The Goals of the Analysis Model

Dorothea Beringer
Laboratoire du Génie Logiciel
LGL-DI-EPFL
CH - 1015 Lausanne
d.i.beringer@ieee.org

December 1996

Abstract

When developing an analysis model, the notation itself does not suffice for determining the content of the analysis model and the end of the analysis process. For this, the goals and intents of the analysis model are used, either implicitly or explicitly. In this report we take a look at various possibilities to define the term analysis and the goals of the analysis model. We start with considering the reference model of the problem solving cycle. We examine the definitions that refer to analysis as the "what" in opposite to the "how", or as the problem definition phase of a problem solving cycle. We then continue with a discussion of various criteria (like technology independence, real word model, unambiguous or essential model) that are often used to specify the goals of an analysis model. Difficulties that arise with these criteria are that they cannot be combined arbitrarily and that some of them are very vague; we propose therefore also some more concrete formulations. In the last part of the report we take a closer look at some misconceptions and intent clashes that are due to a too idealistic model of the software development process. Objective real world models, analysis models that are stable after the analysis phase and analysis models that are equal to the high-level view of the final software system are often attempted, but are normally not realistic. This leads to intent clashes with the desire for a seamless and easy controllable development process. We explain these intent clashes and we also describe the three different approaches taken by software development methods to cope with these intent clashes.

1 Introduction

When developing a model, we hopefully pursue a certain objective which gives us the motivation to undertake that task. Such a motivation may be to give a brief overview for managers, to evaluate the feasibility of a software system, to set up a contract for a specific software system for outsourcing, to provide the necessary requirements specifications to be used by sophisticated developers, to fulfil somehow the notational requirements of certain QA-guidelines, or to try out a new notation.

Depending on the motivation behind the modelling process, we then choose a notation and define the intent or goal of our model (we will use the terms "intent" and "goal" interchangeably). Only that information appears in the content of the model that is describable by the notation and corresponds to the intent of the model (see figure 1).

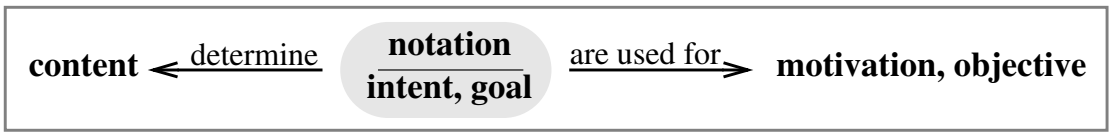


Figure 1: Objective, content and intent of a model

A certain notation can be combined with many different modelling intents. E.g. an ERD can be used to show the logical model of all the tables found in a relational database. It can also be used to model all the data in a certain problem domain. Or it can be used to give a static view of all the objects in an object-oriented system, considering only attributes. Even if the notation is exactly the same, the content of the model, even if it is about the same system, may vary heavily.

Considering intents and goals of models is important for several reasons:

- Most modelling notations can be used in several contexts. Yet many methods do not state explicitly for which modelling intents their notations can or should be used.
- When comparing different methods, e.g. concerning their techniques for modelling the global behaviour, the differences in the goals of the models are at least as important as the differences in the notations used. For choosing a certain method, the achievable goals should even be of higher priority than notational issues.
- Terms like analysis are heavily overloaded. Without making it more precise what the intent of a specific analysis model is, misunderstandings, endless discussions and wasted modelling efforts are inevitable. Stating only that an analysis model should be developed, is not enough.
- Managers, end-users, system engineers and software engineers have different motives for making an analysis model.
- To judge the quality of an analysis model, we do not only want to determine if the notation is used correctly but also if the right content is represented. But appropriate quality criteria depend on the goals of a specific model.
- The definition of the goals of the different models is also important when defining project management guidelines and project control. In a multi-method environment the intents of the models are even the only bases on which a general model architecture can be defined (for more details see [Beringer95]).
- Discussions concerning the seamlessness of object-oriented methods cannot only be based on notational issues. They must take into account the limitations caused by the

differing goals of models on different abstraction levels (for more details see [Beringer94]).

In the following we will discuss possible goals of analysis models. But first, we take a closer look at the term "analysis", and we use for this the reference model of the problem solving cycle.

2 The term "analysis" from the perspective of the problem solving cycle reference model

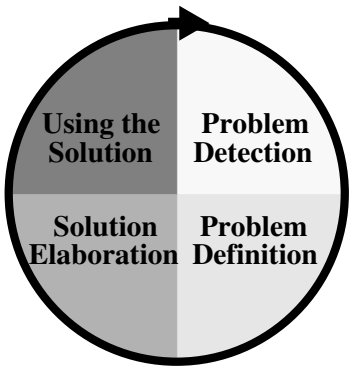
Many methods use the term analysis for the first activities that are undertaken in order to solve a given problem. We therefore describe here the reference model of the problem solving cycle and examine how analysis could be defined, based on this model.

2.1 The problem solving cycle

A software development project tries to solve a certain problem by providing a computerized solution. Yet problem solving is not restricted to software engineering, it happens every day. The problem solving cycle is a reference model that describes the process of solving a problem in a very general way. It has also been used in the software development guidelines of NCR Switzerland and in the QA-guidelines of the GfAI Switzerland.

The four phases of the cycle

The four phases of the problem solving cycle are described in figure 2.



- **Problem Detection**: It is recognized that there is a problem and certain measures are taken. What the problem really is about, is often not yet clear.
- **Problem Definition**: The problem gets analysed and defined. Priorities are set and the compulsory and the nice-to-have objectives are defined. It is determined ***what*** problem needs to be solved.
- **Solution Elaboration**: One or several solutions are evolved, analysed, decided upon and put into practice. If several solutions are considered, these must be checked against the objectives, measured and selected. It is determined ***how*** the problem is solved.
- **Using the Solution**: The solution is in operation. It has to proof that it can yield the promised benefits. If new requirements or defects appear, a new cycle is started.

Figure 2: The problem solving cycle

Omitting the definition of the problem

The different activities during the four phases of the cycle vary heavily depending on the domain we are looking at. In politics, the elaboration of the solution may denote the proposing, discussing and voting on new laws, in engineering this may involve building and trying out prototypes. Yet common to all the different domains is the fact that either the problem is defined *explicitly* before or in parallel to the elaboration of the solution, or this is done *implicitly* during

the elaboration or even using of the solution. Omitting the definition of the problem is not possible. It can only be deferred. The objectives can be hidden behind the solution. But this normally causes unnecessary controversies and additional costs.

Recursive problem solving cycles

At any point in the cycle, new problems can be detected that give rise to starting a new problem solving cycle within the existing cycle. In fact any individual activity in the original cycle can be seen as another smaller cycle. This results in a recursive structure of arbitrary many problem solving cycles.

Iterations and backtracking within the cycle

During the problem definition phase it may turn out that the wrong problem has been identified. Or during the solution elaboration phase the problem definition may appear to be ambiguous or to be unsolvable due to overconstricted objectives. Or when putting a solution into operation, some defects may be detected which necessitate some further elaboration of the solution. In all these cases, we have to go back and iteratively redo the work of previous phases. Of course, it is often a matter of definition if we consider such a backtracking as going back within the same problem solving cycle or as a new recursive problem solving cycle.

2.2 Applying the problem solving cycle to software engineering

Any work done in the realm of software development can also be seen from the viewpoint of the problem solving cycle. We may apply the reference model of the problem solving cycle to planning a new system, to maintenance work, to implementing a reusable class library, or to developing a new software application. Often the life-cycle of a whole software system is looked at as one single problem solving cycle. The development project starts due to a problem detection and ends when the solution is effectively used. The tasks of such a project can be subdivided in those that serve at defining the problem and those that serve at elaborating the solution. They are carried out basically in a sequential order, yet always include some amount of backtracking and iterations.

The recursive nature of the development process

Yet applying the problem solving cycle for the whole life-cycle of a project or software system is only one out of many possibilities. In fact each activity within the project and each subsystem or component corresponds again to a problem solving cycle. Due to its recursive nature, the problem solving cycle can be applied at any level of granularity. Restricting its application to the life-cycle of the project as a whole is an arbitrary decision. This becomes especially evident, when considering more complex software developments where several project teams work at several applications which together make up some large system, composed of organisational, hardware and software elements. From the overall system view, many details that for the individual teams belong into the realm of problem definition are considered as implementation details. Thus exactly the same activities and models may in one project be considered as belonging to the problem definition of a problem A, and in another project as belonging to the elaboration of the solution of a problem B, without ever making it explicit that problem A could be considered as a recursive problem solving cycle within the solution elaboration of problem B. An example for such an approach is found in [Berard93]. Berard proposes a recursive/parallel life-cycle for software development with recursive cycles of problem definition and prob-

lem solution. Exactly the same methods, techniques and phases are used on several levels of decomposition, taking into account the recursive nature of the problem solving cycle.

"What" versus "How"

The problem definition defines *what* the problem is about. The elaboration of the solution shows *how* the problem is solved. When applying this distinction to a software development project, we can say that in a first phase (or in the first phases) the *what* of the system under consideration needs to be defined, and in a second phase the *how*. But the *"what - how"* rule does not really help us to distinguish these two phases or their corresponding models, because often it is not clear if a certain statement describes the "what" or the "how" of the system. This is due to the following reasons:

- The problem solving reference model only tells us that theoretically such a differentiation between statements belonging to the problem definition and belonging to the solution of a specific problem can be made. It cannot give us any guidelines as how to make the differentiations. Such guidelines are not possible, because only by deciding in a particular case that a statement defines the *what* and not the *how* we know what the problem is, not before.
- Davis describes the "*what versus how*" dilemma with "one person's what is another person's how" [Davis93]. For certain persons the exact look of the user interface is the *how*, for others it belongs to the *what*. What it really is depends on the concrete circumstances and on the perspective and expectations of the persons involved. Where we set the borderline between defining the problem and finding a solution depends on what we perceive as being the problem. This again depends on our role in the project (manager, user, network specialist, programmer) and our personal opinion.
- Due to the recursive nature of the problem solving cycle, the *how* of an encapsulated recursive cycle refines the *what* of its outer cycle. Also the *what* of an inner cycle refines the *how* of the outer cycle. Thus a certain statement is at the same time the *what* of one problem and the *how* of another problem.

This dilemma becomes also obvious when considering the criteria that are given by a method to determine if the model of the problem is finished or not. These criteria are often very vague and subjective, and they do not really help to find an objective judgement on the completion of the problem definition.

Limits of the reference model

Though the problem solving cycle is a very valuable reference model for software development and though it is mentioned in one or the other form in many development methods, its significance for practical engineering projects has severe limits. First of all, applying the reference model only to the level of a whole project contradicts its recursive nature. Secondly, the reference model only makes the statement that there exist such problem solving cycles and that these can be decomposed into four phases.

The reference model helps to analyse the software development process and its different activities from a retrospective view, providing one possible perspective of software development (see also [Floyd89]). It enhances our understanding of what is going on in the development project but it does not determine what in a concrete situation should be considered as problem detection, problem definition or problem solution. Thus it cannot serve as a guideline for determining in a project if something belongs to the model of the problem detection, problem definition or to the model of another phase.

2.3 Where does the term "analysis" fit into the problem solving cycle?

Considering the problem solving cycle, there are different elements that we could call "analysis". And in fact, there are several approaches of what activities and results are called analysis.

Analysis defined as modelling the "what"

Often it is said that the analysis model defines the problem and describes the "what", in contrast to the design model which describes the solution, the "how". Examples:

- OMT: "Analysis is understanding a problem; design is devising a strategy to solve the problem..." [Rumbaugh94a].
- [Booch94, page 252]: "The purpose of the analysis is to provide a model of the system's behaviour. We must emphasize that analysis focuses upon behaviour, not form. ...analysis must yield a statement of what the system does, not how it does it."
- [Walden95, page 122] mentions that the current consensus on the meaning of object-oriented analysis seems to be "creating an object-oriented model from the external requirements (a model of the problem)".
- [Shlaer88, page 93]: "The first thing to do is to build the information model to define the conceptual units you will be working with. In doing this it is important to stay focused on the real world - the problem - and not the solution to that problem, which will be supplied with the automated computer system."
- In [Graham93] analysis is characterized as "identifying large scale structure, just enough to describe the *what* of the system, rather than the detail of the *how*."
- [Rawsthorne95]: "...analytic models - a set of models describing *what* the system will do; design models - a set of models describing *how* the system will be implemented...".
- In [Wilkinson95] analysis and design are differentiated in the following way: "Analysis, or problem modelling, in which the problem is described and represented... Design, or solution modelling, in which a solution to the problem is discovered and represented...". In the design model "additional classes and mechanisms, which support *how* the system works, are built upon the existing classes, which describe *what* the system does".
- Davis describes analysis as "understanding the problem and the user's needs, identifying all possible constraints on a solution; organizing the plethora of assembled information". This in contrast to the requirements specification: "describing the expected behaviour of the product to be built to solve the now understood problem". As synonyms to analysis he uses requirements analysis and problem analysis. [Davis90, Eckert93].

Considering what we have said in the previous section, it becomes evident that these definitions help us to categorize an analysis model in a retrospective view of a project, i.e. when the problem is already defined and we know what belongs to its definition and what to its solution. Yet they cannot provide any precise and objective criteria which could help during the modelling process of analysis to specify the goals of the analysis process and to determine the content of the model[1]. For this, other criteria are necessary.

1. The usefulness of the "what - how" distinction for defining requirements engineering or analysis is quite often questioned, e.g. also in [Siddiqi94], where Siddiqi examines the "universal truth" that requirements describe the "what" of a system and not its "how".

Analysis in its common meaning

If we consider the meaning of analysis as it is used in daily life and in most other disciplines (e.g. as defined by [Webster]: "an examination of a complex, its elements, and their relations"), then analysis activities can be found in any of the different phases of the problem solving cycle. Analytical examinations are necessary to define the problem, but they are also necessary to create a solution or evaluate a solution. Every modelling activity involves the separation of a complex into its elements and relations. This usage of the term "analysis" is referred to by [Walden95, page 122] as being the traditional sense of analysis, prior to its reinterpretation which is specific to current software development methods. Examples for this interpretation of the term analysis are:

- Wirfs-Brock uses the term analysis in this sense when dividing the design process into an exploratory phase and an analysis phase[2] [Wirfs90].
- The term analysis is also used for domain analysis in the sense[3] of taking a closer look at a specific problem domain, business process or enterprise. The resulting domain analysis model has no connection to the future software system, though domain analysis may of course precede a system development effort. Examples are the domain analysis in [D'Souza93] and [Beringer94], the enterprise modelling in [Jacobson92] or the strategic modelling in [OMG92b].
- [Jacobson92] differentiates between the process of requirements analysis which has as input a requirements specification and produces a requirements model, and the process of robustness analysis which investigates this model concerning its robustness and produces the analysis model (and not a robustness model as could be expected). So in OOSE the term analysis is used in two different meanings.

Any modelling activity includes analysis in this general sense. Therefore this original and common definition of analysis does not provide any basis for defining the content and intent of an analysis model in the software development process.

Analysis as being equal to specifying requirements

Another possibility is to use analysis as synonym for requirements specification. In the problem solving cycle we specify requirements during the problem definition as well as during the evaluation and selection of a solution. So this definition would cover the second phase and part of the third phase of the problem solving cycle.

In the realm of software development, not all specification issues are decided upon by the same people. For certain decisions the customer takes responsibility, for other decisions the developers have their full freedom, as long as no requirements constraints of the customer are violated. We could therefore say that analysis only contains all those requirements specifications which are under the responsibility of the customer (this can then be seen from retrospective as the problem definition phase). Such a definition of the term analysis is only possible if in the team where the term is used it is clear, which details of the requirements must be defined by the

2. The analysis phase denotes the refinement of the objects into subsystems and subtyping hierarchies and the specification of the method signatures. It has nothing to do with requirements definition or problem analysis.
3. Other usages of the term "domain analysis" are:
   - Finding the problem domain objects for the analysis model of a software system. E.g. in OOSE the requirements model consists of a problem domain object model and a use case model [Jacobson92].
   - Defining concepts that are valid beyond one specific software application in order to develop reusable software components for one specific application domain. [Neighbors80, Berard93] state: "System analysis states what is done for a specific problem in a domain while domain analysis states what can be done in a range of problems in a domain..."

user and which ones are decided autonomously by the developers. So the content of the analysis model becomes dependant on the organisational context, on the specific project management approach and on contractual conditions. And whenever a participative approach to software development is chosen (as e.g. described by [Reisin90]), this definition of analysis does not work at all.

2.4 Consequences

All these possible definition of the term "analysis" do no provide the necessary criteria to determine the content of the analysis model. Also the reference model of the problem solving cycle only defines that such a model exists, and gives reasons for it. But due to the retrospective view at the analysis process, it is of limited help as a guideline for the analysis model.

3 Criteria for defining the goals of analysis models

When looking at the literature, various criteria for the goals of the analysis model are mentioned that should support the developer in determining the content of the analysis model and the end of the analysis process. Figure 3 gives an overview on those criteria that go beyond the simple "what-how" we discussed in the previous chapter. Some of the criteria we present in the following sections are very vague, so we propose more precise definitions where possible. Moreover, some criteria exclude other criteria. We mention some of these contradictions.

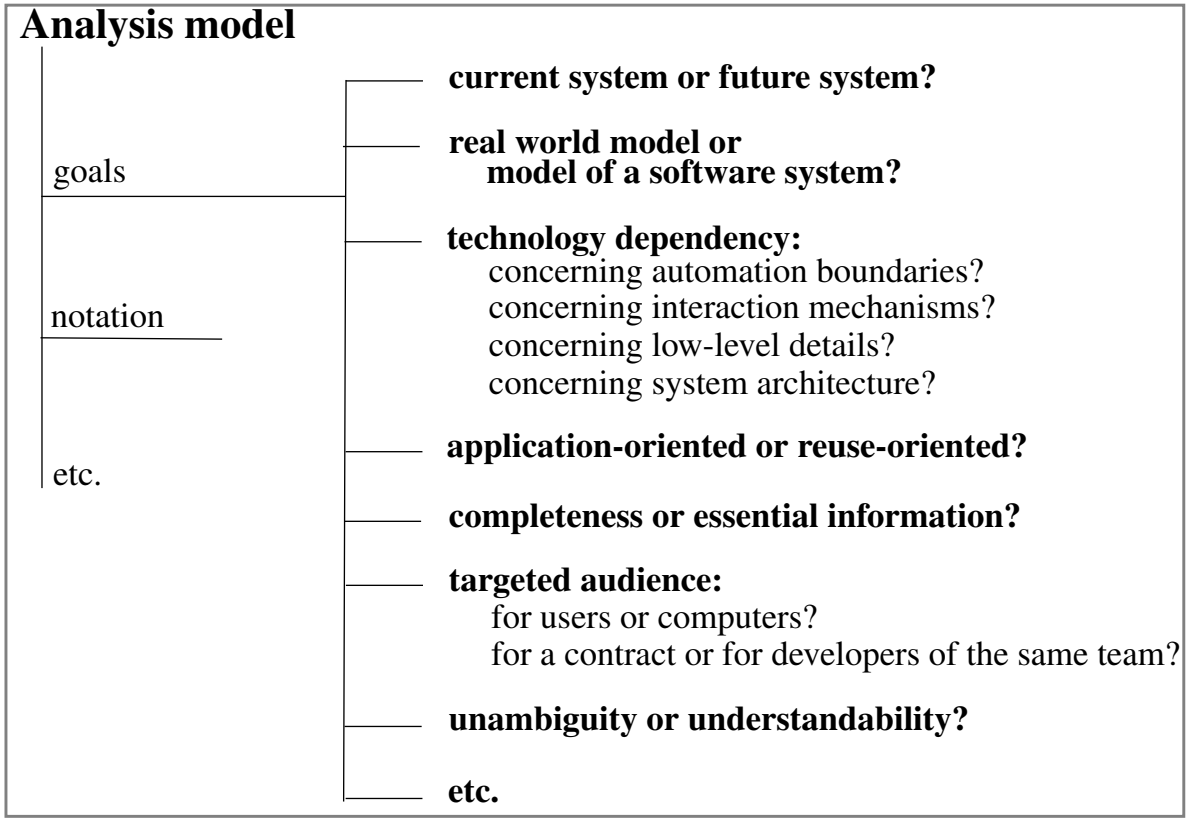


Figure 3: Goals of analysis models

3.1 Model of the current system versus model of the future system

Introduced by DeMarco in 1979 [DeMarco79], the differentiation between the model of the current system and the model of the future system has not lost any importance, though only in rare cases both models are made. Yet it is important to state explicitly which model is the goal, especially when starting by collecting information on the current system in order to model a future system. Even if the objective is only the replacement of an old system or the mere automatisation of manual processes, a new system has the potential of changing substantially current organizations and decision making processes (a list of such changes can be found e.g. in [Huber90]). These differences to the status quo need to be taken into account in an analysis model that should show the requirements or the high-level view of the new system.

3.2 Application-oriented versus reuse-oriented

Methods like Fusion [Coleman94], OMT [Rumbaugh91] or OOSE [Jacobson92] focus on the development of one single application. Their analysis models are thus application specific and have the goal to give a small, accurate, and easily understandable description for one application. If the goal of analysing a problem domain is not a specific application but a library of reusable components or an application framework, then the analysis model should show reusable objects and reusable interaction mechanisms. As is shown in [Berard93][4], application-oriented models differ in their structure from reuse-oriented models.

3.3 "Real world" model or abstracting a software system

An analysis model that reflects the real world respectively models the problem as the user perceives it (also called real world model[5]) is not identical to a model that abstracts a software system (also called target- or solution-oriented model). Differences are e.g. what information is modelled, how data and operations are grouped into objects, how these objects interact with each other, and also how external objects interrelate with the objects within the system. A model that tries to be both, a real world and a target-oriented model, is with the exception of trivial systems either the one or the other. As it is mentioned in [McGinnes92], though many developers pretend to make a model of the reality or of the user's view, often reality is bent to fit into an object model which fulfils quality criteria coming from the software design. Therefore [Høydalsvik93, McGinnes92] emphasize the necessity of a clear distinction. They advocate a model that reflects reality as the user perceives it and is not influenced by any knowledge about what would be good software abstractions. Other methods start right away with a model of the problem domain that uses the abstractions of the software system. Such a model focuses on the targeted software system, and finding good software abstractions is more important than reflecting reality as the user would perceive it before he gets acquainted with the future software system.

4. Reuse-oriented models use inheritance heavily, factor out associations and complex operations into classes of their own, parameterize classes, and provide classes with a complete set of primitive methods. For this, a more complex model must be accepted. In contrast, an application-oriented model describes the characteristics and requirements of exactly one application, and is as small and simple as possible. The focus is on the application specific relationships between perhaps reused objects. The classes only need a sufficient set of methods but may include also complex operations.
5. In [Høydalsvik93] this model is called problem-oriented model. The model describes a certain organisation, business, process or machine without considering how this problem will be reflected in the structure and elements of the software system.

3.4 Technology independent models

Technology independence

Many methods mention that the analysis model should be independent of the design and the implementation of the system, i.e. that technological details should be abstracted away. Formulations like "describing the *what* instead of the *how*" or "modelling problem-oriented instead of target-oriented" also imply a technology independent model. Examples:

- DeMarco suggest to develop logical models of the current and of the future systems which do not contain any physical information [DeMarco79, page 233].
- OMT gives the following guidelines for a technology independent analysis model: "A good analysis captures the essential features of the problem without introducing implementation artifacts that prematurely restrict design decisions." [Rumbaugh91, page 187]. "The successful analysis model states what must be done, without restricting how it is done, and avoids implementation decisions. The result of analysis should be understanding the problem as a preparation for design" [Rumbaugh91, page 148].
- Fusion: "Analysis is about describing what a system does rather than how it does it. Separating the behaviour of a system from the way it is implemented requires viewing the system from the user's perspective rather than that of the machine." [Coleman94, appendix A].
- The analysis model in MSA assumes perfect technology (no errors, no space or time constraints, no costs). Also no differentiation is made between human and automated processes [Yourdan89].
- OOSE: "In analysis an application-oriented specification is developed to specify what the system offers its users. This specification, which we call the analysis model, specifies the functional behaviour of the system under practically ideal circumstances and without regard to a particular implementation environment... It is important, however, to judge whether the analysis model can actually be realized..."[Jacobson92, page 15].
- Sommerville summarises the content of the requirements specification with the following words: "A software requirements specification is a document containing a complete description of what the software will do, independent of implementation details" [Kotonya96].

Many methods do not specify further what exactly they mean with technology independence and from which technologies they want to abstract in the analysis model. For this reason, [Bailer93] differentiates in the context of information systems between several types of technology from which we can abstract in an analysis model:

- *Information or implementation technology*: The system is modelled independently of its future implementation as software or manual system. The system itself with its processors, data-stores and communication paths is considered as perfect.
- *Organisation technology*: The system is modelled independently of the organisational structures found in the present organisation; the organisational environment is considered as being of perfect technology.
- *Production technology*: The system is modelled independently of what it produces. This abstraction does not make any sense in the realm of software engineering, because any software system supports a specific production technology and cannot be modelled without this.

Abstracting away implementation as well as organisation technology brings the great advantage that the whole organisation where a software system should be embedded can be rethought and redesigned, and hopefully also optimized, thus combining system development with business re-engineering. It seems ideal to make an analysis model that abstracts away any technological detail concerning implementation technology and organisation. Yet in practical projects we encounter the following difficulties:

- Even if a new software system is the initiator to redesign the organisation, it is often not efficient to mix the goals of business re-engineering and system requirements modelling; the essential details are too different.
- The metaphor of perfect implementation technology may be useful for batch like information systems. Yet when developing networking software or new user interfaces, the system under consideration would not exist when we assume perfect technology with no automation boundaries. In most projects, the goal is not a model that abstracts away all implementation technology, but abstracts away only certain aspects of the implementation technology.

In practice there is therefore a great amount of uncertainty how to develop and use technology free models in an efficient way. As [Bailer93] mentions, in industry the technology free models are often not made and even many examples found in books are not really technology free. But when we want to describe a model that abstract away only certain technical aspects, we need a further distinction. We propose the following criteria: abstracting the automation boundaries, abstracting the interface and interaction mechanism, abstracting the system architecture, and abstracting low level implementation details.

Abstracting away low level implementation details

When in 1979 DeMarco introduced the differentiation between the physical and the logical model [DeMarco79], it was not yet common to abstract away physical details like file structures, algorithms and help variables. Nowadays, abstracting away low level implementation details in an analysis model, and to some degree even in the design model, is taken for granted by object-oriented development methods. The encapsulation of data and operations into objects, the principle of information hiding and the possibility to define class interfaces by contracts help to create analysis and design models that are not programming language specific and that even do not contain yet all objects necessary for the implementation.

Abstracting away the system architecture

Many methods advocate an analysis model that does not consider the internal system architecture. The internal system architecture is either defined in a separate model after the analysis model (as e.g. in SSP [Jufer92] or OMT [Rumbaugh91]) or it is not dealt with by the method at all (e.g. Fusion [Coleman94]). As long as a method is used for monolithic and simple systems, abstracting away the internal system architecture may appear to be very obvious, a special high-level model that takes into account the system architecture but no other implementation details may be superfluous. For more complex software systems which may include distribution, off-the-shelf software components (like networking software, user interface systems or database systems), or which integrate also hardware components, we certainly need also an analysis model concerning the system architecture, thus having two analysis models with different goals. But not every project starts with an analysis model that abstracts away the system architecture for the following reasons:

- Introducing distribution or off-the-shelf components can change the structure of an object-oriented analysis model totally, causing major remodelling when introducing the system architecture.
- For certain systems all the complexity lies in the system architecture, an analysis model that abstracts this away would be trivial.
- In certain projects the system architecture is already given at the beginning and is not subject to change.

So depending on the concrete circumstances we need to define explicitly for each analysis model if the system architecture is abstracted away or not and which elements of the technical system architecture are taken into account and which ones not.

Abstracting away automation boundaries

To fix the boundaries of the future software system right at the beginning of a project is not only difficult, it can also be very dangerous. In order that the new system really suits the needs of the users, the system boundaries and the organisational aspects of the environment need to be carefully examined and eventually redesigned. An abstraction from all implementation technology is needed. Therefore many methods advocate a universe of discourse larger than the automated system. In MSA [Yourdan89] the automation boundaries are even abstracted away for the whole analysis model (also called essential model), they are introduced during the transition to the design. [Shlaer88] distinguishes between the analysis model, which does not specify which processes will be automated, and the external specification model which focuses on the interactions on the automation boundary. [Cook94] has two analysis models, the essential model which abstracts away any automation boundaries and the specification model which takes them into account. In OOSE and Fusion a problem domain object model is made that is free from the automation boundaries, yet the use case respectively the interface models are made after the automation boundaries have been introduced into the analysis respectively the system object model.

Other methods, especially methods that do not model explicitly the global system behaviour, leave it open to which degree the automation boundaries are abstracted away in the analysis model, often assuming that introducing the system boundaries into the object-oriented object model will not change the structure of the object model. This may be true for certain projects but is not the case in general, as is shown in [Eckert95] for an example using the Fusion method and in [Høydalsvik93] for a model of the OOPSLA conference registration problem.

Abstracting away the low-level interaction mechanisms on the system boundary

When we consider those models that show the interactions between the software system and its environment (scenario models that show the external view of the global behaviour of the system), two questions arise: What does it mean if we state as goal for our model to be technology independent? Which details of the interface can be abstracted away?

As we model the interactions on the system boundary (external events, system services, use cases or scenarios), technology independence certainly does not mean abstracting away the system boundaries. The system boundaries of the software system must be known, but we can decide on which level we want to model the interactions. We can choose a technical level (single user inputs), or we can choose a purely conceptual level (e.g. events that represent whole system functions. We propose that a technology free scenario model should only contain purely conceptual interactions and events.[6] Furthermore, the structure of the scenario model of a technology free analysis must not have any influence onto the scenario model of the design, i.e. dur-

ing the design process the developers are free to change the scenarios arbitrarily, as long as the overall functionality remains the same.

3.5 External versus internal model

When looking at a system, a subsystem, a software component, a process or an object, we differentiate between the internal view which describes the system by its components, and the external view which describes the system as a black box. Both, the external and internal view contain dynamic as well as static aspects.

We can also say that the external view describes the "what" and defines the requirements, whereas the internal view describes the "how" and gives the design of a system, assuming that the external view defines the problem and the internal view a solution. These terms are often used as implying each other; but this is not necessarily the case.

Having two different models

Even when describing only the external view of a component, we need a certain syntax and semantic for this. Basically there are two possible approaches:

- The same notation is used to describe the external behaviour as is used for the internal view. Without additional information you cannot tell if such a model represents an external or an internal view.
- Different notations (and maybe even different paradigms) are used for the external and the internal model. The two models cannot be confused, but two different notations need to be mastered and the transition between the two models may be further impeded.

The advantage of the second approach is the clear separation between the external and the internal view. Due to the different modelling techniques, the models need to be redone when changing the view point. In a top-down approach, where first the external view is modelled and afterwards the internal structure is determined, this may be an advantage. Yet in a bottom-up approach, where we already know the internal components and thus can use these as semantic elements for the external description, it hinders us from having only one model that represents both view points. Here lies the strength of the first approach which works well, as long as the goal of the model is still clear and the external view is not mistaken for the internal view.

Whenever we distinguish between the external and the internal view, we must accept to have two models that look very similar and are about the same system, but differ in their contents and model elements (e.g. an external data model and an internal object model).

Correlations between the internal and the external views

Ideally, the external view does not make any unnecessary restrictions concerning the internal view, and information hiding is ensured. Far too often, this is not the case: the same model elements (e.g. object types and attributes) are used in the internal and the external view.

6. Every scenario model, irrespective of the abstraction level chosen, has a bias to one specific interface design. Therefore a really technology free scenario model cannot exist, we can only define explicitly that the structure of the scenario model of the analysis is not a prejudice for the design model.

3.6 Complete and unambiguous models versus essential models

Often mentioned goals of analysis models are that they confine to the essential information, do not go down into too low details, are unambiguous and complete. But these criteria cannot be fulfilled all at once.

Completeness and unambiguity of object models and scenario models

We might like to consider an object model to be complete, if we have found all the necessary objects and specified all the external views of all the object services. But in order to find all the objects and services we need to determine how the object services work and if they need further services from other objects. Otherwise we cannot verify if all the necessary objects and services are modelled[7]. Furthermore, for an unambiguous model, services names are not enough. The services need to be specified in such precise language. that no misunderstandings are possible. This could be a formal specification language, programming language or an unambiguous domain specific language.

The same is true for the scenario model. Also an unambiguous scenario model would have to use such a language. And in order to be complete, it would have to cover all possible sequences of events, also all exceptional and error scenarios, and not only the most typical ones.

Limits to complete and unambiguous analysis models

There are several points that restrict the development of complete and unambiguous analysis models:

- *Redundancy to the design process*: Total unambiguity necessitates to detail the analysis model (which is not yet the design model) until an unambiguous language level is reached. One of the criticism at SA/SD has been that the work is done twice: first the system is specified down to the level of pseudo code in the process model of SA, modelling the problem domain irrespective of the design of the software system. Afterwards the same level is specified once again in the design, this time for software components. An effort that is normally not justified for every part of the analysis model. Also the effort for a complete analysis model, especially for a complete scenario model, is not always justified by the expected return of investment.
- *Targeted audience*: The language preferred by the targeted audience is not necessarily an unambiguous language. E.g. the scenario models of common object-oriented analysis methods like OOSE, Fusion, OBA and OMT focus at easy understandability, but are not unambiguous.
- *Evolving requirements*: Not in every project all the requirements are known from the beginning and into all details. An unambiguous and complete analysis model can then only evolve with the design and implementation of the system (see e.g. also experience report in [Kolbe95].

Confining to the most essential informations

Many object-oriented methods do not try to provide a complete analysis model. They advocate to focus on the essential aspects. E.g. Booch [Booch94, page 253] distinguishes between primary scenarios (which illustrate key behaviours) and secondary scenarios (behaviour under exceptional conditions); only the first ones are all included in the analysis model. BON [Walden95, page 170] provides only on a very high abstraction level a mere list of all external

7. Most analysis model provide a list of the services offered by an object, but they do not mention which other services are needed to provide theses services. No verification concerning the completeness of the object model is possible.

events and scenarios. A full covering of all possible scenarios on a more detailed level is considered as unrealistic, so only a part of the listed scenarios are also modelled by object diagrams.

But if we specify as a goal of the analysis model to model only the essential information, we have a very vague criterion. On one hand, what "essential" means, depends on the other goals chosen. On the other hand, its interpretation remains always subjective, and must be decided upon in the concrete situations by the developers and reviewers, maybe backed up by risk assessment techniques.

3.7 Targeted audience

To be understood by users (and developers) versus to be understood by machines

If we want that the analysis model is understood by those involved in or affected by the system or problem that is modelled, we have to choose an appropriate language and to model the system in such a way that it reflects the thinking and perception of the users. Ideally such a modelling technique is close to other techniques used in the daily work of the users. Examples for this are the concept maps, event-response-lists and storyboards of [Umphress91], the informal scenarios of [Kilberth93], or the scripts of OBA [Rubin92] (e.g. [Heeg94] likes to work with these scripts, as they are so similar to the tables secretaries work with every day).

On the other hand, there is the desire for an automated verification or even execution of an analysis model. This necessitates a language that can be interpreted by machines, e.g. formal specification languages like Z, B, VDM, CO-OPN or LOTOS. But the resulting analysis models can only be understood by users if these are themselves software engineers. Otherwise the developers first need another analysis model that they can discuss with the users, and based on this they can develop the formal model. Of course, formal specification languages also necessitate that the software developers have the necessary educational background, training and experience for using formal specification languages; up to now this is only true for a minority of project teams.

"Natural" modelling techniques

The targeted audience determines also which modelling techniques are considered as being natural. Certain people consider a purely functional approach as natural [Ward94], others see the world as consisting of passive objects and of processes that manipulate these objects [McGinnes92], for others everything should be modelled as a system of interacting and active objects. Which techniques are perceived as natural is a question of experience, profession, culture, language and personal perspective. As a consequence, not every modelling approach does match well the thinking of the people involved in a specific analysis process, even if it is conceived as being very natural by its originators.

To be used for contracting versus to be used by the same team in an iterative approach

Will the results of the analysis model be used in-house by the same team that has specified the analysis model? Or will the analysis model be used for a contract? And will the analysis model be used for estimating the project size by certain estimation techniques (as e.g. the one proposed by [Moser92])? Also these goals must be known before the modelling process starts in order to avoid bad surprises.

3.8 Consequences

Contradictions

An analysis model cannot fulfil all possible expectations at once. We have already formulated various pairs of opposite goals. But also between these pairs there are dependencies. E.g. an unambiguous model going down to a precise and thus formal language level is not targeted at the user. Application- and reuse-oriented models are always models of the software system. And a system architecture is out of place in a model of the real world. It does not make sense to have goals that contradict each other, neither prescribed by a method description, nor foreseen in the project plan, nor existing implicitly in the heads of the software developers involved in the project.

Many difficulties that arise in the development process are just due to the fact that the goals of the analysis model are only implicit or that contradicting goals are taken over from a method description. As the goals are not defined and discussed explicitly, the effort is undertaken to develop a model that should satisfy contradictory goals. And as this is not possible, frustration is the consequence. But all those difficulties that are due to unclear, contradicting or even inappropriate goals cannot be resolved by an enhanced notation or an improved analysis process.

Using goals for the project plan

In any project, somebody has to define how many different models are to be developed and what their goals are. If this is not done explicitly before the models are constructed, it will be done implicitly. Then the goals are imposed by the person with the greatest influence, or they are determined with much additional effort when arguing about modelling issues which are in fact uncertainties concerning the goal of the model. Far better, the goals are defined by a method, by QA-guidelines and by the project plan (more about general model architecture frames and project plans can be found in [Beringer95]).

Responsibility of the development team

Some of the criteria we have mentioned here can be defined quite precise before the development of the analysis model starts. Other criteria (like e.g. what the essential information is or how detailed the model must be to avoid too much ambiguity but be still not too complex and still understandable by the targeted audience) can only be stated more precisely during the development process. There it is important that the developers and reviewers are aware of the vagueness of these goals. There is no easy guideline that could replace human judgement and human awareness.

4 Intent clashes

Besides the goals mentioned in the previous chapter, we may have further expectations concerning the analysis model and especially concerning the analysis and the design process. Many of the previous goals and of the expectations mentioned here are achievable when the effective developing process (analysis and design processes) is not considered. When it is taken into account, further difficulties arise. We formulate here these difficulties as three misconceptions and two intent clashes, and we also show how the various methods cope with these intent clashes.

4.1 Difficulties with the ideal analysis model

The ideal analysis model

When we are asked for the ideal analysis model and the ideal development process, we might mention the goals and expectations presented in figure 4:

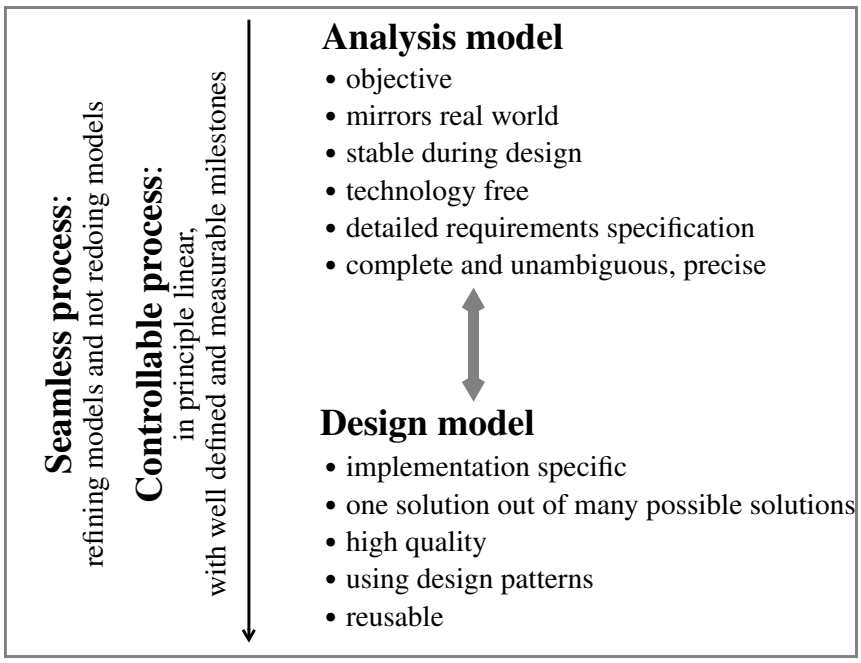


Figure 4: The ideal analysis model

If we specify the goals and expectations in this way, then we have to deal first of all with the problems we mentioned already in the previous chapter: some of these goals are contradictory (e.g. real world model and detailed requirements specification of the software system), and they are not precise enough to be useable guidelines in a concrete project (e.g. technology dependency). But there is also a further difficulty. Some of these expectations are just not achievable. Though it were very nice to fulfil them, they contradict the experiences of software development. In many projects it is not possible to make a stable analysis model, to make an analysis model that is an objective picture of the real world, or to make an initial requirements analysis model that can be seamlessly expanded into a design model.

First misconception: stable model

There exist some software development projects, where the functional requirements are well known from the beginning, and remain stable during the design. But often, this is not the case. On the contrary, there are projects where only a vague idea exists about what the system should do. Evolutionary prototyping or scenario modelling is done in parallel to the design and implementation of the system, in order to find out what the real needs are, and what the requirements should be like. Many people advocate the parallel advancement of design and requirements model, (e.g. [Holbrook90] mentions that the criteria and requirements for a complex design problem evolve at the same time as solutions are being formulated, and [Siddiqi94] quotes: “specification and implementation are intimately intertwined”).

In other projects, the functional requirements are known, or people think they know them, but they refer to the old technology. With a new technology new possibilities arise e.g. for the user interfaces. And these new possibilities not only affect the details of implementation, but also influence the metaphors and work procedures used in the analysis model. Changes in the scenario and object models are thus inevitable.

A further difficulty for a stable analysis model is the immense complexity in large projects. It is not possible to model first all the requirements, and then to proceed to the next phase. Instead, an iterative approach is chosen. First the most basic requirements are picked out, and a first system is made. Incrementally, other requirements are added. Though this approach avoids that the project never proceeds over the analysis phase, the analysis model is never complete, and substantial changes can become necessary as new requirements are added. An example for such an approach to software development is given by Kruchten in [Kruchten95] and [Kruchten96].

Second misconception: objective real world model

It is often said that an object-oriented analysis model models the real world. And it is assumed that in analysis there exists a model that is objective and is free of any design decisions. This assumption is not backed up by everybody. [Floyd89] states that every model is a construction, and that this construction is always subjective, depending on the background of the person who developed or designed it (see also [Siddiqi94]). There is no objective way to look at the world around us, the model is always shaped by the personal perspective. Even if in a project a so called objective real world model is made, it is just the subjective construction of that person in the team with the greatest influence.

Our perception of the real world also changes during a project. Metaphors are found and implemented during the development project, and after some time we may even use these metaphors outside this project when describing similar problems. And when we would make the real world model a second time, we would suddenly use the metaphors that come from the development of a software solution even in the real world model.[8]

Third misconception: initial final high-level view

In a seamless approach the analysis model can be expanded step by step into the design and implementation model (see figure 5). The initial analysis model as it exists after the first phase or milestone of the project is just a subset of the final system model.The same assumption is made for an initial data model that may be part of the analysis model.

Both assumptions hold for text book examples and for trivial systems. But they do not hold when the developers do not know in advance what the structure of the final system will be. If this final system should be of a good quality, then it cannot be a mere expansion of the initial analysis model (see also [Beringer94]).

In reality the model is reworked several times until those design patterns, object abstractions and interaction mechanisms are found that suit the problem, satisfy the quality demands, and will be implemented in the final system. The iterative approach is not a mere expansion and detailing of the model, but includes also changes of the model.

Also the final system model can be shown on various abstraction levels. The higher level views abstract away certain details of the lower level views, but they still reflect the essential

8. These metaphors are often very domain specific, and unfortunately rarely get published. An exception is the “tool-material-aspect” metaphor documented in [Gryczan92].

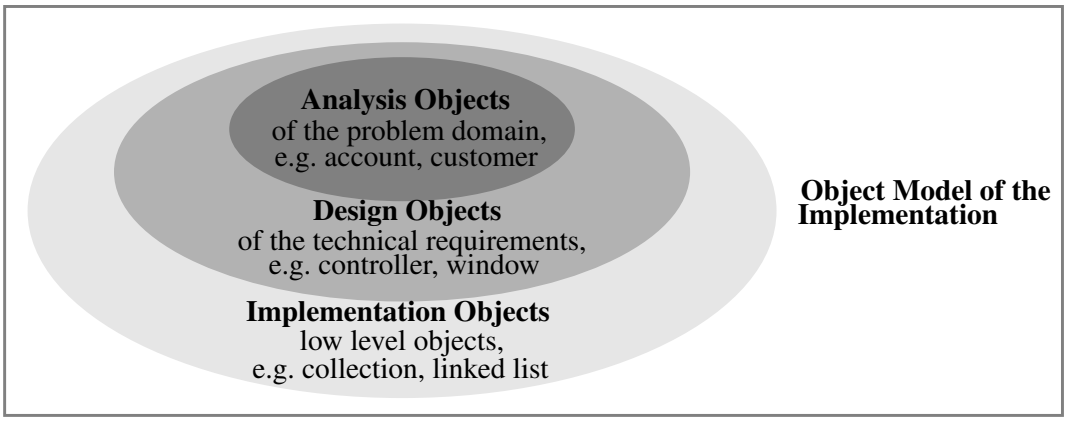


Figure 5: Seamless expansion of the analysis model

structures and concepts of the final model. But this final high-level view is not the same model as the initial analysis model, even if they use similar notations. If we use an approach where we first model the requirements in an analysis model and then develop the implementation, we have two high-level views of the system which are slightly or drastically different from each other (see figure 6). The differences may only lie in the model elements that are used for the description (e.g. the allocation of attributes to object types). Or the difference may even concern the overall structure of the scenario model and the object model.

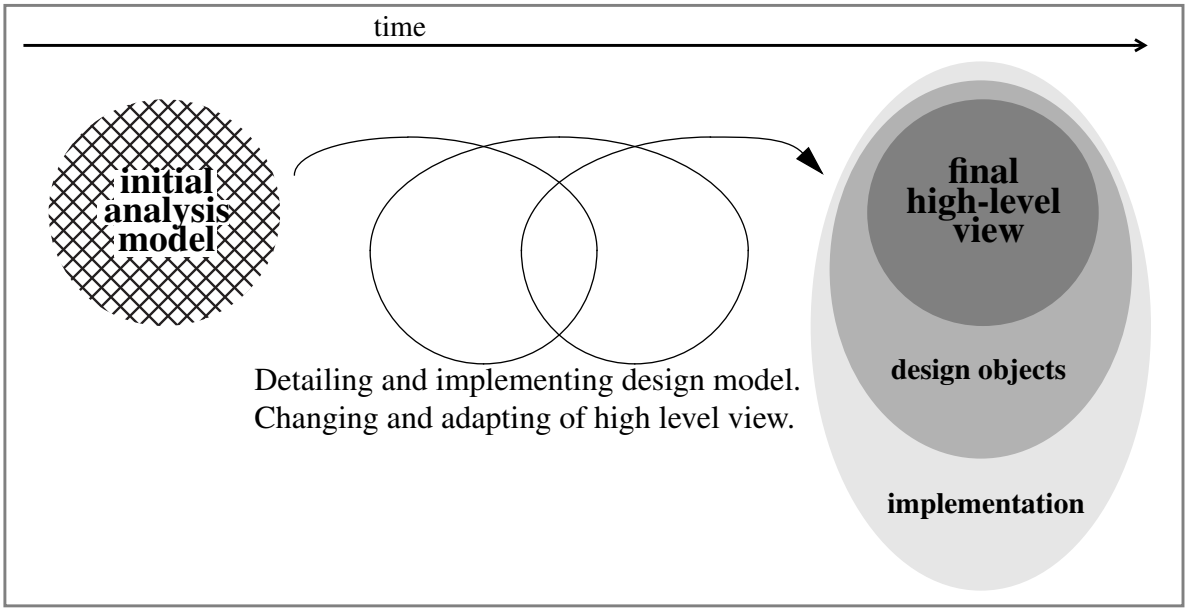


Figure 6: Initial analysis model versus final high-level view

4.2 The two intent clashes

First intent clash: controllable process <--> no stable requirements, no real world model

In industry we need controllable processes with milestones. We must be able to define what we expect from these milestones, and we must measure the results. We need to verify that the required work has been done correctly. This includes that the right work has been done and that

the work does not need to be redone later on. If we cannot guarantee and measure that at a milestone, then project management becomes extremely difficult, and outsourcing and contracting software is a nightmare.

The consequences for the analysis model are straightforward: we want an analysis model that can be completed in a first phase of the project and remains stable afterwards. Further we require that it is a real world model, so the desired content is specified clearly and we can easily decide at the milestone that the model is complete and correct. With such an analysis model and analysis phase, we would have at least at the beginning of the project well defined and controllable milestones. We could also use the analysis model to estimate the size of the remaining work, and to work out contracts for the development of solutions.

As we have shown above, such an ideal analysis model does not exist. We may fake an idealistic linear process in the system documentation (as proposed by [Parnas86]), but that does not work for the process itself. We thus have a clash between the constraints of project management, and the possible goals of analysis models and the needs for a highly iterative, and creative approach (see figure 7). The larger the project, the more difficult it becomes to reconcile these two intents. If we look at the proposals for project management e.g. by [Sadr96] or [Kruchten95] and at other guidelines for iterative object-oriented project management, we can see that they try to find compromises that do not satisfy the "purists" of either side. Their goal is a realistic and cost-efficient process that has some iterations, leaves some freedom for changes and creativeness throughout the development, and is still to some degree controllable.

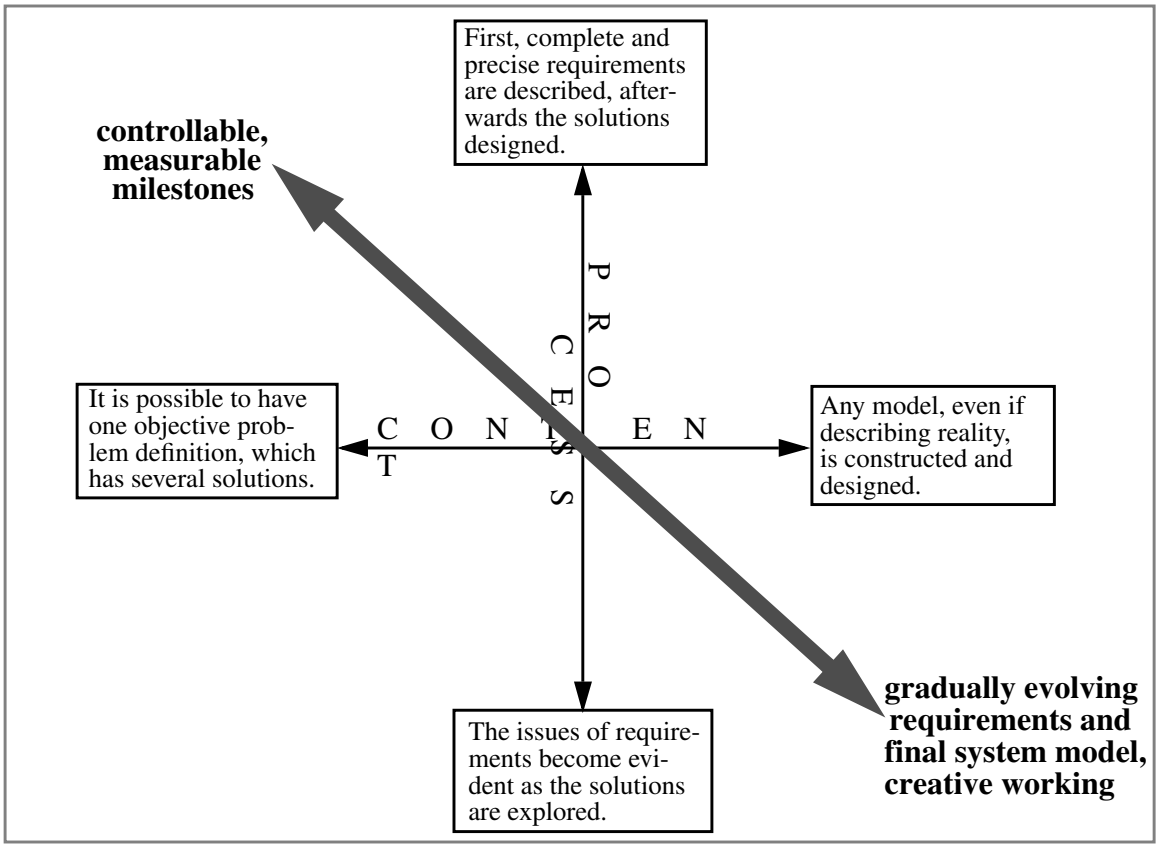


Figure 7: The first intent clash

Second intent clash: seamless process <--> initial final high level view

Even before object-oriented modelling techniques became popular, one tried to find well defined links between the analysis and the design model. There were at least two reasons why a well defined or even better a seamless transition between these models was required:

- A clear connection between the various models is needed in order to enable maintenance.
- Rules (and even automatisms) for deriving the design model from the analysis model and the implementation from the design model are requested, in order to facilitate and control the development process. Often the dream has even been to have an automatic code generation from a problem model.

When object-oriented technologies have been introduced, great hopes arose that with having one paradigm for the whole development process, one would finally have a seamless process. But as we have mentioned before, this works only when modelling the system from a retrospective perspective. So for most projects, the initial analysis model is not identical to the final high-level view. This is very evident in a linear process, when the initial model remains unchanged. But it is also true for an iterative process, only that in an iterative process often the analysis model gets adapted step by step to reflect the concepts found in the final system. So the intent clash between the desire for a seamless process and the impossibility to have an initial analysis model that reflects the final high-level view remains.

4.3 Approaches to handle the intent clashes

The various methods handle the intent clashes differently. We have categorized the methods concerning their treatment of the intent clashes into three groups: methods that have an idealistic approach, methods that just focus on the evolution of the final system model, and methods that bother with developing, reworking and maintaining two different models over the whole development process.

4.3.1 The idealistic approach

The idealistic approach does not really provide a solution to the intent clashes. It just hopes that the clashes are not heavy and can be neglected. A more or less linear approach is chosen, and the goal of the analysis phase is a stable and real world analysis model that defines the functional requirements for the system. At the end of the analysis phase, this model is reviewed and accepted. In the design it is directly enhanced or transformed into the final software model and is thus a high-level view of the design model.

Of course it is admitted that there may be some overlap of the analysis process with the design process and that changes to the analysis model during the rest of the development process cannot be avoided. Yet it is tried to reach the ideal process as close as possible, and to produce at least a documentation that fakes it. Most data-driven or use case driven object-oriented methods fall into this category, though much depends on how they are interpreted and applied in practical projects.

An example: Fusion

In the first phase of the project, the functional requirements are described by an analysis model that consists of the object model and the interface model. The analysis model describes the intended behaviour of the system as it is externally visible, without any implications for the de-

sign and implementation of this behaviour. In the ideal process, the analysis model can then directly be refined and enhanced into the models of the design. Design objects are added and the functions are distributed onto the different objects. The objects and system operations in the analysis model do not only define the external visible behaviour of the system but ideally also correspond to the high-level view (subset of objects, attributes and methods) of the final object-oriented system model. The design model should of course also satisfy the quality criteria for a good object-oriented design.

4.3.2 The one-model approach

In the one-model approach, the goal is always the design model. No analysis model is made, work is started right away at this final model with an iterative and evolutionary approach. In the first versions of this model, a part (or all) of the user's view of the system is modelled on a high abstraction level. In the following versions, the model is gradually reworked and refined, until the final object structure, with the final metaphors, design patterns and object specifications has evolved. No model is kept of the initial user's view, it is assumed that his perspective also adapts to the high-level view of the final software system.

The one-model approach provides quite a seamless process (not by expanding the analysis model but by reworking it) and has only become possible by having the object-oriented paradigm throughout the whole development process. It overcomes the second intent clash by not maintaining any initial analysis model. But its weakness concerns the controllable process. It is very hard to define easy controllable and measurable milestones. This becomes also evident when looking at the methods using this approach: they can only provide quite vague guidelines how the project could proceed, but cannot provide any predefined milestones with well measurable criteria. Some methods even do not mention the difficulties of project control at all, which does not cause any problem as long as they are used in an environment where no project control is required, as is the case for many small or in-house projects.

Another drawback of this approach is that there is often no documentation of the user's view or of the high-level view. Even if it is assumed that the user's view adapts to the vocabulary and structure of the final software system, having only a detailed design model is not enough for the maintenance of the system.

An example: Booch

"Analyse a little, design a little, test a little" is the motto of the development process of Booch [Booch93].In addition to this microprocess, a macroprocess is suggested that helps to bring some structure into the development process. Each phase of the macroprocess consists of several iterations of the microprocess. Beside the static class diagram also other views and diagrams are supported. Yet their purpose is only to help evolving the object model of the final system.

An example: FORAM

"Object-oriented analysis is analysis, but also contains an element of synthesis. Abstracting user requirements and identifying key domain objects are followed by the assembly of those objects into structures of a form that will support physical design at some later state. The synthetic aspect intrudes precisely because we are analysing a *system*, in other words imposing a structure on the domain." (page 2 of training note 2 [Graham93]).

An example: BON

BON uses right from the beginning a notation that represents systems of interacting objects. During the development process the initial model is reworked to improve its structure, and further details are added. BON describes the development process with *activities* and *tasks*. The activities are orthogonal to the tasks, though some occur mainly in the analysis tasks, others mainly in the design tasks. Analysis consists of tasks for gathering information and describing the gathered structure, and is mainly concerned with the problem domain per se (to which degree technical and implementation dependant details are considered right from the beginning depends on the constraints of the project). Yet also analysis is already concerned in finding good object-oriented abstractions, the domain is described solution-oriented. All tasks of the development process as targeted at evolving and specifying the interfaces of the classes. Other models than the class specifications, e.g. the object scenarios, are either discarded as soon as they are no more needed to determine the class interfaces, or they are updated to reflect the final system and kept for documentation purposes. The amount of changes that can affect other views is minimized in that these views do not show much details, e.g. the object scenarios do not indicate the operation and parameter names.

An example: Floyd

In [Floyd89] a constructivistic perspective is the basis for the development process. Therefore the requirements are not analysed, but constructed. The goal is the implementation of the final system, user satisfaction has the highest priority. The participative design allows that users learn from developers, and developers from users, and that together the new system and the new ways how work should be carried out in the future are worked out (see also [Reisin90] and [Schauer93]). This will also result in new metaphors, work patterns and design patterns. But there is no predefined process for reaching this goal. The process evolves as the system evolves.

4.3.3 The two-model approach

Another approach is chosen by those methods that explicitly develop two models, an analysis model for the initial definition of the requirements and a design model for documenting the final software system. The first one reflects the user's perspective as it is at the beginning of the project, and it describes the functional requirements as complete as possible, and in a precise and detailed manner. The notation is chosen in such a way that it mirrors best the thinking of the users (e.g. in [Ward94] this is a purely functional description of the system behaviour), a seamless transition to the design is of low importance. Either the analysis model is finished and reviewed after the first phase of the project, afterwards it is considered as more or less stable, and the user does not participate in the design. Or the analysis model is developed in parallel with the design model, but there is a clear distinction between the analysis activities and the design activities, even different teams may be involved. The analysis model is only adapted to changing or to new requirements, it is not adapted to the structure or model elements of the design model. The design patterns, data structures and scenario structures used in the design do not affect the analysis model.

Most methods that use a two model approach also use different modelling techniques and sometimes even modelling paradigms for analysis and design. Very naturally this prevents from carrying over any modelling decisions from the analysis into the design, the analysis model only serves as an information source. In the design a totally new model has to be made, and the model elements of analysis cannot be directly reused, even if the two models go down to the

same level of detail and precision. The advantage of this approach is a clear separation of catching requirements and designing the system. [Ward94] even considers the missing seamlessness to the design and the change in the notation as an advantage, as it forces the developer to search for a good design model, and prevents him from carrying over the constructions of the analysis into the design. The disadvantage is a high effort for developing and maintaining two different models in different techniques for things that are quite similar. Avoiding changes in the analysis model may also hinder the evolution of a new understanding of business processes, the creation of new concepts and their transfer to the users.

Examples for this approach are MSA-GfAI, MSA as defined by [Yourdan89], and the approaches proposed in [Høydalsvik93], [Holbrook90] and [Ward94].

The best approach...

Which approach is the best one depends on the risks a specific project faces. If the greatest risk is to miss the optimal object-oriented structure in the final system and to get fixed on old work procedures, then probably the one model approach is the best one. If the greatest risk is that the project gets out of control, then maybe one of the other approaches is better. None of these approaches is the silver bullet. The intent clashes cannot be overcome, only weakened, and every project has to fight to some degree with them.

5 Conclusions

We discussed in this report various interpretations of the term "analysis" and we examined if we can use these interpretations to determine the content of the analysis model and the end of the analysis process. We also considered the conflicts that exist among some of the goals used to characterize the analysis model, and we discussed the intent clashes that exist between some requirements of the analysis process and some goals of the analysis model. Finally, we presented three different approaches for dealing with these intent clashes.

Due to the inherent tensions between contradicting requirements, we could not present a set of well defined analysis goals that is applicable for every project. Compromises are project specific, taking into account the project specific risks. But this does not mean that we may omit the explicit definition of the analysis goals in a software development project. Rather a higher awareness of the influence of the goals onto the content of the analysis model is needed.

6 References

[Bailer93] B. Bailer; *Geschäftsmodelle, Verständnis und Folgen der Technologieneutralität*; Institutsbericht Nr. 93.43, Institut für Informatik der Universität Zürich, 1993

[Berard93] E.V. Berard; *Essays on Object-Oriented Software Engineering, Volume1*; Prentice Hall, 1993

[Beringer94] D. Beringer; *Limits of Seamlessness in Object-Oriented Software Development*; Proceedings of TOOLS Europe '94, Versailles, 1994

[Beringer95] D. Beringer; *The Model Architecture Frame: Quality Management in a Multi Method Environment*; Proceedings of SQM'95, Seville, 1995

[Booch94] G. Booch; *Object Oriented Analysis and Design*; Benjamin/Cummings, 1994

[Coleman94] D. Coleman et al.; *Object-Oriented Development: the Fusion Method*; Prentice-Hall, 1994

[Cook94] S. Cook, J. Daniels; *Designing Object Systems, Object-Oriented Modelling with Syntropy*; Prentice Hall, 1994

[Davis90] A. M. DAVIS; *Software Requirements: Analysis and Specification*; Prentice Hall, 1990

[Davis93] A. M. DAVIS; *Software Requirements: Objects, Functions and States*; Prentice Hall, 1993

[DeMarco79] T. DeMarco; *Structured Analysis and System Specification*; Yourdon Press, 1979

[D'Souza93] D. D'Souza; *Comparing OO Analysis and Design Methods*; Proceedings of the OOP 93 in Muenchen

[Eckert93] G. Eckert, P. Golder; *Improving Object-Oriented Analysis*; rapport technique No 93/32, Departement d'Informatique, EPFL, 1993

[Eckert95] G. Eckert; *Improving the Analysis Stage of the Fusion Method*; Fusion in the Real World, edited by R. Malan, R. Letsinger and D. Coleman, Prentice-Hall, 1995

[Floyd89] C. Floyd; *Softwareentwicklung als Realitaetskonstruktion*; Proceedings of the Fachtagung Software-Entwicklung, of the GI, Marburg, 1989

[Graham93] I. Graham; *Financial Object-Oriented Rapid Analysis Method (FORAM), Training Notes 1 to 4*; seminar given at the SBV in Basel

[Gryczan92] G.Gryczan, H.Züllighoven; *Objektorientierte Systementwicklung, Leitbild und Entwicklungsdokumente*; Informatik-Spektrum Nr. 15, 1992

[Heeg94] G. Heeg; *Object Behavior Analysis - Von der Suche nach Objekten*; talk given at a meeting of Choose and INTEC-OO (Bern, Switzerland, 1994)

[Holbrook90] H. Holbrook; *A Scenario-Based Methodology for Conducting Requirements Elicitation*; ACM SIGSOFT, vol 15, no 1, 1990

[Høydalsvik93] G.H. Høydalsvik, G. Sindre; *On the Purpose of Object-Oriented Analysis*; Proceedings of OOPSLA 93, 1993

[Huber90] G. P. Huber; *A Theory of the Effects of Advanced Information Technologies on Organizational Design, Intelligence, and Decision Making*; Academy of Management Review, Vol 15, No 1, 1990

[Jacobson92] I. Jacobson; *Object-Oriented Software Engineering*; Addison Wesley, 1992

[Jufer92] R. Jufer; *Strukturierte Spezifikation*; seminar of the GfAI Gruppe für angewandte Informatik AG, 1992

[Kilberth93] K. Kilberth, G. Gryczan, H. Zuelligkoven; *Objektorientierte Anwendungsentwicklung: Konzepte, Strategien, Erfahrungen*; Verlag Vieweg, 1993

[Kolbe95] K. Kolbe; *How do you know you are building the "right" software? Experiences with Use Cases*; Workshop on "Requirements Engineering: Use Cases and More", OOPSLA 95, 1995

[Kotonya96] G. Kotonya, I. Sommerville; *Requirements engineering with viewpoints*; Software Engineering Journal, January 1996

[Kruchten95] P. B. Kruchten; *The 4+1 View Model of Architecture*; IEEE Software, November 1995

[Kruchten96] P. B. Kruchten, C. J. Thompson; *Iterative Software Development for Large Ada Programs*; Proceedings of Ada-Europe'96, Montreux, June 1996

[McGinnes92] S. McGinnes; How Objective is Object-Oriented Analysis?; Proceedings of CAiSE'92, 1992

[Moser92] S. Moser; *Metrics and Estimation, Aufwandschätzungen in Informatik-Projekten*; seminar of the GfAI Gruppe für angewandte Informatik AG and of IBM Switzerland, 1992

[Neighbors80] J.M. Neighbors; *Software Construction Using Components*; Technical Report 160, Department of Information and Computer Sciences, University of California, Irvine, 1980

[OMG92b] *Object Analysis and Design, Volume 1: Reference model, Draft 7.0*; OMG, 1992

[Parnas86] D.L.Parnas and P.C.Clements; *A Rational Design Process: How and Why to Fake It*; IEEE Trans. on Soft. Eng., Vol 12, No. 2, Febr. 1986

[Rawsthorne95] D. A. Rawsthorne; *Transaction Based Analysis*; Workshop on "Requirements Engineering: Use Cases and More", OOPSLA 95, 1995

[Reisin90] F.Reisin; *Kooperative Gestaltung in partizipativen Softwareprojekten*; Dissertation, Technische Universität Berlin, 1990

[Rubin92] K.S. Rubin, A. Goldberg; *Object Behavior Analysis*; Communications of the ACM, Sept. 1992

[Rumbaugh91] J. Rumbaugh et al.; *Object Oriented Modelling and Design*; Prentice-Hall, 1991

[Rumbaugh94a] J. Rumbaugh; *The life of an object model*; JOOP March-April 1994

[Sadr96] B. Sadr, P. J. Dousette; *An OO Project Management Strategy*; IEEE Computer, vol29 no9, 1996

[Schauer93] H. Schauer, B. Kuhnt; *Partizipative Software-Entwicklung*; Institutsbericht Nr. 93.35, Institut für Informatik der Universität Zürich, 1993

[Shlaer88] S. Shlaer, S. J. Mellor; *Object-Oriented Systems Analysis, Modeling the World in Data*; Yourdon Press, 1988

[Siddiqi94] J.Siddiqi; *Challenging Universal Truths of Requirements Engineering*; IEEE Software, March 1994

[Umphress91] D. A. Umphress, S.G.March; *Object-oriented requirements determination*; in JOOP Focus on A&D, edited by R. S. Wiener, SIGS Publications, 1991

[Waldén95] K. Waldén, J.M. Nerson; *Seamless Object-Oriented Software Architecture*; Prentice Hall, 1995

[Ward94] R.Ward, J. Stevens; *Object Orientation Is Not Always Best!*; Proceedings of Ada in Europe, Copenhagen, 1994

[Webster] *Webster's Ninth New Collegiate Dictionary and Webster's Collegiate Thesaurus*; NeXT Digital Version, 1988/1992

[Wilkinson95] N. M. Wilkinson; *Using CRC Cards: an informal approach to object-oriented development*; Prentice Hall, 1990

[Wirfs90] R. Wirfs-Brock et al.; *Designing Object-Oriented Software*; Prentice Hall, 1990

[Yourdon89] E. Yourdon; *Modern Structured Analysis*; Prentice Hall, 1989

The Goals of the Analysis Model

Dorothea Beringer
Laboratoire du Génie Logiciel
LGL-DI-EPFL
CH - 1015 Lausanne
d.i.beringer@ieee.org

December 1996

Abstract

When developing an analysis model, the notation itself does not suffice for determining the content of the analysis model and the end of the analysis process. For this, the goals and intents of the analysis model are used, either implicitly or explicitly. In this report we take a look at various possibilities to define the term analysis and the goals of the analysis model. We start with considering the reference model of the problem solving cycle. We examine the definitions that refer to analysis as the "what" in opposite to the "how", or as the problem definition phase of a problem solving cycle. We then continue with a discussion of various criteria (like technology independence, real word model, unambiguous or essential model) that are often used to specify the goals of an analysis model. Difficulties that arise with these criteria are that they cannot be combined arbitrarily and that some of them are very vague; we propose therefore also some more concrete formulations. In the last part of the report we take a closer look at some misconceptions and intent clashes that are due to a too idealistic model of the software development process. Objective real world models, analysis models that are stable after the analysis phase and analysis models that are equal to the high-level view of the final software system are often attempted, but are normally not realistic. This leads to intent clashes with the desire for a seamless and easy controllable development process. We explain these intent clashes and we also describe the three different approaches taken by software development methods to cope with these intent clashes.

1 Introduction

When developing a model, we hopefully pursue a certain objective which gives us the motivation to undertake that task. Such a motivation may be to give a brief overview for managers, to evaluate the feasibility of a software system, to set up a contract for a specific software system for outsourcing, to provide the necessary requirements specifications to be used by sophisticated developers, to fulfil somehow the notational requirements of certain QA-guidelines, or to try out a new notation.

Depending on the motivation behind the modelling process, we then choose a notation and define the intent or goal of our model (we will use the terms "intent" and "goal" interchangeably). Only that information appears in the content of the model that is describable by the notation and corresponds to the intent of the model (see figure 1).

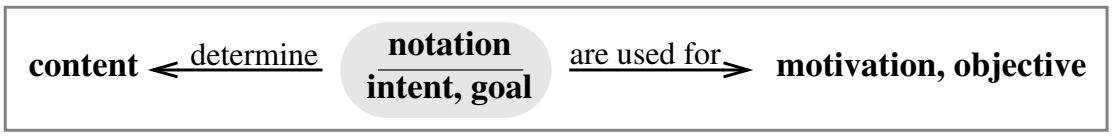


Figure 1: Objective, content and intent of a model

A certain notation can be combined with many different modelling intents. E.g. an ERD can be used to show the logical model of all the tables found in a relational database. It can also be used to model all the data in a certain problem domain. Or it can be used to give a static view of all the objects in an object-oriented system, considering only attributes. Even if the notation is exactly the same, the content of the model, even if it is about the same system, may vary heavily.

Considering intents and goals of models is important for several reasons:

- Most modelling notations can be used in several contexts. Yet many methods do not state explicitly for which modelling intents their notations can or should be used.
- When comparing different methods, e.g. concerning their techniques for modelling the global behaviour, the differences in the goals of the models are at least as important as the differences in the notations used. For choosing a certain method, the achievable goals should even be of higher priority than notational issues.
- Terms like analysis are heavily overloaded. Without making it more precise what the intent of a specific analysis model is, misunderstandings, endless discussions and wasted modelling efforts are inevitable. Stating only that an analysis model should be developed, is not enough.
- Managers, end-users, system engineers and software engineers have different motives for making an analysis model.
- To judge the quality of an analysis model, we do not only want to determine if the notation is used correctly but also if the right content is represented. But appropriate quality criteria depend on the goals of a specific model.
- The definition of the goals of the different models is also important when defining project management guidelines and project control. In a multi-method environment the intents of the models are even the only bases on which a general model architecture can be defined (for more details see [Beringer95]).
- Discussions concerning the seamlessness of object-oriented methods cannot only be based on notational issues. They must take into account the limitations caused by the

differing goals of models on different abstraction levels (for more details see [Beringer94]).

In the following we will discuss possible goals of analysis models. But first, we take a closer look at the term "analysis", and we use for this the reference model of the problem solving cycle.

2 The term "analysis" from the perspective of the problem solving cycle reference model

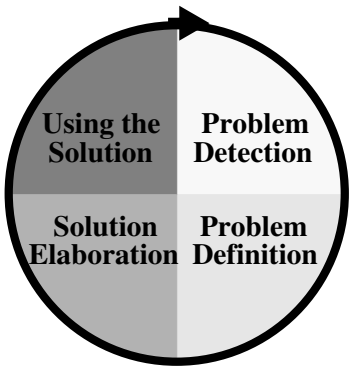
Many methods use the term analysis for the first activities that are undertaken in order to solve a given problem. We therefore describe here the reference model of the problem solving cycle and examine how analysis could be defined, based on this model.

2.1 The problem solving cycle

A software development project tries to solve a certain problem by providing a computerized solution. Yet problem solving is not restricted to software engineering, it happens every day. The problem solving cycle is a reference model that describes the process of solving a problem in a very general way. It has also been used in the software development guidelines of NCR Switzerland and in the QA-guidelines of the GfAI Switzerland.

The four phases of the cycle

The four phases of the problem solving cycle are described in figure 2.



- **Problem Detection**: It is recognized that there is a problem and certain measures are taken. What the problem really is about, is often not yet clear.
- **Problem Definition**: The problem gets analysed and defined. Priorities are set and the compulsory and the nice-to-have objectives are defined. It is determined ***what*** problem needs to be solved.
- **Solution Elaboration**: One or several solutions are evolved, analysed, decided upon and put into practice. If several solutions are considered, these must be checked against the objectives, measured and selected. It is determined ***how*** the problem is solved.
- **Using the Solution**: The solution is in operation. It has to proof that it can yield the promised benefits. If new requirements or defects appear, a new cycle is started.

Figure 2: The problem solving cycle

Omitting the definition of the problem

The different activities during the four phases of the cycle vary heavily depending on the domain we are looking at. In politics, the elaboration of the solution may denote the proposing, discussing and voting on new laws, in engineering this may involve building and trying out prototypes. Yet common to all the different domains is the fact that either the problem is defined *explicitly* before or in parallel to the elaboration of the solution, or this is done *implicitly* during

the elaboration or even using of the solution. Omitting the definition of the problem is not possible. It can only be deferred. The objectives can be hidden behind the solution. But this normally causes unnecessary controversies and additional costs.

Recursive problem solving cycles

At any point in the cycle, new problems can be detected that give rise to starting a new problem solving cycle within the existing cycle. In fact any individual activity in the original cycle can be seen as another smaller cycle. This results in a recursive structure of arbitrary many problem solving cycles.

Iterations and backtracking within the cycle

During the problem definition phase it may turn out that the wrong problem has been identified. Or during the solution elaboration phase the problem definition may appear to be ambiguous or to be unsolvable due to overconstricted objectives. Or when putting a solution into operation, some defects may be detected which necessitate some further elaboration of the solution. In all these cases, we have to go back and iteratively redo the work of previous phases. Of course, it is often a matter of definition if we consider such a backtracking as going back within the same problem solving cycle or as a new recursive problem solving cycle.

2.2 Applying the problem solving cycle to software engineering

Any work done in the realm of software development can also be seen from the viewpoint of the problem solving cycle. We may apply the reference model of the problem solving cycle to planning a new system, to maintenance work, to implementing a reusable class library, or to developing a new software application. Often the life-cycle of a whole software system is looked at as one single problem solving cycle. The development project starts due to a problem detection and ends when the solution is effectively used. The tasks of such a project can be subdivided in those that serve at defining the problem and those that serve at elaborating the solution. They are carried out basically in a sequential order, yet always include some amount of backtracking and iterations.

The recursive nature of the development process

Yet applying the problem solving cycle for the whole life-cycle of a project or software system is only one out of many possibilities. In fact each activity within the project and each subsystem or component corresponds again to a problem solving cycle. Due to its recursive nature, the problem solving cycle can be applied at any level of granularity. Restricting its application to the life-cycle of the project as a whole is an arbitrary decision. This becomes especially evident, when considering more complex software developments where several project teams work at several applications which together make up some large system, composed of organisational, hardware and software elements. From the overall system view, many details that for the individual teams belong into the realm of problem definition are considered as implementation details. Thus exactly the same activities and models may in one project be considered as belonging to the problem definition of a problem A, and in another project as belonging to the elaboration of the solution of a problem B, without ever making it explicit that problem A could be considered as a recursive problem solving cycle within the solution elaboration of problem B. An example for such an approach is found in [Berard93]. Berard proposes a recursive/parallel life-cycle for software development with recursive cycles of problem definition and prob-

lem solution. Exactly the same methods, techniques and phases are used on several levels of decomposition, taking into account the recursive nature of the problem solving cycle.

"What" versus "How"

The problem definition defines *what* the problem is about. The elaboration of the solution shows *how* the problem is solved. When applying this distinction to a software development project, we can say that in a first phase (or in the first phases) the *what* of the system under consideration needs to be defined, and in a second phase the *how*. But the *"what - how"* rule does not really help us to distinguish these two phases or their corresponding models, because often it is not clear if a certain statement describes the "what" or the "how" of the system. This is due to the following reasons:

- The problem solving reference model only tells us that theoretically such a differentiation between statements belonging to the problem definition and belonging to the solution of a specific problem can be made. It cannot give us any guidelines as how to make the differentiations. Such guidelines are not possible, because only by deciding in a particular case that a statement defines the *what* and not the *how* we know what the problem is, not before.
- Davis describes the "*what versus how*" dilemma with "one person's what is another person's how" [Davis93]. For certain persons the exact look of the user interface is the *how*, for others it belongs to the *what*. What it really is depends on the concrete circumstances and on the perspective and expectations of the persons involved. Where we set the borderline between defining the problem and finding a solution depends on what we perceive as being the problem. This again depends on our role in the project (manager, user, network specialist, programmer) and our personal opinion.
- Due to the recursive nature of the problem solving cycle, the *how* of an encapsulated recursive cycle refines the *what* of its outer cycle. Also the *what* of an inner cycle refines the *how* of the outer cycle. Thus a certain statement is at the same time the *what* of one problem and the *how* of another problem.

This dilemma becomes also obvious when considering the criteria that are given by a method to determine if the model of the problem is finished or not. These criteria are often very vague and subjective, and they do not really help to find an objective judgement on the completion of the problem definition.

Limits of the reference model

Though the problem solving cycle is a very valuable reference model for software development and though it is mentioned in one or the other form in many development methods, its significance for practical engineering projects has severe limits. First of all, applying the reference model only to the level of a whole project contradicts its recursive nature. Secondly, the reference model only makes the statement that there exist such problem solving cycles and that these can be decomposed into four phases.

The reference model helps to analyse the software development process and its different activities from a retrospective view, providing one possible perspective of software development (see also [Floyd89]). It enhances our understanding of what is going on in the development project but it does not determine what in a concrete situation should be considered as problem detection, problem definition or problem solution. Thus it cannot serve as a guideline for determining in a project if something belongs to the model of the problem detection, problem definition or to the model of another phase.

2.3 Where does the term "analysis" fit into the problem solving cycle?

Considering the problem solving cycle, there are different elements that we could call "analysis". And in fact, there are several approaches of what activities and results are called analysis.

Analysis defined as modelling the "what"

Often it is said that the analysis model defines the problem and describes the "what", in contrast to the design model which describes the solution, the "how". Examples:

- OMT: "Analysis is understanding a problem; design is devising a strategy to solve the problem..." [Rumbaugh94a].
- [Booch94, page 252]: "The purpose of the analysis is to provide a model of the system's behaviour. We must emphasize that analysis focuses upon behaviour, not form. ...analysis must yield a statement of what the system does, not how it does it."
- [Walden95, page 122] mentions that the current consensus on the meaning of object-oriented analysis seems to be "creating an object-oriented model from the external requirements (a model of the problem)".
- [Shlaer88, page 93]: "The first thing to do is to build the information model to define the conceptual units you will be working with. In doing this it is important to stay focused on the real world - the problem - and not the solution to that problem, which will be supplied with the automated computer system."
- In [Graham93] analysis is characterized as "identifying large scale structure, just enough to describe the *what* of the system, rather than the detail of the *how*."
- [Rawsthorne95]: "...analytic models - a set of models describing *what* the system will do; design models - a set of models describing *how* the system will be implemented...".
- In [Wilkinson95] analysis and design are differentiated in the following way: "Analysis, or problem modelling, in which the problem is described and represented... Design, or solution modelling, in which a solution to the problem is discovered and represented...". In the design model "additional classes and mechanisms, which support *how* the system works, are built upon the existing classes, which describe *what* the system does".
- Davis describes analysis as "understanding the problem and the user's needs, identifying all possible constraints on a solution; organizing the plethora of assembled information". This in contrast to the requirements specification: "describing the expected behaviour of the product to be built to solve the now understood problem". As synonyms to analysis he uses requirements analysis and problem analysis. [Davis90, Eckert93].

Considering what we have said in the previous section, it becomes evident that these definitions help us to categorize an analysis model in a retrospective view of a project, i.e. when the problem is already defined and we know what belongs to its definition and what to its solution. Yet they cannot provide any precise and objective criteria which could help during the modelling process of analysis to specify the goals of the analysis process and to determine the content of the model[1]. For this, other criteria are necessary.

1. The usefulness of the "what - how" distinction for defining requirements engineering or analysis is quite often questioned, e.g. also in [Siddiqi94], where Siddiqi examines the "universal truth" that requirements describe the "what" of a system and not its "how".

Analysis in its common meaning

If we consider the meaning of analysis as it is used in daily life and in most other disciplines (e.g. as defined by [Webster]: "an examination of a complex, its elements, and their relations"), then analysis activities can be found in any of the different phases of the problem solving cycle. Analytical examinations are necessary to define the problem, but they are also necessary to create a solution or evaluate a solution. Every modelling activity involves the separation of a complex into its elements and relations. This usage of the term "analysis" is referred to by [Walden95, page 122] as being the traditional sense of analysis, prior to its reinterpretation which is specific to current software development methods. Examples for this interpretation of the term analysis are:

- Wirfs-Brock uses the term analysis in this sense when dividing the design process into an exploratory phase and an analysis phase[2] [Wirfs90].
- The term analysis is also used for domain analysis in the sense[3] of taking a closer look at a specific problem domain, business process or enterprise. The resulting domain analysis model has no connection to the future software system, though domain analysis may of course precede a system development effort. Examples are the domain analysis in [D'Souza93] and [Beringer94], the enterprise modelling in [Jacobson92] or the strategic modelling in [OMG92b].
- [Jacobson92] differentiates between the process of requirements analysis which has as input a requirements specification and produces a requirements model, and the process of robustness analysis which investigates this model concerning its robustness and produces the analysis model (and not a robustness model as could be expected). So in OOSE the term analysis is used in two different meanings.

Any modelling activity includes analysis in this general sense. Therefore this original and common definition of analysis does not provide any basis for defining the content and intent of an analysis model in the software development process.

Analysis as being equal to specifying requirements

Another possibility is to use analysis as synonym for requirements specification. In the problem solving cycle we specify requirements during the problem definition as well as during the evaluation and selection of a solution. So this definition would cover the second phase and part of the third phase of the problem solving cycle.

In the realm of software development, not all specification issues are decided upon by the same people. For certain decisions the customer takes responsibility, for other decisions the developers have their full freedom, as long as no requirements constraints of the customer are violated. We could therefore say that analysis only contains all those requirements specifications which are under the responsibility of the customer (this can then be seen from retrospective as the problem definition phase). Such a definition of the term analysis is only possible if in the team where the term is used it is clear, which details of the requirements must be defined by the

2. The analysis phase denotes the refinement of the objects into subsystems and subtyping hierarchies and the specification of the method signatures. It has nothing to do with requirements definition or problem analysis.
3. Other usages of the term "domain analysis" are:
   - Finding the problem domain objects for the analysis model of a software system. E.g. in OOSE the requirements model consists of a problem domain object model and a use case model [Jacobson92].
   - Defining concepts that are valid beyond one specific software application in order to develop reusable software components for one specific application domain. [Neighbors80, Berard93] state: "System analysis states what is done for a specific problem in a domain while domain analysis states what can be done in a range of problems in a domain..."

user and which ones are decided autonomously by the developers. So the content of the analysis model becomes dependant on the organisational context, on the specific project management approach and on contractual conditions. And whenever a participative approach to software development is chosen (as e.g. described by [Reisin90]), this definition of analysis does not work at all.

2.4 Consequences

All these possible definition of the term "analysis" do no provide the necessary criteria to determine the content of the analysis model. Also the reference model of the problem solving cycle only defines that such a model exists, and gives reasons for it. But due to the retrospective view at the analysis process, it is of limited help as a guideline for the analysis model.

3 Criteria for defining the goals of analysis models

When looking at the literature, various criteria for the goals of the analysis model are mentioned that should support the developer in determining the content of the analysis model and the end of the analysis process. Figure 3 gives an overview on those criteria that go beyond the simple "what-how" we discussed in the previous chapter. Some of the criteria we present in the following sections are very vague, so we propose more precise definitions where possible. Moreover, some criteria exclude other criteria. We mention some of these contradictions.

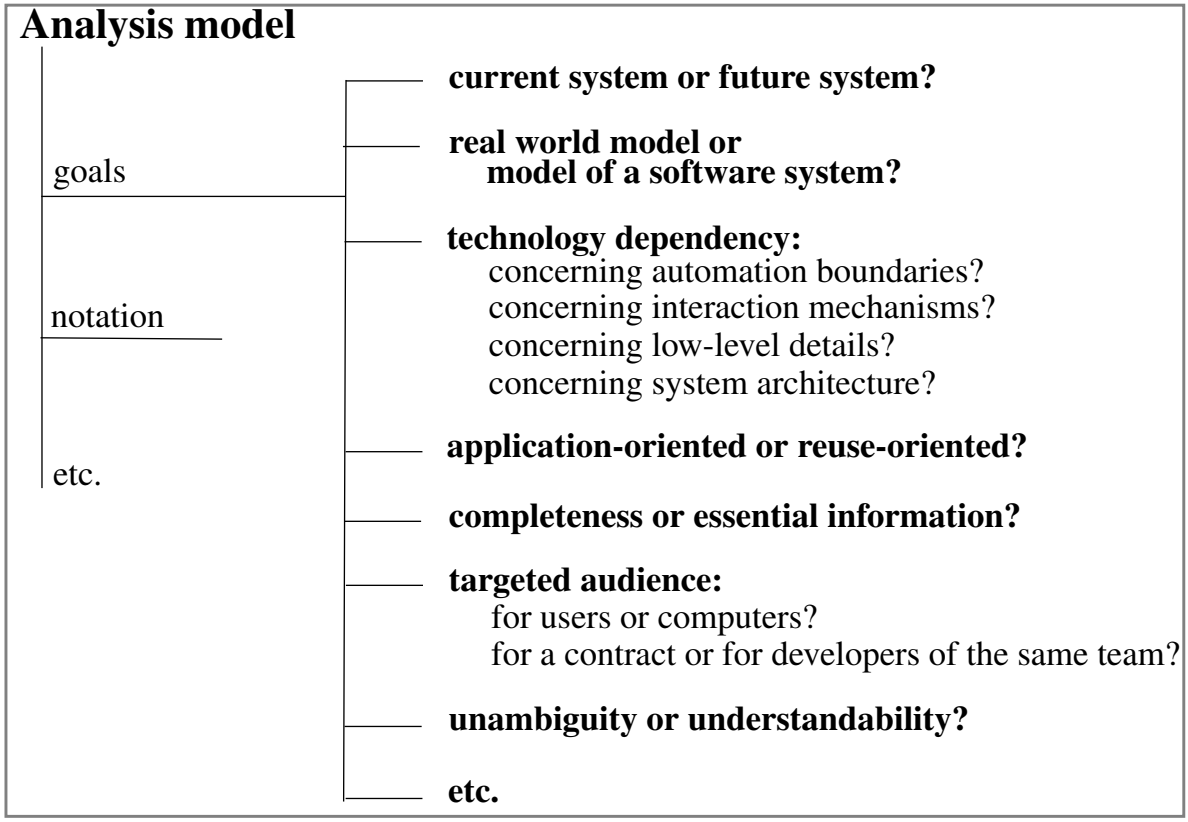


Figure 3: Goals of analysis models

3.1 Model of the current system versus model of the future system

Introduced by DeMarco in 1979 [DeMarco79], the differentiation between the model of the current system and the model of the future system has not lost any importance, though only in rare cases both models are made. Yet it is important to state explicitly which model is the goal, especially when starting by collecting information on the current system in order to model a future system. Even if the objective is only the replacement of an old system or the mere automatisation of manual processes, a new system has the potential of changing substantially current organizations and decision making processes (a list of such changes can be found e.g. in [Huber90]). These differences to the status quo need to be taken into account in an analysis model that should show the requirements or the high-level view of the new system.

3.2 Application-oriented versus reuse-oriented

Methods like Fusion [Coleman94], OMT [Rumbaugh91] or OOSE [Jacobson92] focus on the development of one single application. Their analysis models are thus application specific and have the goal to give a small, accurate, and easily understandable description for one application. If the goal of analysing a problem domain is not a specific application but a library of reusable components or an application framework, then the analysis model should show reusable objects and reusable interaction mechanisms. As is shown in [Berard93][4], application-oriented models differ in their structure from reuse-oriented models.

3.3 "Real world" model or abstracting a software system

An analysis model that reflects the real world respectively models the problem as the user perceives it (also called real world model[5]) is not identical to a model that abstracts a software system (also called target- or solution-oriented model). Differences are e.g. what information is modelled, how data and operations are grouped into objects, how these objects interact with each other, and also how external objects interrelate with the objects within the system. A model that tries to be both, a real world and a target-oriented model, is with the exception of trivial systems either the one or the other. As it is mentioned in [McGinnes92], though many developers pretend to make a model of the reality or of the user's view, often reality is bent to fit into an object model which fulfils quality criteria coming from the software design. Therefore [Høydalsvik93, McGinnes92] emphasize the necessity of a clear distinction. They advocate a model that reflects reality as the user perceives it and is not influenced by any knowledge about what would be good software abstractions. Other methods start right away with a model of the problem domain that uses the abstractions of the software system. Such a model focuses on the targeted software system, and finding good software abstractions is more important than reflecting reality as the user would perceive it before he gets acquainted with the future software system.

4. Reuse-oriented models use inheritance heavily, factor out associations and complex operations into classes of their own, parameterize classes, and provide classes with a complete set of primitive methods. For this, a more complex model must be accepted. In contrast, an application-oriented model describes the characteristics and requirements of exactly one application, and is as small and simple as possible. The focus is on the application specific relationships between perhaps reused objects. The classes only need a sufficient set of methods but may include also complex operations.
5. In [Høydalsvik93] this model is called problem-oriented model. The model describes a certain organisation, business, process or machine without considering how this problem will be reflected in the structure and elements of the software system.

3.4 Technology independent models

Technology independence

Many methods mention that the analysis model should be independent of the design and the implementation of the system, i.e. that technological details should be abstracted away. Formulations like "describing the *what* instead of the *how*" or "modelling problem-oriented instead of target-oriented" also imply a technology independent model. Examples:

- DeMarco suggest to develop logical models of the current and of the future systems which do not contain any physical information [DeMarco79, page 233].
- OMT gives the following guidelines for a technology independent analysis model: "A good analysis captures the essential features of the problem without introducing implementation artifacts that prematurely restrict design decisions." [Rumbaugh91, page 187]. "The successful analysis model states what must be done, without restricting how it is done, and avoids implementation decisions. The result of analysis should be understanding the problem as a preparation for design" [Rumbaugh91, page 148].
- Fusion: "Analysis is about describing what a system does rather than how it does it. Separating the behaviour of a system from the way it is implemented requires viewing the system from the user's perspective rather than that of the machine." [Coleman94, appendix A].
- The analysis model in MSA assumes perfect technology (no errors, no space or time constraints, no costs). Also no differentiation is made between human and automated processes [Yourdan89].
- OOSE: "In analysis an application-oriented specification is developed to specify what the system offers its users. This specification, which we call the analysis model, specifies the functional behaviour of the system under practically ideal circumstances and without regard to a particular implementation environment... It is important, however, to judge whether the analysis model can actually be realized..."[Jacobson92, page 15].
- Sommerville summarises the content of the requirements specification with the following words: "A software requirements specification is a document containing a complete description of what the software will do, independent of implementation details" [Kotonya96].

Many methods do not specify further what exactly they mean with technology independence and from which technologies they want to abstract in the analysis model. For this reason, [Bailer93] differentiates in the context of information systems between several types of technology from which we can abstract in an analysis model:

- *Information or implementation technology*: The system is modelled independently of its future implementation as software or manual system. The system itself with its processors, data-stores and communication paths is considered as perfect.
- *Organisation technology*: The system is modelled independently of the organisational structures found in the present organisation; the organisational environment is considered as being of perfect technology.
- *Production technology*: The system is modelled independently of what it produces. This abstraction does not make any sense in the realm of software engineering, because any software system supports a specific production technology and cannot be modelled without this.

Abstracting away implementation as well as organisation technology brings the great advantage that the whole organisation where a software system should be embedded can be rethought and redesigned, and hopefully also optimized, thus combining system development with business re-engineering. It seems ideal to make an analysis model that abstracts away any technological detail concerning implementation technology and organisation. Yet in practical projects we encounter the following difficulties:

- Even if a new software system is the initiator to redesign the organisation, it is often not efficient to mix the goals of business re-engineering and system requirements modelling; the essential details are too different.
- The metaphor of perfect implementation technology may be useful for batch like information systems. Yet when developing networking software or new user interfaces, the system under consideration would not exist when we assume perfect technology with no automation boundaries. In most projects, the goal is not a model that abstracts away all implementation technology, but abstracts away only certain aspects of the implementation technology.

In practice there is therefore a great amount of uncertainty how to develop and use technology free models in an efficient way. As [Bailer93] mentions, in industry the technology free models are often not made and even many examples found in books are not really technology free. But when we want to describe a model that abstract away only certain technical aspects, we need a further distinction. We propose the following criteria: abstracting the automation boundaries, abstracting the interface and interaction mechanism, abstracting the system architecture, and abstracting low level implementation details.

Abstracting away low level implementation details

When in 1979 DeMarco introduced the differentiation between the physical and the logical model [DeMarco79], it was not yet common to abstract away physical details like file structures, algorithms and help variables. Nowadays, abstracting away low level implementation details in an analysis model, and to some degree even in the design model, is taken for granted by object-oriented development methods. The encapsulation of data and operations into objects, the principle of information hiding and the possibility to define class interfaces by contracts help to create analysis and design models that are not programming language specific and that even do not contain yet all objects necessary for the implementation.

Abstracting away the system architecture

Many methods advocate an analysis model that does not consider the internal system architecture. The internal system architecture is either defined in a separate model after the analysis model (as e.g. in SSP [Jufer92] or OMT [Rumbaugh91]) or it is not dealt with by the method at all (e.g. Fusion [Coleman94]). As long as a method is used for monolithic and simple systems, abstracting away the internal system architecture may appear to be very obvious, a special high-level model that takes into account the system architecture but no other implementation details may be superfluous. For more complex software systems which may include distribution, off-the-shelf software components (like networking software, user interface systems or database systems), or which integrate also hardware components, we certainly need also an analysis model concerning the system architecture, thus having two analysis models with different goals. But not every project starts with an analysis model that abstracts away the system architecture for the following reasons:

- Introducing distribution or off-the-shelf components can change the structure of an object-oriented analysis model totally, causing major remodelling when introducing the system architecture.
- For certain systems all the complexity lies in the system architecture, an analysis model that abstracts this away would be trivial.
- In certain projects the system architecture is already given at the beginning and is not subject to change.

So depending on the concrete circumstances we need to define explicitly for each analysis model if the system architecture is abstracted away or not and which elements of the technical system architecture are taken into account and which ones not.

Abstracting away automation boundaries

To fix the boundaries of the future software system right at the beginning of a project is not only difficult, it can also be very dangerous. In order that the new system really suits the needs of the users, the system boundaries and the organisational aspects of the environment need to be carefully examined and eventually redesigned. An abstraction from all implementation technology is needed. Therefore many methods advocate a universe of discourse larger than the automated system. In MSA [Yourdan89] the automation boundaries are even abstracted away for the whole analysis model (also called essential model), they are introduced during the transition to the design. [Shlaer88] distinguishes between the analysis model, which does not specify which processes will be automated, and the external specification model which focuses on the interactions on the automation boundary. [Cook94] has two analysis models, the essential model which abstracts away any automation boundaries and the specification model which takes them into account. In OOSE and Fusion a problem domain object model is made that is free from the automation boundaries, yet the use case respectively the interface models are made after the automation boundaries have been introduced into the analysis respectively the system object model.

Other methods, especially methods that do not model explicitly the global system behaviour, leave it open to which degree the automation boundaries are abstracted away in the analysis model, often assuming that introducing the system boundaries into the object-oriented object model will not change the structure of the object model. This may be true for certain projects but is not the case in general, as is shown in [Eckert95] for an example using the Fusion method and in [Høydalsvik93] for a model of the OOPSLA conference registration problem.

Abstracting away the low-level interaction mechanisms on the system boundary

When we consider those models that show the interactions between the software system and its environment (scenario models that show the external view of the global behaviour of the system), two questions arise: What does it mean if we state as goal for our model to be technology independent? Which details of the interface can be abstracted away?

As we model the interactions on the system boundary (external events, system services, use cases or scenarios), technology independence certainly does not mean abstracting away the system boundaries. The system boundaries of the software system must be known, but we can decide on which level we want to model the interactions. We can choose a technical level (single user inputs), or we can choose a purely conceptual level (e.g. events that represent whole system functions. We propose that a technology free scenario model should only contain purely conceptual interactions and events.[6] Furthermore, the structure of the scenario model of a technology free analysis must not have any influence onto the scenario model of the design, i.e. dur-

ing the design process the developers are free to change the scenarios arbitrarily, as long as the overall functionality remains the same.

3.5 External versus internal model

When looking at a system, a subsystem, a software component, a process or an object, we differentiate between the internal view which describes the system by its components, and the external view which describes the system as a black box. Both, the external and internal view contain dynamic as well as static aspects.

We can also say that the external view describes the "what" and defines the requirements, whereas the internal view describes the "how" and gives the design of a system, assuming that the external view defines the problem and the internal view a solution. These terms are often used as implying each other; but this is not necessarily the case.

Having two different models

Even when describing only the external view of a component, we need a certain syntax and semantic for this. Basically there are two possible approaches:

- The same notation is used to describe the external behaviour as is used for the internal view. Without additional information you cannot tell if such a model represents an external or an internal view.
- Different notations (and maybe even different paradigms) are used for the external and the internal model. The two models cannot be confused, but two different notations need to be mastered and the transition between the two models may be further impeded.

The advantage of the second approach is the clear separation between the external and the internal view. Due to the different modelling techniques, the models need to be redone when changing the view point. In a top-down approach, where first the external view is modelled and afterwards the internal structure is determined, this may be an advantage. Yet in a bottom-up approach, where we already know the internal components and thus can use these as semantic elements for the external description, it hinders us from having only one model that represents both view points. Here lies the strength of the first approach which works well, as long as the goal of the model is still clear and the external view is not mistaken for the internal view.

Whenever we distinguish between the external and the internal view, we must accept to have two models that look very similar and are about the same system, but differ in their contents and model elements (e.g. an external data model and an internal object model).

Correlations between the internal and the external views

Ideally, the external view does not make any unnecessary restrictions concerning the internal view, and information hiding is ensured. Far too often, this is not the case: the same model elements (e.g. object types and attributes) are used in the internal and the external view.

6. Every scenario model, irrespective of the abstraction level chosen, has a bias to one specific interface design. Therefore a really technology free scenario model cannot exist, we can only define explicitly that the structure of the scenario model of the analysis is not a prejudice for the design model.

3.6 Complete and unambiguous models versus essential models

Often mentioned goals of analysis models are that they confine to the essential information, do not go down into too low details, are unambiguous and complete. But these criteria cannot be fulfilled all at once.

Completeness and unambiguity of object models and scenario models

We might like to consider an object model to be complete, if we have found all the necessary objects and specified all the external views of all the object services. But in order to find all the objects and services we need to determine how the object services work and if they need further services from other objects. Otherwise we cannot verify if all the necessary objects and services are modelled[7]. Furthermore, for an unambiguous model, services names are not enough. The services need to be specified in such precise language. that no misunderstandings are possible. This could be a formal specification language, programming language or an unambiguous domain specific language.

The same is true for the scenario model. Also an unambiguous scenario model would have to use such a language. And in order to be complete, it would have to cover all possible sequences of events, also all exceptional and error scenarios, and not only the most typical ones.

Limits to complete and unambiguous analysis models

There are several points that restrict the development of complete and unambiguous analysis models:

- *Redundancy to the design process*: Total unambiguity necessitates to detail the analysis model (which is not yet the design model) until an unambiguous language level is reached. One of the criticism at SA/SD has been that the work is done twice: first the system is specified down to the level of pseudo code in the process model of SA, modelling the problem domain irrespective of the design of the software system. Afterwards the same level is specified once again in the design, this time for software components. An effort that is normally not justified for every part of the analysis model. Also the effort for a complete analysis model, especially for a complete scenario model, is not always justified by the expected return of investment.
- *Targeted audience*: The language preferred by the targeted audience is not necessarily an unambiguous language. E.g. the scenario models of common object-oriented analysis methods like OOSE, Fusion, OBA and OMT focus at easy understandability, but are not unambiguous.
- *Evolving requirements*: Not in every project all the requirements are known from the beginning and into all details. An unambiguous and complete analysis model can then only evolve with the design and implementation of the system (see e.g. also experience report in [Kolbe95].

Confining to the most essential informations

Many object-oriented methods do not try to provide a complete analysis model. They advocate to focus on the essential aspects. E.g. Booch [Booch94, page 253] distinguishes between primary scenarios (which illustrate key behaviours) and secondary scenarios (behaviour under exceptional conditions); only the first ones are all included in the analysis model. BON [Walden95, page 170] provides only on a very high abstraction level a mere list of all external

7. Most analysis model provide a list of the services offered by an object, but they do not mention which other services are needed to provide theses services. No verification concerning the completeness of the object model is possible.

events and scenarios. A full covering of all possible scenarios on a more detailed level is considered as unrealistic, so only a part of the listed scenarios are also modelled by object diagrams.

But if we specify as a goal of the analysis model to model only the essential information, we have a very vague criterion. On one hand, what "essential" means, depends on the other goals chosen. On the other hand, its interpretation remains always subjective, and must be decided upon in the concrete situations by the developers and reviewers, maybe backed up by risk assessment techniques.

3.7 Targeted audience

To be understood by users (and developers) versus to be understood by machines

If we want that the analysis model is understood by those involved in or affected by the system or problem that is modelled, we have to choose an appropriate language and to model the system in such a way that it reflects the thinking and perception of the users. Ideally such a modelling technique is close to other techniques used in the daily work of the users. Examples for this are the concept maps, event-response-lists and storyboards of [Umphress91], the informal scenarios of [Kilberth93], or the scripts of OBA [Rubin92] (e.g. [Heeg94] likes to work with these scripts, as they are so similar to the tables secretaries work with every day).

On the other hand, there is the desire for an automated verification or even execution of an analysis model. This necessitates a language that can be interpreted by machines, e.g. formal specification languages like Z, B, VDM, CO-OPN or LOTOS. But the resulting analysis models can only be understood by users if these are themselves software engineers. Otherwise the developers first need another analysis model that they can discuss with the users, and based on this they can develop the formal model. Of course, formal specification languages also necessitate that the software developers have the necessary educational background, training and experience for using formal specification languages; up to now this is only true for a minority of project teams.

"Natural" modelling techniques

The targeted audience determines also which modelling techniques are considered as being natural. Certain people consider a purely functional approach as natural [Ward94], others see the world as consisting of passive objects and of processes that manipulate these objects [McGinnes92], for others everything should be modelled as a system of interacting and active objects. Which techniques are perceived as natural is a question of experience, profession, culture, language and personal perspective. As a consequence, not every modelling approach does match well the thinking of the people involved in a specific analysis process, even if it is conceived as being very natural by its originators.

To be used for contracting versus to be used by the same team in an iterative approach

Will the results of the analysis model be used in-house by the same team that has specified the analysis model? Or will the analysis model be used for a contract? And will the analysis model be used for estimating the project size by certain estimation techniques (as e.g. the one proposed by [Moser92])? Also these goals must be known before the modelling process starts in order to avoid bad surprises.

3.8 Consequences

Contradictions

An analysis model cannot fulfil all possible expectations at once. We have already formulated various pairs of opposite goals. But also between these pairs there are dependencies. E.g. an unambiguous model going down to a precise and thus formal language level is not targeted at the user. Application- and reuse-oriented models are always models of the software system. And a system architecture is out of place in a model of the real world. It does not make sense to have goals that contradict each other, neither prescribed by a method description, nor foreseen in the project plan, nor existing implicitly in the heads of the software developers involved in the project.

Many difficulties that arise in the development process are just due to the fact that the goals of the analysis model are only implicit or that contradicting goals are taken over from a method description. As the goals are not defined and discussed explicitly, the effort is undertaken to develop a model that should satisfy contradictory goals. And as this is not possible, frustration is the consequence. But all those difficulties that are due to unclear, contradicting or even inappropriate goals cannot be resolved by an enhanced notation or an improved analysis process.

Using goals for the project plan

In any project, somebody has to define how many different models are to be developed and what their goals are. If this is not done explicitly before the models are constructed, it will be done implicitly. Then the goals are imposed by the person with the greatest influence, or they are determined with much additional effort when arguing about modelling issues which are in fact uncertainties concerning the goal of the model. Far better, the goals are defined by a method, by QA-guidelines and by the project plan (more about general model architecture frames and project plans can be found in [Beringer95]).

Responsibility of the development team

Some of the criteria we have mentioned here can be defined quite precise before the development of the analysis model starts. Other criteria (like e.g. what the essential information is or how detailed the model must be to avoid too much ambiguity but be still not too complex and still understandable by the targeted audience) can only be stated more precisely during the development process. There it is important that the developers and reviewers are aware of the vagueness of these goals. There is no easy guideline that could replace human judgement and human awareness.

4 Intent clashes

Besides the goals mentioned in the previous chapter, we may have further expectations concerning the analysis model and especially concerning the analysis and the design process. Many of the previous goals and of the expectations mentioned here are achievable when the effective developing process (analysis and design processes) is not considered. When it is taken into account, further difficulties arise. We formulate here these difficulties as three misconceptions and two intent clashes, and we also show how the various methods cope with these intent clashes.

4.1 Difficulties with the ideal analysis model

The ideal analysis model

When we are asked for the ideal analysis model and the ideal development process, we might mention the goals and expectations presented in figure 4:

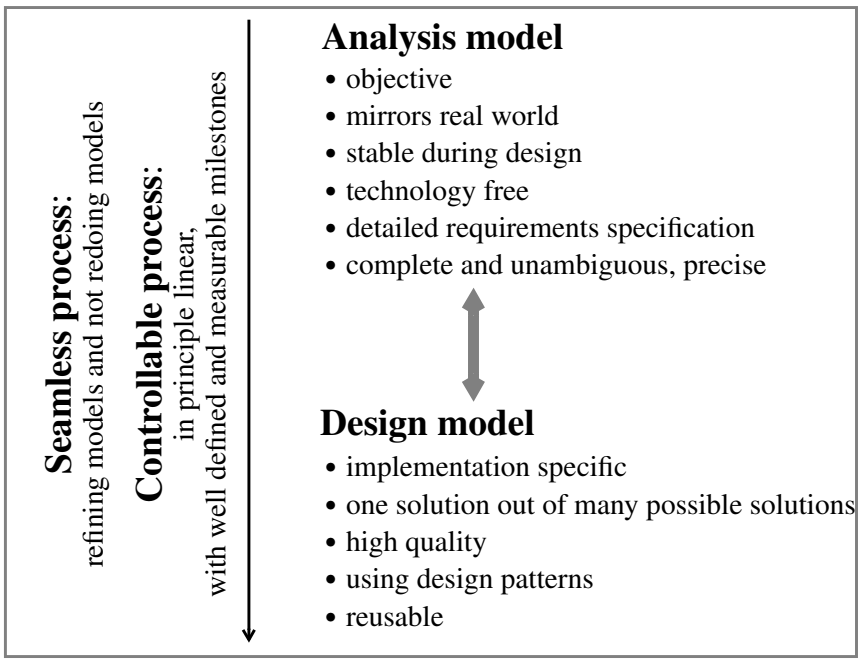


Figure 4: The ideal analysis model

If we specify the goals and expectations in this way, then we have to deal first of all with the problems we mentioned already in the previous chapter: some of these goals are contradictory (e.g. real world model and detailed requirements specification of the software system), and they are not precise enough to be useable guidelines in a concrete project (e.g. technology dependency). But there is also a further difficulty. Some of these expectations are just not achievable. Though it were very nice to fulfil them, they contradict the experiences of software development. In many projects it is not possible to make a stable analysis model, to make an analysis model that is an objective picture of the real world, or to make an initial requirements analysis model that can be seamlessly expanded into a design model.

First misconception: stable model

There exist some software development projects, where the functional requirements are well known from the beginning, and remain stable during the design. But often, this is not the case. On the contrary, there are projects where only a vague idea exists about what the system should do. Evolutionary prototyping or scenario modelling is done in parallel to the design and implementation of the system, in order to find out what the real needs are, and what the requirements should be like. Many people advocate the parallel advancement of design and requirements model, (e.g. [Holbrook90] mentions that the criteria and requirements for a complex design problem evolve at the same time as solutions are being formulated, and [Siddiqi94] quotes: “specification and implementation are intimately intertwined”).

In other projects, the functional requirements are known, or people think they know them, but they refer to the old technology. With a new technology new possibilities arise e.g. for the user interfaces. And these new possibilities not only affect the details of implementation, but also influence the metaphors and work procedures used in the analysis model. Changes in the scenario and object models are thus inevitable.

A further difficulty for a stable analysis model is the immense complexity in large projects. It is not possible to model first all the requirements, and then to proceed to the next phase. Instead, an iterative approach is chosen. First the most basic requirements are picked out, and a first system is made. Incrementally, other requirements are added. Though this approach avoids that the project never proceeds over the analysis phase, the analysis model is never complete, and substantial changes can become necessary as new requirements are added. An example for such an approach to software development is given by Kruchten in [Kruchten95] and [Kruchten96].

Second misconception: objective real world model

It is often said that an object-oriented analysis model models the real world. And it is assumed that in analysis there exists a model that is objective and is free of any design decisions. This assumption is not backed up by everybody. [Floyd89] states that every model is a construction, and that this construction is always subjective, depending on the background of the person who developed or designed it (see also [Siddiqi94]). There is no objective way to look at the world around us, the model is always shaped by the personal perspective. Even if in a project a so called objective real world model is made, it is just the subjective construction of that person in the team with the greatest influence.

Our perception of the real world also changes during a project. Metaphors are found and implemented during the development project, and after some time we may even use these metaphors outside this project when describing similar problems. And when we would make the real world model a second time, we would suddenly use the metaphors that come from the development of a software solution even in the real world model.[8]

Third misconception: initial final high-level view

In a seamless approach the analysis model can be expanded step by step into the design and implementation model (see figure 5). The initial analysis model as it exists after the first phase or milestone of the project is just a subset of the final system model.The same assumption is made for an initial data model that may be part of the analysis model.

Both assumptions hold for text book examples and for trivial systems. But they do not hold when the developers do not know in advance what the structure of the final system will be. If this final system should be of a good quality, then it cannot be a mere expansion of the initial analysis model (see also [Beringer94]).

In reality the model is reworked several times until those design patterns, object abstractions and interaction mechanisms are found that suit the problem, satisfy the quality demands, and will be implemented in the final system. The iterative approach is not a mere expansion and detailing of the model, but includes also changes of the model.

Also the final system model can be shown on various abstraction levels. The higher level views abstract away certain details of the lower level views, but they still reflect the essential

8. These metaphors are often very domain specific, and unfortunately rarely get published. An exception is the “tool-material-aspect” metaphor documented in [Gryczan92].

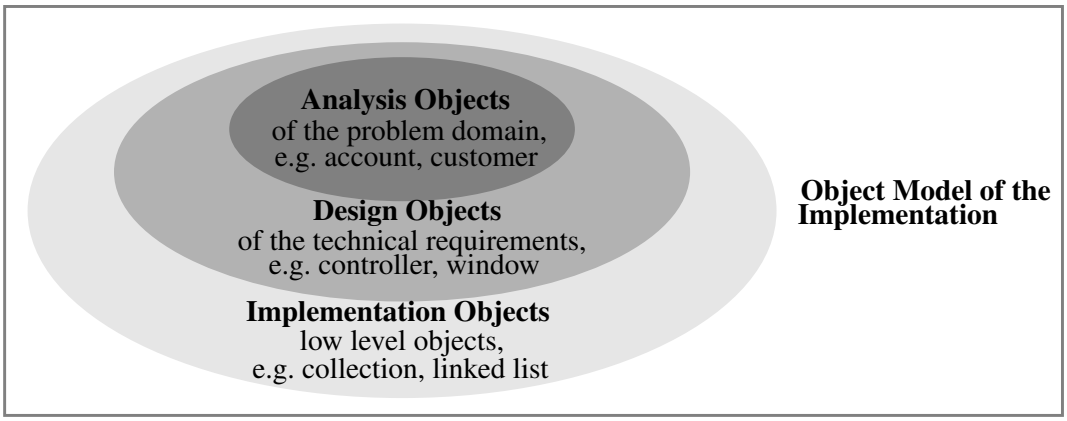


Figure 5: Seamless expansion of the analysis model

structures and concepts of the final model. But this final high-level view is not the same model as the initial analysis model, even if they use similar notations. If we use an approach where we first model the requirements in an analysis model and then develop the implementation, we have two high-level views of the system which are slightly or drastically different from each other (see figure 6). The differences may only lie in the model elements that are used for the description (e.g. the allocation of attributes to object types). Or the difference may even concern the overall structure of the scenario model and the object model.

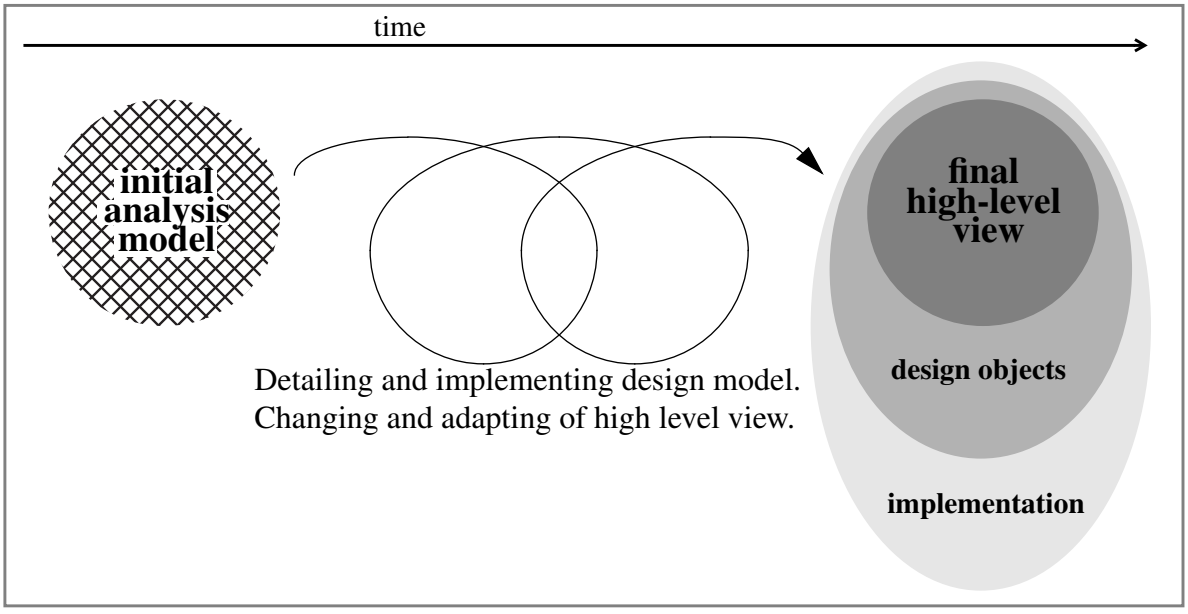


Figure 6: Initial analysis model versus final high-level view

4.2 The two intent clashes

First intent clash: controllable process <--> no stable requirements, no real world model

In industry we need controllable processes with milestones. We must be able to define what we expect from these milestones, and we must measure the results. We need to verify that the required work has been done correctly. This includes that the right work has been done and that

the work does not need to be redone later on. If we cannot guarantee and measure that at a milestone, then project management becomes extremely difficult, and outsourcing and contracting software is a nightmare.

The consequences for the analysis model are straightforward: we want an analysis model that can be completed in a first phase of the project and remains stable afterwards. Further we require that it is a real world model, so the desired content is specified clearly and we can easily decide at the milestone that the model is complete and correct. With such an analysis model and analysis phase, we would have at least at the beginning of the project well defined and controllable milestones. We could also use the analysis model to estimate the size of the remaining work, and to work out contracts for the development of solutions.

As we have shown above, such an ideal analysis model does not exist. We may fake an idealistic linear process in the system documentation (as proposed by [Parnas86]), but that does not work for the process itself. We thus have a clash between the constraints of project management, and the possible goals of analysis models and the needs for a highly iterative, and creative approach (see figure 7). The larger the project, the more difficult it becomes to reconcile these two intents. If we look at the proposals for project management e.g. by [Sadr96] or [Kruchten95] and at other guidelines for iterative object-oriented project management, we can see that they try to find compromises that do not satisfy the "purists" of either side. Their goal is a realistic and cost-efficient process that has some iterations, leaves some freedom for changes and creativeness throughout the development, and is still to some degree controllable.

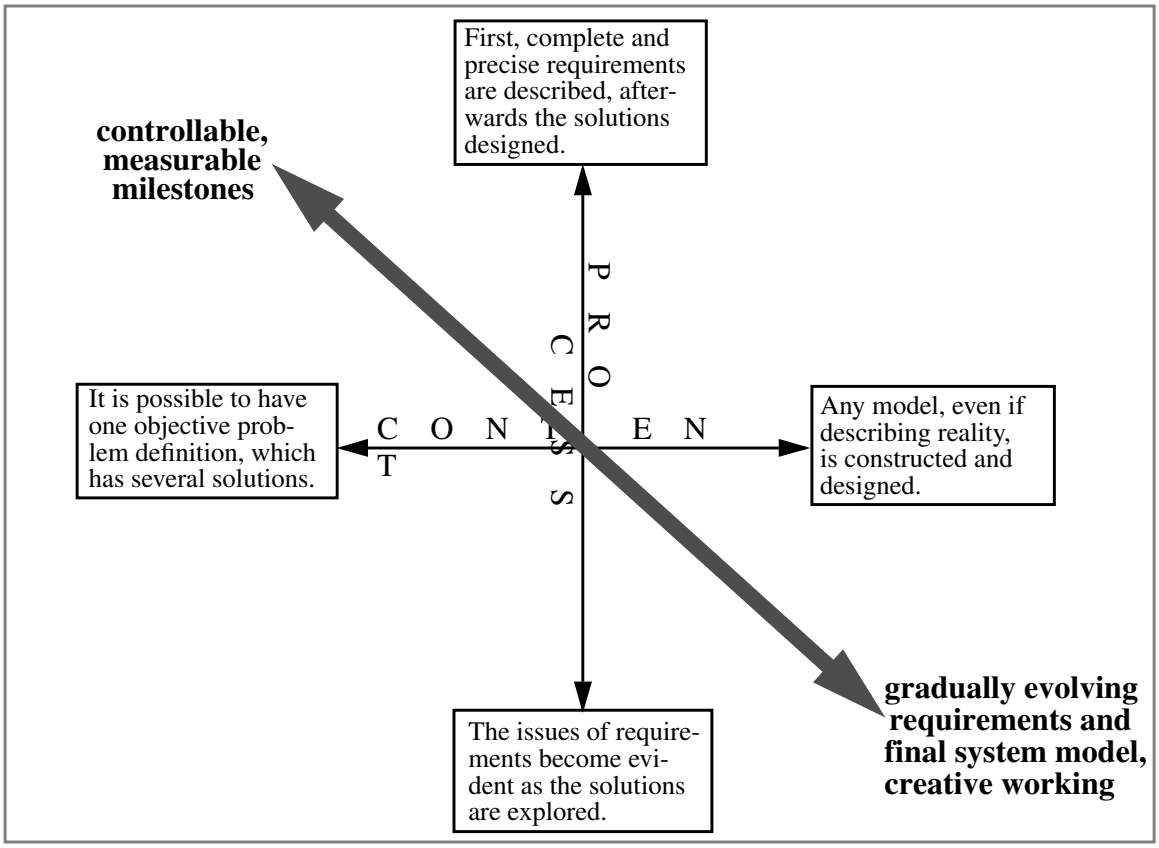


Figure 7: The first intent clash

Second intent clash: seamless process <--> initial final high level view

Even before object-oriented modelling techniques became popular, one tried to find well defined links between the analysis and the design model. There were at least two reasons why a well defined or even better a seamless transition between these models was required:

- A clear connection between the various models is needed in order to enable maintenance.
- Rules (and even automatisms) for deriving the design model from the analysis model and the implementation from the design model are requested, in order to facilitate and control the development process. Often the dream has even been to have an automatic code generation from a problem model.

When object-oriented technologies have been introduced, great hopes arose that with having one paradigm for the whole development process, one would finally have a seamless process. But as we have mentioned before, this works only when modelling the system from a retrospective perspective. So for most projects, the initial analysis model is not identical to the final high-level view. This is very evident in a linear process, when the initial model remains unchanged. But it is also true for an iterative process, only that in an iterative process often the analysis model gets adapted step by step to reflect the concepts found in the final system. So the intent clash between the desire for a seamless process and the impossibility to have an initial analysis model that reflects the final high-level view remains.

4.3 Approaches to handle the intent clashes

The various methods handle the intent clashes differently. We have categorized the methods concerning their treatment of the intent clashes into three groups: methods that have an idealistic approach, methods that just focus on the evolution of the final system model, and methods that bother with developing, reworking and maintaining two different models over the whole development process.

4.3.1 The idealistic approach

The idealistic approach does not really provide a solution to the intent clashes. It just hopes that the clashes are not heavy and can be neglected. A more or less linear approach is chosen, and the goal of the analysis phase is a stable and real world analysis model that defines the functional requirements for the system. At the end of the analysis phase, this model is reviewed and accepted. In the design it is directly enhanced or transformed into the final software model and is thus a high-level view of the design model.

Of course it is admitted that there may be some overlap of the analysis process with the design process and that changes to the analysis model during the rest of the development process cannot be avoided. Yet it is tried to reach the ideal process as close as possible, and to produce at least a documentation that fakes it. Most data-driven or use case driven object-oriented methods fall into this category, though much depends on how they are interpreted and applied in practical projects.

An example: Fusion

In the first phase of the project, the functional requirements are described by an analysis model that consists of the object model and the interface model. The analysis model describes the intended behaviour of the system as it is externally visible, without any implications for the de-

sign and implementation of this behaviour. In the ideal process, the analysis model can then directly be refined and enhanced into the models of the design. Design objects are added and the functions are distributed onto the different objects. The objects and system operations in the analysis model do not only define the external visible behaviour of the system but ideally also correspond to the high-level view (subset of objects, attributes and methods) of the final object-oriented system model. The design model should of course also satisfy the quality criteria for a good object-oriented design.

4.3.2 The one-model approach

In the one-model approach, the goal is always the design model. No analysis model is made, work is started right away at this final model with an iterative and evolutionary approach. In the first versions of this model, a part (or all) of the user's view of the system is modelled on a high abstraction level. In the following versions, the model is gradually reworked and refined, until the final object structure, with the final metaphors, design patterns and object specifications has evolved. No model is kept of the initial user's view, it is assumed that his perspective also adapts to the high-level view of the final software system.

The one-model approach provides quite a seamless process (not by expanding the analysis model but by reworking it) and has only become possible by having the object-oriented paradigm throughout the whole development process. It overcomes the second intent clash by not maintaining any initial analysis model. But its weakness concerns the controllable process. It is very hard to define easy controllable and measurable milestones. This becomes also evident when looking at the methods using this approach: they can only provide quite vague guidelines how the project could proceed, but cannot provide any predefined milestones with well measurable criteria. Some methods even do not mention the difficulties of project control at all, which does not cause any problem as long as they are used in an environment where no project control is required, as is the case for many small or in-house projects.

Another drawback of this approach is that there is often no documentation of the user's view or of the high-level view. Even if it is assumed that the user's view adapts to the vocabulary and structure of the final software system, having only a detailed design model is not enough for the maintenance of the system.

An example: Booch

"Analyse a little, design a little, test a little" is the motto of the development process of Booch [Booch93].In addition to this microprocess, a macroprocess is suggested that helps to bring some structure into the development process. Each phase of the macroprocess consists of several iterations of the microprocess. Beside the static class diagram also other views and diagrams are supported. Yet their purpose is only to help evolving the object model of the final system.

An example: FORAM

"Object-oriented analysis is analysis, but also contains an element of synthesis. Abstracting user requirements and identifying key domain objects are followed by the assembly of those objects into structures of a form that will support physical design at some later state. The synthetic aspect intrudes precisely because we are analysing a *system*, in other words imposing a structure on the domain." (page 2 of training note 2 [Graham93]).

An example: BON

BON uses right from the beginning a notation that represents systems of interacting objects. During the development process the initial model is reworked to improve its structure, and further details are added. BON describes the development process with *activities* and *tasks*. The activities are orthogonal to the tasks, though some occur mainly in the analysis tasks, others mainly in the design tasks. Analysis consists of tasks for gathering information and describing the gathered structure, and is mainly concerned with the problem domain per se (to which degree technical and implementation dependant details are considered right from the beginning depends on the constraints of the project). Yet also analysis is already concerned in finding good object-oriented abstractions, the domain is described solution-oriented. All tasks of the development process as targeted at evolving and specifying the interfaces of the classes. Other models than the class specifications, e.g. the object scenarios, are either discarded as soon as they are no more needed to determine the class interfaces, or they are updated to reflect the final system and kept for documentation purposes. The amount of changes that can affect other views is minimized in that these views do not show much details, e.g. the object scenarios do not indicate the operation and parameter names.

An example: Floyd

In [Floyd89] a constructivistic perspective is the basis for the development process. Therefore the requirements are not analysed, but constructed. The goal is the implementation of the final system, user satisfaction has the highest priority. The participative design allows that users learn from developers, and developers from users, and that together the new system and the new ways how work should be carried out in the future are worked out (see also [Reisin90] and [Schauer93]). This will also result in new metaphors, work patterns and design patterns. But there is no predefined process for reaching this goal. The process evolves as the system evolves.

4.3.3 The two-model approach

Another approach is chosen by those methods that explicitly develop two models, an analysis model for the initial definition of the requirements and a design model for documenting the final software system. The first one reflects the user's perspective as it is at the beginning of the project, and it describes the functional requirements as complete as possible, and in a precise and detailed manner. The notation is chosen in such a way that it mirrors best the thinking of the users (e.g. in [Ward94] this is a purely functional description of the system behaviour), a seamless transition to the design is of low importance. Either the analysis model is finished and reviewed after the first phase of the project, afterwards it is considered as more or less stable, and the user does not participate in the design. Or the analysis model is developed in parallel with the design model, but there is a clear distinction between the analysis activities and the design activities, even different teams may be involved. The analysis model is only adapted to changing or to new requirements, it is not adapted to the structure or model elements of the design model. The design patterns, data structures and scenario structures used in the design do not affect the analysis model.

Most methods that use a two model approach also use different modelling techniques and sometimes even modelling paradigms for analysis and design. Very naturally this prevents from carrying over any modelling decisions from the analysis into the design, the analysis model only serves as an information source. In the design a totally new model has to be made, and the model elements of analysis cannot be directly reused, even if the two models go down to the

same level of detail and precision. The advantage of this approach is a clear separation of catching requirements and designing the system. [Ward94] even considers the missing seamlessness to the design and the change in the notation as an advantage, as it forces the developer to search for a good design model, and prevents him from carrying over the constructions of the analysis into the design. The disadvantage is a high effort for developing and maintaining two different models in different techniques for things that are quite similar. Avoiding changes in the analysis model may also hinder the evolution of a new understanding of business processes, the creation of new concepts and their transfer to the users.

Examples for this approach are MSA-GfAI, MSA as defined by [Yourdan89], and the approaches proposed in [Høydalsvik93], [Holbrook90] and [Ward94].

The best approach...

Which approach is the best one depends on the risks a specific project faces. If the greatest risk is to miss the optimal object-oriented structure in the final system and to get fixed on old work procedures, then probably the one model approach is the best one. If the greatest risk is that the project gets out of control, then maybe one of the other approaches is better. None of these approaches is the silver bullet. The intent clashes cannot be overcome, only weakened, and every project has to fight to some degree with them.

5 Conclusions

We discussed in this report various interpretations of the term "analysis" and we examined if we can use these interpretations to determine the content of the analysis model and the end of the analysis process. We also considered the conflicts that exist among some of the goals used to characterize the analysis model, and we discussed the intent clashes that exist between some requirements of the analysis process and some goals of the analysis model. Finally, we presented three different approaches for dealing with these intent clashes.

Due to the inherent tensions between contradicting requirements, we could not present a set of well defined analysis goals that is applicable for every project. Compromises are project specific, taking into account the project specific risks. But this does not mean that we may omit the explicit definition of the analysis goals in a software development project. Rather a higher awareness of the influence of the goals onto the content of the analysis model is needed.

6 References

[Bailer93] B. Bailer; *Geschäftsmodelle, Verständnis und Folgen der Technologieneutralität*; Institutsbericht Nr. 93.43, Institut für Informatik der Universität Zürich, 1993

[Berard93] E.V. Berard; *Essays on Object-Oriented Software Engineering, Volume1*; Prentice Hall, 1993

[Beringer94] D. Beringer; *Limits of Seamlessness in Object-Oriented Software Development*; Proceedings of TOOLS Europe '94, Versailles, 1994

[Beringer95] D. Beringer; *The Model Architecture Frame: Quality Management in a Multi Method Environment*; Proceedings of SQM'95, Seville, 1995

[Booch94] G. Booch; *Object Oriented Analysis and Design*; Benjamin/Cummings, 1994

[Coleman94] D. Coleman et al.; *Object-Oriented Development: the Fusion Method*; Prentice-Hall, 1994

[Cook94] S. Cook, J. Daniels; *Designing Object Systems, Object-Oriented Modelling with Syntropy*; Prentice Hall, 1994

[Davis90] A. M. DAVIS; *Software Requirements: Analysis and Specification*; Prentice Hall, 1990

[Davis93] A. M. DAVIS; *Software Requirements: Objects, Functions and States*; Prentice Hall, 1993

[DeMarco79] T. DeMarco; *Structured Analysis and System Specification*; Yourdon Press, 1979

[D'Souza93] D. D'Souza; *Comparing OO Analysis and Design Methods*; Proceedings of the OOP 93 in Muenchen

[Eckert93] G. Eckert, P. Golder; *Improving Object-Oriented Analysis*; rapport technique No 93/32, Departement d'Informatique, EPFL, 1993

[Eckert95] G. Eckert; *Improving the Analysis Stage of the Fusion Method*; Fusion in the Real World, edited by R. Malan, R. Letsinger and D. Coleman, Prentice-Hall, 1995

[Floyd89] C. Floyd; *Softwareentwicklung als Realitaetskonstruktion*; Proceedings of the Fachtagung Software-Entwicklung, of the GI, Marburg, 1989

[Graham93] I. Graham; *Financial Object-Oriented Rapid Analysis Method (FORAM), Training Notes 1 to 4*; seminar given at the SBV in Basel

[Gryczan92] G.Gryczan, H.Züllighoven; *Objektorientierte Systementwicklung, Leitbild und Entwicklungsdokumente*; Informatik-Spektrum Nr. 15, 1992

[Heeg94] G. Heeg; *Object Behavior Analysis - Von der Suche nach Objekten*; talk given at a meeting of Choose and INTEC-OO (Bern, Switzerland, 1994)

[Holbrook90] H. Holbrook; *A Scenario-Based Methodology for Conducting Requirements Elicitation*; ACM SIGSOFT, vol 15, no 1, 1990

[Høydalsvik93] G.H. Høydalsvik, G. Sindre; *On the Purpose of Object-Oriented Analysis*; Proceedings of OOPSLA 93, 1993

[Huber90] G. P. Huber; *A Theory of the Effects of Advanced Information Technologies on Organizational Design, Intelligence, and Decision Making*; Academy of Management Review, Vol 15, No 1, 1990

[Jacobson92] I. Jacobson; *Object-Oriented Software Engineering*; Addison Wesley, 1992

[Jufer92] R. Jufer; *Strukturierte Spezifikation*; seminar of the GfAI Gruppe für angewandte Informatik AG, 1992

[Kilberth93] K. Kilberth, G. Gryczan, H. Zuelligkoven; *Objektorientierte Anwendungsentwicklung: Konzepte, Strategien, Erfahrungen*; Verlag Vieweg, 1993

[Kolbe95] K. Kolbe; *How do you know you are building the "right" software? Experiences with Use Cases*; Workshop on "Requirements Engineering: Use Cases and More", OOPSLA 95, 1995

[Kotonya96] G. Kotonya, I. Sommerville; *Requirements engineering with viewpoints*; Software Engineering Journal, January 1996

[Kruchten95] P. B. Kruchten; *The 4+1 View Model of Architecture*; IEEE Software, November 1995

[Kruchten96] P. B. Kruchten, C. J. Thompson; *Iterative Software Development for Large Ada Programs*; Proceedings of Ada-Europe'96, Montreux, June 1996

[McGinnes92] S. McGinnes; How Objective is Object-Oriented Analysis?; Proceedings of CAiSE'92, 1992

[Moser92] S. Moser; *Metrics and Estimation, Aufwandschätzungen in Informatik-Projekten*; seminar of the GfAI Gruppe für angewandte Informatik AG and of IBM Switzerland, 1992

[Neighbors80] J.M. Neighbors; *Software Construction Using Components*; Technical Report 160, Department of Information and Computer Sciences, University of California, Irvine, 1980

[OMG92b] *Object Analysis and Design, Volume 1: Reference model, Draft 7.0*; OMG, 1992

[Parnas86] D.L.Parnas and P.C.Clements; *A Rational Design Process: How and Why to Fake It*; IEEE Trans. on Soft. Eng., Vol 12, No. 2, Febr. 1986

[Rawsthorne95] D. A. Rawsthorne; *Transaction Based Analysis*; Workshop on "Requirements Engineering: Use Cases and More", OOPSLA 95, 1995

[Reisin90] F.Reisin; *Kooperative Gestaltung in partizipativen Softwareprojekten*; Dissertation, Technische Universität Berlin, 1990

[Rubin92] K.S. Rubin, A. Goldberg; *Object Behavior Analysis*; Communications of the ACM, Sept. 1992

[Rumbaugh91] J. Rumbaugh et al.; *Object Oriented Modelling and Design*; Prentice-Hall, 1991

[Rumbaugh94a] J. Rumbaugh; *The life of an object model*; JOOP March-April 1994

[Sadr96] B. Sadr, P. J. Dousette; *An OO Project Management Strategy*; IEEE Computer, vol29 no9, 1996

[Schauer93] H. Schauer, B. Kuhnt; *Partizipative Software-Entwicklung*; Institutsbericht Nr. 93.35, Institut für Informatik der Universität Zürich, 1993

[Shlaer88] S. Shlaer, S. J. Mellor; *Object-Oriented Systems Analysis, Modeling the World in Data*; Yourdon Press, 1988

[Siddiqi94] J.Siddiqi; *Challenging Universal Truths of Requirements Engineering*; IEEE Software, March 1994

[Umphress91] D. A. Umphress, S.G.March; *Object-oriented requirements determination*; in JOOP Focus on A&D, edited by R. S. Wiener, SIGS Publications, 1991

[Waldén95] K. Waldén, J.M. Nerson; *Seamless Object-Oriented Software Architecture*; Prentice Hall, 1995

[Ward94] R.Ward, J. Stevens; *Object Orientation Is Not Always Best!*; Proceedings of Ada in Europe, Copenhagen, 1994

[Webster] *Webster's Ninth New Collegiate Dictionary and Webster's Collegiate Thesaurus*; NeXT Digital Version, 1988/1992

[Wilkinson95] N. M. Wilkinson; *Using CRC Cards: an informal approach to object-oriented development*; Prentice Hall, 1990

[Wirfs90] R. Wirfs-Brock et al.; *Designing Object-Oriented Software*; Prentice Hall, 1990

[Yourdon89] E. Yourdon; *Modern Structured Analysis*; Prentice Hall, 1989