Requirements Engineering: An Overview *

Klaus Pohl

Informatik V, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Germany

pohl@informatik.rwth-aachen.de

1 Introduction

Traditionally, requirements engineering (RE) has been seen as the first phase of the software life cycle in which a specification is produced from informal ideas. During RE, the functional and non-functional requirements to be met by the system, as well as the criteria for measuring the degree of their satisfaction, must be elicited and documented in a *requirements specification*. If the specification describes both hardware and software, it is called *system requirements specification*; if it describes only software, it is called *software requirements specification* (cf. [IEEE-830, 1984]). The process of developing a requirements specification has been called *requirements engineering* (RE). Since the establishment of RE as a distinct field in the mid 1970s (see [TSE, 1977]) a great deal of progress has been made.

Nowadays, RE is seen as a key issue for the development of software systems with the responsibility for maintaining the requirements of a system over time and across traditional and organizational boundaries (cf. [Jarke and Pohl, 1994; Loucopoulos and Karakostas, 1995]). Correct understanding (elicitation), documentation (specification) and validation of user/customer needs are becoming more and more crucial as the ultimative measurement for systems quality is the degree of user satisfaction, i.e. the ability of the system to meet the user needs. Thus, RE is becoming the essential activity within

* A version of this paper appears in Encyclopedia of Computer Science and Technology, Volume 36, Marcel Dekker, Inc., New York

the software life cycle in which a variety of stakeholders must be involved. The growing importance of the field is also reflected in new international RE conferences and symposia started in the 1990s (see [Fickas and Finkelstein, 1993; Davis and Siddiqi, 1994; Harrison and Zave, 1995]).

RE as a discipline is still immature. It is commonly accepted that only *what* a system should do has to be defined in a requirements specification and *not how* it should do it. Whereas almost everybody agrees that requirements must be elicited, specified, and validated/verified little uniformity is reached in the terminology[1] of the activities performed (cf. [Davis, 1988]). For example, the understanding of *what* the problem being solved is without defining *how* it will be solved is called requirements analysis [Charette, 1986; Wasserman *et al.*, 1986] but also problem analysis [Davis, 1990], problem definition [Roman *et al.*, 1984] and requirements definition [Berzins and Gray, 1985]. Another indication for the immaturity is the existing vast amount of literature covering distinct facets and individual isolated contributions to philosophical and technical problems of the area.

To provide an overview of the field, we first reflect on some definitions of RE (Section 2) and typical products of the RE process (Section 3). Despite of the fact that the knowledge about the RE process is poor, four tasks to be performed can be identified (Section 4).

Towards a common understanding we define RE as a process of *"establishing vision in context"* (Section 5). For the case of information systems, which are increasingly becoming an integral part of our everyday life, we use a framework for structuring the context, namely the four worlds of information systems (*development, usage, subject,* and *system* world). The RE process itself can be characterized by three orthogonal dimensions, namely the *agreement, representation,* and *specification* dimension (Section 6). These dimensions reflect that RE is faced with social, technical, and cognitive problems. The consequences of these definitions on the RE process, its products, and requirements traceability are outlined in Section 7. In Section 8 we briefly summarize the contributions made and sketch the future perspective of RE.

2 Definitions of Requirements Engineering (RE)

Unfortunately there is no common definition of RE and existing definitions differ in their focus. For example, some definitions concentrate on the elicitation of requirements and thereby focus on the interaction with the user, whereas the focus of others is more on the documentation (specification) of the requirements. In general, recent definitions tend to cover more aspects of RE than older ones.

In a recent book, RE is defined in accordance to [Pohl, 1994] as

> *"a systematic process of developing requirements through an iterative co-operative process of analyzing the problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of the understanding gained." [Loucopoulos and Karakostas, 1995, p. 13]*

This definition reflects the fact that RE has to deal with representational, social, and cognitive aspects (see Section 6; cf. [Pohl, 1994]).

A status report characterizes RE as

> *"all activities which are related to*
>
> - *identification and documentation of customer and user needs*

[1] An overview on different terminologies proposed can be found in [Davis, 1993, p. 22].

- *creation of a document that describes the external behavior and the associated constraints that will satisfy those needs*
- *analysis and validation of the requirements document to ensure consistency, completeness and feasibility, and*
- *evolution of needs. " [Hsia* et al., *1993, p. 75]*

and thereby, also with a different stance, covers aspects of the elicitation, validation, and specification of requirements.

The IEEE standard [IEEE-610.12, 1991] defines RE as requirements analysis:

> *"(1) The process of studying user needs to arrive at a definition of system, hardware, or software requirements. (2) The process of studying and refining system, hardware or software requirements."*

The term "requirement" is defined as

> *"(1) A condition or capability needed by a user to solve a problem or achieve an objective.*
> *(2) A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification or other formally imposed document.*
> *(3) A documented representation of a condition or capability as in (1) or (2)."*

In contrast, Gause and Weinberg [Gause and Weinberg, 1989] see RE as

> *"... the part of development in which people attempt to discover what is desired";*

and the process of developing requirements as

> *"a process of developing a team of people who*
>
> (1) *understand the requirements;*
> (2) *(mostly) stay together to work on the project;*
> (3) *know how to work effectively in a team."*

Obviously, the above definitions share some common aspects, e.g. gaining a definition of what a user needs or desires. Nevertheless, they differ in focus. Therefore some aspects of RE are only covered by a subset of the definitions, e.g. co-operation, change management, validation.

3 Requirements Engineering Products[2]

It is widely agreed that the primary product of the RE process is the requirements specification which should state *what* a system should do and not *how* it should do it. The fact that RE should focus on the essence of the system in contrast to possible incarnations (implementations) of the system was made popular by the book of McMenamin and Palmer (cf. [McMenamin and Palmer, 1984]).

A requirements specification should be complete, consistent, modifiable, traceable, unambiguous, verifiable, and usable during development, operation, and maintenance of the system (cf. [IEEE-830,

[2] In Section 7.2 a more comprehensive definition of the RE product is given.

1984]). Moreover, a specification should state both the *functional* and *non-functional* requirements of the system.[3]

However, existing requirements specifications vary in structure, content, and the representation formats used. One reason for this is the coexistence of many standards and guidelines which define the content and the structure for a "good" requirement specification document in different ways. Another reason is the influence of the RE method (or methods) used during the process.

To provide an overview on common RE products we first sketch the various outlines of a "good" requirements specification proposed by the standards and guidelines (Section 3.1) and characterize the underlying modelling perspectives of common (quasi standard) RE methods (Section 3.2).

3.1 Standards and Guidelines for Requirements Documentation

An excellent overview of existing standards and guidelines for requirements specifications can be found in [Dorfman and Thayer, 1990]. Since the different standards and guidelines vary a lot in scope and content, a comprehensive requirements specification model, called RSM, for the specification of information systems was proposed (cf. [Gibbels, 1994; Pohl, 1996b]). In the following we provide a brief overview of RSM and characterize the coverage of the 21 requirements specification standards and guidelines using RSM.

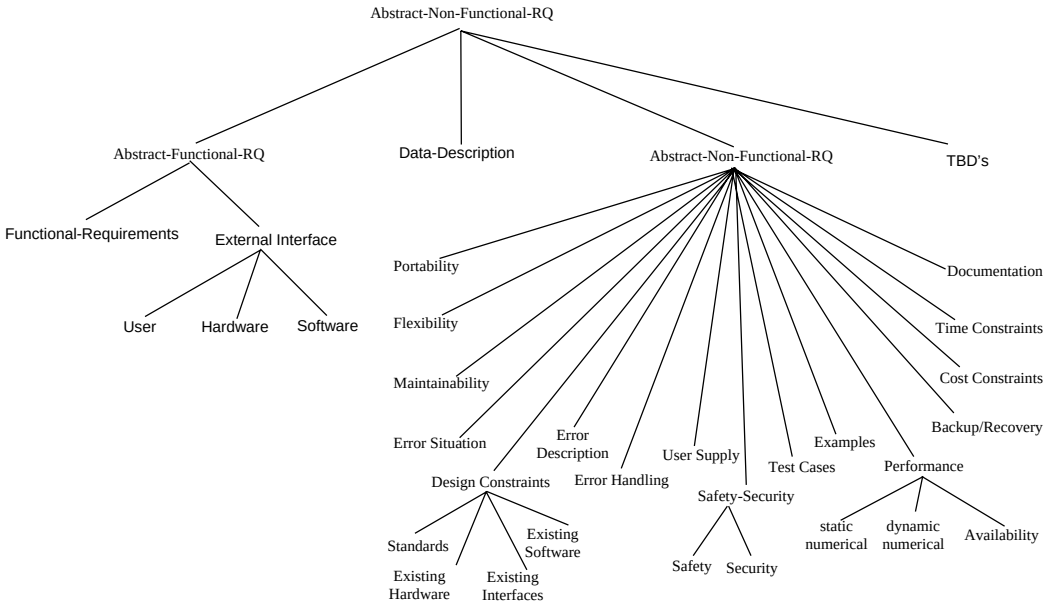

Fig. 1 The specialization hierarchy of RSM

To avoid redundancy, RSM was defined using the well known abstraction mechanisms *classification, aggregation, specialization (inheritance),* and *attribution*. Consequently, for each type of requirement a class definition was established (*classification*). The characteristics of each type are defined by the attributes of the corresponding class (*attribution*). The classes themselves are organized in a specialization hierarchy (*specialization*) shown in figure 1. *Abstract requirements classes* were defined which provide common attributes of the corresponding subclasses but cannot be directly instantiated

[3] There have been several discussions whether there should be a differentiation between functional and non-functional requirements and if so what the differences are. Whether a requirement is classified as functional or non-functional, depends on the viewpoint people have. What is functional for one person, can be non-functional for another one. For example, the shape of a sky scraper is a non-functional requirement for the structural engineer, whereas it could be a functional one for the town planning engineer, e.g. if the sky scraper should serve as a point of interest or even as the town attraction (see [Pohl *et al.*, 1994b] for details).

RSM - FUNCTIONAL REQUIREMENTS

Requirement \ Standard	IEEE-STD-830-1984	BS 6719, 1986	CSA 243.15.4-1979	FIPS PUB 101	ESA PSS-05-0 Issue 1	SMAP DID-P100	SMAP DID-P200SY	SMAP DID-P200SW	SMAP DID P210	JPL D-4003	JPL D-4005	FIPS PUB 38 FKT	FIPS PUB 38 SYS	DoD 2167A	DI-MCCR-80023	DI-CMAN-80008A	DI-MCCR-80025A	DI-MCCR-80026A	DoD-7935A FKT	DoD-7935A SYS	Starts Guide Vol.I
Abstract Requirement																					
ID	+						**o**	**o**									-				
Name	+		**o**								**o**						-				
Description	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+	+
Creator																					
Date of Creation																					
Relation			**o**				**o**	**o**		**o**				**o**			**o**		**o**	**o**	**o**
Priority	+	+			+		-	-		**o**				**o**				-	**o**	-	+
Stability	+																				
Acceptance Criteria		+		-	+	-				**o**			-				**o**				+
Alternatives		+																			
Abstract-Function-RQ																					
Inputs	+	+	**o**				-	-		**o**	-	**o**			-		**o**		+	+	+
Source	+	+	**o**								**o**						**o**	-		+	
Volume	+	+								**o**		**o**					**o**		+	+	
Format	+	+								**o**		**o**					**o**	-	+	+	
Range	+	+								**o**		**o**					**o**	-	+	+	
Errors			**o**																		
Outputs	+	+	**o**				-	-		**o**	-	**o**			-		**o**		+	+	+
Target	+										**o**						**o**	-		+	
Volume	+	+								**o**		**o**					**o**		+	+	
Format	+	+								**o**		**o**					**o**	-	+	+	
Range	+									**o**		**o**					**o**	-	+	+	
Accuracy		+	**o**							**o**		**o**	-				**o**	-	+	+	
Errors	+																				
External Interface	+	+	**o**	-	**o**	-	**o**	**o**	-	**o**	**o**	**o**	-	-	-	**o**	**o**	**o**	+	**o**	+
Time Behaviour																			+		+
Description of External System	+																				
User Interface	+	+			**o**	-			-		-					**o**		**o**	+		+
Type	+																				
Monitor Layout	+																				
Language		+																			+
Hardware Interface	+		**o**		**o**	-			-		-					**o**		**o**	+		+
Software Interface	+		**o**		**o**	-			-		-	**o**				**o**		**o**	+		
Functional Requirement	+	+	**o**		**o**		-	-		**o**		**o**									+
Processing	+	+	**o**		**o**		-	-		**o**					-						+
Validation of Inputs	+		**o**									**o**									
Operations	+	+					-	-													
Subordinate Functions	+																				
Validation of Outputs	+		**o**									**o**									
Error Handling	+	+																			**o**
Error Tolerance																					
Errors	+		**o**					**o**		**o**									+		+
Data Description			**o**			-	-	-		**o**		+					+	**o**	+		+
TBD	+				+					+	+										

"+" detailed description; "o" described; "-" only mentioned; " " not mentioned

Table 1 Influences of the standards and guidelines on RSM: functional requirements.

(e.g., the class *Abstract-Functional-RQ* for functional and the class *Abstract-Non-Functional-RQ* for non-functional requirements). As shown in figure 1 RSM distinguishes between functional requirements,

RSM - NON-FUNCTIONAL REQUIREMENTS

Requirement \ Standard	IEEE-STD-830-1984	BS 6719, 1986	CSA 243.15.4-1979	FIPS PUB 101	ESA PSS-05-0 Issue 1	SMAP DID-P100	SMAP DID-P200SY	SMAP DID-P200SW	SMAP DID P210	JPL D-4003	JPL D-4005	FIPS PUB 38 FKT	FIPS PUB 38 SYS	DoD 2167A	DI-MCCR-80023	DI-CMAN-80008A	DI-MCCR-80025A	DI-MCCR-80026A	DoD-7935A FKT	DoD-7935A SYS	Starts Guide Vol.I
Abstract-Non-Functional-RQ																					
Performance	+	**o**	**o**	-	**o**	-	**o**	**o**		**o**	**o**	**o**				**o**	-		+	+	+
Static Numerical	+	+			-		**o**	**o**				**o**	-	**o**					+		+
Dynamic Numerical	+	+			-		**o**	**o**		**o**		**o**		**o**			-		+	+	+
Availability	+	+			-	-	**o**	**o**		**o**						**o**					+
Time Constraints		+																			
Cost Constraints		+			**o**							-		-					+		
Test Cases					**o**									-			**o**				
User Support		+												**o**		**o**					+
Documentation																					+
Examples		+																			
Backup/Recovery	+	+				-	**o**	**o**											+		+
Standards																					
Maintainability	+		**o**		**o**	-	**o**	**o**		**o**	**o**					**o**					
Standards																					
Variability		+	**o**		**o**	-													+	+	
Flexibility		+	**o**									**o**	**o**			**o**			+		
Planned												**o**	**o**								
Impossible																			+		
Portability	+	+				-	**o**	**o**		**o**	**o**					**o**	**o**				
Planned																					
Impossible																					
Security	**o**	+	**o**		**o**	-	**o**	**o**		**o**	**o**	**o**	-	-		**o**	-		+		
Prohibit of Communication	+																				
Cryptograhical Techniques	+						**o**	**o**													
Access Mechanism			**o**				**o**	**o**													
Auditing	+																		+		
Standards																					
Safety	**o**	+	**o**		**o**	-	**o**	**o**		**o**	**o**	**o**	-	-		**o**	-		+	+	+
Checksum	+																				
History	+																				
Standards																					
Design Constraints	+	+	**o**	-	**o**		**o**	**o**		**o**	**o**	**o**	-	-			-		+	+	+
Standards	+	+	**o**	-						**o**				-		-				+	+
Hardware	+	+			**o**		**o**	**o**				**o**	-	-					+	+	+
Software		+	**o**		**o**		**o**	**o**				**o**							+	+	+
Existing Interfaces	+																				+

"+" detailed description; "o" description; "-" only mentioned; " " not mentioned

Table 2 Influences of the standards and guidelines on RSM: non-functional requirements.

data descriptions, non-functional requirements and TBD's (To Be Determined), i.e. known gaps in the specification. The specialization hierarchy indicates that the standards and guidelines (on which RSM is based) differentiate between various kinds of non-functional requirements whereas all functional requirements are treated the same.

The RSM classes, their direct attributes, and the influence of the various standards and guidelines on the RSM model are depicted in table 1 (functional requirements) and table 2 (non-functional

requirements). The RSM classes and their direct attributes[4] are shown at the left side of each row. Whereas for the class *Functional-Requirements* over 30 attributes (including inherited ones) have been defined, the class *Abstract-Non-Functional-RQ* and its subclasses have only a few attributes. In other words, according to the standards and guidelines functional requirements must be defined more precisely than non-functional ones.

Every column of the tables is associated with one standard. The influences of the standards on the RSM classes and their attributes are characterized by four symbols. The symbol "+" indicates that the standard describes the related attribute or class very clear and extensive, "*o*" indicates that the attribute or class is briefly described by the standard, "–" indicates that the requirement is just mentioned but not described by the standard, and " " indicates that the standard does not even mention the requirement. For example, the *IEEE-830* standard provides a very comprehensive definition for specifying the inputs of functions (indicated by the "+" symbol at the intersection of *Input-Description* and *IEEE-830*). In contrast, the errors which may occur (*Input-Errors*) are not even mentioned by this standard.

It is interesting to observe that, e.g., *IEEE-830, BS-6719,* and *STARTS-Guide* provide comprehensive definitions for the functional part of a requirements specification, whereas the non-functional part is almost neglected. Looking at table 1 and 2, rows can be detected which are not related to any standard, e.g. the attribute *Creator* of the class *Abstract-Requirement*. These attributes and classes were added during the definition of the model based on different RE literature, e.g. [Roman, 1985; Meyer, 1985] (cf. [Gibbels, 1994] for details).

3.2 Common RE Methods/Products

In contrast to the standards and guidelines, RE methods (e.g. Entity Relationship approaches, Structured Analysis approaches, object-oriented approaches) do not only define the type of knowledge to be captured in the requirements specification. In addition, they provide a more or less detailed description of a standard way of working for eliciting and documenting the requirements. Most existing RE methods[5] focus on a particular modelling perspective.[6] In this section we sketch the three main perspectives: data (Section 3.2.1), function (Section 3.2.2), and behavior (Section 3.2.3). Typical for the seventies and early eighties was the dichotomy of data and functional modelling approaches resulting in many discussions about the pros and cons of data versus functional modelling techniques. Nowadays, both aspects are seen as equally important for modelling systems under different perspectives (cf. [Davis, 1993; Loucopoulos and Karakostas, 1995; Nuseibeh *et al.*, 1994; Pohl, 1996b]). In addition, the need for modelling the behavior of the system is commonly accepted. Recently proposed object-oriented methods offer some kind of (weak) integration of the three modelling perspectives (Section 3.2.4) and provide additional abstraction mechanism.

3.2.1 Modelling the Data of the System

The main purpose of the majority of batch systems in the (early and mid) seventies was data transformation. Thus, modelling the data of the system was seen as an essential activity. To abstract from the data structure used by the system a large variety of *semantic data models* (and methods) were proposed.[7] These models can be traced back to entity-relationship-attribute notations (originated by [Chen, 1976]) or object-role formalisms [Smith and Smith, 1977]. Briefly, the purpose of semantic data models is to define the data of the system independent from the physical structure of the underlying database (file) system at an abstract (intentional) level. Typical modelling concepts (cf. figure 2)

[4] Each RSM class inherits the attributes of its superclass. The class hierarchy is depicted in figure 1.
[5] A good overview of current methods can be found in [Davis, 1993]
[6] Also called viewpoints, e.g. [Finkelstein *et al.*, 1992; Nuseibeh *et al.*, 1994; Maiden *et al.*, 1995]
[7] An excellent overview on semantic data modelling is given in [Hull and King, 1987].

are therefore objects (often represented as named rectangles and called entities), relationships for representing relations between objects (named rhombus), cardinality of the relations (min:max notation between the rectangle and the rhombus), and attributes (named ovals) by which the objects and relations can be specified in more detail. Recent models provide additional concepts for the well known abstraction principles of generalization and aggregation. Besides the implementation independence the resulting data models are closer to human perception than the data structures.

Obviously, this modelling perspective was (is) preferred by software engineers, programmers and database managers/administrators.

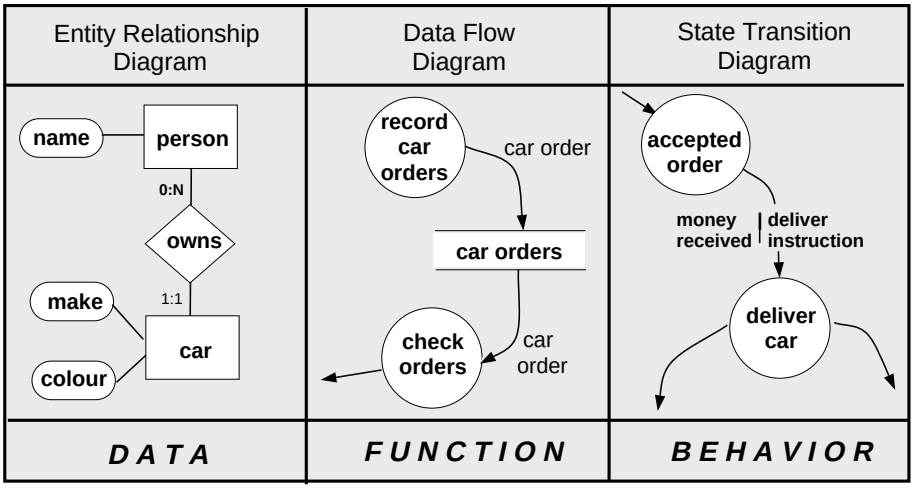

Fig. 2 Typical notations for modelling the data, function, and behavior of a system.

3.2.2 Modelling the Functions of the System

In contrast to the data modelling approaches functional modelling approaches focus on the integration of the system in the organization, i.e. on the functions performed by the system and their contribution to, e.g., business processes.

Function oriented approaches like Structured Analysis [Gane and Sarson, 1979; DeMarco, 1979; Yourdon, 1989] or PSL/PSA [Teichroew and Hershey, 1977] describe the system in a hierarchy of functions (so called processes, bubbles, activities, transformations) and thus focus mainly on the functional decomposition of the system. The most general (abstract) function(s) of the system is defined at the root of the hierarchy, whereas the leaf nodes of the hierarchy describe concrete (the least abstract) functions. Most of the function oriented methods use a combination of data flow diagrams (DFDs) and data dictionaries. DFDs have been used for specifying various systems prior to the advent of computer systems. They can be used for specifying the functions of a company, an organization, a computer system or any combination of them. DFDs (cf. figure 2) consist of data transformations, called processes (named bubbles), data flows (named arrows), static data storage (two parallel lines, named), and sources/destinations of the data, called terminators (named rectangles; not shown in figure 2). In addition, so called mini-specifications are defined in structured English for all processes which are not refined, i.e. for functions which are defined at the lowest level of the hierarchy. Whereas the hierarchy of functions is organized in different levels of DFDs, the content of the various data stores and data flows is defined in the data dictionary.

3.2.3 Modelling the Behavior of the System

The third perspective to be considered during RE is the behavior of the system. The purpose of behavioral modelling techniques is to provide conceptual formalisms for specifying what inputs are expected by the system, the outputs to be generated by the system dependent on the current state and the inputs received, and the relationships that exist between those inputs and outputs.

A system is viewed as a artefact with a certain behavior and functionality. State oriented specification techniques like finite state machines [Hopcroft and Ullman, 1979], statecharts (an extension of finite state machines introduced by Harel [Harel, 1987]), or Petri-Nets (e.g., [Reisig, 1991]) are used for specifying the behavior of the system. A state machine is a hypothetical machine which can be in only one of a given set of states at any specific time. A machine in the state (*S-old*) generates (*G*) an output (*O*) in response to an input (*I*) and changes (*C*) the state (*S-new*). The output produced and the new state (*S-new*) depend only on the old/current state (*S-old*) and the input (*I*) received (i.e. *S-new = C(S-old, I)* and *O = G(S-old, I)*). Common notations for defining the behavior of the system with finite state machines are state transition diagrams (STD, cf. figure 2) and state transition matrices (Mealy and Moore notations). In STDs a circle denotes a state, a directed arrow between two states denotes a potential transition, and the label of the arrow denotes the input which triggers the transition and the output produced by the transition (both separated by a slash).

Besides finite state machines, the behavior of the system is often described using decision trees and decision tables. Moreover, various extensions for DFDs have been introduced which enable the combination of functional and behavior modelling by opposing a control flow on DFDs (e.g. [Ward, 1986; Hatley and Pirbhai, 1987]). Overviews on techniques for modelling system behavior can be found in [Davis, 1988; Davis, 1993].

In the past (till mid eighties) behavioral approaches were mainly used for specifying real time application.

3.2.4 Object-Oriented Modelling Techniques

Nowadays, all three aspects explained above are seen as (more or less) equally important for specifying a system. For example, a flight booking system has to handle a large amount of data about various subjects like costumers, airplanes, or flights. This data must be consistent at each time, persistently stored, and shared with other systems. In addition, the system has to react in real time (to satisfy the customer) and support the business processes of the flight companies and the travel agencies in an adequate manner. Since within the three modelling perspectives (data, function, behavior) requirements about the system are stated which partially overlap, the perspective must be somehow integrated (related). Recently proposed object-oriented approaches [Shlaer and Mellor, 1988; Coad and Yourdon, 1990; Rumbaugh *et al.*, 1991; Booch, 1991; Jacobson *et al.*, 1992] offer some kind of (weak) integration of the three aspects and provide abstraction mechanism like classification, aggregation and specialization (inheritance).

Information local to an object is typically encapsulated by some kind of object description. Such a description considers

- the structural perspective: attributes (properties) of the object, methods provided by the object and static relationships between the objects (aggregated in the object diagram);
- the behavioral perspective: the object lifecycle and the events to occur in the lifecyle specified for each object (aggregated in, e.g., a state transition diagram for each object);

- and the process perspective: dynamic and temporal relations between objects, triggering of operations (methods) by some events, synchronization between events (aggregated by data and control flow diagrams).

These aspects are treated in a variety of ways depending on the origins of a particular object-oriented technique. According to Davis [Davis, 1993] object-oriented techniques originate from the following four areas: object-oriented design (e.g. [Booch, 1991]), data base design (e.g. [Shlaer and Mellor, 1988]), requirements analysis (e.g. [Coad and Yourdon, 1990]), and structured analysis (e.g. [Yourdon, 1989]). Comparisons of object-oriented techniques can be found in [Fichman and Kemerer, 1992; de Champeaux and Faure, 1992; Monarchi and Puhr, 1992; Stein, 1994; Lenzen, 1994].

4 Requirements Engineering Processes

Despite the fact that RE has happened many times in practice, little is known about the process itself. Existing methods like *modern structured analysis* [Svoboda, 1990; Yourdon, 1989], *JSD* [Camaron, 1986], *SREM* [Alford, 1980], *entity-relationship modelling* [Chen, 1976], *information engineering* [Martin, 1990], and even *object-oriented* methods [Shlaer and Mellor, 1988; Coad and Yourdon, 1990; Sutcliffe, 1991; Rumbaugh *et al.*, 1991; Booch, 1991; Jacobson *et al.*, 1992] only roughly describe the way of producing a specification. Formal approaches[8] for requirements specification like *ALBERT* [Dubois *et al.*, 1994], *Larch* [Wing, 1987], *Telos* [Mylopoulos *et al.*, 1990; Jeusfeld, 1992], *VDM* [Bjoerner and Jones, 1988], or *Z* [Spivey, 1990] have a clear definition of syntax and semantics for representing the requirements, but say nothing about the elicitation and documentation of the requirements using the formal language. Thus, the (so called) formal methods focus mostly on representational aspects and provide hardly any guidance for the development of a specification.

Current methods are either top-down oriented or bottom-up oriented. Methods of the first category (e.g. *structured analysis* [DeMarco, 1979]) assume that the RE process starts with a very abstract description of the current and/or future reality. The initial (abstract) model is then put in more concrete forms during the process. In contrast, bottom up oriented approaches (e.g. *ethnography* [Sommerville *et al.*, 1993]) start with observations about the real world (concentrate on the instance level) and build abstract descriptions from the observations as the process proceeds. Although it is obvious that both approaches bear unique advantages and are therefore essential for developing a specification, methods offering an integration of both approaches are still missing. Moreover, existing methods ignore the fact that RE is an iterative process in which the RE team *learns* about the current and/or future reality, e.g. the customer gets an understanding of his/her *real* needs (cf. [Gause and Weinberg, 1989; Jarke and Pohl, 1994]). In other words, existing methods do not support the integration of changes into an existing specification.

Despite the heterogeneous picture of RE processes given above four tasks to be performed can be identified, namely the *elicitation*, the *negotiation*, the *specification/documentation*, and the *verification/validation* of requirements. Figure 3 depicts the four tasks, their relations as well as some potential players in the RE process.[9] Typically, a requirement is first *elicited*. In a second step the various stakeholders *negotiate* about the requirement, agree on it or change it accordingly. The requirement is then in the *specification/documentation* task integrated with the existing documentations and finally in the *validation/verification* task checked if it corresponds to the original user/customer needs (adapted to the limitations opposed on the requirements process by constraints) or conflicts with other documented requirements. In following we describe the main goals of the four tasks and their relations.

[8] Comparisons and descriptions of formal methods can be found in [Olle *et al.*, 1988; Zave, 1990; Wing, 1990; Davis, 1993].

[9] In Section 5.2 we give some guidelines for drawing the right people in the RE team.

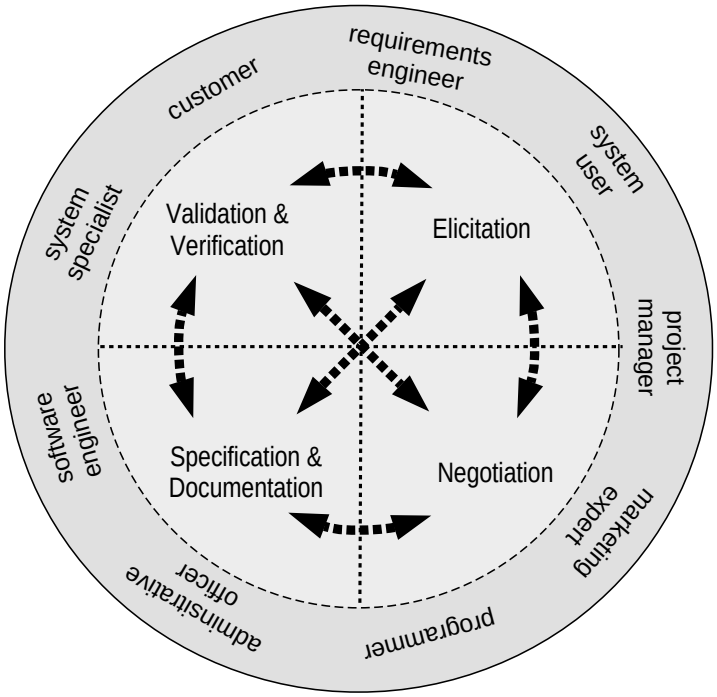

Fig. 3 Four tasks of the RE process.

4.1 Elicitation of Requirements

The first thing you have to do if you have to solve somebody else's problem is to find out more about it [Gause and Weinberg, 1989; Loucopoulos and Karakostas, 1995]. Thus, every RE process somehow starts with the elicitation of the requirements, the needs, and the constraints about the system to be developed. As depicted in figure 3, the elicitation of requirements/needs is an ongoing activity.

Most often, the relevant knowledge about the problem (system) is distributed among many stakeholders, laws, standards and even hidden in existing systems; i.e. the knowledge is not available from one source, e.g. a particular user or customer. Therefore, the identification of the relevant sources and appropriate consideration of them during the elicitation task is essential. Adding to complexity, quite often the knowledge is available in a variety of representations (notations) which range from mental models through pictures, sketches, and natural language descriptions to formal models of the problem domain.

Independent of the sources and representations the goal of the requirements elicitation task is to make the hidden knowledge about the system (problem) explicit in a way that everybody involved in the process is able to understand it. To achieve this goal a variety of techniques for requirements elicitation exist[10], e.g. interview techniques (e.g. [Gause and Weinberg, 1989]), scenario and goal based approaches (e.g. [Benner *et al.*, 1993; Potts *et al.*, 1994], natural language acquisition (e.g. [Rolland and Proix, 1992; Miriyala and Harandi, 1991]), form based acquisition (e.g. [Mannino and Tseng, 1989]), simulation and prototyping (e.g. [Dhar and Jarke, 1985; Luqi, 1993]), ethnography (e.g. [Sommerville *et al.*, 1993; Goguen and Linde, 1993]). In general, domain knowledge which defines the kinds of requirements to be specified for a certain system or a particular functionality should be used to guide the elicitation process (cf. [Adelson and Soloway, 1985; Loucopoulos and Champion, 1988; Johnson and Feather, 1990; Reubenstein and Waters, 1991; Maiden, 1992]). Moreover, to reduce costs and to avoid specification errors existing specifications (or parts) should be reused wherever possible (cf. [Sutcliffe and Maiden, 1990; Maiden, 1991; de Antonellis *et al.*, 1991; Maiden and Sutcliffe, 1992; Constantopoulos *et al.*, 1995]).

[10] A description of the various elicitation techniques can be found in [Loucopoulos and Karakostas, 1995; Gause and Weinberg, 1989].

4.2 Negotiation about Requirements

The goal of the negotiation task is to establish an agreement on the requirements of the system among the various stakeholders involved in the process. Since the people involved have different backgrounds and responsibilities they have conflicting goals and aims. Technically, *view points* [Finkelstein *et al.*, 1992; Finkelstein *et al.*, 1993; Maiden *et al.*, 1995] seem to provide appropriate concepts for representing the different views on the system. Unfortunately, existing conflicts are most often not explicitly stated, i.e. conflicts are often unknown. The goal of the negotiation task is therefore threefold.

First, conflicts must be made explicit and purely emotional conflicts should be avoided. A good facilitator can help to prevent most of the inessential (emotional) conflicts which often arise in meetings [Gause and Weinberg, 1989]. Technically, design collaboration support systems (e.g. [Klein, 1993]) provide a means for supporting the detection of possible conflicts. They are mostly based on black board approaches and design tools by which the interrelation of different structures can be made more visible and thus the user can detect conflicts easier. On the representational level, conflicts can be automatically detected and semi-automatically resolved using view integration and viewpoint resolution techniques (e.g. [Leite, 1989; Leite and Freeman, 1991]).

Second, the negotiation task must assure that for each known conflict the relevant alternatives, the argumentations, and the underlying rationales are made explicit. Good cooperation and collaboration between the people involved in the RE process is a prerequisite for being able to elicit all relevant alternatives and argumentations. Technically, conflict management systems (cf. [CERA, 1994; AAA, 1994]), successfully applied in other areas, can be adapted to support conflict detection and conflict resolution in RE.

Third, the negotiation task must assure that the "right" decisions are made, i.e. that, based on the known argumentations always the best alternative is chosen. Technically, the decision process can be supported by simple voting systems [Jarke *et al.*, 1992] or by multi criteria decision support systems [Bui, 1987; Hwang and Lin, 1987]. Even quality assurance methods like quality function deployment (QFD) [Hauser and Clausing, 1988] may be adopted to RE to facilitate good decision making. Coherent support for all three goals mentioned above, as suggested by negotiation support systems (cf. [HIC, 1994; HIC, 1995]) which are based, e.g., on game theory or sociological approaches, is currently not in sight.

More important than the technical support for the negotiation task is to involve the right people at the right time. In other words, the questions *who* (e.g., user or maintenance people) should be involved in *what* (e.g., user interface definition, functional specification) and *how* (e.g., meeting, interview, brain storming) must be continuously answered. If an important stakeholder was not involved at the right time it is quite likely that the resulting requirements are subject to revision, i.e. are unstable. But even if always the right people have been drawn into the negotiation process the decisions made are often revised; mostly due to a better understanding of the problem later on in the process. Thus, the recording of the decisions and their rationales has proven quite useful (cf. [Conklin and Begeman, 1988; Potts and Bruns, 1988; Jarke and Pohl, 1992; Ramesh and Dhar, 1992; Greenspan *et al.*, 1993; Ramesh, 1993; Pohl, 1996b]).

The negotiation task is not a straightforward process but a cooperative process [Macaulay, 1993] in which people must communicate, exchange their different opinions and arguments and, of course, at some stage make decisions about the requirements to be met by the system.

4.3 Specification/Documentation of Requirements

Commonly, the derivation of a (as formal as possible) requirements specification to be used in

subsequent development stages or for procurement is seen as the main goal of the specification and documentation task. Most of the RE approaches and the standards/guidelines for defining requirements specifications (Section 3.1) assume that the output of the specification/documentation task is one monolithic model (document) which covers all requirements to be met by the system.

From our point of view it is more appropriate to see the output of the specification task as a whole bunch of models which

- consider the viewpoints of the various stakeholders;
- represent not only the final specification but also intermediate results;
- are traceable and consistent.

Thus, the documentation task has to deal with a wide variety of models expressed in various representation formats (notations) which must be kept consistent (cf. [Greenspan, 1984; Bigelow, 1988; Garg and Scacchi, 1990; Fraser *et al.*, 1991; Miriyala and Harandi, 1991; Johnson *et al.*, 1992; Haumer, 1994; Pohl and Haumer, 1995; Pohl, 1996b]), e.g. formal models for the software engineers, graphical models for the manager, natural language description or templates (forms) for the users and customers.

The specification/documentation of the requirements can be supported by using formal specification languages (e.g., PAISLey [Zave, 1991], VDM [Bjoerner and Jones, 1988; Hoare, 1990], Z [Spivey, 1990]) or knowledge representation languages (e.g., ERAE [Hagelstein, 1988], RML [Greenspan, 1984; Greenspan *et al.*, 1994], Telos [Mylopoulos *et al.*, 1990; Jeusfeld, 1992]). They offer the advantage of automatic reasoning and therefore enable the requirements engineer to detect gaps and inconsistencies in a specification (cf. [Borgida *et al.*, 1985; Meyer, 1985; Loucopoulos and Champion, 1988; Wing, 1990]). But applying them to RE is not straightforward since there are many situations in which inconsistencies in the specification should be tolerated (cf. [Balzer *et al.*, 1978; Hall, 1990; Balzer, 1991; Feather and Fickas, 1991; Nissen *et al.*, 1996]).

In most cases, the input for the specification/documentation task comes from the elicitation task (e.g., user statements or descriptions of the old system) and negotiation task (e.g., the solution of a conflict or the revision of an earlier decision), typically in various formats which must be converted into the format used for specifying the requirements. Correct transformations of the input received into a specification requires feedback from various stakeholders, e.g. to resolve inconsistencies and ambiguities. Another major goal of the specification/documentation task is the consistent integration of changes, e.g. revision of a requirement or decision, in all existing specification/documentation models. A prerequisite for the consistent change integration is the creation and maintenance of cross-references within the specification produced as well as between the specification and the inputs received from the other tasks; i.e. traceability between the various requirements models must be established (cf. [Flynn and Dorfmann, 1990; Gotel and Finkelstein, 1993; Ramesh, 1993; Ramesh and Edwards, 1993; Pohl, 1996b]).

As depicted in figure 3 the specification/documentation task can also receive information from the validation/verification task, e.g. detected inconsistencies in the specification model or new requirements detected during validation. Vice versa, the specification/documentation task can initiate the execution of an elicitation task or a negotiation process. For example, the detection of conflicting requirements could lead to a negotiation about the conflict whereas the need of additional information for the specification of a particular requirement could initiate an elicitation process.

4.4 Validation/Verification of Requirements

The main goal of this task is to validate and verify the specified requirements. Boehm (cf. [Boehm, 1984]) has provided a succinct expression of the difference between validation and verification. He

defines verification as *"am I building the product right?"* and validation as *"am I building the right product ?"*. In other words, the purpose of the verification task is to check the specification according to formally defined constraints, whereas the purpose of the validation task is to certify that the specified requirement are consistent with the user/customer intentions. The need for validation/verification appears whenever a (set of) requirement was specified. Thus, validation/verification is an ongoing activity performed for the final specification as well as for intermediate results.

There are two kinds of validation/verification tasks. On the one hand, a specification can be checked for internal consistency. Such activities are most often based on formal verifications [Alford, 1980; Greenspan, 1984; Jarke, 1993; Flynn and Dorfmann, 1990; Greenspan *et al.*, 1994] or on validations using walkthrough or inspection techniques [Fagan, 1986; Yourdon, 1989; Freeman and Weinberg, 1990; Bush, 1990]. On the other hand, a specification can be (externally) validated with the user, the customer or/and other stakeholders. Techniques used for external validation include (a) interactions with users, customers, domain experts, etc. (e.g., [Gause and Weinberg, 1989; Leite and Freeman, 1991; Benner *et al.*, 1993], (b) prototyping (e.g., [Dhar and Jarke, 1985; Henderson, 1986; Hallmann, 1990; Puncello *et al.*, 1988; Luqi, 1993]), (c) checking the specification models against domain knowledge (e.g., [Fickas and Nagarajan, 1988; Puncello *et al.*, 1988; Maiden, 1992], (d) conducting experiments and analyzing the results (e.g., [Gause and Weinberg, 1989]), (e) natural language paraphrasing (e.g., [Rolland and Proix, 1992]), and (f) animation or simulations (e.g., [Dubois *et al.*, 1994]). Independent of the verification/validation technique used the aim of the validation task is to ensure that the right problem is being tackled at any time in the RE process.

The techniques and approaches mentioned in the last sections were assigned to the four tasks due to their major impact. However, many of them support more than one task.

5 Requirements Engineering: Establishing Visions in Context

In the preceding sections we have reflected on the definitions of RE and have sketched typical modelling perspectives and RE products (specifications). Finally, we have identified and characterized four main tasks of the process, namely elicitation, negotiation, specification/documentation, and validation/verification. The description given so far indicates that the area of RE is far away from a common understanding. For example, there exists no method which supports all four tasks described in the last Chapter.

Towards a common understanding we define RE in this section as a process of *"establishing vision in context"* and divide the context of information systems into four worlds (*system, usage, subject,* and *development* world). Using an example we explain the four worlds framework and sketch the expected benefits. We then define a three dimensional framework for the RE process which takes place in the development world (Section 6).

5.1 The System Vision

The tasks described in the last section consume of course resources and are not at all inexpensive. Thus, like any expensive activity, RE is not conducted out of the blue. Instead, every RE process is triggered by some need for changes induced by either perceived opportunities or threats, or by political decision making. The need for change is typically stated in a simple manner which we call the *system vision*. A good example is John F. Kennedy's "send a man to the moon before the end of the decade and bring him safely back again". Thus, RE is not an undirected analysis process, but a process of transforming a vision into a requirements specification which can then, in the design and implementation tasks, serve as a framework for making the necessary changes in the real world.

Many habits exist within the world in which the vision has to be realized. Some are based on formally stated goals, policies, or competing visions[11]. Others are just regularly observable phenomena for which no predefined structure or reasons are known a priori. Thus, on the one hand, relevant habits must be analyzed and the goals, policies, and visions behind them must be made explicit. This is essentially a goal-directed abstraction process of existing practice [Dardenne *et al.*, 1992]. On the other hand, the new vision must be established as a mission in the existing context. Propagation of their consequences leads to the detection and resolution of conflicts among different viewpoints [Finkelstein *et al.*, 1993]. During this process, the vision is often shifting. Therefore, many projects appoint a *vision holder* to make sure that the vision does not get totally lost in the constraints of current practice.

Establishing a vision in an existing context remains an empty phrase if we do not understand what parts of the real world are relevant and how these parts are related to the development process. Due to the diversity of RE, we need some kind of domain ontology in order to provide a basic understanding of what RE is concerned with. This ontology should have a very simple structure to be acceptable to a broad spectrum of developers in that it is easily understood and does not overly constrain them during the RE process.

5.2 Structuring the Context of Information Systems: Four Worlds

Each type of system has its own typical context. For example, in the case of information systems the output is typically used by humans to fulfill a task better whereas in the case of embedded systems the output of the software system is used by other systems, e.g. in an airplane the output of the navigation system is used by the auto pilot system to adjust the course. Due to these differences the context of RE depends on the type of system to be built. In the following we focus on information systems which are increasingly becoming an integral part of our everyday lives.

An information system can be described in analogy to a sharable telescope through which a user community observes a domain of interest more effectively than without it (cf. [Jarke, 1990]). The domain of interest may or may not overlap with the user community itself and it may or may not be changeable by the user community. But it makes sense to distinguish, from a cognitive as well as a social viewpoint, between the *usage* world, the *subject* domain world, and the *system* world, and to describe their relationships (cf. figure 4). A fourth world, the *development* world, has the basic task of assisting the vision holder in realizing the vision in the context of the other worlds. In addition, the development world must consider its internal development context of people, methods, experiences, and tools. It is the world in which the RE process takes place.

Each world is associated with certain groups of stakeholders who should be included in requirements decision making. The framework makes the social prediction that different areas of expertise, different languages, and different interests exist and need to be integrated. Moreover, it predicts certain role-bound non-functional goals which can be associated with the relationships between the worlds: dealing with such standard viewpoints resolution and negotiation tasks become a natural target for RE methods and tools.

Finally, we can expect that representatives from each world will have implicit or explicit models about their own as well as the other worlds, which leads to further social, cognitive, and technical problems. We have found that these models are (and should be) quite different for each of the four worlds (cf. [Jarke and Pohl, 1993a]).[12]

Let us elaborate these general observations for each of the four worlds (cf. figure 4).

[11] Where the various goals come from, their role, and how they are used within the RE process is discussed in [Jarke and Pohl, 1993b].

[12] This observation is exploited in the design of the NATURE environment described in [Jarke *et al.*, 1993].

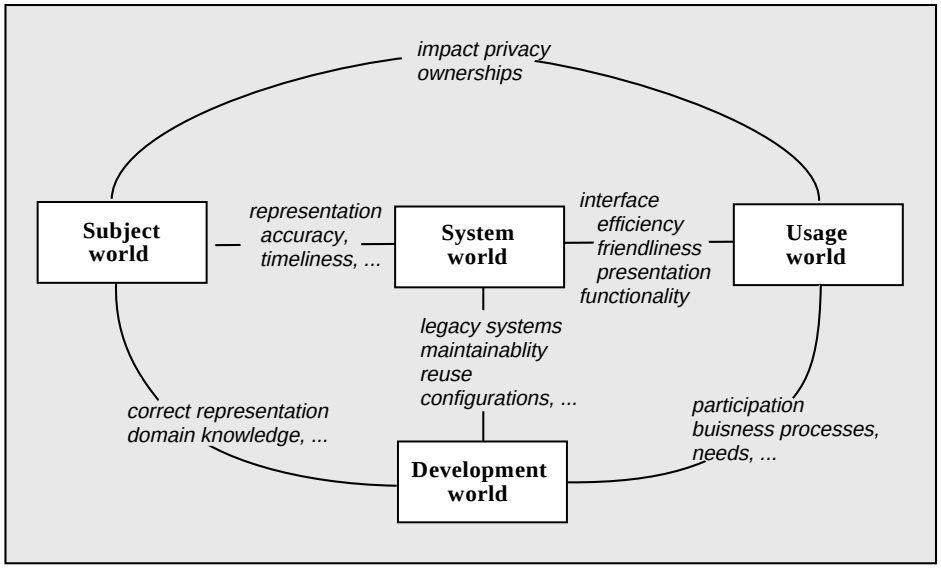

Fig. 4 The *four worlds* of information systems modelling

The *subject world* is the domain the system is intended to maintain information about and is traditionally studied in database design. Stakeholders are the subjects being represented (e.g. in a criminal record or hospital system), or people who have stakes in these subjects but are not system users (e.g. owners of real estate about which information is managed by a brokerage system). Relationships to the usage world are often governed by legal concerns such as privacy and ownership. Relationships to the development world are frequently difficult to establish, since subjects know nothing about the degree to which they are administered by systems and often do not influence the system development until it is too late. Therefore domain experts must participate in the RE process to ensure correct considerations of the subjects. The separation between the subject world and the system world enables to distinguish between the evolution of the real world (the subjects, often called universe of discourse) and its description in the system. The relations between the subject world and the system world define who and what is controlled by the system. They can be described by quality-of-information factors such as accuracy or timeliness.

The *usage world* comprises stakeholders who are owners and direct and indirect users of the system. The relationships between users and owners can vary widely, but might be defined in the organizational structure. As mentioned before, it is important to draw the right people of the usage world into the RE team to assure that all relevant business goals and needs can be elicited and are accordingly considered during the specification of the system. With respect to the system world, quality-of-interaction factors such as response time, user friendliness, and rich functionality are of interest. Such factors influence the representation of the information within the system, e.g. to assure a certain response time, as well as the presentation of the information (or aggregations of them) to the user, e.g. displaying condensed information in a diagram to facilitate understanding.

The *system world* is represented either by people involved in the operation and/or maintenance of the system, or simply by the observed fact that it is very hard to change, either due to its internal complexity or to its established relations to user and developer communities. Existing systems provide an excellent source for requirements for the new system. In addition, expensive rework can be avoided by analyzing existing defects of the system as well as by learning from error corrections made during the life-time of the system. Due to diversity and evolution in the other subworlds, a system often exists

in different versions. It may be part of different usage environments, and may have lost much of its initial structure by changes before the present "vision" came up.

All of this must be considered in the fourth world, the *development world* in which the RE process takes place. Based on the context partitioning expressed by the other three worlds, the people to be involved in the RE process can be identified. The process starts with a vision holder who establishes the vision in the social context through communication with other people, typically by drawing them into a project team for a certain period of time. This team becomes a social reality of its own and is responsible for the integration of the vision in the existing context. It must ensure adequate observation (cognitive aspect) and representation (technical aspect) of the other three worlds and adequately consider resource constraints as well as competing role-bound and individual goals. The RE process itself is characterized and explained in detail by a three dimensional framework introduced in section 6.

The *vision for change* can come from any of the worlds. It can be driven by a technology push in the system world (e.g. moving from files to databases), changes in the subject domain (e.g. protests by privacy pressure groups or new theories about the subject domain suggesting knowledge reorganization), or long-term development concerns (improved maintainability, change of development responsibility from computer professionals to end users). Most frequently, of course, visions arise in the usage world (application pull). The four worlds framework is important to position the vision within the context, and to predict where the main obstacles in the RE process may come from and how they can be overcome.

5.3 Establishing Visons in Context: An Example

In this section we provide an example to clarify the ideas of the four worlds framework as well as the view of RE as a process of establishing a system vision in an existing context. Suppose the overall system vision was *"to develop an information system for supporting customers in their selection of a suitable car"*.

Whether or not an object must be considered during system development depends on the system vision. For example, given the above vision objects of the real world (the context) like turtles or clocks are not considered during system development (cf. figure 5).

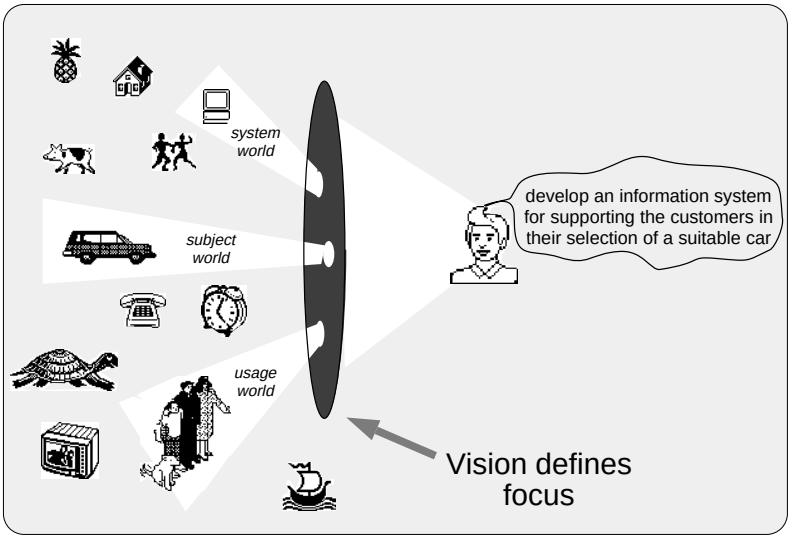

Fig. 5 The overall system vision distinguishes subject, system, and usage world.

Moreover, the system vision defines the boundaries of each world. In our example, the *system world* consists of the cars, whereas the customer of the car dealer (the user of the system) obviously belongs to the *usage world*.

Note that the four worlds are not necessarily disjoin. Depending on the system vision, an object can belong to more than one world. For example, if for selecting a particular car type information about the customer is used by the system (e.g. preferred make like Volkswagen) the customer belongs to the subject world (since information about him is stored by the system) and the usage world.

To identify the other objects belonging to the usage world and to define the system world, the system vision has to be put in a more concrete form, e.g. *"the system has to be developed for the car dealer."* If the system is to run at the car dealer's, the car dealer's existing systems, certain policies for developing information systems, the people of the computer department, etc. form the *system world*. If it is going to run at home by the customer, the TV-set of the customer could belong to the system world.

In our example, the RE team of the development world (cf. figure 6) might consist of the car dealer's system manager (system world), the car dealer's manager, a representative of the sales department (subject world), a customer representative (usage world), and a specialist for human computer interfaces (development world).

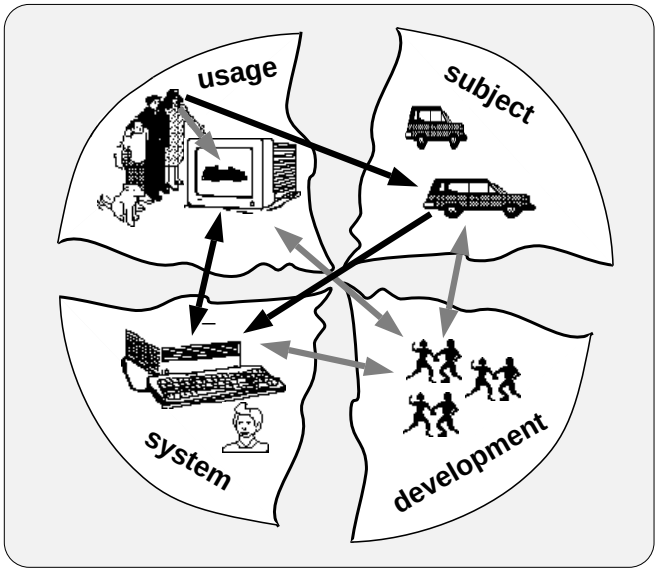

Fig. 6 Four possible worlds of a car information system

The relations between the subject, usage, and system world can be characterized as follows: first, information about cars is conceptualized and represented in the system using some kind of notation (depicted by the black arrow between the subject and system world; cf. figure 6). Second, the system presents information to the user based on the representation (conceptualization) as well as certain user inputs (double black arrow between system and usage world). Third, the presentation is interpreted and associated with the real world by the user. Hence, the presentation must ensure that the user is able to identify the "right" parts (objects) of the real world (black arrow between usage and subject world). For example, a set of pictures of the car or even a video can be used as presentations to ensure that the customer can identify the corresponding real world object. It is important that the object which was conceptualized (the particular car shown in the subject world of figure 6) and the object identified by

the user based on the displayed information of the system are identical, i.e. the presentation should assure that the family in figure 6 associates the information shown on the screen with the right car type.

Within the development world, information about the other three worlds is captured and maintained (as depicted by the three light gray double arrows). Such information may include:

- the maximum speed of the car, its weight, the horse power, the available colors, ability to transport dogs (subject world)
- available hardware, design methods, existing systems (system world)
- sales strategies, suitable presentations, embedding of the system in the organization (usage world)

Obviously, this information is never stable. For example, a new sales strategy may be introduced, certain car properties may change, or even a new car model may be produced. Moreover, the role of the system within the organization may change over time. Since the requirements depend on the facts of the usage, subject, and system world, it is important that within the development world the relations between this information is recorded: either by recording the information in suitable models (for the usage, system, subject world) in the development world itself, or by interrelating the specification with models which already exist in the other worlds, e.g. a business model existing in the usage world. In the latter case, the models are reusable, i.e. the same model can be used for more than one system development process, and thus relationships between different system types can be expressed, e.g. relations about possible sales systems and the book keeping systems. Ideally, such models should describe a world independent of a particular system.

Changes within the subject, system, or usage world lead to modifications of the corresponding model. For example, introducing a new sales strategy causes changes to the business model. If the relations between the business model and the requirements specification are captured the consistent integration of changes can be supported, i.e. the corresponding parts of the system specification affected by the change can be identified. Without a suitable interrelation, the propagation of changes is almost impossible; at least changes are harder to integrate and more expensive.

Summarizing, the identification of the context is driven by the system vision and can be divided into three worlds (*usage, system, subject*). By establishing a RE team a new distinguishable social reality is created: the *development* world.

5.4 The Requirements Process within the Development World

Based on the four worlds framework and the example sketched above some conclusions about the RE process can be drawn.

First, the RE process takes place in the development world. The process is initiated and *driven by the overall system vision.*

Second, the RE *team* is established according to the system vision. It is important to consider aspects of each of the four worlds during RE. To ensure this, at least one representative of each world should participate in the RE team.

Third, each stakeholder involved has his/her own *personal views* on the system which of course heavily depend on the background of the stakeholder, the world he/she comes from. For example, a user representative cares more about the user interface of the system whereas a person of the system world focuses more on internal interfaces or technical integration aspects. The different views, aims, and goals are discussed during the RE process with the aim to reach an sufficient agreement on a final specification of the system.

Fourth, decisions are made during the RE process. Thus, the process is *unpredictable*. The different views and aims lead to discussions in which the team members exchange arguments. Based on these argumentations, e.g. pros and cons about the suitable presentation of the selected car model to the customer, decisions are made about certain system properties, e.g. to use pictures to present the selected car. These conversations and cooperations as well as the resulting decisions are not predictable.

Fifth, the *specification* of a system depends *on the context* of the system. Knowledge about the subject, system, and usage worlds may already be expressed in models within each world, e.g. business models provide a source for information about an enterprise and (depending on the vision) may be part of the usage world. Such models provide sources for the reuse. If abstract (conceptual) models are needed, but are not available from the three worlds, they have to be defined in the development world itself.

Sixth, the requirements specification must be related with the context information, i.e. must be *traceable*. Since the context (or even the vision) changes continuously the specification is never stable. To support the integration of changes into the requirements specification, relations between the context and the requirements are needed, i.e. each requirement must be traceable back to its origin, the part of the context it was influenced by, and vice versa.

6 Requirements Engineering: A Three Dimensional Framework

For defining a framework for the RE process, we follow an approach proposed by McMenamin and Palmer [McMenamin and Palmer, 1984] and distinguish between the essence of a process (system) and its incarnation. On an abstract level, the essence of a system can be seen as transforming the inputs received into the desired set of outputs. How this transformation is achieved is unimportant since we focus on the essence of the process, i.e. assume perfect technology.

6.1 The Initial Input of the Requirements Engineering Process

In Section 5 RE was defined as a process of establishing the overall system vision into existing context. Consequently, the system vision is the initial input for the RE process. When the vision is established some features of the system are obvious and well known whereas others are vague. Typical for many RE processes is the *opaque* understanding of the problem (system) at the beginning.

The stakeholders involved in the RE process come from different worlds (subject, system, usage) and therefore have different backgrounds, skills, and knowledge. The individual goals and aims of the stakeholder cause heterogeneous and conflicting *personal views* on the system which are typical for the beginning of the RE process. For representing this views each stakeholder uses his/her preferred representation format. Some of them may not use explicit representations, i.e. may just think about the system, others may make notes using natural language, or draw pictures or graphics. As indicated by surveys, e.g. [Lubars *et al.*, 1993], mainly *informal representations* are used at the beginning of the RE process.

Besides the clearly defined system vision, the initial input of the RE process can be characterized as an *opaque* understanding of the problem and a set of heterogeneous and conflicting *personal views* which are mostly represented using *informal languages*.

6.2 The Desired Output of the Requirements Engineering Process

At the end of a RE process there should be a as complete as possible requirements specification which serves as a basis for the next software development phase (at least for the current version of the system). There are two kinds of completeness. The first one deals with the coverage of the problem in the specification, i.e. it states if all relevant requirements are captured by the final specification.

The second one defines if each known requirement is well defined, i.e. if all requirements are defined according to the standard chosen for specifying the system. As mentioned in Section 3.1 there are many standards and guidelines which define how a particular type of requirement should be specified. Of course, it is almost impossible to define a specification which is complete in both senses. Nevertheless, the RE team should strive for a *complete specification*; obviously the first characteristic of the output of the RE process.

If the system specification is expressed using (for example) natural language, different people may understand the same specification in different ways. This may lead to an unexpected design and implementation. To avoid different interpretation of a specification more and more people suggest the use of formal languages. As indicated by many researchers (e.g. [Pohl *et al.*, 1994a]), it is not appropriated to specify the requirements using a single representation language. Therefore, depending on the kind of requirements and the intended usage of the representation, e.g. to prove a particular property of the specification or to explain the specification to the manager, a set of suitable representation formalism should be selected for representing the requirements in an adequate manner. However, it is commonly that at least parts of the specification should be formalized to enable the proof of certain features of the specification and to enable computer support during later development stages. Thus, at the end of the RE process (at least) parts of the specification should be expressed using a *formal language*.

But, even a complete formal requirements specification is an insufficient output of the RE process. In addition, a *common agreement* must be reached on the specification.[13] Assume that a functionality called `work control` is well defined and that there is no problem in mapping this part of the specification into a design and an implementation later on. But within the RE team only a few people agree on this functionality promoted by the people who are responsible for cost control. The representatives of the users do not like this functionality at all. If no common agreement is reached during the RE process the problems caused by this disagreement must be solved later on. As experience has shown, more effort is needed to correct errors in the following development phases [Boehm, 1984]. Thus, to avoid expensive error corrections a sufficient agreement on the final specification must be reached.

Summarizing, the *desired output* of the RE process can be described as a as complete as possible system specification on which all people involved agree. Furthermore, (part of) the specification should be expressed in a formal language.

6.3 The Three Dimensions of Requirements Engineering

From the *initial inputs* and the *desired outputs* of the RE process sketched above, three main goals of the process can be identified:

- developing a as complete as possible system specification out of an opaque system understanding;
- providing integrated representation formalisms and supporting the transformations between them;
- accomplishing a sufficient common agreement on the final specification allowing personal views.

Out of these goals the three dimensions of RE have been developed, namely the **specification**, **representation** and **agreement** dimension [Pohl, 1994].[14] Along the three dimensions, the initial input, as well as the desired output can be characterized. This is shown in figure 7, where the typical *initial*

[13] Not all people must agree on all facets of the specification. It is sufficient if they agree on the parts of the specification by which they are affected, interested in, or/and for which they are responsible. The agreement reached must be at least sufficient for starting building the system.

[14] The three dimensions were originally derived as a result of a comprehensive literature survey (cf. [Pohl, 1993]).

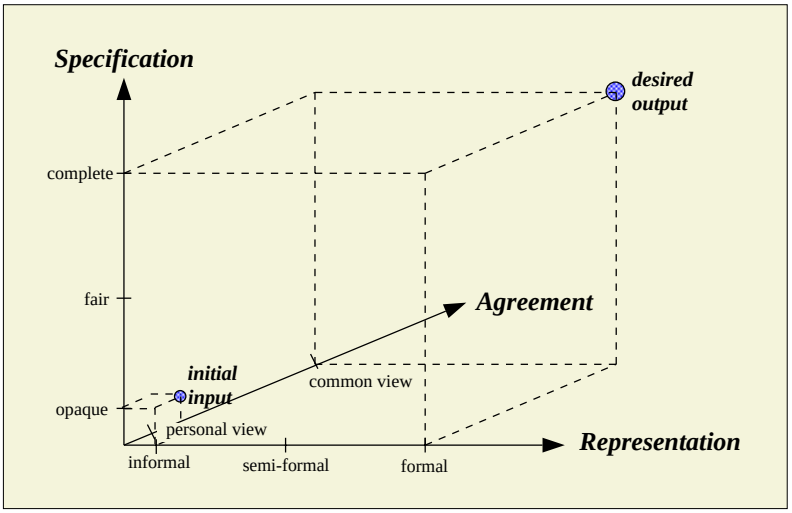

Fig. 7 The three dimensions of RE.

input (usually located somewhere in the lower left corner of the cube) is characterized by personal views, opaque system specification and informal representation, and the *desired output* (which should be located near the upper right corner of the cube) is sketched as common agreement, complete system specification and formal representation. In the following we briefly characterize each of the three dimensions.

6.3.1 The Specification Dimension

The *specification* dimension deals with the degree of completeness of the requirements specification typically measured against some standard, guideline, or domain model. At the beginning of the RE process the specification of the system and its environment is more or less opaque. This goes along with a vague understanding of the system at the early stage of the RE process. The aim of RE is to transform the operational need into a complete system specification through an iterative process of definition and validation (e.g. analysis, trade-off-studies, prototyping). Moves along this axis mostly face cognitive and psychological problems of requirement engineering.

As outlined in Section 3.1 several standards and guidelines describe how a “good” requirements specification should look like. What the system should do is stated by the functional requirements, whereas the non-functional requirements oppose constraints on the system. For example, IEEE–830 distinguishes between *functional requirements*, *performance requirements*, *design constraints*, *external interface requirements*, and quality *attributes*. Performance requirements deal with the system’s execution time and computational accuracy. Design constraints are predefined designs imposed on the software development by the customer, e.g., standard compliance. External interface requirements define aspects of the environment the system (the software) must deal with, e.g., constraints from other standards, hardware, or people. Quality attributes restrict design and implementation decisions, e.g., maintainability or availability (see [Keller *et al.*, 1990; Pohl *et al.*, 1994a; Pohl and Peters, 1995; Krogstie *et al.*, 1995] for examples of quality attributes and their definitions). In addition, a distinction between *vital* requirements and *desirable* requirements should be made (cf. British Standard 6719 [BS-6719, 1986], IEEE-830 [IEEE-830, 1984]). *Vital* requirements must be completely accomplished by the system, whereas *desirable* requirements may be relaxed and need not be met within the stated limits. Some standards propose to include *costs* and *schedule* information in the requirements specification (e.g. British Standard 6719) whereas others separate them from RE (e.g. IEEE Statement of Work).

The progress along the specification dimension goes along with a better problem understanding and reflects the learning of the people about the system (problem). As the process proceeds, on the one

hand, more and more requirements are specified, i.e. the number of hidden requirements is reduced and thereby the specification gets more and more complete. On the other hand, each individual requirement is specified in more detail, i.e. each requirement is defined according to the requirements opposed on the specification by a particular standard. Both improvements lead to a more and more complete specification of the system. As stated, e.g. in [Nakajima and Davis, 1994], it is not provable if a specification is really complete, i.e. it can not be proven that a requirement has been forgotten. Nevertheless, the use of domain models (cf. [Maiden, 1992]), problem frames ([Jackson, 1995]), or/and a particular standard or guideline helps to produce complete requirements specifications.

Summarizing, the *specification* dimension captures the degree of completeness of the requirements specification and thereby indicates the understanding of the system (problem) reached. Moves along this dimension mostly face the cognitive problems of RE.

6.3.2 The Representation Dimension

The *representation* dimension recognizes the use of different languages (informal and formal languages, graphics, sounds etc.) used to represent the requirements of the system. Modelling and relating these representations is mostly a technical problem albeit one whose solution needs good knowledge about human-computer interactions. Within RE there are three categories of languages. The first includes all informal representations, such as arbitrary graphics, natural language, descriptions by examples, sounds and animations. The second category subsumes semi-formal languages such as data flow diagrams (DFDs) or ER-diagrams (cf. figure 2 for example notations). The third category covers formal languages such as specification languages (e.g. VDM [Bjoerner and Jones, 1988], Z [Spivey, 1990]) or knowledge representation languages (e.g. ERAE [Hagelstein, 1988], Telos [Mylopoulos *et al.*, 1990]).

Each of these categories offers unique advantages. *Informal representations* like natural language are user-oriented and used in everyday life. The expressive power offered by informal representation is very high and all kinds of requirements freedom are available (e.g. ambiguity, inconsistency, contradictions; see [Balzer *et al.*, 1978; Feather and Fickas, 1991] for more details). *Semi-formal representations* like DFDs or ER diagrams provide a graphical visualization of certain features of the system. These representations are widely used within industry and are easy to understand and provide a good overview of the system (“one picture says more than a thousand words”). The formal semantics offered by semi-formal languages is very poor so that most of the represented knowledge has no formal meaning. In contrast, *formal representation* languages have richer, well defined formal semantics. Therefore automated reasoning about most of the represented knowledge is possible. Even executable code can be (partially) generated out of them. Thus, formal representation languages are more system oriented.

The use of a particular representation language has two main reasons. On the one hand, the choice of a particular language depends simply on personal preference. Due to the advantages of each representation format, different people prefer different representations. For example, the system user may like natural language, whereas the system specialist may prefer a formal language. On the other hand, the choice of a language depends on the state of the specification. At the beginning of the RE process informal languages are used more often, whereas towards the end formal languages are used to define unambiguous and consistent specifications. However, it is commonly that different representation languages have to be used in parallel to represent different views on the system and thus enable the stakeholders to understand the specification from their perspective.

As a consequence, the various representation formats must be integrated and must be kept consistent. Assume that a requirement was expressed by the customer using natural language. The system specialist

has derived a formal specification out of this requirement. If, e.g. the informal requirement is revised, it must be ensured that the formal specification is changed accordingly.

The use of a particular representation language does not imply if the developed specification is vague or precise, i.e. the *representation* dimension is orthogonal to the *specification* dimension. A vague imagination of the system can be expressed using natural language, but it is also possible to hide poor understanding of the system using a complex formalism. Concrete (formally defined) ideas can obviously be represented using a formal representation language, but can also be exactly stated using natural language (e.g. lawyers try to do so). Looking at the specification `'the age of Carl is 10 years'` and on a formal specification, e.g. using first order logic, `'age (Carl, 10, years)'` no difference can be recognized, whereas the vague specification *'Carl is young'* is also vague if it is represented in first order logic `'young (Carl)'`. Hence the difference between the two specifications, vague versus precise, remains the same independent of the representation language used.

Summarizing, during the RE process different representation languages are used. At the beginning of the process the knowledge about the system is preferably expressed using informal representations, whereas at the end of RE parts of the specification should be formalized.[15] Thus, the goal of the RE process along the representation dimension is threefold. First, different representations must be offered and interrelated. Second, the transformation between the representations (e.g. informal to semi-formal, informal to formal) must be supported. Third, the different representations must be kept consistent. Thus, moves along this dimension are mostly faced with the technical problems of RE.

6.3.3 The Agreement Dimension

The third dimension deals with the degree of agreement reached on a specification. In the following, the expression *common system specification* is used for the part of the specification on which the RE team has agreed. Obviously this axis deals mainly with the social aspects of RE.

At any time in the RE process there are requirements which are shared among the team (common system specification), whereas others exist only within the personal views of the various stakeholders. Let us focus on a simple example. Assume that a library system is currently being specified and the RE team has agreed that data about the real world object `'book'` must be recorded by the system. Thus `'book'` belongs to the common system specification. At the same time each stakeholder may define (from his/her point of view) the properties of the object `'book'`. For example, the user defines the properties `'book-title, author-name, year'` using natural language, the system analyst the properties `'book-id, status-of-book (loaned | available | defect | stolen | ordered)'` using a formal representation language and the librarian defines the properties `'names of authors, keywords, classification-no., location,...'`. Whereas the object book belongs to the common system specification, the various properties of book are pertained by the personal views, i.e. are not shared between the requirements engineering team. Adding to complexity, the coexistent specifications (common and personal views) are expressed using different representation languages.[16]

---

[15] Which parts of the specification should be formalized and which language should be used for which kind of formalization are still open research topics (as identified, e.g. at the First International Workshop on Requirements Engineering REFSQ'94, and the workshop on Formal Models for Information System Dynamics, both held in conjunction with CAiSE '94).

[16] The observation that each person has his/her own views has led to the introduction of the notion of viewpoints [Finkelstein *et al.*, 1992]. A viewpoint is a complex structure and has its own process definition, special representation, process trace, etc. (see [Finkelstein *et al.*, 1992] for details). In general we agree with the definition made by Finkelstein et al. [Finkelstein *et al.*, 1992], but in contrast, we assume that there exists also one "common" viewpoint, the specification. The specification is constructed during the RE process through communication and negotiation. The specification could be understood as a consistent configuration over parts of the individual viewpoints which were made public by communication. Consequently, not every person must understand the whole specification (since not all of the information may be in the personal viewpoint(s) of the person). Even if a final specification exists, there may be parts of personal viewpoints (not covered in the specification) which are inconsistent. However, both the evolution of each personal viewpoint and the construction of the final specification can be characterized by the three dimensions of RE [Maiden *et al.*, 1994].

Different views of the same system have positive effects on the RE process. First of all, they provide a good basis for requirements elicitation (cf. [Leite and Freeman, 1991]). Second, the examination of the differences can be used as a way of assisting in the early validation of requirements. Hence, different views enable the team to detect additional requirements. Third, different views enable the specification of incompatible requirements and thus existing conflicts can be detected during the integration of the views which would otherwise have remained hidden [Nissen *et al.*, 1996]. It is important to distinguish between the integration of different views (or parts of them) at the representation level (e.g. transforming formally represented views into a comprehensive view) and the agreement on the integrated view among the people involved in the process. The fact that a (part of the) view was formally integrated has nothing to do with the agreement on this view. A detected conflict must be solved through communication among people. Of course this communication has the aim of attaining an agreement (solving the conflict), but as a side effect additional unknown arguments (requirements) could be detected (cf. Section 4.2).

To summarize, the *agreement* dimension is orthogonal to and as important as the *representation* and *specification* dimensions and mainly deals with the social aspects of RE. We have pointed out that several specifications expressed in different representation formats may exist at the same time. Allowing different views and supporting the evolution from these personal views to a common agreement on the final specification (common view) is the third main goal of the RE process.

7 Consequences of the Three Dimensions

In the following we briefly describe the consequences of the three dimensional framework for the RE process (Section 7.1) and the RE product (Section 7.2). In Section 7.3 we outline the need for requirements traceability and define the kind of knowledge to be captured for establishing traceable requirements specifications.

7.1 The Requirements Engineering Process within the Three Dimensions

Based on the three dimensional framework introduced in section 6 the overall aim of the RE process can be stated as getting from the *initial input* to the *desired output*. The trace of the RE process is an arbitrary curve within the cube spanned by the three dimensions (cf. figure 8).

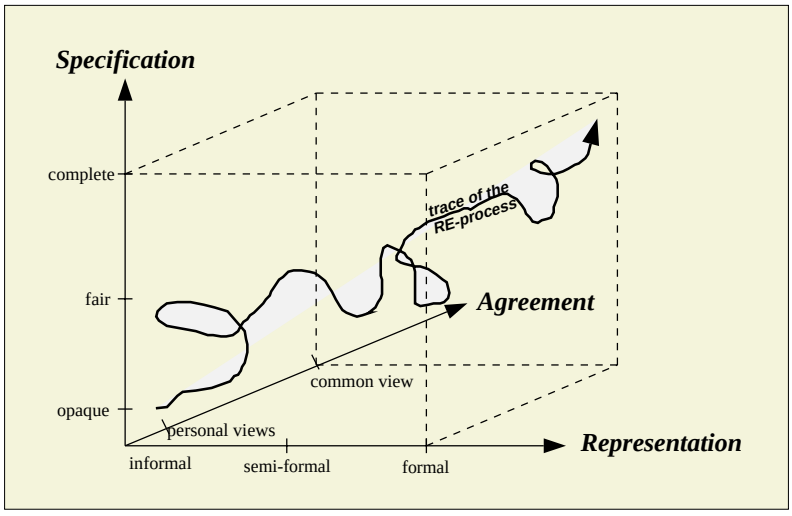

Fig. 8 The RE process within the three dimensions.

Task	*Characterization of the Task*	*Main Influence*
elicitation	makes knowledge (requirements) about the system explicit and thus leads to a better understanding of the problem (system)	specification dimension
negotiation	makes existing conflicts, argumentations and rationales explicit and assures that the "rigth" decisions are made; establishes an agreement between the various stakeholder	agreement dimension
specification/ documentation	deals with the representation of the existing viewpoints in different representation formats; assures consistency and cross-references between the various representation; establishes a (partially) formal requirements specification	representation dimension
validation/ verification	assures that the right problem is being tackled at any time in the process; checks the internal consistency of the specification; controls if the specified requirement are consistent with the user/customer intentions	all three dimensions;

Table 3 The four tasks of RE and their main influence on the three dimensions.

The transformation of the *initial input* into the *desired output* is an interactive process in which four types of interrelated tasks (elicitation, negotiation, specification/documentation, and validation/verification) are performed (cf. Section 4). In table 3 the main relations between the four tasks and the three dimensions are characterized.

The execution of a single task can of course affect more than one dimension; improving one dimension often leads to a setback in another dimension. For example, the transformation of an informally represented requirement into a formal requirements specification could affect all three dimensions. Obviously, this transformation causes an improvement within the *representation* dimension, since an informally stated requirement is transformed into a formal specification. In addition, the formalization of the requirement may lead to the detection of existing contradictions, e.g. recognized by automated reasoning. This contradictions can either be solved by negotiations among the stakeholders involved in the conflict (improvement in the agreement dimension), or/and lead to an elicitation task by which additional knowledge about the conflict is made explicit (improvement in the specification dimension).

Based on the three dimensions a prediction on how the specification will change after a certain action type has been executed can be made. This prediction can be used to establish a process models for guiding the RE process within the three dimensions. Due to the immature knowledge about RE processes only few action types can be precisely predefined (cf. [Rolland and Prakash, 1993; Jarke *et al.*, 1994; Pohl, 1996b]).

Adding to complexity, the RE process is not only influenced by the three dimensions (i.e., has to deal with the cognitive, social and technical barriers) but is in addition affected by three other factors, namely:

- *methods:* the process is driven by the method chosen for developing the specification. Obviously, the resulting requirements specification is strongly influenced by the method applied. For example, using Structured Analysis will result in a specification which is significantly different from the one gained by applying an object-oriented technique;
- *tools:* the quality of the final specification depends on the tools used during the process. For example, if the development of a formal specification is supported by a reasoning tool, inconsistencies could be detected which otherwise would have remained in the final specification;

- *economical constraints:* economical constraints limit the resources (people, money, tools, etc.) of the RE process. It is not always true that with more resources better results are obtained, but if the available resources are below a certain limit the output of the process will be of inferior quality. Economical constraints especially effect the degree of completeness reached in the final specification.

Discussing these influences in detail is beyond the scope of this contribution. But it should be clear that they are not unique to the RE process. Most of the existing processes, e.g. production processes, are influenced by these factors.

For these reasons it is necessary to distinguish between problems which are *original* RE problems and those problems which are caused by one of the three main influences mentioned above. The problem of keeping data flow diagrams and ER-diagrams as well as the data-dictionary consistent is an example for a problem caused by one of the three influences mentioned above (methods). *Original* RE problems are all the problems caused by the three dimensions and the four tasks identified in section 4.

7.2 The Product Viewed From the Three Dimension

In contrast to the existing standard and guidelines which define the software requirements specification as the main product of the RE process (cf. Section 3), the three orthogonal dimensions of RE imply a richer definition of the RE product. According to the three dimensional framework and the process description given above the product of the RE process should:

- ***document the problem understanding reached:*** the understanding of the problem reached should be documented independent of the representation formats used for specifying the requirements. Therefore a generic specification model is needed in which the content of the specification is defined independent of the actual representations of the requirements. For example, a functionality of the system could be graphically defined in a DFD. At the same time the input of the function could be defined in an ER-diagram and the behavior could be stated in natural language and, in addition, in a state transition diagrams. The specification model should express that the function, the input of the function, and the behavior of the function are known and specified and thus provide an overview on the content of the specification;
- ***offer different views on the specification:*** the RE process should not only produce a as formal as possible requirements specification. Instead, the output along the representation dimension should be a set of requirements models (textual, graphical, formal, ...) which reflect the various views on the system as well as the different modelling perspectives (data, function, behavior, organization, user, etc.). The different views on the specification should be used to master the obstacles caused by the different recognition of the stakeholders involved in the process. For example, for explaining the functionality of the future system to the manager DFDs may be an appropriate representation format whereas the functionality may be better illustrated to the user by prototypical window layouts. Of course, these views must be consistent and must conform with the specification at any time. In addition, cross-references should be provided between the views to enable easier change integration and view comparison;
- ***document the agreement reached:*** the progress made along the agreement dimension is an important part of the RE product. Although neglected by almost all existing specification standards and guidelines the results of the negotiation tasks must be documented, e.g., the decisions made during the RE process together with their rationales. For recording the decision and their underlying rationale, e.g. an IBIS based model could be used (cf. [Pohl, 1996b]). Recording the decisions and their rationale enables future adaptation of the specification

according to new needs, increases the acceptance of the system, and enables people not involved in the RE process to understand the decisions made. Thus, the documentation of the agreement reached is a substantial part of the RE product;

- ***state the relations to the other three worlds:*** in addition to the above, the relations of the specification to the subject, system, usage world should be explicitly stated within the RE product, i.e. the context in which the system is going to operate and its influence on the specification should be clearly defined. For example, the contribution structure should be part of the product [Gotel and Finkelstein, 1994];
- ***be traceable:*** for various reasons (cf. Section 7.3) the RE product must be traceable. Briefly, it must be assured that the life of every single requirement can be reconstructed and that people not involved in the process can understand why the requirements specification was produced in this particular way.

The comprehensive RE product proposed above (in the following called requirements specification) provides the basis for high quality design and implementation, for easy adaptation of the system to future needs, and for improving the RE process based on experience [Pohl, 1996b].

7.3 Requirements Traceability

Requirement traceability is a prerequisite for building high quality software systems and thus capturing and maintaining traces is an essential activity to be performed during RE (e.g. [IEEE-830, 1984; DoD-2167A, 1988; Wright, 1991; Ramesh, 1993; Gotel and Finkelstein, 1993; Pohl, 1994; Pohl, 1996a]). A comprehensive overview of possible usage of trace information and the expected benefits can be found in, e.g., [Gotel and Finkelstein, 1993; Ramesh, 1993; Pohl, 1996b]. Among others, these reports indicate that traceable specifications are essential for change integration, lead to less errors during system development, play an important role in contract situations, and improve the acceptance of the (software) system. Of course, traceability adds to what has to be done across the entire life cycle, but it will reduce the non-productive work by much more [Ramesh, 1993, p.18]

On the one hand, traceability from the requirements specification down to design and implementation and vice versa is needed to understand and therefore accept the current system better. On the other hand, the development process leading to the requirements specification must be traceable to enable the understanding of the requirements themselves, i.e. to trace a requirement back to its origin.

Thus, commonly a differentiation between the two kinds of traceability is made. Whereas the traceability of the refinement, deployment, and use of a requirement is called *post-traceability* (or forward traceability), the traceability of a requirement back to its origin is named *pre-traceability* (or backward traceability) (cf. [IEEE-830, 1984; Davis, 1990; Gotel and Finkelstein, 1994; Pohl, 1996b])[17]. Requirements post-traceability mainly supports the requirements engineer and the software engineer in keeping track of the requirements and ensuring that all requirements are properly flowed down to all specification levels and to the design of the system with no requirement lost and none added in (cf. [Alford, 1980; Flynn and Dorfmann, 1990; Alford, 1990]). Requirements pre-traceability is a prerequisite for managing the evolution of the system on the specification level. It assures that the rationale and goals behind each requirement and all existing dependencies, e.g., between organizational goals and the requirements, are recorded. Thereby, it empowers the identification of the requirements effected by a change request, e.g. a change of an organizational policy or an extended usage of the system. Consequently, requirements pre-traceability is as important as requirements post-

[17] An excellent survey on existing approaches can be found in (cf. [Gotel and Finkelstein, 1993])

traceability – especially for systems which are embedded in a continuously changing environment (cf. [Gotel and Finkelstein, 1993; Ramesh and Edwards, 1993; Pohl, 1996a]).

For establishing requirements pre-traceability the information about the execution of the RE process must be recorded, i.e. in addition to the comprehensive product defined in section 7.2 the process steps and the agents performing these steps must be captured and related to the product produced. A process and repository centered approach which enables the recording and selective retrieval of all this information, called PRO-ART, is described in [Pohl, 1996b; Pohl, 1996a].

8 Conclusion and Future Perspectives

Like software engineering, many other engineering and business disciplines are experiencing the need to understand better the early phases of their processes, and to maintain this information over time and across traditional technical and organizational boundaries. Within software engineering, RE has traditionally been responsible for this kind of task in which a fuzzy initial idea is transformed into a precise specification. The good news is that none of the other disciplines seem to be better than RE in terms of understanding the actual experience, and offering adequate methods and tools for the early phases of product development.

In this contribution we have sketched the state of the art and practice of RE by reflecting on different definitions and characterizing the product, the requirements specification, as well as the RE process. Besides the common modelling perspectives (*data, function, behavior, object-oriented*), we have described four main tasks (*elicitation, negotiation, specification/documentation, validation/verification*) to be performed during the RE process. To give a more comprehensive picture of RE we have than proposed a comprehensive framework for RE: the *four worlds* (development, subject, system, usage) for structuring the context on information systems and the *three dimensions* (agreement, representation, specification) by which the RE process was characterized.

The influences of this broader definition on the RE process and its product were sketched in the last section. Although we have focused on the specification of information systems, which are increasingly becoming an integral part of our everyday live, most of the results presented, especially the three dimensional framework, are applicable to the specification of any kind of system.

In the future, RE will move from a one shot activity in the development process towards a virtual image that accompanies the changing reality of a system [Jarke and Pohl, 1994]. In other words, RE will become the most important activity in the software development life-cycle with the main responsibility for producing high quality (software) systems and managing their evolution. In the following we sketch the areas of RE where improvements are needed to come up to these expectations:[18]

- **better understanding of the RE process:** a comprehensive understanding of the RE process is critical to rise to the new application challenges. We do have some knowledge about methods at the very fine-grained level, e.g. at the notational level, and at the very coarse-grained level of, e.g. five step procedures. But there is a big gap between the two kinds of knowledge which can only be filled by careful analysis of existing and evolving practice. Methods and tools for requirements traceability and experience based process improvement are therefore central to our understanding of what RE actually involves [Jarke *et al.*, 1994; Pohl, 1996b].
- **evolution of systems (change management):** the challenge of RE is to take a much more important role in many domains, as its products (the conceptual specifications, the interrelated

[18] The changing role of RE is elaborated in more detail in [Jarke and Pohl, 1994].

viewpoints and their instantiations in prototypical scenarios of virtual reality) are becoming instruments through which interdisciplinary teams can understand and communicate about the change reality.

- **use of scenarios:** animations of requirements specification and prototyping of design solutions have been among the successful techniques to support requirements validation and elicitation. Typically, specific examples (test cases, scenarios) are used by which the specification is instantiated in order to provoke critique. In the future, animations and prototypes are needed which go beyond this scope by including the context the system is going to operate. High-performance computing could enable animations in a rich interactive multimedia style of virtual reality and thereby various stakeholders could be enabled to understand, criticize, and to improve the specification.
- **broader user participation:** the four worlds model indicates that we should broaden the participation in RE teams. Of course, user participation has been a long-standing concern of RE, but remains a difficult issue even just from the viewpoint of the question of who are the 'users' to be involved. More important, many systems nowadays have impact beyond their usage and development environment. Thus, drawing the right people in the RE team and offering support for reaching an agreement is an important task which will (and has to) be supported much better in future. In the representation dimension ways must be found to help these stakeholders (which are neither domain experts nor technical specialists) with analyzing the system impact from their own perspective.
- **additional modelling perspectives:** the broader role of RE within an organization requires richer specification models. Among others, this is indicated by the increasing efforts spend in goal modelling, workflow modelling, and business process modelling. Thus, there is at least a need for better consideration of the organizational perspective. In addition, the increasing role of the system users must be supported by providing methodical advise and concepts for specifying human computer interactions like interactive user guidance, help facilities, explanation components, user based adaptability or ergonomic user interfaces. More general, the different relations between the usage, subject, system, and development world must be adequately specified if the requirements specification should fulfill the role of a virtual image by which the system evolution is managed.
- **construction of software:** last but not least, the reuse of existing components at the specification level (not on the implementation level) will be fundamental for the success of future system development. As the software market matures low-cost standard packages will be available which have to be adequately considered during the specification of a system. Consequently, RE has to move towards an engineering discipline in which a software system is constructed rather than created from scratch.

Acknowledgments. This work was supported in part by the European Community under ESPRIT Basic Research Project 6353 (NATURE), the European-Australian Cooperation Project ISI (ECAUS003) and the Ministry of Science and Research of Nordrhein-Westfalen.

Thanks are due to the NATURE team for many stimulating discussions, especially to Matthias Jarke for encouragement and freedom which have made this contribution possible. For helpful comments on earlier drafts of this paper I am grateful to Ralf Dömges, Matthias Jarke and Klaus Weidenhaupt.

9 References

[AAA, 1994]
AAAI Workshop on Models on Conflict Management in Cooperative Problem Solving, 1994.

[Adelson and Soloway, 1985]
B. Adelson and E. Soloway. The Role of Domain Experience in Software Design. *IEEE Transactions on Software Engineering 11*, 11 (1985).

[Alford, 1980]
M. W. Alford. Software Requirements Engineering Methodology (SREM) at the Age of Two. In *Proc. of the Fourth Intl. Computer Software and Applications Conf.*, pp. 866–874, New York, NY, 1980. IEEE Computer Society Press.

[Alford, 1990]
M. W. Alford. Software Requirements Engineering Methodology (SREM) at the Age of Eleven – Requirements Driven Design. In P. A. Ng and R. T. Yeh (Eds.), *Modern Software Engineering*. Van Nostrand Reinhold, 1990.

[Balzer *et al.*, 1978]
R. Balzer, N. Goldman and D. Wile. Informality in Program Specifications. *IEEE Transactions on Software Engineering 4*, 2 (1978), pp. 94–103.

[Balzer, 1991]
R. Balzer. Tolerating Inconsistency. In *Proc of the Thirteenth Intl. Conf. on Software Engineering*, pp. 158–165, Austin, TX, May 1991.

[Benner *et al.*, 1993]
K. M. Benner, M. S. Feather, W. L. Johnson and L. A. Zorman. Utilizing Scenarios in the Software Development Process. In N. Prakash, C. Rolland and B. Pernici (Eds.), *Proc. of the IFIP WG 8.1 Conf. on Information System Development Process*, pp. 117–134, Como, Italy, September 1993. Elsevier B. V. (North Holland).

[Berzins and Gray, 1985]
V. Berzins and M. Gray. Analysis and Design in MSG 84: Formalizing Functional Specifications. *IEEE Transactions on Software Engineering 11*, 8 (August 1985), pp. 865–885.

[Bigelow, 1988]
J. Bigelow. Hypertext and CASE. *IEEE Software* (March 1988), pp. 23–27.

[Bjoerner and Jones, 1988]
D. Bjoerner and C. B. Jones. *VDM'87 VDM-A Formal Method at Work*. No. 252, LNCS. Springer-Verlag, 1988.

[Boehm, 1984]
B. W. Boehm. Verifying and Validating Software Requirements and Design Specifications. *IEEE Software 1*, 1 (January 1984), pp. 75–88.

[Booch, 1991]
G. Booch. *Object Oriented Design with Applications*. Benjamin/Cummings Publishing Company Inc., Redwood City, CA, 1991.

[Borgida *et al.*, 1985]
A. Borgida, S. Greenspan and J. Mylopoulos. Knowledge Representation as the Basis for Requirements Specifications. *IEEE Computer 18*, 4 (April 1985), pp. 82–91.

[BS-6719, 1986]
BS-6719. British Standard Guide to Specifying User Requirements for Computer-Based Systems. 1986. British Standard Institute.

[Bui, 1987]
T. X. Bui. *Co-oP: A Group Decision Support System for Cooperative Multiple Criteria Group Decision Making*. LNCS 290. Springer Verlag, Berlin, 1987.

[Bush, 1990]
M. Bush. Improving Software Quality: The Use of Formal Inspections at the Jet Propulsion Laboratory. In *Proc. of the Twelfth Intl. Conf. on Software Engineering*, pp. 196–199, Nice, France, March 1990.

[Camaron, 1986]
J. R. Camaron. An Overview of JSD. *IEEE Transactions on Software Engineering 12*, 2 (February 1986), pp. 222–240.

[CERA, 1994]
CERA. Special Issue on Conflict Management in Concurrent Engineering. *Journal on Concurrent Engineering Research and Application, CERA 2*, 3 (1994).

[Charette, 1986]
R. Charette. *Software Engineering Environments*. McGraw Hill, New York, 1986.

[Chen, 1976]
P. P. S. Chen. The Entity-Relationship Approach: Towards a Unified View of Data. *ACM Transactions on Database Systems 1*, 1 (1976).

[Coad and Yourdon, 1990]
P. Coad and E. Yourdon. *Object Oriented Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1990.

[Conklin and Begeman, 1988]
J. Conklin and M. J. Begeman. gIBIS: A Hypertext Tool for Exploratory Policy Discussion. *ACM Transactions on Office Information Systems 6*, 4 (1988), pp. 303–331.

[Constantopoulos *et al.*, 1995]
P. Constantopoulos, M. Jarke, J. Mylopoulos and Y. Vassiliou. The Software Information Base: A Server for Reuse. *VLDB Journal 4*, 1 (1995), pp. 1–43.

[Dardenne *et al.*, 1992]
A. Dardenne, S. Fickas and A. van Lamsweerde. Goal-Directed Concept Acquisition in Requirements Elicitation. In *Proc. of the Sixth Workshop on System Specification and Design*, pp. 14–21, Como, Italy, 1992.

[Davis and Siddiqi, 1994]
A. Davis and J. Siddiqi (Eds.). *Proc. of the First Intl. Conf. on Requirements Engineering*. IEEE Computer Society Press, April 1994.

[Davis, 1988]
A. M. Davis. A Comparison of Techniques for the Specification of External System Behavior. *Communications of the ACM 31*, 9 (1988), pp. 1098–1115.

[Davis, 1990]
A. M. Davis. The Analysis and Specification of Systems and Software Requirements. In R. H. Thayer and M. Dorfman (Eds.), *Systems and Software Requirements Engineering*, pp. 119–134. IEEE Computer Society Press — Tutorial, 1990.

[Davis, 1993]
A. M. Davis. *Software Requirements*. Prentice Hall, Upper Saddle River, NJ, 1993.

[de Antonellis *et al.*, 1991]
V. de Antonellis, B. Pernici and P. Samarati. F-ORM Method: Methodology for reusing Specifications. *ITHACA Journal*, 14 (1991), pp. 1–24.

[de Champeaux and Faure, 1992]
D. de Champeaux and P. Faure. A Comparative Study of Object-Oriented Analysis Methods. *Journal of Object-Oriented Programming* (1992), pp. 21–33.

[DeMarco, 1979]
T. DeMarco. *Structured Analysis and System Specification*. Prentice Hall, Englewood Cliffs, NJ, 1979.

[Dhar and Jarke, 1985]
V. Dhar and M. Jarke. Learning from Prototypes. In *Proc. of the sixth Intl. Conf. on Information Systems*, pp. 114–133, Indianapolis, IN, 1985.

[DoD-2167A, 1988]
DoD-2167A. Military Standard: Defense System Software Development. 1988. U.S. Department of Defense.

[Dorfman and Thayer, 1990]
M. Dorfman and R. H. Thayer. *Standards, Guidelines and Examples on System and Software Requirements Engineering*. IEEE Computer Society Press – Tutorial, 1990.

[Dubois *et al.*, 1994]
E. Dubois, P. DuBois, F. Dubru and M. Petit. Agent-Oriented Requirements Engineering: A Case Study using the ALBERT Language. In *Proc. of the Fourth Intl. Working Conf. on Dynamic Modeling and Information Systems*, Noordwijkerhoud, The Netherlands, September 1994.

[Fagan, 1986]
M. E. Fagan. Advances in Software Inspections. *IEEE Transactions on Software Engineering 12*, 7 (1986), pp. 744–751.

[Feather and Fickas, 1991]
M. S. Feather and S. Fickas. Coping with Requirements Freedom. In *Proc. of the Intl. Workshop on the Development of Intelligent Information Systems*, pp. 42–46, Niagara-on-the-Lake, Ontario, Canada, April 1991.

[Fichman and Kemerer, 1992]
R. G. Fichman and C. F. Kemerer. Object-Oriented and Conventional Analysis and Design Methodologies (Comparison and Critique). *IEEE Computer* (October 1992), pp. 22–39.

[Fickas and Finkelstein, 1993]
S. Fickas and A. Finkelstein (Eds.). *Proc. of the First Intl. Symp. on Requirements Engineering*. IEEE Computer Society Press, January 1993.

[Fickas and Nagarajan, 1988]
S. Fickas and P. Nagarajan. Critiquing Software Specifications. *IEEE Software* (November 1988), pp. 37–47.

[Finkelstein *et al.*, 1992]
A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein and M. Goedicke. Viewpoints: A Framework for Integrating Multiple Perspectives in System Development. *Intl. Journal of Software Engineering and Knowledge Engineering 1*, 2 (May 1992).

[Finkelstein *et al.*, 1993]
A. Finkelstein, D. Gabbay, A. Hunter, J. Kramer and B. Nuseibeh. Inconsistency Handling in Multi-Perspective Specifications. In *Proc. of the Fourth Europ. Software Engineering Conf.*, pp. 84–99, Garmisch-Partenkirchen, Germany, September 1993. Springer-Verlag.

[Flynn and Dorfmann, 1990]
R. F. Flynn and D. Dorfmann. The Automated Requirements Traceability System (ARTS): An Experience of Eight Years. In R. H. Thayer and M. Dorfman (Eds.), *Systems and Software Requirements Engineering*, pp. 423–438. IEEE Computer Society Press — Tutorial, 1990.

[Fraser *et al.*, 1991]
M. D. Fraser, K. Kumar and V. K. Vaishnavi. Informal and Formal Requirements Specification Languages: Bridging the Gap. *IEEE Transactions on Software Engineering 17*, 5 (May 1991), pp. 454–466.

[Freeman and Weinberg, 1990]
D. P. Freeman and G. M. Weinberg. *Handbook of Walkthroughs, Inspections and Technical Reviews*. Dorset House Publishing, New York, 1990.

[Gane and Sarson, 1979]
C. Gane and T. Sarson. *Structured Systems Analysis: Tools and Techniques*. Prentice Hall, Englewood Cliffs, NJ, 1979.

[Garg and Scacchi, 1990]
P. K. Garg and W. Scacchi. A Hypertext System to Manage Software Life-Cycle Documents. *IEEE Software* (May 1990), pp. 90–98.

[Gause and Weinberg, 1989]
D. C. Gause and G. M. Weinberg. *Exploring Requirements: Quality Before Design*. Dorset House Publishing, New York, 1989.

[Gibbels, 1994]
F. Gibbels. A Comprehensive Specification Model for Information Systems. Master's thesis, RWTH-Aachen, Germany, 1994. (in German).

[Goguen and Linde, 1993]
J. A. Goguen and C. Linde. Techniques for Requirements Elicitation. In *Proc. of the First Intl. Symp. of Requirements Engineering*, pp. 152–164, San Diego, CA, January 1993. IEEE Computer Society Press.

[Gotel and Finkelstein, 1993]
O. Gotel and A. Finkelstein. An Analysis of the Requirements Traceability Problem. Technical Report TR-93–41, Imperial College, Department of Computing, 1993.

[Gotel and Finkelstein, 1994]
O. Gotel and A. Finkelstein. An Analysis of the Requirements Traceability Problem. In *Proc. of the First Intl. Conf. on Requirements Engineering*, pp. 94–102, Colorado Springs, CO, April 1994. IEEE Computer Society Press.

[Greenspan *et al.*, 1993]
S. J. Greenspan, M. Alford, G. Fischer, J. Lee, C. Potts and D. Weiss. Panel: Recording Requirements Assumptions and Rationale. In *Proc. of the First Intl. Symp. of Requirements Engineering*, pp. 282–285, San Diego, CA, January 1993. IEEE Computer Society Press.

[Greenspan *et al.*, 1994]
S. Greenspan, J. Mylopoulos and A. Borgida. On Formal Requirements Modelling Languages: RML Revisited. In *Proc. of the Sixteenth Intl. Conf. on Software Engineering*, pp. 135–148, Sorrento, Italy, May 1994. IEEE Computer Society Press.

[Greenspan, 1984]
S. J. Greenspan. *Requirements Modeling: A Knowledge Representation Approach to Software Requirements Definition*. PhD thesis, Dept. of Computer Science, University of Toronto, 1984.

[Hagelstein, 1988]
J. Hagelstein. Declarative Approach to Information Systems Requirements. *Knowledge Based Systems 1*, 4 (1988), pp. 211–220.

[Hall, 1990]
A. Hall. Seven Myths of Formal Methods. *IEEE Software 7*, 9 (September 1990), pp. 11–19.

[Hallmann, 1990]
M. Hallmann. *Prototyping of Complex Software Systems*. Teubner Verlag, 1990. in German.

[Harel, 1987]
D. Harel. STATECHARTS: A visual Formalism for Complex Systems. *Science of Computer Programming 8* (1987), pp. 231–274.

[Harrison and Zave, 1995]
M. Harrison and P. Zave (Eds.). *Proc. of the Second Intl. Symp. on Requirements Engineering*. IEEE Computer Society Press, April 1995.

[Hatley and Pirbhai, 1987]
D. J. Hatley and I. A. Pirbhai. *Strategies for Real-Time System Specification*. Dorset House, New York, 1987.

[Haumer, 1994]
P. Haumer. A Hypertext System for Structuring and Integrating Informal Requirements. Master's thesis, RWTH-Aachen, Germany, 1994. (in German).

[Hauser and Clausing, 1988]
J. R. Hauser and D. Clausing. The House of Quality. *Harvard Business Review* (May 1988), pp. 63–73.

[Henderson, 1986]
P. Henderson. Functional Programming, Formal Specification, and Rapid Prototyping. *IEEE Transaction on Software Engineering 12*, 2 (February 1986), pp. 241–249.

[HIC, 1994]
Proc. of the Twentyseventh Hawaii Intl. Conf. on System Science, HICSS'94, Hawaii, 1994.

[HIC, 1995]
Proc. of the Twentyeighth Hawaii Intl. Conf. on System Science, HICSS'95, Hawaii, 1995.

[Hoare, 1990]
C. A. R. Hoare (Ed.). *Intl. Conf. on VDM and Z*, No. 428, LNCS. Springer-Verlag, 1990.

[Hopcroft and Ullman, 1979]
J. E. Hopcroft and J. D. Ullman. *Introduction into Automata Theory: Language and Computation*. Addison-Wesley, Massachusetts, 1979.

[Hsia *et al.*, 1993]
P. Hsia, A. M. Davis and D. C. Kung. Status Report: Requirements Engineering. *IEEE Software 10*, 6 (November 1993), pp. 75–79.

[Hull and King, 1987]
R. Hull and R. King. Semantic Database Modeling: Survey, Applications and Research Issues. *ACM Computing Surveys 19*, 3 (1987), pp. 201–260.

[Hwang and Lin, 1987]
C. L. Hwang and M. J. Lin. *Group Decision Making under Multiple Criteria*. Lecture Notes in Economics and Mathematical Systems Volume 281. Springer Verlag, 1987.

[IEEE-610.12, 1991]
IEEE-610.12. IEEE Standard Glossary of Software Engineering Terminology. 1991.

[IEEE-830, 1984]
IEEE-830. Guide to Software Requirements Specification. 1984. ANSI/IEEE Std. 830.

[Jackson, 1995]
M. Jackson. *Software Requirements & Specifications*. Addison Wesley, 1995.

[Jacobson *et al.*, 1992]
I. Jacobson, M. Christerson, P. Jonsson and G. Övergaard. *Object Oriented Software Engineering*. Addison Wesley, 1992.

[Jarke and Pohl, 1992]
M. Jarke and K. Pohl. Information System Quality and Quality Information Systems. In *Proc. of the IFIP 8.2 Working Conf. on the Impact of Computer-Supported Techniques on Information Systems Development*, Minneapolis, MN, June 1992.

[Jarke and Pohl, 1993a]
M. Jarke and K. Pohl. Establishing Visions in Context: Towards a Model of Requirements Processes. In *Proc. of the Intl. Conf. on Information Systems*, Orlando, FL, December 1993.

[Jarke and Pohl, 1993b]
M. Jarke and K. Pohl. Vision Driven System Engineering. In N. Prakash, C. Rolland and B. Pernici (Eds.), *Proc. of the IFIP WG 8.1 Conf. on Information System Development Process*, pp. 3–23, Como, Italy, September 1993. Elsevier B. V. (North Holland).

[Jarke and Pohl, 1994]
M. Jarke and K. Pohl. Requirements Engineering in 2001: (virtually) managing a changing reality. *Software Engineering Journal* (November 1994).

[Jarke *et al.*, 1992]
M. Jarke, S. Jacobs and K. Pohl. *Group-Decision Support und Qualitätsmanagement*, pp. 143–166. Erich Schmidt Verlag, 1992.

[Jarke *et al.*, 1993]
M. Jarke, K. Pohl, S. Jacobs, J. Bubenko, P. Assenova, P. Holm, B. Wangler, C. Rolland, V. Plihon, J. R. Schmitt, A. Sutcliffe, S. Jones, N. Maiden, D. Till, Y. Vassiliou, P. Constantopoulos and G. Spanoudakis. Requirements Engineering: An Integrated View of Representation, Process and Domain. In *Proc. of the Fourth European Software Engineering Conf.*, pp. 100–114, Garmisch-Partenkirchen, Germany, 1993. Springer-Verlag.

[Jarke *et al.*, 1994]
M. Jarke, K. Pohl, C. Rolland and J. R. Schmitt. Experience-Based Method Evaluation and Improvement: A Process Modeling Approach. In *IFIP WG 8.1 Conf. CRIS '94*, Maastricht, Netherlands, 1994.

[Jarke, 1990]
M. Jarke. DAIDA: Conceptual Modeling and Knowledge Based Support of Information Systems Development Processes. *Technique et Science Informatiques 9*, 2 (1990), pp. 122–133.

[Jarke, 1993]
M. Jarke (Ed.). *Database Application Development with DAIDA*. Springer-Verlag, 1993.

[Jeusfeld, 1992]
M. Jeusfeld. *Change Control in Deductive Object Bases*. INFIX Pub, Bad Honnef, Germany, 1992. (in German).

[Johnson and Feather, 1990]
W. L. Johnson and M. Feather. Building An Evolution Transformation Library. In *Proc. of the Twelfth Intl. Conf. on Software Engineering*, pp. 428–438, Nice, France, March 1990.

[Johnson *et al.*, 1992]
W. L. Johnson, M. S. Feather and D. R. Harris. Representation and Presentation of Requirements Knowledge. *IEEE Transactions on Software Engineering 18*, 10 (October 1992).

[Keller *et al.*, 1990]
S. E. Keller, L. G. Kahn and R. B. Panara. Specifying Software Quality Requirements with Metric. In R. H. Thayer and M. Dorfman (Eds.), *Systems and Software Requirements Engineering*, pp. 145–163. IEEE Computer Society Press — Tutorial, 1990.

[Klein, 1993]
M. Klein. Supporting Conflict Management in Cooperative Design Teams. *Group Decision and Negotiation 2*, 9 (1993), pp. 259–278.

[Krogstie *et al.*, 1995]
J. Krogstie, O. I. Lindland and G. Sindre. Towards a Deeper Understanding of Quality in Requirements Engineering. In *Proc. of the CAiSE '95*. Springer Verlag, LNCS 932, June 1995.

[Leite and Freeman, 1991]
J. C. S. P. Leite and P. A. Freeman. Requirements Validation Through Viewpoint Resolution. *IEEE Transactions on Software Engineering 17*, 12 (December 1991), pp. 1253–1269.

[Leite, 1989]
J. C. S. P. Leite. Viewpoint Analysis: A Case Study. In *Proc. of the Fifth Intl. Workshop on Software Specification and Design*, pp. 111–119, Pittsburgh, PA, 1989.

[Lenzen, 1994]
C. Lenzen. Object Oriented Design: Evaluation, Selection, and Implementation. Master's thesis, RWTH-Aachen, Germany, 1994. (in German).

[Loucopoulos and Champion, 1988]
P. Loucopoulos and R. Champion. Knowledge-Based Approach to Requirements Engineering Using Method and Domain Knowledge. *Knowledge-Based Systems 1*, 3 (1988).

[Loucopoulos and Karakostas, 1995]
P. Loucopoulos and V. Karakostas. *System Requirements Engineering*. McGraw-Hill, 1995.

[Lubars *et al.*, 1993]
M. Lubars, C. Potts and C. Richter. A Review of the State of the Practice in Requirements Modeling. In *Proc. of the First Intl. Symp. on Requirements Engineering*, San Diego, CA, January 1993. IEEE Computer Society Press.

[Luqi, 1993]
Luqi. How to Use Prototyping for Requirements Engineering. In *Proc. of the First Intl. Symp. on Requirements Engineering*, San Diego, CA, January 1993. IEEE Computer Society Press.

[Macaulay, 1993]
L. Macaulay. Requirements Capture as a Cooperative Activity. In *Proc. of the First Intl. Symp. on Requirements Engineering*, pp. 174–181, San Diego, CA, January 1993. IEEE Press.

[Maiden and Sutcliffe, 1992]
N. Maiden and A. Sutcliffe. Exploiting Reusable Specifications Through Analogy. *Communications of the ACM 35*, 4 (1992), pp. 55–64.

[Maiden *et al.*, 1994]
N. Maiden, A. Sutcliffe, D. Till, C. Taylor, H. Nissen, M. Jarke, K. Pohl, R. Dömges, P. Assenova, J. Bubenko, B. Wangler, P. Johannesson, G. Spanoudakis, K. Halkia, V. Plihon, C. Rolland, S. Si-Said and G. Grosz. Distributed Requirements Engineering within NATURE. NATURE-Report, 1994. submitted for publication.

[Maiden *et al.*, 1995]
N. Maiden, P. Assenova, P. Constantopoulos, M. Jarke, P. Johanneson, H. W. Nissen, G. Spanoudakis and A. Sutcliffe. Computational Mechanisms for Distributed Requirements Engineering. In *Proc. Seventh Intl. Conf. on Software Engineering and Knowledge Engineering,*, Knowledge Sciences Institute, June, 1995.

[Maiden, 1991]
N. Maiden. Analogy as a Paradigm for Specification Reuse. *Software Engineering Journal* (1991).

[Maiden, 1992]
N. Maiden. *Analogical specification Reuse during Requirements Analysis*. PhD thesis, City University London, 1992.

[Mannino and Tseng, 1989]
M. Mannino and V. Tseng. Inferring Database Requirements from Examples in Forms. In *Proc. of the Seventh Intl. Conf. on Entity-Relationship Approach*, pp. 391–405. Elsevier Publishers B. V. (North-Holland), 1989.

[Martin, 1990]
J. Martin. *Information Engineering Books*, Volume I-III. Prentice Hall, Englewood Cliffs, NJ, 1990.

[McMenamin and Palmer, 1984]
S. M. McMenamin and J. F. Palmer. *Essential System Analysis*. Yourdon Press, Prentice Hall, Englewood Cliffs, NJ, 1984.

[Meyer, 1985]
B. Meyer. On Formalism in Specifications. *IEEE Software* (January 1985), pp. 6–26.

[Miriyala and Harandi, 1991]
K. Miriyala and M. T. Harandi. Automatic Derivation of Formal Software Specifications From Informal Descriptions. *IEEE Transactions on Software Engineering 17*, 10 (October 1991), pp. 1126–1142.

[Monarchi and Puhr, 1992]
D. E. Monarchi and G. I. Puhr. A Research Typology for Object-Oriented Analysis and Design. *Communications of the ACM 35*, 9 (September 1992), pp. 35–47.

[Mylopoulos *et al.*, 1990]
J. Mylopoulos, A. Borgida, M. Jarke and M. Koubarakis. Telos: Representing Knowledge about Information Systems. *Transactions on Information Systems 8*, 4 (1990), pp. 325–362.

[Nakajima and Davis, 1994]
T. Nakajima and A. M. Davis. Classifying Requirements Errors for Improved SRS Reviews. In *Proc. of the First Intl. Workshop on Requirements Engineering: Foundation of Software Quality*, pp. 88–100, Utrecht, The Netherlands, 1994. Augustinus-Verlag.

[Nissen *et al.*, 1996]
H. Nissen, M. Jeusfeld, M. Jarke, G. V. Zemanek and H. Huber. Requirements Analysis from Multiple Perspectives: Experience with Conceptual Modeling Technology. *IEEE Software, March* (1996).

[Nuseibeh *et al.*, 1994]
B. Nuseibeh, J. Kramer and A. Finkelstein. A Framework for Expressing the Relationship Between Mulitple Views in Requirements Specification. *IEEE Transaction of Software Engineering 20*, 10 (1994), pp. 760–773.

[Olle *et al.*, 1988]
T. W. Olle, J. Hagelstein, I. G. MacDonald, C. Rolland, H. S. Sol, F. J. V. Assche and A. A. Verrijn-Stuart. *Information Systems Design Methodologies*. Addison Wesley, Wokingham, England, 1988.

[Pohl and Haumer, 1995]
K. Pohl and P. Haumer. HYDRA: A Hypertext Model for Structuring Informal Requirements Representations. In *Proc. of the Second Intl. Workshop on Requirements Engineering: Foundation of Software Quality (REFSQ'95)*, June, 12–13, Jyväskylä, Finland, 1995. Augustinus, Aachen, Germany.

[Pohl and Peters, 1995]
K. Pohl and P. Peters (Eds.). *Proc. of the Second Intl. Workshop on Requirements Engineering: Foundation of Software Quality*, Jyväskylä, Finland, 1995. Augustinus-Verlag.

[Pohl *et al.*, 1994a]
K. Pohl, G. Starke and P. Peters (Eds.). *Proc. of the First Intl. Workshop on Requirements Engineering: Foundation of Software Quality*, Utrecht, The Netherlands, 1994. Augustinus-Verlag.

[Pohl *et al.*, 1994b]
K. Pohl, G. Starke and P. Peters. REFSQ'94: Workshop Summaries. *EMISA Forum*, 2 (August 1994), pp. 87–95. also appeared in ACM-Sigsoft Notes, Jan. 1995.

[Pohl, 1993]
K. Pohl. The Three Dimension of Requirements Engineering. In *Proc. of the Fifth Intl. Conf. on Advanced Information Systems Engineering*, pp. 275–292, Paris, France, June 1993. Springer-Verlag.

[Pohl, 1994]
K. Pohl. The Three Dimension of Requirements Engineering: A Framework and its Application. *Information Systems 3*, 19 (June 1994), pp. 243–258.

[Pohl, 1996a]
K. Pohl. PRO-ART: Enabling Requirements Pre-Traceability. In *Proc. of the Second Intl. Conf. on Requirements Engineering (ICRE'96)*, Colorado Springs, CO, 1996. IEEE Computer Society Press.

[Pohl, 1996b]
K. Pohl. *Process Centered Requirements Engineering*. RSP marketed by J. Wiley & Sons Ltd., UK, 1996.

[Potts and Bruns, 1988]
C. Potts and G. Bruns. Recording the Reasons for Design Decisions. In *Proc. of the Tenth Intl. Conf. on Software Engineering*, Singapore, April 1988.

[Potts *et al.*, 1994]
C. Potts, K. Takahashi and A. I. Anton. Inquiry-Based Requirements Analysis. *IEEE Software 12*, 2 (1994).

[Puncello *et al.*, 1988]
P. P. Puncello, P. Torrigiani, F. Pietri, R. Burlon, B. Cardile and M. Conti. ASPIS: A Knowledge-Based CASE Environment. *IEEE Software* (March 1988), pp. 58–65.

[Ramesh and Dhar, 1992]
B. Ramesh and V. Dhar. Supporting Systems Development by Capturing Deliberations During Requirements Engineering. *IEEE Transactions on Software Engineering 18*, 6 (1992), pp. 498–510.

[Ramesh and Edwards, 1993]
B. Ramesh and M. Edwards. Issues in the Development of a Requirements Traceability Model. In *Proc. of the First Intl. Symp. on Requirements Engineering*, San Diego, CA, January 1993. IEEE Computer Society Press.

[Ramesh, 1993]
B. Ramesh. A Model of Requirements Traceability for Systems Development. Technical report, Naval Postgraduate School, Monterey, CA, September 1993.

[Reisig, 1991]
W. Reisig. *Petrinetze: Eine Einführung*. Springer Verlag, Studienreihe Informatik, 1991. in German.

[Reubenstein and Waters, 1991]
H. B. Reubenstein and R. C. Waters. The Requirements Apprentice: Automated Assistance for Requirements Acquisition. *IEEE Transactions on Software Engineering 17*, 3 (March 1991), pp. 226–240.

[Rolland and Prakash, 1993]
C. Rolland and N. Prakash. Reusable Process Chunks. In *Proc. of the Intl. Conf. Database and Expert Systems Applications*, Prague, Slovakia, September 1993.

[Rolland and Proix, 1992]
C. Rolland and C. Proix. A Natural Language Approach for Requirements Engineering. In *Proc. of the Fourth Intl. Conf. on Advanced Information Systems Engineering*, LNCS 593, 1992.

[Roman *et al.*, 1984]
G. C. Roman, M. J. Stucki, W. E. Ball and W. D. Gillett. A Total System Design Framework. *IEEE Computer 17*, 5 (May 1984), pp. 14–26.

[Roman, 1985]
G. C. Roman. A Taxonomy of Current Issues in Requirements Engineering. *IEEE Computer 18*, 4 (1985), pp. 14–22.

[Rumbaugh *et al.*, 1991]
J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen. *Object Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, NJ, 1991.

[Shlaer and Mellor, 1988]
S. Shlaer and M. Mellor. *Object Oriented System Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1988.

[Smith and Smith, 1977]
J. M. Smith and D. C. P. Smith. Database Abstractions: Aggregation and Generalization. *ACM Trans. on Database Systems 2*, 2 (1977), pp. 105–133.

[Sommerville *et al.*, 1993]
I. Sommerville, T. Rodden, P. Sawyer, R. Bentley and M. Twidale. Integrating Ethnography into the Requirements Engineering Process. In *Proc. of the First Intl. Symp. of Requirements Engineering*, pp. 165–173, San Diego, CA, 1993. IEEE Computer Society Press.

[Spivey, 1990]
J. M. Spivey. An introduction to Z and Formal Specifications. *Software Engineering Journal 4*, 1 (1990), pp. 40–50.

[Stein, 1994]
W. Stein. *Objektorientierte Analysemethoden: Vergleich, Bewertung, Auswahl*. BI Wissenschaftsverlag, 1994. (in German).

[Sutcliffe and Maiden, 1990]
A. Sutcliffe and N. Maiden. Software Reusability: Delivering Productivity Gains or Short Cuts. In *Proc. of the INTERACT*, pp. 948–956. Elsevier B. V. (North Holland), 1990.

[Sutcliffe, 1991]
A. Sutcliffe. Object Oriented Systems Analysis: The Abstract Question. In *Proc. of the IFIP WG 8.1 Conf. on the Object Oriented Approach in Information Systems*, Quebec City, Canada, 1991.

[Svoboda, 1990]
C. P. Svoboda. Structured Analysis. In R. H. Thayer and M. Dorfman (Eds.), *Systems and Software Requirements Engineering*, pp. 218–227. IEEE Computer Society Press — Tutorial, 1990.

[Teichroew and Hershey, 1977]
D. Teichroew and E. A. Hershey. PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems. *IEEE Transaction of Software Engineering 1*, 3 (1977), pp. 41–48.

[TSE, 1977]
Special Issue on Requirements Engineering. *IEEE Transaction on Software Engineering 1*, 3 (1977).

[Ward, 1986]
P. T. Ward. The Transformation Schema: An Extension of Data Flow Diagram to Represent Control and Timing. *IEEE Transaction on Software Engineering 12*, 2 (1986), pp. 198–210.

[Wasserman *et al.*, 1986]
A. I. Wasserman, P. A. Pircher, D. T. Shewmake and M. L. Kersten. Developing Interactive Information Systems with the User Software Engineering Methodology. *IEEE Transactions of Software Engineering 12*, 2 (1986), pp. 326–345.

[Wing, 1987]
J. M. Wing. Writing Larch Interface Language Specification. *IEEE Transactions on Programming Languages and Systems* (January 1987), pp. 1–24.

[Wing, 1990]
J. M. Wing. A Specifier's Introduction to Formal Methods. *IEEE Computer 23*, 9 (September 1990), pp. 8–24.

[Wright, 1991]
S. Wright. Requirements Traceability – What? Why? and How? In *Proc. of the Colloquium on Tools and Techniques for Maintaining Traceability during Design*, pp. 1–2, London, UK, December 1991. IEE Professional Group C1 (Software Engineering), IEE, London, UK.

[Yourdon, 1989]
E. Yourdon. *Modern Structured Analysis*. Prentice Hall, Englewood Cliffs, NJ, 1989.

[Zave, 1990]
P. Zave. A Comparison of the Major Approaches to Software Specification and Design. In R. H. Thayer and M. Dorfman (Eds.), *Systems and Software Requirements Engineering*, pp. 197–199. IEEE Computer Society Press — Tutorial, 1990.

[Zave, 1991]
P. Zave. An Insider's Evaluation of PAISLey. *IEEE Transactions on Software Engineering 17*, 3 (March 1991), pp. 212–225.

10 Basic Readings for Further Studies

Alan M. Davis: *"Software Requirements: Objects, Functions, & States"*, Prentice Hall, New Jersey, 1993, (2nd Edition).

This book covers an excellent introduction to the thoughts of requirements engineering and discusses most of the major methods and approaches. It also contains on of the largest annotated bibliography on requirements engineering (over 700 entries). Good for novice and as basis for teaching material.

Merlin Dorfman and Richard H. Thayer: *"Standards, Guidelines and Examples on System and Software Requirements Engineering"*, IEEE Computer Society Press – Tutorial, 1990.

This tutorial provides a comprehensive selection of the existing standards and guidelines for requirements specifications. Besides defining the structure and the content of a "good" specification it gives valuable hints in developing and maintaining the requirements documents.

Donald C. Gause and Gerald M. Weinberg: *"Exploring Requirements: Quality Before Design"*, Dorset House Publishing, New York, 1989

In contrast to most other contributions, this book deals mainly with the social and cognitive aspects of requirements engineering. Many useful hints and advises are given on how to successfully perform requirements engineering in practice.

Pericles Loucopoulos and Vassilios Karakostas: *"System Requirements Engineering"*, McGraw-Hill, 1995.

The book is organized along three concurrent tasks of requirements engineering: elicitation, documentation, and validation/verification. These task are not described from a methodical viewpoint and therefore "cookbook" solutions are avoided. Instead the important issues, models, and tools applicable for each task are discussed.

Klaus Pohl: *"Process Centered Requirements Engineering"*, RSP distributed by John Wiley & Sons, UK, 1996.

In this book a comprehensive framework for requirements pre-traceability and experience based process (method) improvement is proposed. The applicability of the framework is demonstrated by a process centered requirements engineering environment, called PRO-ART. PRO-ART records the traces in a central process repository, offers interactive and situated methodical advise to the requirements engineer, and supports consistent change integration.

Richard H. Thayer and M. Dorfman: *"Systems and Software Requirements Engineering"*, IEEE Computer Society Press – Tutorial, 1990.

This tutorial covers a cohesive collection of papers which touch on almost all important aspects of requirements engineering. Enriched by the case studies on real world experiences in requirements engineering this tutorial offers a good overview on the field.

Recent research contributions to the field of requirements engineering can be found in:

- IEEE Software, March, 1996
- Proceedings of the Second Intl. Conf. on Requirements Engineering (ICRE'96), IEEE Computer Society Press, 1996
- Proceedings of the Second IEEE Intl. Symposium on Requirements Engineering (RE'95), IEEE Computer Society Press; 1995
- Proceedings of the Second Intl. Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'95), Augustinus Verlag, Aachen, Germany, 1995
- IEEE Software, March, 1994
- Proceedings of the First Intl. Conf. on Requirements Engineering (ICRE'94), IEEE Computer Society Press, 1994
- Proceedings of the First Intl. Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ'94), Augustinus Verlag, Aachen, Germany, 1994
- Proceedings of the First IEEE Intl. Symposium on Requirements Engineering (RE'93), IEEE Computer Society Press, 1993
- IEEE Transaction on Software Engineering, No. 3, Vol. 17, 1991