Information and Software Technology 51 (2009) 1646–1669

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Definitions and approaches to model quality in model-based software development – A review of literature

Parastoo Mohagheghi *, Vegard Dehlen, Tor Neple

SINTEF, P.O. Box 124, Blindern, N-0314 OSLO, Norway

ARTICLE INFO

Article history:
Available online 21 April 2009

Keywords:
Systematic review
Modelling
Model quality
Model-driven development
UML

ABSTRACT

More attention is paid to the quality of models along with the growing importance of modelling in software development. We performed a systematic review of studies discussing model quality published since 2000 to identify what model quality means and how it can be improved. From forty studies covered in the review, six model quality goals were identified; i.e., correctness, completeness, consistency, comprehensibility, confinement and changeability. We further present six practices proposed for developing high-quality models together with examples of empirical evidence. The contributions of the article are identifying and classifying definitions of model quality and identifying gaps for future research.

1. Introduction

For years, modelling has been advocated as an important part of software development in order to tackle complexity by providing abstractions and hiding technical details. Due to the wide application of modelling, numerous informal and formal approaches to modelling have been developed, such as Entity Relationship Diagrams (ERD) for modelling data, Specification and Description Language (SDL) for modelling telecommunication systems, formal modelling languages such as *Z* and *B*, and the Unified Modeling Language (UML) which is the most widely modelling language used by industry today.

Modelling was initially applied for communication between stakeholders and providing sketches (also called models or diagrams) of what a software system must do or its design. Nowadays, industry tends to use models more and more for tasks other than describing the system, for example simulation, generating test cases and parts or all of the source code. The growing attention on using models in software development has subsequently brought the quality of models as a research area in forefront. In late 2000, the MDA (Model-Driven Architecture)[1] initiative was launched by OMG (Object Management Group) to promote using models as the essential artefacts of software development. Followed by MDD (Model-Driven Development) or MDE[2] (Model-Driven Engineering), we face a new paradigm in software development where models are the primary software artefacts and transformations are the primary operations on models. In MDE, there is a lot to consider regarding the quality of models to ensure that right artefacts are generated. Finally, since defects can be earlier detected and corrected in models, improving the quality of models will ultimately reduce maintenance costs [8].

The QiM (Quality in Model-driven engineering)[3] project at SINTEF is concerned with the quality of artefacts and activities in model-based software development in general and MDE specifically. The term "model-based software development" in this article covers a spectrum of approaches where models are widely used in software development for more than visualizing the source code or providing informal sketches of design. Quality in model-based software development covers the quality of models, modelling languages, modelling tools, modelling processes and even transformations performed on models. One of the outcomes of the QiM project has been identification of constructs needed to develop quality models as described in [13]. With a "quality model" we mean a set of quality goals (also called quality attributes or quality characteristics in the literature) and their relations, accompanied by a set of practices or means to achieve the quality goals, evaluation methods for evaluating quality goals and links to related literature. The focus of this article is on identifying quality goals for models together with practices in model-based software development that can improve the quality of models, by performing a systematic review of literature on model quality. By identifying practices we take a preventive approach to software quality. Some tools and methods for assessing model quality and empirical evidence that are reported in the covered literature are presented in this article as well.

* Corresponding author. Tel.: +47 22067497; fax: +47 22067350.
E-mail addresses: parastoo.mohagheghi@sintef.no (P. Mohagheghi), vegard.dehlen@sintef.no (V. Dehlen), tor.neple@sintef.no (T. Neple).

[1] http://www.omg.org/mda.

[2] We use the term MDE in the remainder of this article to cover approaches where development is mainly carried out using models at different abstraction levels and possibly from different viewpoints, and where models provide a precise foundation for refinement as well as transformation so that other artefacts are generated from models; thus also covering MDA and MDD.

[3] http://quality-mde.org/.

0950-5849/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.infsof.2009.04.004

While model quality is covered in previous literature and various quality models are proposed; among them in [P15,P19,P22,P26,P38],[4] these quality models have some shortcomings as discussed in [12] and none of them provide their classification of model quality goals based on an analysis of previous work. Therefore we have performed a systematic review of literature discussing model quality to answer the following research questions:

- *RQ1. What quality goals are defined in literature for models in model-based software development?*
- *RQ2. What practices are proposed to achieve or improve the above quality goals?*
- *RQ3. What types of models and modelling approaches are covered in literature?*

Since UML is currently the most widely used modelling language, UML models are the subject of most work related to the quality of models. However, UML may be used in multiple ways; from providing informal sketches to full system specification end even as a programming language and extended for specific domains. Therefore literature on UML models covers different approaches to modelling, while approaches where models play a central role in software development are in focus here.

We emphasize that the focus of this article is on the quality of models representing or describing software systems and not the quality of system design or implementation; for example patterns and practices for object-oriented design and metrics on the design level. Unhelkar defines model quality as "the correctness and completeness of software models and their meanings" and separates it from code quality and architecture quality [P38]. We share the same view in this article. It is also clear that the quality of modelling languages, modelling tools and the expertise of people performing modelling will impact the quality of developed models. These issues are not covered in this article while some related work is discussed in [9].

The remainder of this article is organized as follows. The review process and the literature covered in this review are presented in Section 2 and validity threats are discussed. In Section 3, we discuss what models are and what roles they have in software development. Section 4 answers *RQ1* by identifying six model quality goals. These quality goals have been discussed previously in literature but never put together and defined in relation to one another. In Section 5, we discuss *RQ2* by presenting the practices proposed in literature in order to improve the quality of models, together with types of models and modelling approaches related to *RQ3* and the results of empirical studies whenever reported. Section 6 provides a summary of the results while Section 7 is discussion. Finally, the article is concluded in Section 8, answers to research questions are summarized and gaps for future research are discussed.

2. The review process and the included literature

In [7], Kitchenham et al. provide guidelines for performing systematic literature reviews (or in short systematic reviews) in software engineering. A systematic review is a means of evaluating and interpreting all available research relevant to a particular research question, topic area, or phenomenon of interest. A systematic review is therefore a type of "secondary study" that reviews "primary studies" relating to a specific research question. A primary study in [7] is defined as an empirical study investigating a specific research question. The process of performing a systematic review should be rigorous and auditable and include a review protocol. The above guidelines also discuss research question types in systematic reviews which are mostly related to evaluate the effect, cost or acceptance rate of a technology, thus reviewing empirical studies with quantitative data. However, systematic reviews can cover other research questions of interest to researchers as well. The goal of this review is to provide definitions and classifications while empirical evidence is also collected.

As discussed by Jørgensen and Shepperd, the process of defining an appropriate classification for any purpose is usually a bottom-up process by which the researchers read some studies, specify some categories based on the papers they have read and their own experience of the domain, then read more and refine the categories as necessary [5]. We therefore identified a set of publication channels where we had experienced that work on model quality would be published. We searched these publication channels for papers published since 2000, starting the search in March 2007 and ending it in October 2007.

The following publication channels were fully searched for papers discussing quality of models and the model-driven approach:

- The Software and Systems Modeling (SoSyM) journal since 2002 (the first issue).
- Proceedings of the UML conference from 2000 to 2004, succeeded by the MoDELS conference.
- Proceedings of The European Conference on MDA-Foundations and Applications (ECMDA-FA) started in 2005.
- Proceedings of the International Conference on Software Engineering (ICSE).
- Proceedings of OOPSLA, Conference on Object-oriented Programming Systems, Languages, and Applications.
- Proceedings of the Quality in Modelling (QiM) workshops at MoDELS conference started in 2006.

The following publication channels were searched with keywords; i.e., "quality + model", "quality + model driven" and "model driven + experience":

- Journal of Systems and Software.
- Information and Software Technology Journal.
- Software Quality Journal.
- Empirical Software Engineering Journal.
- IEEE Xplore.
- ACM digital library.

During the search, we identified candidate papers by evaluating their title and abstract. All candidates were registered in a file. We then drew a map over the subjects covered in the papers (categorization) and selected relevant papers to this review. The main criterion for including a paper in this review is that the paper provides definitions of model quality or discusses approaches to improve model quality. Some additional papers were found in the references of the detected papers and we also included a number of relevant books.

This review covers 40 "Primary studies (P)" (or in short "studies" in the remainder of the article[5]) related to model quality; including 3 books, 1 Ph.D. thesis, 7 papers published in journals, 11 papers published in the proceedings of conferences, 17 papers published in the proceedings of various workshops, and 1 paper published online. One of the studies is published in 1994 (it is included since it is an important work and is referred in several other studies), 2 in 2000, 3 in 2001, 2 in 2002, 1 in 2003, 11 in 2004, 4 in 2005, 9 in

[4] References beginning with "P" refer to primary studies covered in this review as given in Appendix I.

[5] Not all the studies in our review are empirical studies and therefore a primary study in this article refers to studies covered in this review.

2006, and 7 in 2007. It is clear that the subject of model quality has gained more attention since 2004.

A list of included studies is given in Appendix I numbered from P1 to P40, while Appendix II provides details on the number of studies detected in each publication channel and a short description of studies is given in Table 1. Table 1 also shows modelling language or modelling approach in each study with bold font. Examples are "UML", "MDE" and "quality model" (for studies discussing quality models). Not surprisingly, most of the studies are concerned with the quality of UML models. However, studies include a spectrum of approaches to modelling; from capturing system requirements to detailed design models, and to MDE including UML profiles and domain-specific modelling languages.

We have followed the steps of defining a review protocol and performing a systematic search as recommended in [7] while there are some deviations from the process of a systematic review; i.e., (a) we have not registered all the studies detected by using keywords but only the candidates for categorization; (b) we have searched a limited set of publication channels; and (c) we have only searched for recent publications. A more comprehensive review may be performed later by using this review's classifications as search keys or focusing on single aspects of model quality.

The main threat to the validity of the results of the review is *publication bias*; i.e., undetected studies when keywords are used and the uncovered publication channels. To improve the coverage, other search engines and additional keywords may be used which may detect new studies that can improve the results. However, we mean that the publication channels covered in this review are highly relevant for the subject of review. One may also search for studies published before 2000 or include more recent studies. There are conferences and workshops dedicated to model comprehension and model layout issues that are not covered in this review and could be covered if this aspect of model quality is in focus. A second threat is that we may have overlooked some relevant studies during identification and categorization. Following the steps of a systematic review as recommended in [7] would increase the validity of identification and the confidence in the results. After providing an initial classification of concepts, other results are easier to be added if they were undetected in the review process. Regarding the interest and knowledge of authors of the subject, all authors work in projects that include MDE but we are not aware of any biases when identifying and categorizing papers.

3. Models and model-based software development

Models are representations of a (software) system at an abstract level [17]. In this review we use the term "model" as a description or representation of a software system or its environment for a certain purpose, developed using a modelling language and thus conforming to a metamodel. A model may consist of several "diagrams" where each diagram type gives a different view on the described system. For example UML 2.0 has 13 diagram types such as use case diagram, class diagram etc. In MDE, It is common to model a system at different abstraction levels as well; thus having several models of the same system.

The role of models varies a lot in software development approaches applied in companies. Fowler has identified three modes of UML use[6]:

- UMLAsSketch: the emphasis of sketches is on selective communication rather than complete specification. These sketches should be acceptable by users.
- UMLAsBlueprint: blueprints are developed by a designer whose job is to build a detailed design for a programmer to code up and thus UMLAsBlueprint requires correctness and completeness of models to some degree.
- UMLAsProgrammingLanguage: semantics is added to UML models to make them executable. Here models should have the quality required for the purpose of execution.

Brown has also discussed the spectrum of modelling as presented in [18] and depicted in Fig. 1. The left hand side of the spectrum represents the traditional development without graphical modelling – the code is the main artefact. The right hand side of the spectrum represents the opposite of it, the code playing a secondary role and the development is done solely using models (e.g., utilizing executable modelling techniques). The model-centric approach is an ambitious goal of MDE as it still is based on code while the models are the main artefacts. Most (or all, if possible) of the code is generated from models; the developers, however, are given a possibility to add the code and synchronize it with models. The fact that the code can be altered after it is generated and it can be synchronized is close to the idea of roundtrip engineering, where the code and the model coexist and one is synchronized once the other is updated. Such a usage scenario can be seen as an advanced usage of models which is the extension of the idea of basic modelling. The basic modelling represents a situation when models are used as a documentation and as basic (usually architectural only) sketches of the software to be built. The models and the code coexist but the code is the main artefact which is used in the course of software development. In the code visualization scenario the code is the main artefact; models are generated automatically and are not used to develop software, but to provide means of understanding the code.

Staron writes that there is no sharp borderline between which of the usage scenarios (except for "code only") can be seen as a realization of MDE [18]. In this article we refer to model-based software development for approaches where models play a central role in software development, which covers the right hand side of Fig. 1. The covered literature covers approaches from basic modelling to full MDE, while most refer to modelling for more than providing sketches as in basic modelling.

When moving from the left hand to the right hand side of the spectrum, quality requirements (or goals) for models change and quality of models become more important. For example if models and code coexist, models must be correct and complete (to some degree) and also be easy to modify to keep them in sync with the code. Thus quality goals vary depending on the purpose of models.

MDE covers approaches where development is carried out using models; often at different abstraction levels and from multiple views. Although UML is the core language of the MDA initiative, MDE does not rely on UML. In fact it is often impossible to express the details of models required in a MDE approach in UML, which is the reason for extending UML and developing "UML profiles" or even Domain-Specific Modelling Languages (DSML). In this article we therefore cover research on the quality of UML models in addition to research with MDE focus, including UML profiles and DSMLs.

In MDE, models are subject of transformation to other models or text such as source code. The OMGs' MDA approach differs between CIM (Computational Independent Model), PIM (Platform independent Model) and PSM (Platform Specific Model) where more abstract models (CIM or PIM) can be transformed to PSM or directly to the implementation. For example, it might not be possible to completely specify an application as a PIM, but at least a large part of the application's static structure and interface design should be captured and then translated into PSM or code [19]. We

[6] See his blog http://martinfowler.com/bliki/.

Table 1
A short summary of primary studies.

Ref.	Short description
[P1]	This book describes a collection of standards, conventions and guidelines for creating effective **UML** diagrams. It includes some general diagramming guidelines and some guidelines for common UML elements and diagrams. It also presents the Agile Modelling (AM) approach
[P2]	The authors have originally developed MCC+, a plug-in for Poseidon for model consistency checking using Description Logics (DL). In order to achieve portability, the tool is upgraded into a tool for software product lines, MCC-SPL, that may be instantiated for several different **UML** modelling tools that admit the use of plug-ins
[P3]	The first part of the paper describes heuristics and processes for creating semantically correct **UML** analysis and design models. It provides a set of conventions for different UML diagrams. The second part of the paper briefly describes the internal research tool that was used to analyze Siemens models
[P4]	This paper describes a CMMI (Capability Maturity Model Integration) compliant approach to measurement and analysis during a model-driven requirements development process. It presents a set of metrics for **UML** requirement models that were used successfully on several Siemens projects, describing team dynamics, project size and staffing, how the metrics were captured and used, and lessons learned
[P5]	The authors define consistency problems in the context of component-based development with the **KobrA** method, and suggest a checking mechanism using environment modelling. The approach is automated using the SPIN model checker
[P6]	The authors evaluated quality differences between **UML** analysis and design models created with and without modelling conventions. Modelling conventions were provided either as a list or a list supported by a tool for detection of their violation. The conclusion is that UML modelling conventions (regarding layout and syntax) are unlikely to affect representational quality. Conventions are needed that clarify which types of information are relevant to particular future model usages; e.g., in implementation and testing
[P7]	The author proposes a set of criteria to improve the aesthetics of **UML class diagrams** based on HCI principles. The paper also includes a layout algorithm which respects all these features and an implementation of a graph drawing framework which is able to produce drawings according to these criteria
[P8]	The author proposes a set of aesthetic criteria for **UML class diagrams** and discusses the relation between these criteria, HCI and design aspects of object-oriented software
[P9]	This paper proposes a development methodology for distributed applications based on the principles and concepts of **MDA**. The paper identifies phases and activities of an MDA-based development trajectory, and defines the roles and products of each activity in accordance with the Software Process Engineering Metamodel (SPEM)
[P10]	The authors present a way to relate informal requirements, in form of **UML** use cases, to more formal specifications, written in OCL. The formal specification can improve the informal understanding of the system by exposing gaps and ambiguities in the informal specification
[P11]	This paper discusses ways to manage inconsistency (by analysis, monitoring and construction) and focuses on the second and third. The focus is on different views in **conceptual models**. The MERMAID modelling tool is used. The use of the Command pattern allows to implement the consistency by monitoring approach, and the use of complex commands and the Observer pattern allows for the realization of consistency by construction
[P12]	The paper considers the problem of consistency within and between artefacts. Based on UML, **a new langauge** is formally defined with less number of views. OCL is used to formulate both inter, and intra-consistency rules. These rules were partly implemented within OCL Evaluator tool
[P13]	This paper describes the ongoing **MDD** research efforts at Philips, introducing VAMPIRE – a light-weight model-driven approach to domain-specific software development
[P14]	The authors describe a process for constructing **UML** analysis model of an embedded system. The process uses goal models to capture requirements which also have constraints specified in formally-analyzable natural language properties. UML class diagrams and state diagrams are used to model structure and behaviour of the system and are transformed to formal specifications and formally analyzed for adherence to the behavioural properties captured in the goal model
[P15]	The authors discuss different modelling approaches applied for **conceptual modelling.** They further extend the **quality model** of [P22] with new quality types such as social quality and empirical quality. The identified means are also extended. The framework is used to evaluate the **Object Modeling Technique** (OMT) which is the ancestor of UML for conceptual modelling, and shortcomings and strengths of the language are discussed
[P16]	The paper elaborates on the role of stereotypes from the perspective of **UML**, and describes a controlled experiment aimed at evaluation of the role – in the context of model understanding. The results of the experiment support the claim that stereotypes with graphical icons for their representation play a significant role in comprehension of models and show the size of the improvement
[P17]	The authors present a **quality model** for managing **UML**-based software development. The quality model includes purposes, quality characteristics and some metrics. They further discuss experiences in applying the quality model to several industrial case studies. Finally a tool is presented that visualizes the quality model
[P18]	This work reports on a controlled experiment to explore the effect of modelling conventions on **UML** diagrams on defect density and modelling effort. The results indicate that decreased defect density is attainable at the cost of increased effort when using modelling conventions, and moreover, that this trade-off is increased if tool-support is provided
[P19]	The Ph.D. thesis gives a thorough discussion of **UML** usage, its diagrams and quality issues related to different purposes of modelling. Different quality models are also presented before presenting own contribution which is described also in [P17]. Different experiments performed by Lange and others as in [P21,P20] are discussed with more details to cover the aspect of quality defects
[P20]	The authors report a multiple case study, in which 16 industrial **UML** models are explored for defects. The analysis results were discussed with the model developers to gain deeper insights into the quality problems. The level of occurrence for several defect types is discussed. Additionally, the paper explores the influence of factors such as model size, time pressure, and developers' skill on model quality
[P21]	In this paper the authors identify tasks of model-centric software engineering and information that is required to fulfil these tasks. They propose views to visualize the information needed to support fulfilling the tasks, and metrics required to evaluate the status of tasks. The focus is on **UML** models
[P22]	The authors examine attempts to define quality as it relates to **conceptual models** and propose their own **quality model** with three types of model quality: syntactic, semantic and pragmatic quality. The authors also propose means to improve the quality of models, for example performing inspections and simulating models
[P23]	The authors use a formal Object-Oriented specification Language (OOL) to formalize and combine **UML diagrams**. It is thus possible to transform the consistency of UML models to the well-formedness of OOL specifications
[P24]	The authors introduce a general framework for formalizing a subset of **UML diagrams** in terms of different formal languages based on a homomorphic mapping between metamodels describing UML and the formal language. The resulting specifications enable execution and analysis through model checking
[P25]	This white paper presents a set of guidelines for developing high-quality architecture models in **UML**
[P26]	This paper describes a theoretically-based set of best practices for ensuring that each step of a modelling process is performed correctly, followed by a proof of concept experiment demonstrating the utility of the method for producing a representation that closely reflects the real world. The paper introduces a **quality model** which differs between perceptual quality, descriptive quality, semantic, syntactic, pragmatic and inferential quality

(*continued on next page*)

Table 1 (*continued*)

Ref.	Short description
[P27]	This paper compares a model developed in The **Object-Process Methodology** (OPM), which has only a single diagram, to a model developed in **Object Modeling Technique** (OMT) with several diagrams. OPM shows to be more effective in terms of a better system specification and some differences in comprehension are observed
[P28]	Five **UML** graphical notations are compared in this paper: for each, two semantically equivalent, yet syntactically different, variations were chosen from published texts. The purpose is to evaluate which notations are easier to understand for humans
[P29]	This paper reports on experiments assessing the effect of individual aesthetics in the application domain of **UML diagrams**. Subjects' preferences for one diagram over another were collected as quantitative data. Their stated reasons for their choice were collected as qualitative data. The work is similar to [P28]
[P30]	**Object-Process Methodology's (OPM)** single-diagram approach is compared with **UML**s multi-diagram approach regarding the level of comprehension and the quality of the constructed Web application models. The results suggest that OPM is better than UML in modelling the dynamics aspect of the Web applications. In specifying structure and distribution aspects, there were no significant differences. The quality of the OPM models was superior to that of the corresponding UML models
[P31]	Business process models (diagrams) and object life cycles can provide two different views on behaviour of the same system, requiring that these diagrams are consistent with each other. The paper proposes an approach to establish their consistency. Object state diagrams are used to generate life cycles for each object type used in the process. The diagrams are developed in **UML**
[P32]	The authors propose the integrated technique related to metrics in a **MDD** context. The following topics are covered; (1) the application of a meta modelling technique to specify formally model-specific metrics, (2) the definition of metrics dealing with semantic aspects of models (semantic metrics) using domain ontologies, and (3) the specification technique for the metrics of model transformations based on graph rewriting systems. They use class diagram plus OCL to represent meta models with metrics definitions
[P33]	After presenting the context of modelling and the rationales behind the decision to use **DSM**, the paper describes the approach to the problems of promotion, process integration, usability and sustainable deployment of domain-specific solutions. The conclusion is that most challenges to deploy DSMs are not technical but human by nature
[P34]	In this paper, the key factors for the efficient accomplishment of the **MDA** are discussed by means of an industrial case study. The factors identified are grouped into two categories – associated with usage and development of an MDA-based framework
[P35]	The paper describes a set of controlled experiments which were aimed at evaluating the role of stereotypes in improving comprehension of **UML** models. The results of the experiments show that stereotypes play a significant role in the comprehension of models and the improvement achieved both by students and industry professionals
[P36]	In model re-factoring behaviour should be preserved as specified by the model. This paper defines some behaviour preserving properties inherent to model re-factorings. A **UML profile** is defined with stereotypes extending the UML dependency relationship and its specializations to express the preservation properties between connected UML artefacts. Using a logic approach dependency relations are formalized and checked
[P37]	This is an experience report emphasizing the synergy resulting from combining **MDE** and SPL (Software Product Line) technologies. The paper also discusses some challenges with using domain-specific solutions
[P38]	This book defines three aspects of a **quality model**: syntax, semantics and aesthetics. The book further defines three types of models: Models of Problem Spaces (MOPS), Models of Solution Space (MOSC) and Models of Background Space (MOBS). For each type of model, the author discusses necessary diagrams and quality check. Finally, For each type of **UML** diagrams, checklists are provided in these three dimensions
[P39]	The authors report their experiences with a **DSL** for reinsurance business and financial accounting. One experience is that that soft constraints (i.e., warnings instead of errors) are indispensable and should become an intrinsic part of DSLs
[P40]	The authors discuss that model correctness is fundamentally important in MDE. They discuss techniques to prove model correctness which cover model checking and automated theorem proving. Model analysis is applying semantic rules that look for situations that are semantically incorrect or suspicious. Different types of errors that might be detected or not detected by model analysis are discussed. Model analysis techniques are applied to industry design models developed in **UML** and **DSL**

emphasize especially that future usage of models such as generating test cases, code (partially or totally) or simulation drive identifying quality goals. In the next section we provide a definition of model quality goals that are important in model-based software development approaches depending on the purpose of models.

4. A classification of model quality goals (*RQ1*)

In [12] we have presented previous classifications of model quality goals[7] such as:

- The Lindland et al.'s quality framework has conceptual models in mind (models of specification statements such as requirement models) and classifies model quality into *syntax* (adhering to language rules), *semantic* (correct meaning and relations) and *pragmatic* quality (comprehensibility by the intended users) [P22].
- Additional model quality goals are added by Krogstie and Sølvberg to the Lindland et al. framework; such as *organizational* quality (whether a model fulfils the goals of modelling and that all the goals of modelling are addressed through the model) and *technical pragmatic* quality defined as being interpretable by tools [P15].
- Unhelkar classifies model quality goals into *syntax* (with focus on correctness), *semantics* or meaning (with focus on completeness, consistency and representing the domain), and *aesthetics* (with focus on symmetry and consistency in order to improve the look and to help understanding) [P38]. His work is on UML models.
- Nelson and Monarchi provide an overview of quality models and discusses that modelling is a transformation from real world to an implementation in multiple steps [P26]. In each step one

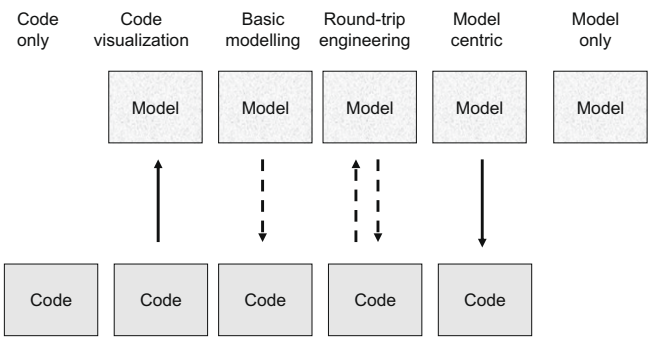


Fig. 1. Model-driven software development adoption spectrum, from [18].

[7] The terms quality goals, characteristics, attributes, properties etc. are all used in literature and in different quality models. We use the term "quality goal" here to cover them all.

should assure that the content is persevered. The author defines *perceptual*, *descriptive*, *semantic*, *syntactic*, *pragmatic*, and *inferential* quality as quality types where some definitions match previous work in [P22] and [P15].

Although the above definitions are useful, we do not see them often used in literature on model quality. Besides, the boundary between syntax and semantics is sometimes blurred [4] and these terms are used inconsistently in literature. Therefore it is useful to have a classification that is close to the concepts used in the literature related to modelling.

Based on the results of this review and our earlier work on developing a quality model for MDE as presented in [9,10,12], we have identified six classes of model quality goals that are introduced in the remainder of this section. We also present related work that point to the origin of the definitions.

C1-Correctness. Correctness is defined as:

(a) Including right elements and correct relations between them, and including correct statements about the domain;
(b) Not violating rules and conventions; for example adhering to language syntax (well-formedness or syntactic correctness according to [P22]), style rules, naming guidelines or other rules or conventions.

Several quality models define *syntactic correctness* relative to the modelling language and *semantic correctness* relative to the domain and our understanding of it. Nelson and Monarchi write that syntactic quality is determined by comparing the representation to the language while the meaning of the elements should be preserved, called as semantic quality [P26].

Including right elements and relations is related to our understanding of the domain and is called *semantic validity* in the framework of Lindland et al. [P22]. Berenbach calls a UML model *semantically correct* if it includes correct relations and is complaint with good practices and corporate standards [P3]. Unhelkar rather talks of errors and mentions that CASE tools keep language syntax errors to a minimum while the semantic aspect requires that the diagrams faithfully represent the underlying reality [P38].

C2-Completeness. Completeness is defined as having all the necessary information that is relevant [P22] and being detailed enough according to the purpose of modelling.

Berenbach defines requirement and analysis models as complete when they specify all the black-box behaviour of the modelled entity [P3,P4]. Complete models can then be used to define test cases and create project tasks. Others do not define completeness bur rather discuss incompleteness as missing elements in models [P27]. Mitchell writes that one should discover key problem domain concepts and make sure that these are modelled in software; from system analysis models, through design and into code [P25]. Nelson and Monarchi write that the perception transformation should be complete and should not include anything that is not in the real world [P26].

C3-Consistency. Consistency is defined as no contradictions in the model. It covers consistency between views or diagrams that belong to the same level of abstraction or development phase (*horizontal consistency*), and between models or diagrams that represent the same aspect, but at different levels of abstraction or in different development phases (*vertical consistency*). It also covers semantic consistency between models; i.e., the same element does not have multiple meanings in different diagrams or models.

Lange writes that the multi-diagram approach of UML entails the risk for contradictions between diagrams, so called inconsistencies. Besides, consistency defects can occur not only within a model between different diagrams, but also between models at different abstraction levels [P19]. Consistency between diagrams or models of a system is important for correct interpretation of them. Ambler emphasizes consistency when modelling a system in the sense that common elements have common names to avoid confusing readers [P1]. Berenbach also discusses consistency of definitions across all diagrams [P3]. Several other studies discuss inconsistency problems and how to avoid them as discussed later in Section 5.

C4-Comprehensibility. Comprehensibility is defined as being understandable by the intended users; either human users or tools. Lindland calls this *pragmatic quality* [P22], which is the term used by Krogstie and Sølvberg [P15] and Nelson and Monarchi [P26] as well.

For humans, several aspects impact comprehensibility such as aesthetics of diagrams [P29,P38], organization of a model [P1,P3], model simplicity or complexity [P1], using concepts familiar for the users or selected from the domain ontology [P1,P25], and finally using the correct type of diagram for the intended audience. For example, Berenbach writes that UML is flexible regarding the choice of diagrams for defining a process. Sequence, collaboration, activity and state diagrams can all be used. However, he recommends using sequence diagrams which is his experience are easiest to read for non-technical reviewers [P3]. Ambler has a set of guidelines to improve readability of diagrams which are both related to aesthetics (such as avoiding crossing lines) and organization of elements on a diagram [P1]. Unhelkar writes that once the syntax and the semantics are correct, we need to consider the aesthetics of the model [P38]. Aesthetics is simply style or "look and feel" which has a bearing on the models readability or comprehensibility. Focus of most literature is on comprehensibility by humans.

For tools, having a precise or formal syntax and formal semantics helps analysis and generation. Krogstie and Sølveberg define "technical pragmatic quality" as to what extent tools can be constructed to understand the models [P15].

C5-Confinement. Confinement is defined as being in agreement with the purpose of modelling and the type of system; such as including relevant diagrams and being at the right abstraction level. A model is a description from which detail has been removed intentionally. A confined model does not have unnecessary information and is not more complex or detailed than necessary.

Ambler writes that models should be kept simple and one must avoid details not necessary for the purpose of modelling [P1], where the motivation is to improve readability. Other motivations may be to avoid keeping several models in sync and reducing the effort spent on modelling. Mitchell emphasizes that adding details to a high-level diagram is different from adding design details [P25]. The first makes an imprecise model more precise while the second adds unnecessary information.

Developing the right model for the type of system or purpose also depend on selecting the right modelling language. However, our focus here is on model quality.

C6-Changeability. Changeability is defined as supporting changes or improvements so that models can be changed or evolved rapidly and continuously.

Changeability is not mentioned in previous work as a separate quality goal although the need for updating models is obvious and mentioned in several studies. It is required since both the domain and our understanding of it or requirements of the system evolve with time. Mitchell writes that a system must resemble the problem so that it may be changed when the problem domain changes [P25]. Jonkers et al. write that the modelling discipline should be close to the problem domain which enables simpler maintenance and evolution of models [P13]. Berenbach recommends avoiding early packaging since it implies partitioning and may result in frequent reorganizations [P3].

Changeability should be supported by modelling languages and modelling tools as well. For example Krogstie and Sølvberg have

identified modifiability of language as a practice that helps achieving validity and completeness [P15].

6C goals. We call the above quality goals collectively for the ***6C* (model quality) goals**. Fig. 2 shows the *6C* goals and their relations to other elements involved in modelling. Compared with the Lindland et al.'s framework in [P22], we have added modelling rules and organization (defining the goals of modelling) to the framework.

Fig. 3 is another view of the *6C* goals that shows when in the development process they are important. The figure is inspired by the idea of viewing modelling as a set of transformations [P26]. A model is a representation of a system and should be complete relative to the system it wants to represent and according to the modelling goals defined by the organization. It should also contain correct relations between elements and correct meanings. All these properties depend on the perception of the modeller of the domain and the purpose of modelling. The developed models are required to be correct relative to the language and modelling rules or conventions, and be comprehensible for interpretation by humans or by tools for the purpose of generation, simulation or analysis. Of course precise definition of quality goals depends on the context and the purpose of modelling, especially whether models are used for implementation, testing and maintenance of systems.

Finally, we mean that other quality goals discussed in literature can be satisfied if the *6C* goals are in place. For example a model that is correct, complete and consistent does not allow multiple interpretations and all of the above goals are important in order to support reusability of models. The *6C* goals are identified based on the analysis of literature covered in this review and the list may therefore be modified or extended if new requirements are detected.

In the next section we present the proposed approaches for improving the quality of models which are referred to as "practices" in our quality model, together with the reported empirical evidence.

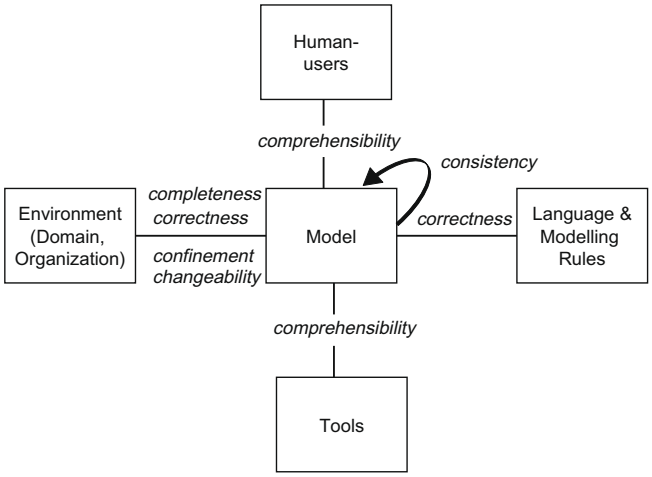


Fig. 2. The *6C* model quality goals.

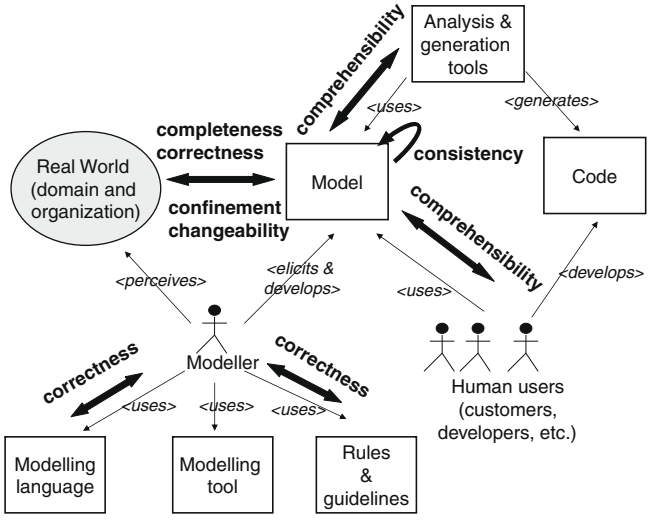


Fig. 3. Model-based software development with transformation of real world to running software.

5. Practices proposed to improve the quality of models (*RQ2* and *RQ3*)

In this section we discuss the practices proposed in the studies to be applied during modelling to improve the quality of models. Most practices are concerned with *error prevention*, while some also facilitate *error detection*. We have identified six classes of practices which are presented throughout this section together with examples of empirical work. We also discuss their impact on the *6C* goals introduced in Section 4. The six practices are divided in two groups:

(a) The first group is related to "modelling process" and covers having a model-based development process, using modelling conventions and the single-diagram approach[8];
(b) The second group is related to "formal approaches and automation" and covers formal models, domain-specific solutions and generating models from models.

Table 2 shows an overview of the studies covered in this review ordered after the proposed practice. The impact of practices on quality goals, the name of tool used or developed and the type of empirical evidence is also given. There are four studies that cover quality models in general and refer to most or all of the quality goals; i.e., [P15,P22,P26,P38]. In Table 2 there is a column called "Demo or Empirical approach" where the values are:

- "–" for studies that are pure discussion. This covers three studies.
- "Example" which shows that the proposed practice is applied on an example application to demonstrate its usefulness. An example is not empirical evidence. 16 studies include such examples.
- "Student experiment" indicates that a controlled experiment is performed with students as subjects; described in 9 studies.
- "Industrial case" refers to describing experience from applying a practice in industry. Industrial cases detected in this review do not have the level of formality required of a "case study" as defined in [20] such as a precise definition of research questions, context, data collection methods and results. We found description or reference to industrial cases in 14 studies.

The sum is 42 since two studies cover both student experiments and industrial cases; i.e., [P35,P19]. Although the focus of this systematic review has not been on collecting evidence and appraising approaches, the data provide examples for evaluation and future

[8] Selecting a single-diagram approach depends on the modelling language. However, we chose to group it under modelling process since selecting suitable languages and diagrams is often a step of modelling processes as discussed later.

Table 2
A classification of primary studies regarding practices and their impact on quality goals.

Ref.	Type of model	Practice and impact on quality goals	Demo or empirical approach	Tool
Model-based software development				
[P22,P15]	Conceptual models	Activities such as filtering and inspection should be performed to improve **comprehensibility by humans**	Example	–
[P26]	Conceptual models	A good modelling process will provide mechanisms for error prevention, detection and correction. Descriptive quality (regarding **correctness** and **completeness**) can be prevented by interviewing key informants. Semantic errors (**correctness**) are best prevented through a cycle of data reduction, data display, and verification. Human inspection (sometimes by multiple users) is also proposed to improve the models regarding **correctness**, **completeness** and **comprehensibility**	Industrial case	–
[P1]	UML diagrams	The Agile Modelling (AM) approach is proposed which uses agile practices to increase agility (related to **changeability**)	Example	–
[P17,P19]	UML diagrams/metrics	Quality goals are identified based on the purpose of models, which is related to **confinement**; i.e., focus on purpose. Metrics are defined for different goals	Industrial case	QualityView for collecting metrics and visualization
[P21]	MDE approach/metrics	Metrics are collected from models and visualized in order to support **comprehension** of diagrams, design quality evaluation and **completeness**	Industrial case	MetricView Evolution for collecting metrics and visualization
[P32]	MDE approach/metrics	Metrics are proposed to be formally defined and collected from various models/ diagrams	Example	–
[P40]	MDE approach/SDL and UML models	Model analysis techniques should be applied to improve **correctness** of models. Rules can be defined in the model analysis tools	Industrial case	–
[P9]	MDE approach	A traceability strategy should be defined to improve **consistency**. An activity is defined for selecting a modelling language that is expressive enough for the domain and the needs of modelling, which is related to **confinement**	Example	–
[P34]	MDE approach	MDE process is focused on transformations: based on the requirements, the team elicits requirements specific for the PIM and PSM transformations which in turn form the basis for UML profiles definition. See later under UML profiles for benefits	Industrial case	–
Modelling conventions				
[P28,P29]	UML class and collaboration diagrams	Conventions are regarding layout of UML diagrams. Some notations perform better than others on **comprehensibility by humans**	Student experiment	–
[P7,P8]	UML class diagrams	Layout of diagrams is discussed related to **comprehensibility by humans**. Some problems with comprehensibility may indicate poor design	Example	Graph drawing framework SugiBib
[P35]	UML architectural models	Models should be checked for having enough details (**completeness**) and cross-checked for **consistency**. Correct organization of models helps **comprehensibility by humans**. Identifying domain concepts and keeping analysis models free of implementation helps **confinement**	–	–
[P5]	UML models in KobrA	Conventions are proposed to improve inter-component and intra-component **consistency** in the KobrA approach	Example	SPIN model checker
[P3]	UML analysis and design models	Examples of conventions are given in Table III. Except changeability, **all other quality goals** are mentioned. The tool checks models for some of the conventions	Example	DesignAdvisor
[P1]	UML diagrams	Conventions are proposed that cover **all quality goals.** Examples are given in Table III	Example	–
[P38]	UML diagrams	Conventions are proposed that cover **all quality goals.** Examples are given in Table 3	Example	–
[P4]	UML use case and other requirement diagrams	Conventions are proposed to improve **completeness** of requirement models. These can be defined as rules and be programmatically verified	Industry case	DesignAdvisor
[P18]	UML diagrams	Conventions cover **correctness** and **comprehensibility** of models. Reduced defect density is attainable at the cost of increased effort and this trade-off is increased if tool-support is provided	Student experiment	–
[P19,P20]	UML diagrams	An analysis of several industrial models for defects showed that lack of conventions creates defects regarding **consistency** (for example in class names) and **correctness** (for example defaults regarding visibility and naming are kept where they should be changed)	Industrial case	SDMetrics for collecting metrics
[P6]	UML analysis and design models	The impact of modelling conventions on representational quality is not verified. Conventions should be defined from the purpose of models; i.e., related to **confinement**	Student experiment	–
Single-diagram approach				
[P27]	OPM vs. UML models	The single-diagram approach of OPM is compared to multiple diagrams of OMT. **Completeness** and in some aspects **comprehensibility by humans** are improved. Completeness is easier to verify as well	Student experiment	–
[P30]	OPM vs. UML models	OPM models (single diagram) were more **correct** than UML models for Web applications. Also OPM model was easier to **comprehend** than UML diagrams since information is spread over several diagrams. **Consistency** of the model after an update is also better with a single-diagram approach	Student experiment	–

(*continued on next page*)

Table 2 (*continued*)

Ref.	Type of model	Practice and impact on quality goals	Demo or empirical approach	Tool
Formal models				
[P15,P22]	Conceptual models	Formal syntax helps syntactic **correctness**. Formal semantics helps semantic **correctness** (validity) and **completeness**. Exacutability of models helps **comprehensibility by humans**	Example	–
[P24]	UML class and behaviour diagrams	Formalizing UML diagrams allows model checking and simulation, which helps **comprehensibility by humans** and **consistency**	Industrial case	SPINs model checking and simulation, Hydra
[P23]	UML class and sequence diagrams, state machines	Formalizing UML diagrams allows checking them for **consistency**	Example	–
[P11]	Conceptual models	Defining a formal syntax and semantics for different views and implementing patterns such as the observer pattern improves **consistency** between diagrams. A new language is defined that implements the proposal	–	–
[P2]	UML diagrams	**Consistency** checks for UML diagrams are defined in Description Logic (DL)	Example	MCC+ (Model Consistency Checker) and MCC-SPL
[P14]	UML diagrams and a goal model	Behaviour **consistency** can be established between the goal model and UML behavioural diagrams by transforming UML models to formal ones and analyzing them for adherence to goals. SPIDER tool is used to translate natural language constraints into formal specifications	Example	Hydra for UML formalization, SPIDER
[P10]	UML use cases	By adding OCL constraints to requirement models, ambiguity is removed from informal specifications which improves **correctness**. One may also check that the conditions are not contradictory which improves **consistency**. On the other hand, formal specifications are more difficult to read for humans (**negative impact on comprehensibility**)	Example	–
[P12]	Context model, use case model and analysis model (based on UML)	Adding OCL constraints allows us to check models for both intra and inter-**consistency** by using tools	Example	OCL Evaluator tool
UML Profiles and DSMLs				
[P16,P35]	UML profiles	Stereotyped models are better understood (related to **comprehensibility by humans**)	Student experiments, industrial case	–
[P34]	UML profiles	Effective usage of UML in industrial applications strives for its customization for the specific purpose – thus the definition of a domain-specific modelling language. Being suitable for a domain is classified as **confinement**. Constraints can be added to models to define restrictions on the usage of base modelling elements and thus improve **correctness** of the models	Industrial case	–
[P36]	UML profiles	Adding OCL constraints to stereotypes in UML profiles restricts the wrong usage of elements and thus improves **correctness**. UML profiles allow defining **correct** transformation	Example	RACOoN (Racer for Consistency)
[P33,P37,P39]	DSML	DSMLs bridge communication gap between engineers and domain or business experts; related to **comprehensibility**. Raising the abstraction level also improves **comprehensibility**. A DSML includes only elements and diagrams necessary for the domain and thus improves **confinement**. However, developing a DSML and subsequently editors and code generators require high programming expertise and updating it is costly	Industrial case	–
[P13]	DSML	Modelling discipline should be close to the problem domain which enables simpler maintenance and evolution of models; related to **confinement**, and improved **comprehensibility** by domain experts	Industrial case	–
Generating models/diagrams from other models/diagrams				
[P31]	UML diagrams	Generating one diagram (here object life cycles) from another (object state diagrams) improves **completeness** and **consistency** between models/diagrams	Example	–
[P11]	Conceptual models	The observer pattern is implemented which generates necessary elements in diagrams when changes in one is observed. It improves **consistency** between diagrams	–	–

work. We introduce the six practices in Sections 5.1–5.6, and provide a summary in Section 5.7.

5.1. Model-based development process

Several authors discuss the advantages of developing a model-based process or adapting the existing ones to MDE in order to improve the quality of models and the generated assets. A good modelling process will have mechanisms for preventing, detecting, and correcting errors at each step from observation to elicitation to analysis to final representation [P26]. Berenbach writes that in his experience from industry (1) lack of process contributed to a large number of errors since modellers do not have process that guides where to start and conventions that provide uniformity; (2) lack of quality assurance (for example reviews) led to a staggering number of errors; and (3) design models that originated from analysis had fewer errors thon those originated as designs [P3]. Nelson and Monarchi event write that instead of evaluating the

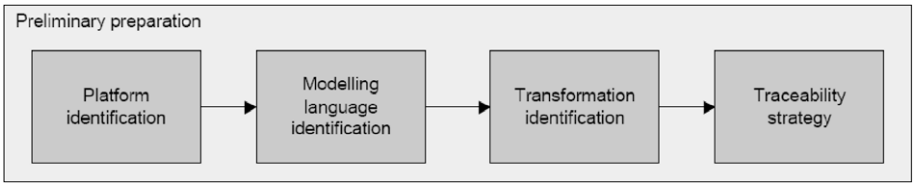


Fig. 4. Preliminary preparation phase in an MDE-based project as defined in [P9].

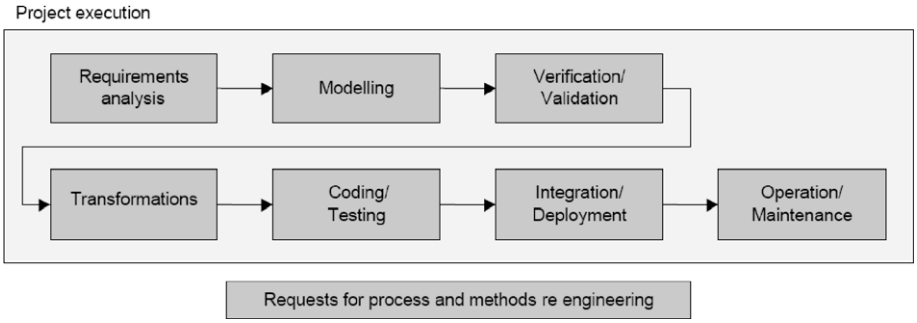


Fig. 5. Project execution activities as defined in [P9].

quality of the final representation, the representation process itself can be evaluated [P26].

5.1.1. Examples and the impact on model quality

There have been several attempts to adapt existing development processes to MDE by adding artefacts, activities and roles. Firstly, Gavras et al. have identified phases and activities of a model-driven based development trajectory process that include phases related to MDE artefacts and tasks [P9]. These phases precede the core development phase and include choosing technologies and developing needed artefacts (such as metamodels, modelling language, transformations, validation rules and tools), as depicted in Fig. 4. These artefacts are then deployed for use in the project execution phase, as depicted in Fig. 5. The project execution phase has an activity related to validation of models. The paper presents an outline of an example study that has been carried out where the roles and products of each activity are defined in accordance with the Software Process Engineering Metamodel (SPEM), which is an OMG specification to describe software development processes.[9]

In the same spirit as Gavras et al., Staron et al. have identified activities in a MDE based development process [P34]. The defined process is iterative and involves normal tasks of software development such as requirements elicitation, development, test and deployment; but with additional MDE activities. For example based on the requirements, the team elicits requirements specific for the PIM and PSM transformations which in turn form the basis for UML profiles definition. There are seven phases in the MDE development process as shown in Fig. 6. The process is developed within an industrial case study.

The above two papers touch on an obviously important topic as having control of the quality of the MDE tooling is necessary to have control of the quality of systems being developed using the tools. Staron et al. emphasize defining transformations prior to profiles: the model-driven software development process is focused on transformations – thus it is transformations that are identified during the requirements elicitation phase. As profiles are used as a means of making the transformations automated by providing storage for additional information, they are not considered at this phase. Thus their process add identifying transformations and defining profiles to the Gavras et al. process while it lacks defining a traceability strategy.

There are also approaches that include metrics and tool support for evaluating the progress and the quality of development artefacts. Firstly, Lange et al. have identified tasks of model-centric software engineering (meaning that UML models are used for more than describing the system) and the information that is required to fulfil these tasks [P21]. Examples of tasks are program understanding or completeness evaluation. A task may consist of a number of questions about UML models. They propose to visualize the information required to support fulfilling the tasks and have developed a tool to support their approach on UML models; i.e., the MetricViewEvolution tool. The authors write that feedback from industrial case studies and a light-weight user experiment has been positive. The advantage of this tool over other tools that collect metrics from UML models is relating these metrics to performing tasks in a model-centric development process.

Secondly, Berenbach proposes collecting metrics to evaluate completeness of activities, in his approach related to requirement modelling [P4]. The author proposes a model-centric requirement

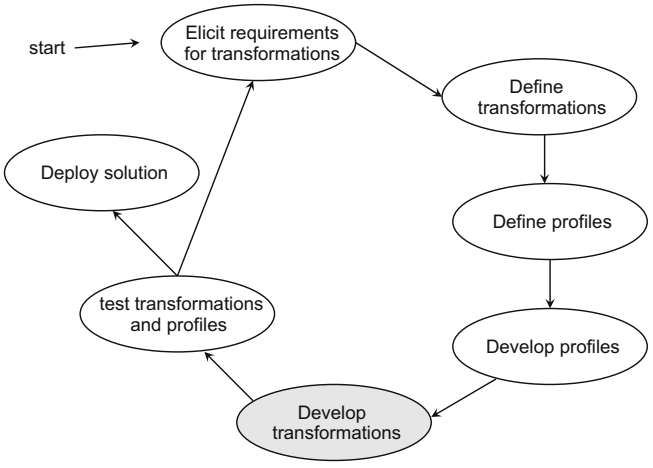


Fig. 6. MDA-based framework development process defined by Staron et al. [P34].

[9] http://www.omg.org/technology/documents/formal/spem.htm.

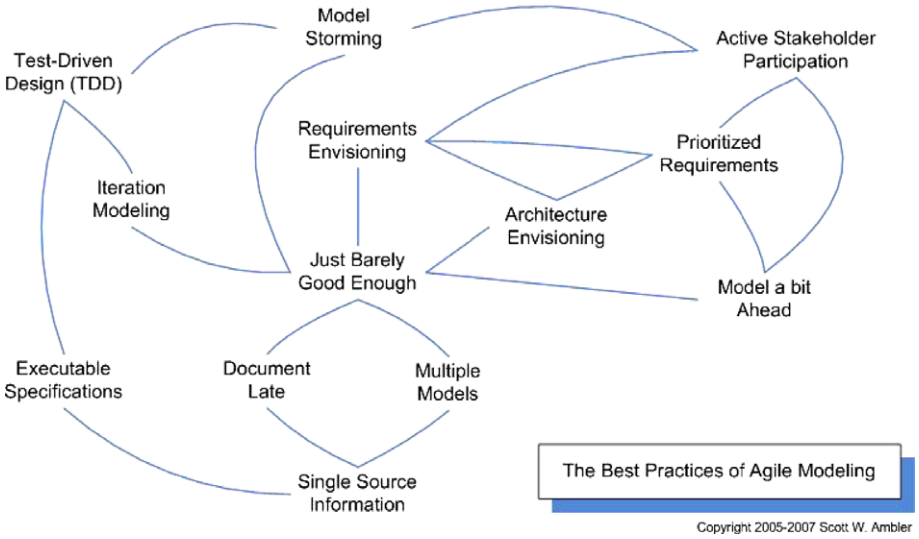


Fig. 7. The best practices of Agile Modelling (AM) from [P1].

process which covers steps for developing, reviewing and analyzing requirement models, including a set of project and quality metrics. Berenbach writes that quality requirements are defined in a way that they could be programmatically verified. For example completeness of requirement models are defined as:

- Every leaf use case (a use case with no included, extending or inheriting use cases) is concrete, with a special stereotype. The stereotypes of functional and non-functional requirements were used, and the stereotypes were given special icons for ease of visual inspection.
- The leaf use cases become the high level requirements that are subject to a formal review. The model is also subject to a review, primarily to check for completeness and defects.

Quality metrics are defined towards requirement models and measured frequently, giving the possibility to give insight into the project progress. Examples are "the number of objects on diagrams" or "concrete use cases not defined". It seems clear that having a high quality requirements process is important, and having ways of saying "how good" the requirements are is thus a clear advantage.

In traditional development, it is quite common to perform code review, so in the model-based development process, model reviews should be natural as proposed in [P15,P22,P26]. Other quality assurance techniques such as collecting metrics by tools and analyzing them should be part of a modelling process as well. Although we do not intend to cover these techniques, we have referred to them whenever they are mentioned in the studies.

Some papers have also provided guidelines or conventions regarding the modelling process. Examples are:

- Berenbach recommends the following [P3]: the early modelling effort should cover the entire breadth of the domain. Identify "out of scope" use cases as early as possible. To support this, all the actors in the model should be shown on the "context" diagram. Discover business objects and their services through use case definition with sequence diagrams. Elicit requirements and processes by starting at interfaces and modelling inward.
- Mitchell warns against assuming that all domain modelling must happen in the first phase, and all design in later phases [P25]. I.e., an iterative approach is recommended; as also by Ambler [P1].

Ambler has developed the Agile Modelling (AM)[10] approach which is a collection of practices to be used in modelling and documentation of software systems, as depicted in Fig. 7. Models in AM are sketches developed for communication and not generation and the focus of Ambler is on UML models. The interesting aspect is, however, the focus of AM on change and agility in modelling which is often ignored by other modelling processes. Many of the practices apply for a model-based software development approach, such as:

- Apply the right artefact(s). Each artefact has its own specific applications (related to confinement).
- Create simple content; the implication is that you should not add additional aspects to your models unless they are justifiable (related to confinement).
- Single source information; you should also model a concept once and once only, storing the information in the best place possible (to avoid inconsistency).

The promoters of developing process thus assume that the definition of artefacts and procedures up-front will give a concise set of artefacts. Mandating their use will reduce the use of "non-normative" techniques, and should make it easier to control quality. It is also a requirement that processes should include quality assurance activities, evaluating project progress and evaluating the quality of the developed artefacts. Finally, it is also an advantage if artefacts in one phase may be used to develop or generate artefacts in the next phase in order to achieve completeness and consistency where one example is discussed in Section 5.6. For example use cases may be used to identify objects and their states.

5.1.2. Examples of empirical evidence

Some of the proposed processes are either developed by industry or are evaluated in industrial cases:

- Staron et al. performed a case in Volvo Information Technology and concluded that the project was successful [P34].
- Lange et al. evaluated the feedback from industry regarding their approach and tool as positive [P21].

[10] http://www.agilemodeling.com/.

Table 3
Examples of modelling conventions. In classifying the conventions, we have used the motivation given by the author. Berenbach has marked the conventions verifiable by tools by "auto" [P3]. Notice that selecting conventions depend on the purpose of modelling.

Correctness
- The model should have a single entry point (auto) [P3]
- Extending use case relationships can only exist between concrete use cases (auto) [P3]
- Abstract use cases must be realized with included or inheriting concrete use cases (auto) [P3]
- A concrete use case cannot include an abstract use case (auto) [P3]
- An interface should only communicate with a concrete use case (auto) [P3]
- Every actor in the model should communicate with use cases through interfaces (auto) [P3]
- Every state with no outgoing transitions must model a terminal state in the world being modelled [P25]

Completeness
- Every diagram should have an associated description and status to detect incomplete work (auto) [P3]
- Every artefact in a UML model should be visible on a diagram [P3]
- Every artefact in a UML model should be visible on a diagram (in order to avoid having artefacts that are not used in a model) (auto) [P3]
- Coherent low-level processes should be defined with state or activity diagrams (auto) [P3]
- Indicate unknowns with a question mark (in order to complete them later) [P1]
- When you add detail to a model, it can be detail that adds precision to an imprecise model. Adding detail does not necessarily mean you've moved to a lower level of abstraction [P25]

Consistency
- The definition of a use case must be consistent across all diagrams defining the use case (auto) [P3]
- Every class in the design model should trace back to a use case in the analysis model (auto) [P3]
- An interface class should derive from an analysis boundary class (auto) [P3]
- Every expression in a specification of behaviour can be cross-checked to concepts in a model of structure [P25]
- The terms in every expression in the pre-condition and the post-condition must be defined in the type model [P25]

Comprehensibility (including aesthetics)
- Class name should be a singular noun [P3]
- The use case should be named from the point of view of customer [P3]
- Use sequence rather than collaboration diagrams to define one thread/path for a process [P3]
- Avoid crossing lines, diagonal or curved lines, close lines and sized symbols [P1]
- Reorganize large diagrams into several smaller ones. Prefer single-page diagrams [P1]
- Prefer well-known notation over esoteric notation [P1]
- Apply common domain terminology in names (especially for requirements and analysis diagrams) and apply language naming conventions on design diagrams [P1]
- Do not model every dependency or implied relationships (completeness here is in conflict with comprehensibility) [P1]. The focus is on high-level models
- The depth of inheritance trees partially limits the physical dimensions of the drawing. Therefore these trees should be clearly visible and spatially separated from each other [P7,P8]
- Class with a high number of outgoing relations indicate that classes depend too much on other classes [P7,P8]. (Note: from model quality to design quality)

Confinement
- Avoid realization relationships and artefacts in the analysis model. Analysis model should be free of realization or implementation [P3]
- A use case described as a sequence of interactions necessarily includes some design decisions. The same use case described using a pre-condition and a post-condition can be free of such design decisions [P25]
- Apply the right artefact; for example avoid modelling data and user interface components in UML since UML does not yet address modelling these [P1]
- Indicate visibility of operations only in the design model since this is a design issue [P1]
- Check that the diagrams are free from unnecessary objects [P38]
- Check that no attributes or operations are shown in object diagrams [P38]

Changeability
- Avoid the early use of packages since they will require reorganizing [P3]

- Berenbach reports that the proposed metrics for requirement modelling were applied in three different projects at Siemens and write that "as the team members gained experience with the measurement tools and increased ability with the UML, their productivity and confidence rose dramatically" [P4]. The improvement is, however, not quantified.

The feedbacks are positive but we so far lack detailed case studies on the impact of development processes on the quality of models.

5.2. Modelling conventions

Ambler defines conventions as "guidelines for creating effective (UML) diagrams; based on proven principles that will lead to diagrams that are easier to understand and work with" [P1]. The term "convention" in the remainder of this article refers to modelling rules and styles as well. Lange et al. have identified four categories of conventions proposed for UML models [P18]:

- *Design conventions*; e.g., high cohesion and low coupling.
- *Syntax conventions*; Ambler presents a collection of 308 conventions for the style of UML. His conventions aim at understandability and consistency, and address naming issues to avoid inconsistency, layout issues and the simplicity of design [P1].
- *Diagram conventions*; deal with issues related to the visual representation of UML models in diagrams, such as those proposed in [P29].
- *Application-domain specific conventions*; such as using stereotypes in UML profiles.

Design conventions are related to the quality of design and are not covered in this review. A good overview of such conventions and rules is provided by the VIDE project (VIsualize all moDel-drivEn programming) [16]. VIDE has performed an extensive review of literature to identify quality defects in MDE and have identified several classes of conventions such as design principles, anti-guidelines, aging symptoms and modelling styles. Most of the conventions are not specific to models but to software design in general. Application-specific conventions are covered in Section 5.5 related to domain-specific approaches. Examples of other conventions are presented in the remainder of this section and in Table 3. Several studies mention that modelling conventions should be integrated in a modelling process and be supported by tools to be best effective.

5.2.1. Examples and the impact on model quality

We found several examples of conventions proposed for UML models. Some would be relevant independent of the modelling language. Examples are:

- In the book "The elements of UML 2.0 style", Ambler describes a collection of conventions for creating effective UML diagrams [P1]. The book contains some general guidelines applicable to any type of UML diagrams, guidelines for common UML modelling elements such as notes and stereotypes, and guidelines for specific UML diagrams.
- In the book "Verification and validation for quality of UML 2.0 models", Unhelkar provides guidelines for modelling and checklists to check UML diagrams for syntax, semantic and aesthetic issues [P38].
- Berenbach presents a set of heuristics for creating "complete" UML analysis and design models, which may further be analyzed by tools [P3]. The proposed conventions cover model organization in general, use case definitions, analysis models, business object models, and design models, and affect most of the quality goals. Several large models at Siemens were evaluated using the DesignAdvisor tool which checks models for some of the proposed conventions, while other conventions may be checked by inspections.
- The KobrA method makes components the focus of the entire software development process by adopting a product-line strategy for their creation, maintenance, and deployment [1]. In KobrA, each component is described by a suite of UML diagrams as if it were an independent system in its own right. Choi and Bunse write that the use of UML diagrams and the recursive nature of KobrA introduce two consistency issues in general; static consistency and behavioural consistency. Static consistency mainly refers to structural and naming/type consistency among specifications describing different aspects of a component. A total of 59 rules are proposed to improve static consistency which can be ensured either by manual inspection or by mechanical syntactic name checking. However, KobrA does not include behaviour consistency rules. Choi and Bunse have therefore proposed some rules and propose to check them using the model checker SPIN [P5].
- In a white paper by Mitchell, the author discusses some principles for creating high-quality models based on their experience [P25]. The focus is mainly on cross-checking between architectural models and other models (use case descriptions, object models, and behaviour models such as state diagrams). The conventions cover several quality goals such as comprehensibility (regarding organization of a model, e.g., divide a system into technical domains and subject domains) and completeness (e.g., make sure these concepts are modelled in software, from systems analysis models, through design, and into code). Some of the conventions are related to improving the process of modelling and were discussed in Section 5.1.
- Eichelberger criticises UML for the lack of aesthetic principles, and UML tools for ignoring aesthetic principles [P7,P8]. The author proposes a set of aesthetic criteria for UML class diagrams and discusses the relation between these criteria, HCI (Human Computer Interaction) and design aspects of object-oriented software. Some aesthetic problems indicate design problems, for example "a class with a high number of outgoing relations indicate that the class depends on too many other classes" or "many classes at the borders of a package and few classes in the centre imply coupling problems".

Many of the proposed conventions may be enforced by tools, but the problem today is that most modelling tools enforce syntactical and aesthetic constraints very weakly and semantic constraints are not enforced at all [P17]. For error detection, one may use tools such as DesignAdvisor or perform inspections.

5.2.2. Examples of empirical evidence

Lange et al. have performed a controlled experiment with students as subjects on the effect of (UML) modelling conventions on modelling effort and defect density [P18]. The conventions they have included in the experiment yield correctness (e.g., an abstract class should not be a leaf), comprehensibility (e.g., the number of sequence diagrams per use case should indicate how well the functionality of a use case is documented or described) and design issues (e.g., low coupling). The experiment results indicate that decreased defect density is attainable at the cost of increased effort when using modelling conventions, and moreover, that this trade-off is increased if tool-support is provided. Since we do not have the full list of conventions and the detailed results of experiments, it is difficult to say which types of errors are prevented.

In another student experiment described in [P6], DuBois et al. investigated whether using conventions has any effect on *representational quality* of a model; defined as the clarity, completeness and

validity of the information the model is meant to represent. They did not observe any difference in representational quality and concluded that conventions should rather focus on identifying and consistently representing those types of information required for the model's future usage, e.g., in implementation and testing.

Purchase et al. performed several student experiments on UML diagrams and the students' preference of one diagram over another, checking layout issues such as the number of crosses, the number of bends, use of colours or fonts and the width of diagrams, specially for class and collaboration diagrams [P28]. Also five UML experts performed the experiment. The comprehension task was that of matching a given textual specification against a set of diagrams, indicating whether each diagram correctly matches the specification or not. The set of diagrams included both correct and incorrect diagrams. Both the response time and the accuracy of responses were measured. Two examples of notations used in the experiment are shown in Fig. 8. When matching diagrams to the specifications, the (a) notations performed better, while when identifying errors in the diagrams, the (b) notations which are less intuitive and more ambiguous had better performance. It appears that subjects are less at ease with these notations and are more likely to detect errors in the diagrams.

There have also been experiments by Cox and Phalp on applying conventions and styles to textual use cases [3,15]. The results showed that their impact on quality is not always obvious and in an experiment comparing two guideline sets, the leaner one performed as well as the other.

Thus there is little empirical evidence in the covered literature regarding the benefits of conventions and the results of few student experiments are not conclusive. The quality impact of conventions seems to depend on the task, the complexity of conventions and tool support as well and empirical studies should describe these factors better in order to help evaluating the usefulness and cost.

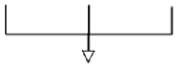
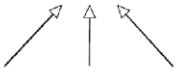
Notational Difference	Variation (a)	Variation (b)
Inheritance direction (N1)	 (Page-Jones 2000)	 (Purchase, Allder & Carrington. 2000)
Inheritance arcs (N2)	 (Page-Jones 2000)	 (Rumbaugh et al. 1999)

Fig. 8. Examples of the notational variations used in the experiments of Purchase et al. [P28].

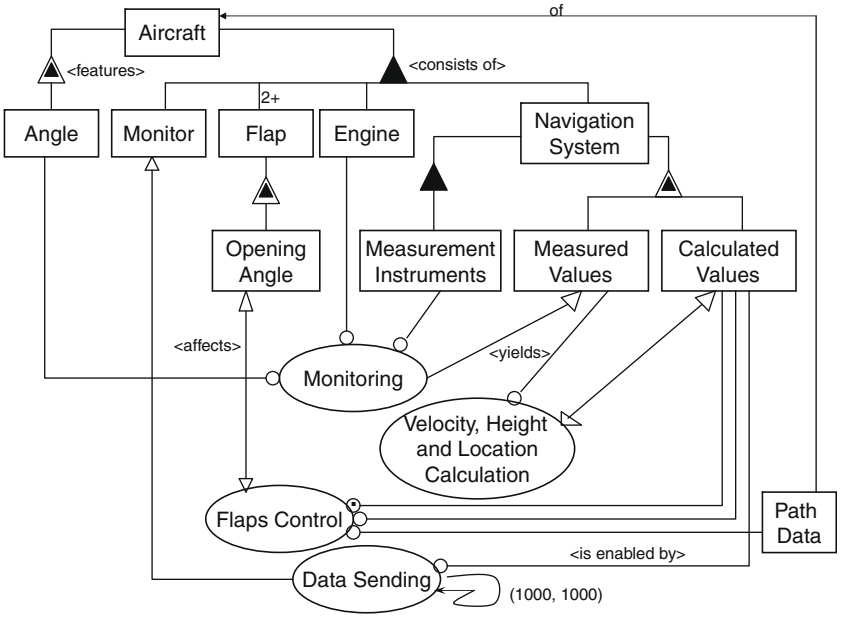


Fig. 9. An OPD showing a top-level view of an avionics navigation system [P27].

5.3. The single-diagram approach

The Object Modeling Technique (OMT), the main ancestor of the Unified Modeling Language (UML), and UML are both approaches that advocate modelling a system in several diagrams. The main benefit is being able to focus on one aspect at a time. On the other hand, it is easy to introduce inconsistencies between diagrams and if a reader must concurrently refer to multiple diagrams in order to understand an aspect, comprehensibility may decrease. Peleg and Dori call this *the model multiplicity problem* [P27] (or to be more consistent "the diagram multiplicity problem"). Lange also writes that in its recent version, UML 2.0 provides thirteen diagram types, such as the use case diagram, the class diagram and the activity diagram. Each diagram type provides a different view on the described system and may be of interest of a specific stakeholder. Eventually all diagrams of a model describe the same system. Thus there is overlap between diagrams which entails the risk for contradictions between diagrams, so called inconsistency defects [P19]. Reinhartz-Berger and Dori write that UML is unnecessarily complex in many ways, and this inherent deficiency hinders coherent modelling and comprehension of systems [P30].

Technical solutions that involve sophisticated CASE tools to impose consistency alleviate manual consistency maintenance, but they do not address the core problem of mental integration. Based on the above arguments, some advocate use of single-diagram (Peleg calls the approach for single model) approaches.

5.3.1. Examples and the impact on model quality

We found one example of a single-diagram approach discussed and evaluated by student experiments in two papers; i.e., [P27] and [P30]. OPM specifies both graphically and textually the system's static-structural and behavioural–procedural aspects through a single unifying model. The elements of the OPM ontology are entities (things and states) and links. A thing is a generalization of a (stateful) object and a process – the two basic building blocks of any OPM-based system model. Links can be structural or procedural. A structural link expresses a static relation between two objects. Procedural links describe the behaviour of a system. Fig. 9 shows an example from [P27]. Objects are shown with rectangles while processes are shown with ovals. Objects can participates in processes as shown by circles and receive events shown by arrows.

One may, however, argue that UML and other modelling languages are multi-diagram because they are meant for industrial-sized application, and that a single-diagram language just doesn't scale up to real-world use. Peleg and Dori write that in their experience with real-life applications, OPM is easily and naturally scalable. The scaling mechanism of OPM is based on detail-level decomposition (which includes zooming in/out, unfolding/folding, and state expressing/suppressing) rather than the viewpoint-based decomposition (into separate diagrams for the static aspect, the dynamic aspect, the functional aspect, etc.) which multi-diagram languages employ. UML 2.0 supports also detail-level decomposition for example in sequence diagrams.

5.3.2. Examples of empirical evidence

Peleg and Dori write that two major open questions related to diagram multiplicity vs. singularity have been (1) whether or not a single diagram, rather than a combination of several diagrams, enables the synthesis of a better system specification and (2) which of the two alternative approaches yields a specification that is easier to comprehend [P27]. They have addressed these two questions through a double-blind controlled experiment with students as subjects. The two approaches for modelling used in the experiment are OPM and OMT extended with Timed Statecharts. The students were divided in two groups: The participants of the first group were asked to specify a system described textually in the problem statement using OMT, while the participants in the second group were asked to specify the same system using OPM. The quality of the resulting specifications was thoroughly analyzed. The results showed that OPM was significantly better than OMT in terms of overall specification quality; evaluated by the number of errors in the specifications such as missing events or missing feature.

For comprehension experiment, the first group received specifications of a system in OPM and the second in OMT and they were requested to answer a questionnaire. The specification comprehension results show that there were significant differences between the two methods in specific issues. The study is inconclusive regarding overall specification comprehension. Retrieving information from a single OMT diagram was easier in some cases than retrieving the same information from the single OPM model, which is more complex. The relatively large number of issues for which no difference between the modelling languages exists underlines the advantage of focusing on a single diagram over spreading the specification information over several diagrams.

Reinhartz-Berger and Dori have also performed an experiment with students comparing UML and OPM for modelling of web applications [P30]. The goal of the experiment was to compare OPM to UML with respect to two aspects: (1) comprehension, namely the level of comprehending a given model expressed in each language, and (2) construction, i.e., the quality and ease of modelling a given system in each language. In some questions when evaluating comprehension, UML scored higher but the general conclusion while OPM was easier to understand and apply by untrained users.

Both experiments show that there are cases in which the existence of separate views can potentially help answering certain questions about a specific aspect of a system, which is expressed in only one type of diagram. But if answering a question needs combining information from multiple diagrams, the single-diagram approach scores higher. The single-diagram approach may also help identifying incompleteness and inconsistency defects. DSMLs tend to have fewer diagrams and thus solve the problems introduced by using multiple diagram modelling languages. Unfortunately, our search did not return any empirical evidence from industry on the benefits of the single-diagram approach.

5.4. Formal models

The syntax and semantics of UML and some other modelling languages are informal and imprecise, making analysis of models difficult. Formal models have the advantage of being precise, support formal analysis and proof, and allow execution and generation. However, the usage of formal models has been limited: they are not expressive enough for many real world applications, formal models are often complex and hard to read, and constructing a formal model may be a difficult, error prone and expensive process.

Since UML is almost the de-facto modelling language in many domains and its informality is considered as a problem, many have tried at making it formal; either by relating it to a formal language, using OCL constraints and tools that may verify the constraints, or developing UML profiles. We discuss domain-specific approaches including UML profiles in Section 5.5. In most approaches, the authors have focused on a few UML diagrams since formalizing all diagrams would require a lot of effort and increase the complexity of the problem.

5.4.1. UML in combination with formal languages

Under the multiple views of UML, the developers can decompose a software design into smaller parts of manageable scales.

However, Liu et al. refer to literature that discusses several challenging issues that inevitably arise from such a multi-view and multi-notational approach [P23]:

- Consistency: the models of various views need to be syntactically and semantically compatible with each other (i.e., horizontal consistency).
- Transformation and evolution: a model must be semantically consistent with its refinements (i.e., vertical consistency).
- Traceability: a change in the model of a particular view should lead to corresponding consistent changes in the models of other views.
- Integration: models of different views need to be seamlessly integrated before software production.

Consistency checking and formal analysis of UML models have been widely studied in recent years. The majority of approaches focus on the formalization of individual diagrams and only treat the consistency between one or two views. Liu et al. have used the Object-Oriented specification Language (OOL) to formalize UML system models and check inter-diagram consistency. Class diagrams, sequence diagrams and state machines are formalized with adding OOL statements. Thus the consistency of UML models is transformed to the well-formedness of OOL specifications.

In another paper, Haesen and Snoeck discuss the problems of consistency checking in general, and horizontal consistency checking in conceptual models specifically [P11]. They describe a method called MERODE which has tackled the problem of inconsistency by defining a formal syntax and semantics for different views. In order to keep this manageable, MERODE has drastically reduced the number of views and concepts that can be used. It should therefore be considered as a Domain-Specific Language (DSL) for the conceptual modelling of management information systems.

Other than solving the inconsistency problem, advantages of formalism are model execution, simulation and analysis. For example, there have been various approaches to make formal specification models, such as using executable *Z* or relating the formal specification language *B* to UML (see [P10] for references). McUmber and Cheng have developed a general framework for formalizing a subset of UML diagrams in terms of different formal languages based on a homomorphic mapping between metamodels describing UML and the formal language [P24]. This framework enables the construction of a consistent set of rules for transforming UML models into specifications in the formal language. The resulting specifications derived from UML diagrams enable either execution through simulation or analysis through model checking, using existing tools. The paper includes an example of formalizing UML in terms of Promela, the system description language for SPIN. They have also constructed a prototype tool called "Hydra" and evaluated it on an industrial case of an embedded system. Using Hydra, they were able to move directly from UML class and behaviour diagrams to model checking and simulation. According to the authors, SPINs model checking and simulation capabilities were extremely useful during the behaviour analysis.

Konrad et al. have also translated UML models to formal specifications using Hydra [P14]. UML models were extended with constraints to check their behaviour consistency for adherence to constraints defined in a goal model. The work is part of a process called i^{2}MAP (incremental and iterative Modelling and Analysis Process). Finally, Choi and Bunse propose translating UML diagrams into the input language of SPIN in order to check behaviour consistency [P5].

5.4.2. Using OCL constraints

UML focused primarily on the diagrammatic elements and gave meaning to those elements through English text. Later, a constraint language was added to the specification, the Object Constraint Language or OCL.[11] OCL allows the integration of both well-formedness rules and assertions (i.e., pre-conditions, post-conditions, invariants) in UML models. The former are useful to validate especially the syntax of a UML model, whereas the latter can be exploited to verify the conceptual constraints. Pre-conditions and post-conditions provide a mechanism to specify the properties required before and after the execution of an operation, respectively, but do not specify how that operation internally works. The recent development of version 2 for both OCL and UML is a breakthrough in order to completely define the semantics of a method in an object-oriented system. In these latest versions, it is possible to define a behaviour specification in OCL for any query operation (an operation without side-effects).

Adding OCL constraints allows analysis and verification by tools. Giese and Heldal propose adding OCL constraints to the post-conditions of UML models to expose gaps and ambiguities in informal descriptions [P10]. Since formal models are harder to read, the authors propose producing informal models from formal ones; for example in a natural language.

Hnatkowska and Walkowiak discuss the problem of consistency within and between artefacts [P12]. They have developed a development approach called the Robust Software Development Profile (RSDP) which introduces three models: context model, use case model, and analysis model. They have used the OCL language to formulate both inter, and intra-consistency rules for these models. The rules are partly implemented within the OCL Evaluator tool. The above-mentioned models were defined in Rational Rose and Poseidon for UML, next they were transformed to XML format, partially rewritten (if needed) and verified within the OCL Evaluator tool. However, Rational Rose and Poseidon were not consistent with the XMI MOF specification, which limits their usage for consistency checking.

5.4.3. Examples of empirical evidence

In our survey on MDE experiences form industry presented in [11], we described a case from Motorola that refers to the benefits of formalism for catching defects by simulation. Otherwise, the studies related to formal models include only examples to demonstrate the practice. The only industrial case is mentioned in [P24] where details are left out.

5.5. Domain-specific modelling languages and UML extensions

A *domain* consists of all possible statements that would be correct and relevant for solving a problem. Hence, every unique problem has a unique, usually, evolving domain [P22]. A Domain-Specific Language (DSL) is a language designed to be useful for a limited domain, in contrast to a General Programming Language (GPL) that is supposed to be useful for multiple application domains. This distinction is orthogonal to many other language classifications. For example, there are indifferently visual (graphical) or textual GPLs or DSLs. Similarly DSLs and GPLs may fall under various categories of being object-oriented, event-oriented, rule-oriented, function-oriented, etc. A Domain-Specific Modelling Language (DSML) is thus a modelling language (usually visual) that is used for Domain-Specific Modelling (DSM). DSM has been subject of a recent book by Kelly and Tolvanen [6] that we use here for definitions. In DSM, modelling is done by using domain concepts and a DSML also includes domain rules that prevent designers from making illegal designs. The implementation details are hidden and thus the level of abstraction is high. Domain-specific models are directly transformed to code by using domain-specific

[11] http://www.omg.org/technology/documents/modeling_spec_catalog.htm.

generators. Models can equally be used for executing, testing and debugging applications since they are formal.

Also when using GPLs like UML, the base language may be extended with domain-specific enhancements using profiles that let us add new attribute types for model elements, classify them with stereotypes and have domain-specific constraints by using OCL [6].

Thus DSMLs may be designed from scratch or by extending a base language. In UML there are two different ways of describing extensions; i.e., by MOF metamodel extensions and UML profiles (by defining stereotypes with tags and constraints). Every approach has advantages and disadvantages:

- Designing a language from scratch or extending a metamodel needs expertise in language engineering and is a lot of work; also to develop an editor, code generator and modify tools. Safa writes that DSLs with a limited user base are costly to develop and maintain, and may have poor design or not be easy to use [P33]. Trask et al. also mention the difficulties in changing a metamodel and subsequently editors and code generators [P37]. The advantage is the possibility to design a language that meets the needs of a target domain.
- UML profiles offer a limited extension mechanism since totally new types cannot be added to the language. Also UML profiles cannot allow taking anything away from UML and one cannot change the semantics of UML elements. The complexity of working with UML is therefore still in place. Besides, many tools do not know how to deal with stereotyped elements. The advantage is being able to use third-party modelling tools.

Although there is still weak support by tools for both approaches, they have gained attention by industry because of the short-comings of GPLs and the promises of DSLs.

5.5.1. Examples and the impact on model quality

Several aspects of model quality are improved by using a DSML or UML profile as discussed below (since profiles are also a first step towards DSML, we will use the DSML expression to cover both):

- Comprehensibility by tools is improved by adding semantics to an informal language which facilitates analysis and generation. Stereotypes can also add additional properties – information – to the stereotyped element [P34].
- Using a language close to the domain improves comprehensibility by humans, especially by non-technical domain experts. The solution domain gets closer to the problem domain which also improves maintainability of models. Profiles or DSM introduce simplicity to the design [P39] and narrow the communicative gap between engineering and business.
- Constraints can be added to models to define restrictions on the usage of base modelling elements and thus improve correctness of the models [P34]. These constraints may be defined in OCL when using UML.

Note that a DSML is defined formally supported by some tool [6]. Therefore the advantages of being formal are achieved, in addition to better comprehensibility and confinement (suitable for the domain). The impact on changeability may be positive or negative as discussed below.

5.5.2. Examples of empirical evidence

Empirical evidence reported in the studies cover student experiments on the benefits of using stereotypes and industry experience reports on the benefits and risks of using DSMLs.

Kuzniarz et al. have performed a controlled experiment with students as subjects in order to evaluate the influence of UML stereotypes on the understanding of UML models [P16]. Two sets of models were used in the experiment; one stereotyped and one not. Understandability of the designs was measured by two dependent variables: (1) total score (NRESP) – the number of correct answers for each subject when asked questions about the design; (2) time (TSEC) – the time (in seconds) which was required to fill in a questionnaire on the design of the systems. The results of the experiment support the claim that stereotypes with graphical icons for their representation play a significant role in comprehension of models. The subjects understood the stereotyped models better (the improvement achieved in this category was 52%) and they were more consistent in their answers when the stereotypes were involved. On average the relative amount of time for a correct answer was shorter for the stereotyped model.

Staron et al. describe two additional experiments in order to verify the credibility of the design of the above experiment before replicating it in industry [P35]. The improvements were not significant but stereotypes helped in particular when it comes to having correct answers. The industrial experiment was conducted at Volvo Information Technology, at their site in Gothenburg, Sweden with only four professionals. The number of correct answers was higher for the stereotyped models but the time spent varied. Staron et al. concluded that in general stereotypes improve the correctness of understanding of UML models. Improvements were achieved in all experiments – and half of the results were statistically significant.

We have found several cases describing experience and benefits of using UML profiles or DSMLs in industry, but often without quantitative data. Two cases are mentioned here while other examples are to be found in [11]. The benefits are often related to improved generation of correct code by adding semantics in domain concepts, and improved understandability by using the language of the domain:

- In a paper describing development at Phillips, Jonkers at al. write that using small DSMLs, as opposed to a universal modelling language such as UML, brings the modelling discipline much closer to the domain experts and at the same time enables simpler maintenance and evolution of such models, which contributes to the desired productivity increase as well as to the agility of the model-driven development [P13].
- In a paper from Matsushita Electric Works in Japan, Safa writes that practitioners considered UMLs object-oriented notations too far apart from the *C* procedural world used for implementation [P33]. A corporate language has been evolved naturally over the years to express requirements, designs and implementations matters. It has notations, conventions and semantics that map precisely the problem domains, and it evolves incrementally when the problem domain changes. The approach has risks as well. Due to the metamodeling delay necessary to define visual languages, editors and compilers, the domain-specific modelling tool lags behind practice, so it is at risk of being perceived as constraining, especially for practitioners used to drawing with free-format whiteboards, pen and paper and general purpose diagram tools like Microsoft© PowerPoint.

5.6. Generating models or diagrams from other models or diagrams

Abstraction is one of the main techniques to handle complexity of software development. In MDE, abstraction is combined with stepwise refinement of models by transformations. Transformations are either Model-to-Model (M2M) or Model-to-Text (M2T). A transformation takes one or several models as input and produces a model (or models) or text, as output. During transformation, output models are supplied with information not present in

the input model. An example of such information is the platform concept during transformation of a PIM model into a PSM model. Thus generation supports separation of concerns and adding details later; not by manual work but by applying transformations, where links between these artefacts can be preserved and used for analyzing the impact of changes, model debugging or synchronizing artefacts.

Obviously, vertical consistency between models is improved when they are generated from other models. Transformations may also be applied between diagrams on the same abstraction level to improve horizontal consistency in multi-diagram modelling approaches. Haesen and Snoeck discuss that consistency checking can be done *by analysis* (an algorithm detects inconsistencies between deliverables), *by monitoring* (meaning that a tool has a monitoring facility that checks every new specification), and *by construction or generation* (meaning that a tool generates one deliverable from another and guarantees semantic consistency) [P11]. While monitoring is useful for error prevention and analysis for error detection, the focus of this section is on generation for error prevention.

5.6.1. Examples and the impact on model quality

Ryndina et al. write that there are two approaches for solving the horizontal consistency problem between two diagrams [P31]. One is to determine the *overlap* between two given diagrams where consistency conditions can be defined and checked. The problem with this approach is that the two diagrams are defined in two modelling languages and consistency conditions should be defined across language boundaries. The second approach is to generate one diagram from the other. With this approach, consistency conditions are defined between diagrams expressed in the same language. Fig. 10 shows the two approaches applied on business process and object life cycle diagrams. The latter approach is beneficial according to the writers, as defining consistency conditions and interpreting inconsistencies between two diagrams in the same language is easier than across language boundaries. A prototype tool can verify that consistency conditions are held.

Haesen and Snoeck have proposed to use the observer pattern to achieve consistency between different views in conceptual models: as the user changes a specification in one view, all other views are informed about the modification [P11]. For example the command of adding an object type creates the object type and the default finite state machine for that object type. As a result, the developer will initially receive a ready-made default finite state machine in which (s)he can add, modify or delete states and transitions to model the behaviour. This is an example of achieving consistency by construction using their terminology.

5.6.2. Examples of empirical evidence

Studies covered in this review included only examples on applying the practice but no empirical evidence.

5.7. Summary of the section

This section provided an overview of practices proposed to improve the quality of models. We identified six classes of practices and discussed them with examples, their impact on model quality and examples of empirical studies. While some practices are evaluated by experiments and industrial cases, others are only demonstrated on examples. Table 4 summarizes this section where "**+**" indicates benefits and "**!**" indicates drawbacks of practices.

Practices can of course be combined, for example DSMLs often include constraints and formal semantics to prevent errors and facilitate generation, and the number of diagrams is often reduced. Having a model-based development process that includes guidelines for modelling will help developers to set the practice in life. Using a practice will sometimes enhance one quality goal while affecting another one negatively. For example formal models are easier to analyze and verify by tools, but are not easier to comprehend by humans. Implementing each practice has some cost which should be weighted against the benefits to find the balance. In the next section we discuss the impact of practices on the *6C* goals introduced in Section 4.

6. Integrating the results

We introduced the *6C* model quality goals in Section 4 and the six practices proposed to improve the quality of models in Section 5, while the impact of practices on the *6C* goals are summarized in this section.

6.1. The impact of the proposed practices on the 6C goals

For each quality goal, the two classes of practices are discussed. We remind that the first group related to "modelling process" in-

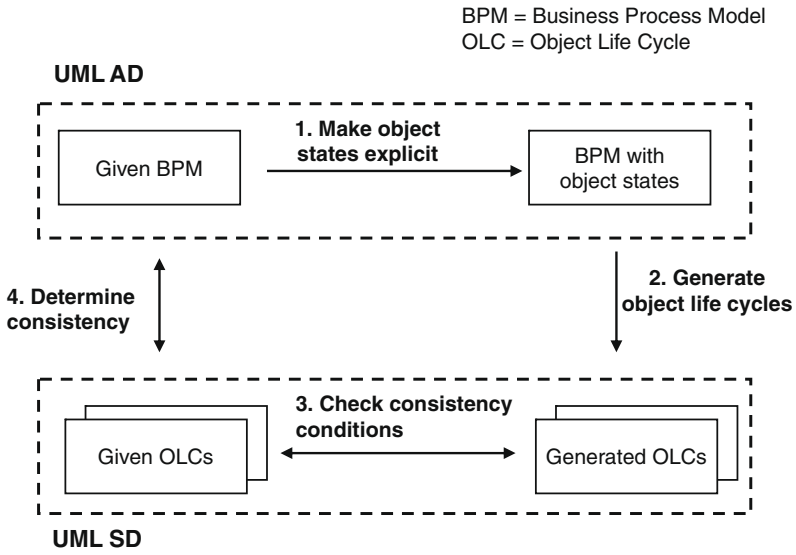


Fig. 10. Solution overview in [P31].

Table 4
The benefits (marked with "+") and drawbacks (marked with "!") of the proposed practices and the state of evidence from empirical studies in the covered literature.

Approach	Main benefits or drawbacks	Empirical evidence
Related to modelling process		
Model-based development process	+ Having a development process adapted to MDE is important for controlling the quality of activities and artefacts and avoiding non-uniformity + Metrics may be added to evaluate progress and the quality of artefacts + Best practices may be included in the process ! Developing or updating a process and training people on the process require effort ! Having a process does not guarantee that it is followed	Although there is agreement in experience reports from industry regarding the importance and positive effect of process, the impact of processes on the productivity or quality is not properly documented
Modelling conventions	+ Easy to provide based on experience and literature ! Increases effort; should not be overwhelming	Student experiments show that tool support for enforcing and checking of conventions is important The impact on error prevention or detection is not always straightforward to predict. For example less intuitive conventions may be better for error detection
Single-diagram approach	+ It is easier to detect incompleteness if the information is presented in one diagram ! All information gathered in one diagram may improve comprehensibility compared to cases where information from multiple diagrams must be integrated for comprehension	The method is evaluated in student experiments that confirm single-diagram models have fewer errors while comprehensibility depends on the task
Related to formal approaches and automation		
Formal models	+ Support for formal analysis and proof, execution and generation + Model checking and simulation tools are available ! Formal models are more difficult to comprehend. However, they may be combined with informal ones ! Formality comes at a cost	Only described by examples
DSMLs and UML extensions	+ Models developed using domain concepts are easier to comprehend for domain experts + Enough details for transformations may be added and models may be made formal + Constraints may be added to prevent wrong usage ! Developing a language and supporting editors and tools need expertise and is costly	A number of student experiments are performed which verify the improvement in comprehensibility by suing UML stereotypes, although not always significant DSMLs are receiving growing attention by industry, with positive feedback. However, keeping the language updated with the domain is difficult
Generating models/diagrams from models/diagrams	+ Supports separation of concerns since details may be added during transformation (like in DSLs) + May be applied to improve both horizontal and vertical consistency between artefacts ! Transformations are costly and should preserve model characteristics ! Transformation is a core practice in MDE supported by tools while the challenges of traceability and synchronizing artefacts are not properly addressed	Discussed by examples

cludes model-based development process, modelling conventions and the single-diagram approach, while the second group related to "formal approaches and automation" covers formal models, domain-specific solutions and generation.

C1-Correctness. Regarding modelling process, several studies propose using conventions in order to prevent syntactic and semantic errors [P1,P3,P38], while Unhelkar [P38] also provides checklists to check the correctness of UML diagrams. Berenbach proposes enforcing some of the conventions by tool [P3]. Performing inspections [P15,P22,P26] and model analysis techniques [P40] are also proposed for quality assurance of models.

Regarding formal approaches and automation, Giese and Heldal [P10], Staron et al. [P34] and van Der Straeten [P36] propose using OCL constraints in order to remove semantic ambiguities from models and prevent wrong usage of elements. Using stereotypes [P34] and defining formal semantic of elements (for example in a DSML) are also techniques that remove ambiguity of informal semantics and thus improve semantic correctness. McUmber and Cheng introduce a framework for formalizing a subset of UML diagrams by mapping between metamodels describing UML and the formal language to be able to use model checking and simulation tools, for example SPIN to check for correctness and other quality goals such as consistency [P24].

C2-Completeness. Regarding modelling process, some research hints that single-diagram approaches have fewer defects regarding missing elements compared to the multi-diagram approaches [P27]. Conventions provided as guidelines, checklists or best practices in modelling processes are also proposed to improve completeness of models, as in [P1,P3,P25,P38]. Examples are to be found in Table 3. Tool-support is discussed by Berenbach who proposes defining completeness rules that can be programmatically verified [P4] while metrics for completeness evaluation are proposed by Berenbach [P4] and Lange et al. [P21] who propose defining a special task for completeness evaluation that is performed based on metrics collected from models.

Regarding formal approaches and automation, generating models from other models is a technique for achieving completeness with an example given in [P31] where object models are generated from process models with all the necessary states.

C3-Consistency. Regarding modelling process, conventions regarding consistency between diagrams are proposed in [P1,P5,P3, P25,P38] with examples given in Table 3. Gavras et al. propose defining a traceability strategy in the modelling process, which refers to the ability to establish relationships between model concepts that represent the same concept in different models [P9].

Regarding formal approaches and automation, consistency has received more attention in literature than correctness and completeness since it may be improved to a large extent by using tools and formal languages, for example inter- and intra-consistency rules may be checked by OCL evaluator tools [P12]. Consistency constraints may also de defined in a DSML [P39] and consistency conditions can be checked during transformations. By using a formal language, consistency of UML diagrams can be transformed to well-formedness of the language specifications [P11,P23]. Finally, Haesen and Snoeck propose using tools that implement the observer patterns and generate the necessary elements when diagrams are updated to keep diagrams consistent with one another [P11].

C4-Comprehensibility. Regarding modelling process and when it comes to comprehensibility by humans, we found conventions for naming and structuring of models [P1,P3,P25] (since information is easier to locate in a proper-named and well-organized model), aesthetics of diagrams [P1,P28,P29,P7], closeness to users' view [P3] and to the problem domain [P25], and documentation of models [P1,P3]. Examples can be found in Table 3. Finally, experiments on the single-diagram approach indicate that these models are sometimes easier to understand compared to the multi-diagram approach where information is spread over several diagrams, while comprehensibility is decreased if answering a question requires information on only one aspect that may be expressed in one UML diagram [P27,P30].

Regarding formal approaches and automation, while using formal languages in combination with UML is proposed to improve consistency, it is also discussed that formal models are more difficult to read for humans and it is better to generate them from the informal ones and constraints should be first written in a human-readable form [P34]. On the other hand, formality allows simulation and analysis which may improve comprehensibility by humans [P22,P24,P34]. Using concepts close to the domain as in a DSML is also supposed to improve the comprehensibility of models especially by non-technical experts [P13,P39,P33] and some experiments suggest that stereotyped models are understood better [P16,P35].

Comprehensibility by tools is achieved by having a formal or precise semantics by adding such semantics in models, stereotypes and DSMLs, and of course during transformations.

Some research discuss that problems with comprehensibility by users may indicate poor design [P7,P8] while good design improves also comprehensibility. We have not covered design quality in this review but this observation is interesting to have in mind.

C5-Confinement. Regarding modelling process, Berenbach [P3], Mitchell [P25] and Ambler [P1] have proposed conventions that impact confinement such as using the right modelling artefacts, being free of implementation and design decisions for an analysis model, having focus on correct separation of concerns (also in [P5]), identifying scope as early as possible, having focus on domain concepts and starting modelling process from them, clear separation between models and what they should cover, and modelling from different views. Many of these conventions may be integrated in a model-based development process, for example Gavras et al. emphasize defining an activity for selecting modelling languages and tools appropriate for the domain and the needs of modelling [P9].

Regarding formal approaches and automation, sometimes developing a DSML is the right solution for an organization [P13,P33,P39] since it includes only elements and diagrams necessary for the domain.

C6-Changeability. Maintainability of models and updating them is a major challenge as it is with code, especially when models gets large and complex as it is in many industry applications. However, maintenance and evolution of models has not received much attention by now.

Regarding modelling process, maintenance of models is discussed in Agile Modelling (AM) which has recommendations such as a single source of information, creating simple content and depicting them simply.

Regarding formal approaches and automation, Jonkers et al. discuss that modelling in a DSML brings the modelling discipline much closer to domain experts and at the same time enables simpler maintenance and evolution of such models, which contributes to the agility of the model-driven development [P13]. On the other hand, Safa writes that updating the metamodel of a DSL and the associated tools with a limited user base is costly and as a result the language may lag behind the changes in the domain [P33]. We also think that generating models from other models is a practice that allows keeping models in synch with one another when changes in one happen.

Many practices regarding maintenance of code also apply to models. For example models that are well-organized and well-documented are easier to maintain and update. A model which is easier to communicate may reduce the "mythical man month", the time that is normally taken to learn a new system, which will improve the cost effectiveness of maintaining systems [8].

Summary. Fig. 11 summarizes the literature on the impact of the practices on the *6C* goals where dashed lines indicate that both positive and negative impact is observed. As depicted in Fig. 11, the proposed practices often impact several quality goals:

- Improving the modelling process may impact all quality goals if proper activities are included.
- Coding conventions or styles have earlier been promoted to improve the quality of code. In order to improve the uniformity of models and prevent defects, some authors advocate the use of modelling conventions. Styles, rules and conventions are kind of best practices proposed to improve all aspects of model quality and should be included in a model-based development process.

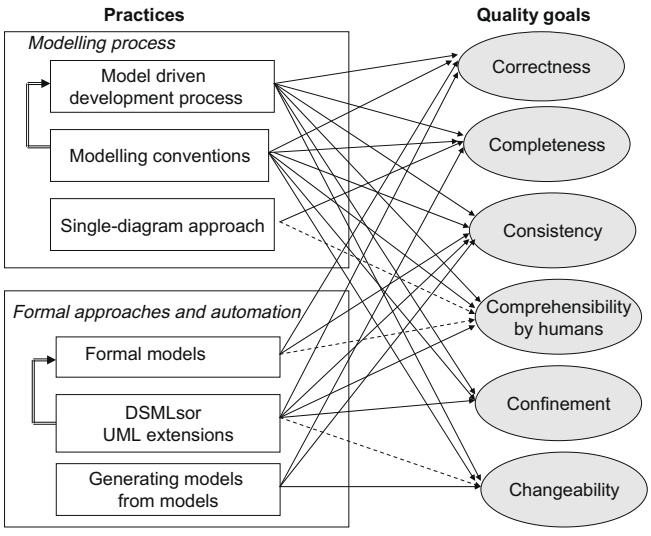


Fig. 11. The impact of practices on model quality goals. Continuous lines indicate positive impact while dashed lines indicate that the impact may be positive or negative.

- Using less number of views is proposed to reduce the complexity of modelling and improve completeness. It may impact comprehensibility by humans positive or negative.
- Using a formal modelling language improves correctness and consistency of models and may also improve comprehensibility by humans if models may be simulated. Formal models are on the other hand more difficult to read for humans.
- DSMLs or UML profiles allow developing models with the vocabulary of the domain that is more comprehensible for humans. Other advantages are developing models that are formal, more concise, correct and suitable for code generation. However, updating the language and editors is difficult and models should be updated with changes in the metamodel.
- Model-based generation by transformations improves consistency between artefacts and a transformation tool can check models for their correctness and completeness during transformation.

6.2. Assessing quality

In this section we present some observations regarding tool support and quality assurance techniques from the covered literature. The main methods for detecting errors or assessing quality are:

- *Inspections*: several quality goals such as consistency, completeness and confinement can be assessed by means of manual human inspections; as proposed in [P3,P22,P26], also by using checklists [P38]. Both modelling experts and non-technical experts should be involved in inspections; especially for evaluating comprehensibility and confinement aspects. The OORT techniques (Object-Oriented Reading Techniques) is an example of systematic inspection techniques to inspect (“compare”) UML diagrams with each other for completeness and consistency (vertical and horizontal) [2].
- *Tools for error detection*: some have developed tools that check models for inconsistency, incompleteness and incorrectness problems such as naming conflicts, missing associations and incorrectly defined interfaces. Examples are the DesignAdvisor tool [P3] and SDMetrics [P20]. Tools for model checking based on formal approaches (adhering to rules and constraints for example related to consistency and requirement goals) are covered as well such as the SPINs model checking and simulation environment [P5,P24], Hydra framework [P14], MCC+ [P2] and OCL Evaluator tool [P12].
- *Collecting metrics from models*: Berenbach proposes collecting metrics from models to evaluate their completeness [P4] and Lange et al. have developed the MetricViewEvolution tool for collecting metrics and visualization them [P21]. Saeki and Kaiya propose defining metrics at the metamodel level [P32].

For evaluating the usefulness of practices, we found examples of:

- *Experiments*: controlled experiments are often performed in academia, for example related to using UML profiles [P16] or the single-diagram approach [P27].
- *Pilot studies*: some industry cases described in studies are actually pilot studies performed in order to evaluate usefulness of an approach in a specific context, for example in [P26] and related to DSMLs [P33,P39].
- *Feedback from practitioners* : some studies have systematically collected feedback in industrial cases and analyzed them, such as in [P34].

7. Discussion

This article covered mainly two issues: identifying model quality goals (*6C* goals) important in model-based software development and an overview of practices proposed in literature to improve model quality, both based on the results of a systematic literature review. The question facing us is whether the *6C* classification is more useful than other classifications defined in [P15,P22,P26,P38]; such as dividing quality goals into syntax, semantics and pragmatic quality. When trying to define what exactly syntactic, semantic or other quality goals mean in the above studies, the authors often tend to use the same terminology covered by the *6C* goals; i.e., correctness, completeness, consistency etc. There are other quality goals for models – such as being simple – that are defined as being comprehensible and easy to change, which are thus covered by our classification.

In addition to using a simple terminology, the *6C* classification is based on the results of a systematic review of literature on model quality. In our opinion using the review results provides relevance. For example one may talk of consistency as a semantic quality type while the term “consistency between models” is better understood by practitioners and also used in several studies.

Another question is whether we can join any two goals or remove any without significant impact on the discussion, which we mean is not possible. Other quality goals may be added if necessary and future research on the subject is necessary.

Quality of models is especially important in MDE:

- Since models are transformed into other models, they should be correct. Otherwise, the principle of “garbage in, garbage out” applies [P40].
- Model completeness is a prerequisite for transformation, consistency checking and implementation. Coverage is one of the requirements of completeness; for example all the use cases are covered in the implementation.
- Consistency between diagrams becomes important if information from separate diagrams should be combined for the purpose of understanding or generation. Consistency between models of the same system is important for keeping them in sync for future evolution.
- Comprehensibility either by tools or humans is the main reason for doing any modelling.
- MDE often involves developing several models of the system with different purposes. Thus models should include information depending on the purpose.
- Finally, models should be easy to change in order to support evolution and maintenance. Persuading industry to use models depends on whether modellers can change models easily and continuously.

In short, it is hard or impossible to create something complete and correct from something incomplete, erroneous or inconsistent. Persuading industry to use MDE requires taking away some burden of development by providing tools and methodologies that support developing and maintaining high-quality models. For example a recent article on model-based testing states that the ultimate success of the approach relies on the quality of models that support them [14], such as having enough details (being complete).

As the results of this review show, several practices are proposed for improving the quality of models and some studies also include empirical evidence. Studies on the impact of modelling processes on the quality of models report positive feedback from industry without providing details. While one may find a lot of literature on the benefits of software processes in general, the relevant tasks and activities for a MDE approach should be evaluated in industry cases. Studies discussing the benefits of formal models and generating models or diagrams from other models or diagrams contained only small examples, while modelling conventions, domain-specific approaches and the single-diagram modelling approach are supported by industrial cases and student experiments. However,

the evidence from the domain-specific cases supports the benefits of formal methods to some degree since DSMLs are usually formal.

Some practices may be enforced by CASE tools, for example OCL constraints prevent making wrong choices and tools can prevent syntactic errors and may keep models consistent with one another if consistency rules are defined and support for checking is provided. Experiments on applying modelling conventions have confirmed that tool-support is important for reducing the effort spent on modelling when using conventions [P18]. Using the proposed practices especially supported by tools is the *preventive approach* or quality by construction. The second approach to quality is an *assessing approach* as discussed in Section 6.2 based on static analysis of models: model checking for formal models, collecting metrics from models or getting feedback via inspections and interviews. Finally, there is also a *predictive approach*, for example the quality of models may be evaluated from the quality of predictions made from them if they are used for simulation and prediction. However, the studies covered in this review did not include any examples on using this approach.

8. Conclusions and future work

This article reviewed literature on the quality of models to answer three research questions. The results are summarized below.

RQ1. What quality goals are defined in literature for models in model-based software development? We identified six model quality goals relevant for model-based software development; i.e., correctness, completeness, consistency, comprehensibility by humans and tools, confinement (as having precise modelling goals and being restricted to them) and changeability (as being easily extensible and modifiable). While some of these quality goals such as consistency are studied in depth and solutions are proposed and implemented in tools, others – such as changeability – are less discussed in the covered literature.

RQ2. What practices are proposed to achieve or improve the above quality goals? We identified six practices and divided them in two groups. The first group is related to modelling process and covers having a model-based development process, using modelling conventions and the single-diagram approach. The second group is related to formal approaches and automation and covers formal models, UML profiles and domain-specific modelling languages, and generating models or diagrams from other models or diagrams. We discussed the impact of the proposed practices on the *6C* goals with examples and empirical evidence reported in the covered literature.

RQ3. What types of models and modelling approaches are covered in literature? Most research covered UML models, however, in approaches where models play a central role in software development or on the right hand side of the spectrum shown in Fig. 1. However, even when models are merely sketches, their quality has gained attention since high-quality models ease communication between development teams. We also found literature covering UML profiles and Domain-Specific Languages in the spirit of model-driven engineering.

Empirical evidence in the covered literature is also included in the article. Modelling conventions and the single-diagram modelling approach have been subject of student experiments that confirm some benefits but question others. For example the impact of conventions depends on the task and tool-support. The benefits of model-based development process and domain-specific modelling approaches (including UML profiles) are observed in industrial cases while formal models and generating models/diagrams from models/diagrams are mostly discussed by examples and no empirical evidence was detected in the covered literature. Additional evidence may, however, be detected by performing a review with focus on empirical studies.

The main purpose of this article has been to provide definitions and classifications that can be part of a quality model with focus on model quality. We have developed a tool for visual specification of quality models as presented in [12] where we intend to insert the results of this review. The next challenge of improving model quality is to select quality goals for a given context and to identify practices that may be applied in that context. Quality goals vary in the lifecycle of a project and for different types of models. For example the degree of required formality and detail vary. Models may also be the intermediate or the final products of software development. In short, a model should "fit for the purpose". Thus a goal-driven process for selecting quality goals and practices is proposed which is subject of our future work.

Other research gaps are identified as well. While traditional quality assurance techniques such as inspections and measurement are applicable to models, they should be adapted to modelling purposes, tasks and artefacts involved. Managing changeability and complexity of large and complex models, keeping them consistent and verifying quality on the model level are challenges in model-driven engineering that are not yet properly covered.

Performing literature reviews is time consuming and integrating the results is not easy. The main benefits are, however, to provide new insight and identify research gaps. One challenge of this review was selecting a terminology for classifying model quality goals that is based on the existing work and is considered useful, without being difficult to understand for practitioners. Since our classification is based on the terminology used in the reviewed literature, we mean that it provides relevance and understandability. We must further improve the classification by increasing the breath of search for studies especially with focus on quality promises of the model-driven engineering approach. We are involved in the MODELPLEX project[12] which has the vision to evolve modelling technologies and tools for complex system development. In MODELPLEX, an empirical research plan is defined in order to evaluate the impact of modelling technologies and tools on several attributes such as the productivity of software developers and the quality of models or generated artefacts. The results of empirical work will be used to evaluate the quality impact of model-drive engineering and the usefulness of our classification.

Acknowledgements

This work has been funded by the Quality in Model-Driven Engineering project (cf. http://quality-mde.org/) at SINTEF and the European Commission within the 6th Framework Programme project MODELPLEX Contract Number 034081 (cf. http://www.modelplex.org). We thank Dr. Marcela Fabiana Genero Bocco and Dr. Michel Chaudron for their valuable comments and suggestions.

Appendix I. List of primary studies included in the review

[P1] S.W. Ambler, The Elements of UML 2.0 Style, Cambridge University Press, 2005.

[P2] M.C. Bastarrica, S. Rivas, P.O. Rossel, Designing and implementing a product family of model consistency checkers, in: Proceedings of the Workshop on Quality in Modelling (QiM'07) held at MODELS 2007, 2007, pp. 36–49.

[P3] B. Berenbach, The evaluation of large, complex UML analysis and design models, in: Proceedings of the 26th International Conference on Software Engineering (ICSE'04), 2004, pp. 232–241.

[12] http://www.modelplex-ist.org/.

[P4] B. Berenbach, Metrics for model-driven requirements development, in: Proceedings of the 28th International Conference on Software Engineering (ICSE'06), 2006, pp. 445–451.

[P5] Y. Choi, C. Bunse, Behavioral consistency checking for component-based software development using the KobrA approach, in: Proceedings of the 3rd International Workshop, Consistency Problems in UML-based Software Development III – Understanding and Usage of Dependency Relationships, held at UML 2004, 2004, pp. 83–98.

[P6] B. DuBois, C. Lange, S. Demeyer, M. Chaudron, A qualitative investigation of UML modeling conventions, in: Proceedings of the 1st Workshop on Quality in Modeling (QiM'06) held at MoDELS 2006, 2006, pp. 79–84.

[P7] H. Eichelberger, Aesthetics of class diagrams, in: Proceedings of the 1st IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT 2002), IEEE CS Press, 2002, pp. 23–31.

[P8] H. Eichelberger, Nice class diagrams admit good design? in: Proceedings of the ACM Symposium on Software Visualization, 2003, pp. 159-ff.

[P9] A. Gavras, M. Belaunde, L.F. Pires, J.P.A. Almeida, Towards an MDA-based development methodology for distributed applications, in: Proceedings of the EWSA'04, 1st European Workshop on Software Architecture, LNCS, vol. 3047, Springer Berlin/Heidelberg, 2004, pp. 230–240.

[P10] M. Giese, R. Heldal, From informal to formal specifications in UML, in: Proceedings of the UML 2004, LNCS, vol. 3273, Springer Berlin/Heidelberg, 2004, pp. 197–211.

[P11] R. Haesen, M. Snoeck, Implementing consistency management techniques for conceptual modelling, in: Proceedings of the 3rd International Workshop, Consistency Problems in UML-based Software Development III – Understanding and Usage of Dependency Relationships, held at UML 2004, 2004, pp. 99–113.

[P12] B. Hnatkowska, A. Walkowiak, Consistency checking of USDP models, in: Proceedings of the 3rd International Workshop Consistency Problems in UML-based Software Development III – Understanding and Usage of Dependency Relationships, 2004, pp. 59–70.

[P13] H. Jonkers, M. Stroucken, R. Vdovjak, Bootstrapping domain-specific model-driven software development within Philips, in: Proceedings of the 6th OOPSLA Workshop on Domain Specific Modeling (DSM'06), 2006, 10 p.

[P14] S. Konrad, H.J. Goldsby, B.H.C. Cheng, i^{2}MAP: an incremental and iterative modelling and analysis process, in: Proceedings of the MoDELS 2007, LNCS, vol. 4735, 2007, pp. 451–466.

[P15] J. Krogstie, A. Sølvberg, Information Systems Engineering: Conceptual Modeling in a Quality perspective, Kompendiumforlaget, Norway, 2000.

[P16] L. Kuzniarz, M. Staron, C. Wohlin, An empirical study on using stereotypes to improve understanding of UML models, in: Proceedings of the 12th IEEE International Workshop on Program Comprehension, 2004, pp. 14–23.

[P17] C.F.J. Lange, Managing model quality in UML-based software development, in: Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice (STEP'05), 2005, pp. 7–16.

[P18] C.F.J. Lange, B. DuBois, M.R.V. Chaudron, S. Demeyer, An experimental investigation of UML modeling conventions, in: Proceedings of the MODELS'06, 2006, pp. 27–41.

[P19] C.F.J. Lange, Assessing and Improving the Quality of Modeling – A Series of Empirical Studies about the UML, Ph.D. thesis, Technische Universiteit Eindhoven, 2007.

[P20] C.F.J. Lange, M.R.V. Chaudron, Defects in industrial UML models – a multiple case study, in: Proceedings of the Workshop on Quality in Modelling (QiM'07) at MODELS 2007, 2007, pp. 50–65.

[P21] C.F.J. Lange, M.A.M. Wijns, M.R.V. Chaudron, A visualization framework for task-oriented modeling using UML, in: Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS 2007), 2007, pp. 289a–289a.

[P22] O.I. Lindland, G. Sindre, A. Sølvberg, Understanding quality in conceptual modelling, IEEE Software 11 (2) (1994) 42–49.

[P23] Z. Liu, H. Jifeng, X. Li, Y. Chen, Consistency and refinement of UML models, in: Proceedings of the 3rd International Workshop Consistency Problems in UML-based Software Development III – Understanding and Usage of Dependency Relationships, 2004, pp. 23–40.

[P24] W.E. McUmber, B.H.C. Cheng, A general framework for formalizing UML with formal languages, in: Proceedings of the IEEE International conference on Software Engineering (ICSE'01), 2001, pp. 433–442.

[P25] R. Mitchell, High-quality modeling in UML, 2001, <http://www.inferdata.com/resources/whitepapers/HQmodeling.4.pdf>.

[P26] H.J. Nelson, D.E. Monarchi, Ensuring the quality of conceptual representations, Software Quality Journal 15 (2) (2007) 213–233.

[P27] M. Peleg, D. Dori, The model multiplicity problem: experimenting with real-time specification methods, IEEE Transactions on SE 26 (8) (2000) 742–759.

[P28] H.C. Purchase, L. Colpoys, M. McGill, D. Carrington, C. Britton, UML class diagram syntax: an empirical study of comprehension, in: Proceedings of the Australian Symposium on Information, 2001, pp. 113–120.

[P29] H.C. Purchase, J.A. Allder, D. Carrington, Graph layout aesthetics in UML diagrams: user preferences. Journal of Graph Algorithms and Applications 6 (3) (2002) 255–279.

[P30] I. Reinhartz-Berger, D. Dori, OPM vs. UML – experimenting with comprehension and construction of web application models, Empirical Software Engineering 10 (1) (2005) 57–80.

[P31] K. Ryndina, J.M. Küster, M. Call, Consistency of business process models and object life cycles, in: Proceedings of the 1st Workshop on Quality in Modelling (QiM'06) at MoDELS 2006, LNCS, vol. 4364, 2006, pp. 80–90.

[P32] M. Saeki, H. Kaiya, Model metrics and metrics of model transformation, in: Proceedings of the 1st Workshop on Quality in Modeling (QiM'06) held at MoDELS 2006, 2006, pp. 31–46.

[P33] L. Safa, The practice of deploying DSM, report from a Japanese appliance maker trenches, in: Proceedings of the 6th OOPSLA Workshop on Domain Specific Modeling (DSM'06), 2006, 12 p.

[P34] M. Staron, L. Kuzniarz, L. Wallin, Case study on a process of industrial MDA realization: determinants of effectiveness, Nordic Journal of Computing 11 (3) (2004) 254–278.

[P35] M. Staron, L. Kuzniarz, C. Wohlin, Empirical assessment of using stereotypes to improve comprehension of UML models: a set of experiments, Journal of Systems and Software 79 (5) (2006) 727–742.

[P36] R. van Der Straeten, Formalizing behaviour preserving dependencies in UML, in: Proceedings of the 3rd International Workshop Consistency Problems in UML-based Software Development III – Understanding and Usage of Dependency Relationships, 2004, pp. 71–82.

[P37] B. Trask, D. Paniscotti, A. Roman, V. Bhanot, Using model-driven engineering to complement software product line engineering in developing software defined radio components and applications, in: ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'06), 2006, pp. 846–853.

[P38] B. Unhelkar, Verification and Validation for Quality of UML 2.0 Models, Wiley, 2005.

[P39] H. Wegener, Balancing simplicity and expressiveness: designing domain-specific models for the reinsurance industry,

in: Proceedings of the 4th OOPSLA Workshop on Domain-Specific Modelling (DSM'04), 2004.

[P40] F. Weil, B. Mastenbrook, D. Nelson, P. Dietz, A. van der Berg, Automated semantic analysis of design models, in: Proceedings of the MoDELS 2007, LNCS, vol. 4735, 2007, pp. 166–180.

Appendix II. Publication channels and summary of the primary studies

The numbers of primary studies identified in each publication channel are in (). The titles of publication channels with more than three primary studies appear in italic.

Books (3)
Ph.D. thesis (1)
Software and Systems Modeling (SoSyM) Journal (0)
Software Quality Journal (1)
Empirical Software Engineering Journal (1)
Journal of Systems and Software (1)
Information and Software Technology (0)
UML/MoDELS conferences (4)
QiM workshops at MoDELS (5)
Other workshops at UML/MoDELs conferences (5)
ECMDA-FA conferences (0)
ICSE, International Conference on Software Engineering (3)
OOPSLA, Conference on Object-oriented programming systems, languages, and applications (1)
DSM workshops at OPPSLA (3)
Digital libraries (6)
Searched Internet for publications of authors as given in references of other papers (Lange and Staron) (4)
Other Internet search (1)
Known from before (1) (this is [P22])

References

[1] C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wust, J. Zettel, Component-based Product Line Engineering with UML, Addison-Wesley Publishing Company, 2002.
[2] R. Conradi, P. Mohagheghi, T. Arif, L.C. Hedge, G.A. Bunde, A. Pedersen, Object-oriented reading techniques for inspection of UML models – an industrial experiment, in: Proceedings of the European Conference on Object-Oriented Programming (ECOOP'03), LNCS, vol. 2749, 2003, pp. 483–501.
[3] K. Cox, K. Phalp, Replicating the CREWS use case authoring guidelines experiment, Empirical Software Engineering 5 (2000) 245–267.
[4] D. Harel, B. Rumpe, Modeling Languages: Syntax, Semantics and All that Stuff. Technical Paper Number MCS00-16, The Weizmann Institute of Science, Rehovot, Israel, 2000.
[5] M. Jørgensen, M. Shepperd, A systematic review of software development cost estimation studies, IEEE Transactions on SE 33 (1) (2007) 33–53.
[6] S. Kelly, J.P. Tolvanen, Domain-Specific Modelling, Enabling Full Code Generation, IEEE Computer Society Publications, 2008.
[7] B. Kitchenham, Guidelines for performing systematic literature reviews in software engineering, v2.3, EBSE Technical Report 2007-01, developed by Software Engineering Group at Keele University and Department of Computer Science at University of Durham, 2007, 65 p.
[8] A. MacDonald, D. Russell, B. Atchison, Model-driven development within a legacy system: an industry experience report, in: Proceedings of the Australian Software Engineering Conference, 2005, pp. 14–22.
[9] P. Mohagheghi, J.Ø. Aagedal, Evaluating quality in model-driven engineering, in: International Workshop on Modeling in Software Engineering (MiSE'07) ICSE Workshop, 2007, 6 p.
[10] P. Mohagheghi, V. Dehlen, Developing a quality framework for model-driven engineering, in: Proceedings of the 2nd Workshop on Quality in Modeling at MoDELS 2007, LNCS, vol. 5002, 2007, pp. 275–286.
[11] P. Mohagheghi, V. Dehlen, Where is the proof? – a review of experiences from applying MDE in industry, in: Proceedings of the 4th European Conference on Model Driven Architecture Foundations and Applications (ECMDA'08), LNCS, vol. 5095, 2008, pp. 432–443.
[12] P. Mohagheghi, V. Dehlen, T. Neple, Towards a tool-supported quality model for model-driven engineering, in: Proceedings of the 3rd Workshop on Quality in Modelling (QiM'08) at MODELS 2008, 2008, 15 p.
[13] P. Mohagheghi, V. Dehlen, A metamodel for specifying quality models in model-driven engineering, in; Proceedings of the Nordic Workshop on Model Driven Engineering, 2008, pp. 51–65.
[14] A.D. Neto, R. Subramanyan, M. Vieira, G.H. Travassos, F. Shull, Improving evidence about software technologies, a look at model-based testing, IEEE Software 23 (3) (2008) 10–13.
[15] K.T. Phalp, J. Vincent, K. Cox, Improving the quality of use case descriptions: empirical assessment of writing guidelines, Software Quality Journal 15 (4) (2007) 383–399.
[16] J. Rech, A. Spriestersbach, D.4.1: quality defects in model-driven software development, deliverable of Vide project 2007, <http://www.vide-ist.eu/extern/VIDE_D4.1.pdf>.
[17] J. Rumbaugh, I. Jacobson, G. Booch, The Unified Modeling Language Reference Manual, Addison-Wesley, 1999.
[18] M. Staron, Adopting model driven software development in industry – a case study at two companies, in: Proceedings of the MoDELS 2006, LNCS, vol. 4199, 2006, pp. 57–72.
[19] A. Watson, A brief history of MDA, upgrade, European Journal for the Informatics Professional IX (2) (2008). URL <http://www.upgrade-cepis.org>.
[20] R.K. Yin, Case Study Research: Design and Methods, Saga Publications, 2003.