Dealing with Software Model Quality in Practice

Experience in a Research Project

Jose Luis de la Vara
Simula Research Laboratory
Lysaker, Norway
jdelavara@simula.no

Huáscar Espinoza
Tecnalia
Zamudio, Spain
huascar.espinoza@tecnalia.com

***Abstract*—Although past research has resulted in different means to deal with software model quality, creation of adequate software models remains challenging. Any modelling effort must be carefully analysed and planned before it starts, and definition or adoption of modelling guidelines is usually necessary. In addition, the amount of publications addressing model quality in practice is low, and the knowledge about others' experience regarding model quality is limited. This paper reports on our experience in dealing with software model quality in the context of a project between industry and academia. Such a project corresponds to a large-scale research project in which modelling has been used both as part of the necessary work for executing the project and for creating project results. We present how we have dealt with model quality in requirements modelling and in conceptual model specification, as well as a set of lessons learned. The insights provided can help both researchers and practitioners when having to deal with software model quality.**

Keywords-software model; quality; practice; requirements; conceptual model; lessons learned

I. INTRODUCTION

Software model quality has been the focus of much research during the last two decades. Starting from quality needs for conceptual modelling [31], many researchers have studied and provided specific means to deal with model quality for business process management (e.g., [4]), model-driven development (e.g., [33]), and requirements engineering (e.g., [15]), among other fields. Other authors have presented general, abstract frameworks (e.g., [36]).

Despite the results provided by the research community, the creation of adequate software models is usually difficult [10][23][27]. If a modelling effort is not carefully analysed and planned, problems such as lack of homogeneity or conceptual incorrectness in models can easily arise. These problems can be even more difficult to solve when different people and with different backgrounds (e.g., academia vs. industry) collaborate in model creation [8][16]. It is important to define or adopt means targeted at model quality.

Furthermore, the available knowledge about the state of the practice on software model quality and thus about others' experience in dealing with it is limited. Recent studies have shown that the ratio of papers that have addressed software model quality and presented insights from real projects is very low. For example, a systematic review on UML model quality [19] identified that only a 5% percentage of studies had applied case study research. Although this percentage has raised up to 35% in some study [35] and the problem does not only affect research on software model quality but also other software research fields (e.g., [12][25][37][38]), more publications presenting insights into real modelling efforts are necessary. Otherwise, it is difficult to meet the acknowledged needs for more evidence-based [17] or for more industry-driven and industry-oriented research [7], and thus to increase research impact and maturity.

This paper aims to extend the available knowledge about software model quality in practice by presenting our experience in a project between industry and academia. Such a project is called OPENCOSS [45], and corresponds to a large-scale European research project that aims to devise a common certification framework for the automotive, avionics, and railway domains. Modelling has been and is an important activity in OPENCOSS, both for analysing and refining project needs and for creating its results.

We present different situations that have happened in the project and in which we have had to address software model quality. The situations presented are in the scope of requirements modelling and of conceptual modelling specification. For each situation, we outline the problems faced and the decisions made. Having to face these situations has allowed us to learn several lessons.

The contribution of the paper is two-fold. First, the insights provided can help practitioners to gain awareness of issues (i.e., situations or problems that can lead to a lower model quality) that can arise in the modelling efforts in which they participate, and of possible training needs to better deal with such efforts. Second, researchers can benefit by increasing their knowledge about the current state of the practice on software model quality and about the degree of awareness of research results in industry. Academia can also use the results presented in this paper to define new research efforts targeted at improving software model quality in practice or at facilitating technology transfer. As an ultimate result, the paper contributes to the maturity and rigour [26] of research on software model quality.

The rest of the paper is organized as follows. Section II presents the background. Sections III and IV describe software model quality aspects related to requirements modelling and conceptual model specification, respectively. Section V summarises our insights by providing a set of lessons learned. Finally, Section VI presents our conclusions.

II. BACKGROUND

This section introduces the OPENCOSS project, outlines the software model quality framework used as a reference in this paper, and reviews related work.

A. OPENCOSS

OPENCOSS is a FP7 European project that aims to (1) devise a common certification framework that spans different vertical markets for automotive, avionics, and railway industries, and (2) establish an open-source safety certification infrastructure. The ultimate goal of the project is to bring about substantial reductions in recurring safety certification costs and at the same time reduce certification risks through the introduction of more systematic safety assurance practices. The project deals with: (1) creation of a common certification conceptual framework; (2) compositional certification; (3) evolutionary chain of evidence; (4) transparent certification process, and; (5) compliance-aware development process. The project consortium consists of 17 partners from nine countries, and only four partners are from academia. Most of the partners have been previously involved in software modelling activities.

OPENCOSS consists of both research and development activities. Software modelling has been used in such tasks both for specification of project needs (e.g., requirements [47]) and for specification of project results (e.g., a conceptual model for safety assurance and certification [46][48]).

B. Software Model Quality Framework

Different software model quality frameworks have been proposed in the literature for the last two decades. Among them, we use the framework proposed in [4] to indicate the overall quality aspects for which issues were found in OPENCOSS. Although the framework was proposed in the scope of modelling of business process, it is generic and can be adopted in other modelling activities. In addition, we believe that this framework is simple and thus easy to use and understand, but not simplistic.

The framework indicates six quality aspects of a model and its creation process at which concrete guidance can be targeted:

- Correctness, which is mainly related to the syntax and semantics of a software model, according to some reference framework (e.g., a metamodel or some guidelines);
- Relevance, which is mainly related to the selection of the relevant universe of discourse to be represented in a model;
- Economic efficiency, which is mainly related to the cost/benefit and feasibility of creating a software model;
- Clarity, which is mainly related to the need of a software model to be readable, understandable, usable, and thus useful;
- Comparability, which is mainly related to possible need for determining similarities and differences between software models, and;
- Systematic design, which is mainly related to the need of well-defined relationships between different models and to the possibility of making modelling auditable and repeatable.

Examples of other software model quality frameworks can be found in publications such as [22][31][32][36][39].

C. Related Work

As mentioned above, the current knowledge about software model quality in industry is limited and we believe that more publications reporting insights into the state of the practice are necessary. Nonetheless, there are works that have provided some insights.

Examples of software-related modelling techniques for which insights into model quality in real projects have been provided are the BPMN notation for business process modelling [15][61], the EKD framework for organizational modelling [56][57], the EPC notation for business process modelling [9], the i* goal-oriented notation [18], the UML language for software modelling [11][27], UML profiles for modelling of real-time systems [28][29], and use cases for requirements modelling [13]. Weaknesses identified in the techniques in these works that can lead to a lower model quality are concept redundancy (e.g., in BPMN), understandability difficulties (e.g., in i*), and concept overload (e.g., in UML).

Examples of other publications presenting specific lessons learned or insights from practitioners related to software model quality in the context of model-driven development are [3][5][6][15][23][51][59]. The lessons reported include the difficulty in applying a new modelling technique, the importance of determining the different concerns to model and of selecting suitable techniques for each concern, the problems that some people can have for thinking about abstract concepts, and practitioners' possible reluctance to adopt new guidance.

Some authors have presented more generic, general insights into software model quality in practice. The aspects reported include the need for training for practitioners before using a modelling approach [8], the importance of modelling experience [14], and the difficulties that can arise when clear guidance does not exist [24]. Overall issues in the collaboration between industry and academia in research projects such as the need for involving all the relevant stakeholders and the need for ask them for early feedback on the solutions have been presented, for instance, in [21][60]. These issues are related to the modelling effort in OPENCOSS.

Although they have not provided many insights into the state of the practice, other works have provided valuable guidance for software model quality for activities such as specification of requirements (e.g., [1][52][55]), identification of classes (e.g., [15][58]), creation of conceptual schemas (e.g., [40][49][50]), or modelling of business processes (e.g., [34] [53][54]).

Last but not least, extensive literature reviews related to software model quality can be found in works such as [19][32][35].

In summary, this paper contributes to widen the current knowledge about software model quality in practice by presenting specific situations and lessons learned. The paper also supports the insights provided by other authors and thus help to increase the amount of accumulated evidence for these insights. In this sense, the degree of detail presented can help others to analyse the extent to which the insights presented are related to their own situations [26].

III. REQUIREMENTS MODELLING

This section presents the main situations during requirements modelling in OPENCOSS in which we have found issues regarding software model quality. Such situations are: (1) definition of requirements levels; (2) determination of styles for each level; (3) determination of the granularity of the requirements at each level, and; (4) adoption of guidelines for business process modelling. Although these situation overlap to some extent, they correspond to different model quality issues.

Table I shows the quality aspects most related to the situations presented (correctness, CO; relevance, RE; economic efficiency, EE; clarity, CL; comparability, CM; systematic design, SD). Works that have provided similar insights are [6][13][15][20][34][54][61].

The results of the OPENCOSS task in which these situations were mainly encountered can be found in [47]. Such results correspond to a requirements specification for a safety-related evidence management software tool. It must be mentioned that when referring to software model quality in this section we do not only refer to graphical models, but also to text-based model specification.

The overall process adopted for the specification of requirements is shown in Figure 1. It is mainly based on the approaches proposed in [15][20].

TABLE I. QUALITY ASPECTS ADDRESSED IN REQUIREMENTS MODELLING

Situation	Software Model Quality Aspect					
	CO	RE	EE	CL	CM	SD
RM1		X		X	X	X
RM2	X			X	X	X
RM3	X				X	
RM4	X	X	X	X	X	X

RM1. Definition of Requirements Levels

One of the first issues for requirements specification was that the requirements elicited from different stakeholders corresponded to needs at different abstraction levels. For example, some stakeholders stated business goals such as "Facilitate evidence combination" (which refers to the need for better ways to link the pieces of safety evidence used for safety assurance and certification), whereas others stated very low-level requirements such as "I want to visualize artefacts relationships in traceability matrices".

The main consequence was that the initial requirements specifications contained requirements at different abstraction levels, as well as with different granularities. This situation resulted in problems regarding relevance, clarity, comparability, and systematic design in the specification of requirements.

As a solution, we decided to adopt and tailor the requirements abstraction model proposed in [20]. We defined the following four requirement levels:

- Product level, which corresponds to the goals whose achievement will be possible thanks to the development of a system;
- Feature level, which corresponds to features that a system must support in order to meet the goals of the product level;
- Function level, which corresponds to the functions and actions that a user should be able to do, and;
- Component level, which corresponds to the boundary between requirements and design.

We also defined requirement as proposed in [15]: requirements are activities, capabilities, or conditions, external to a system, that the system must support, possess, or meet, respectively, to fulfil stakeholders' needs. Design was considered to correspond to the internal characteristics of a system, which cannot be observed by an external user.

RM2. Determination of Styles for Each Requirement Level

Once the requirement levels had been defined, the next step was to determine the styles (i.e., formats) to be used at each level. Without this decision, correctness, clarity, comparability, and systematic design could be negatively affected in the specification of requirements.

For product level and feature level requirements, we used textual specifications (lists with descriptions), partially supported by the Maps approach for goal modelling [55] to facilitate goal and feature discovery and analysis (Figure 2). For function level requirements, we used uses cases [44]. For component level requirement, we used use case scenarios [2], textual specification of functional and non-functional requirements (in a template containing, among other parts, a textual description according to the structure proposed in [52]; Figure 3), and user interface mock-ups [30].

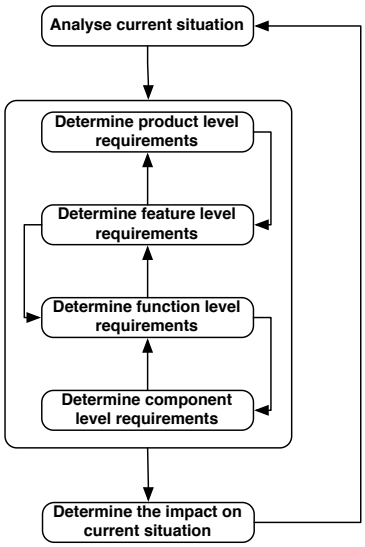

Figure 1. Overall process for the specification of requirements

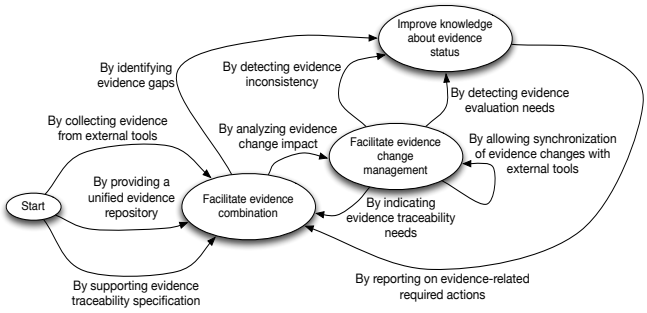

Figure 2. Example of Map diagram

RM3. Determination of the Granulartity of the Requirements at Each Level

Although requirement levels and styles for each level had been defined, we still needed to provide more guidance regarding the granularity that the requirements at each level should have. Otherwise, correctness and comparability in the specification of requirement could be hindered.

As general guidelines, and based on insights provided in other works such as [15][52][55], we proposed that:

- Product level requirements should represent goals corresponding to the business weaknesses, problems, or needs to solve by means of OPENCOSS results and that exist independently of the existence of these results (e.g., "Facilitate evidence combination");
- Feature level requirements should be characterised by (a) representing an abstraction of the functionality of a system, (b) corresponding to a system characteristic that is valuable for customer stakeholders, and (c) not being testable (i.e., a feature must be refined or broken down in order to verify that a system supports it);
- As a rule of thumb, function level requirements should be detailed and complete enough to kick-start system design, but not detailed and complete enough so that, for instance, two separate development teams implemented a same system (specification) and that the systems for both teams provided the same functionality and/or services, and;
- Component level requirements should be specified in such a detailed and precise way that would allow two developments to implement two systems with (almost) the same functionality and/or services, and at the same time it should be possible to assign these requirements to the system (architecture) components that will provide such functionality and/or service.

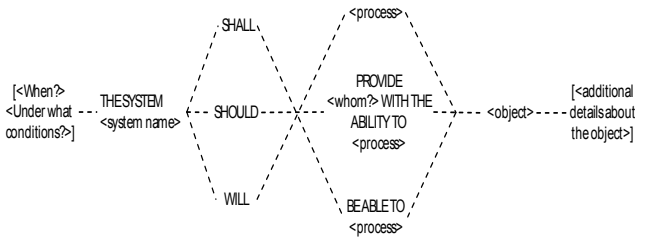

Figure 3. Text structure for requirements specification [52]

RM4. Adoption of Guidelines for Process Modelling

BPMN (Business Process Model and Notation; [41]) was the notation used for business process modelling as a part of the steps to take in the specification of requirements. This notation was not only used in the task whose results are reported in [47], but also in other OPENCOSS tasks for modelling both the current business process regarding activities such as safety assurance and certification and usage scenarios for OPENCOSS results. We observed several weaknesses in the business process models created, which hindered all the six quality aspects of the framework presented in Section II.B.

For example, we realised that different models created by different people resulted in models whose comparability was low. We also noticed the inadequate use of some BPMN concepts (e.g., the lanes of the pools), mainly because of insufficient knowledge about the notation. This affected model correctness. Problems with economic efficiency were detected as a result of the lack of a predefined approach for modelling and of domain knowledge.

Although we could not impose much guidance on how to create some models because we were not directly involved, suggesting and later adopting some guidelines based on those proposed in [15] significantly helped to mitigated the negative effects on the quality aspects.

IV. CONCEPTUAL MODEL SPECIFICATION

This section presents the main situations in conceptual model specification during OPENCOSS in which we have found issues regarding software model quality. Such situations are: (1) determination of the need for a new class; (2) class refactoring; (3) class merging; (4) class removal; (5) decision upon graphical modelling or textual speciation of constraints; (6) software model reuse; and; (7) differentiation between conceptual and implementation aspects.

Table II shows the quality aspects most related to the situations presented. Works that have provided similar insights are [13][15][28][29][32][49][50][40].

The results of the OPENCOSS task in which these situations were mainly encountered can be found in [46][48]. Such results correspond to a conceptual model for safety assurance and certification in the form of a class diagram. We have been involved in modelling in this task both in model creation and in checking models created by others.

In the first case, other OPENCOSS partners reviewed our models and we had to discuss with them the suitability of the models and possible modifications. In the second case, we had to check models created by OPENCOSS partners as well as other models that we wanted to use as a reference such as SACM (Structured Assurance Case Metamodel; [43]). This metamodel corresponds to an OMG specification that consists of an argumentation metamodel and of an evidence metamodel. Initially we had the intention to almost simply adopt these metamodels, but we later discovered several weaknesses that made us believe in the need for creating another conceptual model, although strongly based on this OMG specification. We use SACM in the rest of this section to show examples for some of the situations presented.

TABLE II. QUALITY ASPECTS ADDRESSED IN CONCEPTUAL MODEL SPECIFICATION

Situation	Software Model Quality Aspect					
	CO	RE	EE	CL	CM	SD
CM1	X	X				X
CM2			X	X		X
CM3	X	X	X	X		X
CM4		X	X	X	X	X
CM5				X	X	X
CM6			X			X
CM7	X	X	X	X	X	X

CM1. Determination of the Need for a Class

One of the issues that we had while specifying the conceptual model was modellers' lack of understanding about when modelling of only one class was insufficient and thus conceptually incorrect to represent safety assurance and certification concepts. Some modellers also had problems to decide if more than one class was necessary to represent some notions. Two examples of these issues are presented.

The first example corresponds to the concept of safety evidence. Safety evidence can be defined as the artefacts that contribute to gain confidence in the safe operation of a system [37]. Safety evidence also aims to show fulfilment of the requirements of a safety standard in the context of safety compliance and certification. In essence, artefacts are used as safety evidence for claims about system safety.

Despite the fact that artefacts and safety evidence are not the same, we did not find any model that had explicitly and clearly differentiated them when reviewing related work (e.g., [43]). For example, using an artefact as evidence for several safety claims implies the existence of emerging properties and relationships, different for each claim, such as the confidence in the evidence for supporting a claim. Therefore, and based on conceptual modelling principles (e.g., [40]), artefacts and pieces of evidence are different concepts and two classes are necessary in our conceptual model (Figure 4).

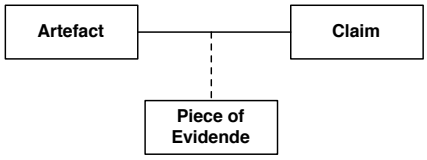

Figure 4. Basic model of safety evidence

The second example correspond to the need for modelling both the concrete relationships between the artefacts managed in a safety assurance project and the types of artefact relationships that a safety standard recommends or prescribes to be created (Figure 5). For example, a safety standard might require that traceability and thus relationships are maintained between requirements and code.

Someone might think that a class would be enough to represent both artefact relationships and artefact type relationships. However, we think that this has negative effects on model quality. For example, we consider that it would not be adequate that a single class corresponding to the merging of the characteristics (i.e., attributes and relationships) of artefact relationship and artefact type relationship. There would not exist any instance of that single class for which all the attributes and relationships would be instantiated. To our understanding, this is conceptually incorrect and thus negatively affects, for instance, semantic correctness, relevance, and clarity.

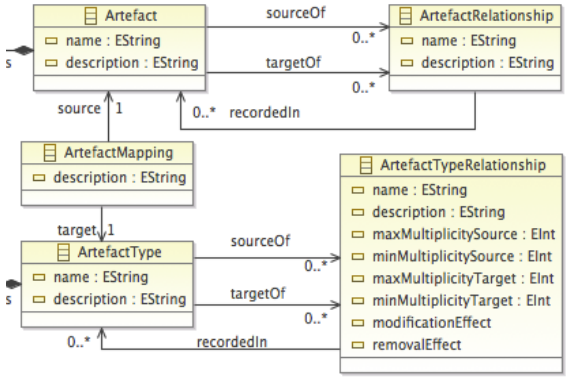

Figure 5. Basic model of artefact relationship reference artefact type relationship

CM2. Class Refactoring

One of the needs that we had while creating the conceptual model was to have to refactor some class. By refactoring we refer to the situation in which a large set of classes share a set of attributes or relationships and thus modelling of a class (usually abstract) will facilitate, for instance, maintenance of a model. We consider that this is a common need in the creation of practically any conceptual model or class diagram.

As an example, Figure 6 shows the Describable Element abstract class that we specified for the conceptual model for safety assurance and certification. After having modelled many classes with the name and description attributes (e.g., Activity and Participant), we realised that including Describable Element would contribute to economic efficiency, clarity, and systematic design.

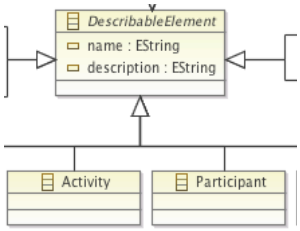

Figure 6. Describable element class

CM3. Class Merging

One of the weaknesses that we noticed in the models created by others was the introduction of unnecessary classes in a conceptual model or metamodel. In most of the cases, this phenomenon indicates the need for merging classes, because they correspond to the same concept. Figure 7 shows an example from SACM.

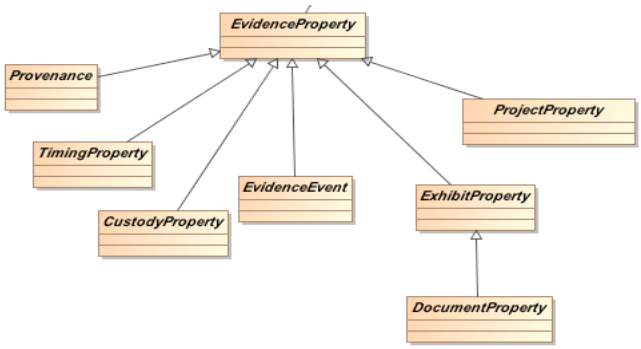

Figure 7. Example of candidate classes for merging [43]

In this case, we realised that some classes were redundant: the same phenomenon could be modelled by using and thus based on different SACM concepts. More concretely, we found overlaps in the Provenance, Custody Property, and Evidence Event classes of SACM. As a solution, we specified a single class in our conceptual model, called Assurance Asset Event, to merge the common aspects of these three SACM classes.

CM4. Class Merging

Related to the previous situation but not exactly the same, we observed cases in which classes had simply to be removed from a conceptual model. The most frequent reason that we noticed was that some information did not really corresponded to a concept and thus to a class to be modelled in a conceptual model, but to, for instance, a relationship between two classes.

In the example shown in Figure 8, we understand that the classes Created By, Approved By, Owned By, and Performed By could and should have been modelled as associations, not as classes. These classes do not have any specific attributes or relationships. That is, they do not have emerging properties, in contrast to the example explained in CM1 regarding artefacts an their use as safety evidence.

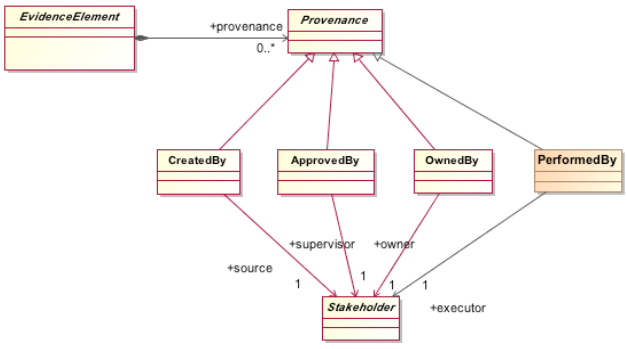

Figure 8. Example of candidate classes for removal [43]

CM5. Decision upon graphical modelling or textual specification of constraints

One aspect that we consider important and for which we are not sure if a clear, objective criterion can be defined is the decision upon graphically modelling some constraints (e.g., by means of more relationships) or textually specifying them (e.g., with OCL; Object Constraint Language; [42]).

Including constraints graphically in a model will make the constraint more explicit in the model but at the same time will increase its size. The understandability gained by modelling the constraint could not pay off because of the lost understandability as a result of making the model larger.

On the contrary, textually specifying constraints, outside of a model, might result in a model whose restrictions might become more difficult to perceive, despite the fact that keeping the model smaller might facilitate its understanding.

In essence, we have not been able to find a completely objective criterion to decide when to graphically model constraints or when to textually specify them. The number of elements of a model might be a possible indicator. Nonetheless, regardless of the criteria adopted for this issue in a modelling effort, we think that the alternative to adopt should be that regarded as the most helpful for model stakeholders to understand a model.

CM6. Software Model Reuse

A strategy that can definitely contribute to economic efficiency in the specification of a conceptual model is the reuse of models of fragments from models created by others. This can be especially important and useful when aiming to relate the results of a modelling effort with those from some group such as the OMG. The reuse of a model would make explicit its relationship with another model.

In OPENCOSS, we have tried to align the conceptual model created with the results from the OMG System Assurance Task Force. Apart from aiming to develop compatible solutions, OPENCOSS will also try to standardize its conceptual results. Such standardisation might be done through this task force.

Another specification that has been used as a reference is UML [44]. More concretely, and as an example, one characteristic that we plan to reuse from this specification is its mechanism for the dynamic specification of (data) types and values. This mechanism is shown in Figure 9.

Data and process aspects of the conceptual model have also taken into account models such as BPMN [41] or the conceptual frameworks proposed in publications such as [15] [40].

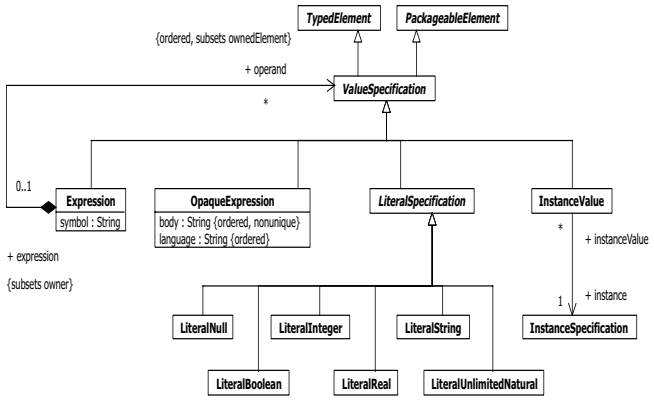

Figure 9. UML mechanism for types and values definition [44]

CM7. Differentiation between conceptual and implementation aspects

Last but not least, we have experienced some problems when specifying the conceptual model because of the lack of: (1) a common understanding of conceptual and of implementation needs, and; (2) a clear understanding of their difference and of the need for taking this differences into account when creating the conceptual model.

We use Figure 10 as an example of what we mean by the difference between conceptual and implementation aspects. The figure has been taken from SACM, and specifies for instance, that a document is an evidence item. We think that this figure does really represent real-world phenomena (see Figure 4), but a decision of how the implementation and thus the tool representation of such phenomena should be.

Without a clear and explicit differentiation of the conceptual and implementation aspects of a model, freedom for determining the most suitable tool support for a conceptual model in a given context is hindered, as well as quality aspects such as semantic correctness or relevance.

Inclusion of implementation decisions in a model, and when this is not its purpose, can also result in other problems such an unnecessary growth in size and difficulties to understand the application domain and the phenomena being represented in the model.

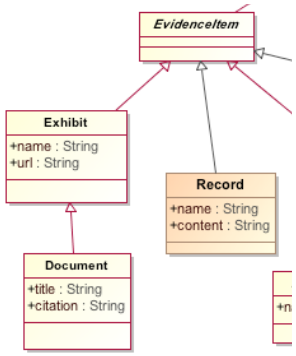

Figure 10. Example of implementation detail in a model [43]

V. LESSONS LEARNED

This section presents the set of main lessons learned after dealing with software model quality in the situations presented in Sections III and IV. Such a set consists of 12 lessons.

Table III shows the situations presented in the two previous sections from which the lessons have been learned.

LL1. The Top Priority of a Set of Modelling Guidelines Should Be to Help Stakeholders Understand a Model

Among all the quality aspects of a model and its creation process, we believe that probably the most important aspect is clarity, and more concretely understandability.

Although the rest of aspects are also important, we think that in a situation in which, for instance, a modelling decision can be positive for understandability but negative for another quality aspect, then understandability has a higher priority. If the model stakeholders cannot understand and thus validate a model, it will be useless.

It must be mentioned that we are assuming that any model will always be syntactically correct, and that trying to improve understandability in a model will not have a very negative effect in other quality aspects (e.g., relevance).

LL2. Feedback from Model Stakeholders is Necessary for Assessing Model Quality

Related to the previous lesson, our experience is that modellers should always ask model stakeholders for feedback. It is practically impossible that a modeller can assess on his own quality aspects such as relevance. Model stakeholders such as domain experts must indicate if they consider that the quality of a model is acceptable or some modification would improve a model.

LL3. Model Stakeholders Must Clearly and Explicitly Be Determined

A basic need for any modelling effort is to identify, and as soon as possible, the different stakeholders and types of stakeholders of a model. Such stakeholders can be, for instance, the people that participate in a modelling effort as modeller, the representative experts whose domain or phenomena are being modelled and thus who must validate the model, and the people who will correspond to the users of the model (e.g., the people responsible for creating the instances of the model).

Each stakeholder will have different needs and constraints, and not all the parts of or decision regarding a model and its creation process might be relevant to them. For example, we decided that the example shown in Figure 9 concerning UML type mechanism would not be relevant to domain experts and that thus this fragment should not be shown to them. It is a very specific way of modelling that such experts might not easily understand.

TABLE III. SITUATIONS FROM WHICH THE LESSONS HAVE BEEN LEARNED

Lesson	Situations
LL1	RM2, RM3, RM4, CM1, CM4, CM5, CM6, CM7
LL2	RM1, RM3, CM1, CM5, CM6, CM7
LL3	RM1, RM2, RM3, RM4, CM1, CM2, CM3, CM4, CM5, CM6, CM7
LL4	RM1, RM2, RM3, RM4, CM1, CM2, CM3, CM4, CM5, CM6, CM7
LL5	RM1, RM2, RM3, RM4, CM1, CM2, CM3, CM4, CM5, CM6, CM7
LL6	RM1, RM2, RM3, RM4, CM1, CM2, CM3, CM4, CM5, CM6, CM7
LL7	RM1, RM4, CM1, CM4, CM6, CM7
LL8	CM6
LL9	RM1, RM2, RM3, RM4, CM1, CM2, CM3, CM4, CM5, CM6, CM7
LL10	RM1, RM2, RM3, RM4, CM1, CM2, CM3, CM4, CM5, CM6, CM7
LL11	RM1, RM2, RM3, RM4, CM1, CM2, CM3, CM4, CM5, CM6, CM7
LL12	RM1, RM2, RM3, RM4, CM1, CM2, CM3, CM4, CM5, CM6, CM7

With regard to the users, their involvement is especially important when having to select the technology that will be used to support model instantiation. For example, and in the context of OPENCOSS, model users such as safety assurance managers might not be familiar with modelling technologies and tools, or with the creation of graphical models (e.g., an object diagram). Therefore, a tool such as a form-based interface, for which a model behind the interface exists for information management purposes, could be more usable for safety assurance managers.

LL4. It Is Important to Define and Try to Agree Before Starting a Modelling Effort on The Modelling Guidelines to Use

One of the issues that we have experienced in OPENCOSS is that we did not define detailed guidelines for many quality aspects of a model before starting model creations. As a result, we observed that, for instance, models created by different people were heterogeneous in aspects such granularity or use of terms. This situation can also happen when a model has a single creator.

Therefore, we recommend modellers to define or adopt modelling guidelines before starting any modelling effort. Based on our experience creating models both alone and with others, the use of pre-defined guidelines definitely leads to the creation of higher-quality models.

LL5. A Bad Decision Regarding Model Quality at Some Moment Can Have a Negative Impact Later

We have observed that if a sloppy decision is made during the creation of a model regarding its quality, then this decision can have undesired consequences at later modelling stages.

For example, we initially tried to directly adopt SACM in OPENCOSS despite the fact that we knew that it had some quality weaknesses. This turned to be a greater problem later when having to address OPENCOSS-specific modelling needs. Creating a model based on the original SACM specification became practically impossible. As a result, we had to re-execute some modelling activities, adapting some parts of SACM or directly discarding its use, depending on the case.

Therefore, modellers should carefully think about what to model and how when creating a model, taking into account different quality aspects and determining which aspect is the most important when having to choose between two, or if prioritisation of a quality aspect can have a negative effect on another.

LL6. Making All the People Involved in Model Creation Understand and Agree Upon Modelling Guidelines Might Not Be Possible

One of the greatest challenges that we have faced in OPENCOSS is to make all the people involved in model creation (for the tasks presented in this paper) to follow a same set of guidelines. Indeed, it turned not to be possible. Some OPENCOSS partners had a different opinion about the need or suitability of some guidelines, thus they did not agree upon their use.

LL7. It Can Be Very Difficult to Manage to Make All The People Involved in Model Creation Follow a Set of Guidelines Correctly

Related to the previous lesson, we have also experienced the situation in which some modellers did not follow all the guidelines defined, not because of disagreement, but because of, for instance, lack of attention or of use of a systematic approach for modelling. We had to remind these people to please use the guidelines in order to increase model quality, in some case several times.

LL8. The Purpose of a Model Should Be Clearly and Explicitly Specified

Another source of problems during model creation in OPENCOSS was the lack of a common understanding of the purpose of a specific model.

For example, modellers had to be reminded during requirements modelling that we did not have to address design aspects in the corresponding models. For the conceptual model, some partners had to be reminded several times that such a model should be simply a representation of some real-world phenomena and thus technology-independent. It should not contain implementation details.

These reminders resulted in an increased model quality.

LL9. Reuse of An Existing Model Might Not Always Be Possible or Recommendable

As described in the previous section, and as an example, we aimed to reuse SACM in the conceptual model, initially directly and later by modifying it. In both cases, such reuse led to many problems regarding model quality.

In essence, model reuse has to be carefully analysed before making a final decision.

LL10. It Is Necessary Sometimes to Make Trade-Offs in Model Quality to Ease Model Creation

Despite all the insights and recommendations that we have provided in the paper, we have to acknowledge that in some situations we realised that it would be better to make others not to follow some guidelines. The modelling effort would become too strict and difficult for them, and the effect of not following such guidelines would not be very negative.

LL11. Modelling Leaders Need to Be Aware of The Fact That Not Everyone Has The Same Knowledge and Experience in Model Specification

A very important aspect of which modelling leaders must be aware is that the same model quality cannot be expected from every modeller.

First, every modeller does not have the same knowledge of model quality, or of modelling in general. Second, modellers' lack of experience in modelling can also have negative effects in a modelling effort, despite that fact that their (theoretical) knowledge can be large.

LL12. It Is Necessary to Make Model Stakeholders Aware of The Importance of Model Quality

Last but not least, and to some extent overlapping with other lessons, we have realised the importance of making

everyone involved in a modelling effort (both modellers and other model stakeholders) aware of the importance of model quality, of modelling guidelines, and thus of the need for following modelling guidelines.

A good strategy to this end is to show examples of how the lack of care about model quality can result in models that are difficult to understand, use, or maintain.

VI. CONCLUSION

This paper has presented our experience in dealing with software model quality in the context of a large-scale European research project between industry and academia. The experience is based on two main activities: requirements modelling and conceptual model specification.

In requirements modelling, we had to address quality aspects regarding: (1) definition of requirements levels; (2) determination of styles for each level; (3) determination of the granularity of the requirements at each level, and; (4) adoption of guidelines for business process modelling. For conceptual model specification, we had to deal with: (1) determination of the need for a new class; (2) class refactoring; (3) class merging; (4) class removal; (5) decision upon graphical modelling or textual specifiation of constraints; (6) software model reuse, and; (7) differentiation between conceptual and implementation aspects.

For each situation above, we have shown the quality aspect at which the decisions made were targeted. The quality aspects considered are correctness, relevance, economic efficiency, clarity, comparability, and systematic design.

Addressing the aspects above allowed us to learn a set of 12 lessons. Such lessons are related to the definition and adoption of modelling guidelines, to the work of the people involved in model creation, and to the involvement of model stakeholders. In our opinion, the main meta-lessons learned are that selecting adequate modelling guidelines is very important but at the same time difficult, that all the model stakeholders must be made aware of the adoption of modelling guidelines in a project and ideally agree upon them, and that problems can arise in collaborative modelling efforts because of model stakeholders' different backgrounds and perspectives.

The insights presented can be very valuable and useful for both academia and industry. They can help researchers and practitioners to better know the needs and problems in similar modelling situations and to define strategies to address them. The insights can be especially relevant for projects in which industry and academia have to collaborate, given their different background, perspectives, and sometimes priorities.

As future work, we plan to continue compiling and reporting the issues found and the lessons learned regarding software model quality in OPENCOSS and in other projects in which we participate. This information can be very valuable for others. We would also like to empirically assess aspects of the software models created in OPENCOSS such as their understandability. Finally, it would be relevant to study how often and under what circumstances other people have faced the issues reported in the paper.

ACKNOWLEDGMENT

The research leading to this paper has received funding from the FP7 programme under the grant agreement n° 289011 (OPENCOSS) and from the Research Council of Norway under the project Certus-SFI. We would also like to thank the OPENCOSS partners that have participated in the modelling activities reported in the papers, and QUAMES 2013 reviewers for their valuable suggestions for improving the paper.

REFERENCES

[1] I.F. Alexander and R. Stevens,Writing Better Requirements, Pearson, 2002

[2] I.F. Alexander and N. Maiden (eds.). Scenarios, Stories, Use Cases, John Wiley and Sons, 2004

[3] B. Anda, et al., "Experiences from introducing UML-based development in a large safety-critical project", Empirical Software Engineering, vol. 11(4), pp. 555-581, 2006

[4] J. Becker, et al., "Guidelines of Business Process Modeling", in BPM 2000, pp. 39-40

[5] L. Bendix and P. Emanuelsson, "Collaborative work with Software Models - Industrial experience and requirements", in MBSE'09, pp. 60-68

[6] B. Berenbach, et al., Software and Systems Requirements Engineering: in Practice, McGraw-Hill, 2009.

[7] L.C. Briand, "Embracing the Engineering Side of Software Engineering", IEEE Software, vol. 29(4), pp. 96, 2012

[8] L.C. Briand, et al., "Research-Based Innovation: A Tale of Three Projects in Model-Driven Engineering", in MoDELS 2012, pp. 793-809

[9] E.C.S., Cardoso, et al., "Requirements Engineering Based on Business Process Models: A Case Study", in EDOCW 2009, pp. 320-327

[10] M.R.V. Chaudron, "Quality Assurance in Model-Based Software Development - Challenges and Opportunities", in SWQD 2012, pp. 1-9

[11] M.R.V. Chaudron, et al., "How effective is UML modeling? An empirical perspective on costs and benefits", Software and Sytems Modeling, vol. 11, pp. 571-580, 2012

[12] N. Condori, et al., "A systematic mapping study on empirical evaluation of requirements specifications techniques", in ESEM 2009, pp. 502-505

[13] K. Cox and K.T. Phalp, "Practical experience of eliciting classes from use case descriptions", Journal of Systems and Software, vol. 80(8), pp. 1286-1304, 2007

[14] I. Davies, et al., "How do practitioners use conceptual modeling in practice?", Data & Knowledge Engineering, vol. 58(3), pp. 358-38, 2007

[15] J.L. de la Vara, "Business process-based requirements specification and object-oriented conceptual modelling of information system", PhD Thesis, Universidad Politécnica de Valencia, 2011 (http://riunet.upv.es/handle/10251/11445)

[16] J.L. de la Vara, et al., "Towards Customer-Based Requirements Engineering Practices", in EmpiRE 2012, pp. 37-40

[17] T. Dybå, et al., "Evidence-Based Software Engineering for Practitioners", IEEE Software, vol. 22(1), pp. 58-65, 2005

[18] H. Estrada, et al., "An Empirical Evaluation if the i* Framework in a Model-Based Software Generation Environment", in CAiSE 2006, pp. 513-527

[19] M. Genero, et al., "A Systematic Literature Review on the Quality of UML Models", J. Database Management, vol. 22(3), 46-70, 2011

[20] T. Gorschek and C. Wohlin, “Requirements Abstraction Model”, Requirements Engineering, vol. 11(1), pp. 79-101, 2006

[21] T. Gorschek, et al., “A Model for Technology Transfer in Practice”, IEEE Software, vol. 23(6), pp. 88-95, 2006

[22] C. Houy, et al. “Understanding Understandability of Conceptual Models - What Are We Actually Talking about?”, in ER 2012, pp. 64-77

[23] J. Hutchinson, et al., “Empirical Assessment of MDE in Industry”, in ICSE’11, pp. 471-480

[24] M. Indulska, et al., “Business Process Modeling: Current Issues and Future Challenges”, in CAiSE 2009, pp. 501-514

[25] M. Ivarsson and T. Gorschek, “Technology transfer decision support in requirements engineering research: a systematic review of REj”, Requirements Engineering, vol. 14(3), pp. 155-175, 2009

[26] M. Ivarsson and T. Gorschek, “A method for evaluating rigor and industrial relevance of technology evaluations”, Empirical Software Engineering Journal, vol. 16(3), pp. 365-395, 2011

[27] C.F.J. Lange, et al., “In Practice: UML Software Architecture and Design Description”, IEEE Software, vol. 23(2), pp.40-46, 2006

[28] F. Lagarde, et al., “Improving uml profile design practices by leveraging conceptual domain models”, in ASE 2007, pp. 445-448

[29] F. Lagarde, et al., “Leveraging Patterns on Domain Models to Improve UML Profile Definition”, in FASE 2008, pp. 116-130

[30] S. Lauesen and M.B. Harning, “Virtual Windows: Linking User Tasks, Data Models, and Interface Design”, IEEE Software, vol. 20(2), pp. 58-65, 2003

[31] O.I. Lindland, et al., “Understanding Quality in Conceptual Modeling”, IEEE Software, vol. 11(2), 42-49, 1994

[32] B. Marín, et al., “A Quality Model for Conceptual Models of MDD Environments. Adv. Software Engineering 2010”, Advanced Software Engineering, 2010

[33] B. Marín, et al., “Using a Functional Size Measurement Procedure to Evaluate the Quality of Models in MDD Environments”, TOSEM (accepted paper), 2012

[34] J. Mendling, et al., “Seven process modelling guidelines (7PMG)”, Information and Software Technology, vol. 52(2), pp. 127-136, 2010

[35] P. Mohagheghi, et al., “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

[36] D.L. Moody, “The method evaluation model: a theoretical model for validating information systems design methods”, in ECIS 2003, pp. 1327-1336

[37] S. Nair, et al., “Classification, Structuring, and Assessment of Evidence For Safety: A Systematic Literature Review”, in ICST 2013

[38] S. Nair, et al., “A Review of Traceability Research at the Requirements Engineering Conference”, in RE’13 (accepted paper)

[39] H.J. Nelson, et al., “A conceptual modeling quality framework”, Software Quality Journal, vol. 20, pp.201-228, 2012

[40] A. Olivé, Conceptual Modeling of Information Systems, Springer, 2007

[41] OMG, Business Process Model and Notation (BPMN) Specification v.2.0, 2011 (http://bpmn.org)

[42] OMG, Object Constraint Language (OCL) Version 2.0, 2006 (www.omg.org/spec/OCL/2.0/)

[43] OMG, Structured Assurance Case Metamodel (SACM) - Version 1.0, 2013 (http://www.omg.org/spec/SACM/)

[44] OMG, Unified Modeling Language (UML) - Version 2.4.1, 2011 (http://uml.org)

[45] OPENCOSS project, http://opencoss-project.eu

[46] OPENCOSS project, “Deliverable D4.3 - Intermediate Common Certification Language: Conceptual Model”, 2012

[47] OPENCOSS project, “Deliverable D6.2 - Detailed requirements for evidence management of the OPENCOSS platform”, 2012

[48] OPENCOSS project, “Deliverable D4.4 - Common Certification Language: Conceptual Model”, 2013

[49] J. Parsons and Y. Wand, “Choosing Classes in Conceptual Modeling”, Communications of the ACM, vol. 40(6), pp. 63-69, 1997

[50] J. Parsons and Y. Wand, “Emancipating Instances from the Tyranny of Classes in Information Modeling. ACM Transactions on Database Systems, vol. 25(2), pp. 228-268, 2000

[51] S. Pilemalm, et al. “Practical Experiences of Model-Based Development: Case Studies from the Swedish Armed Forces”, Systems Engineering, vol. 15(4), pp. 407-421, 2012

[52] K. Pohl, Requirements Engineering, Springer, 2010

[53] J. Recker, “Opportunities and constraints: the current struggle with BPMN. Business Process Management Journal, vol. 16(1), pp. 181-201, 2010

[54] J. Recker, Evaluations of Process Modeling Grammars: Ontological Qualitative and Quantitative Analyses Using the Example of BPMN, Springer, 2011

[55] C. Rolland, “Capturing System Intentionality with Maps”, in Conceptual Modelling in Information Systems Engineering, pp. 141-158 Springer, 2007

[56] J. Stirna, et al., “Participative Enterprise Modeling: Experiences and Recommendations”, in CAiSE 2007, pp. 546-560

[57] J. Stirna and A. Persson, “Anti-patterns as a Means of Focusing on Critical Quality Aspects in Enterprise Modeling”, in EMMSAD 2009, pp. 497-418

[58] D. Svetinovic, et al., “Concept Identification in Object-Oriented Domain Analysis: Why Some Students Just Don't Get It”, in RE’05, pp. 189-198

[59] M. Torchiano, et al. “Relevance, benefits, and problems of software modelling and model driventechniques - a survey in the Italian industry”, Information and Software Technology (accepted paper), 2013

[60] C. Wohlin, et al., “The Success Factors Powering Industry-Academia Collaboration”, IEEE Software vol. 29(2), pp. 67-73, 2012

[61] M. zur Muehlen and D.T. Ho, “Service Process Innovation: A Case Study of BPMN in Practice”, in HICSS-41 2008