Characterizing the Development and Usage of Diagrams in Embedded Software Systems

Conference Paper · August 2017

DOI: 10.1109/SEAA.2017.13

3 authors:

Deniz Akdur
ASELSAN Inc.
29 PUBLICATIONS **171** CITATIONS
SEE PROFILE

Onur Demirors
Izmir Institute of Technology
236 PUBLICATIONS **2,084** CITATIONS
SEE PROFILE

Vahid Garousi
Queen's University Belfast
199 PUBLICATIONS **4,616** CITATIONS
SEE PROFILE

Characterizing the development and usage of diagrams in embedded software systems

Deniz Akdur
ASELSAN Inc., Ankara, Turkey
Department of Information Systems, Informatics Institute
Middle East Technical University (METU)
Ankara, Turkey
denizakdur@aselsan.com.tr

Onur Demirörs
Department of Information Systems, Informatics Institute
Middle East Technical University (METU)
Ankara, Turkey
demirors@metu.edu.tr

Vahid Garousi
SnT center, University of Luxembourg, Luxembourg
garousi@svv.lu

***Abstract*—To cope with growing complexity of embedded software, modeling has become popular. The usage of models in embedded software industry and the relevant practices usually vary since the purposes of diagram development and usage differ. Since a large variety of software modeling practices used in embedded software industry, it is important to understand its state-of-the-practice and its usage degree while investigating the relations between its attributes (e.g., modeling rigor, purpose, code correspondence, stakeholder, medium used while modeling, etc). To achieve this, we have designed and conducted a survey in our earlier work. In this paper, we present a conceptual model of development and usage for software modeling that is based on the findings of this survey and incorporates expert opinions. The conceptual model, which characterizes the attributes of a diagram development and usage, will help to express the meaning of terms used by domain experts to discuss the problems and find the relationships between these attributes.**

Keywords—Embedded systems; embedded software; software modeling; sketch; model-based; model-driven; model-driven engineering; attributes of modeling

I. INTRODUCTION

Embedded software shapes our world and it is difficult to imagine day-to-day life without it. However, design, development and testing of software for modern embedded software are not trivial due to their multiple constraints across different dimensions of performance and quality, which makes their development more challenging.

Software modeling is a widely used approach in the embedded industry and a tool to manage complexity of these systems. However, the modeling approaches in embedded software vary since the attributes of a diagram development and usage differ among systems as well as among sectors. At one extreme, some practitioners use software modeling at a very informal level, where diagrams are sketched on a paper or a white board in order to help communicate ideas with colleagues. These diagrams are usually either discarded or quickly become inaccurate since they are not kept updated along with the source code [1]. At the other extreme, modeling turns into programming with automated generation of code from models and the corresponding diagrams have more lifespan and archivability. Moreover, different departments within the same company can use models for different purposes with different stakeholders within different software development life cycle (SDLC) [2].

To understand the state-of-the-practice of modeling in embedded systems industry, we have designed and conducted a survey in spring of 2015 [3]. The survey showed that the embedded software professionals use modeling approaches in varying degrees (e.g., either as a sketch or a model) with different constrainment and enforcement depending on their needs. All of the usages could be effective depending on the purpose of modeling. The results also showed that there is a modeling rigor depending on the attributes of a diagram development and usage in embedded software industry.

Based on the results of the survey [3] and our Action Research (AR) project [4] as well as others incorporating different classifications about software modeling, this study systematically presents a conceptual model of development and usage for software modeling. The conceptual model is enriched by expert opinions (via semi-structured interviews), which is one of the crucial steps to investigate the attributes of a diagram development and usage. The model attempts to clarify the meaning of various, usually ambiguous terms, and ensure that problems with different interpretations of the concepts cannot occur. With the help of this model, the relations between these attributes are also investigated. These correlations and our other research findings will be used to investigate different software modeling usage patterns in the embedded industry besides constructing a model for a recommender system. By guiding a modeling stakeholder to determine the optimum degree of software modeling pattern for a cost-effective approach, we believe that both embedded software professionals and also researchers benefited from our results and this will hopefully encourage more academia-industry collaborations in this area.

The remainder of this paper is structured as follows. Section 2 gives the terminologies used in our studies and a brief overview of our survey. Section 3 discusses the research methodology. Section 4 presents the conceptual model and the attributes of a diagram development and usage. Section 5 presents the limitation and threats to validity. Finally, Section 6 concludes this study by giving future directions.

II. BACKGROUND

A. Terminologies and Concepts

In the literature, there are various terminologies in the context of software modeling. According to Brambilla et al. [5],

model-driven development (MDD) treats models as the primary artifact of the development process. Usually, in MDD, the implementation is automatically generated 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, 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., on paper or by using modeling tools), but then these models are directly handed out to the programmers to manually write the code (i.e., there is no automated code generation). In that sense, MBE refers to a 'lighter' version of MDE [6]. Therefore, all model-driven processes are model-based but not the other way round. Fig. 1 (adopted from [6]) visually depicts the concepts of MBE, MDE and MDD as discussed above.

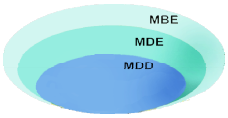

Fig. 1. Venn diagram - The relationship among MDD, MDE and MBE

In this study, the terminology offered by [5] for differentiating between "model-based" and "model-driven" approaches (i.e., for prescriptive modeling) is enriched and synthesized with the concept given during the talk of Bran Selic [7] (i.e., for descriptive modeling) since we think that it reflects the best real industrial context after getting our recent survey results (i.e., about sketching). Accordingly, software modeling usage has two main modeling categories (i.e., descriptive and prescriptive modeling) and four main modeling patterns (i.e., no modeling, sketching, model-based and model-driven) as depicted in Fig. 2.

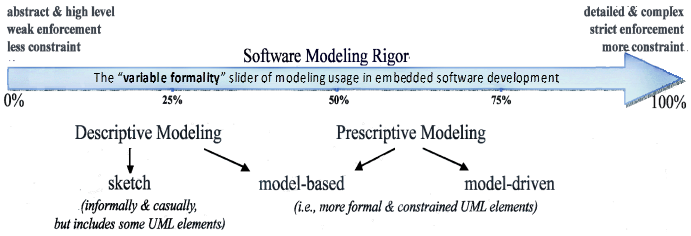

Fig. 2. Terminology used in this study

Descriptive models classify actual objects, events, and processes into categories; whereas prescriptive ones specify what is expected of systems components and how to develop them [2]. Descriptive models are created from observations for a specific intent and once the intent is satisfied, they might lose its importance. Moreover, when the intent change, the model might become obsolete and needs to be updated. On the other hand, in the context of prescriptive models, the subject does not exist yet and the model is derived from the information available at that time. The primary purpose of descriptive models are understanding and communication, while the primary purpose of prescriptive models is development [2]. That distinction provides a formal justification between analysis and design models, which – we think that - depends on the purpose of modeling and directly affects the modeling rigor (formality). As seen from Fig. 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 attributes of a diagram, software modeling usage degree varies.

B. *A Brief Overview of our Recent Survey*

Although there have been a few prior surveys related to modeling in the embedded software industry (e.g., [8-10]), they have either focused on only one aspect of modeling, (e.g., the use of Unified Modeling Language (UML)) or the use of formal models, or modeling in regional contexts (e.g., UML and MDE in Brazil or in Greece). There are also some surveys, whose participants were involved with model-based/driven techniques on a single sub-domain of embedded systems (i.e., the automotive industry [8]). However, our survey took a larger and more holistic scope on the subject and with a higher scale (i.e., data from worldwide).

The identified target audience is anyone working in the embedded software industry, with a variety of different software engineering roles. In terms of descriptive statistics of the dataset gathered via the survey, most of the participants had "Software Developer/Programmer" role (69%); "Software Designer" (19%), "Software Architect" (17%) and "Software Tester" (12%) roles were the other frequent responses. The majority of respondents had 10+ years (52%), followed by 6-10 years (40%) of work experience. As for the modeling experience, the majority were in 6-10 years range (46%), followed by 10+ years of modeling experience (40%). More detailed demographics can be found in [3].

After analysis of responses, there are several interesting observations, which are directly related to this study:

- Software engineers use software modeling approaches in varying degrees, which usually depends on their needs. 11% of respondents have not been using any software modeling approaches (neither formal nor informal); whereas the remaining 89% are somehow (partially or fully) using software modeling.
- Although there is a wide spectrum in terms of the latest modeling practices in different industrial sectors, the C programming language and Eclipse-based tools seem to be the most popular choices. The majority of respondents use UML and the second most frequently reported response is "Sketch/No formal modeling language".
- Using modeling software on PC for modeling is the most used medium; whereas modeling using pen and paper is the next common approach depending on the purpose of modeling.

- 59% of all participants do not use any model-driven techniques; these participants are either using model-based approaches or using sketches/no formal modeling. The remaining 30% are model-driven users.
- The respondents reported that they use MDE for mostly documentation and code generation, and then for understanding the problem domain at an abstract level.

Further details on the results can be found in [3].

III. Research Methodology

The research methodology for this study is mixed: both quantitative and qualitative methods are used.

- We conducted a world-wide survey to understand the state-of-the-practice of modeling in embedded software industry. The research approach used in the survey is the Goal, Question, Metric (GQM) [11]. Stated using the GQM's goal template, while achieving the goal, our survey identified to what degree, why and how software modeling is conducted with its challenges and shortcomings (see Section II.B).
- The results on this survey lead us to create a conceptual model of development and usage for modeling in embedded software development. Based on standards on software engineering about modeling and incorporating different views and classifications [6, 12, 13], a conceptual model was developed with the light of the results taken from our survey [3] and our AR project about MDE in support of development, test and maintenance [4], in which the benefits and challenges of using MDE were investigated.

In other words, by synthesizing the views and classification found in the literature and also using the results taken by the survey and AR, this conceptual model is developed. We have also utilized expert opinions via semi-structured interviews to modify the model later.

IV. Conceptual Model and Attributes of a Diagram Development and Usage

A. Conceptual Model

In order to investigate the best real industrial context for modeling in embedded software after the interesting results of our survey (e.g., "Sketch/No formal modeling language" is the second most frequently reported response), it was decided to create a conceptual model on this subject. Since the completeness & correctness of conceptual model is very crucial 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, semi-structured interviews in different sectors were conducted over four months with 20 embedded software professionals. During the planning phase, it was necessary to decide from whom to take feedback. Due to the qualitative nature of this, it is recommended to select these professionals in the embedded systems based on differences instead of trying to replicate similarities [14]. This means that it is good to try to involve different industrial sectors, different roles, different experiences and different practices in embedded software.

The semi-structured interviews were conducted mostly in face-to-face meetings, but if it is 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). The day before the interview, the pre-designed model was sent to give him/her chance to think about and criticize these artifacts. The consulted software professionals list is given in Table 1.

Given their feedbacks and suggestions, the model was refined and updated by including new artifacts and changing the terminology for full industrial coverage (e.g., merging "preliminary/systems analysis" with "business process analysis" under "Analysis" phase of SDLC; or using "Formalized" instead of using "Formal" modeling language to get rid of any misunderstanding in the terminology). In this way, our conceptual model used information was obtained from our survey results, AR Project, similar related works [8-10, 12, 15-17] and finally feedbacks during semi-structured interviews. (Note that the attributes of a diagram development and usage in embedded software development, which is presented next is also dependent on these inputs.)

TABLE 1. SEMI-STRUCTURED INTERVIEWERS' DEMOGRAPHICS

Company / Organization	Target Sector	Experience*	University Degree(s)**
ASELSAN, Turkey	Defense & Aerospace	18	EE
		22	CENG
TAI, Turkey		13	CENG
Curtiss Wright, USA		23	EE
Airbus, Germany		16	EE
Vestel, Turkey	Consumer Electronics	21	CS
		18	EE, CENG
Cabot Comm., UK		19	CS
		14	EE
Arcelik, Turkey		15	EE
Novatek, Taiwan		16	EE, CENG
Vaisala, Finland		24	CS
Turk Telekom, Turkey	IT & Telecommunications	17	EE, CENG
Turkcell, Turkey		14	CS
Thoratec, USA	Healthcare & Biomedical	29	EE
		12	CS
Akbank, Turkey	Finance & Banking	14	CS
Garanti Tech, Turkey		16	EE, SE
Koc University, Turkey	Academic & Consultancy	13	CS
METU, Turkey		24	EE
Total: 20 software professionals, 358 years of work experience			
* Software development experience in years ** Computer Science (CS), Computer Engineering (CENG), Software Engineering (SE), Electrical/Electronics Engineering (EE)			

The final outcome for the conceptual model is given in Fig. 3. Note that this conceptual model is also descriptive model (i.e., for understanding and communication), in which there are at least some UML elements (e.g., some class diagram or use case diagram elements as inheritance or actors, but selectively and informally as in [18]).

We have decomposed the model into five conceptual areas, where "*Diagram*", which is developed and used in different SDLC, is the backbone of this conceptual model. According to the 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 our 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, we next explain these five conceptual areas by giving their corresponding references based on our previous studies (e.g., specific survey question and its answer set).

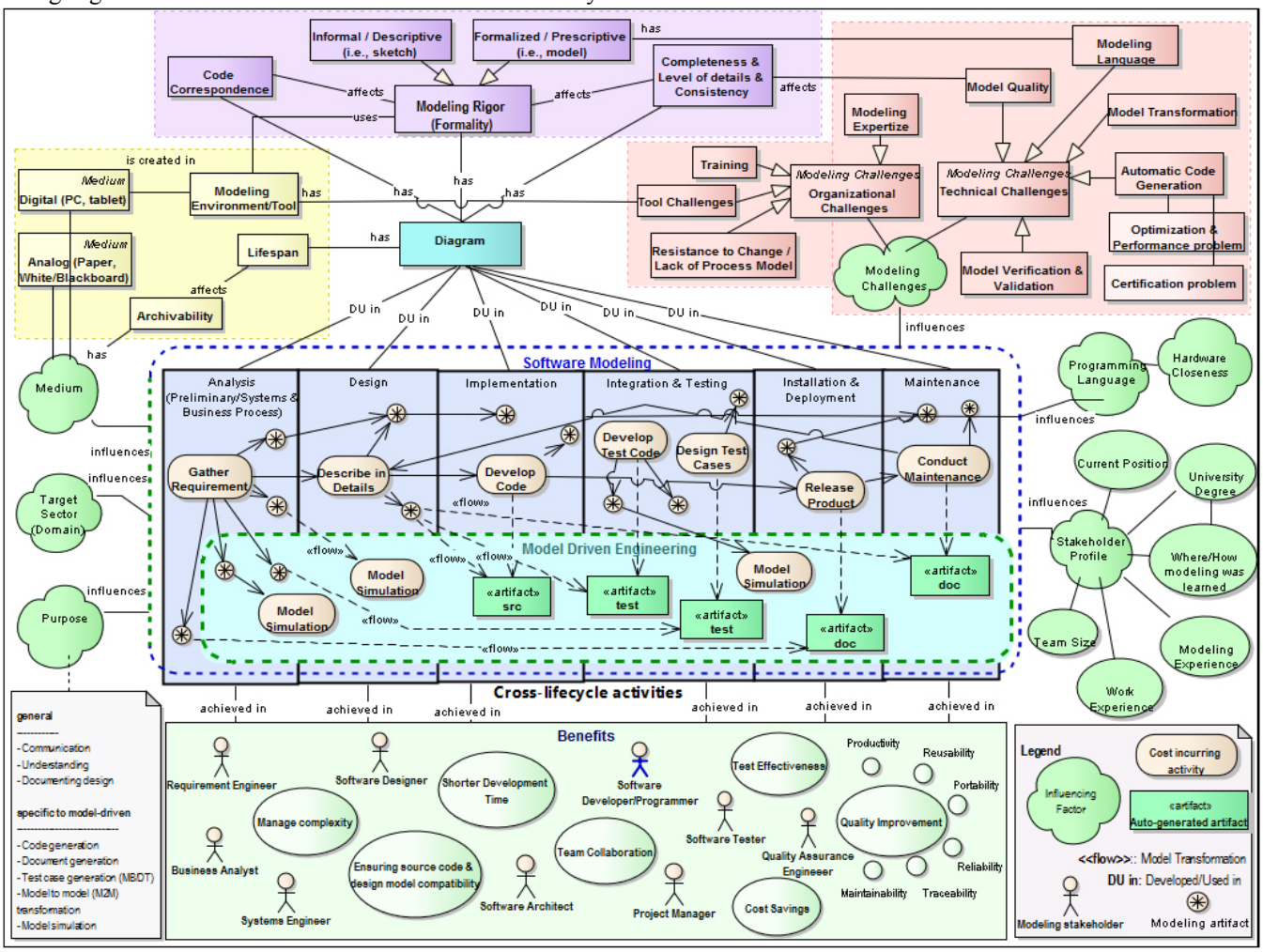

Fig. 3. Conceptual model of development and usage for modeling in embedded software development

- Area 1: Modeling rigor and modeling categories

On the upper middle part of the model, we have defined this area. 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. Our survey results showed that this formality affects the usage of modeling in varying degrees (i.e., Q10 and Q19). At that point, our terminology used in Section II.A. plays a critical role (i.e., descriptive modeling versus prescriptive modeling). As mentioned in Section II.A, "*the variable formality*" slider of modeling usage (i.e., modeling rigor) depends on these categories of software modeling.

Diagram has also *"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 [17], the rigor and styles of modeling using UML affect the model-code correspondence; the higher the similarity between models and the code, the higher the correspondence is. Moreover, our AR Project 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 given in our terminology and in [17], "*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, which has weak enforcement).

- Area 2: Benefits for modeling stakeholders

On the lower part of the model, we have defined the potential benefits of modeling for different stakeholders. 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, took some responses 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 related works analyzed, the most significant benefits in [9] were associated with *"Quality Improvement", "Portability"*, *"Maintainability"* and *"Productivity"*. On the other hand, according to [10], 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: Medium type used and its effects

In the left upper part of the model, we defined this area. As we investigated in our survey (i.e. Q13), 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 of the 5-point Likert-scale 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.

Our semi-structured interviews and also related works (e.g., [12]) showed that the medium type has a "*Archivability*" and this directly affects the "*Lifespan*" of this diagram. As reported in [12], 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 (i.e., Q16): 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%. As a comparison, in the dataset of the survey reported in [10], the majority (%50) used Matlab/Simulink/Stateflow, followed by Eclipse-based tools, Enterprise Architect, in-house tools and IBM Rational Software Modeler. 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: Modeling Challenges

In the right upper part of the model, we defined modeling challenges. Our survey results showed that there are different *"Organizational"* and *"Technical"* challenges while modeling, which the researchers can pick those as areas for possible improvements [3]. 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 took some responses [3]; however, *"Tool Challenges", "Modeling Expertize"* in the company and *"Resistance to Change*" are the most encountered challenges. Related works also mentioned such challenges. In [8], "Tool costs" and *"Training"* were seen as a negative aspect of MDE. In [9], 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, [15] pointed out the need of a longer training period so as to overcome the lack of UML expertise, which is also in parallel with the "*Modeling Expertize*" challenge in our survey. According to [10], "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*" attribute, which directly depends on modeling rigor. Moreover, as our survey and the 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"* challenge.

- Area 5: 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, 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 attributes of software modeling and their relations between each other plays a crucial role to find the best solution.

B. Attributes of a Diagram Development and Usage

With the help of the conceptual model, we identified the attributes of a diagram development and usage in embedded software development as seen in Fig. 4. Accordingly, there are 11 main attributes, where some sub-attributes affect its main attribute. Based on our previous studies' results, we also present the relations between these attributes to each other in this section.

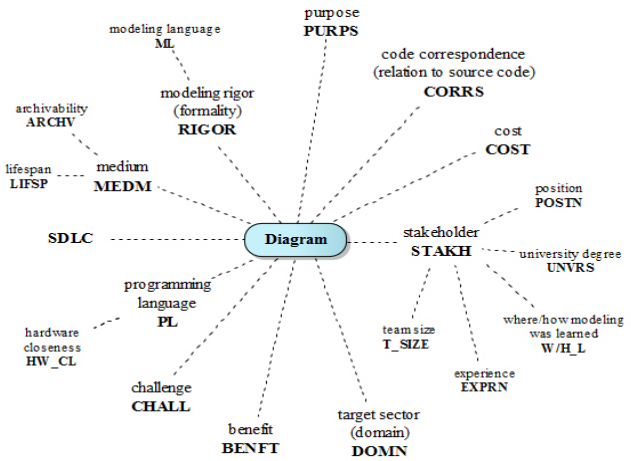

Fig. 4. Attributes of a diagram development and usage while modeling

As our survey results showed that RIGOR (i.e., modeling rigor) is affected by all other attributes (i.e., Q10 and Q19 results are correlated with these attributes), therefore it is crucial to analyze other attributes' 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 where modeling is used), BENFT (i.e., the benefits of modeling), CHALL (i.e., the challenges of modeling), PL (i.e., programming language used while modeling), 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 [3]. These attributes – somehow- influence RIGOR based on "*the variable formality*" slider, which explains the difference and the notions between descriptive (e.g., sketch) and prescriptive (e.g., model-based or model-driven) modeling. We plan to investigate their significance of each correlation coefficient individually.

Fig. 5 chart explains the relations of attributes 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 (i.e., either descriptive or prescriptive) are mapped, if applicable (e.g., there is no PL or DOMN attribute on this chart since it is not easy to put these attributes on this column-based category).

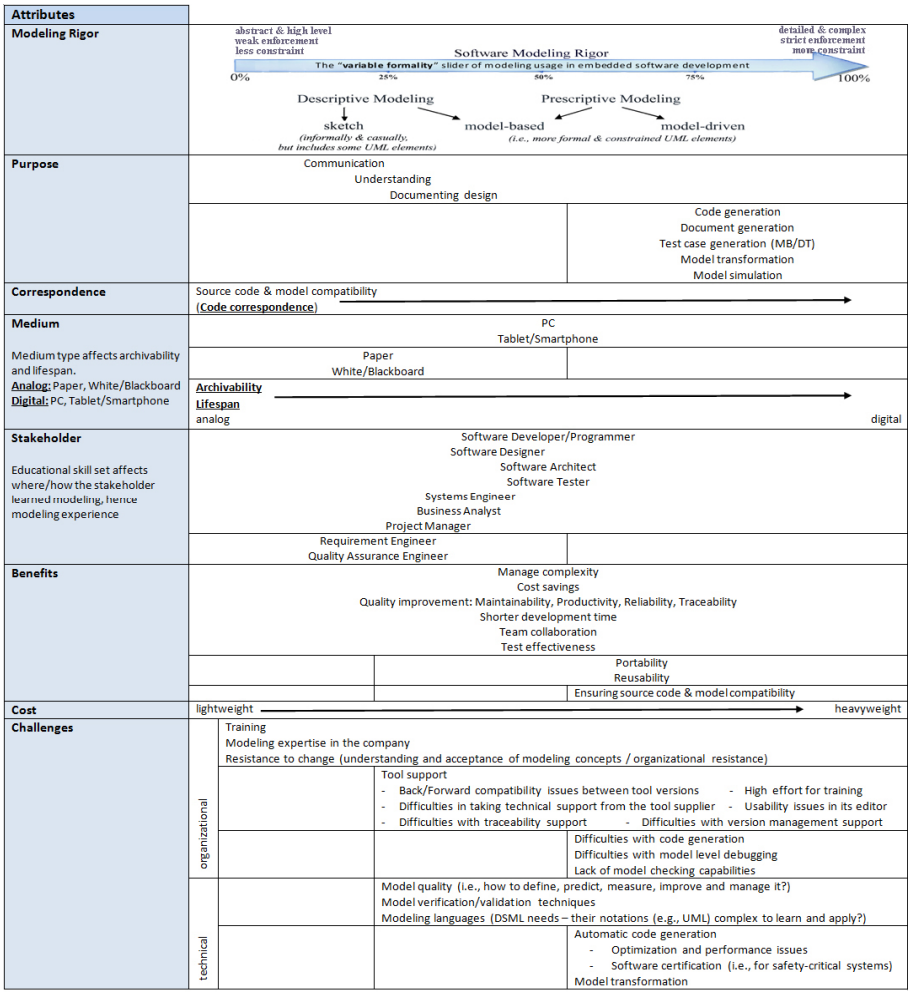

Fig. 5. Chart showing the relations between attributes

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 [17].

PURPS of the modeling and the category of software modeling are strongly related with the MEDM used. If there is no auto-generation of any software artifacts (e.g., no code, document or test scripts generation, as in “sketch” or “model-based”), analog media like paper or whiteboard are enough for communication or understanding a problem at an abstract level. It does not mean that model-driven users 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). In that sense, digital media are usable for all 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). Our 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.

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. In that sense, the digital mediums like PC or tablet/smartphone have advantageous on archiving and have longer lifespan. During our semi-structured interviews with embedded software professionals, 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. By this way, 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). Our 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 attributes of a diagram. Software engineering roles in Fig. 5 are identified by our survey, which has a wide range of embedded professionals including from developer to tester and project manager to quality assurance engineer. We should note that, as it has been established in studies on information quality (for example by Garvin [19]), people in different positions see and rate importance of different issues differently and in general have varying viewpoints on software engineering and related processes. Accordingly, our 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). We 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 were 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 and this affects the degree of modeling and its relevant practices. The answer set for “university degree” and “where/how modeling was learned”, which are used in this study can be accessible from [3]. 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). Seven possible choices were pre-provided in the survey for target sector, which were designed in discussions with embedded industry partners [20]. The most popular target sector of the products developed by the company employing the participants is “Consumer Electronics”, followed by “Defense & Aerospace” and “IT & Telecommunications”. 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. As a modeling language, all DOMN uses UML at most, but it is interesting that the second most frequently selected response is "Sketch/No formal modeling language", which is descriptive modeling. Another interesting result is that some respondents chose both UML and "Sketch/No formal modeling language", which show that these participants use modeling both formally and informally as in [18] depending on their purposes (in Q14 of our survey). Moreover, DOMN characteristics affect modeling languages choices. Our survey showed that DSL is mostly used in "Automotive & Transportation", where AUTOSAR (AUTomotive Open System ARchitecture) usage is ~15% although it was not in the pre-given answer set of the survey. Besides, 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 also interesting results [21].

Depending on PURPS, SDLC phases, where software modeling is used are affected. For example, if PURPS requires only descriptive modeling (e.g., communication and understanding) "preliminary/systems analysis" might be sufficient; however if there is a code generation, perhaps all SDLC phases use software modeling [3] (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. Our survey consisted two questions on both programming languages and diagram types used while modeling (Q15 and 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. MATLAB, C#, BPEL, Ada, Delphi and Smalltalk took some responses, which were in the pre-given answer set. Participants were also asked about the diagram types that they use. 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 in [18]. Therefore, these participants, 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). We also observed similar cases during interviews with embedded software professionals. By an in-depth look at the data, we found that most people use sequence diagrams informally to convey the communication among the entities in a given system. 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 programming language according to our 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), which is directly related to RIGOR. Furthermore, we also investigated that STAKH position (e.g., systems engineer) affects PL (e.g., MATLAB) for a specific PURPS (e.g., model simulation) on "*the variable formality*" slider during SDLC (e.g., in systems analysis and design).

We also investigated during interviews that software's closeness to hardware affects RIGOR and its relevant practices (e.g., modeling languages, diagram types, etc) via PL selection What meant by HW_CL is that firmware or digital signal processing (DSP) software is closer to hardware than User Interface (UI) or middleware software. This attribute indirectly affects RIGOR, but the best industrial real 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 our AR project [4] 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.

V. LIMITATIONS AND THREATS TO VALIDITY

In this section, we discuss the possible validity concerns in our study, which we minimized or mitigated.

Construct validities are concerned with the extent to which the objects of study truly represents theory behind the study [22]. We collected our data from different sources (i.e., different countries, industrial sectors, etc.) in order to avoid mono-operation bias. It is common for people to deflect their answers when they feel being evaluated. To mitigate this, we informed participants prior to the survey that our motive was to take a snapshot of the industry and that we will not collect any identifying information so that participants will remain anonymous. Moreover, during the semi-structured interviews to take feedbacks on the pre-design conceptual model, as recommended [14], we selected consulted professionals based on differences instead of trying to replicate similarities (see Table 1). Furthermore, we tried to reduce the threat related with definitions given in Section II.A as understood by that survey participants and also interviewees (i.e., consulted professionals) by making sure they distinguished these terminologies. In order to prevent any misunderstanding and potential threat in this terminology, we conducted a pilot study and we met with them to assess their common understanding of the terminologies regarding sketch, model-based and model-driven. However, the definition provided by Brambilla et al. [5] and our extended terminologies sadly may still leave room for subjectivity and we could not come up with better definitions. Thus, this issue stays as a potential threat, e.g., a given practitioner might in fact use model-based, even though s/he stated to use model-driven approach.

Internal validity reflects whether all causal relations are studied or if unknown factors affect the results [22]. Instrumentation was improved by using a pilot study for the survey. This reduces the likelihood for learning effects and, hence, maturation effects.

External validity is concerned with the extent to which the results can be generalized [22]. In order to decrease the effect of possible dominant participant number in a specific sector

due to authors' work experiences' network (i.e., defense & aerospace, consumer electronics, academia), the survey has been distributed to embedded professionals via various social network sites for different industrial sectors. Therefore, we have done our best to reach the subjects with a variety of different backgrounds representative for the embedded software industry. Our 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, followed by Asia and then the Americas (See [3]). Nevertheless, note that we used non-probabilistic sampling design and thus external validity is limited for the survey.

Conclusion validity of a study deals with whether correct conclusions are reached through rigorous and repeatable treatment [22]. 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.

VI. Conclusion and Future Works

With the help of this study, the attributes of a diagram development and usage in embedded software development was better understood in the best real industrial context. The conceptual model clarified the meaning of various modeling terms and investigated the correlations between these attributes, which affect each other. These investigations would provide practical benefits to embedded professionals (e.g., understanding their modeling characteristics and attributes), researchers (e.g., focusing on what industry uses the most and their challenges) and also educators (e.g., improving software modeling education for different stakeholder profiles). By this way, we believe that both embedded software professionals and also researchers benefited from our results and this will hopefully encourage more academia-industry collaborations.

As a future work, we plan to investigate all software modeling usage patterns (and also sub-patterns) in the embedded industry depending on these attributes, our survey results and also knowledge database. By constructing a model for a recommender system, in which we identify the stakeholders' software usage pattern, we plan to guide him/her to determine the optimum degree of modeling pattern for a cost-effective approach. In order to validate this throughout our findings, it is intended to arrange different companies in different target sector to understand how modeling are used within the company by conducting empirical case studies.

Acknowledgment

The authors would like to thank all embedded software professionals, who contributed to this study. Vahid Garousi was supported by the National Research Fund, Luxembourg FNR/P10/03

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

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

[7] (2016). *Career Award Talk - Bran Selic*. Available: https://www.youtube.com/watch?v=9qPbGksB3d4

[8] 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.

[9] 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.

[10] 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.

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

[12] S. Baltes and S. Diehl, "Sketches and diagrams in practice," presented at the Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, Hong Kong, China, 2014.

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

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

[15] 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.

[16] 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.

[17] A. Nugroho and M. R. Chaudron, "A survey into the rigor of UML use and its perceived impact on quality and productivity," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, 2008, pp. 90-99.

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

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

[20] 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.

[21] 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.

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