The Model Architecture Frame: Quality Management in a Multi Method Environment

Dorothea Beringer

Swiss Federal Institute of Technology, Software Engineering Laboratory, LGL-DI-EPFL, CH-1015 Lausanne, Switzerland
E-mail: dorothea.beringer@di.epfl.ch

Abstract

Most quality management systems (QMS) which integrate software development are based on a single software development method. This particular method is enforced on all projects. The resulting QMS has to fulfil the ISO 9000 standards [13] and enables the control of the projects and the improvement of the development process, supporting it with all necessary guidelines and providing one terminology for the whole company. However, using only one specific method is not necessarily the best solution for a company with very heterogeneous projects and products. Requirements of customers, existing experience and training in the teams, projects still using structured methods and newer ones using object-oriented methods, adaptation of methods for special circumstances, and evolving modelling techniques, all of these factors call for the coexistence of different modelling methods within one company. If a QMS is based on only a single method, many projects might run outside the quality management regulations, unless the QMS excludes the details of the software development process.

This problem can be overcome by the use of model architectures and model architecture frames. A model architecture describes the models that are developed within one specific method or project. All the different model architectures are derived from the generic model architecture frame. Such a frame, and not just one specific method, becomes the basis for the company wide regulations.

This paper describes reasons for having multiple methods and life cycle models within one QMS, explains the basic concepts of defining a QMS using model architectures and model architecture frames, and shows how this approach may be applied in a project.

1 Introduction

Companies with many small and medium sized projects need a QMS that helps the upper management to supervise and to control their projects. Usually this has been achieved by integrating the modelling methods, the project life cycle and quality assurance (QA) into one large development standard (e.g. SSADM or NCR-CASE [15]). These standards are centred around forward engineering projects with one single method and a waterfall like life cycle. Such a company wide standard is quite successful, as long as all the projects are developed within this framework, which provides a common vocabulary and common engineering processes, supports all the necessary QA activities, and may even allow to apply metrics for measuring and estimating project size.

However, the following requirements are not adequately met by this kind of QMS:

- **Non linear life cycles**: Newer methods often use an iterative, evolutionary or incremental life cycle. Projects adapting such a life cycle often escape the classical project control that is based on linear phases and on developing exactly one set of models per phase.
- **Maintenance projects**: Often QA regulations focus only on forward engineering projects and they are not adequate for the needs of other project types. However, in many companies, maintenance, enhancement, integration, reverse engineering and documentation projects make up the vast majority of the software development effort. Such projects are as complex, but they do not need the same phases. Depending on the concrete circumstances, the development process may vary greatly. Many QMS do not support the necessary adaptability. The inadequacy of the QMS may in this case impose a large overhead that is not justified by its benefit.
- **Evolution of software development methods**: Every development method must undergo constant improvements. The evolution of the methods is bottom up: new adaptations are tried out in pilot projects before they are propagated into the QMS. If the QMS does not provide a framework for pilot projects, improvements are either discouraged or pilot projects are run outside the QMS.
- **Introducing new development methods**: Many new methods (especially object-oriented methods) deny similarities to existing methods. By overloading existing terms with a new meaning and using new terms for old contents, they make the reuse of experience and invested training difficult. The transition to new methods is unnecessarily hindered because the differences and similarities between the methods are not explicit.
- **Coexistence of projects using different methods:** There does not yet exist a common standard concerning vocabulary and models. This hinders the communication between projects using different methods and causes time consuming controversies. Having different methods is a reality for many companies, imposed by old systems, by preserving existing models or by customer requirements. This will not change quickly, as the need for maintaining older yet already documented systems will increase. A QMS should enable the upper management or a specific QA group to carry out cost efficient project control, including the monitoring of the project progress and the supervision of how the methods and QA activities are instantiated, even in such an environment.
- **Cooperation with other companies**: The names of the software development phases may be enforced by the customer. Even worse, several methods and life cycles may coexist within one project because there are several companies involved. In both cases, a mapping between the phases and the different models to be developed must take place to prevent chaos. Unfortunately his mapping is often done intuitively and is not documented explicitly.

In order to address the requirements described above, we describe a QMS that is less rigid than the monolithic one method approaches. Trying to find the balance between regulating everything on a company wide level and having the total freedom for each project, this approach still ensures the benefits of project control, improvable engineering processes and common understanding of terms. The use of methods is defined and regulated on two levels: company wide and project specific. In the company wide QMS-manual a common vocabulary and understanding of the modelling process is provided by means of a model architecture frame and a kit of model architectures. More responsibility, but also more flexibility, is given to the individual projects by using a project specific model architecture.

Among similar approaches we would like to mention the IEEE Standard for Developing Software Life Cycle Processes [12], in which the concept of defining a mapping between the project specific development process and a reference model is a key element, too. The MA as presented below has many similarities to the entity process model described by Humphrey [11], with the difference that our focus is on having a minimal QMS in a multi method environment and not primarily on improving one specific software process model.

2 Basic elements of a model architecture

Model architectures are used to describe software development methods. Therefore, before we can define what we mean by a model architecture or a model architecture frame, we have to show what aspects of methods we will look at and what terms we will use to describe these aspects.

2.1 Methods

A method describes many different aspects of the software development process. The important aspects in the context of this paper are shown in figure 1. The static side of a method consists of all the results, i.e. the **models** that the method brings forth. These models include e.g. feasibility studies, enterprise models, analysis models, design models, architecture models and the code which is also called implementation model. Not all of these models are part of the final software product. Some are only of a transient nature, being only helpful at a certain stage of the development. As they are only interim results, they are discarded later on and are not maintained. In addition to the **transient models** there are, of course, the **product models**. They make up the final software product and need to be maintained during initial development as well as during maintenance. Clearly, they are only of any value if they are kept up to date with the evolution the software product undergoes.

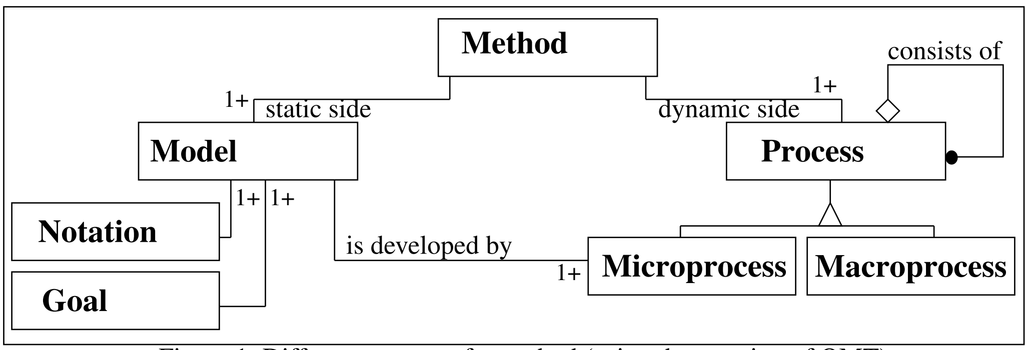

Figure 1: Different aspects of a method (using the notation of OMT).

A model is created by using a certain **notation** which can be graphical or textual. Every notation should have a well defined syntax (set of symbols and of rules about the composition of these symbols) and a well defined semantic (interpretation of the symbols). Many popular methods provide just the definition of a notation. However, the semantic of the notation limits the spectrum of information that can be shown in a certain model, but it does not define the intended content of a certain model. Therefore, the description of a model should be complemented by an explicit definition of the goal. The **goal** determines the potential content of a model and shows its intentions and the purpose that it fulfils for the overall development process. Goals define what view of the system and what way of thinking the models reflect. For an analysis model, the definition of the goals could include the following points [1,10]: Does the analysis model represent an environmental or internal view? Is it problem-oriented or target-oriented? Is it application specific or reusable? Does it reflect the actual or the future situation? What is its level of technology-independence?

The dynamic side of a method contains the **processes** that define how the results are achieved. The processes consist of all the activities that are necessary to transform the user's requirements into software, i.e. to develop the software product with all its models. The **macroprocesses** [5] describe the overall progressing and define the sequence in which the models of the product are developed, revised and extended. For each individual model, there are **microprocesses** that define the prerequisites and the steps that are necessary to achieve this model. There are, e.g., microprocesses for the initial creation of a model, for different uses of prototyping, verification and validation, and for maintenance. These microprocesses also include guidelines, rules and heuristics that apply to a certain model. We prefer the notion of microprocesses to the term activities, because microprocesses need not be elementary. Along with the models they belong to, they can be decomposed into finer grain microprocesses, depending on the project size and context. Whereas e.g. the testing of one code module may be considered as elementary in one project, in a larger context it may be decomposed into several microprocesses like code-reading, white-box tests by the programmer, testset creation, etc. Generally, in a given project, we will find an aggregation hierarchy of microprocesses and macroprocesses.

2.2 The model architecture

The **model architecture** (MA) is a description of all the models that are used within a certain development method. A MA contains the following information (see figure 2):

- names, notation and goals of all the models used
- static relationships (for traceability) and dynamic dependencies between models
- microprocesses for the development of individual models (not every microprocess needs to be described explicitly, as many of them follow straightforwardly from the definition of the goals and the notation)

Most books on software development methods do not explicitly describe the MA the method is based on, and several of the above points are not mentioned at all. Some method descriptions at least distinguish between the process and the notation, e.g. Booch [5]. A few give an explicit overview that also includes the definition of the goals of the different models, and Jacobson describes the notation of the models in OOSE [14] explicitly in terms of a MA, although his MA does not contain all of the information mentioned above.

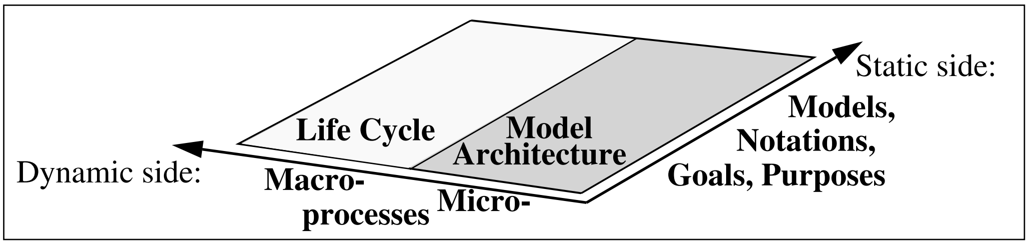

Figure 2: Elements of a method, represented by the model architecture
or by the project life cycle.

Not included in the MA are the macroprocesses. They are part of the **project life cycle** and are kept separate in order to give the MA the necessary flexibility that it needs to be used within the QMS. The MA does not imply any order in which the models need to be developed, changed or updated. For example, the models describing the technical requirements may be available already before any analysis modelling starts, or they may be produced parallel to the design processes. Depending on the special circumstances of a project, a first version of the user interface specification (a prototype) may be developed before or in parallel with any conceptional analysis models, in order to help the user in detecting and describing his business requirements, or because the prototype is needed for contractual purposes.

2.3 The model architecture frame

A **model architecture frame** (MAF) is a reference model for model architectures. It defines the goals and purposes of the different models, but neither their notation nor their microprocesses. The MAF is more general than the MA of a specific method, because it concentrates on the goals of the models. Yet the MAF is still specific with respect to the domain of the software projects. The definition of the models and their purposes may vary depending on what kind of software will be developed: In commercial software development the enterprise models and the technology free problem domain models play an essential role, whereas for an embedded realtime system supplementary design models must be included. The different domains have an influence on what models are mentioned in a MAF and how their purposes is defined. An example of a MAF will be given in chapter 4.

3 The use of model architectures in a QMS

To create an adaptable and flexible QMS we distinguish between the **QMS-manual**, containing the company wide QA regulations, and the **QA-plan**, which is specific to one project (see figure 3). The term company is used here as a generic term. It may refer to a whole company, but it may also be used just for a software development department.

3.1 Company wide QA regulations

The QMS-manual defines issues that are applicable to all the projects, e.g. the MAF. For other issues, like the definition of milestones, the QMS-manual gives guidelines how they are to be addressed in the project specific QA-plan, providing the necessary framework and terminology.

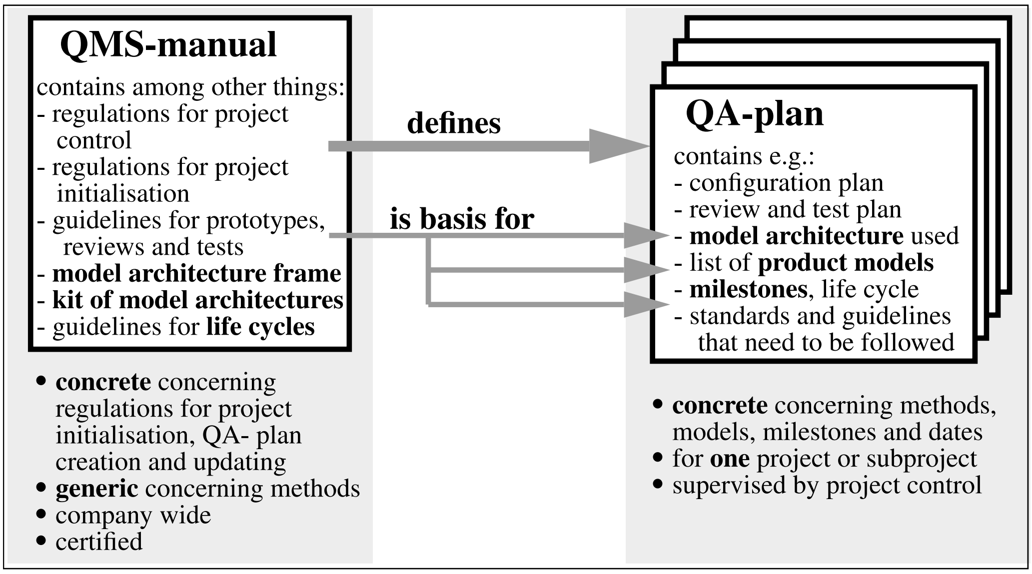

Figure 3: Relationship between the QMS-manual and the project specific QA-plan.

3.1.1 The model architecture frame

The MAF a company defines should be adapted to the domain of the company and to the vocabulary and the methods the company has used so far. The domain has much influence on the structure and content of the MAF. As the definition of goals and intents depends on the domain, the MAF should be formulated generic enough so that it can be used by all the projects. This generality requires independence of specific methods as well as independence of the type of project (maintenance, extension, reengineering, new system).

Investment in training methods and quality assurance must not be underestimated. If during years a certain vocabulary and a certain MA have been accepted and understood, this knowledge should not be discarded lightly, even if the MAs were never formulated explicitly. It is worthwhile to build on the present foundation and to adapt the MAF to the historically grown environment and vocabulary.

3.1.2 The kit of model architectures

The MAF provides the necessary reference model and defines a common vocabulary. But before it can be used as a common basis for individual projects, explicit MAs for the different methods are needed, and these MAs must be mapped onto the reference model. For the methods used most often in a company, it makes sense to provide an explicit MA and a mapping onto the MAF on a company wide basis. These mappings are collected in a **kit of MAs**. The kit may be complemented by suggestions concerning the mixing and combination of different methods and by guidelines concerning the metrics that can be used with certain methods. In contrast to the MAF that is compulsory for all projects, the kit is seen as a service of the QMS from which every project may take what seems suitable.

For many methods it is by no means easy to provide a mapping onto the MAF. This is due to the fact that often only the notation of the models is defined clearly. It is left to the user to determine which models are appropriate at what time, and to distinguish between transient and product models. The definition of the goals of the models is in this case often done implicitly in the projects. Even irrespective of a QMS it is worth while to make these decisions explicit, i.e. to interpret or enhance the method by the description of the precise goals and intentions of the different models, and to strive for a generally accepted set of product models. Most methods do not provide modelling techniques for all parts of a MAF (e.g. many object-oriented methods provide no models for the external specification of the user interface or for the software architecture and subsystem structure). When defining the project specific MA and its mapping to the MAF, such deficiencies become evident and can be taken care of.

3.1.3 The MAF as a basis for different project life cycle models

In forward engineering projects, the order in which the models are developed and revised varies from strictly sequential in the waterfall life cycles (e.g. [9]) to highly iterative in the spiral [4], recursive-parallel [3] or fountain life cycles [8]. For reengineering or maintenance projects, a middle-out approach may be more appropriate, where the model in which the essential changes take place is updated first, and then the consequences of these changes are cascaded to higher and lower level models. As more and more projects deal with existing software systems, alternative life cycles that suit a vast variety of different circumstances become more important. These different life cycles can be considered as special cases of a general iterative incremental life cycle.

In the iterative incremental life cycle [18] the system and its transient and product models are allotted to increments. An **increment** may consist of only one model of a certain part of the system (e.g. an analysis model of a part of the domain or a specification model of the user interface) or may be a complete deliverable release (e.g. the detailed design model and tested implementation of the messaging subsystem). All models undergo iterations, allowing to correct errors and to improve models based on new insight gained by developing other parts of the system. Prototyping may be part of the iterative incremental life cycle. It is a microprocess that helps to evolve and validate certain aspects of a product model. The prototypes that are created during these processes are transient models.

3.2 Project specific QA regulations

Software projects are just one special type of operational processes found in an enterprise and described by a QMS [6]. Like all other operational processes of the enterprise, the project is started by an explicit initialisation, where the QA-plan is initiated.

Of course not every detail concerning the development process is defined from the beginning. During project initialization, the project specific MA, the set of product models, the project life cycle and the major milestones are defined. Based on the initial decisions, the transient models and the details concerning increments and microprocesses are determined as the project proceeds, taking the minimization of risk into account (risk assessment). The deferred definition of the details and the repeated revision of the QA-plan are due to the reflexive and dynamic nature of software process modelling [21].

3.2.1 The project specific model architecture

Normally the MA of the project is derived from a method specific MA provided by the kit of MAs (figure 4). The MA is adjusted in the initialisation phase and at each milestone to the project specific needs. Possible adjustments include:

- Adding more models or exchanging models by mixing two methods
- Omitting some models, if they are not necessary in a particular project
- Increasing the number of product models

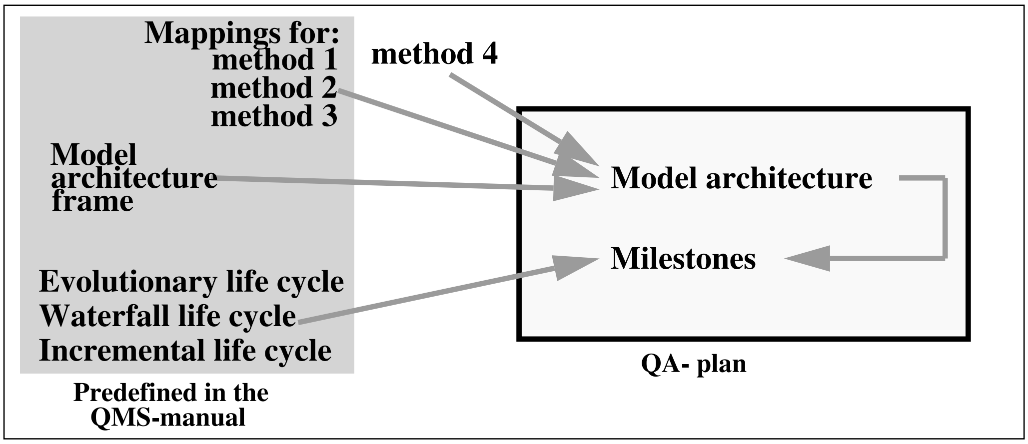

Figure 4: Definition of milestones.

Of course, if a new method is used that is not already described in the kit of MAs, the whole MA and its mapping onto the MAF must be defined from scratch.

3.2.2 Milestones

Milestones are used to define the project's structure in time. We avoid the term phase in this context, because it is commonly used to denote both processes and the project's progress in time, assuming that no overlapping, no incremental and no iterative development takes place. In contrast to phases, milestones define only the deadlines for the development of the increments, with the purpose to support project planning and to enable external project control.

Milestones are derived from the project specific MA and the project life cycle (figure 4), trying to meet the project specific requirements as good as possible. For each milestone the deliverables are specified. They concern contractual, financial and organisational aspects and, of course, the product and transient models of the software system. The deliverables of a milestone do not need to correspond exactly to one model of the MA. The mapping between MA and milestones is done by the increments. For each milestone, both quantitative and qualitative aspects of the increments, as well as their dependencies and preconditions, are described in the QA-plan. Models and subsystems to be included are listed, and the microprocesses that must be applied are specified, thus defining the required level of completeness and consistency for each increment. This allows e.g. to distinguish between a deliverable that is only a first draft and a deliverable that undergoes several iterations and is formally reviewed and validated by the user.

The distinction between models, processes and milestones, and the fact that the MA must be defined before the increments and milestones helps to avoid many sources of confusion in planning and controlling projects.

3.2.3 Project control

Project control is made possible by the explicit definition of the project specific MA and its mapping on the MAF, even in a multi method environment. The management supervises project initialisation, monitors the project progress at every milestone and reviews the QA-plan and its revisions regularly. Having a well defined vocabulary and specifying the purpose and contents of models eliminates many problems that ultimately might lead to a failure of project control. However, this approach requires more methodological insight from the reviewers and gives them more responsibility than in the case of a more rigid QMS.

There are distinct advantages of basing project control on a MA and not on predefined procedures: By leaving the definition of the project life cycle to the project, by having decisions concerning increments and microprocesses throughout the project's progress, and by allowing the project leaders and software developers to adapt a given MA or to specify a new one, enough flexibility is given to adjust the software development process to the project specific circumstances, to take professional practices into account [20], and to promote creativity and initiative, thereby increasing both motivation and software quality.

4 Examples

4.1 Example of a model architecture frame for information systems

Figure 5 gives an overview of the different model levels of a MAF for information systems. We cannot give here a complete description of the whole MAF, but limit ourselves to examples of the goals for two of the models contained in the MAF. A more detailed description of a similar MAF may be found in [2].

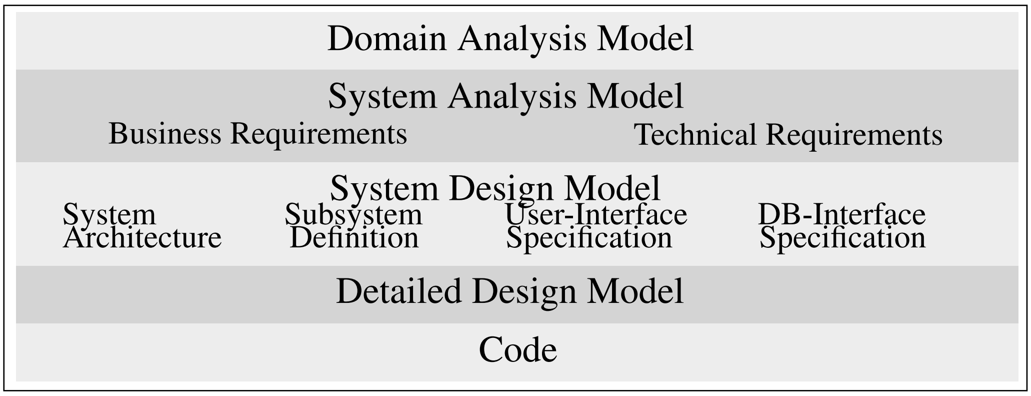

Figure 5: Overview of a model architecture frame.

The **domain analysis model** describes a logical model of the domain and gives the necessary basis to decide on the boundaries, requirements and scope of any system to be described by a system analysis model. Further goals of the domain analysis model are:

- To describe the universe of discourse in the language and in the way of thinking of the domain experts and users
- To identify concepts and models which span multiple applications in a given domain
- To show only the essential concepts, abstracting away details and making no difference between manual, software and hardware processes

The goals of the **business requirements** of the system analysis model are:

- To give a logical view of the future system
- To model the system from the user's point of view, in his way of thinking
- To show an external view of the software system on a conceptual and not on a technical level (the specification of the user interface is given in the system design model)
- To show the essential information the system deals with (without considering any technological constraints)
- To show the overall functionality that the system provides (the external

stimuli, the reactions of the system and their conditions, the information flows between the system and the user)

4.2 Overview of some model architectures

Of course, within the context of this paper, it is not possible to present complete examples of MAs with the description of all the different models, their notations, microprocesses and interconnections. Therefore, we have restricted ourselves to show just an overview of different mappings in figure 6. Further, for a given method, different mappings are conceivable, depending on how the method is interpreted and complemented in a particular company.

As an example, we describe the mapping of the system analysis and system design levels for OOSE. The goal of the requirements model of OOSE is to capture the functional requirements of the software system from the user's perspec-

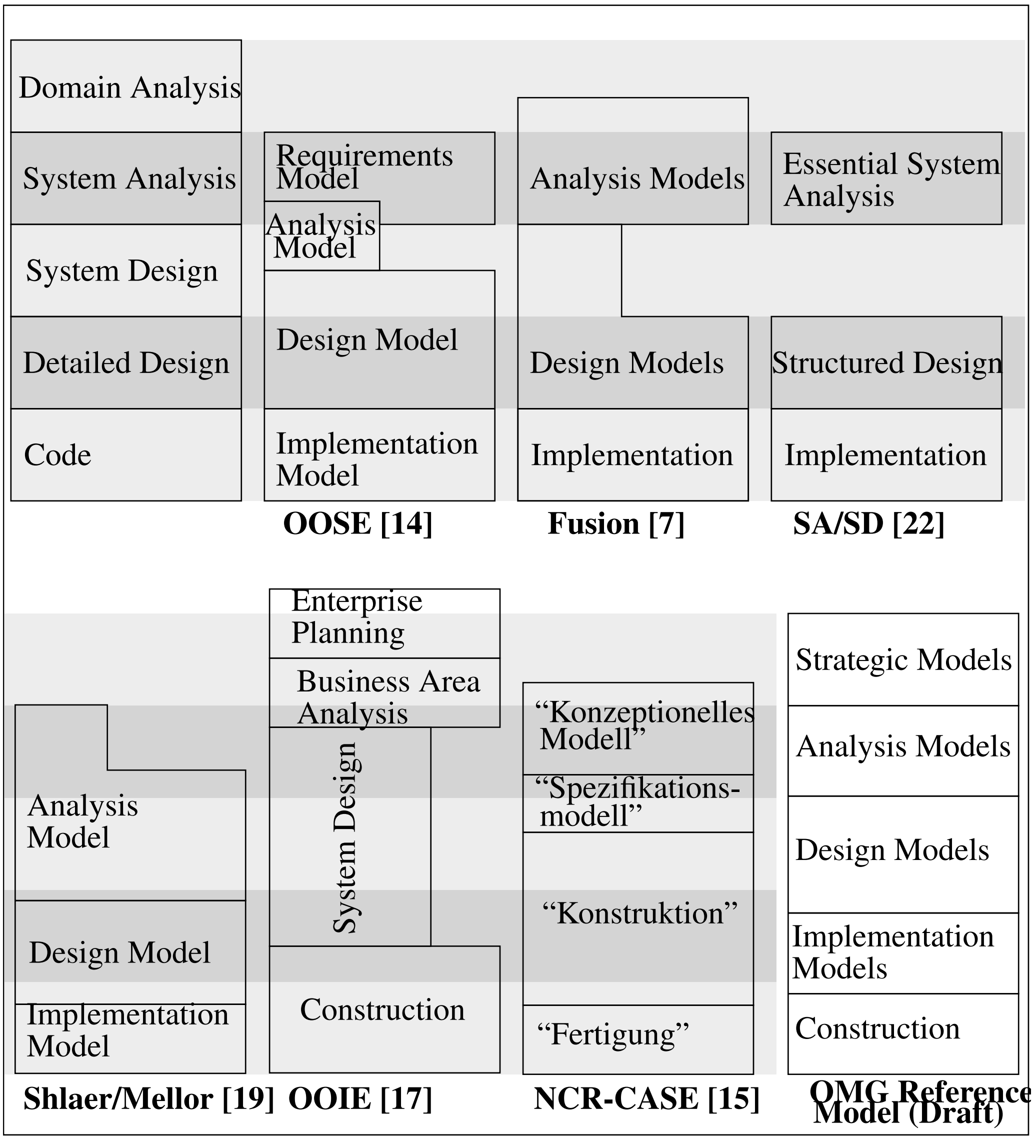

Figure 6: Comparison of different mappings.
The grey bars show the model levels of the MAF. From the MAs of the individual methods only the model levels are shown. The OMG-reference model [16] is given only for comparison.

tive and to show how the user will use the system. The analysis model represents the software system in a robust and changeable object structure, assuming an ideal implementation environment.

Thus for the system analysis level the mapping could be the following (all models labelled with (P) belong to the minimal set of product models):

- Use-case model of OOSE (P)
- Domain object model of the system according to OOSE (P)
- List of all essential technical requirements

The system design level might then contain the following models:

- System architecture: Domain charts according to Shlaer/Mellor (P) and subsystem interactions with interaction diagrams of OOSE (P)
- Logical object model of each subsystem (analysis model of OOSE)
- User interface specification: textual, state charts (P) and prototype
- Interface to relational database: logical view of database as ERD (P)

5 Conclusions

Developers and project leaders frequently complain that their QMS limits their flexibility and creativity. While protecting against disaster, it makes adapting to circumstances difficult. Often the QMS does not address the needs of companies who have to live with a variety of methods, who adopt new development life cycles or work on maintenance projects. In this article we have presented an approach which we hope provides the necessary flexibility for heterogeneous projects by basing the QMS not on a single software development method, but on a more general MAF that supports several methods within a well defined framework.

References

1. Bailer, B. *Geschäftsmodelle, Verständnis und Folgen der Technologieneutralität*, Bericht des Instituts für Informatik der Universität Zürich, 1993
2. Beringer, D. *Limits of Seamlessness in Object Oriented Software Development*, Technical Report, Swiss Federal Institute of Technology, Department of Computer Science, Lausanne 1993
3. Berard, E.V. *Essays on Object-Oriented Software Engineering, Volume 1*, 1993
4. Boehm, B.W. *A spiral model of software development and enhancement*, IEEE Computer, May 1988
5. Booch, G. *Object-Oriented Analysis and Design with Applications* Second Edition, Benjamin/Cummings, 1994
6. Brantschen, S. & Zeder, S. *"Wir haben kein Q-System - wir sind eines!"*, Informatik, Zeitschrift der schweizerischen Informatikorganisationen, Nr. 6, Dez. 1994
7. Coleman, D. et al *Object-Oriented Development: The Fusion Method*, Prentice Hall, 1994
8. Henderson-Sellers, B. & Edwards, J.M. *The O-O-O methodology for the object-oriented life cycle*, ACM SIGSOFT, Software Engineering Notes, Vol 18, No 4, October 1993
9. *HERMES - Handbuch der Elektronischen Rechenzentren des Bundes*, eine Methode für die Entwicklung von Systemen, Bundesamt für Organisation, Bern,1980
10. Hoydalsvik, G.M. & Sindre, G. *On the purpose of Object-Oriented Analysis*, Proceedings of OOPSLA 1993, Conference on Object-Oriented Programming Systems, Languages, and Applications, Washington DC, USA, 1993
11. Humphrey, W.S. *Managing the Software Process*, Addison-Wesley, 1989

12. *IEEE Standard for Developing Software Life Cycle Processes*, IEEE Std 1074-1991, IEEE Computer Society, 1992

13. *Quality management and quality assurance standards - Part3: Guidelines for the application of ISO 9001 to the development, supply and maintenance of software*, ISO 9000-3, 1991

14. Jacobson, I. et al *Object-Oriented Software Engineering - A Use Case Driven Approach*, Addison-Wesley, 1992

15. *NCR-CASE, Entwicklungshandbuch*, NCR Schweiz, 1991

16. *Object Analysis and Design, Volume 1: Reference model*, Draft 7.0, Object Analysis and Design Special Interest Group, Object Management Group, Boston USA, 1992

17. Odell, J. & Murhpy, K. *OOIE: Object-Oriented Information Engineering*, Object Analysis and Design, Description of Methods, edited by A. T. F. Hutt, OMG, 1994

18. Rubin, K.S. *Succeeding with Objects*, Tutorial at Object-Technology at Work, Joint Conference between Choose and DBTA, Zuerich, 1994

19. Shlaer, S. & Mellor, S.J. *Object Lifecycles - Modeling the World in States*, Prentice Hall, 1992

20. Sommerville, I. et al *Supporting Informality in the Software Process* and *Process Modelling and Development Practice*, Proceedings of the Third European Workshop on Software Process Technology, Villard de Lance, France, 1994

21. Starke, G. *Urgent Research Issues in Software Process Engineering*, ACM SIGSOFT, Software Engineering Notes, Vol 18, No 4, October 1993

22. Yourdan, *E.N. Modern Structured Analysis*, Yourdan Press, 1989