they are directly related with implementation and testing phases of SDLC (remember SDLC phases for descriptive modeling vs prescriptive modeling); hence "model driven purpose with code generation or MBT" and the remaining model driven purposes (i.e., document generation, model simulation and model transformation) are the two sub-groups. On the other hand, in no-model-driven purposes, "Documenting design" is significant since it might affect other characteristics (e.g., medium type used and hence archivability). Therefore, "no model-driven purpose with documenting design" and "without documenting design" are the other two sub-groups. By this way, a new derived attribute on survey data, "purpose(s) set" includes four choices: model-driven with code generation or MBT, model-driven without code generation or MBT, no model-driven with documenting design and no model-driven without documenting design.

Similarly, besides being a multiple-response question, *modeling language* might include "free-text" area in survey data. Therefore, it is also important to get appropriate subsets in this characteristics. Mainly in this response, the participants reported that any combinations of sketch/no formal modeling, UML and DSL-like languages (e.g., any DSL/DSML or UML profiles). Hence, this new derived set, "modeling language(s) set" includes seven such combinations (i.e., 2^3-1). By this way, finding out the clustering groups would be easier.

In short, to eliminate unnecessary combinations, derived attributes on survey data are generated after determining the most critical characteristics as starting points. During this

process, different alternatives for the combination of software modeling characteristics were tried by using RapidMiner Studio [79] and Excel on survey results (e.g., with scatter and bars stacked charts) to get these critical characteristics. After visualizing these groups in RapidMiner, it was seen that further analysis would be done on *SDLC phase* to identify possible model-based patterns (i.e., ~38,5% of the survey data) since there is a difference between the phases of SDLC where modeling is used. This SDLC phase difference is again based on the existence of "Implementation" or "Testing" phases to understand descriptive versus prescriptive usage. See APPENDIX C – Pre-investigated modeling patterns' visualizations for the details of this process.

After the analysis and visualization on these groups, the necessary characteristics, which have critical importance on the categorization are derived: *"purpose", "medium type", "archivability", "modeling language, if any"* and *"SDLC phase".* Accordingly, nine modeling patterns were pre-investigated as in Table 8, in which the percentage of these patterns in the survey results are also given (Note that "model-based" and "sketching" category were in the same group in the survey, see Appendix B.1 – Survey design and execution).

Table 8 Modeling patterns investigated after survey data analysis

<table>
<tr><th>Main pattern</th><th colspan="4">Patterns pre-investigated</th><th colspan="2">% in survey results</th></tr>
<tr><td rowspan="3">model-driven</td><td>3.3</td><td>With DSL-like[9]</td><td rowspan="2">Purpose set includes "Code generation" or "Test case generation (MB/DT)"</td><td>With "any DSL/DSML or UML profiles"</td><td>16,9</td><td rowspan="3">29,5</td></tr>
<tr><td>3.2</td><td>Without DSL-like</td><td>Without "any DSL/DSML or UML profiles"</td><td>6,5</td></tr>
<tr><td>3.1</td><td>Limited</td><td colspan="2">Only with "Document generation" or "Model simulation" or "Model transformation" purpose</td><td>6</td></tr>
<tr><td rowspan="2">model-based</td><td>2.2</td><td>Prescriptive</td><td colspan="2">SDLC set includes "implementation" or "testing"</td><td>24,9</td><td rowspan="4">59,5</td></tr>
<tr><td>2.1</td><td>Descriptive</td><td colspan="2">SDLC set does not include "implementation or testing"</td><td>13,7</td></tr>
<tr><td rowspan="3">sketching</td><td>1.3</td><td>Archived</td><td colspan="2">Purpose sets includes "Documenting Design"
Analog media usage >= Digital media usage</td><td>3,6</td></tr>
<tr><td>1.2</td><td>Selective</td><td colspan="2">Casually & informally with some formalized modeling language (most probably, UML elements) (UML-like sketching)
Modeling Language set includes sketch & any formalized modeling language (e.g., UML &| DSL, BPML, etc.)</td><td>13,1</td></tr>
<tr><td>1.1</td><td>Ad-hoc</td><td colspan="2">Purpose sets includes only "Understanding" or "Communication"
Only pen & paper / free format
(e.g., without any formalized modeling language, e.g., UML, elements)
Medium type is analog (paper or whiteboard)
Modeling Language set includes only "no formal modeling/sketch"</td><td>4,1</td></tr>
<tr><td>none</td><td>0</td><td>No modeling</td><td colspan="2">Not using any modeling approach.</td><td colspan="2">11</td></tr>
</table>

[9] "With DSL-like" means that the modeling language set of the stakeholder includes any DSL-like language (e.g., any DSL (provided by tool provider or their own design), any UML profiles such as MARTE, SysML, SoaML, any BPML, MATLAB Modeling Utilities, AUTOSAR, EAST-ADL, AADL, etc.)

Since there might be some hidden patterns[10], which could not be found out from the analysis of survey data, there is a need to validate these pre-investigated modeling patterns with deeper and more qualitative strategy via in-depth interviewing. The next section addresses this issue and validated the investigation of modeling patterns in embedded software development projects.

4.4 Case Study to Validate Modeling Patterns via Interviews

In order to find out possible hidden patterns on pre-investigated pattern set, this section presents case study to validate and improve modeling patterns, which were investigated after the analysis of survey data.

4.4.1 *Research Methodology*

In order to deal with topics in detail, data collection through interviews is one of the most frequently used sources of evidence [38]. In that sense, after getting survey results, various interview sessions were conducted to get detailed opinions and experiences of software modeling in embedded domain. As a part of long interviewing session, the empirical study reported here included a series of semi-structured interviews [38, 59, 80], which were conducted over eight months with 53 embedded software professionals across a variety of target industrial sectors and roles to validate and improve our pre-findings on modeling patterns.

The main goals of this study are designed as specific as possible with the corresponding RQs:

RQ1.2_CaseStudyRQ1: What is main software modeling usage pattern of the interviewee (i.e., no modeling, sketch, model-based or model-driven)?

RQ1.2_CaseStudyRQ2: What is the current state of software development techniques/approaches of the interviewee (e.g., programming, modeling (if any), etc.) based on her/his main modeling usage pattern? In other words, what are the characteristics of a diagram development and usage?

RQ3.1_CaseStudyRQ3: Does the modeling pattern of the participant belong to the pre-investigated pattern set? If not (i.e. hidden pattern), what are the main characteristics?

Each of the above RQs, which are cross-cutting with survey and complementing each other, is used to derive "interview questions", in which some questions were taken from the survey (i.e., demographics) and some of them were improvised and detailed during the interviews.

4.4.2 *Interview Design and Execution*

As a general rule during its design phase [38], different industrial sectors, roles, experiences and practices in embedded software industry were involved in the interview. The semi-structured interviews in this section were conducted mostly in face-to-face meetings, but if it is inconvenient, on Skype as in the case of intercontinental interviews. All interviewees were promised that only anonymous data (see Table 9) would be presented and the interviewer

[10] In this research, "hidden patterns" are the groups, which do not know exactly their software modeling characteristics (especially their main modeling pattern and modeling rigor); hence their modeling patterns could not be identified by only structured quantitative data (e.g., only with survey data analysis).

would take notes on what he spontaneously found relevant and to be later transcribed for analysis.

During the interview, there was a clear and complete list of general topics (i.e., interview instrument), which cover about both personal and companies' software modeling usage patterns, purposes, motivations and challenges besides their success and failure stories, the attitudes of people to the adoption of modeling and so on. However, the order of the questions was not fixed and it was necessary to let the interviewee develop ideas and speak more in detail with open-ended answers [80]. Therefore, the interviewee was encouraged to provide more detail and rich information by changing the order of questions and the length of time devoted to each question. In that sense, a 'timeglass' structure [38] "*with an open introduction with protocol, more specific questions in the middle, and ending with very open questions"* was followed.

The interviews lasted approximately more than ~1 hour and the protocol was straightforward, presenting the objectives of the interview and explaining how the data would be used. Then a set of questions about demographics were asked (The demographics questions are exactly the same as in our survey [67]). After getting demographics data, the key/critical question was "*How often do you use software modeling in your software development life cycle? (either informal and/or formalized: i.e., sketches and/or models)*". The goal of this question was to categorize the interviewee according to main modeling usage pattern. Depending on the response, if the answer is "Never", which means that the main pattern (i.e., category) is "no modeling", the interviewee was asked a series of questions why they don't use any software modeling during their SDLC for the purpose of investigating their development practices besides the reasons of not modeling. Otherwise, if the response to this key/critical question was different from "Never" (e.g., sometimes, frequently, always, etc.), the interviewer tried to understand their modeling practices (i.e., as sketching, model-based or model-driven). Then, by giving the necessary terminology on MDE, MBE and sketching, the second key/critical question about model-driven usage pattern was asked as in the survey (See Appendix B.1 – Survey design and execution). Moreover, since it was also found out in Section 4.2 that modeling purpose directly affected the modeling rigor (hence modeling approach), the interviewer also asked "*Is there any listed purpose while you are modeling?*" by showing the model-driven specific purpose list in the conceptual model (i.e., code generation, documentation generation, test case generation, model transformation and model simulation). Again, if the answer is "*Never/No*", which means that this interviewee is at "sketching" or "model-based" pattern, corresponding in-depth questions were asked to these interviewees. Otherwise, this means that the participant uses "model-driven" techniques -at some degree- and its state-of-practices besides the benefits, challenges, consequences and adoption of MDE were investigated. During the interview session, the interviewer asked the set of questions according to main modeling usage pattern, listened to the answers and followed up answers with additional questions when necessary. During this process, the interviewer tried to find "*a good balance between asking questions, listening to the interviewee's answers, and monitoring what questions have been answered"* by ensuring that all important topics were covered, but in a flexible way [38].

After the interview session, before the analysis was started, a number of activities were conducted. When the interview has been noted and before being transcribed into text, it is recommended to have these notes reviewed by the participant to provide the opportunity for the interviewee to correct, clarify or validate the answers [38]. Therefore, if possible (due to time constraints), after the interview, the taken notes were shown or most critical parts (e.g., the critical characteristics of a diagram to derive pre-investigated modeling patterns) were repeated to the interviewee to give an opportunity for clarification and expansion of specific

answers. When analyzing the data, the interviewer tried to investigate interesting key findings and observations from the informal conversations during the interviews.

4.4.3 *Findings*

53 interviews in 14 different companies had been carried out. In total, interviewees represented about different software engineering roles with different university degrees within different target sectors (Table 9). Our interviewees have, cumulatively, 756 years of software development experience.

Table 9 Interview - company profiles, interview type and number of interviewees

Company / Organization	**Target Sector**	**Type**	**# of Interviewees**
CE-1, Turkey	Consumer Electronics	Face-to-face	6
CE-2, UK		Face-to-face / Skype	4
CE-3, Turkey		Face-to-face	3
CE-4, Taiwan		Skype	2
CE-5, Finland		Face-to-face	2
DA-1, Turkey	Defense & Aerospace	Face-to-face	11
DA-2, Turkey		Face-to-face	5
DA-3, USA		Skype	3
DA-4, Germany		Face-to-face / Skype	3
IT-1, Turkey	IT & Telecommunications	Face-to-face	4
IT-2, Turkey		Face-to-face	2
HB, USA	Healthcare & Biomedical	Skype	3
FB-1, Turkey	Finance & Banking	Face-to-face	3
FB-2, Turkey		Face-to-face	2
Total: 53 embedded software professionals with 756 years of work experience			

In the following sub-sections, the findings and observations on the main software modeling usage patterns (i.e., “no modeling”, “sketching”, “model-based” and “model-driven) to validate and improve our pre-findings are presented. Moreover, important informal question & answer session results are also presented with verbatim quotes of interviewees to understand modeling practices and challenges in these patterns. All non-English quotes (i.e., only Turkish) have been translated to English, as precise as possible, by the interviewer. For this study, due to space constraints, not all, but the interesting points and observations on modeling patterns are reported.

4.4.3.1 *Patterns in “no modeling”*

As the survey (Q10) showed that 11% of respondents have not been using any software modeling (neither informal nor formalized). As seen from our pre-investigated pattern set (Table 8), this pattern (i.e., “0”) needs further analysis to understand why. When interview data is analyzed, there are mainly two sub-patterns, who do not use any software modeling: Some of these participants do not have any software modeling experience (i.e., “not experienced”), whereas some of them do not use it although they have some experienced on that (i.e., “bad experienced”[11]). There are totally six interviewees in this main pattern; two of

[11] As a terminology, “bad experienced” pattern indicates the embedded software professionals, who don’t use any kind of modeling due to disappointing and insufficient experiences of software modeling.

them are “not experienced”, the other four are “bad experienced”. By this way, the interview divides “pattern 0” into two patterns, i.e., “pattern 0.0” and “pattern 0.1”.

When the survey data was analyzed, the ones who stated that they don’t use any software modeling approach, are mainly Physics, Industrial Engineering, Mechanical Engineering and Electrical/Electronics Engineering (EE) graduates. These respondents from the stated backgrounds most probably have not learn any software modeling during university (e.g., from SE courses) and do not need it in their job history, so that they did not take any training on that. On the other hand, Q12 of the survey asked where/how the participant learned software modeling. (i.e., participants might learn modeling in the university and from formal corporate trainings) and the answers are compatible with the question, which investigates the modeling experience (Q11). This result showed that these “not experienced” (i.e., “pattern 0.0”), who did not take any SE courses on software modeling, did not learn it during the job or with corporate training; therefore they did not know about software modeling (and any possible benefits of modeling perhaps) and do not use it even as a sketch. There are only two participants in this interview session, who both graduated from EE, whose replies complied with the survey results.

On the other hand, the survey did not give any further information why some participants do not use any software modeling although they know it. As interviews showed that these participants have bad or poor experiences and failure stories on modeling. A verbatim quote from a firmware engineer, who has 29 years of software development experience: “*Very few firmware projects I have participated in over the last 30 years have used software modeling. The few for which it was tried caused me to come to the conclusion that modeling provides little to no benefit for the vast majority of embedded projects*”. When the reason of such an opinion was asked, he continued: *“The problem is that each embedded system has a unique hardware platform that is unlike any other. It takes more work to try to set up the model to accurately behave as if it were the real hardware than is worth it. Most embedded systems are on the small side, with the code written usually by one or maybe two firmware engineers*”. Another software developer, who has 18 years of software development experience on this pattern mentioned about projects size: “*Small projects simply don't benefit much from modeling as modeling itself requires a significant amount of setup, and the modeling doesn't do anything for you that can be done by simply creating a ‘prototype’ of the code based on the requirements”*. Very similar arguments were given by another software architect: “*Modeling would most likely be of more benefit on large, complex projects*” and “*We have honed our design methods throughout the decades and have become quite successful at embedded systems design without having to use modeling tools*”. As seen, these experienced embedded software engineers think and experienced that modeling is costly for their business due to hardware closeness, uniqueness and project size (i.e., the characteristics of a diagram: HW_CL and T_SIZE (i.e., team size)).

It was interesting that all of these “bad experienced” professionals mentioned about modeling tools’ problems, which is a mandatory for “model-driven” approach, but not for “sketching” or “model-based”. They had some resistances on modeling (e.g. one of the modeling organizational challenges in the survey Q25) and it is difficult to change their negative attitude. During the interview, some findings about modeling benefits, which are claimed to manage the complexity of embedded systems were presented. The experienced firmware engineer states: “*Embedded firmware has always been complex - I can tell you that complex embedded system are nothing new at all, and my career began at the firmware industry's beginning in the mid 80's*”. A verbatim quote from software developer: “*Nothing you said to me is anything new to me at all - I've heard it all many times before over the years and I'm not swayed by "academic" arguments such as yours since they sound great but usually don't have much "ground truth" factored in”*. A project manager, who used to be an experienced board support

package (BSP) developer answers: *"If you are in the BSP business then I wish you the best of luck - it's not going to be easy to try to succeed by going over old ground where dozens of others have failed in the past trying to do this*". Apart from these common opinions, there are also some other issues, like "understanding the notation" of UML (A verbatim quote of one developer: "*too complex and not necessary syntax*" and "*I am sure that even a software modeling professor also might not know the difference between "aggregation" and "composition" in UML, so do I*."), cost of training (e.g., just because of training, they might not complete the project within the required time and budget), and the synchronization problem between model & source code (since they badly experienced "sketching" or "model-based"), made these interviewees (i.e., "pattern 0.1") not use any software modeling.

4.4.3.2 *Patterns in "sketching"*

As survey and interviews showed that there are various patterns in "sketching". To be a sketch user, modeling purpose set of the stakeholder should not include any model-driven specific purpose (i.e., code generation, documentation generation, test case generation, model transformation or model simulation; but might include any general modeling purpose as communication and/or understanding).

During the interviews, it was observed that there exist some "sketch" users, who does not know that they actually do is software modeling. This hidden pattern, could not be investigated from the survey results since such participants might indicate that they do not use any software modeling. In that sense, with the help of this interview session, this hidden pattern (i.e., "pattern 1.x) is figured out in the embedded software development projects.

A verbatim quote from an experienced software developer and designer, whose response to the first critical question in the interview (i.e., "*How often do you use software modeling in your software development life cycle? (either informal and/or formalized: i.e., sketches and/or models)*") was "Never", states *"I tried several modeling tools before, none of them ever delivered something of benefit that was worth the extra time and cost of doing the modeling… These kinds of tools have been promising great things yet I have never seen a single one deliver anything that was really needed to complete a successful embedded design project*". When the interviewer made him remember that his answer to the critical question was not drawing any diagram or sketch on a paper, but now, he mentioned about modeling tools' problems, the interviewer again asked whether he uses any informal sketch or even a state machine to explain something to his colleague on a paper without using a modeling tool. He continued: "*Indeed, yes. Sometimes we use some sketches, similar to statechart diagrams, but informally. We are not using any modeling tool... I assumed that I did not count such drawings as software modeling and just because of that I said "We are not using any software modeling". But, as I said, these drawings are also very rare in our development".*

Similar responses were taken from some EE graduates, who learned modeling after the university, in the job from books and formal corporate trainings. They use both state machines and also sequence diagram-like (i.e., includes some UML elements for informally and casually) drawings. A verbatim quote from one of DSP engineers: "*I used these diagrams just for understanding a problem at an abstract level or for communication purpose; but these are not UML. Are these still counting as software modeling although I do not obey any UML formality?".* Moreover, a systems engineer, who claimed that he does not use any software modeling, but explained during the interview that he used some pen & paper stuff to explain the system scenarios to the necessary stakeholders (e.g., different software engineers, hardware engineers, and also his systems engineer colleagues) without using any formalized modeling language elements (e.g., UML), but with some personal drawings. In fact, with this

information taken during the interview session, his pattern is one of the pre-investigated patterns (i.e., pattern 1.1, "Ad-hoc"), which is presented next.

The common characteristic of these "unaware of modeling" (i.e., pattern 1.x) is that they have not taken any SE courses during the university and try to explain something intuitively and informally without knowing that they actually do –somekind of- software modeling.

Another pattern (i.e. pattern 1.1) in "sketching" is an obvious usage, which is only "pen & paper" with free-format (e.g.., without any formalized modeling language elements). Their purposes are just communication or understanding a problem at an abstract level on an analog media like paper or white/blackboard. Notice that it does not mean that all other main usage patterns (i.e., model-based or model-driven) do not use paper or whiteboard; indeed, such analog mediums might be a quick solution for a better communication and faster idea sharing technique in some situations (Q13); but this pattern "only" uses such an approach on an analog media as ad-hoc. Both our survey results and interviews showed that mainly systems engineers, requirement engineers and low-level (e.g., BSP, DSP) engineers are in this category. A verbatim quote from a BSP engineer: *"Sometimes we use sketches, similar to component diagrams to show the relations between some drivers, chips or processors, but informally... While explaining the input of a BSP driver chip to my colleague, I use some boxes, circles with pen & paper; but we are not using any modeling tool"*. In fact, depending on his purpose, his modeling approach satisfies his motivation (i.e., in that case, for team collaboration some sort of sketch is enough for communication and understanding); so no need for any other (e.g., more formal) approach.

In the survey, the respondents, who state that they were doing informal modeling, make the sketches, which include some essences of UML (i.e., some elements of state machine/charts, but not dependent on strict UML rules) as reported in [33], where some participants use UML elements informally. Therefore, these participants (i.e., pattern 1.2), who do informal modeling, answered this question by selecting some diagram types. (i.e., some participants, who use "Sketch/No formal modeling language", draw a use case diagram or sequence diagram informally). Semi-structured interviews also showed that most people use sequence diagrams informally to convey the communication among the entities in a given system. Their purpose is just a quick communication and understanding a scenario. This sketching might have occurred either on analog or digital media, but without any documentation purpose (e.g., documenting design).

Another pattern for sketching (i.e., pattern 1.3) is based on the purpose of modeling (i.e., documenting design) and the medium type while modeling (i.e., digital or analog, which affects "archivability" and "lifespan" of the diagrams). The lifespan of the sketches created on analog media are less than the ones created digitally via PC or tablet/smartphone (Remember the relations between PURPS (purpose), MEDM (medium type), ARCHV (archivability) and LFSP (lifespan)). In this pattern, in short, there is a "documenting design" purpose, but analog media usage is more frequent than to the digital ones. These modeling stakeholders use some transitions, after the modeling process, during documenting (e.g., during the semi-structured interviews, it was observed transitions from one medium to another to achieve more archivability and hence lifespan). For example, some analog models (e.g., either in paper or in white/blackboard) are archived by saving a digital picture or by redrawing them digitally for the customer requirement.

4.4.3.3 *Patterns in "model-based"*

The survey results showed that some characteristics of model-based category and some patterns of sketching are very similar (i.e., documenting design as a purpose but with different

media type usage degree). Note that with pattern 1.3, all the upper pattern (i.e., model-based and model-driven approaches) has "documenting design" purpose with other characteristics of a diagram [75]. Although there are some exceptions, in which more rigor (with strict enforcement) without model-driven purpose (i.e., artifact generation) is used, almost all interviewees in this pattern use UML selectively and often informally. Here, the differentiation point is SDLC phase(s), where software modeling is used (Q18). It was realized that the patterns are originated and depend on whether SDLC set include "Implementation or Testing" or not. If the diagram might be an input for implementation or testing phase, this modeling might be close to prescriptive approach; otherwise descriptive approach. In that sense, one of the patterns in model-based (i.e., pattern 2.1) use modeling very close to descriptive approach during systems/software analysis, business process analysis, systems/software design or maintenance phase of SDLC. The other pattern (i.e., pattern 2.2) uses the diagrams either in implementation or testing phase (or both of them).

A system engineer states: *"With the help of a good sequence diagram, we save lots of cost and time since we get rid of unnecessary meetings between stakeholders"*. He continued if all Software Configuration Item (SCI)[12] and modules are well-depicted in a complete sequence diagram with the necessary inputs (e.g., message interfaces in Interface Control Document (ICD)[13] during a system scenario, every software engineer can understand the corresponding scenario without looking at the "text description" of it, which might cause some misinterpretation. In fact, when these sequence diagrams are analyzed with further questions during the interview, it is seen that they were drawn in MS Visio without strict UML. The creator of this sequence diagram (i.e., the systems engineer, who uses this diagram in "analysis" phase) gives this input to another modeling stakeholder (i.e., a software developer, who uses this diagram both in "analysis and implementation") without any model-driven specific purpose. This shows that in the same main modeling pattern, there are some cross-life cycle activities, in which one modeling stakeholder's input might be another's output, whose patterns are different.

Note that this pattern might be close to model-driven if the stakeholders in this pattern would use more prescriptive approach with more constrainment so that their format is readable by a machine). However, some of the interviewees had also bad/poor experiences on the modeling environment and tool, which might be the reason of not using any model-driven approaches. When the interviewer asked why they thought that the tool they experienced for their systems is not sufficient for their needs, a software developer said: "*The only people who actually know enough about embedded design to be able to create effective modeling tools for embedded are those embedded engineers who have lots of experiences on hard-core embedded*

[12] Software Configuration Item (SCI) is an entity designated for configuration management, which may consist of multiple related work products like process description, requirements, design, source code, test or interface description. In practice, "configuration item" may be interpreted as "configuration component" or "configuration unit" as appropriate. In this context, this system engineers use this term to indicate "configuration unit" as an executable, which has some design documents.

[13] Interface Control Document (ICD) in systems engineering and software engineering, describes the interfaces between subsystems or to a system. For example, a communications interface is described in terms of data items and messages passed, protocols observed and timing and sequencing of events. An ICD may also describe the interaction between a user and the system, a software component and a hardware device or two software components.

development". He continued: "*Nobody else understands the real embedded development process, they pretend they do - they think they do, but no*". Another verbatim quote of 26 years of software development experienced developer: "*Just send me an in mail, I'll make sure you don't make a bad investment in developing/marketing embedded tools, that's my guarantee and I'll stand behind it*". This experienced software engineer knows about model-driven concepts and the necessity of tool in this approach; therefore the interviewer also suggested to develop in-house modeling tool for their needs with their own DSML and benefitted from auto generation of some software artifacts like code, document, etc. His responses was "*When I compare the pros/cons of such an investment, I can say that we are good at what we are doing without model-driven. We can benefit from communication, abstraction, understanding and documentation for the new comers and that is enough*". As seen, if there is a bad and disappointing experience on modeling or MDE, it is very difficult to change the attitudes and technology acceptance. But, the critical question, do we need it or do they have to use model-driven approaches? It depends on the characteristics of a diagram (mainly, purpose) [75].

In "model-based" patterns, some tool challenges with the used programming language were also observed. One of the user interface (UI) designer for medical devices, who use C programming language for informative touch-screen, stated that there is not any embedded modeling tool in their toolset. He continued that he uses modeling concepts in some diagrams, but he could not use their toolset for code generation. A verbatim quote from this developer: "*I have lots of colleagues in different industrial sectors, who also do some UI development. However, they use another language than C (e.g., Java, C# or C++) and they can easily use modeling and code-generation of these models. Currently, I am not benefiting from this facility"*. He thought that his programming language and corresponding toolset restrict him while modeling. He continued "*I also wanted to guarantee that my state machine is reflected to my code. Now, I draw this state machine on PC and I implement it manually*". After asking about the synchronization issue on these state machines (i.e., source code & model compatibility), he answered that this is really a problem. He laughed by saying "*Human factor!*". He mentioned about organizational culture: "*The team must be motivated to use the new approach. If your team members do not like modeling and also there is no good tool support, your attitude towards a new technology or approach does not make any sense as an individual since there is no visible real benefit according to your experienced developers, who say the last word as a decision maker*". He was willing to use model-driven approaches if his tool support and organizational culture challenges are coped with in the future. He stated: "*I know MDE benefits besides its challenges. In the near future, we will try C++ for our new chipset. At that case, we can use a new tool, which easily support modeling. Then, I hope I can show some "real" benefits of modeling and I can convince my technical leaders or project managers*". In fact, Platform Independent Modeling (PIM)[14] concept might achieve modeling independent from programming language. However, in some cases, programming language choice affects both modeling attitude and also development process due to tool support. Therefore, another observation is that organizational resistance might disappear with a relevant tool support, which shows "real" model-driven benefits.

4.4.3.4 *Patterns in "model-driven"*

In "model-driven" category, first, it was also observed that there are some interviewees (i.e., pattern 3.x), who actually use MDE without knowing it, so they are "unaware of MDE". This

[14] Platform Independent Modeling (PIM) is independent of platform or implementation technology of the system. In this modeling, the model, which contains no reference to the underlying technological platform, focuses on the high-level business logic.

hidden pattern is also derived from the interview results as in "pattern 1.x". In these situations, generally, there are mainly DSL/DSML usages while modeling. Although the modeling stakeholders benefit from model-driven concepts (e.g. with model-driven purpose(s) via DSL or UML profiles usages), they are a bit confused since there is not UML related diagrams while modeling.

During one of the interviews, a participant, who stated "model-driven" usage since he benefits from "code generation" and "documentation generation" of an MDE tool, confessed that he was a bit confused about "modeling" terminology. In fact, the tool, which the interviewee is using, has its own DSL and there is no UML element in this tool; the inputs are specifications of the interface messages and their parameters; then the tool generates MDE artifacts. This participant asked: "*Yes, I know, there is a code and document generation in that tool, but are these related to "model"? In fact, there is not any UML element in the tool, right? Is this really a software model? Where is the model?"*. The similar conflicting terminologies were also encountered with some systems engineers, who use Matlab/Simulink for model simulation, in which there is no specific UML diagram.

As survey data analysis showed and it was also observed during interviews that some participants are using model-driven techniques in a limited way (i.e., pattern 3.1) without benefiting from "code generation", "model transformation" or MBT. These "limited" model-driven users mainly use diagrams for "document generation", but not for code generation, or they use some models (i.e., Simulink) for just "model simulation" (i.e., a systems engineer, who uses model simulation in designing an algorithm; but not sharing this with any software engineer during implementation phase; or a software developer, who just wants to generate documents from the models).

The most common consumers of prescriptive models (i.e., model-driven) are model transformations [15]. Therefore, it is not so important to have a graphical syntax to represent the model (as in UML), but these models should be represented in a format that is readable by a machine with a language (as in DSL). Furthermore, one of Eclipse Committee member and CEO of OBEO, Etienne Juliot presented "UML" as "Utopian Markup Language" not "Unified Modeling Language" during his speech on "DSL vs Standards" at one of the modeling-related event [81]. He said that it would be utopia that UML can be used for all purposes in all SDLC phases. He claimed that for maximum benefit, there should be a customization on DSL, models and tools (as code generators and visual editors) besides using UML profiles. Therefore, we also investigated that model-driven users are characterized whether they use a DSL/DSML/UML profile or not.

Moreover, survey data clustering in RapidMiner showed that DSL usage affects the patterns in "model-driven" usage (APPENDIX C – Pre-investigated modeling patterns' visualizations). If modeling language set (e.g., Q14) does not include any DSL/DSML/UML profiles or used diagram types does not include any DSL-based diagrams (i.e., Q17), these users are mainly using UML (i.e., pattern 3.2). Notice that since UML is a general-purpose modeling language, its usage is not only restricted to modeling software (See Q14). As observed in [15], "*UML is not so popular for prescriptive models*" since the semantics of UML models is not exactly defined and this would hamper the automatic translation towards other models or code. On the other hand, the model-driven users, whose modeling language set includes any DSL/DSML (besides any possible UML profiles) are in pattern 3.3. Therefore, as the survey and interviews showed that there is a distinction and grouping while categorizing "model-driven" approach according to their modeling purpose(s) and DSL-like modeling language usage.

As a result, with the help of these interviews, the different modeling usage patterns in embedded software development were better understood by validating the survey data analysis

in the pre-investigated patterns; but more importantly, the hidden patterns are identified with deeper and more personalized modeling experiences. As a lesson learned, some interviewees (depending on their university degree, where/how modeling was learned and hardware closeness) have some resistances and misbeliefs for modeling and MDE. Some of them think that software modeling is only done with a tool via some (formal?) UML drawings; however software modeling is not restricted with UML since it also includes descriptive modeling as sketching or DSLs without UML diagrams.

After validating and improving our pre-investigated patterns, 12 patterns are characterized in embedded software development projects as seen in Table 10. Notice that, all quantitative results taken from our previous findings (i.e., survey and interview results) are depicted in this table with their ratio (Note again that in our survey, sketching and model-based usage were in the same category; so their merged ratio is 59,5%). Since there is no "*pattern 3.x*" or "*pattern 1.x*" in the survey results; when their ratios are merged into the corresponding possible pattern (e.g., "pattern 1.x" might be "pattern 1.1" or "pattern 1.2"; and "pattern 3.x" might be "pattern3.1" or "pattern 3.3"), it is interesting to have similar results in both survey and interview. To validate this similarity, T-test[15] for Interview Result vs Survey Result was applied and its results is given in Table 10. As seen T-value[16] in this test is "-0,01", which shows the similarity between results (Note that "unaware" patterns were counted as 0% in survey results; and "none" ratio was counted as 11,2% for the interview results and 11% for the survey results in this test; hence N=11).

Table 10 Interview results on modeling patterns by comparing survey results with T-test

Main pattern	**Patterns**		**interviewees**			**% in survey results**	
			#	**%**			
model-driven	3.3	With DSL-like	8	15,1	32,1	16,9	29,5
	3.2	Without DSL-like	4	7,5		6,5	
	3.1	Limited	3	5,6		6	
	3.x	Unaware of MDE	2	3,7		-	
model-based	2.2	Prescriptive	10	18,9	30,1	24,9	59,5
	2.1	Descriptive	6	11,3		13,7	
sketching	1.3	Archived	2	3,7	26,5	3,6	
	1.2	Selective	7	13,2		13,1	
	1.1	Ad-hoc	2	3,7		4,1	
	1.x	Unaware of modeling	3	5,6		-	
none	0.1	Bad experienced	4	7,5	11,2	11	
	0.0	Not experienced	2	3,7			

[15] T-tests are handy hypothesis tests in statistics when you want to compare means and tells you how significant the differences are. In this research, two-sample T test was applied to compare the means of interview and survey results.

[16] The T-value is a ratio between the difference between two groups and the difference within the groups. The larger the t value, the more difference there is between groups. The smaller the t value, the more similarity there is between groups

Two-sample T for Interview Result vs Survey Result

	N	Mean	StDev	SE Mean
Interview Result	11	0,0905	0,0522	0,016
Survey Result	11	0,0907	0,0767	0,023

Difference = μ (Interview Result) - μ (Survey Result)
Estimate for difference: -0,0003
95% CI for difference: (-0,0593; 0,0588)
T-Test of difference = 0 (vs ≠): **T-Value = -0,01** P-Value = 0,992 DF = 17

4.5 Modeling Cultures

By analyzing the common/different characteristics of software modeling and applying merging techniques between some patterns to get maximum benefit from this categorization (e.g., to better guide stakeholders with necessary and sufficient process & tool improvements for an effective modeling approach), six modeling cultures in embedded software development projects are identified: *None, Performed, Formalized, Archived, Prescripted and Auto-generated*.

- "*Auto-generated*" culture includes pattern 3.x, pattern 3.1, pattern 3.2 and pattern 3.3,
- "*Prescripted*" culture includes pattern 2.2,
- "*Archived*" culture includes pattern 2.1 and pattern 1.3,
- "*Formalized*" culture includes pattern 1.2 and pattern 1.x
- "*Performed*" culture includes pattern 1.1 and pattern 1.x
- "*None*" culture includes pattern 0.1 and pattern 0.0.

Accordingly, we can say that a culture (as a particular group of modeling patterns) consists of different characteristics of diagram development and usage (e.g., modeling rigor, purpose, medium used while modeling, SDLC phase where modeling is used, etc.). In this categorization, a "higher" culture can use the characteristics of the "lower" cultures and the modeling stakeholder might apply their lower level patterns' modeling practices, if necessary; but not vice versa. For example, a modeling stakeholder, who is at pattern 3.3, can also use analog medium type (e.g., paper) besides digital ones (e.g., modeling tools in PC) while modeling, i.e., sketching without any modeling rigor as being at pattern 1.1. Therefore,a "higher" culture does not necessarily entail a more "correct" or "mature" use of modeling with respect to job/task requirements of the stakeholder although a change into a "higher" pattern might allow the stakeholder to better use software modeling with possibly some extra costs and possible challenges. Notice that whenever a modeling stakeholder goes to a higher level pattern from a lower one, the initial cost and challenges of this modeling approach increase; however the benefits of this approach also increase. These cultures' characteristics are given in Table 11 with their main focus. Remember that there is no "*maturity level*" as MML in this categorization (See Section 2). Since the cultures depicted are based on all modeling characteristics of the *individual* stakeholder (e.g., purpose, task/responsibility, SDLC phase, etc.), this scheme also differs maturity models based on organizational concepts in that it focuses on individual practices.

Table 11 Modeling cultures of embedded software development projects and their characteristics

Approach/ Main Pattern	Culture	Focus	Patterns	Characteristics
model-driven	**Auto-generated**	Guaranteeing model software artifacts compatibility	3.3 3.2 3.1 3.x	This is the culture, where software modeling turns into programming (automated generation of code from models) since programmers deal with diagrams instead of focusing on the implementation details. Apart from automatic code generation, there are also documentation and test case generation from these precise models. In this culture, the diagrams have more lifespan and archivability since the modeling tool/environment, which should be digital, plays a crucial role while generation software artifacts. Since it is a more detailed and complex modeling approach, which has strict enforcement, the modeling stakeholder is very close to 100% of "the variable formality" slider of modeling, hence the rigor. In this culture, the modeling stakeholder ensures about the synchronization of model and the other software artifacts (i.e., source code, test driver, documentation and also simulation). "Model transformation" (i.e., Model2Model, Model2Text or Text2Model) is very crucial in this culture. In fact, model + transformation -> software. Although initial cost and challenges in this culture is much more than the other cultures, overall benefits increase since code correspondence is guaranteed.
model-based	**Prescripted**	Prescriptive without auto generation	2.2	This is the culture, where the modeling stakeholder uses mostly prescriptive diagrams mainly in "Implementation" or "Testing" of SDLC. The diagrams are more precise and there is more strict enforcement while using modeling languages. Therefore, rigor also increases. However, there is still "*human factor*" while coding since these diagrams are not necessarily the key artifacts of the development. For example, designers specify the diagrams (i.e., on paper or by using modeling tools), but then these diagrams are directly handed out to the developers to manually write the code.
sketching	**Archived**	Archivability & Lifespan	2.1 1.3	This is the first culture, where a diagram, which is drawn on either analog or digital media, becomes archived. The diagram might be either descriptive or prescriptive, which means that the purpose might be "communication" or "understanding "as in sketching, but with an extra "documenting design" purpose. In this culture, there might be some situations in which the diagram was originally drawn on an analog media; but, while archiving, there might be some transition between analog to digital *(For example, taking a photo and save it as JPEG; or re-drawing it digitally)*. In that sense, archivability affects the lifespan of these diagrams; hence some quality factors (i.e. maintainability, traceability, reusability, etc.)

	Table 11 (continued)			
	Formalized	Modeling Language	1.2 1.x*	This is the first culture, where a diagram becomes a representation of the software being built with some formalized modeling language elements. Modeling stakeholders use the diagrams casually & informally with some UML elements, but selectively. Therefore, this culture is something UML-like sketching. The diagrams are not just a roadmap, but show the general structure by becoming “somekind" of true representation (reflection) of the software. The rigor and enforcement start to play a role, but in a light way and not depending on the strict UML rules. Keeping the diagrams up-to-date with the source code is seen to be unimportant and time consuming. Therefore, update problems still exist since there is no archivability. Programmer still makes business decisions. No guaranteeing to eliminate possible "*human factor*" problems while implementing the code.
	Performed	Ad-hoc	1.1 1.x*	This is the starting level for software modeling via “ad-hoc” approach by using pen & paper with some free-format drawings (e.g., boxes & lines) but without any formalized modeling language element. In other words, this culture is the starting of “descriptive modeling” as sketching without any formalized modeling elements *(not precise, no strict enforcement, no rigor – it is very close to 0% on the variable formality slider of modeling)* The main purposes here might be selective communication or understanding instead of specification (i.e., communicate ideas with colleagues or understanding the problem at an abstract level) The modeling stakeholder has no standard process or approach for software modeling. It looks like a specification of software with some high level diagrams to explain the overall architecture/system/requirements, etc. by showing the main parts of the system under development. There is no details of the diagram. (e.g., no attributes & fields, no operations/methods) Not archived, hence impossible to keep up-to-date (the lifespan of these sketches are very short – depend on the lifespan of the analog medium; i.e., paper or white/blackboard) Modeling education and awareness are the main challenges in this culture.
no modeling	**None**		0.1 0.0	Bad experienced: Not using any software modeling although they know it. The possible reasons are bad/poor and insufficient experience, failure stories (time, cost, etc.): Misbelieves, resistances, misunderstandings on the terminology, hardware closeness & uniqueness, project size, understanding the notation of UML, cost of training, synch/consistency problems. Not experienced: Not took any SE courses on modeling during university and no need it in the job. Typically, non-CS/CENG/SE graduates.

By this way, all modeling patterns and cultures in the embedded software development projects are identified by addressing RQ1.

4.6 The Characterization Model: MAPforES

After identifying the patterns and cultures in the embedded software industry with their characteristics based on prior findings (i.e., survey and interview results data), this section proposes a characterization model called MAPforES, which identifies and defines a modeling stakeholder's pattern and culture as commonsense practices by presenting what the similar profiles in the embedded domain is doing while modeling (via the database constructed with survey data). By this way, besides identifying and defining the current pattern and culture, this model identifies the widespread modeling practices (e.g., process and tool) in embedded software development projects by referencing to a set of commonsense industrial practices.

During the creation of the MAPforES, various prediction methods (e.g., an artificial neural network (ANN), lazy (k-NN) or support vector machine (SVM)) were applied to get the best results for the available training set. However our data set size, which feed the model was restricted (i.e., <1K) and it was very difficult to formalize a model with any deep learning mechanism even the survey data might be split to augment the training data with some techniques (i.e., remember that the survey includes a multiple-response question (Q7), which was about the target sectors of the products developed by the company, in which the respondent is working; and this data might be split based on a single target sector to have more data). However, the data would be still missing to get a well-suited model to achieve the concept. The straightforward technique was "decision tree" mechanism since all the necessary & significant characteristics were derived during the identification of all patterns (except "hidden patterns").

After deciding to construct the decision tree, it was necessary to take feedback from software professionals via expert opinion strategy before finalizing it. Then, by taking feedback from 14 software professionals (See Table 12), the final outcome of the decision tree is constructed as seen in Figure 26. Note that none of these software professionals participated in the previous interviews for this study; however, it cannot be guaranteed whether any of them participated the survey or not.

Table 12 Expert opinion demographics for "decision tree" used in the model

Organization	**Target Sector**	**Position**	**# of experts**
DA, Turkey	Defense & Aerospace	Software Architect & Developer	2
		Software Developer	2
		Software Tester	1
CE, Turkey	Consumer Electronics	Software Architect	3
UN-1, Turkey	Academia	Academician	3
UN-2, Turkey		Academician	2
UN-3, Turkey		Academician	1
Total: 14 software professionals with 234 years of software development experience			

This decision tree is the heart of the model, which identifies and defines the modeling stakeholders' current pattern and culture.

Accordingly, the model, firstly, takes the characteristics of diagram development and usage of the modeling stakeholder (See Section 4.2). The model has pre-given sets for software modeling characteristics as purpose of modeling, medium type used while modeling, SDLC phase where diagrams are used and modeling language properties, if any. Depending on the characteristics of a modeling stakeholder, the current modeling pattern and culture is found with this model.

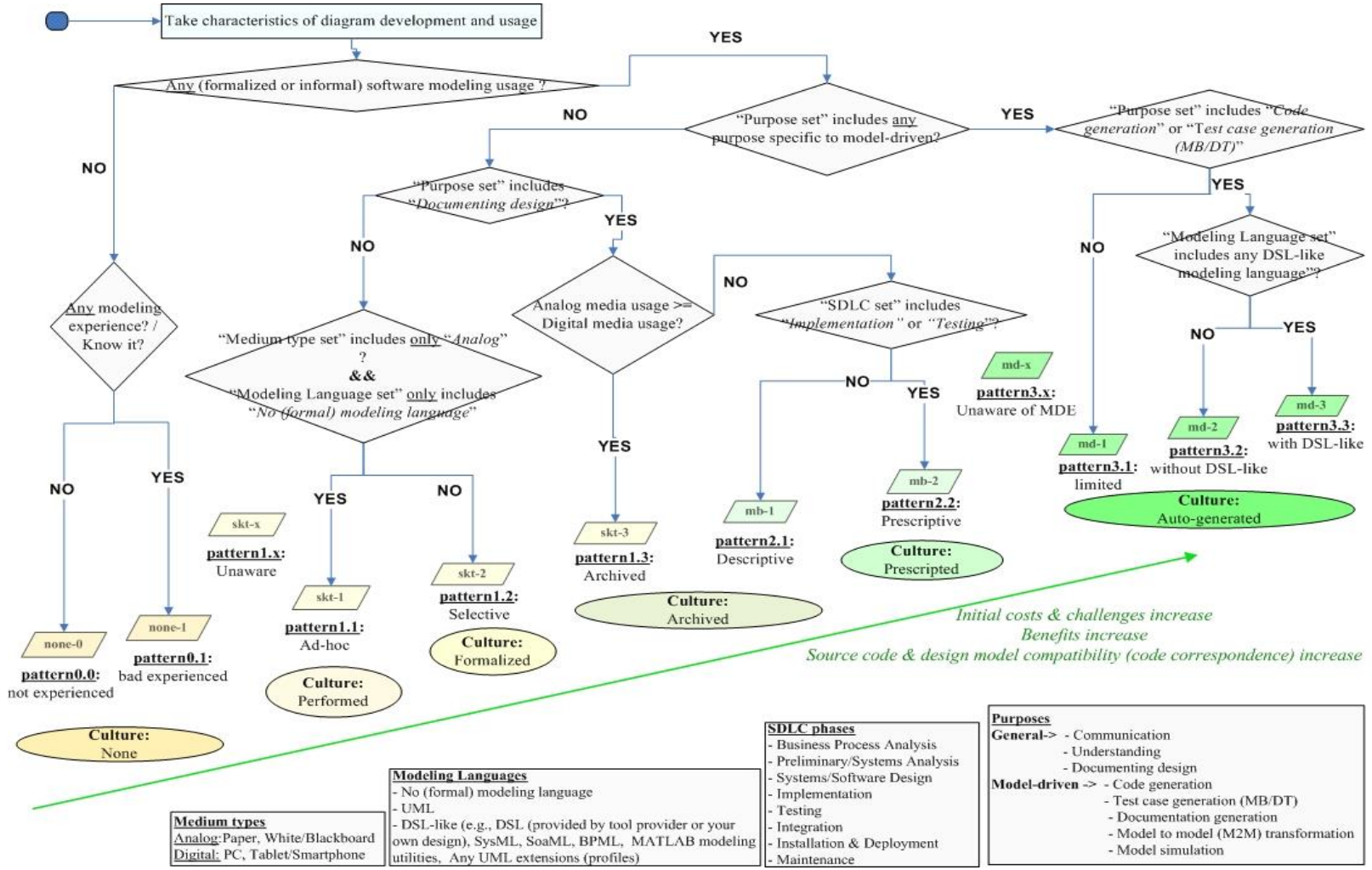


Figure 26: The Decision Tree of the Characterization Model

Moreover, since the model takes the characteristics of modeling stakeholder as inputs, it presents the similar demographics' modeling practices based on the available survey data [77] as a set of commonsense industrial modeling practices. By querying the similar demographics in the survey database with the stakeholder's input, the stakeholder learns as suggestions what their competitors do in the same industrial sector while modeling. By this way, the stakeholder gets answers to strategically important questions like the necessary modeling approaches, languages, tools, etc. in this domain (RQ2.1) Notice that all these characteristics of this model is implemented during the case studies, which will be presented in Chapter 5.

After finding out the current modeling pattern and culture, it is necessary to see all patterns and cultures with their corresponding characteristics of a diagram in a single chart. To achieve this, Figure 27 is depicted. Remember that there is no "maturity level" in this categorization since different characteristics of software modeling might have an effective modeling approach (See Section 2).

As seen from Figure 27, the characteristics of diagram development and usage are mapped to the patterns and cultures. The "*arrows*" in the figure means that whenever the modeling stakeholder goes in that direction, the value of a corresponding characteristic increases (e.g., in the upper cultures, modeling rigor, archivability & lifespan and code & model correspondence are increasing). On the other hand, as mentioned, the upper pattern and culture might have all below patterns and cultures' characteristics, but not vice versa. For example, a modeling stakeholder, whose pattern is 3.3, might have "communication" purpose while modeling, besides "code generation"; however any modeling stakeholder, whose pattern is 1.1, cannot have any challenges related to tool or code generation. These mappable characteristics make the analysis and further recommendation to the modeling stakeholder easy and usable.

With the help of this chart, the modeling stakeholder can understand some relations between characteristics (e.g., the possible benefits and challenges of modeling practices, if the current pattern and culture are wanted to be changed). The part of this model provides a place to start and a common language with the benefits of a modeling community's prior experiences (via survey, interview and empirical study results). In that sense, the model provides a roadmap for software modeling practices improvement, if necessary (RQ2.2). Remember that, according to the characteristics of a diagram, the stakeholder might not want to change its current modeling pattern and culture (e.g., depending on the purpose of modeling).

Modeling Approach	Main Pattern	Sub-pattern		% in survey results (627)		% in interview results (53)		Modeling Culture	Characteristics of diagram development and usage							
									PURPS	RIGOR /ML	MEDM	ARHV/ LIFSP	SDLC	CORRS	BENFT	CHALL
Prescriptive	Pattern3 (model-driven)	.3	With DSML	15,4		15,1		Auto-generated	CodeGen MBT						GuaranComp for source code & test driver	DiffCodeGen: *Opt&Perf, SwCert ModTrans
		.2	Without DSML	6.5	29,5	7,5	32,1									
		.1	Limited	7,6		5,6			DocGen ModSim Transfor		MD tool/env				GuaranComp for documentation	DiffModDbg LackOfModChck
		.x	Unaware of MDE (3.1&3.3)			3,7.										
	Pattern2 (model-based)	.2	Prescriptive	25,2		18,9		Prescripted					Imp Tst			
Descriptive		.1	Descriptive	12,7		11,3		Archived	DocDes						QualImp: *Port	ModQual ModVV ML
	Pattern1 (sketch)	.3	Archived	3,5	59,5	3,7	56,4								QualImp: *Maint, Trace, Reuse, Relia	ToolSprt Resist
		.2	Selective	13,6		13,2		Formalized		some UML elements	Digital		Des		CostSvng QualImp: *Prodc ShrtDevTime TestEff	Expr Training
		.1	Ad-hoc	4,5		3,7		Performed	Commun Underst		Analog		Ana Mai Int I&D		ManCompl TeamCol	Education Awareness
		.x	Unaware of modeling (1.1&1.2)			5,6										
No	Pattern0	.1	Bad experienced		11	7,5.	11,3	None								
		.0	Not experienced			3,7										

Abbreviations

Purpose	**CodeGen**	Code generation	**Underst**	Understanding	
	Commun	Communication			
	DocDes	Documenting design			
	DocGen	Documentation generation			
	MBT	Test case generation (MB/DT)			
	ModSim	Model simulation			
	Transfor	M2M transformation			
Benefit	**CostSvng**	Cost savings			
	GuaranComp	Guaranteeing software artifact & model compatibility			
	Maint	Maintainability			
	ManComp	Manage complexity			
	Port	Portability	**TeamCol**	Team collaboration	
	Prodc	Productivity	**TestEff**	Test effectiveness	
	Relia	Reliability	**Trace**	Traceability	
	Reuse	Reusability			
	QualImp	Quality improvement			
	ShrtDevTime	Shorter development time			

SDLC	**Ana**	Business Process Analysis / Preliminary/Systems Analysis
	Des	Systems/Software Design
	Imp	Implementation
	I&D	Installation & Deployment
	Int	Integration
	Mai	Maintenance
	Tst	Testing
Challenge	**DiffCodeGen**	Difficulties with automatic code generation
	DiffModDbg	Difficulties with model level debugging
	Expr	Modeling expertise in the company
	LackOfChck	Lack of model checking capabilities
	ModQual	Model quality (i.e. how to define, assure, predict, measure, improve and manage it?)
	ModTrans	Model transformation/merging ((.e. how to integrate/merge models in different projects?)
	ModVV	Model verification/validation techniques
	ML	Modeling languages' itself (i.e. DSML needs)
	Opt&Perf	Optimization and performance issues
	Resist	Resistance to change
	SwCert	Software certification (i.e., for safety-critical systems)
	ToolSprt	Tool support

Figure 27: The relations between modeling patterns and cultures with the corresponding characteristics of a diagram

CHAPTER 5

APPLICATION OF THE CHARACTERIZATION MODEL

The goal of this chapter is to apply the characterization model, which finds out the modeling patterns and cultures of embedded software development projects via case study strategy. The empirical study reported here is based on multiple case studies, which included a series of both structured and semi-structured one-to-one interviews to evaluate the model presented in Chapter 4 by capturing detailed contextual description and observations. The study includes two companies geographically distributed over two cities in Turkey. One of the companies has two different organizations, which operates different subsectors of embedded software industry; therefore, the research includes three cases (i.e., organizations) based on different subsectors. The interviews were conducted over two months with 35 embedded software professionals. The first section gives the research methodology, the second section presents the research process and findings with its threats to its empirical validity. The results for each case study with participants' answers are presented in [82] as a technical report to record all the data digitally (166 pages of raw data of all participants' evaluations) besides their corresponding case study database as all the actual documents and other evidences collected on paper.

5.1 Research Methodology

The research methodology that is undertaken is the multiple case study, which provides triangulation of both quantitative and qualitative data in accordance with empirical SE research principles [38, 41, 83, 84].

5.1.1 *Goal and Research Questions*

The goal of these case studies is to apply and observe the usefulness of the model presented in previous chapter, by identifying and defining the current modeling usage pattern and culture based on the characteristics of diagram development and usage. It is also used to identify stakeholder's modeling processes for commonsense practices in the multi-faceted industrial context (e.g., in different industrial embedded sectors). In order to achieve this, the characterization model and related artefacts are used to give recommendations to the participants where the interviewer also acts as an evaluator, who analyzes the participants' modeling characteristics and derived recommendations (e.g., commonsense and popular modeling practices like languages, tools, etc.) from the matching demographics. Based on the above goal, the following RQs and sub-RQs are raised and stated in Section 1.2 to test the hypotheses in practice:

Note that as an initial input for the model, to understand the modeling stakeholder's pattern and culture, "interview questions" (questionnaire) was used to get the necessary characteristics of a diagram (See APPENDIX D – Questionnaire used in multiple case studies). In this questionnaire, some questions were similar to the survey, and some of them were improvised and detailed specifically during the interviews and after direct observations.

RQ3: Is the proposed model useful and generalizable?
- **RQ3.1.** Does the model reflect stakeholder's current modeling pattern and culture?
- **RQ3.2.** Is the model useful and conceptually insightful?

To address this RQ, an evaluation form (See APPENDIX F – Evaluation form template) is used to evaluate the result of the model with respect to validation criteria [85] (Note that this evaluation form includes the questions in Table 22, which will be detailed in Section 5.2.4).

5.2 Research Process

Benbasat et al. argues that multiple case studies can be used when the aim of the research is description, theory building or theory testing [40]. When conducting a case study, there are five major processes: case study design, preparation for data collection, collecting evidence, analysis of data and reporting [38]. On the other hand, by concentrating on multiple case studies, both Runeson et al. [38] and Yin [41] illustrate how multiple case studies may be conducted as in Figure 28.

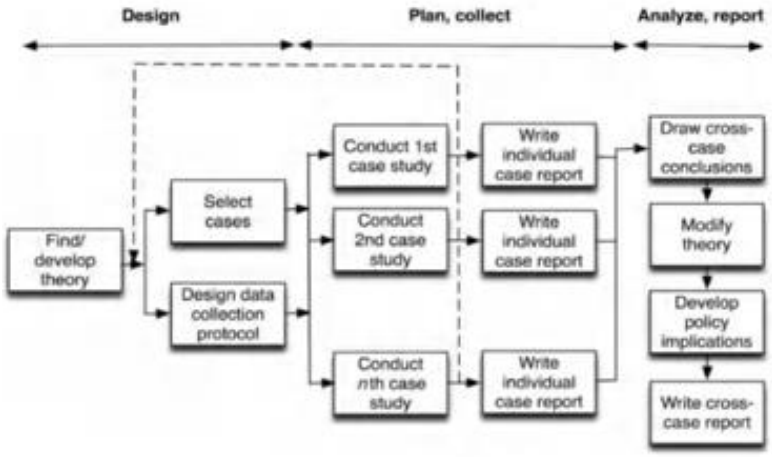


Figure 28: Example process for multiple case studies adapted from [38, 41]

Main phases of a multiple case study reported here have been applied to be detailed in the corresponding sub-sections as shown in Table 13 [38, 41].

Table 13 Multiple case studies research process

	According to [41]	**According to [38]**	**In this study**
Design	Find/Develop Theory	High level design Goal RQs Model	See Section 5.1.1. Theoretical Framework (which was derived by previous findings)
	Design Data Collection Process	Detailed design and preparation for data collection	See Section 5.2.1
	Select Cases		See Section 5.2.2
Plan, Collect	Conduct Case Study	Plan & Data Collection	See Section 5.2.3
Analyze, Report	Analysis / Modify the model, if necessary Reporting		See Section 5.2.4

5.2.1 *Design*

The case study protocol in this study is "flexible" since it includes both structured and semi-structured parts [38].

During the data collection, "*the principal decisions on methods for data collection are defined at design time for the case study, although detailed decisions on data collection procedures are taken later"* [38]. In this study, we use interviews mainly for two reasons: (1) to take stakeholder's demographics and software modeling practices to understand the characteristics of diagram development and usage (which is the structured part); (2) to get personal experience of the stakeholder after the responses to confirm and validate the responses via face-to-face in depth analysis besides direct observations (which is the semi-structured part).

To prevent misinterpretations during data collection, a presentation on "*Modeling patterns and cultures of embedded software development projects*" was designed to be given on the site as the first step of the study including brief information about the study, terminology used, etc.

For data collection process design, we planned to apply the principles suggested by Verner et al. [86]:

(1) Use multiple sources of data,

(2) Create a case study database and

(3) Validate data.

Questionnaires are used to fulfill the data collection need of the structured part of this research [83]. Before the company visits, the data to be gathered is outlined as a questionnaire (See APPENDIX D – Questionnaire used in multiple case studies), and there are "evaluator notes" parts in it, which are filled out after the first round of the interview when the evaluator takes notes on all given responses. By this way, the interview results have both closed-ended and open-ended answers.

The questionnaire provides all necessary inputs to the model by taking necessary characteristics of diagram development and usage (during first round of the interview). This first part is answered individually by the participant without any interaction of the evaluator. After the completion, these questionnaire forms are not collected until the second round, which is conducted face-to-face. During this second part, the responses of the participants are checked whether there is any misunderstanding or any missing critical information in the questionnaire (e.g., wrong data for modeling practices, which is caused by "unawareness" of modeling characteristics). In order to increase data consistency, besides the interviews, several extra source of information about modeling practices (e.g., any written material, medium type used) are planned to be used extensively during this process.

After the data collection process, all answers are analyzed and the model is applied to the participant's characteristics of a diagram during the break. After this stage, the evaluator send two forms to the participants via email. The first form summarizes the interview results (See APPENDIX E – Evaluator notes/observations & Results) and the second is used to evaluate the model usefulness by the participants (See APPENDIX F – Evaluation form template). This evaluation form, which addresses RQ3 was developed with an evaluation strategy, which was adopted from [85, 87] for this study.

Accordingly, it has the following criteria given in Table 14:

Table 14 Validation criteria used in the evaluation strategy

Result Validation	*"This criterion investigates the opinion of the potential stakeholders about the model"* Specifically, does the model produce expected and relevant results? It is concerned with the quality of the model with respect to its benefits
Utility Validation	*"This criterion investigates whether the model is useful"* Specifically, does it produce helpful results so that the model becomes useful?
Comparison Validation	*"This criterion investigates whether the model provides a new insight and is better than what was available previously".* Specifically, it is related with comparing & contrasting with alternative approaches (if any).

The last day on the site starts with a face-to-face meeting upon request to the participant, who requested to elaborate on the results of the model and possible suggestions sent by email. The availability of such an interview slot is announced to all participants and is performed optionally upon the request.Then, after this session, which is conducted with individual participants, all participants are kept together in the meeting room during the closing meeting. In that session, all general results on the charts, which include all participants' modeling pattern and cultures, with the general recommendations (e.g., set of common industrial practices) are planned to be presented.

The agenda template for all these planned processes is given in Table 15.

Table 15 Agenda for data collection, analysis and reporting process on the organization visit

2 days	~1 hour: Acquaintance and give presentation about "*Modeling Patterns and Cultures of Embedded Software Development Projects*" and terminology used ~ 30 minutes: Give the questionnaire separately, let them answer this structured part individually; but after completion do not collect the forms ~* hours (30 minutes per participant): Collect the forms by validating/confirming what each participant gives as answers in the questionnaire. Take notes in the questionnaire form, collect evidences *(In this semi-structured part, direct observations and improvisation play a critical role)*
	* *For Case Study A: (17 participants) -> 8,5 hours* *For Case Study B: (10 participants) -> 5 hours* *For Case Study C: (8 participants) -> 4 hours*
break Aim: Analyze the answers, evaluate them and apply the model. Subtask1: Investigate the current (via observation and interview) and according to our model (via model inputs) software modeling patterns and culture. Subtask2: Give what the similar profiles are doing while modeling Subtask3: Give recommendation for commonsense modeling practices. Subtask4: E-mail the results and evaluation form to the participants to evaluate the model	
3rd day	~2 hour: Interview with the participants, who want to meet individually about the results. ~2 hour: Show the general results on the chart. Repeat validatory questions about the model and make them elaborate their answers for both individual and project results ~10 min: Thank you and complete the session

5.2.2 *Selecting the Cases and Data*

It was intended to have the cases, in which certain characteristics of the software modeling may be considered to find variation points such as target domain (e.g., consumer electronics, defense & aerospace), business model (e.g., market or contract driven), customer (e.g., private, public, internal) [38, 40, 41, 83, 84, 88]. Therefore, based on differences instead of similarities, we selected our three cases and data as in Table 16. Notice that the interviewees in the case study are working in the same software development project with different SE roles.

Moreover, note that none of these interviewees participated in the previous interviews for this study; however, it cannot be guaranteed whether any of them participated the survey or not since the survey is completely anonymous.

Table 16 Case and data selection in multiple case studies

Case	Organization	Target Sector	Project Type	Business Model / Customer	Interviewee Size (Software project team distribution)	
A	Org1	Defense & Aerospace	Radar software	Contract-driven / Public Private	17	10 software developer | designer | architect 3 software tester 2 systems engineer 1 project manager 1 quality assurance engineer
B	Org2	Automotive & Transportation	Bus software	Contract-driven Market / Public Private	10	6 software developer | designer | architect 2 software tester 1 systems engineer 1 project manager
C	Org3	Consumer Electronics	TV software	Market / Private	8	5 software developer | designer | architect 2 software tester 1 project manager

In the next sub-sections, brief information about these companies will be given.

5.2.2.1 *About Org1 and Org2*

Org1 and Org2 operate independently but within the umbrella of a larger organization in the same company, whose product portfolio comprises communication and IT, radar and electronic warfare systems, weapon systems, air defense and missile systems, command and control systems, transportation, traffic and automation. The number of employees working in R&D engineering roles in this company is more than 3000. Having a CMMI-3 certification, both Org1 and Org2 are specialized in developing products with high-end software development techniques like agile programming, software product lines and reusable components.

As a specific target sector (i.e., defense & aerospace), Org1 is a global provider of advanced radar systems serving both military and civilian markets. For this study, a radar software project was chosen as Case Study A. The size of a typical software development team in Org1, which includes different SE roles is 15-25 people. In this study, Case Study A includes 17 interviewees, which covers all SE roles in this project.

The second case study (i.e., Case Study B) was also chosen from the same organization, but from different target sector (i.e., automotive & transportation). Org2 designs, develops and builds innovative custom solutions, subsystems & critical components for mobility of platforms on railways, roads and public networks. The size of a typical software development team in Org2, which includes different SE roles is 5-10 people and our case study includes 10 interviewees, which also covers all SE roles in this project. Among all other alternatives, bus software project was seen one of the best choices for the case study, since both Case Study A and B belongs to the same organization, but with different target sector, business model and possibly different software modeling approaches and practices.

5.2.2.2 *About Org3*

Org3, as a subgroup of one of the largest manufacturing companies in Turkey, operates in Consumer Electronics sector, which is a member of a consortium for several international R&D projects and continuously participates in numerous programs and initiatives. The number of employees working in R&D engineering roles in this company is about 800. For this study, a TV software project, which is Org3's one of well-known products, was chosen as Case Study C. The size of a typical software development team in Org3, which includes different SE roles is 5-10 people and Case Study C includes eight interviewees, which covers all SE roles in this project. This software group is mainly specialized in developing innovative and popular products with agile programming techniques.

5.2.3 *Collecting Evidence*

As mentioned in Section 5.2.1, the agenda template, which is given in Table 15, was applied in all three case studies. First of all, in order to give brief information about our model and get rid of any misunderstanding on the terminology used in this study, before the interview sessions, the presentation on "*Modeling patterns and cultures of embedded software development projects*" was given on the company's meeting room to all interviewees. Moreover, the evaluator informed all interviewees about the research before the interviews to gain initial trust and avoid unethical issues such as disclosing possible industrial secrets. This session took ~1 hour 15 mins and included a question & answer session.

Then, the questionnaire, which is the main data collection source for the first part of the interview, was distributed to the participants in order to obtain individual answers. The participants filled out the questionnaire alone; this part took ~30 minutes. After the completion of this first part, the forms were not collected.

After the lunch break, all forms were collected and one by one, semi-structured, face-to-face part of this study was carried out. The aim of this session was validating/confirming the participants' answers in the case of misunderstanding or unawareness of something (e.g., unawareness of software modeling usage or DSL usage during SDLC). In order to increase data consistency, besides the interviews, any extra source of information about modeling practices (e.g., any written material, medium type used) are analyzed during this process. Direct observation also helped to understand daily use of modeling and capture the details, which were not taken and clarified by the first round. During this session, the evaluator took notes on the questionnaire to collect evidence and found out some of the hidden characteristics such as DSL usage or sketching as ad-hoc (See [82] for the evaluator notes on the original questionnaire). Thus, multiple sources and cross checking of these data (e.g., what the evaluator observed and learnt during this semi-structured session) provided more robust conclusions. Note that the interviews were performed without any voice recorder since there are some confidentiality regulations for the first and second cases (i.e., in Org1 and Org2) and the participants in the third case (i.e., in Org3) did not want to it to be used.

For Case Study A, as the first case study, after the analysis of collected data and reporting the results, the evaluator sent the evaluation form (See APPENDIX F – Evaluation form template) to the participants to evaluate the model usefulness with respect to evaluation criteria [85]. In the email, which gave the results (i.e., the identification of modeling pattern and cultures and the suggestions), the participants were requested to fill out these forms (with their handwriting, if possible) before the closing meeting. However, since not all participants filled out this form before this session, in the second and third case studies, besides sending this form within result email, these forms were distributed as hard copy after the completion of the interview. Therefore, the evaluation form distribution and collection procedure varied. Accordingly,

majority of the participants (i.e., 72%) filled these forms with their handwriting, then they submitted these before the closing meeting session. Five of them (i.e., 14%) filled these forms in digitally and sent them to the evaluator via email just before the closing meeting (i.e., they did not use the form distributed by the evaluator). Minority of the participants (i.e., 14%), who did not have enough time to fill these forms until the closing meeting. However, this difference did not affect the overall evaluation for these participants since they elaborated their answers based on the results sent via email during closing meeting, which became more like a a brainstorming session since all participants were influenced by others' opinions.

Note that all data (both questionnaires with evaluator's notes and evaluation form of participants) were saved in case study database as a paper repository and then were digitized during the analysis when transcribed by taking the photo of each page (See [82]).

The analysis were done manually on all collected evidences (See [82]) during the break session before the closing meeting (for Case Study: A, it took 2 days, for the other case studies, it took one day). During the analysis, MAPforES was applied with the modeling stakeholder's characteristics to derive the modeling patterns and cultures both from the interview & observations results and also according to what the model predicted. Moreover, by querying these characteristics in the survey result database, MAPforES presented what the similar demographics do as commonsense industrial modeling practices (e.g., according to SE role, target sector, project size, etc., the model increased the awareness of commonsense practices such as the modeling languages specific to the target sector like AADL, Markov Chain Modeling Language) [82].

5.2.4 *Results*

Applying MAPforES as seen from Table 17, all case study results are depicted according to their ratio (as percentage values) by comparing with all prior works (i.e., survey and interview). Note that a relation between software modeling practices versus target sector of the products has already been identified; and this "target sector" is one of the characteristics of diagram development and usage in the embedded software development (See Section 4.2). According to [70], Defense & Aerospace sector is the most model-driven user sector; and Automotive & Transportation is the second one. The result of this multiple case studies show similar results when "target sector" of products are considered. Moreover, although the participant numbers are different for survey, interview and case studies (i.e., 657, 53 and 35 respectively), the results provide insight for the modeling patterns and cultures in the embedded software industry, which MAPforES identified.

The results also showed that there is a noticeable percentage of "unaware"s participants of modeling or MDE (i.e., ~11%) in the embedded software community. These "unaware"s results were identified during the direct observation or face-to-face semi-structured interview via question & answer session (i.e., after the completion of structured part of the questionnaire while taking the model parameters as input). In the following tables, all three case study results are given in individual tables, in which these "unawares" are also depicted (Note that the abbreviations used in the tables are given in Table 18).

Table 17 Case study results summary: comparison with survey and interview with respect to pattern & culture percentages

Patterns & Cultures				% in Survey Results (657)		% in Interview Results (53)		% in Case A (17)		% in Case B (10)		% in Case C (8)		% in Case Studies (35)	
model-driven	3.3	With DSL-like	Auto-generated	15,4	29,5	15,1	32,1	11,8	35,2	20	30	12,5	25	14,3	31,4
	3.2	Without DSL-like		6,5		7,5		5,8		10		-		5,7	
	3.1	Limited		7,6		5,6		11,8		-		-		5,7	
	3.x	Unaware of MDE (3.1 | 3.3)		-		3,7**		5,8**		-		12,5**		5,7	
model-based	2.2	Prescriptive	Prescripted	25,2		18,9		23,5		20		25		22,9	
	2.1	Descriptive	Archived	12,7	16,2	11,3	15	5,8	17,6	10	20	12,5	25	8,6	20
sketching	1.3	Archived		3,5		3,7		11,8		10		12,5		11,5	
	1.2	Selective	Formalized	13,6		13,2*		11,8		10		12,5		11,5	
	1.1	Ad-hoc	Performed	4,5		3,7*		5,8		-	10	-	12,5	2,9	8,6
	1.x	Unaware of modeling (1.1|1.2)*		-		5,6		-		10		12,5		5,7	
none	0.1	Bad experienced	None	11		7.5	11,2	5,8	5,8	-	10	-	-	2,9	5,7
	0.0	Not experienced				*3.7*		-		*10*		-		*2,9*	

** 1.x pattern might be either 1.1 or 1.2; hence its value makes "performed" and "formalized" culture increase.*
*** 3.x pattern might be either 3.1, 3.2 or 3.3; but since all of them are in "auto-generated", no need for further analysis as in 1.x pattern.*

Table 18 Abbreviations used in Table 19, Table 20 and Table 21

Position	Software Developer/Programmer	Dev		Systems Engineer	Sys		Software Tester	Tstr
	Software Designer	Desg		Project Manager	PM			
	Software Architect	Arch		Quality Assurance Engineer	QA			
University Degree	Computer Science	CS		Computer Engineering	CENG		Software Engineering	SE
	Electrical/Electronics Engineering	EE		Information Systems	IS		Mechanical Engineering	ME

The results in Table 19 show that different stakeholders in the same SE roles may have different patterns and cultures. Although participant#11, participant#12 and participant#13 are in software tester role in the same project, their patterns and cultures are different. Participant#11 tests UI application modules of the radar software project and mainly writes UI test simulators in Java or C++, which he described the simulators to be developer as "low" in terms of hardware closeness[17]. He also used their own MDE tool (which is based on DSML) to generate test cases as MBT. Therefore, he benefits from both UML diagrams besides DSL-like diagrams during analysis, design and test phases of SDLC. Participant#12 tests the communication protocol parts and message interfaces between middleware and DSP modules of the radar software, which are deployed in the main processor card (not in PC). She described the simulators she developer as "medium" in terms of hardware closeness. She does not use any model-driven techniques although she took some modeling languages courses during her MSc in CENG. She benefits from sequence diagrams, use case diagrams and communication diagrams during analysis and test phases of SDLC. On the other hand, participant#13, whose academic background is different from other testers (i.e., he is an EE graduate and did not take any SE courses on modeling) tests DSP algorithms and he does not use any programming language directly related with modeling. Besides, he mentioned that he never uses any digital medium (e.g., PC) while modeling although he limitedly uses some use case or sequence diagrams just to communicate with other colleagues without archiving them (e.g., lifespan of these diagrams are very less since they are soon discarded after conversation). As seen, although participant#11, participantt#12 and participant#13 are in the same project with the same SE role, since their task/responsibility are different (e.g., testing different modules of the same software), their modeling characteristics, hence their modeling patterns are different. Similar situations happened for the same SE roles (e.g., developers or systems engineers), which shows the difference on modeling patterns is caused not only by project or role, but also the tasks and responsibilities of that particular participant in that role.

Moreover, partipant#8 is at pattern 3.3 according to model, but during the second round of the interview, face-to-face conversation revealed that he is one of the "unawares" of MDE (as participant#28 in Case Study C). These "unawares" filled the questionnaire as they have benefitted from automatic code generation or documentation generation with sketch and UML usage. However, it was observed that they actually used DSL-like modeling languages, which categorizes them as pattern 3.x. For further details of participants' response, see [82].

[17] what meant by hardware closeness is that firmware or DSP software is closer to hardware than UI or middleware software

Table 19 Case study A results, Defense & Aerospace sector, Radar software project

Participant#	Position	Degree		Experience (in years)		PL & HW closeness	Modeling Language(s)	Modeling Pattern		Modeling Culture	
		Academic	University	Work	Modeling			Observation / Interview	According to Model	Observation / Interview	According to Model
6	Dev, Desg, Arch	MSc	CS, IS	10+	10+	C, C++, Java Medium	Sketch, UML, UML profiles, DSL	3.3	3.3	Auto-generated	Auto-generated
11	Tstr	MSc	CENG	10+	10+	Java, C++ Low	Sketch, UML, DSL	3.3	3.3	Auto-generated	Auto-generated
3	Dev, Desg, Arch	MSs	EE, CENG	10+	10+	C, C++ Medium	Sketch, UML	3.2	3.2	Auto-generated	Auto-generated
9	Dev	MSc	EE	10+	6-10	C, MATLAB High	Sketch, UML, MATLAB	3.1	3.1	Auto-generated	Auto-generated
14	Sys	MSc	EE, SE	6-10	6-10	MATLAB Low	Sketch, UML, MATLAB, SysML	3.1	3.1	Auto-generated	Auto-generated
8	Dev, Desg	MSc	EE, CENG	10+	10+	C, C++ Medium	Sketch, UML, DSL*	3.x	3.3	Auto-generated	Auto-generated
1	Dev, Desg	MSc	EE, IS	10+	10+	C++ Medium	Sketch, UML	2.2	2.2	Prescripted	Prescripted
7	Dev	BSc	EE	10+	10+	C, C++ Medium	Sketch, UML	2.2	2.2	Prescripted	Prescripted
10	Dev	BSc	EE	6-10	6-10	C High	Sketch, UML	2.2	2.2	Prescripted	Prescripted
12	Tstr	MSc	EE, CENG	10+	10+	C++ Medium	Sketch, UML	2.2	2.2	Prescripted	Prescripted
17	QA	MSc	CENG	10+	10+	BPEL Not applicable	Sketch, UML, BPML	2.1	2.1	Archived	Archived
2	Dev	BSc	EE	6-10	2-5	C High	Sketch, UML	1.3	1.3	Archived	Archived
16	PM	MSc	ME	10+	10+	Not applicable	Sketch, UML	1.3	1.3	Archived	Archived
4	Dev	BSc	EE	2-5	2-5	C High	Sketch, UML	1.2	1.2	Formalized	Formalized
13	Tstr	BSc	EE	6-10	2-5	Not applicable	Sketch, UML	1.2	1.2	Formalized	Formalized
15	Sys	Bsc	EE	10+	6-10	Not applicable	Sketch	1.1	1.1	Performed	Performed
5	Dev	MSc	EE	10+	10+	C Very high	-	0.1	0.1	None	None

In Case Study B, almost all results after observation/interviews are compatible with the model found out except participant#21. This participant (i.e., PM, whose university degree is ME and did not take any SE courses related to modeling) filled the questionnaire as "modeling experience is "0" and "never" using software modeling (either informal or formalized). However, after face-to-face interview and observations, it was noticed that he thought that modeling is limited to formal UML diagrams; but in fact, he uses sketches on either paper and whiteboard during the meetings with the systems and software teams. He mentioned that he uses some sketches (via some boxes and lines) to understand a problem or process at an abstract level. Therefore, he is one of the "unaware"s of modeling with a hidden pattern (i.e., pattern 1.x) [82].

The same "unawreness" of modeling occurred in Case Study C, in participant#32, who has again PM role. After the interview, it was realized that he indeed uses sketching on whiteboard in an ad-hoc pattern during analysis phase of SDLC [82].

As our AR project [5] showed that in the same software development project, different layers of the software might use different programming languages (e.g., DSP team uses "C", middleware team uses "C++", and UI team uses "Java") and their modeling practices are different. We also previously found out that that software's closeness to hardware affects modeling rigor and its relevant practices (e.g., modeling languages, diagram types, etc.) via programming language selection [75]. An interesting finding from all case studies is whenever programming language used for modeling goes from high level to low level (e.g., from Java, C++ to C, or not applicable), the use of modeling decreases (e.g., the relation between PL & HW closeness column vs modeling patterns column). These case studies confirmed our previous findings.

As to the relationships between university degrees and modeling cultures, there is not any participant in "Auto-generated" culture, whose university degree does not include any combinations of Computing Disciplines (e.g., CS, CENG, SE or IS) except participant#9 and participant#28, who use MDE limitedly (e.g., without code generation or MBT) and graduated from EE (e.g., participant#9 uses MATLAB for model simulation; and participant#28 uses MDE for documentation generation [82]). We have already found out that educational skill set affects where/how the stakeholder learned software modeling, hence modeling approaches and its relevant practices through modeling experience. For example, a stakeholder, who graduated from EE, have learned software modeling after graduation with formal corporate training, or on his/her own; however any stakeholder who graduated from a Computing Discipline has learned software modeling at the university from SE courses [75]. These case studies also confirmed these findings since they showed there is a relation between the academic background and the modeling approaches if the task/responsibility of the stakeholders does not force him/her to do specific modeling practices.

Table 20 Case study B results, Automotive & Transportation sector, Bus software project

Participant#	Position	Degree		Experience (in years)		PL & HW closeness	Modeling Language(s)	Modeling Pattern		Modeling Culture	
		Academic	University	Work	Modeling			Observation / Interview	According to Model	Observation / Interview	According to Model
23	Dev, Des	MSc	CENG	10+	10+	C++ Medium	Sketch, UML, DSL, (AUTOSAR)	3.3	3.3	Auto-generated	Auto-generated
20	Dev, Des, Arch	MSc	CS, SE	10+	10+	C, C++ Medium	Sketch, UML, DSL,(AUTOSAR)	3.3	3.3	Auto-generated	Auto-generated
18	Dev, Arch	PhD	EE, CENG	10+	10+	C++ Low	Sketch, UML	3.2	3.2	Auto-generated	Auto-generated
27	Dev, Arch	MSc	CENG	10+	10+	C, C++ Medium	Sketch, UML	2.2	2.2	Prescripted	Prescripted
22	Tstr	BSc	CENG	6-10	6-10	C++, Java Medium	Sketch, UML	2.2	2.2	Prescripted	Prescripted
19	Sys	MSc	EE, CENG	6-10	2-5	Not applicable	Sketch, UML	2.1	2.1	Archived	Archived
26	Dev	BSc	EE	6-10	2-5	C High	Sketch, UML	1.3	1.3	Archived	Archived
24	Dev	BSc	EE	10+	6-10	C High	Sketch, UML	1.2	1.2	Formalized	Formalized
21	PM	BSc	ME	10+	>0*	Not applicable	Sketch*	1.x	1.1	Performed	Performed
25	Tstr	BSc	EE	2-5	-	C Very high	-	0.0	0.0	None	None
*Totally, 113 years of software development experience. *: this information was obtained during the interview after the participant's completion of the questionnaire*											

Table 21 Case study C results, Consumer Electronics sector, TV software project

Participant#	Position	Degree		Experience (in years)		PL & HW closeness	Modeling Language(s)	Modeling Pattern		Modeling Culture	
		Academic	University	Work	Modeling			Observation / Interview	According to Model	Observation / Interview	According to Model
30	Tstr	MSc	CS, CENG	10+	10+	C, C++, Python Medium	Sketch, UML, DSL (Markov Chain)	3.3	3.3	Auto-generated	Auto-generated
28	Dev, Arch	MSc	EE	10+	10+	C++, Python Medium	Sketch, UML, DSL (own, Doxygen)*	3.x	3.1	Auto-generated	Auto-generated
35	Dev, Arch	MSc	CS, CENG	10+	10+	C++, Medium	Sketch, UML	2.2	2.2	Prescripted	Prescripted
29	Dev	MSc	EE, CENG	10+	6-10	C++, Medium	Sketch, UML	2.2	2.2	Prescripted	Prescripted
31	Dev	BSc	EE	6-10	6-10	C High	Sketch, UML	2.1	2.1	Archived	Archived
34	Tstr	BSc	EE	6-10	2-5	Not applicable	Sketch, UML	1.3	1.3	Archived	Archived
33	Dev	BSc	EE	6-10	2-5	C Very high	Sketch, UML	1.2	1.2	Formalized	Formalized
32	PM	BSc	EE	10+	-	Not applicable	Sketch*	1.x	1.1	Performed	Performed

Totally, 95 years of software development experience.
**: this information was obtained during the interview or direct observation after the participant's completion of the questionnaire*

After the closing meeting, all evaluation forms (See APPENDIX F – Evaluation form template) were analyzed. Note that this evaluation form (which is also given in Table 22) were prepared to address RQ3 with respect to Table 14 criteria. Moreover, during the closing meeting, all participants were influenced by others' opinions. In this process, some stakeholders, who explicitly gave opinions about the model and their modeling practices, indirectly encouraged other stakeholders to share their comments, complaints and also challenges on this topic. This session became like a brainstorming on modeling patterns and cultures of embedded software development projects.

Table 22 Evaluation questions to achieve validation criteria

#	Question	Addressed RQ	Validation
1	When you think about the presentation you took about *"Modeling patterns and cultures of embedded software development project", does our model really reflect your current modeling pattern and culture?* In other words, did this produce expected and relevant results for you?	RQ3.1 RQ3.2	Result & Utility
2	In that sense, do you think that the model is helpful? Please elaborate your answer.	RQ3.1 RQ3.2	Result & Utility
3	Have you ever been experienced or used such a model before? In other words, do you think that this model is better than what was available previously or not?	RQ3.2	Comparison
4	Do you think that learning what your competitors (i.e., similar demographics) are doing while modeling might affect your future modeling practices? Please elaborate your answer.	RQ3.2	Utility
5	Do you think that the recommendations, which our model gave you, is useful or not? Please elaborate your answer.	RQ3.2	Utility

According to these evaluation forms collected from the participants, the model reflected the expected, relevant and useful results with respect to validation criteria. All qualitative and quantitative data gathered through these forms (which we will discuss for all questions) and the attitudes of the participants during the closing meeting have shown that the model has been useful in creating awareness and guidance on software modeling [82]. Therefore, MAPforES enabled and guided the modeling stakeholder's process and tool improvements by referencing to a set of commonsense industrial practices.

Although the evaluation form is in English, four participants (~11%) answered in Turkish. Note that if the participant's answer is in English, the phrase is not corrected even if it might be grammatically incorrect on the original forms. However, we have corrected these sentences in this paper to improve their understandability with additional words added for clarity, when necessary, shown in brackets. Due to space constraints, the selected evaluations to each question with verbatim quotes taken from the original evaluation forms will be given in the following (See [82] for all evaluation responses).

Evaluations for the first question: All responses mentioned that the model produced expected and relevant results (which means that the results of RQs were satisfactory). One of the evaluation form includes: *"In fact, I really did not know whether I have been modeling; but in fact, I now realized that I have been a sketcher for more than 10 years; yes I am a modeler but part of a 'performed'[culture]".* A project manager indicated the presentation and recommendations benefits by writing *"Before presentation I didn't think modeling was important for me; but now I can say that at least I will try to investigate [further] these recommendations".* All participants explicitly stated that the model satisfied them in term of proposed benefits [82].

Evaluations for the second question: The second question, which investigates the usefulness of the model, reflected the benefits of the model. Almost every participant (94%) mentioned

about the necessity to understand different modeling patterns and cultures so that the model might guide the modeling stakeholder to obtain commonsense modeling practices. Accordingly, they think that the model provides a common language among modeling stakeholders with the already achieved benefits of embedded software modeling community's experiences.

Depending on the modeling stakeholder position, the evaluations varied. For example, one of the software developers in first case study states that "*knowing the characteristics of what I am modeling is helpful to put my modeling approaches into [perspective in terms of] embedded software industry categorization. Learning the importance of DSL in embedded software industry pushes me to nvestigate further a cost-effective and domain specific (defense) solution".* Another participant (i.e., systems engineer) wrote on the evaluation form that "*the model is helpful to understand different modeling approaches of different roles such as sw developers, systems engineers (such as me) and even PMs. As far as I understood, all of their approaches might be the 'best', so there is no just one best!".*

One of the benefits of the model is making the stakeholders aware about their modeling practices. One of the software developer in the third case study said: *"We now know that we are using DSL in fact :)*". There are many participants (83%), who mentioned about that the presentation given before the interview was very beneficial since knowing the relations between the characteristics of a diagram would give practical benefits to modeling stakeholder.

Evaluations for the third question: All answers for the third question are *"No"*. There is not any participant, who has experienced such a model before and there is no alternative approach. Therefore, the model provides a new insight.

Evaluations for the fourth question: As mentioned, the model (depending on survey result data) presented a modeling practices set by querying the similar profiles' modeling practices. For example, a participant presented his/her characteristics of modeling (e.g., SE role, target sector, project size, etc.) and the evaluator reported the similar profiles' modeling practices to increase the awareness of them (e.g., the modeling languages specific to the target sector such as AADL, Markov Chain Modeling Language or modeling tools, which might be free (open source)). By this way, the model guided process and tool improvements for modeling by referencing to a set of commonsense industrial practices [82]. According to the majority of participants (i.e., 74,2% of participants used "useful" explicitly in their evaluation forms), this set is very useful so that their modeling practices might be affected according to these suggestions. *"Learning what the similarly profiled [embedded software practitioners] are doing is useful to analyze the approaches [before embarking on a project with modeling]; it will save time"* or *"Knowing alternative practices (for example modeling tools) might affect our practices. If they are cheaper than what we use, we, of course, will use and apply these practices in future"* are some example quotes from the participants.

Although the attitudes towards these suggestions were always positive; some participants mentioned about some organizational and managerial issues. One of the software developers in the first case study states: *"We have an organizational decision to use a modeling tool, I don't know whether we can change this; but the managerial decision on that tool might be affected if there are cheaper alternatives"*. Another software architect in the same software development team wrote in the evaluation form as *"Of course, 'stand on the shoulders of giants' :). If some of their choices [in modeling approaches and tools] fit our organization, why not?".* One participant stated: *"Yes, I believe that our competitors' modeling ways could be a soruce for inspiration about future projects, but I am not sure about my managers' possible concerns about what our competitors are doing; and they have the last word".*

The set of modeling languages used based on the stakeholder position was also appreciated by the participants. One of the systems engineers stated: *"In fact there are not many systems engineer in the industry, therefore it is very interesting to learn what they do. Specifically I want to learn more about SysML"*. A software developer in the first case study also stated: *"I don't know about such DSL usage in our industry; I should analyze some of them like MARTE and EAST-ADL."*. One of the project managers commented that *"Being PM, learning the other PMs' modeling usage is very interesting; perhaps I should analyze some BPM diagrams to get some benefits"*.

Evaluations for the fifth question: Almost every participant's answers (91%) satisfied utility validation criteria for this question, which investigates whether the model is useful or not. By using the derived chart for the modeling patterns and cultures with the corresponding characteristics of a diagram and taking prior study results, the recommendations is useful for commonsense modeling practices depending on the specific characteristics (e.g., motivation and purpose).

One of test engineers states: *"Developing company specific [domain specific] tool according to our needs is always a planned action for our test department. Perhaps, [based on the feedbacks] from this study, we can accelerate this process and fully automate all our testing procedure. By this way, all testers might be in the same pattern according to your model"*. One of the software developers, whose pattern is 2.2 (i.e., using prescriptive modeling but not model-driven techniques) said: *"The recommendations are useful with respect to [being aware of] DSL and having own modeling tool. Perhaps, we can try model-driven techniques by comparing pros and cons".* A software developer in the second case study said: *"I think they [the recommendations] will be useful after analyzing the suggested modeling tools and DSLs further (mainly on Papyrus, eclipse-based tools and automotive domain specific DSLs)"*

Moreover, giving these recommendations explicitly (i.e., in a written format) made some participants aware about an easy and straightforward modeling task to get practical benefits. A project manager stated: *"Just taking a photo of whiteboard screen and archiving it is a very easy and effective solution. I am wondering why I did not do that until now".*

The same situation encountered in the answers to fourth questions' responses about organizational decision-making issue was also encountered here. "*Yes, [the recommendations are useful] but since I am not a decision maker, I will also forward your email to my manager"* or *"After trying and experiencing the suggestions, I can [personally] use them, therefore it might affect according [how we work] based on the results of their feasibility analysis; but for my team, I should inform my technical lead"* are some example quotes about this challenge.

Qualitative data gathered through the evaluations has shown that the MAPforES model has been useful in creating awareness and guidance on software modeling in embedded software practitioners.

5.2.5 *Threats to Validity*

Validity and reliability of the research are important factors for qualitative research. *"Quality checks to ensure that the case study is done in a proper manner need to be performed to prevent subjective interpretations"* [38]. As suggested by [38], the draft case study design was reviewed by two academicians and three embedded software professionals. By this way, Table 15 was modified and before the company visit, the agenda template was finalized. During the interviews, the actual (performed) progress of the case study against the planned progress (i.e., the agenda) was reviewed to determine if there are any significant differences. Notice that this agenda was almost always applied with some exceptions (See Section 5.2.3). Moreover, the

evaluator (during the second round of the interview) asked the interviewee to confirm and validate what he/she gave as responses in the questionnaire. This helped to ensure that the interview data provides a fair representation of the interviewee's opinions with correct answers.

Since the relation between the software and the hardware, which this software is running on, is important in the embedded software development, we should note that these case studies do not include any hardware, which is at architectural design or development stage (e.g., the hardware is robust) and there is no specific challenges or problems due to these hardware platforms.

In our study, the following aspects are addressed [38]:

Construct Validity: Construct validity is concerned with the correctness of the interpretation and the theoretical constructs [71]. In this research, multiple sources of evidences were used with case study strategy. All evidences were collected in questionnaires, written notes after interviews and direct observations; and then were kept in a technical report [82]. As mentioned, during the second round of the interview, the evaluator confirmed what the interviewee gave as responses in the questionnaire, which ensured the correctness of collected data.

Internal Validity: In order to mitigate this threat [71], we focused on the study design and checked whether the results are consistent with the data. The case studies reported here cannot be considered as a controlled experiment; however the similar profiles and characteristics of diagram development and usage might be used for pattern matching with further case studies to eliminate any bias. During the first part of the interview, all participants filled out the questionnaire individually and separately so that the interviewer prevented answers of a participant to be influenced by others [89]. By this way, the interviewer avoided any information sharing between interviewees. Note that awareness of modeling or MDE is critical to feed the model with correct data. Since there is no culture difference in case of "pattern 3.x" (i.e., "pattern 3.x" might be "pattern 3.1" or "pattern 3.3" in practice, but all of them are in "auto-generated" culture), the model gives the relevant result for the corresponding culture. However, in the case of "pattern 1.x", if the input is "no modeling", the corresponding pattern would be incorrect. As seen, awareness of modeling, hence data quality, is critical to have relevant results. Moreover, note that none of these software professionals in multiple case studies participated in the previous interviews; however, it cannot be guaranteed whether any of them participated the survey or not. Nevertheless, note that even if they have participated in the survey, when the survey participant number is compared to the survey (e.g., ~5.5%), a threat to internal validity would be limited.

External Validity: The generalizability of the results is focused to mitigate this threat [71]. The limited size and complexity of the case studies restrict the generalizability of our results. Although three cases and participants were selected intentionally (See Section 5.2.2) with variation points (e.g., target domain, position, academic background, experience, hardware closeness), it cannot be stated whether the software team was representative of other embedded software development projects. However, all three cases have similar results and by applying the model in more case studies and projects, the generalizability may be improved.

Reliability: Reliability focuses on replicability of the results by other researchers. This study has a case study protocol and case study database, which were documented and archived systematically so that the replicability and repeatability of the operation of the case study has been ensured.

CHAPTER 6

CONCLUSION

This chapter presents the summary, contributions and future research directions of this study.

6.1 Summary and Concluding Remarks

This dissertation identified and defined different modeling patterns and cultures of embedded software development projects. By understanding the current state-of-practice of modeling and investigating the characteristics of modeling in the embedded software development projects, this study found out the significant parameters to characterize modeling patterns and cultures. In doing so, this study constructed a characterization model called MAPforES, which identifies and defines the modeling patterns and cultures in embedded software development by also guiding modeling stakeholders for a set of common industrial practices while modeling.

Specifically, the goal of the study has been achieved as follows:

At the beginning of the research, a SLR is performed to understand the related work on software modeling and its results were used in later phases of this study (e.g., during designing survey questions and while creating the conceptual model for software modeling).

We then conducted a survey to determine the state-of the-practices of software modeling and MDE in embedded software development with its achievements and challenges.

The survey data, which identifies to what degree, why and how software modeling and MDE is used, was insufficient to answer some qualitative questions (e.g., why they do not use software modeling or what are their specific modeling challenges) and there was a need to conduct in-depth interviewing to capture some detailed, rich contextual analysis concerning the everyday practical realities of software modeling in embedded industry to better characterize modeling patterns and cultures. Therefore, significant characteristics of software modeling were investigated. A conceptual model for the development and usage for software modeling, which is enriched by expert opinions via semi-structured interviews was also presented to better characterize these significant characteristics. These characteristics and the relations between them would be an input to identify and define the modeling patterns.

After investigating the relations between these characteristics, the modeling patterns were identified and categorized in two iterations. During this iterative process, firstly, a preliminary model was created by using all prior findings with survey data analysis. Then, this preliminary version was validated and improved with case studies via semi-structured interviews. After grouping resultant patterns according to their characteristics, the modeling cultures in the embedded software development projects were defined and further refined by expert opinions.

After identifying the patterns and cultures in the embedded software industry with their characteristics based on prior findings (i.e., survey and interview results data), MAPforES model was created. This model identifies and defines a modeling stakeholder’s pattern and culture as commonsense practices by presenting what the similar profiles in the embedded

domain is doing while modeling. Note that the survey results were used to establish a commonsense practices database (i.e., a set of common industrial practices on software modeling). By this way, besides identifying and defining the current pattern and culture, this model identifies the widespread modeling practices (e.g., process and tool) in embedded software development projects by referencing to a set of commonsense industrial practices.

In order to observe the usefulness of MAPforES, we successfully applied this characterization model for three cases (i.e., organizations) based on different subsectors of embedded software industry. We observed that the industrial context reflects what we presented in the characterization model. The results elaborated and validated our findings on modeling patterns that focus on all significant characteristics of modeling (e.g., not only "modeling rigor" but also "purpose", "medium type used", "stakeholder profile", etc.) and fills the gap of what constitutes "software modeling" (e.g., including DSLs and other formal languages beyond UML usage).

As Heldal et al. says, different units within the same company might use different modeling approaches [15]. One-step beyond what they said, we found that even in the same software development project, the same SE roles, might use different modeling practices depending on their tasks and responsibilities for different purposes in different phases of SDLC. We found out that organizations may need different modelling patterns for different projects or even for different individual roles within projects. MAPforES provides an approach to provide feedback to modeling stakeholders thereby creating insight for individuals. The usage of the model has the potential to overcome one of the most significant difficulties of top-down organizational process improvement models; enabling everyone to contribute [90, 91].

We found out that the model is useful since the participants explicitly mentioned about their satisfaction [92] in creating awareness and referencing to a set of commonsense industrial modeling practices. Qualitative data gathered through the evaluations has shown that all participants (100%) thought that the MAPforES is conceptually insightful and the majority (i.e., 74.2%) used "useful" explicitly in their evaluation form.

MAPforES can be applied with a moderate amount of effort (i.e., ~2 hour per the modeling stakeholder) and its benefits will easily overweight its costs as the improvements in individual processes will be accumulated in all the projects to be implemented after that point in time.

MAPforES is a complementary model for process improvement approaches such as CMMi and SPICE [93, 94]. Identifying modelling patterns of individuals and/or projects before an organizational assessment of software modeling practices may be useful in pinpointing the potential threats for institutionalization such as the diversity of techniques utilized. The results would also be beneficial to identify the common techniques for different purposes used, thereby to determine the best standardization approaches. Two organizations in the case studies have CMMi certification and the participants found the model useful, which has increased both their awareness of their own and similar demographics' modeling practices.

6.2 Contribution

The main contribution of this dissertation is the identification of modeling patterns and cultures by investigating the significant characteristics of modeling in embedded software development projects. In doing so, to utilize these identifications, a characterization model (MAPforES) to identify and define modeling patterns and cultures in embedded software development projects with a set of commonsense industrial modeling practices is defined and implemented for software organizations.

Its theoretical contribution is lying in identifying the current state-of-the-practices of modeling and MDE (e.g., with systematic review, survey results) and in characterizing software modeling (e.g., conceptual model of development and usage of software modeling, and the characteristics of modeling) in embedded software development projects.

For its practical contribution, the resulting artifacts of this thesis can be used by any modeling stakeholder in the embedded software industry, with a variety of different SE roles from software developer/programmer to tester, who would benefit from commonsense modeling practices depending on their profiles to achieve an effective modeling approach.

Note that all case study stages in this dissertation (e.g., survey, interviews, etc.) were conducted in both local and global scale with high number of practitioner participants by focusing all aspects of software modeling usage and practices in the world-wide embedded software industry, which is also important with respect to the novelty and validity of this research.

The significance of this study is to being the first research in the literature, which defines and characterizes the modeling patterns and cultures in the embedded software development project by focusing on all significant characteristics of modeling and filling the gap of what constitutes "software modeling" (e.g., including other formal languages beyond UML usage). Additionally, the model presented here, MAPforES, is also known to be the first wide-coverage model, which not only identifies patterns and cultures of the modeling stakeholder, but also enables process and tool improvements for modeling by referencing to a set of commonsense industrial practices in embedded software development projects.

6.3 Future Research Directions

The general validity of the conclusions were restricted by the limited number (i.e., three) of case studies. This research can be enriched with more case studies with different characteristics of software modeling. Such a further study could strengthen the validity of the model.

In the multiple case studies, to take the modeling characteristics of the participant, a questionnaire was used. A recommendation system using AI techniques might transform a more-costly-to-implement technique such as a questionnaire into a virtual assistant for project and program managers implementing policies on software modeling based on a model such as MAPforES of community experience.

Based on the technology acceptance model (TAM) [95], there might be some reasons, which cause people to accept or reject technologies and related practices. Davis et al., describes *perceived usefulness* as *"people tend to use or not use an application (technology or practice) to the extent they believe it will help them perform their job better"* [95]. They continue that although the potential users think that this technology is useful, they might think that its practices is too hard to use and that the benefits of usage are out-weighted by the effort of using it [95]; which is determined by its *perceived ease of use*. Moreover, it is also claimed that the limited adoption of modeling and MDE is due to a variety of social and technical factors [20]. Throughout this study, during the semi-structured interviews, similar situations were encountered in embedded software development projects for modeling (e.g., tools, languages, stakeholders). It is planned to study these factors that influence the adoption of various modeling patterns, specifically the effect of understandability and organizational resistance [96].

MAPforES is the first wide-coverage model of software modeling characteristics for embedded software sector built on extensive input from the industry. The work presented in this article complements the model development effort by applying the MAPforES model successfully in three embedded software projects from two organizations. We hope MAPforES and its applications in the field will establish a useful baseline from which individual and organizational process improvement studies in embedded systems modeling can grow from.

REFERENCES

[1] C. J. Ebert, Capers, "Embedded Software: Facts, Figures, and Future," *IEEE Computer Society,* vol. 42, pp. 42-52, 2009.

[2] J. Schäuffele and T. Zurawka, *Automotive Software Engineering: Principles, Processes, Methods, and Tools*: SAE International, 2005.

[3] P. G. Paulin, C. Liem, M. Cornero, F. Nacabal, and G. Goossens, "Embedded software in real-time signal processing systems: application and architecture trends," *Proceedings of the IEEE,* vol. 85, pp. 419-435, 1997.

[4] M. Lettner, M. Tschernuth, and R. Mayrhofer, "A critical review of applied MDA for embedded devices: Identification of problem classes and discussing porting efforts in practice," in *14th International Conference on Model Driven Engineering Languages and Systems, MODELS 2011* vol. 6981 LNCS, ed, 2011, pp. 228-242.

[5] D. Akdur and V. Garousi, "Model-Driven Engineering in Support of Development, Test and Maintenance of Communication Middleware: An Industrial Case-Study," in *International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2015.

[6] M. A. Vega-Rodríguez, "Design space exploration of embedded systems: A view from diverse domains," *Journal of Systems Architecture,* vol. 59, pp. 1113-1114, 2013.

[7] B. Graaf, M. Lormans, and H. Toetenel, "Embedded software engineering: the state of the practice," *Software, IEEE,* vol. 20, pp. 61-69, 2003.

[8] M. Broy, "Challenges in automotive software engineering," presented at the Proceedings of the 28th international conference on Software engineering, Shanghai, China, 2006.

[9] A. S. Krishna, E. Turkay, A. Gokhale, and D. C. Schmidt, "Model-driven techniques for evaluating the QoS of middleware configurations for DRE systems," in *Real Time and Embedded Technology and Applications Symposium, 2005. RTAS 2005. 11th IEEE*, 2005, pp. 180-189.

[10] C. Walls, *Embedded Software*, Second Edition ed. Oxford: Newnes, 2012.

[11] N. Wang, C. D. Gill, D. C. Schmidt, A. Gokhale, B. Natarajan, J. P. Loyall, *et al.*, "QoS-enabled Middleware," in *Middleware for Communications*, ed: Wiley, 2004.

[12] E. Arisholm, L. C. Briand, S. E. Hove, and Y. Labiche, "The impact of UML documentation on software maintenance: an experimental evaluation," *Software Engineering, IEEE Transactions on*, vol. 32, pp. 365-381, 2006.

[13] D. Thomas, "MDA: revenge of the modelers or UML utopia?," *Software, IEEE*, vol. 21, pp. 15-17, 2004.

[14] W. J. Dzidek, E. Arisholm, and L. C. Briand, "A Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance," *IEEE Transactions on Software Engineering*, vol. 34, pp. 407-432, 2008.

[15] R. Heldal, P. Pelliccione, U. Eliasson, J. Lantz, J. Derehag, and J. Whittle, "Descriptive vs prescriptive models in industry," presented at the Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, France, 2016.

[16] M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," *Synthesis Lectures on Software Engineering*, vol. 1, 2012.

[17] J. Cabot. (2009). *Relationship between MDA,MDD and MDE*. Available: http://modeling-languages.com/relationship-between-mdamdd-and-mde/

[18] (2016). *Career Award Talk - Bran Selic*. Available: https://www.youtube.com/watch?v=9qPbGksB3d4

[19] "Diagram," ed: Cambridge Dictionary, 2017.

[20] J. Hutchinson, J. Whittle, and M. Rouncefield, "Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure," *Science of Computer Programming*, vol. 89, Part B, pp. 144-161, 2014.

[21] P. Liggesmeyer and M. Trapp, "Trends in Embedded Software Engineering," *Software, IEEE*, vol. 26, pp. 19-25, 2009.

[22] A. Gokhale, D. C. Schmidt, B. Natarajan, J. Gray, and N. Wang, "Model Driven Middleware," in *Middleware for Communications*, ed: Wiley, 2004.

[23] R. France and B. Rumpe, "Model-driven Development of Complex Software: A Research Roadmap," presented at the Future of Software Engineering, 2007.

[24] G. Karsai, S. Neema, and D. Sharp, "Model-driven architecture for embedded software: A synopsis and an example," *Science of Computer Programming*, vol. 73, pp. 26-38, 2008.

[25] H. Espinoza, D. Cancila, B. Selic, and S. Gérard, "Challenges in Combining SysML and MARTE for Model-Based Design of Embedded Systems," in *Model Driven Architecture - Foundations and Applications*. vol. 5562, R. Paige, A. Hartman, and A. Rensink, Eds., ed: Springer Berlin Heidelberg, 2009, pp. 98-113.

[26] B. Hailpern, Tarr, P., "Model-driven development: The good, the bad, and the ugly," *IBM System*, vol. Vol 45, No 3, pp. 451-461, 2006.

[27] G. M. Weinberg, *Quality software management (Vol. 1): systems thinking*: Dorset House Publishing Co., Inc., 1992.

[28] "pattern," ed: Cambridge Dictionary, 2017.

[29] (2017, 04/02/2017). *Pattern*. Available: https://www.merriam-webster.com/dictionary/pattern

[30] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*: Addison-Wesley, 1998.

[31] B. P. Douglass, *Real-Time Design Patterns : robust scalable architecture for Real-time systems*. Boston, MA: Addison-Wesley, 2003.

[32] A. G. Kleppe, J. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*: Addison-Wesley Longman Publishing Co., Inc., 2003.

[33] M. Petre, "UML in practice," in *35th International Conference on Software Engineering (ICSE)*, 2013, pp. 722-731.

[34] C. Kobryn, "Will UML 2.0 be agile or awkward?," *Commun. ACM*, vol. 45, pp. 107-110, 2002.

[35] J. Greenfield, K. Short, S. Cook, and S. Kent, *Software Factories - Assembling Application with Patterns, Models, Frameworks and Tools*: Wiley Publishing, 2004.

[36] P. Järvinen, *On Research Methods*, 2001.

[37] D. Budgen, A. J. Burn, O. P. Brereton, B. A. Kitchenham, and R. Pretorius, "Empirical evidence about the UML: a systematic literature review," *Software: Practice and Experience*, vol. 41, pp. 363-392, 2011.

[38] P. Runeson, M. Host, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering: Guidelines and Examples*: Wiley Publishing, 2012.

[39] S. T. March and G. F. Smith, "Design and natural science research on information technology," *Decision Support Systems*, vol. 15, pp. 251-266, 1995/12/01/ 1995.

[40] I. Benbasat, D. K. Goldstein, and M. Mead, "The case research strategy in studies of information systems," *MIS Quarterly*, vol. 11(3):369, 1987.

[41] R. K. Yin, *Case Study Research: Design and Methods*: SAGE Publications, 2003.

[42] B. A. Kitchenham, T. Dybå, and M. Jørgensen, "Evidence-based software engineering," presented at the Proceedings of the 26th International Conference on Software Engineering,(ICSE'04), Washington DC, USA, 2004.

[43] B. Kitchenham, R. Pretorius, D. Budgen, O. Pearl Brereton, M. Turner, M. Niazi, *et al.*, "Systematic literature reviews in software engineering – A tertiary study," *Information and Software Technology*, vol. 52, pp. 792-805, 2010/08/01/ 2010.

[44] D. Akdur. (2017, Last accessed: July 01, 2017). *Tertiary Study for MDE*. Available: https://docs.google.com/spreadsheets/d/1x81mp7pqdXzRAVOvhFLs8bV32gMbZGo40ZDofmk6lak/

[45] M. Broy, S. Kirstan, H. Krcmar, and B. Schätz, "What is the benefit of a model-based design of embedded software systems in the car industry?," in *Emerging Technologies for the Evolution and Maintenance of Software Models*, ed, 2011, pp. 343-369.

[46] L. T. W. Agner, I. W. Soares, P. C. Stadzisz, and J. M. Simão, "A Brazilian survey on UML and model-driven practices for embedded software development," *Journal of Systems and Software*, vol. 86, pp. 997-1005, 2013.

[47] G. Liebel, N. Marko, M. Tichy, A. Leitner, and J. Hansson, "Assessing the State-of-Practice of Model-Based Engineering in the Embedded Systems Domain," in *Model-Driven Engineering Languages and Systems*. vol. 8767, ed: Springer International Publishing, 2014, pp. 166-182.

[48] G. Liebel, N. Marko, M. Tichy, A. Leitner, and J. Hansson, "Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice," *Software & Systems Modeling*, pp. 1-23, 2016.

[49] M. Grossman, J. E. Aronson, and R. V. McCarthy, "Does UML make the grade? Insights from the software development community," *Inf. Softw. Technol.*, vol. 47, pp. 383-397, 2005.

[50] B. Dobing and J. Parsons, "How UML is used," *Commun. ACM*, vol. 49, pp. 109-113, 2006.

[51] C. F. J. Lange, M. R. V. Chaudron, and J. Muskens, "In practice: UML software architecture and design description," *Software, IEEE*, vol. 23, pp. 40-46, 2006.

[52] J. Peneva, S. Ivanov, and G. Tuparov, "Utilization of UML in Bulgarian SME - Possible Training Strategies," *Communication and Cognition-Artificial Intelligence*, vol. 23, pp. 83 -88, 2006.

[53] A. Nugroho and M. R. V. Chaudron, "A survey into the rigor of UML use and its perceived impact on quality and productivity," presented at the Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, Kaiserslautern, Germany, 2008.

[54] P. Fitsilis, V. C. Gerogiannis, and L. Anthopoulos, "Role of unified modelling language in software development in Greece - results from an exploratory study," *Software, IET*, vol. 8, pp. 143-153, 2014.

[55] A. Forward and T. C. Lethbridge, "Problems and opportunities for model-centric versus code-centric software development: a survey of software professionals," in *International workshop on Models in software engineering*, Leipzig, Germany, 2008, pp. 27-32.

[56] J. Hutchinson, J. Whittle, M. Rouncefield, and S. Kristoffersen, "Empirical assessment of MDE in industry," in *33rd International Conference on Software Engineering*, Waikiki, Honolulu, HI, USA, 2011, pp. 471-480.

[57] M. Torchiano, F. Tomassetti, F. Ricca, A. Tiso, and G. Reggio, "Preliminary Findings from a Survey on the MD State of the Practice," in *Empirical Software Engineering and Measurement (ESEM)*, 2011, pp. 372-375.

[58] M. Torchiano, F. Tomassetti, F. Ricca, A. Tiso, and G. Reggio, "Relevance, benefits, and problems of software modelling and model driven techniques—A survey in the Italian industry," *Journal of Systems and Software*, vol. 86, pp. 2110-2126, 2013.

[59] F. Shull, J. Singer, and D. I. K. Sjoberg, *Guide to Advanced Empirical Software Engineering*: Springer-Verlag New York, Inc., 2007.

[60] R. M. Groves, F. J. Fowler, M. P. Couper, J. M. Lepkowski, E. Singer, and R. Tourangeau, *Survey Methodology*, Second ed.: John Wiley & Sons, 2009.

[61] J. Linaker, S. M. Sulaman, R. Maiani de Mello, M. Höst, and P. Runeson, "Guidelines for Conducting Surveys in Software Engineering," 2015.

[62] V. C. Basili, G.; Rombach, D.H., "The Goal Question Metric Approach," in *Encyclopedia of Software Engineering*, ed: Wiley, 1994.

[63] T. Punter, M. Ciolkowski, B. Freimut, and I. John, "Conducting on-line surveys in software engineering," in *Proceedings of International Symposium on Empirical Software Engineering*, 2003, pp. 80-88.

[64] T. R. Lunsford and B. R. Lunsford, "The Research Sample, Part I: Sampling," *J. Prosthetics and Orthotics*, vol. 7, pp. 105-112, 1995.

[65] V. Garousi, A. Coşkunçay, A. Betin-Can, and O. Demirörs, "A Survey of Software Engineering Practices in Turkey," *Journal of Systems and Software*, vol. 108, pp. 148-177, 2015.

[66] D. A. Garvin, *Managing quality: the strategic and competitive edge*: Free Press, 1988.

[67] D. Akdur, V. Garousi, and O. Demirörs, "MDE in embedded software industry, Technical Report," *METU II-TR-2015-55, https://dx.doi.org/10.6084/m9.figshare.4262990*, 2015, Last accessed: Nov. 27, 2016.

[68] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, and A. Tang, "What Industry Needs from Architectural Languages: A Survey," *IEEE Transactions on Software Engineering*, vol. 39, pp. 869-891, 2013.

[69] I. Sommerville, *Software Engineering*: Addison Wesley, 2010.

[70] D. Akdur, V. Garousi, and O. Demirörs, "Cross-factor analysis of software modeling practices versus practitioner demographics in the embedded software industry," in *6th Mediterranean Conference on Embedded Computing (MECO)*, Montenegro, 2017.

[71] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*: Springer Berlin Heidelberg, 2012.

[72] P. S. M. d. Santos and G. H. Travassos, "Action research use in software engineering: An initial survey," presented at the Proceedings of the International Symposium on Empirical Software Engineering and Measurement, 2009.

[73] S. Baltes and S. Diehl, "Sketches and diagrams in practice," presented at the Proceedings of ACM SIGSOFT International Symposium on Foundations of Software Engineering, China, 2014.

[74] N. A. Karagoz and O. Demirors, "Conceptual Modeling Notations and Techniques," in *Conceptual Modeling for Discrete-Event Simulation*, ed, 2010.

[75] D. Akdur, O. Demirörs, and V. Garousi, "Characterizing the development and usage of diagrams in embedded software systems," in *43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Vienna, Austria, 2017.

[76] A. Nugroho and M. R. Chaudron, "A survey into the rigor of UML use and its perceived impact on quality and productivity," in *ACM-IEEE Empirical Software Engineering and Measurement (ESEM)*, 2008, pp. 90-99.

[77] D. Akdur, V. Garousi, and O. Demirörs, "MDE in embedded SW industry-Raw survey data," *https://dx.doi.org/10.6084/m9.figshare.4262972*, 2015, Last accessed: Nov. 27, 2016.

[78] V. Bolón-Canedo, N. Sánchez-Maroño, A. Alonso-Betanzos, J. M. Benítez, and F. Herrera, "A review of microarray datasets and applied feature selection methods," *Information Sciences*, vol. 282, pp. 111-135, 2014/10/20/ 2014.

[79] (12/09/2016). *RapidMiner: Data Science Platform*. Available: https://rapidminer.com/

[80] M. Denscombe, *The Good Research Guide: For Small-scale Social Research Projects*: McGraw-Hill Education, 2014.

[81] E. Juliot, "Model Driven Software Development 2.0," in *International Advanced Topics in Software Engineering (ATSEN)*, İstanbul, Turkey, 2014.

[82] D. Akdur and O. Demirörs, "Multiple Case Studies to Validate Modeling Patterns and Cultures of Embedded Software Development Projects, Technical Report," METU2017.

[83] C. Robson, *Real world research*, 2nd ed., 2002.

[84] R. E. Stake, *The Art of Case Study Research*: SAGE Publications, 1995.

[85] B. A. Kitchenham, S. Linkman, and D. Law, "DESMET: A Methodology for Evaluating Software Engineering Methods and Tools," *Computing and Control Engineering Journal*, 1997.

[86] J. M. Verner, J. Sampson, V. Tosic, N. A. A. Bakar, and B. A. Kitchenham, "Guidelines for industrially-based multiple case studies in software engineering. ," presented at the Third International Conference on Research Challenges in Information Science, Morocco, 2009.

[87] G. Kahraman and S. Bilgen, "A framework for qualitative assessment of domain-specific languages," *Software & Systems Modeling*, vol. 14, pp. 1505-1526, October 01 2015.

[88] B. A. Kitchenham, L. M. Pickard, and S. Pfleeger, "Case studies for method and tool evaluation," *IEEE Software*, vol. 12(4), pp. 52-62, 1995.

[89] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. Hoaglin, K. El Emam, *et al.*, "Preliminary guidelines for empirical research in software engineering," presented at the IEEE Transactions on Software Engineering, 2002.

[90] A. Uskarcı and O. Demirörs, "Do staged maturity models result in organization-wide continuous process improvement? Insight from employees," *Computer Standards & Interfaces*, vol. 52, pp. 25-40, 2017/05/01/ 2017.

[91] A. Dikici, O. Turetken, and O. Demirors, "Factors influencing the understandability of process models: A systematic literature review," *Information and Software Technology*, vol. 93, pp. 112-129, 2018.

[92] D. Akdur, "Modeling Patterns and Cultures of Embedded Software Development Projects," *Thesis, Doctor of Philosophy (PhD), Information Systems, Middle East Technical University (METU), https://www.researchgate.net/publication/322701453_Modeling_Patterns_and_Cultures_of_Embedded_Software_Development_Projects*, February 1, 2018.

[93] A. Dorling, "SPICE: Software process improvement and capability dEtermination," *Information and Software Technology*, vol. 35, pp. 404-406, 1993/06/01/ 1993.

[94] (2018, 12/01/2018). *CMMI Institute*. Available: http://cmmiinstitute.com/

[95] F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *MIS quarterly*, pp. 319-340, 1989.

[96] Ö. Kılıç, B. Say, and O. Demirörs, "An Experimental Study on the Cognitive Characteristics of Modeling Notations," ed Advances in Dynamic and Static Media for Interactive Systems: Communicability, Computer Science and Design, 2011.

[97] F. b. D. Giraldo, S. Espana, and O. Pastor, "Analysing the concept of quality in model-driven engineering literature: A systematic review," in *Research Challenges in Information Science (RCIS), 2014 IEEE Eighth International Conference on*, 2014, pp. 1-12.

[98] A. Mehmood and D. N. Jawawi, "Aspect-oriented model-driven code generation: A systematic mapping study," *Information and Software Technology*, vol. 55, pp. 395-411, 2013.

[99] S. Stavru, I. Krasteva, and S. Ilieva, "Challenges of Model-driven Modernization-An Agile Perspective," in *MODELSWARD*, 2013, pp. 219-230.

[100] S. B. Tajali, V. D. Radonjic, and J.-P. Corriveau, "Challenges of variability in model-driven and transformational approaches: A systematic survey," in *9th Working IEEE/IFIP Conference on Software Architecture (WICSA)*, 2011, pp. 294-301.

[101] P. K. Thalanki, "Classifying Research on UML model inconsistencies with Systematic Mapping," Blekinge Institute of Technology, 2013.

[102] B. Hoisl and S. Sobernig, "Consistency Rules for UML-based Domain-specific Language Models: A Literature Review," *Joint proceedings of ACES-MB 2015–Model-based Architecting of Cyber-physical and Embedded Systems*, p. 29, 2015.

[103] J. Cabot and E. Teniente, "Constraint Support in MDA Tools: A Survey," in *Model Driven Architecture – Foundations and Applications: Second European Conference, ECMDA-FA 2006, Bilbao, Spain, July 10-13, 2006. Proceedings*, A. Rensink and J. Warmer, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 256-267.

[104] P. Mohagheghi, V. Dehlen, and T. Neple, "Definitions and approaches to model quality in model-based software development – A review of literature," *Information and Software Technology*, vol. 51, pp. 1646-1669, 2009.

[105] K. Vanherpen, J. Denil, P. De Meulenaere, and H. Vangheluwe, "Design-Space Exploration in Model Driven Engineering -An Initial Pattern Catalogue," in *1st International Workshop on Combining Modelling with Search- and Example-Based Approaches, CMSEBA 2014 - Co-located with 17th International Conference on Model Driven Engineering Languages and Systems, MODELS 2014*, 2014, pp. 42-51.

[106] P. G. Gadelha Queiroz and R. T. Vaccare Braga, "Development of Critical Embedded Systems Using Model-Driven and Product Lines Techniques: A Systematic Review,"

in *8th Brazilian Symposium on Software Components, Architectures and Reuse, SBCARS* 2014, pp. 74-83.

[107] D. Ameller, X. Burgués, O. Collell, D. Costal, X. Franch, and M. P. Papazoglou, "Development of service-oriented architectures using model-driven development: A mapping study," *Information and Software Technology,* vol. 62, pp. 42-66, 6// 2015.

[108] T. Kosar, S. Bohra, and M. Mernik, "Domain-Specific Languages: A Systematic Mapping Study," *Information and Software Technology,* vol. 71, pp. 77-91, 2016.

[109] A. M. Fernández-Sáez, M. Genero, and M. R. V. Chaudron, "Empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code: A systematic mapping study," *Information and Software Technology,* vol. 55, pp. 1119-1142, 2013.

[110] F. Siavashi and D. Truscan, "Environment modeling in model-based testing: concepts, prospects and research challenges: a systematic literature review," in *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*, 2015, p. 30.

[111] A. Saeed, S. H. Ab Hamid, and M. B. Mustafa, "The experimental applications of search-based techniques for model-based testing: Taxonomy and systematic literature review," *Applied Soft Computing,* vol. 49, pp. 1094-1117, 2016.

[112] P. H. Nguyen, M. Kramer, J. Klein, and Y. L. Traon, "An extensive systematic review on the Model-Driven Development of secure systems," *Information and Software Technology,* vol. 68, pp. 62-81, 12// 2015.

[113] S. Sobernig, B. Hoisl, and M. Strembeck, "Extracting reusable design decisions for UML-based domain-specific languages: A multi-method study," *Journal of Systems and Software,* vol. 113, pp. 140-172, 2016/03/01/ 2016.

[114] C. A. González and J. Cabot, "Formal verification of static software models in MDE: A systematic review," *Information and Software Technology,* vol. 56, pp. 821-838, 2014.

[115] L. Shuang, "Formalizing UML State Machines Semantics for Formal Analysis–A survey (PRELIMINARY VERSION)," 2014.

[116] S. Hansson, Y. Zhao, and H. Burden, "How MAD are we? Empirical evidence for model-driven agile development," in *3rd Extreme Modeling Workshop, XM 2014, Co-located with ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems, MoDELS 2014*, 2014, pp. 2-11.

[117] V. V. G. Neto, M. Guessi, L. B. R. Oliveira, F. Oquendo, and E. Y. Nakagawa, "Investigating the Model-Driven Development for Systems-of-Systems," presented at the Proceedings of the 2014 European Conference on Software Architecture Workshops, Vienna, Austria, 2014.

[118] R. Pretorius and D. Budgen, "A mapping study on empirical evidence related to the models and forms used in the uml," presented at the Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement, Kaiserslautern, Germany, 2008.

[119] J. M. Perez, F. Ruiz, and M. Piattini, "MDE for BPM: a systematic review," in *Software and Data Technologies*, ed: Springer, 2008, pp. 127-135.

[120] H. Javed, N. M. Minhas, A. Abbas, and F. M. Riaz, "Model Based Testing for Web Applications: A Literature Survey Presented," *JSW,* vol. 11, pp. 347-361, 2016.

[121] K. Wakil and D. N. Jawawi, "Model driven web engineering: A systematic mapping study," *e-Informatica Software Engineering Journal,* vol. 9, pp. 107-142, 2015.

[122] A. I. E. S. Eldein and H. H. Ammar, "Model-Driven Architecture for Cloud Applications Development, A survey," 2015.

[123] I. Santiago, Á. Jiménez, J. M. Vara, V. De Castro, V. A. Bollati, and E. Marcos, "Model-Driven Engineering as a new landscape for traceability management: A systematic literature review," *Information and Software Technology,* vol. 54, pp. 1340-1356, 2012.

[124] G. L. Casalaro and G. Cattivera, "Model-driven Engineering For Mobile Robot Systems: A Systematic Mapping Study," 2015.

[125] M. Genero, A. M. Fernández-Saez, H. J. Nelson, G. Poels, and M. Piattini, "Research review: a systematic literature review on the quality of UML models," *Journal of Database Management (JDM),* vol. 22, pp. 46-70, 2011.

[126] J. Jensen and M. G. Jaatun, "Security in model driven development: a survey," in *Availability, Reliability and Security (ARES), 2011 Sixth International Conference on*, 2011, pp. 704-709.

[127] A. Khalil and J. Dingel, "Supporting the evolution of UML models in model driven software development: a survey," Tech. rep., School of Computing, Queen's University Kingston, Ontario, Canada2013.

[128] H. Giese and S. Henkler, "A survey of approaches for the visual model-driven development of next generation software-intensive systems," *Journal of Visual Languages & Computing*, vol. 17, pp. 528-550, 12// 2006.

[129] M. Mussa, S. Ouchani, W. Al Sammane, and A. Hamou-Lhadj, "A survey of model-driven testing techniques," in *Quality Software, 2009. QSIC'09. 9th International Conference on*, 2009, pp. 167-172.

[130] F. Valles-Barajas, "A survey of UML applications in mechatronic systems," *Innovations in Systems and Software Engineering*, vol. 7, pp. 43-51, 2011.

[131] J. A. McQuillan and J. F. Power, "A survey of UML-based coverage criteria for software testing," *Department of Computer Science. NUI Maynooth, Co. Kildare, Ireland*, 2005.

[132] A. C. Dias Neto, R. Subramanyan, M. Vieira, and G. H. Travassos, "A survey on model-based testing approaches: a systematic review," in *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies: held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, 2007, pp. 31-36.

[133] M. Szvetits and U. Zdun, "Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime," *Software & Systems Modeling*, vol. 15, pp. 31-69, February 01 2016.

[134] S. Tiwari and A. Gupta, "A systematic literature review of use case specifications research," *Information and Software Technology*, vol. 67, pp. 128-158, 2015.

[135] M. Nelson and M. Piattini, "A systematic literature review on the quality of uml models," *Innovations in Database Design, Web Applications, and Information Systems Management*, p. 310, 2012.

[136] S. R. A. Meireles and A. C. Dias-Neto, "A Systematic Mapping on Model Based Testing applied to Web Systems," *SAST 2014*, p. 51, 2013.

[137] E. Batot, H. Sahraoui, E. Syriani, P. Molins, and W. Sboui, "Systematic mapping study of model transformations for concrete problems," in *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2016, pp. 176-183.

[138] A. Kaur and V. Vig, "Systematic review of automatic test case generation by UML diagrams," in *International Journal of Engineering Research and Technology*, 2012.

[139] E. Domínguez, B. Pérez, Á. L. Rubio, and M. a. A. Zapata, "A systematic review of code generation proposals from state machine specifications," *Information and Software Technology*, vol. 54, pp. 1045-1066, 10// 2012.

[140] T. H. Haug, "A systematic review of empirical research on model-driven development with UML," 2007.

[141] S. Hooda, S. Dalai, and K. Solanki, "A systematic review of model-based testing in aspect-oriented software systems," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, 2016, pp. 2944-2949.

[142] M. Shafique and Y. Labiche, "A systematic review of model based testing tool support," *Carleton University, Canada, Tech. Rep. Technical Report SCE-10-04*, 2010.

[143] P. H. Nguyen, J. Klein, Y. Le Traon, and M. E. Kramer, "A Systematic Review of Model-Driven Security," in *Software Engineering Conference (APSEC, 2013 20th Asia-Pacific*, 2013, pp. 432-441.

[144] G. Loniewski, E. Insfran, and S. Abrahão, "A systematic review of the use of requirements engineering techniques in model-driven development," in *Model driven engineering languages and systems*, ed: Springer, 2010, pp. 213-227.

[145] S. S. Priya and P. Sheba, "Test Case Generation from UML models-A survey," in *Proc. International Conference on Information Systems and Computing (ICISC-2013), INDIA*, 2013.

[146] M. Aggarwal and S. Sabharwal, "Test case generation from UML state machine diagram: A survey," in *Computer and Communication Technology (ICCCT), 2012 Third International Conference on*, 2012, pp. 133-140.

[147] M. Rashid, M. W. Anwar, and A. M. Khan, "Toward the tools selection in model based system engineering for embedded systems—A systematic literature review," *Journal of Systems and Software*, vol. 106, pp. 150-163, 2015.

[148] D. Torre, Y. Labiche, and M. Genero, "UML consistency rules: a systematic mapping study," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014, p. 6.

[149] M. Misbhauddin and M. Alshayeb, "UML model refactoring: a systematic literature review," *Empirical Software Engineering*, vol. 20, pp. 206-251, 2013.

[150] V. Garousi and T. Varma, "A Replicated Survey of Software Testing Practices in the Canadian Province of Alberta: What has Changed from 2004 to 2009?," *Journal of Systems and Software,* vol. 83, pp. 2251-2262, 2010.

[151] L. Wallace, M. Keil, and A. Rai, "Understanding software project risk: a cluster analysis," *Inf. Manage.*, vol. 42, pp. 115-125, 2004.

[152] D. Akdur, V. Garousi, and O. Demirörs, "MDE in embedded SW industry-Survey Form (Questions)," *https://dx.doi.org/10.6084/m9.figshare.4262978*, 2015, Last accessed: Nov. 27, 2016.

[153] Project FP6-IP 511731 MODELWARE (MODELling solution for softWARE systems), "MDD Maturity Models," *http://www.cin.ufpe.br/~bbm/files/D2.6%20MDD%20Maturity%20Model.pdf* 2014, Last accessed: Sept. 2016.

[154] A. G. Kleppe, J. B. Warmer, and W. Bast, *MDA Explained: The Model Driven Architecture : Practice and Promise*: Addison-Wesley Professional, 2003.

[155] J. Kramer, "Is abstraction the key to computing?," *Commun. ACM,* vol. 50, pp. 36-42, 2007.

[156] D. Dori, "Why significant UML change is unlikely," *Commun. ACM,* vol. 45, pp. 82-85, 2002.

[157] S. Akayama, S. Kuboaki, K. Hisazumi, T. Futagami, and T. Kitasuka, "Development of a modeling education program for novices using model-driven development," presented at the Proceedings of the Workshop on Embedded and Cyber-Physical Systems Education, Tampere, Finland, 2013.

[158] M. Brandsteidl, K. Wieland, and C. Huemer, "Novel Communication Channels in Software Modeling Education," in *Models in Software Engineering: Workshops and Symposia at MODELS 2010, Oslo, Norway, October 2-8, 2010, Reports and Revised Selected Papers*, J. Dingel and A. Solberg, Eds., ed Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 40-54.

[159] S. Flint, H. Gardner, and C. Boughton, "Executable/Translatable UML in computing education," presented at the Proceedings of the Sixth Australasian Conference on Computing Education - Volume 30, Dunedin, New Zealand, 2004.

APPENDICES

APPENDIX A – SYSTEMATIC LITERATURE REVIEW – TERTIARY STUDY

In order to investigate RQ1.1.1, the following RQs were raised for this sub-study:

SLRRQ1: How many secondary studies (i.e., survey, SM, SLR) were published on this topic?
SLRRQ2: Which purpose(s) of software modeling and MDE are being addressed, if any?
SLRRQ3: Which benefit(s) of software modeling and MDE are being addressed, if any?
SLRRQ4: Which challenge(s) of software modeling and MDE are being addressed, if any?

Note that the results of SLRRQ2, SLRRQ3 and SLRRQ4 are presented in Section 2.2.

During the search process, four digital libraries (Science Direct, IEEE Xplore, Google Scholar and Scopus indexing systems) were used. Besides automated searches in these digital libraries, manual search on referenced articles and personal web pages were performed. The original RQ1 was "*until February 2015*", when the need for RQ1.1.2, RQ1.1.3 and RQ1.1.4 were arisen. After using its resulting set during survey design (i.e., after using them in our survey questions), we extended the publication period to "2017" for the tertiary study *(Note that this tertiary study is an input for the survey questions, hence for the conceptual model, which will be given in Section 4.1)*. Notice that since the goal was to get RQ2, RQ3 and RQ4 sets for the rest of the study (e.g., including survey and the characteristics of a diagram), the corresponding results are given based on the finalized search strategy including search strings, inclusion/exclusion criteria, which is given in Table 23.

Table 23 Tertiary Study Search Strategy

Databases searched	Search Engines (Science Direct, IEEE Xplore, Google Scholar, Elsevier Scopus)
	Besides automated searches in 4 digital libraries, manual search on referenced articles and personal web pages are performed.
Search Strings	(model driven OR model-driven OR MDE OR UML OR DSL OR DSML) AND (systematic mapping OR SM OR systematic review OR literature review OR SLR OR survey)
Topic Restriction	Software + Computer Science
Search applied to	Metadata only (Abstract/Summary & Title Text and Indexing Terms/Keywords) – if not possible, full text was searched
Language	Papers written in English
Publication period	until 2017 (exclusive)

Accordingly, by using search strings, there were potentially 2436 relevant papers. Then, by applying exclusion/inclusion criteria, removing duplicates and manually removing "*personal opinion survey*" papers, there were 54 papers in the final pool for attribute identification. All these processes is depicted in Figure 29.

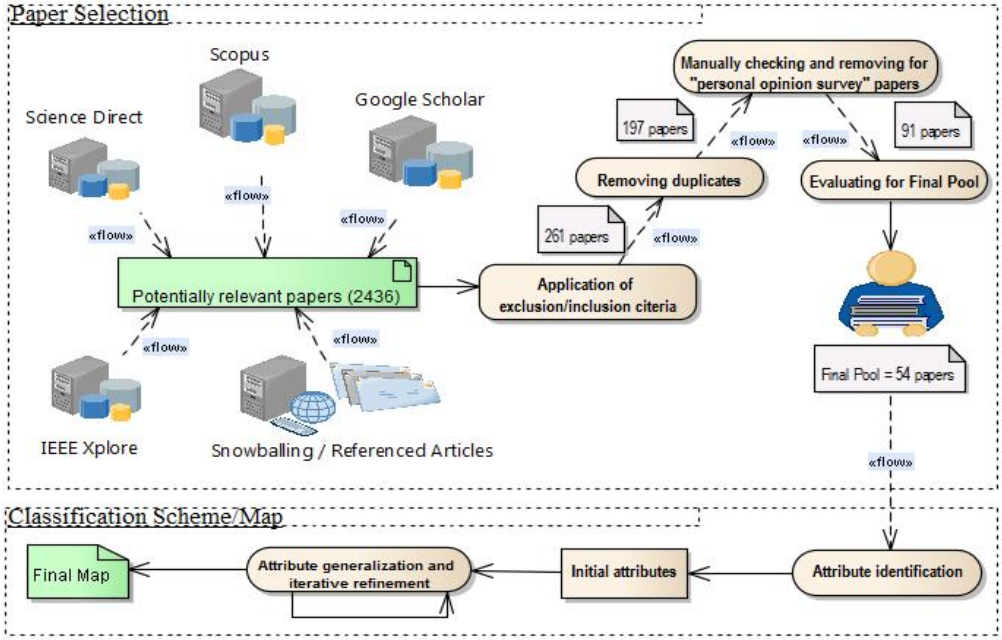


Figure 29: Tertiary study search process and final map

Note that the data extraction procedure for quality and classification was undertaken at the same time. Data extraction result table for final map is given in Table 24. *(For further details with respect to keywording, purposes, benefits, challenges in these secondary studies with their RQs and number of primary studies, see [44]).*

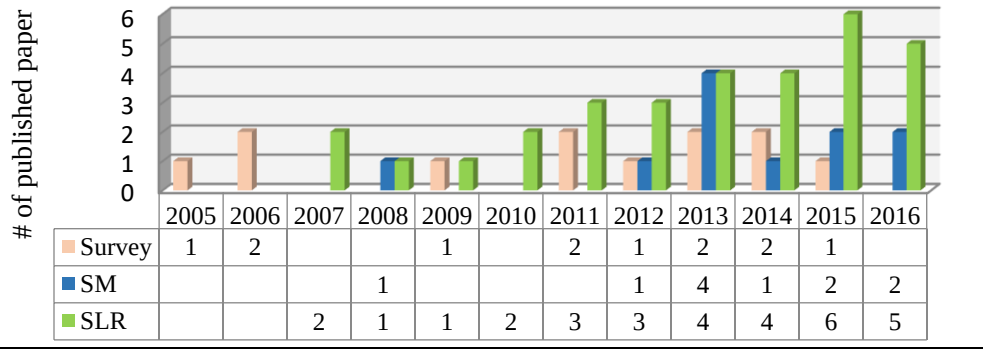
Table 24 Tertiary Study Final Map

Paper Title	**Reference**	**Type of secondary study**
"Analysing the concept of quality in model-driven engineering literature: A systematic review"	[97]	SLR
"Aspect-oriented model-driven code generation: A systematic mapping study"	[98]	SM
"Challenges of Model-driven Modernization-An Agile Perspective"	[99]	SLR
"Challenges of variability in model-driven and transformational approaches: A systematic survey"	[100]	Survey
"Classifying Research on UML model inconsistencies with Systematic Mapping"	[101]	SM
"Consistency Rules for UML-based Domain-specific Language Models: A Literature Review"	[102]	SLR
"Constraint Support in MDA Tools: A Survey"	[103]	Survey
"Definitions and approaches to model quality in model-based software development – A review of literature"	[104]	SLR
"Design-Space Exploration in Model Driven Engineering –An Initial Pattern Catalogue"	[105]	Survey
"Development of Critical Embedded Systems Using Model-Driven and Product Lines Techniques- A Systematic Review"	[106]	SLR
"Development of service-oriented architectures using model-driven development: A mapping study"	[107]	SLR
"Domain-Specific Languages: A Systematic Mapping Study"	[108]	SM

Table 24 (continued)		
"Empirical evidence about the UML: a systematic literature review"	[37]	SLR
"Empirical studies concerning the maintenance of UML diagrams and their use in the maintenance of code: A systematic mapping study"	[109]	SM
"Environment modeling in model-based testing: concepts, prospects and research challenges: a systematic literature review"	[110]	SLR
"The experimental applications of search-based techniques for model-based testing: Taxonomy and systematic literature review"	[111]	SLR
"An extensive systematic review on the Model-Driven Development of secure systems"	[112]	SLR
"Extracting reusable design decisions for UML-based domain-specific languages: A multi-method study"	[113]	SLR
"Formal verification of static software models in MDE: A systematic review"	[114]	SLR
"Formalizing UML State Machines Semantics for Formal Analysis–A survey"	[115]	Survey
"How MAD are we? Empirical evidence for model-driven agile development"	[116]	SLR
"Investigating the Model-Driven Development for Systems-of-Systems"	[117]	SLR
"A Mapping Study on Empirical Evidence related to the Models and Forms used in the UML"	[118]	SM
"MDE for BPM: a systematic review"	[119]	SLR
"Model Based Testing for Web Applications: A Literature Survey Presented"	[120]	SLR
"Model driven web engineering: A systematic mapping study"	[121]	SM
"Model-Driven Architecture for Cloud Applications Development, A survey"	[122]	Survey
"Model-Driven Engineering as a new landscape for traceability management: A systematic literature review"	[123]	SM
"Model-Driven Engineering for Mobile Robot Systems: A Systematic Mapping Study"	[124]	SM
"Research review: a systematic literature review on the quality of UML models"	[125]	SLR
"Security in model driven development: a survey"	[126]	SLR
"Supporting the evolution of UML models in model driven software development: a survey"	[127]	Survey
"A survey of approaches for the visual model-driven development of next generation software-intensive systems"	[128]	Survey
"A survey of model-driven testing techniques"	[129]	Survey
"A survey of UML applications in mechatronic systems"	[130]	Survey
"A survey of UML-based coverage criteria for software testing"	[131]	Survey
"A survey on model-based testing approaches: a systematic review"	[132]	SLR
"Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime"	[133]	SLR
"A systematic literature review of use case specifications research"	[134]	SLR
"A systematic literature review on the quality of uml models"	[135]	SLR
"A Systematic Mapping on Model Based Testing applied to Web Systems"	[136]	SM
"Systematic mapping study of model transformations for concrete problems"	[137]	SM

Table 24 (continued)		
"Systematic review of automatic test case generation by UML diagrams"	[138]	SLR
"A systematic review of code generation proposals from state machine specifications"	[139]	SLR
"A systematic review of empirical research on model-driven development with UML"	[140]	SLR
"A Systematic Review of Model-Based Testing in Aspect-Oriented Software Systems"	[141]	SLR
"A systematic review of model based testing tool support"	[142]	SLR
"A Systematic Review of Model-Driven Security"	[143]	SLR
"A systematic review of the use of requirements engineering techniques in model-driven development"	[144]	SLR
"Test Case Generation from UML models-A survey"	[145]	Survey
"Test case generation from UML state machine diagram: A survey"	[146]	Survey
"Toward the tools selection in model based system engineering for embedded systems—A systematic literature review"	[147]	SLR
"UML consistency rules: a systematic mapping study"	[148]	SM
"UML model refactoring: a systematic literature review"	[149]	SLR

Answer to SLRRQ1: While designing survey questions (i.e., "until February 2015"), there were 39 secondary studies, which were inputs for survey. When we extended the period, there are 54 secondary studies, which were published until 2017. 12 of them are survey, 11 of them are SM and 31 of them are SLR. The result for RQ1 based on published year, is presented in Figure 30. It is seen that the empirical evidence papers on this area, i.e., trend on software modeling and MDE, is increasing to fill the gap on this topic *(Note that between 2 years, i.e., "until February 2015" and "until 2017", 15 secondary studies were published, which show this trend).*



	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016
Survey	1	2			1		2	1	2	2	1	
SM				1				1	4	1	2	2
SLR			2	1	1	2	3	3	4	4	6	5

Figure 30: The trend on software modeling for systematic review studies

Answer to SLRRQ2: There are 48 secondary studies, which explicitly mention about the purposes of software modeling and MDE (i.e., 89% of final map). Since there were different terminologies to indicate the same purpose, to get a common language and get a catalog, similar purposes were combined in a single item. *(See [44] for all data in specific paper).* The final modeling purpose set is given in Table 3.

Answer to SLRRQ3: There are 46 secondary studies, which explicitly mention about the benefits of software modeling and MDE (i.e., 85% of final map). Similar benefits were

combined in a single item as in the case of purposes. *(See [44] for all data in specific paper)*. The final modeling benefit sets is given in Table 4.

Answer to SLRRQ4: There are 36 secondary studies, which explicitly mention about the challenges of software modeling and MDE (i.e., 67% of final map). Similar challenges were combined in a single item. *(See [44] for all data in specific paper).* The final modeling challenge set is given in Table 5.

Apart from these four RQs, which are directly related with this study, in [44], RQ types (e.g., existence, description and classification, descriptive and comparative, frequency, descriptive process, etc.), number of primary studies for each secondary studies, with their before and after exclusion ratio are presented in details.

APPENDIX B – SURVEY DETAILS

Appendix B.1 – Survey design and execution

Sampling method

In this study, even though we wanted to use probabilistic sampling, it was not practically doable to recruit a large pool of embedded software practitioners due to our limited resource constraints. As in the survey guidelines (e.g., [61, 63, 64]), we thus used the 'accidental non-probabilistic' sampling [61] and we targeted participants via our industry contacts, professional social network sites such as LinkedIn, industry events, and forums. Moreover, the survey was also promoted through SE and academic institutional mailing lists. Besides, we also encouraged them to distribute the survey to their colleagues. After receiving this non-probabilistic sampled data, one could possibly perform a-posteriori probability-based sampling. However, this was also infeasible since survey data were fully anonymous.

The 'unit' of interest analysis is another issue in the survey design [60]. The units of analysis in this survey might be anyone working in the embedded software domain, who individually and anonymously participated in the survey. Therefore, for all the statistics and analysis that were reported, these professionals are the unit of analysis and the implications shall be tied to world-wide community under investigation and neither to companies nor projects. Note that taking individual embedded professional as the unit of analysis has been considered a generally acceptable approach in previous surveys reported in the literature (e.g., [150, 151]).

Designing survey questions

Surveys require special considerations [61]. In order to have a survey that would completely cover the latest trends on software modeling, we benefitted from our tertiary study results, reviewed the similar surveys, considered factors given in survey guidelines [61], and prepared a draft set of questions. We conducted a round of peer reviews with nine industrial practitioners from different industry, different software engineering roles, different experiences and five different companies, in which our personal contacts have been working. All peer reviews were conducted face to face and according to their results, we improved four questions (i.e., Q20, Q25, Q26 and Q27). The final survey questionnaire consisted of four sections, each corresponding to each of the study RQs, as shown Table 25. The entire survey are not presented in this paper, but it can be found in an online source [152].

The introduction of the survey is written to attract respondents' attention. Therefore, the survey began with an informed consent, which contained the topic of the study, a confidentiality statement, the expected time to complete the survey and a thank you statement so that the potential respondents will decide whether or not to drop out of the questionnaire based solely on the first page. By clicking through the consent statement and submitting the completed survey, individuals are indicating their willingness to participate.

It is very important to have clear definitions and easy-to-follow instructions in the survey to get high quality data [61]. The first part of the questionnaire gathered personal and organizational demographic data. The 10th question investigated how often any informal or formal software modeling (i.e., sketches and/or models) is used in SDLC by asking "*How often do you use software modeling in your software development life cycle? (informal or formal: i.e., sketches or models*)". Since any informal usage of modeling was seen as "modeling usage" in this survey, the aim of this question was to understand the ratio of participants, who did not use any software modeling. After categorizing this group and made them complete the survey,

the questionnaire continued with modeling approaches questions, which aimed at understanding informal usage of modeling, model-based and model-driven techniques. In other words, this remaining part aimed at gathering RQ1.1. The terminology, which clearly explained the difference between model-based and model-driven techniques was given so that participants could consistently answer subsequent questions:

> *"Please read the following definitions before proceeding with the rest of the survey.*
>
> *In terms of terminology, Model Driven Development (MDD) uses models as the primary artifact of the development process. Usually, in MDD, the implementation is automatically generated from the models.*
>
> *Model Driven Engineering (MDE) is a superset of MDD since it encompasses other tasks of a complete software engineering process like testing and maintenance (i.e., documentation).*
>
> *On the other hand, Model Based Engineering (MBE) is a process, in which software models still play an important role although they are not necessarily the key artifacts of the development. For example, designers specify the models (i.e., by using paper or modeling tool), but then these models are directly handed out to the programmers to manually write the code (no auto generation)."*

With the help of this terminology and given example, we assume that respondents, at least, can understand the concept of "the automatic generation of an artifact", i.e., code, or document. Then, the survey asked about the degree of model-driven techniques in SDLC. In order to prevent any misunderstanding and potential threat in this terminology, pilot study was applied. After the pilot study, instead of asking *"Do you use any model-driven techniques?",* we modified this question into "*When you write code, document or test, to what degree do you use model driven techniques?"* by assuming that the respondent can answer whether there is an automatic generation of some artifact or not.

For each question, the type of answers are also mentioned in Table 25, e.g., single answer from a list, multiple answers, or a Likert scale *(Details of the responses can be found in [152])*

Table 25 List of the questions developed and used in the survey

RQ	Survey Questions (and Metrics)	Type of Answers				
		Single answer from a list	**Multiple answers could be chosen**	**Likert scale**	**Free text field**	**Likert scale (Range value from Never to Always)**
Profiles and demographics of practitioners and companies	**Q1.** Please choose the country that you work in.	x			x	
	Q2. What is your highest academic degree?	x				
	Q3. What is (are) your university degree(s) in?		x		x	
	Q4. What is (are) your current position(s)?		x		x	
	Q5. How many years of work experience do you have in software development?	x				
	Q11. How many years of modeling experience do you have in software development?	x				
	Q12. Where/how did you learn modeling?		x		x	
	Q6. What is the type of the application(s) developed in your company?		x		x	

	Table 25 (continued)					
	Q7. What is the target sector of the product(s) developed?		x		x	
	Q8. What is the number of employees working in software engineering roles?	x				
	Q9. What is the size of your typical software development team?	x				
RQ1.1.1	**Q10.** How often do you use software modeling in your software development life cycle? (informal sketches or formal models)					x
	Q13. What medium do you use to create the sketch or model?		x			x
	Q14. Which modeling language(s) do you use for modeling?		x		x	
	Q15. Which programming languages do you use with the above modeling language(s)?		x		x	
	Q16. Which modeling environment/tool(s) do you use, if any?		x		x	
	Q17. When modeling, which diagrams do you use?		x			x
	Q18. In which phase(s) of software development life cycle do you use modeling?		x			
RQ1.1.2	**Q19.** When you write code, document or test, to what degree do you use model driven techniques?					x
	Q20. What do you use software modeling and MDE for?		x		x	
	Q21. What is the estimated effort (in person-month) of the most representative MDE project in your company?	x				
	Q22. How would you describe your company's maturity in terms of its MDE usage?	x				
	Q23. What have been the motivations (potential benefits) that your company has considered for adopting MDE?		x	x		
RQ1.1.3	**Q24.** Based on your experience, to what degree has each of the above motivations (potential benefits) been achieved?		x	x		
	Q25. What is (are) MDE challenge(s) in your company?		x		x	
	Q26. To what extent do the following problems apply to the MDE environment/tool(s) that you have used?		x	x		
	Q27. Based on your experience, what do you think about the following statements?		x	x		

Survey piloting and execution

Performing a pilot study before distribution is an important step since it would help preventing misinterpretations in large-scale data collection of the survey. Pilot studies are carried out by using the same material and procedures but with a small number of participants from the target population [61]. Before the pilot study, it was necessary to decide whom to use as participants. It is recommended to select participants based on differences instead of similarities [38]. Therefore, the survey was firstly piloted by eight colleagues from different industries working in different software engineering roles, with different experiences and from different nations (four Turkish, two English, one French and one Taiwanese embedded software professionals). This was done to ensure that the wording and terminology used is easily understandable and well-formulated to get high quality data. In order to prevent misunderstandings, which could lead to invalidity of conclusions, great importance was given to clarifying survey questions

and explanations. Given their feedback, the questionnaire was updated by modifying three questions (i.e., Q10, Q19 and Q23), the terminology given at the beginning of 19th question, and five pre-given answers set (i.e., Q14, Q23, Q25, Q26 and Q27). The revised survey was reviewed a second time by five other colleagues with two colleagues, who were participated in the first pilot study. Therefore, the final version of this survey was reviewed by 13 industry professionals. After the revisions, the final version of the survey consisted of 27 questions, in the form of multiple-choice (checkboxes), single-choice (radio buttons) and Likert-scale answers. Where applicable, free-text areas for additional input were provided to respondents as "Other".

To design and execute the survey, we used the Google Forms tool. The ethics approval for the survey was issued by the Human Subjects Ethics Committee of Middle East Technical University (METU) in March 2015. The survey was then executed in the period of April-May 2015. The hyperlink of the survey has been distributed to embedded software professionals via social networks as well as to our network of embedded software professionals working in all around the world.

Pre-analysis Considerations and Data Validation

The last step of the survey process was to analyze the collected data. Although the title of the survey, the protocol part of the survey, the invitations and forums entries are emphasizing on "embedded", some participants chose just "Desktop applications" or "Web applications" for Q6 (What are the type of the applications developed in your company?). The answers, which do not include any "Embedded applications", were considered out of scope of this survey. Some companies develop different kinds of applications (e.g., both embedded and desktop); therefore any answer, which consisted of "Embedded", was included in the sample. Apart from that, there were no other criteria for inclusion or exclusion. By applying this criterion, 15 surveys were excluded. After the data validation phase, we had 627 acceptable responses from 27 different countries. To increase transparency, the raw survey data is made available online [77] for other researchers to validate and replicate. Notice that to ease the analysis in [77], we used abbreviations (i.e., Q18 asks *"In which phase(s) of software development life cycle do you use modeling?"*, but in [77], we shortened the question and used *"SDLC"*); therefore, the wording used in Table 25 is not the same as in [77]. Considering that no incentive was offered to the participants, it is interesting to see that the number of participants is quite high in comparison to previous related surveys.

Appendix B.2 – Results

In this part, a subset of the survey is reported. All other remaining answers in the survey are accessible from the technical report [67].

Demographics

The first survey question asked respondent about their geographical location (Q1). The goal was to reach out to as many countries as possible where there is a presence of embedded software industry. The final dataset had respondents from 27 different countries distributed in all the continents. Figure 31 shows the world heat-map, and also the distribution of responses by continents, showing that most of the responses originating from Europe (66%), followed by Asia (17%) and America (14%). Of course these data do not provide any information in relation with relative sizes of the embedded software industry in different continents. Due to researchers' location (i.e., Turkey), the ratio of European respondents is higher than others.

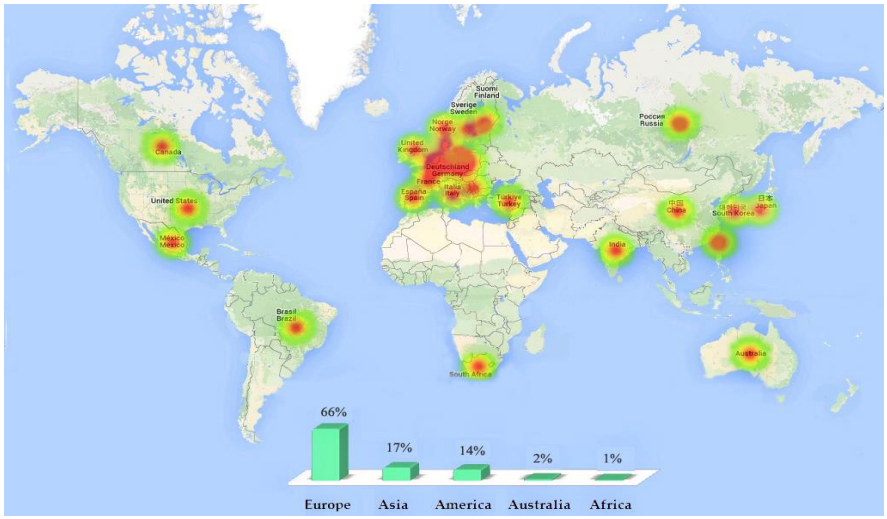


Figure 31: Survey – Countries and geographical distribution of respondents

Respondents were asked about their highest academic degrees (Q2). The result shows that 50% and 11% of respondents have a Master's and PhD's degree respectively. 39% of respondents have Bachelor's degrees. Only three respondents (0.5%) reporting to have High school or lower degree, denoting that the embedded software domain is demanding in terms of background knowledge. Figure 32 shows that our dataset includes more PhD and MSc holders than our expectation, perhaps denoting that the modeling in embedded software is demanding more combination of academic disciplines (i.e., an embedded software engineer whose BSc is in Electrical/Electronics Engineering and MSc is in SE).

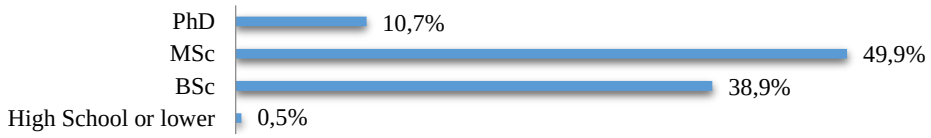


Figure 32: Survey - Highest academic degrees

Q6, in which the type of the applications developed was asked, is the only question, which is used for inclusion or exclusion of data points gathered from the respondents. In this multiple-response question, multiple type of application could be chosen, e.g., a company can develop both embedded and desktop applications. 77% of participants reported developing "Embedded applications and 13% of participants both "Embedded" and "Desktop" applications. Some participants used the free-text area as "Other" (10% of participants) to explicitly indicate their type of applications developed in their company. Some responses (e.g., "Smart TV applications") are also counted to be in the embedded domain and included in our dataset.

To get a sense of the size of the companies, instead of asking the size of the company (in order to eliminate non-engineering roles as technicians, office workers, etc.), the number of employees in SE roles was asked as Q8. Results are shows in Figure 33. A good mixture of participants from different ranges was also present in our survey pool. By this way, a wider spectrum of inputs in terms of number of employees in SE roles in our analysis were covered.

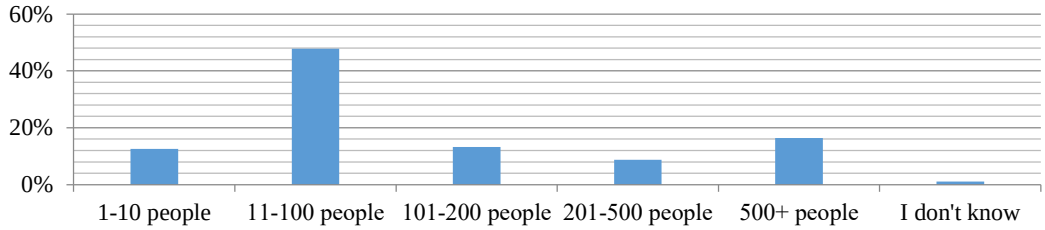


Figure 33: Survey - Number of employees in SE roles

Software Modeling and MDE-related Questions

Q16 was a multiple-response question, in which modeling environments/tools were asked. As seen in Figure 34, the majority of respondents use "Eclipse-based" tools, which is followed by "Microsoft Visio". About 7.2% of the respondents indicated that they do not use any modeling environment or tool, which almost all came from users which reported not using PC-based tools. Again, among the "Other" answers for this question, respondents mentioned modeling tools such as: Papyrus, MaTeLo, argoUML, MetaEdit+, Astah, and Artop (For the details of "Other" answers, please refer to [67]).

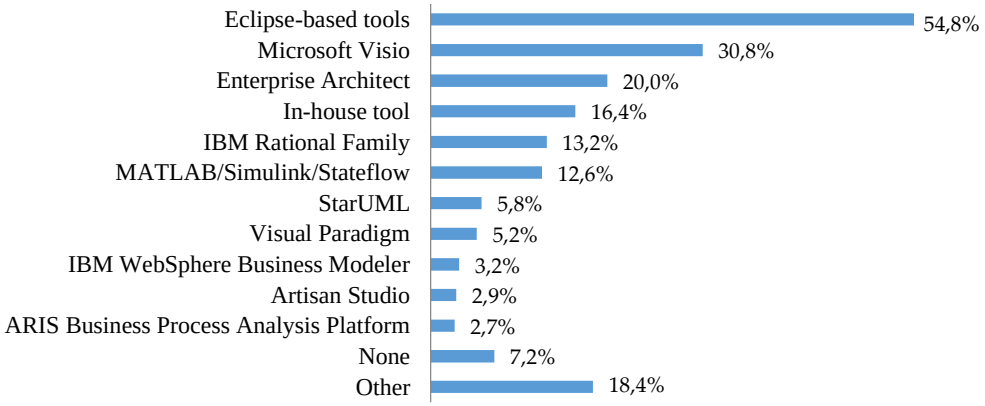


Figure 34: Survey - Modeling tools

[46] stated that survey studies are needed to investigate the types of UML tools used in practice. As a comparison, in the dataset of the survey reported in [47], the majority (%50) used Matlab/Simulink/Stateflow, followed by Eclipse-based tools, Enterprise Architect, in-house tools and IBM Rational Software Modeler.

Q19 investigates how often the participants use MDE. The participants, who mentioned not using MDE at all, i.e., the "Never" option (59.5% of all participants) are either model-based or sketch users; and the remaning (29.5% of all participants) are model-driven users. The results are shown in Figure 35.

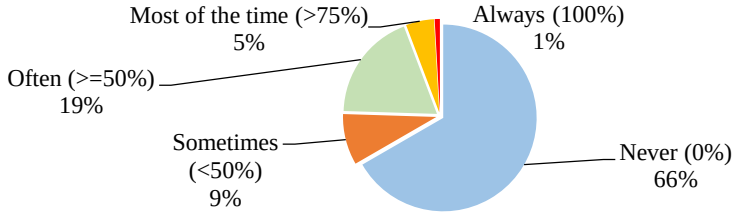


Figure 35: Survey - Degree of using MDE

The results shows that the MDE usage ratio is slightly more than the ratio reported in [46], in which 15.8% of its participants reported knowing MDE and using it. This study reflects a

world-wide picture without limiting itself to specific region (e.g., Brazil) and includes more participants from 27 different countries in five continents. Moreover, time has passed after [46] was executed and most probably, the embedded software industry has gradually further learned the MDE practices more and its usage ratio has increased. Therefore, this difference might be explained with the participants' demographics and the possible increasing popularity in MDE practices in the embedded software industry.

Participants were then asked to describe their company's maturity in its use of MDE (Q22). We were aware of several existing maturity models for MDE, e.g., [153] and [154]. [153] seems to be the most comprehensive maturity models in this context. In choosing a maturity model to be used in the survey, there were two criteria: (1) using the maturity model should not lead to having many questions which would negatively impact the response rate of our survey, and (2) the maturity model should be comparable to existing measurements in the reported surveys. Due to this, the maturity model was adopted as shown in Figure 36. The majority of the participants (57%) are in the Level 4, indicating that they have completed multiple MDE projects. 10% of participants reported that they have the first significant project on MDE (just finished); whereas 6% are in initial exploration phase and 10% are in the prototyping phase of MDE. On the other hand, 9% of participants reported an extensive experience of MDE on many projects and/or over many years.

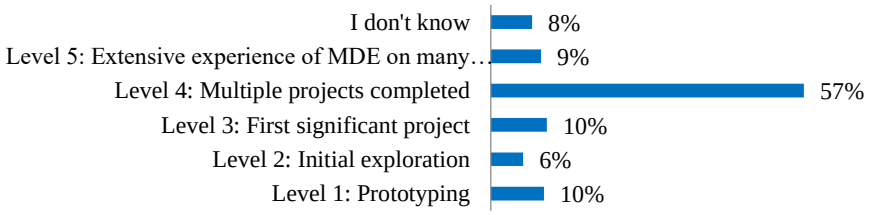


Figure 36: Survey - Maturity of MDE usage

According to [46], since it only focused on UML, 48% of the respondents reported its use as an initial exploration of MDE with UML and only 21% confirmed the development of several complete projects using UML, whereas the others mentioned its use as a first experimental use (13%) and first significant project (17%). On the other hand, concerning the MBE experience in [47], many participants (41%) are well experienced with more than 3 years of usage; whereas 36% state that they have moderate experience and only 23% are new in the field of MBE.

Since the terminologies used in these two studies are different from each other, we want to categorize them in similar groups. According to that categorization, we assume that "initial exploration" in [46] is in the same category in "new" in [47]; "first experimental use and first significant project" in [46] is in the same category in "moderate experience" in [47]; and finally "several complete projects" in [46] is in the same category in "well experienced" in [47] (which is our both "multiple projects completed" and "extensive experience" categories). The maturity level comparison depending on this categorization is depicted in Figure 37. As it can be seen, we can say that maturity level has changed (and increased) depending on either time or generalization of geographical area (i.e., [46] was executed at 2011 in Brazil and [47] was very recent in Europe). Notice that, by no means, these data indicate that the popularity and the usage of MDE have increased, but it gives an insight about its trends.

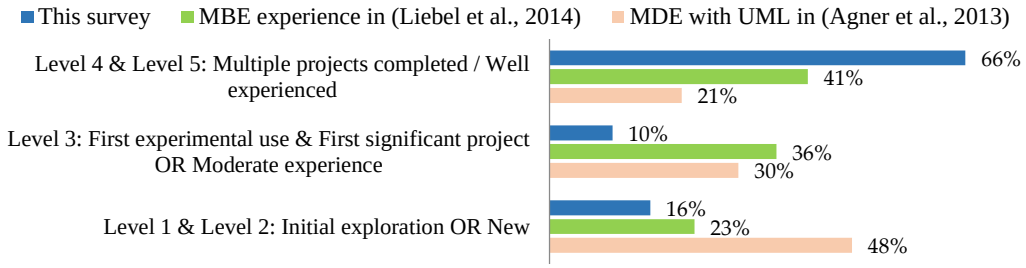


Figure 37: Survey - MDE/MBE maturity level comparing with related works

In Q26, as a both multiple-response and 5-point Likert-scale question, participants were asked about the degree to which the given problems are applied to their MDE environment/tool. All responses are given in Figure 38, whose x-axis indicates the response percentage *(In the figure, red and orange bars indicate the existence of such a problem; whereas green-based bars indicate that there is no such an existence. On the other hand, neutral responses are depicted with yellow bar, and "not applicable" answers are depicted with grey bar)*.

Notice that MDE environments/tools problems are directly related with what MDE is used for (Q20) hence "not applicable" answers (e.g., for the respondents who use MDE for only "documentation generation", "difficulties with code generation capabilities" is not applicable).

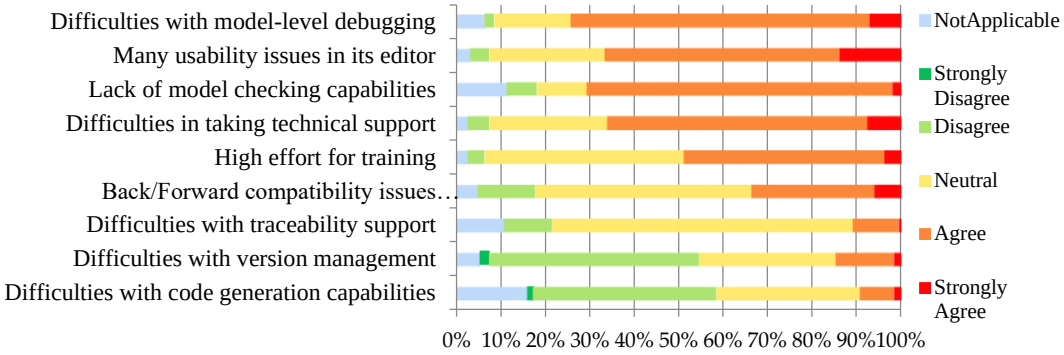


Figure 38: Survey - Problems with MDE environments/tools

According to [47], tool-related problems were reported to be the followings: many usability issues with the tools, difficulties with version management, difficulties of integration with legacy code, impossible/difficult to customize the tools, lack of model checking capabilities and difficulties with code generation capabilities. Such findings are quite similar to our results.

Q27 investigated the impacts of MDE on model-driven code generation and model-based/driven testing as well as the complexity aspects of MDE. By applying a similar design to [56]'s "paired questions", in which they aimed to explore the balance between the types of positive and negative effects of MDE, participants were asked about the consequences of MDE.

Due to the growing complexity of software, it is generally agreed that the only realistic way to manage this complexity is using appropriate methods of abstraction with modeling [155] and model-driven code generation is an important aspect to improve productivity in MDE [46]. However, an interesting result in [55] is that participants working on real-time systems agree that their organizational culture does not endorse software modeling due to automatic code generation. Similarly, as in [50], UML is too complex or according to [51], there are lots of UML complexity problems as reported in previous studies (e.g., [13, 34, 156]). In this question, to address the balance, for example, in model-driven code generation part, the first

statement mentions about the possible positive consequences of MDE on “abstraction”, whereas the second statement mentions about the possible negative consequences of MDE on “abstraction”. Similar approaches are applied for both model-based/driven testing and complexity. As seen in Figure 39, all responses are depicted according to response percentage (in y-axis) and the mean value is also presented with its corresponding color at the below of each statement.

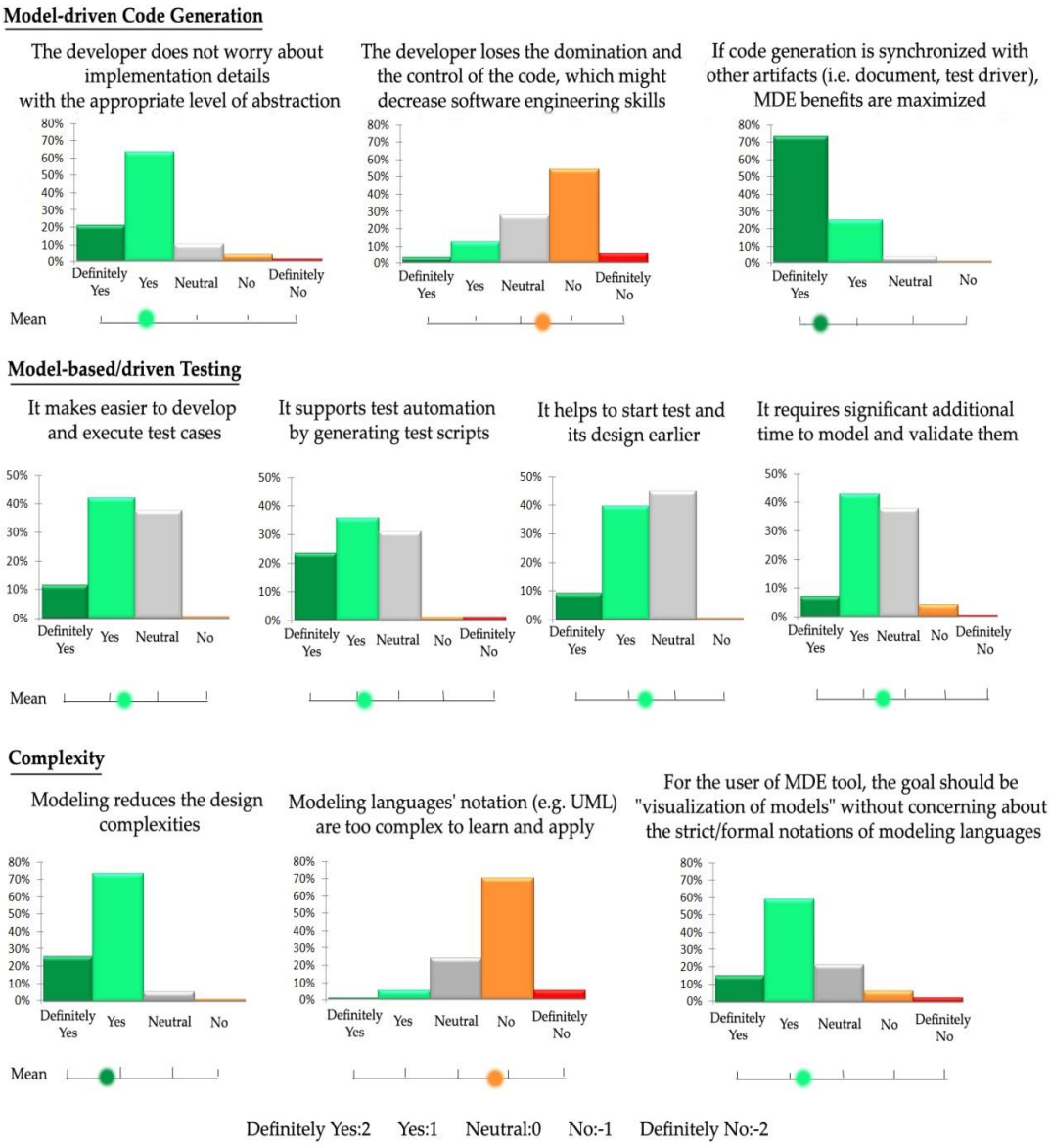


Figure 39: Survey - Consequences and complexity aspects of MDE

Summary

RQ1.1.1 - Summary of the current state of modeling: Software modeling (either formal or informal) is widely used by many embedded professionals (89%). Although there is a wide spectrum in terms of the latest software modeling approaches, languages and tools used by practitioners in different industrial sectors, the C programming language, UML (as the primary modeling language), and Eclipse-based tools seem to be the most popular choices.

As expected, different engineers and companies use software modeling approaches in varying degrees, which usually depends on their experience and project needs. Software modeling is conducted from informal sketches (on paper or by using a modeling tool) to formal models using sophisticated modeling tools.

The majority of respondents use UML. However, depending on the type of industrial sector, a general-purpose modeling language such as UML is usually not sufficient to meet the specific requirements and other modeling languages are used in those cases, e.g., the AUTOSAR language (in "Automotive & Transportation"), models based on the Markov chains (in "Consumer Electronics"), and various other DSLs (e.g., AADL for "Defense & Aerospace").

A variety of modeling tools are used, the most popular ones being the "Eclipse-based" family of tools, followed by "Microsoft Visio"**,** where the ratio of "Other" answers for this question is ~18%.The respondents, who use UML, use different diagrams to varying levels. The most used diagram types are sequence diagrams, state-machine diagram, and class diagram. The majority of respondents use modeling in the systems/software design phase, followed by implementation and requirements/systems analysis phases of SDLC.

RQ1.1.2 - Summary of the current state of MDE and its adoption: Notice that ~30% of all participants use MDE approaches. The respondents reported that they use software modeling and MDE for mostly documentation and code generation, and then for understanding and analysis the problem domain at an abstract level.

To assess MDE maturity levels, we adopted from the literature a 5-level maturity model. Based on that model, we found that the majority of the participants (57%) are in the Level 4, indicating that they have completed multiple MDE projects. This is a generally good sign for the embedded software industry. The other aspect that we explored in terms of the current state of MDE and its adoption was the motivations for adopting MDE. The top motivators were "cost savings", "shorter development time", "reusability" and "quality improvements".

RQ1.1.3 - Summary of the achievements, challenges and consequences of using MDE: In terms of achievements and benefits of MDE, "cost savings", "ensuring source code & design model compatibility", and "shorter development time" were reported the most. In terms of challenges, tool support, and more specifically difficulties with model-level debugging and usability issues of tools were stated as the most impeding issues.

In terms of positive consequences and impacts, model-driven code generation was generally reported to be a beneficial outcome of MDE. Many respondents believed that model-based/driven testing makes it easier to develop and execute test cases by also supporting test automation via test scripts; however, although it helps to start to test and its design earlier; it requires significant additional upfront efforts to model and validate them. The embedded software community largely believes that modeling reduces design complexities and modeling languages are not that complex as reported in many studies.

Appendix B.3 – Implications for Practitioners, Researchers and Educators

Implications for practitioners:

MDE is popular in the embedded software industry and benefitting from what others are doing: We found that software modeling is widely used (89% across the participants' population), across a diverse range of embedded software industries to better handle the growing complexity of their software-intensive products. Embedded software professionals

use different modeling languages, programming languages, modeling environments with different motivations and face different challenges. By looking at the achievements and challenges of MDE, this empirical evidence will help embedded software professionals, who are thinking about adopting MDE in their projects, to know common practices other adopted for their context. In other words, they can use modeling and MDE selectively according to their needs (i.e., motivations or SDLC phases in which modeling is used)

There is a wide variety of practices, motivations and tools: Although we consulted with several industrial practitioners and used our personal industrial experiences when designing the closed-ended questions in the survey, we had a lot of "Other" answers than we expected (e.g.., modeling language (See Q14), programming language (See Q15) or modeling tool (See Q16)). This showed that there is a wide spectrum of in terms of the technology used for software modeling and our results might also help software professionals to get awareness of these new technologies.

Need for better tool support: Tool support is one of the most encountered MDE challenges (See Q25). We have also observed several shortcomings in terms of tool support (See Q26). Supporting MDE with appropriate tools increases modeling achievements. Therefore, we suggest MDE tool vendors to invest more efforts in development and improvement of these tools and including the features that practitioners mentioned in this survey (such as "model verification /validation" and "model quality").

Implications for researchers:

Need for more MDE techniques across all SDLC phases: In Q18, we found that the majority of respondents use modeling in the systems/software design phase, implementation and systems analysis phases. Modeling is used not that widely for integration and testing, although there are lots of academic advances and novel techniques in these areas. This makes us think whether there are issues which decrease the practical application of those techniques in industrial settings. Researchers are encouraged to look into these issues.

Focusing on what industry uses the most: Documentation, code generation and understanding of problems at higher abstract levels were reported to be the most popular reasons for using MDE. Thus, it is recommended that researchers work on developing more industry-relevant tools and techniques in these areas.

Addressing the MDE challenges: Tool support and modeling expertise in the companies were the most encountered challenges. Researchers can work to develop better research-prototype tools and also collaborate with industry to improve modeling expertise of engineers.

Implications for educators:

Improving the software modeling educations: Our results also have implications for software modeling educations, e.g., [157-159], and educators. Our survey results suggest implications for the way in which software modeling is taught (from Q12). Some respondents (especially the electrical and electronics engineering graduates) reported that they have mostly learned software modeling after getting the job or employment (i.e., after graduation, during the job or with some training). Some respondents who were computer and software engineering graduates also reported that they have learned some modeling techniques during their undergraduate studies, but not at the application level in the industrial context.

MDE is not just the analysis and design phase: A typical university SE course teaches a top-down fashion, in which models are first developed for analysis and then refined into design,

implementation and test phases of SDLC. In most software modeling courses, the students study how to design and develop a software system using software modeling techniques, but the focus is generally on the analysis and the design phases and there is a missing part while translating these software models into executable code. Extensions of these courses could focus on the important concepts in MDD, the state-of-the art and practices of MDE approaches, and the corresponding challenges in software modeling projects. Therefore, we believe that the given courses on modeling (or the curriculum) might be updated or enhanced after a further analysis of the results in our survey, which suggest topics that could have been widely covered or emphasized.

Appendix B.4 – Limitations and Threats to Validity

Construct validity: *"Construct validities are concerned with the extent to which the objects of study truly represents theory behind the study"* [71]. In other words, did this survey measure the real-world software modeling approaches in embedded software industry or not. Data were collected from different sources (different countries, different industrial sectors, etc.) in order to avoid mono-operation bias.

When people feel being evaluated based on what they think, they might deflect their answers. In order to mitigate these, participants were informed prior to the survey that our motivation was to take a snapshot of the embedded software industry and that we will not collect any identifying information. Therefore, for the sake of objectiveness, the survey is completely anonymous.

In the measurement strategy, what was done was common with other survey studies—counting the votes for each question and then making statistical inferences. It is believed that results based on such voting data can, to a certain extent, reflect the opinions of the majority of embedded professionals.

Last but not the least is the issue and definitions of MDE vs. MBE as understood by that participants. This threat was tried to be reduced by making sure the participants understood and distinguished the terminologies by providing them the definitions mentioned by [16] (See [152]). In order to prevent any misunderstanding and potential threat in this terminology a pilot phase of the survey in which several practitioners filled the survey was conducted and then we met with them to assess their common understanding of the terminologies regarding MDE, MDD and MBE. However, the definition provided by [16] sadly still leave room for subjectivity and we could not come up with better definitions at 2015, while designing survey questions. Thus, this issue stays as a potential threat, e.g., a given practitioner might in fact use MBE, even though s/he stated to use MDE. Moreover, although there was no specific feedback on the pre-given answer set for some items (i.e., "model checking capabilities", "M2M transformation"), as the terms have not been explicitly specified, there might be different interpretations and we could not be sure that the all respondents have the same understanding.

Internal validity: *"Internal validity reflects whether all causal relations are studied or if unknown factors affect the results"* [71]. Using a pilot study improved instrumentation. The survey took approximately 2-10 minutes to fill out depending on the modeling usage type (i.e., for no modeling usage, it takes ~2 minutes) and was intended to be filled out once by every participant. This reduces the likelihood for learning effects. Moreover, since the wording and terminology used in the survey should be easily understandable to get high quality data and to prevent misunderstandings, which could lead to invalidity, the pilot includes embedded

software professionals with different native languages (English, Turkish, French and Taiwanese), different software engineering roles and different experiences.

External validity: *"External validity is concerned with the extent to which the results of this study can be generalized"* [71]. In order to decrease the effect of possible dominant participant number in a specific sector due to authors' previous and current work experiences' network (i.e., defense & aerospace, consumer electronics, academia), the survey has been distributed to embedded software professionals via various social network sites in all around the world for different industrial sectors. Therefore, we have done our best to reach the participants with a variety of different demographics representative for the embedded software industry. The sample size is quite high compared to previous surveys. While we did our best to achieve an even geographical distribution, the samples were mostly based from Europe (66%), followed by Asia (17%) and then the Americas (14%). Due to researchers' location, ~40% of respondents are from Turkey, which has may led to bias in the results. Nevertheless, note that non-probabilistic sampling design was used and thus external validity is limited. To address this, demographic information of the participants and companies were presented so that the readers will be able to evaluate the applicability in different contexts.

Conclusion validity: *"Conclusion validity of a study deals with whether correct conclusions are reached through rigorous and repeatable treatment"* [71]. This study was designed by one author, who has both researcher and practitioner hat and two other researchers from two different institutions; therefore the risk for "fishing" on the results is reduced. It was attempted to conclude that the modeling approaches in embedded software industry have organizational and economical aspects as well as purely technical ones. For each RQ, the bias by seeking support from the statistical results was reduced. Thus, all the conclusions in this survey are strictly traceable to data. Moreover, to increase transparency, the raw survey data is made available online [77] for other researchers to validate and replicate. Furthermore, the reliability of this study was improved by conducting pilot studies prior to the survey execution.

APPENDIX C – PRE-INVESTIGATED MODELING PATTERNS' VISUALIZATIONS

In order to show the necessity of generating a new derived attribute on the existing survey data, "modeling languages" vs "modeling languages set" is a good example. If the bars stacked chart of "purposes set" and "modeling languages" is depicted, there are lots of combinations of modeling languages as seen in Figure 40 (Notice that "purposes" were used to generate "purposes set", which includes four choices: model-driven with code generation or MBT, model-driven without code generation or MBT, no model-driven with documenting design and no model-driven without documenting design. See Section 4.3).

Figure 40: Bars Stacked Chart- "Purposes Set" vs "Modeling Languages"

However, whenever a derived attribute as a "modeling languages set" is used as a y-axis of the previous bars stacked, the output is as in Figure 41.

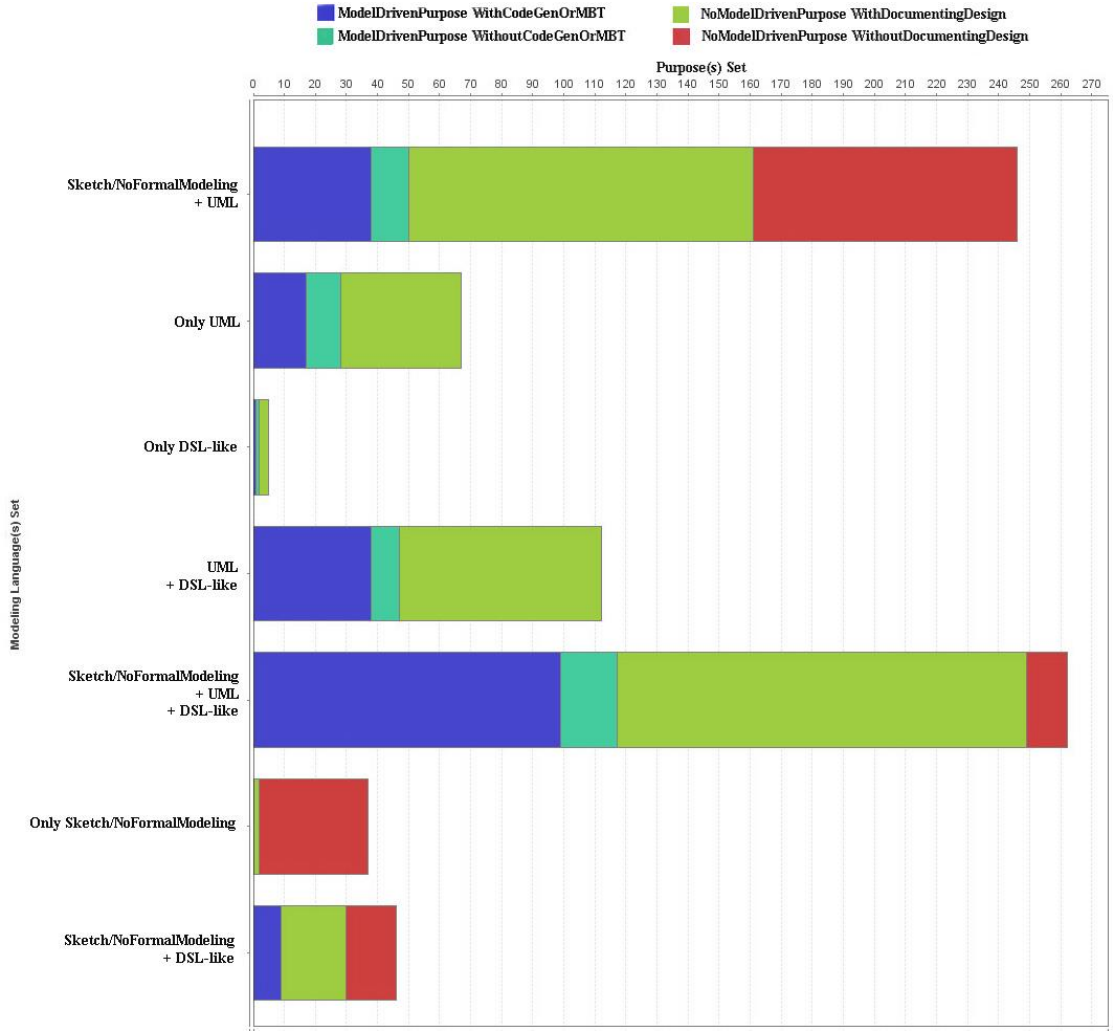


Figure 41: Bars Stacked Chart- “Purposes Set” vs “Modeling Languages Set”

After generating three derived attributes (i.e., for purposes, modeling languages and medium types) as reported in Section 4.3, the scatter chart for modeling languages set versus medium types set with purposes set color column is depicted as in Figure 42.

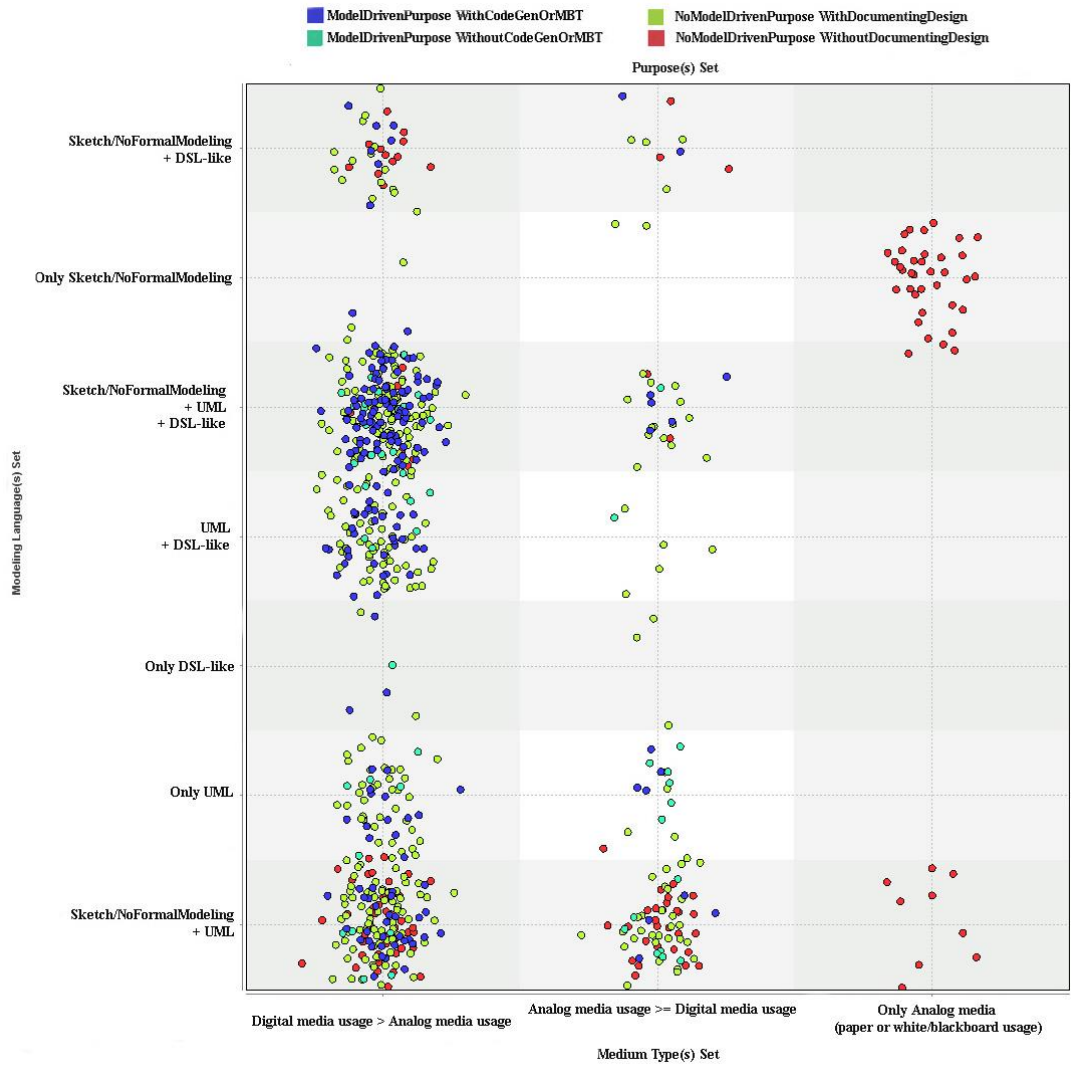


Figure 42: Scatter Chart- "Modeling Languages Set" vs "Medium Types Set" with "Purposes Set" color column

After the analysis on Figure 42, further investigation was done on "NoModelDrivenPurpose WithDocumentingDesign" item on purposes set to differentiate descriptive and prescriptive usage in these groups (i.e., ~38,5% of the survey data). Then, it was observed that the existence of "Implementation" or "Testing" phases of SDLC sub-categorizes this item in the purpose set as descriptive ones (13,7%) and prescriptive ones (24,9%). By this way, nine modeling patterns were pre-investigated as reported in Section 4.3.

APPENDIX D – QUESTIONNAIRE USED IN MULTIPLE CASE STUDIES

Questionnaire – Part A: Modeling Stakeholder Demographics	
1	What is your highest academic degree? ○ PhD ○ MSc ○ BSc ○ High School or lower
2	What is (are) your university degree(s)? (Attribute: STAKH-UNVRS) ☐ Computer Science ☐ Computer Engineering ☐ Software Engineering ☐ Information Systems ☐ Electrical/Electronics Engineering ☐ Industrial Engineering ☐ Mechanical/Mechatronics Engineering ☐ Mathematics ☐ Other: (Please specify)
3	What is (are) your current position(s)? (Attribute: STAKH-POSTN) ☐ Software Developer/Programmer ☐ Software Designer ☐ Software Architect ☐ Software Tester ☐ Systems Engineer ☐ Requirement Engineer ☐ Business Analyst ☐ Project Manager ☐ Quality Assurance Engineer
4	How many years of work experience do you have in software development? (Attribute: STAKH-EXPRN) ○ Less than 2 years ○ 2-5 years ○ 6-10 years ○ 10+ years
5	How many years of modeling experience do you have in software development, if any? (Attribute: STAKH-EXPRN) ○ None ○ Less than 2 years ○ 2-5 years ○ 6-10 years ○ 10+ years
Questionnaire – Part B: Company/Project Demographics	
6	What is the target sector of the product(s) developed? (Attribute: DOMN) ☐ Automotive & Transportation ☐ Consumer Electronics ☐ Defense & Aerospace ☐ Finance & Banking ☐ Healthcare & Biomedical ☐ IT & Telecommunications
7	What is the number of employees working in software engineering roles? (Attribute: STAKH-T_SIZE) ○ 1-10 people ○ 11-100 people ○ 101-200 people ○ 201-500 people ○ 500+ people
8	What is the size of your typical software development team in a project? (Attribute: STAKH-T_SIZE) ○ 1-4 people ○ 5-9 people ○ 10-19 people ○ 20-50 people ○ 50+ people
9	Which programming languages do you use while modeling? *(if applicable)* (Attribute: PL) ☐ Ada ☐ C ☐ C++ ☐ C# ☐ Delphi ☐ Java ☐ MATLAB ☐ Objective-C ☐ Smalltalk ☐ Any programming language for BPM such as BPEL ☐ Other: (Please specify) ○ Not applicable

10	How can you describe your developed software in terms of hardware closeness? (Attribute: PL-HW_CL) ○ Very high (e.g., firmware or BSP applications) ○ Medium (e.g., communication/middleware applications) ○ High (e.g., DSP applications) ○ Low (e.g., UI applications) ○ Not applicable
11	How often do you use software modeling in your software development life cycle? *(informal or formalized: i.e., sketches or models)* (Attribute: RIGOR) ○ Never ○ Sometimes ○ Often ○ Most of the time ○ Always *Evaluator Note:*

Questionnaire – Part C: Modeling Practices *(If you do NOT use any software modeling, you do NOT need to continue)*

12	Where/How did you learn modeling? (Attribute: STAKH-W/H_L) ☐ University (e.g., from software engineering, computer science courses) ☐ Formal corporate training ☐ On your own (e.g., from books, in the job) ☐ Other: (Please specify)
13	What medium do you use to create the diagram? (Attribute: MEDM)

		Never	Sometimes	Often	Most of the time	Always	Tool?
Digital	PC (using software modeling tools)	○	○	○	○	○	
Digital	Tablet/Smartphone	○	○	○	○	○	
Analog	Paper	○	○	○	○	○	
Analog	White/blackboard	○	○	○	○	○	

14	Which modeling language(s) do you use for modeling? (Attribute: RIGOR-ML) ☐ Sketch/No formal modeling language (Free-format) ☐ UML ☐ Any UML extensions (profiles) such as MARTE ☐ Domain Specific Language (DSL) provided by the tool provider or your own design) (Please specify) ☐ Systems Modeling Language (SysML) ☐ Service Oriented Architecture Modeling Language (SoaML) ☐ Any Business Process Modeling (BPM) languages such as Business Process Modeling Language (BPML) ☐ MATLAB modeling utilities ☐ Other: (Please specify) *Evaluator Note:*
15	Do you archive your diagrams? (Attribute: MEDM-ARCHV) ○ Never ○ Sometimes ○ Often ○ Most of the time ○ Always *Evaluator Note: archive mechanism, if any (e.g., analog vs digital?):*

16 Which diagrams do you use and in which SDLC phase(s)? (Attribute: SDLC)

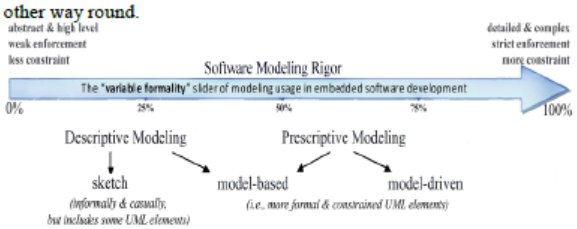
(SDLC Phases: Preliminary/Systems Analysis | Business Process Analysis | Systems/Software Design | Implementation | Testing | Integration | Installation & Deployment | Maintenance)

	Never	Sometimes	Often	Most of the time	Always	SDLC phase(s)
Sketch (free-format)	○	○	○	○	○	
Use Case Diagrams	○	○	○	○	○	
Communication Diagrams	○	○	○	○	○	
Deployment Diagrams	○	○	○	○	○	
Flowcharts/Diagrams	○	○	○	○	○	
Sequence Diagrams	○	○	○	○	○	
Package Diagram	○	○	○	○	○	
State Machines/Charts	○	○	○	○	○	
Activity Diagrams	○	○	○	○	○	
Class Diagrams	○	○	○	○	○	
Object Diagrams	○	○	○	○	○	
Diagrams based on DSL	○	○	○	○	○	
BPMN/EPC	○	○	○	○	○	

Please read the following definitions before proceeding with the rest of questionnaire

Terminology: Descriptive vs Prescriptive && Sketch/Model-based vs Model-driven

In terms of terminology, model-driven development (**MDD**) uses models as the primary artifact of the development process. Usually, in MDD, the implementation is automatically generated from the models. Model-driven engineering (**MDE**) is a superset of MDD since it encompasses other tasks of a complete software engineering process like testing and maintenance (i.e. documentation). On the other hand, model-based engineering (**MBE**) is a process, in which software models still play an important role although they are not necessarily the key artifacts of the development. For example, designers specify the models (i.e. by using paper or modeling tool), but then these models are directly handed out to the programmers to manually write the code (no auto generation). Therefore, all model-driven processes are model-based but not the other way round.



Moreover, if your modeling approaches includes both sketching and/or some selective UML elements (e.g., if you do not obey any strict UML rules, but use it selectively and informally with some UML elements as causally), and you do not have any automatically generated artifact (e.g., source code, test driver, documentation or simulation), you are still at the descriptive modeling side (e.g., sketching or model-based) as seen from the **"variable formality"** slider of modeling usage.

17 When you write code, document, test or simulate, to what degree do you use model driven techniques? (Attribute: RIGOR)

○ Never ○ Sometimes ○ Often ○ Most of the time ○ Always

Evaluator Note:

18	What is (are) your motivation(s) and benefit(s) while modeling? (Attribute: PURPS & BENFT) ☐ Manage complexity / Understanding a problem at an abstract level ☐ Team collaboration / Communication ☐ Shorter development time ☐ Cost savings ☐ Test effectiveness ☐ Model simulation ☐ Quality Improvements ☐ Productivity ☐ Maintainability ☐ Traceability ☐ Reusability ☐ Reliability ☐ Portability ☐ Guaranteeing documentation and model compatibility ☐ Guaranteeing source code / test driver and model compatibility *Evaluator Note:*	

APPENDIX E – EVALUATOR NOTES/OBSERVATIONS & RESULTS

Any (formalized or informal) software modeling usage in SDLC?	
When writing code, document, test or simulate, to what degree using model driven techniques?	
Medium type(s)? Analog usage>=Digital usage?	
Archivability	
Modeling Language set?	
SDLC	
Modeling purpose set?	

Results:

Current (Interview/Observation)		**According to the Model**	
Pattern	Culture	Pattern	Culture

What similar profiles are doing

Recommendations

APPENDIX F – EVALUATION FORM TEMPLATE

When you think about the presentation you took about *"Modeling patterns and cultures of embedded software development project"*, does our model really reflect your current modeling pattern and culture? In other words, did this produce expected and relevant results for you? Please elaborate your answer by indicating the differences and similarities.

In that sense, do you think that the model is helpful? Please elaborate your answer.

Have you ever been experienced or used such a model before? In other words, do you think that this model is better than what was available previously or not?

Do you think that learning what your competitors (i.e., similar demographics) are doing while modeling might affect your future modeling practices? Please elaborate your answer.

Do you think that the recommendations, which our model gave you, is useful or not? Please elaborate your answer.

CURRICULUM VITAE

Personal Information

Deniz Akdur

Date of Birth: 12.02.1981

deniz.akdur@gmail.com

https://www.linkedin.com/pub/deniz-akdur/10/a90/865

https://www.researchgate.net/profile/Deniz_Akdur

Education

Degree	Institution	Year of Graduation	CGPA
PhD	METU, Information Systems	2018	4.00/4.00
MSc	METU, Information Systems	2009	3.70/4.00
BSc	Bilkent University, Computer Science	2004	3.62/4.00
High School	İzmir Buca Anadolu Lisesi	1999	5.00/5.00

Experience

Year	Organization	Position
2009-Present	ASELSAN, Turkey	Lead Software Engineer
2008-2009	VESTEL, Turkey	Software Architect
2004-2008	Cabot Communications, UK & Turkey	Software Architect / Senior Software Engineer

Specialties

- ✓ Software Engineering
- ✓ Embedded Systems & Software
- ✓ Software Modeling
- ✓ Industry-Academia Collaborations
- ✓ Software Quality Management
- ✓ Innovation Management & Entrepreneurship

Languages

- ✓ Turkish (Native)
- ✓ English (Full professional proficiency)
- ✓ French (Limited working proficiency)

Publications

1	Akdur, D., Demirörs, O., & Garousi, V. (2017). Characterizing the development and usage of diagrams in embedded software systems. Paper presented at the 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Vienna, Austria.
2	Akdur, D., Garousi, V., & Demirörs, O. (2017). Cross-factor analysis of software modeling practices versus practitioner demographics in the embedded software industry. Paper presented at the 6th Mediterranean Conference on Embedded Computing (MECO), Montenegro.
3	Akdur, D., Özpolat, E., & Başıbüyük, T. (2017). Model Driven Engineering of Communication Protocol Artifact with Design Pattern Usage in Distributed and Real-Time Embedded Systems: An Industrial Experience. *International Journal of Engineering Science and Application (IJESA), 1(3),* 91-98.
4	Akdur, D., & Garousi, V. (2015, February 9-11, 2015). *Model-Driven Engineering in Support of Development, Test and Maintenance of Communication Middleware: An Industrial Case-Study.* Paper presented at the International Conference on Model-Driven Engineering and Software Development (MODELSWARD).
5	Akdur, D., & Demirörs, O. (2017). Multiple Case Studies to Validate Modeling Patterns and Cultures of Embedded Software Development Projects, Technical Report: METU.
6	Akdur, D., Garousi, V., & Demirörs, O. (2015, Last accessed: Nov. 27, 2016a). MDE in embedded software industry, Technical Report. *METU II-TR-2015-55, https://dx.doi.org/10.6084/m9.figshare.4262990.*
7	Akdur, D., Garousi, V., & Demirörs, O. (2016) *Gömülü Yazılım Endüstrisinde Kullanılan Yazılım Modellemesi ve Model-Güdümlü Tekniklerde Türkiye'nin Dünyadaki Yeri*. Paper presented at the 10th Turkish National Software Engineering Symposium (In Turkish: Ulusal Yazılım Mühendisliği Sempozyumu (UYMS)), Turkey.
8	Akdur, D., Garousi, V., & Demirörs, O. (2015) *Gömülü Sistem Mühendisliğinde Kullanılan Yazılım Modellemesi ve Model Güdümlü Teknikler Anketi: Türkiye Sonuçları.* Paper presented at the 9th Turkish National Software Engineering Symposium (In Turkish: Ulusal Yazılım Mühendisliği Sempozyumu (UYMS)), Turkey.
9	Akdur, D., & Özdemir, Ç. (2014) *Gerçek Zamanlı Gömülü Sistemlerde Yeniden Kullanılabilir ve Yapılandırılabilir Yazılımların Kaliteye Etkisi: Radar Projeleri Destek Kütüphaneleri.* Paper presented at the 8th Turkish National Software Engineering Symposium (In Turkish: Ulusal Yazılım Mühendisliği Sempozyumu (UYMS)), Cyprus