ESPRIT Project 6353 (NATURE), Report AC-92-1

Theories Underlying Requirements Engineering: An Overview of NATURE at Genesis

Matthias Jarke, Informatik V, RWTH Aachen, Ahornstr. 55, 5100 Aachen, Germany
Janis Bubenko, SISU-ISE, Isafjordsgatan 26, 1250 Kista, Sweden
Colette Rolland, Universite Paris 1, rue de al Sorbonne 17, 75231 Paris, France
Alistair Sutcliffe, Business Comp., City University, London ECIV OHB, UK
Yannis Vassiliou, ICS-FORTH, Dedalou 36, 71110 Heraclion, Greece§

NATURE is a collaborative basic research project on theories underlying requirements engineering funded by the ESPRIT III program of the European Communities. Its goals are to develop

- a theory of knowledge representation that embraces subject, usage and development worlds surrounding the system, including 'expressive freedoms'

- a theory of domain engineering that facilitates the identification, acquisition and formalisation of domain knowledge as well as similarity-based matching and classifying of software engineering knowledge

- a process engineering theory that promotes context and decision-based control of the development process.

These theories are integrated and evaluated in a prototype environment constructed around an extended version of the conceptual modeling language Telos.

---

§ This work is supported in part by ESPRIT Basic Research Project 6353 (NATURE). Besides the authors as principal investigators, the following people have contributed to the early phases of the project: S. Jacobs, K. Pohl (RWTH Aachen), Peter Holm, Benkt Wangler (SISU), G. Grosz, J.-R. Schmitt (Université Paris 1), S. Jones, N. Maiden, D. Till (City University), P. Constantopoulos, G. Spanoudakis (FORTH). We are also grateful for fruitful discussions with our transatlantic collaborators, in particular John Mylopoulos.

A version of this paper is presented as a plenary talk to the First International IEEE Symposium on Requirements Engineering, San Diego, Ca, Jan. 4-6, 1993.

1 Introduction

Requirements engineering is perceived as an area of growing importance. As we learn more about the nature of requirements information, the traditional task of *requirements capture* as an early stage in the life cycle is complemented by several new applications of the models it produces, and of the processes that generate these models:

- the explicit and computer-supported *use of requirements*, especially non-functional ones, to drive design decisions in the systems development process
- the *reverse engineering* of requirements models as a central part of systems integration
- the *reuse of requirements models and processes* in the evolution of a system family or in the development of new applications; also the support of such reuse by standard *reference models*
- the *re-engineering* of systems and business processes based on requirements models.

All of these are no longer relevant only to traditional information systems development. They also invade computer-integrated manufacturing, office systems, process control, and other areas where systems interact intensively with their environment.

Research in requirements engineering has been converging on a set of fundamental problems which require theoretical development to progress from intuitive ideas and mono-disciplinary approaches.

Formal specification has been the prime and necessary endeavour. However, in spite of acceptance and some practice, the industrial development community has been slow to adopt these methods and there is growing realisation that training may not be the answer. Furthermore, as shown in figure 1, the requirements specification problem is now conceptualised as the coexistence of informal and formal languages and the orderly transition between the two. This focus on integration with the real world makes requirements engineering a much "softer" task than the other steps in software development.

In an attempt to bridge the gap from linguistic expression to formal modelling, layered semantic specification languages have been developed. These still require a formal notation and restrict modelling to the perspective of what is to be developed.

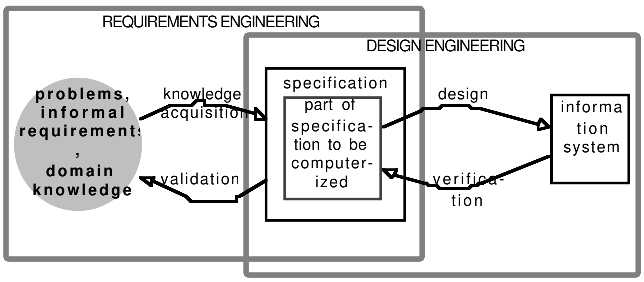

Figure 1: Requirements vs. design engineering

In reality, software applications are embedded in a world of organisations, people and usage. The first problem, therefore, is to expand the modelling horizon both vertically to span informal and formal expression, and horizontally to model the system in its context of domain, usage, and evolution (cf. also [Feather 1987]).

The influence of domain knowledge on requirements analysis and software specification has been recognised by endeavours of domain analysis [Freeman 1987], the use of domain knowledge in intelligent assistant tools for requirements analysis (ASPIS [Punchello et al.

1988]), and development of domain knowledge frames in several reuse-oriented projects (e.g. cliches [Reubenstein & Waters 1991], reusable patterns [Biggerstaff & Richter 1987], generalized application frames [Constantopoulos et al. 1992]). Domain knowledge is also seen as a critical part of the description problem in reverse engineering. Current domain descriptions are intuitive and inconsistent, preventing any generalisation or integration of this research. A formal theory is required to define what a domain is and what constitutes domain knowledge. Such a theory would then provide guidance to such important questions as how to structure and evaluate the reference models currently under development for numerous domains.

The third convergence is process modelling, the influence of domain knowledge on transformational activities [Grosz & Rolland 1990] and analysis of the context in which software engineering activities should be applied. The process modelling community has progressed from sequential models of activity to dynamic views of the software process, however, the context of activity is poorly understood.

The European community has recognized the growing importance of basic research in this area by initiating a collaborative project called NATURE (Novel Approaches to Theories Underlying Requirements Engineering) which involves the institutions of the authors in five European countries. This paper provides an overview of the project goals and approach. Since the project has just recently begun, technical results are largely based on previous work. Section 2 presents the general approach, centering around the idea of several interrelated worlds of software information. Sections 3 to 5 present specific knowledge representation, domain, and process theories and section 6 concludes with a brief summary of prototype implementation and theory evaluation strategies.

2 Basic Approach and Methodology

The basic premise of this work is that requirements engineering differs from system specification in that it focuses on the embedding of systems in their environment rather than on the prescription of the system's functionality or structure. Specifically, our basic ontology of requirements engineering (cf. figure 2 [Jarke 1990]) suggests that an information system can be characterized by its embedding in at least three different parts of the world. Making these worlds explicit does not only provide a certain degree of guidance in building a requirements model but also allows the association of non-functional goals with certain stakeholders.

The system must represent its *subject world* with accuracy, timeliness, well-organized abstractions, etc. The system must fit into its *usage world* with task-oriented functionality, be compatible with users' models and conform to good human factors principles of design. The system must evolve in its *development world* with reasonable time and cost, consistent with standard procedures, and possibly under reuse of existing experience, knowledge and products.

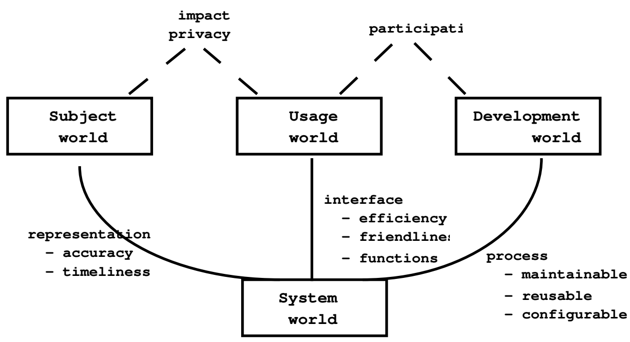

Figure 2: Embedding systems in the world

Previous work has typically focused on one of these worlds at a time: database modeling on the subject world, human-computer interaction and office systems research on the usage world, and software engineering on the development world.

The relationships among the different "worlds" must not only be understood but actively designed. They change the interrelationships among the external sub-worlds. For example, a police information system may change the relationship between its usage world -- the police -- and its subject world -- the rest of the population. Similarly, the communication between users (usage world) and developers (development world) defines much of the software process.

The broad view taken here precludes a "complete" capturing of all relevant knowledge. Indeed, many of the implicit premises of formal methods are *not* present in requirements engineering. It is *productive* to maintain inconsistencies (conflict fosters creativity), to be incomplete (oversimplification improves understanding), etc. [Feather & Fickas 1991]. It is therefore unclear if formal specification languages such as VDM [Jones 1986] or Z [Spivey 1988] are applicable to requirements engineering as defined here, although they play an important role as possible languages for the system world. In fact, it is doubtful whether any single formalism can capture the richness of the requirements engineering task.

The literature has advanced a number of proposals what we can do in such situations:

- Rely on intuition and communication of people in addition to formalism.
- If you cannot formalize the product, try to formalize the process.
- If you cannot do it in general, offer structure for specific domains.
- If you cannot be sure you did it right, provide efficient backtracking through documentation.

Our denial of the existence of a complete formalism does not mean that it is impossible to formalize any of the above aspects. Indeed, a unification of the above ideas is the main goal of the project. However, the tools are different, stemming from a combination of extended knowledge representation technology (e.g., with exceptions and multiple conflicting viewpoints), semi-formal graphics, and informal hypermedia representations. To improve communication, the process of requirements engineering includes a frequent two-way transition between informal and formal, functional and non-functional, product and process-oriented aspects. A comprehensive theory of this problem should

- be compatible, on the formal side with emerging formal specification methods and reasoning mechanisms; and on the informal side with industrial practice
- cover domain as well as method and process information
- be made operational using state-of-the-art representation, reasoning, and management languages and tools.

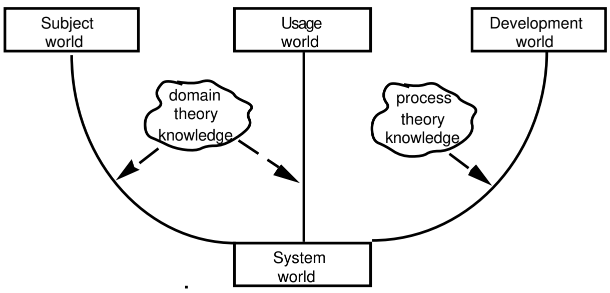

Figure 3: Relating domain and process to the knowledge representation framework

The project is organized as three interacting streams, one concerned with domain abstraction, the second with process, the third with representation and reasoning. Figure 3 indicates how our global model of requirements information can be used as an integration concept for these streams. To demonstrate this integration more effectively, prototypical models and tools will be developed around the same linguistic framework (Telos [Mylopoulos et al. 1990]) and the same management environment (ConceptBase [Jarke ed. 1992]).

3 Knowledge Representation Theory

Requirements can be represented using formal specification languages (e.g. VDM and Z), semi-formal methods (e.g. data flow or entity-relationship diagrams]) or informal aids (hypertext, graphic, examples). Each of these representational forms has certain strengths and weaknesses, so a combination is necessary.

Formal specification languages offer the advantage of unambiguous, precise reasoning support, and the clear representation of interrelations between requirements. They have been used mostly for describing the functions and the internal behaviour of systems.

Semi-formal methods are foremost a communication device between users and developers. They often have to represent interrelations between requirements through comments in natural language, can be ambiguous, and only offer limited reasoning support.

Informal representations of requirements like text, pictures, or animations have not yet been considered for integration into formal methods, because there was no clear opinion how to interrelate this kind of information. Hypertext offers a opportunity to do this (e.g. ARIES [Johnson & Harris 1990]; gIBIS [Conklin & Begeman 1988]). gIBIS, a hypertext system based on the Issue Based Information Systems (IBIS) method, was designed to facilitate the capture of early design deliberations. Research has shown that the integration of hypertext and the IBIS-method offers many advantages. However, only the communication structure, not the contents of communication is supported [Ramesh & Dhar 1992].

Formal specifications are normally expected to be complete, consistent and unambiguous. However, during the initial definition and revision of formal requirements, they are typically fragmented, contradictory, incomplete, inconsistent and ambiguous. Furthermore, the expressions may include various levels of abstraction (concrete examples, general properties, scripts etc.) and different types of styles (text, graphic, formula, notations). Since formal requirements are built out of non-formal, the acquisition process must allow many freedoms (incompleteness, inconsistency, redundancy, ambiguity, different levels of abstraction, heterogeneous forms of expressions [Balzer 1991, Feather & Fickas 1991]). Only a few existing systems support the transition between formal and non-formal requirements and offer parts of the necessary freedoms (e.g. KATE [Fickas 1987]; Requirements Apprentice [Reubenstein & Waters 1991]). However, they neither represent non-formal knowledge (forms, pictures, text etc.) nor its interrelation to the acquired formal knowledge in an adequate manner. Semi-formal and non-formal methods offer the necessary freedom at the beginning of the acquisition process, but have not much support for reasoning about the specified requirements. They are therefore not able to support the requirements process in a later stage.

Initially, requirements modeling was completely informal, consisting of text documentation and drawings. In the late 1970s, *semi-formal techniques* were proposed which combined graphical notations with an implicitly understood but rarely formalized semantics. This stage -- typified by Entity-Relationship and Data Flow diagrams -- represents the commercial state-of-the-art today. The lack of formal semantics has inhibited integration of different viewpoints (data, process, user interface) as well as automated propagation of requirements into designs. Moreover, semi-formal techniques are sometimes not *informal* enough: users need examples, animations and prototyping whose automatic generation pre-supposes a formal understanding of semi-formal representations.

Borgida et al. [1985] proposed the use of formal knowledge representation as a backend to semi-formal techniques. Besides the advantages of formalization in terms of consistency and completeness checking, mapping support and validation assistance, they emphasized organizational principles such as aggregation (composability of requirements), generalization (avoiding redundancy by inheritance), and classification (meta-modeling). Several formalizations of specific semi-formal approaches were proposed. Perhaps the best-known are Greenspan´s formalization of SADT in RML [Greenspan 1984] and the formalization of an extended entity-relationship model in ERAE [Hagelstein 1988]. This activity continues with object-oriented requirements models such as Objects with Roles [Pernici 1990], the MORSE language used in the database design tool SECSI [Bouzeghoub & Métais 1991], and the TEMPORA models [Loucopoulos et al. 1991] which extend Structured Analysis and Entity-Relationship by temporal aspects. All of these systems (some on paper, others in various stages of implementation) support a fixed metamodel, using logic as the basis for formalization.

However, the formalizations of individual semi-formal techniques must be integrated, and the need for adapting generic representations to particular domains and process structures became evident. A third generation of languages for requirements modeling is therefore emphasizing extensibility and adaptability. This necessitates the availability of one or more *meta-levels* in the language system such that data and process models can be user-defined in a common framework.

A few meta-modeling environments have been designed. One group, exemplified by the MetaEdit tool [Smolander et al. 1991], emphasizes the dynamic creation of graphical user interfaces for requirements meta-models but only supports a pre-defined set of constraints for semantics definition. *Telos* [Mylopoulos et al. 1990], a substantial generalization of RML, allows the association of predicative deduction rules and integrity constraints with meta classes, thus supporting semantic definition and consistency checking of multiple interacting metamodels.

Telos cannot only manage a requirements model as an evolving knowledge base but also the available metadata and process models. Metamodels can be specialized to particular domains and methodologies. This provides the basis for integrating domain and process engineering theories into a knowledge representation framework for requirements engineering. This statement is backed by experience which covers, among other things:

- the integration of multiple language models and mappings between them, also the propagation of change across multiple representations [Jarke et al. 1992]
- the integrated evolution of multiple levels of granularity through meta models and tools for version and configuration management [Rose et al. 1991]
- formally supported development of informal hypermedia documentation [Eherer & Jarke 1991].

The version of Telos implemented in the knowledge base management system ConceptBase [Jarke ed. 1992] appears to cover a large proportion of the features to be expected from an extensible knowledge representation language for requirements modeling:

- full support for all three abstraction principles (except automatic object classification [Borgida et al. 1989])
- no distinction is made between attributes and relationships -- attributes are first-class objects
- full extensibility through the combination of meta–classing with deductive rules and integrity constraints
- integration of temporal information about the evolution of the modelled worlds as well as about the history of the database content

- data model that lends itself to hypertext-like switching between graphical and textual (frame-oriented) representations, thus being fully compatible with the usual semi-formal graphical techniques
- client-server KBMS with support for teamwork in requirements modeling and process management which is itself based on appropriate meta models.

However, rethinking and further development is needed in at least three areas.

Firstly, Telos may, in its present form, have gone too far in formalizing requirements engineering -- it provides too few of the freedoms a requirements modeling team needs to capture or reuse the actual requirements of some organization. Temporary oversimplification, differences in opinion, and incompleteness of requirements are not adequately addressed although means are provided to work "around the system".

Secondly, visualization should extend beyond semi-formal graphics to natural language texts, informal drawings, even video-scenes of possible system usage. Though we have already prototyped an integration of Telos with a hypermedia system [Eherer & Jarke 1991], we have scarcely begun to understand the interplay of informal and formal representations in requirements engineering.

Finally, in possible contrast to what was said before, requirements engineers may need more content-oriented formal guidance in developing requirements models. As a first step in this direction, we have been defining a set of "worlds" whose relationships to the planned system should be discussed in a requirements process (cf. fig. 2): the subject world, the usage world, and the development world. We claim that explicit consideration of all these worlds is necessary but at least not covered explicitly in any existing methodologies. Moreover, we hypothesize that the explicit inclusion of information about such worlds and their relationships to the intended system also addresses an important open question in requirements engineering: the representation and exploitation of non-functional requirements. We have already shown some ideas on this in fig. 2. Last not least, the distinction of interrelated concerns expressed in the "worlds" model can serve as a starting point for more detailed domain models of requirements and their development and usage processes.

In the remainder of this section, we elaborate our plans for dealing with these extensions, organized by the kind of "freedom" they provide.

Extension of the Telos Formalism. Considering the intended use of requirements models in requirements capture, reuse, and re-engineering, several extensions to the present Telos version are needed, without however, giving up the advantages of formal semantics and automated reasoning support. Mostly, these concern completeness and consistency of requirements models.

The current version of Telos requires referential integrity although it does allow to keep the definition of reference objects to remain completely abstract. An alternative to explicitly defining the referenced object is to have them defined automatically and added to a list of problems to be taken care of. A research issue is how that will influence the process and at what points all references should be sufficiently specified. More generally, the question of automatic recovery from integrity violations will have to be investigated; to do this, the system may have to make guesses about reasonable constraint repairs, which may have to be retracted consistently later.

We argued earlier that temporary inconsistency of requirements specifications may be productive as a basis for finding out what are the real goals, what is important, etc. Views have been a traditional mechanism to allow for this kind of inconsistency. However, there is also another use of inconsistency: oversimplification of models to make them more understandable. We envision a layered knowledge base in which the higher layers represent the general rules whereas the lower layers may contain exceptions to them as well. Of course, this has been a traditional subject of default representations such as nonstrict inheritance. Only recently, the semantics of such models is becoming more clearly understood so that relatively clean mechanisms for non-monotonicity and hypothetical analyses can be developed. For example, querying mechanisms and visualization tools should produce multiple oversimplified representations of a complex requirements base.

Since requirements engineering involves specifying the change from some unsatisfactory current situation to a more desirable future, also because reorganization of abstractions may be often needed in a requirements process, these non-monotonic structures must support powerful manipulation and restructuring operations as well as the better-known querying and presentation operations.

Integrating Formal, Semi-Formal, and Informal Representations. The mapping between the well-known semi-formal representations (DFDs, E-R diagrams, etc.) and an underlying knowledge base is by now fairly well understood. Formally, such representations are views on the conceptual requirements model which -- at the end of the requirements process -- should be complete and consistent. Thus, a semi-formal graphics tool can be used to generate a rough sketch of a data or process model which can be automatically translated in an initial version of a knowledge base. A remaining problem is how and when to reconciliate the different viewpoints, also how to exploit conflict detection for the generation of fruitful discussions rather than suppression of differences.

The relationship between formal and informal representations is much less understood. As a first attempt to classify the issues, we distinguish between static and dynamic aspects.

Statically, a formal model can view an informal object (such as an image or a video clip) as an uninterpreted node in a Telos model described by some attributes which give an inkling about its content. This is a very familiar approach in software databases where the actual software is stored in files and the database contains a description of the software, plus a pointer to the file. Part of the description may be information about the creator of the informal object, in a cooperative setting also its role as an utterance in a particular discourse (e.g., an argument in a debate). Thus, the formal object can be considered an access path to the informal object. Conversely, the informal object is a visualization or animation of the formal description which is easier to understand.

Dynamically, we have to consider the transition between formal and informal representations. In the creation process, the transition from informal to formal corresponds to the classical knowledge acquisition viewpoint, whereas the transition from formal to informal is associated with validation (cf. fig. 1). Briefly, the informal-to-formal transition corresponds to an abstraction process, the other direction to concretization and authoring.

Many individual methods for both directions have been investigated, including manual ones (with or without explicit representation of the abstraction process that takes place) and automatic ones (such as natural language understanding resp. generation). Rather than devising specific techniques (this is done in domain and process engineering as well as by many specialized projects), we are interested in the representation of these static and dynamic relationships in a concise and usable manner and in furthering the basic understanding of their roles in the requirements process. The integration of hypermedia and knowledge-based approaches seems to provide a suitable representational and technological basis for this work.

Representing and Using Quality Requirements. Though non-functional or quality requirements play a crucial role in requirements engineering, they have traditionally been poorly understood. Important research questions concern both the acquisition and the usage of non-functional requirements. Based on previous results, we plan to study several fundamentally different but complementary acquisition methods.

One proposition is that non-functional requirements are those that are expressed in a communication among designers; argumentation structures such as Rittel´s IBIS or Toulmin´s Argumatics provide the formal framework for introducing non-functional requirements in this manner [Conklin & Begeman 1988, Hahn et al. 1991, Ramesh & Dhar 1992]. The problem with this approach is that requirements come in haphazard and unstructured manner.

The other extreme is a detailed study of particular classes of non-functional requirements considered important *in general* for software development. Research in the requirements of efficiency (e.g., complexity theory, queuing theory) and reliability (e.g., fault tolerance, formal verification, recovery) has progressed quite far. Recently, some information system specific requirements such as data accuracy and security have also been investigated [Chung et al. 1991, Mylopoulos et al. 1992].

A third approach is the half-axiomatic, half-empirical derivation of non-functional requirements from process-oriented theories of the firm, such as the theory of Critical Success Factors developed by Jack Rockart at MIT [Rockart 1979] or the theory of value-added chains introduced by Michael Porter at Harvard [Porter 1985]. Such business-oriented domain theories pave the way for achieving impact beyond information systems. Managers can analyse and re-organize strategic processes inside or across organizations, with or without introducing information systems technology in the process.

Requirements specification must pay attention to the organizational and business environment. For instance, the subject world model may incorporate concepts like organizational actors, positions, channels, and business functions, whereas a usage world model on the organizational level may be defined in terms of users, IS services, IS use acts etc. [Scheer 1991]. Adequate mappings between these and other concepts must be defined in order to allow for the specification of e.g. security requirements.

From a knowledge representation and reasoning viewpoint, the representation of individual non-functional requirements is not difficult if we introduce ordered domains in the language (so that we can distinguish good and bad achievement). More difficult is the representation and reasoning for (a) deriving and selecting process alternatives from individual requirements, and (b) for handling trade-offs in the process. Some experiences in multi-criteria group decision support [Jarke 1988] may assist in approaching a solution to these very difficult problems.

Integration of Domain and Process Knowledge. Non-functional requirements are closely linked to questions of domain analysis and process engineering. Systems development, maintenance, and usage processes should all be organized as to ensure continued quality (as perceived from the usage, the subject, and the development world), and these quality criteria can be expected to depend on the domain at hand. Very little is known about how to do this; we hope to gain some insight from a parallel project on managing life-cycle wide quality assurance in industrial engineering [Jarke & Pohl 1992].

A second, hopefully "easier" issue related to the integration of domain and process knowledge is that the knowledge representation and reasoning mechanisms provided by our approach have to be powerful enough to capture the essential specializations needed for representing domain knowledge and dynamics adequately and concisely. Meta modeling is the most crucial requirement for such a language, and we have argued that we are approaching a reasonably good understanding both of the formalisms and of their presentation at the user interface level. Nevertheless, it is well-known that too general such mechanisms lead to intractability or even undecidability of the associated reasoning tasks, whereas domain-specific operators may reduce such problems. A construction of extensibility that can resolve this conflict is still an unsolved problem [Borgida 1991].

4 Domain Theory

The importance of domain knowledge has been recognised from two directions. First, cognitive studies of software engineering have demonstrated that experts use a memory schema of domain knowledge to help construction of conceptual models ([Guindon & Curtis 1988, Guindon 1990, Sutcliffe & Maiden 1992]). Second, there has been a progressive growth in semantic richness of specification languages to model more domain concepts. The latter trend, manifest in semantic specification languages such as Telos, has culminated in the recognition of different descriptive worlds (cf. section 2) in which domain knowledge is necessary for effective system development.

Several studies have demonstrated that domain knowledge is employed with analogical reasoning by software engineers [Vitalari & Dickson 1983, Sutcliffe & Maiden 1990]. Experimental studies have shown that analogical matched specifications can be reused and that abstraction is used by experts as strategy for understanding and matching application domains. A model of domain abstraction appears to be necessary for retrieving application domain knowledge from memory and then understanding the implications of that knowledge in a new context. Further studies of the matching problem have demonstrated that software engineering problems can be described as abstract models and analogically matchedusing a meta schema of goal related, structural and domain knowledge. This approach synthesises concepts of templates with object-oriented methodology and Gentner's [1983] structure mapping theory which defines analogy as the matching of a structured set of propositions.

Within the NATURE project, two aspects of domain theory are investigated: its basic structuring principles, and its usage in similarity-based reuse.

Principles of Domain Abstraction. The domain theory aims to describe the knowledge structures people develop, and ultimately remember, when they are investigating problems. Abstractions for various domains have been proposed as templates for analogical transfer of knowledge between domains belonging to the same class [Gick & Holyoak 1983, Greiner 1988, Reubenstein 1990]. The assumptions of our preliminary theory [Maiden 1991, Sutcliffe 1991] are drawn from cognitive models of memory: models of natural categories [Rosch 1991], hierarchical memory schemas and categories of dynamic memory [Schank 1982]. These assert in slightly different forms that human memory is organised in an informal hierarchy of classes. Generally there is good evidence that memory for several different types of knowledge (object, procedure, plan) is organised hierarchically.

The domain theory starts with a preliminary model consisting of a metaschema of domain knowledge, adapted from TKS [Johnson et al. 1988], and a set of domain abstractions which are modelled as classes and specialised by addition of further knowledge to describe different views on a single domain. The basic hypothesis that domain knowledge is organised in a class hierarchy and that classes share general features and can be distinguished by a small number of key determining features. The assumption is that most software engineering domains can be ascribed to one of a tractably small set of domain classes.

The preliminary model proposes a schema of seven knowledge types which define abstract domain models:

- state transitions,
- object structure knowledge,
- goal/purpose statements,
- object type,
- conditions on state transition,

- transformation,
- external event-activity triggers.

Abstract domain classes are differentiated by actions leading to state changes of objects with respect to parts of the system structure. System structure is a set-theoretic concept of object membership linked to the transactional purpose of the system. To illustrate the concept, in stock control sets of objects (products) are held by suppliers, in stock and with customers (delivered products). A non-renewable resource management abstraction, of which library loans is an example, can be distinguished from a renewable resource management abstraction (e.g. stock control) by the key transition of return. The return action causes the object to change from an on-loan state to a resource-available state.

System purpose has been identified as an important determinant of abstract models [Maiden 1991], so goal related semantics are defined in terms of states which the system attempts to achieve or maintain. Information systems have different purposes with regard to object sets. Stock control domains attempt to maintain a minimum quantity of items-in-stock, while the library system attempts to maintain stock constancy so the books-on-loan are returned. Activity (i.e. a set of actions, or algorithms) leading to key state transitions is determined by system purpose.

Other knowledge types (triggers, conditions) play a supplementary role in differentiating domain classes. Equivalent state transitions can be distinguished by their triggering events. Each transition is either triggered by the information system or events beyond that system, which have important influences on the information system. For instance, allocation action in the airline booking domain can be differentiated from allocation of stock to orders in a warehouse domain by the triggers: the former allocation is triggered by the information system while the latter is not. Object types, which have been promoted as determinants of analogical reuse by [Lee & Harandi 1991], are differentiated by their role in the domain, for instance stock items and airline seats both act as resources.

The final part of the theory is a process to operationalise the models by matching rules and heuristics. These enable selection of the appropriate domain abstraction for a set of predicates describing a new application domain and avoids the computationally inefficient approach of linear searching multi variate sets of properties, typified by faceted classification.

The theory has been operationalised in an intelligent retriever-matcher process for reuse. It has been successfully evaluated proving the theoretical predictions of the domain classes which should be matched with respect to a set on input facts in predicate format [Maiden 1992]. However, a number of problems remain to be addressed.

(a) The granularity problem: The theory adopts an intuitive view of model granularity for domain abstractions. This has withstood the relatively weak test of evaluation with 20 domain examples. However, it has become apparent that matching may occur at different modelling levels (object model with relationship knowledge, information system model with transformations and procedures). Further research is required to identify and define matching processes at different levels of granularity and between different types of knowledge.

Object-structure matching, corresponds approximately to entity relationship models found in structured methods (e.g. SSADM, MERISE, IE). Matching of dynamic aspects will extend the theory from the domain space towards the design space. This endeavour is tractable because mapping at the design level has been effective in transformation programming [Johnson & Feather 1990].

The domain theory will add an intermediate level of abstraction linking designs to requirements through the concepts of system purpose. The theory will be extended to identify larger and smaller abstractions, using aggregation and specialisation. This will enrich the theory with concepts of an overall domain structure as a framework for object and information system abstractions. For instance aircraft management control is an aggregate of collision monitoring, flight plan adherence, landing, pilot communication and aircraft scheduling.

(b) The coverage problem: The current theory is limited in scope by the number of abstract domain classes which have been described. Further example-based studies of complex software engineering analogies will be used to validate the definition of abstract domain classes. This will be combined with further operational testing by examples cited in the literature to give a sample with sufficient breadth to establish confidence in near complete coverage. Domain abstractions are formalised using Telos to refine the definitions and eliminate any redundancies in definitions. These formal studies are combined with schema development from a cognitive science viewpoint.

Memory schema theories [Schank 1982, Rosch 1991] and their more pragmatic instantiations such as TKS form the basis of remodelling the domain theory metaschema to accommodate recommendations from cognitive theories with set theoretic leaning in the natural category class. This is expected to lead to further elaboration of schemas with reference to salience in human memory. The aim is to describe types and prototypical instance of facts which can be predicted as being important and recognisable by people. The cognitive dimension will be tested by questionnaires, interviews and comprehension tests.

Theoretical development will be informed by empirical investigation of mental abstractions possessed by expert software engineers. Empirical studies elicit these knowledge structures from experts for specific and abstraction domain classes using established knowledge acquisition techniques of repertory grid analysis and multi dimensional scaling (e.g. [Littman 1987, Cooke & McDonald 1987, Garg-Janardan & Salvendy 1987]). This will provide further heuristics for organising the matching process , data about the process of abstraction followed by expert practitioners and valuable input to the problems of granularity in domain knowledge.

Validation of the domain theory will be undertaken in two contexts: specification reuse and intelligent assistance for requirement capture. The analogical matching process is based on a design which employs multiple search heuristics and a partitioned search space. The prototype process will follow theoretical prescriptions to define further rules and search heuristics for matching domain abstractions at different levels of granularity. These predicates will be matched against input terms derived from users. The theory of abstraction will be used to guide the input dialogue with the analyst. Some translation of domain terminology will be necessary via a semantic lexicon.

In requirements capture, domain models will be embedded in prototype tools to enable explanation and active guidance. Initial fact gathering will receive active guidance using diagrams of abstract templates to help conceptual model formation. Visualisation of analogies and knowledge structures is known to improve learning [Gick & Holyoak 1983]; hence we see specification development and fact capture as an iterative process of explanation and analysis in which the set of potential abstract class matches is narrowed as the software engineer's understanding of the domain increases. The theory will be used to determine which types of knowledge (e.g. structural, goal related, constraint) should be acquired in what order with respect to the problem. A further test will involve a view integration prototype.

Similarity of Requirements. A complementary approach to domain theory takes a more taxonomic approach by studying the notion of similarity between domains and extending the idea to software design solutions. The prime motivation behind this effort is that similarity could be exploited in dealing with the problem of retrieving descriptions stored in some repository [Symonds 1988] — one of the main operational problems of systems supporting reuse [Biggerstaff & Richter 1987].

Similarity could facilitate the provision of ranked answer-sets in querying, enhance the ability of a retrieval system for fuzzy queries, and help browsing strategies.

Viewing similarity in its broad sense (i.e the related–ness of two entities considering both their commonalities and their differences), it seems reasonable to expect that it could be also be employed in solving other operational problems of reuse, such as acquiring, modifying and classifying descriptions. For instance, the matching of a partial problem specification with an existing frame of abstractions could indicate strategies of acquiring the remaining knowledge, or even suggest automatic completion of the specification. In a different usage mode, the same concept could be the basis of a classification mechanism providing clusters, according to measures of similarity and dissimilarity. A similar idea about the automatic re-modularization of software systems is presented in [Schwanke 1991].

Similarity research in NATURE starts from work on the retrieval problem in the ITHACA Software Information Base (SIB) [Constantopoulos et al. 1991]. In the SIB, similarity is modelled as typed links which associate descriptions of requirements, designs or code. This work soon revealed that it was necessary to quantify links to express the intensity of association and that well-defined criteria were needed for assessing association.

The underlying assumption of previous claims about the potential of using similarity for reuse is that it could provide some sort of analogical reasoning for component matching. However, only certain kinds of similarity would be relevant to the desired inferences while others could be totally irrelevant or even misleading [Kedar-Cabelli 1988]. NATURE will first determine candidate dimensions (e.g. functional similarity, structural similarity, complexity, similarity on reliability), then evaluate these candidates.

Empirical investigations into the influence of similarity on analogical problem solving [Holyoak & Koh 1987] have indicated that there are kinds of similarities that affect primarily the retrieval of the source analogues (*surface similarities*) and other kinds that affect the actual transfer of knowledge between the source and the target analogues (*structure similarities*). Concepts of similarity therefore fit into the lexical dimension of the domain theory. It is important to establish the set of semantic primitives and their range of values which may be candidates for type definition in the domain theory schema and their role in describing source analogues and matching to targets.

In general, requirements descriptions are expected to have features which could be classified into relevant (common or distinguishing) and irrelevant ones with respect to the estimation of their similarity [Davies 1988, Subramanian & Genesereth 1987]. This direction of research will produce sets of features, characterizing certain forms of similarity, based on investigations of the concepts and their causal relations within the area of requirements engineering. These sets will provide the a-priori knowledge that could speed up the similarity-computation process by suggesting heuristics to limit the search space, and quantification of relationships in models so that fuzzy and Bayesian processes could be applied to matching. On the other hand, it will be necessary to provide a flexible computation scheme that could operate even in the absence of some of these features by default reasoning.

The next steps will formally determine the similarity concept in the context of requirements engineering, giving emphasis to the identification of properties (e.g reflexivity, symmetry) that would reflect its actual semantics, the metrics that would be appropriate for quantifying it and the algorithms for estimating these metrics. A study of the interference between similarity and other non-attribute-level general constructs of knowledge representation (e.g InstanceOf, Isa) will enable the construction of hierarchies whose semantics incorporate some notion of relevance (e.g all the instances of a class are similar in sharing its common structure [Wegner 1987]).

5 Process Theory

Traditionally, the software development process has been viewed differently by the project management and development method communities. In the context of project management, the development process is considered through the planning of activities and the allocation of resources (personnel, material, etc ...). In software development methodologies, the development process tends to be more closely related to the system it creates. Transformation activities are gathered into phases according to different levels of abstraction of the product.

A clean separation between the product and the development process [Olle et al. 1988] is also visible. Considerable work has been done on the former and much less on the latter. This remark applies also to the requirements engineering phase. However, the research effort devoted to the development process is growing and results are beginning to appear. According to Dowson [1987], process models can be classified in three categories:

- activity-oriented models;
- product-oriented models;
- decision-oriented models.

Activity-oriented process models come from an analogy with problem-solving, i.e. finding and executing a plan of actions leading to the solution. They are sequential in nature and provide a frame for manual management of projects developed in a linear fashion. Such linear view of design process is inadequate for methodologies which support backtracking, reuse of previous design, and engineering activities parallelism. The first widely used model, the Waterfall model [Royce 1970], falls into this category, along with the spiral model [Boehm 1988] and the fountain model [Henderson-Sellers & Edwards 1990] which try to eliminate the well recognised lack of flexibility of the Waterfall model.

Product-oriented process models [Finkelstein et al. 1990; Akman et al. 1990] represent the development process through the evolution of the product. They are more synergetic with systemic methodologies that do not place constraints on the design process. Furthermore, they allow design tracing in terms of the performed transformations and their resulting products.

The most recent class of process models have progressed to a *decision-oriented paradigm*. This gives the ability to more deeply integrate the semantics attached to the evolutionary aspects of the design. The notion of decision in the design process allows a better understanding of the designer's intention facilitating, in turn, better reuse of its results. The process models of the DAIDA project [Jarke et al. 1990; Rose et al. 1991] and of [Potts 1989] fall into this category.

All of these process models still suffer from two major limitations:

- a granularity problem;
- semantic expression power.

Most existing models look at the process at the macroscopic level. They are not able to precisely describe what actually happens in the development process at a low level of detail. In other words, they lack a sufficiently low level of granularity. This has resulted in CASE technology which is efficient in storing and representing the products but only supports the process with approximate milestones. Currently available process models only represent the development process as a sequence of phases without the rationale justifying the performance of these activities. The semantic expressiveness of these models appears to be insufficient for providing the requirements engineer with any prescriptive guidance.

Semantically powerful process models are one of the key aspects of the next generation of CASE tools. They require process planning and control and a knowledge base to support these activities. Prototypes like SECSI [Bouzeghoub & Métais 1991] or OICSI [Cauvet et al. 1988] develop an expert system design approach in which the knowledge base is composed of heuristic and experimental knowledge on process activities combined with more formal modeling knowledge.

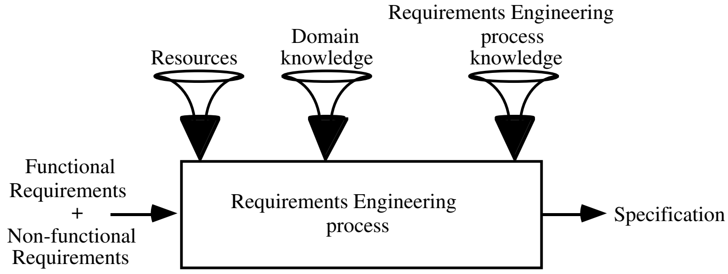

Figure 4 : Requirements engineering process

The aim of the requirements process theory is to allow processes to be modelled:

- in the large as well as in the small, i.e. at several levels of granularity;
- with an emphasis on the semantics of their activities, representing the HOW (what the process does), the WHEN (conditions to perform activities) and the WHY (purposes and goals of activities);
- by means of a collection of basic building blocks.

In other words, the process theory looks at the RE process as a system and will provide means for describing on one hand, its static, structural part - i.e the components of the process and their intra and inter relationships - and, on the other hand, its dynamic part - i.e the behaviour of components.

The starting point is the notion of "triplet" of the OICSI project [Grosz & Rolland 1990]. It is assumed in this approach that the basic building block of any process can be modelled as a triplet: <*situation, decision, action*>. It associates the situation the designer has to deal with to one of the decisions he can take to solve the local problem and to one of the actions to be performed to apply the decision. Furthermore, the record of the selected triplets allows the tracing of the development.

The triplet formalism is also used to represent development process "chunks" in which the designer uses his domain knowledge to improve the conceptual schema [Grosz & Rolland 1991]. The domain knowledge is composed of frequently used generic patterns that the designer can tailor to his needs. The situation part of the triplet is such a pattern, the decision part reflects the tailorization and the action part is composed of the actual transformations of the conceptual schema to be performed.

Process modeling in the large and in the small. The process theory will model software development activities at macro and micro levels of detail. As a first sketch of definition of the macro and micro levels, we can recognize that activities involved in the requirements engineering process fall into two categories :

- engineering activities are those which deal with the RE artifacts (the specifications) and allow their creation, evolution, modification and deletion.
- monitoring activities are required to organise and order engineering activities, select the appropriate engineering activity for a certain situation in the process and allocate the resources needed to perform the engineering activity.

This classification yields a model representing the RE process at two levels of abstraction :

- at the first level, the RE process is an organised collection of monitoring activities;
- at the second level, the RE process is a collection of organised engineering activities.

The macro level of monitoring activities control the way the engineering activities are performed and allocate available resources (personnel, material, ...) to their performance. Macro activities can be repeatedly decomposed into more detailed macro activities. Similarly micro-activities can be viewed at different levels of detail.

Situational and contextual analysis of activities. The process theory will explore a new paradigm by recognising situational links between activities and their organizational contexts. Theoretical development is necessary to understand how activities are related to contexts and how to characterize contexts. The study will concentrate in particular on the characterization of contexts which relate to view integration, those which deal with the requirements expressive freedoms and activities for redundancy and ambiguity elimination and those which trigger structuration and transformational activities for schema refinement.

Our emphasis on the contextual situation analysis is a departure from the traditional linear process view. At the macro level the organization and planning of activities is based on the analysis of the domain and organizational context of the system development. At the micro level, the decision to perform a transformational activity will be based on the analysis of the current situation of the software specification.

The next step will be the definition of a taxonomy of decisions represented as a hierarchy of decision classes. It will be studied how the micro and macro levels fit into the hierarchy to help in the triplet triggering control previously mentioned. Finally, the nature of activities and their classification ìnto a hierarchy will be studied in connection with the taxonomy of decisions. This will provide a general frame for guiding processes of software engineering. Particular attention will be paid to explain the differences between macro-level triplets representing macro process engineering activities and micro-level triplets dealing with the detailed transformations of specifications.

Formalizing the links between domain knowledge and process activities. In order to limit the breadth of the context analysis, the problem will be rationally restricted to the study of contextual links that relate domain properties to activities for refining conceptual schemata, i.e. to contexts that trigger micro-activities which aim at producing and transforming system specifications.

Under this perspective we investigate the concept of contextual influence on process activities. This will focus on linking with the domain theory to model how types of domain knowledge affect the process and product of specification activity. The first step of the study will consider the domain knowledge types defined in the domain theory. Appropriate activity templates will be defined for each domain type. The link between the domain theory and the process theory will be established through the notion of triplet. The situation part corresponds to the domain type and the template will be modelled in terms of decisions and transformational activities.

A library of templates will serve as a knowledge base for providing automated assistance in the software engineering process. Process is driven by the contextual analysis. Domain knowledge types are used to recognize the situation the designer has to deal with. Templates provide him with a predefined organization of modeling activities. The decisions he makes allow the adaptation of the template to the particular application. In the second step of the study, the retrieval and matching process by which appropriate templates can be put in correspondence with a specific situation will be investigated.

6 Implementation and Validation

The project follows the research method sketched in figure 4. From observation of current practice and research, theories are hypothesized which are falsifiable through predictive power. To test the theories, suitable models, methods, and formal representations are derived from them. Selected important aspects of these will be supported by prototypical tools. The final goal of the theory is to improve the practice of requirements engineering. From the three major applications, case studies will be made in coooperation with application-oriented projects including F-CUBE (requirements capture), ITHACA (reuse of object-oriented software), and WibQus (quality management).

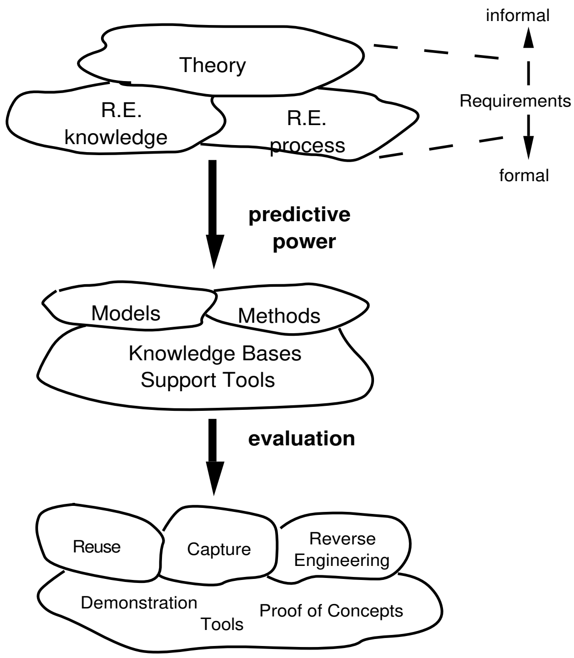

Figure 5: Overview of research method

An important part of testing and evaluating our knowledge representation, domain and process theories involves the development of tool prototypes based on these theories. In other words, we want to define tools which make use of the domain and process theory, use the freedoms and the links between non-formal and formal knowledge described above. They demonstrate how to use formal and non-formal knowledge, how and when to match acquired knowledge against the domain and process theory, and how to reuse existing specifications. The tools also support the transition between non-formal and formal descriptions, offer descriptions for reuse during the acquisition process and make use of formal and non-formal knowledge to support the validation of requirements.

We will use existing software tools such as SISU´s RAMATIC shell as a framework in which to develop prototypes to demonstrate our ideas. RAMATIC is an extensible CASE tool development environment which allows new tools to be configure on a repository base. Prototypes will be developed for view integration, requirements validation and reuse matching. More than one theory will be tested in a single prototype, for instance the matching prototype will validate both the domain theory and similarity approaches.

The *verification and validation* tool consists of a conceptual schema analysis and diagnosis subsystem and a subsystem which generates natural language descriptions from formal descriptions according to the chosen representational framework. It is obvious that each one of these systems may make use of domain knowledge and/or help in acquisition of such knowledge. Correspondingly, results from the process engineering theory will be integrated into the prototypes in order to control the applicability of each subtool and to guide the users.

The *reuse advisor* will match domain abstractions and new domain descriptions input by the user. Prototype implementation will involve creation of knowledge bases for holding specification descriptions, development of more efficient versions of the matching process and a means of adding knowledge of new domains. User-system interfaces will be developed for fact acquisition and decision support to guide analysts in selecting suitable reusable components. The system will have a limited ability to classify new terms according to the meta schema and hence add new domain specific terms.

The *view integration* tool will assist the users in integrating locally (and independently) developed subschemata, using part of the domain knowledge base, into a global, integrated schema. The work must extend existing approaches, primarily in the sense that the integration subsystem must provide more intelligent service to the user of the tool. This can be achieved by augmenting the syntactically based integration algorithms by support from the domain knowledge that will help the tool to better "understand" the concepts and their relationships used in the local schemata. This understanding is a prerequisite for meaningful integration as well as for the restructuring of the schemata.

7 Summary and Conclusions

Reiterating, the goal of the NATURE project is to provide a set of interacting theories that relate knowledge representation, domain analysis, and process support aspects of requirements engineering. The ontological foundations of the approach are given on the structural side by organizing requirements knowledge according to four related "worlds" each coming with an appropriate set of domain abstractions. On the process side, a situation-based and decision-oriented ontology is advocated which allows for a natural integration of both the domain-oriented context and the role of teamwork and non-functional requirements.

In the representational dimension, the joint usage of formal, semi-formal, and informal representations is being studied, supported by a technical environment that integrates knowledge representation, graphical views on these, and hypermedia technology.

Contributions to the evaluation of requirements engineering theories are made through the further development of prototypical tools and knowledge management platforms, and through the extension of standard literature examples as to capture the hard problems of requirements engineering.

References

[Akman et al. 1990] Akman V., ten Hagen P.J.W., Tomiyama T.: A fundamental and theoretical framework for an intelligent CAD System; Computer Aided Design 22, 6.

[Balzer 1991] Balzer R.: Tolerating inconsistency. Proc. 13th International Conference on Software Engineering, Austin Texas, 158-165

[Biggerstaff & Richter 1987] Biggerstaff T., Richter C., Reusability framework, assessment and directions, IEEE Software, March 1987

[Boehm 1988] Boehm B.W.: A spiral model of software development and enhancement; IEEE Computer 21.

[Borgida 1991] Borgida A.: A position paper on intelligent information systems. In Balzer R., Mylopoulos J. (eds.): Intl. Workshop on Development of Intelligent Information Systems, Niagara-on-the-Lake, 10-13

[Borgida et al. 1985] Borgida A., Greenspan S.J. and Mylopoulos J.: Knowledge re–presentation as the basis for requirements specification. IEEE Computer, April 1985, 82-91

[Borgida et al. 1989] Borgida A., Brachman R.J., McGuinness D.L., and Resnick L.A.: CLASSIC: A structural data model for objects. Proc. ACM-SIGMOD Conf., Portland, Oregon, 58-67

[Bouzeghoub & Métais 1991] Bouzeghoub M. and Métais E.: Semantic modelling of object oriented databases. Proc. 17th Intl. Conf. Very Large Data Bases, Barcelona, Spain

[Cauvet et al. 1988] Cauvet C., Rolland C., Proix C. : Information systems design: an expert system approach, Proc. Intl. Conf. Extending Database Technology, Venice, March 1988.

[Chung et al. 1991] Chung, L., Katalagarianos, P., Mertikas, M., Mylopoulos, J., Vassiliou, Y.: From information systems requirements to design: a mapping framework. Information Systems 16, 4.

[Conklin & Begeman 1988] Conklin J. and Begeman M.L: A hypertext tool for exploratory policy discussion. ACM Trans. Office Information Systems 6, 4, 140-151

[Constantopuoulos et al. 1991] Constantopoulos P., Jarke M., Mylopoulos J., Vassiliou Y.: Software Information Base: A server for reuse. Report, ESPRIT project ITHACA, ICS-FORTH, Heraclion, Greece.

[Cooke & McDonald 1987] Cooke N.M., McDonald J.E.: The application of psychological scaling techniques to knowledge elicitation for knowledge-based systems, Intl. J. Man-Machine Studies 26, 553-550

[Davies 1988] Davies T.: Determination, uniformity and relevance: normative criteria for generalization and reasoning by analogy. Analogical Reasoning, Kluwer Academic Publishers, 1988

[Dowson 1987] Dowson M.: Iteration in the software process. Proc. 9th Intl Conf Software Engineering, Monterey, Ca.

[Eherer & Jarke 1991] Eherer S., Jarke M.: Knowledge base support for hypermedia co-authoring, Proc. Database and Expert Systems Applications, Berlin, 465-470

[Feather 1987] Feather M.: Language support for the specification and development of composite systems. *ACM Trans. Programming Languages and Systems 9,* 2, 198-234

[Feather & Fickas 1991] Feather M.S. and Fickas S.: Coping with requirements freedom. In: Proc. Intl. Workshop Development of Intelligent Information Systems, Niagara-on-the-Lake, Canada, 42-46

[Fickas 1987] Fickas S.: Automating analysis: An example. Proc. 4th Intl.Workshop on Software Specification and Design, Washington DC, 58-67

[Finkelstein et al. 1990] Finkelstein A., Kramer J., Goe–dicke M.: Viewpoint-oriented software development; Proc. Conf Le Génie Logiciel et ses Applications, Toulouse, 337-351

[Freeman 1987] Freeman P.: Software reusability. In IEEE Tutorial on Software Reuse, IEEE Press

[Garg-Janardan & Salvendy 1987] Garg-Janardan C., Salvendy G., A Caonceptual Framework for Knowledge Elicitation, Intl. J. Man-Machine Studies 26, 521-531

[Gentner 1983] Gentner D.: Structure mapping: a theoretical framework for analogy, Cognitive Science 5, 121-152

[Gick & Holyoak 1983] Gick M.L. and Holyoak K.J. Schema induction and analogical transfer, Cognitive Psychology 15, 1-38

[Greenspan 1984] Greenspan S.J.: Requirements modeling: A knowledge representation approach to software requirements definition. Univ. of. Toronto, Toronto, Tech. Report. CSRG 155.

[Greiner R. 1988] Greiner R.: Abstraction-based Analogical Inference, in Analogical Reasoning, edited by D.H. Helman, Kluwer Academic Publishers,p. 147-170

[Grosz & Rolland 1990]: G. Grosz, C. Rolland; Using artificial intelligence techniques to formalize the information system design process; Proc. Intl. Conf. Databases and Expert Systems Applications, 374-380.

[Grosz & Rolland 1991] G. Grosz, C. Rolland; Why and how should we hide conceptual models; Proc. 3rd Intl. Conf. Software Engineering and Knowledge Engineering (SEKE), Skokie, USA, 28-33

[Guindon 1990] Guindon R.: Designing the design process: exploiting opportunistic thoughts, Human-Computer Interaction Journal 5, 305-344

[Guindon & Curtis 1988] Guindon R. and Curtis B.: Control of cognitive processes during software design: what tools are needed? Proc. ACM-CHI'88, 263-269

[Hagelstein 1988] Hagelstein J.: Declarative approach to information systems requirements. Knowledge Based Systems, 1, 4, 211-220

[Hahn et al. 1991] U. Hahn, M. Jarke, T. Rose: Teamwork support in a knowledge-based information systems environment. IEEE Trans. Software Eng., 17,5

[Henderson-Sellers & Edwards 1990] B. Henderson-Sellers, J. M. Edwards, The object-oriented systems life cycle, Comm. ACM, Sept. 1990

[Holyoak & Koh 1987] Holyoak K., Koh K., Surface and structural similarity in analogical transfer. Memory and Cognition 15, 4

[Jarke 1988] Jarke M.: The design of a database for multiperson decision support. Annals of Operations Research 16, 393-412

[Jarke 1990] Jarke, M.: DAIDA -- conceptual modeling and knowledge-based support for in–formation systems development processes. Technique et Science Informatique 9, 2, 121-133

[Jarke ed. 1992] Jarke M. (ed.): ConceptBase V3.1 User Manual, Aachener Informatik Berichte 92-17, RWTH Aachen, Germany

[Jarke et al. 1990] Jarke M., Jeusfeld M., Rose T.: A software process data model for knowledge engineering in information systems. Information Systems 15, 1, 85-116

[Jarke & Pohl 1992] Jarke M., Pohl, K.: Information systems quality and quality information systems. Proc. IFIP 8.2 Working Conf., Minneapolis, Minn.

[Jarke et al. 1992] Jarke M., Mylopoulos J., Schmidt J., Vassiliou Y.: DAIDA — an environ–ment for evolving information systems. ACM Trans. Information Systems 10, 1, 1-50.

[Johnson & Feather 1990] Johnson W.L. and Feather M.: Building an evolution transformation library: Proc. 12th Intl. Conf. Software Engineering, Nice, France, 238-248

[Johnson & Harris 1990] Johnson W.L. and Harris D.: The ARIES Project, Proc. 5th KBSA Conf., Liverpool, N.Y., 121-131.

[Johnson et al. 1988] Johnson P., Johnson H., Waddington R. and Shouls A.: Task-related Knowledge Structures: analysis, modelling and application, Proc. HCI '88, Cambridge University Press, 35-61

[Jones 1986] Jones, C.: Systematic Software Development Using VDM. Englewood Cliffs, NJ: Prentice-Hall

[Kedar-Cabelli 1988] Kedar-Cabelli S., Analogy - from a unified perspective, Analogical Reasoning, Kluwer Academic Publishers, 1988

[Lee & Harandi 1991] Lee H.Y., Harandi M.T.: Some interactive sspects of a software design schema acquisition tool, USC-ISI Tech. Report RS-91-287

[Littman 1987] Littman D.C., Modelling human expertise in knowldege engineering: some preliminary observations, Intl. J. Man-Machine Studies 26, 81-92

[Loucopoulos et al. 1991] Loucopoulos, P., Papastamatiou G, Pantazis D, Diakonikolaou G.: Design and execution of event/action DB applications. Second Intl. Workshop Deductive Approach to Information Systems and Databases, Aiguablava, Spain, 1-21

[Maiden 1991] Maiden N.: Analogy as a paradigm for specification reuse, Software Engineering J. 6, 1, 6-15

[Maiden 1992] Maiden N.: Analogical specification Reuse during Requirements Analysis, Ph.D. Thesis, City University of London.

[Mylopoulos et al. 1990] Mylopoulos J., Borgida A., Jarke M. and Koubarakis M.: Telos:Representing Knowledge About Information Systems. ACM Trans. Information Systems 8, 4, 325-362.

[Mylopoulos et al. 1992] Mylopoulos J., Chung L., Nixon B.: Representing and using non-functional requirements: a process-oriented approach. IEEE Trans. Software Eng. 18, 6, 483-497

[Olle et al. 1988] Olle T.W., Hagelstein J., MacDonald I.G., Rolland C., Sol H.G., Van Assche F.J.M., Verryn-Stuart A.A. Information System Design Methodologies: A Framework for Understanding, Addison-Wesley, 1988.

[Pernici 1990] Pernici B.: Objects with Roles (ORM). Proc. ACM Conf. Office Information Systems, Cambridge Massachusetts, 205-215

[Porter 1985] Porter M.E.: Competitive Advantage. New York: Free Press.

[Potts 1989] Potts C.; A generic model for representing design methods; Proc. 11th Intl. Conf. Software Engineering

[Punchello et al. 1988] Punchello P.P., Torrigiani P., Pietri F., Burion R., Cardile B. and Conit M.: ASPIS: a knowledge-based CASE environment, IEEE Software, March, 58-65

[Ramesh & Dhar 1992] Ramesh B. and Dhar V.: Supporting systems development by capturing deliberations duing requirements engineering. IEEE Trans. Software Eng. 18, 6, 498-510

[Reubenstein 1990] Reubenstein H.B.: Automated Acquisiition of Evolving Informal Descriptions, Ph.D. Dissertion, AI Laboratory, Massachusettes Institute of Technology

[Reubenstein & Waters 1991] Reubenstein H.B. and Waters R.C.: The Requirements Apprentice: Automated Assistance for Requirements Acquisition. In: IEEE Transaction on Software Engineering, March 1991, p. 226-240

[Rockart 1979] Rockart J.F.: Critical Success Factors. Harvard Business Review, March-April 1979, 81-91.

[Rosch 1991] Rosch E.: Prototype classification and logical classification: the two systems. In E.K Scholnick (ed.): New Trends in Conceptual Representation: Challenges to Piaget's Theory ? LEA.

[Rose et al. 1991] Rose T., Jarke M., Gocek M., Maltzahn C., Nissen H.: A decision-based configuration process environment. Software Engineering Journal 6, 5.

[Ross 1985] Ross D.T.: Applications and extensions of SADT. IEEE Computer, April 1985, 24-35

[Royce 1970] : Royce W.W.; Managing the development of large software systems; Proc. IEEE WESCON.

[Schank 1982] Schank R.C., Dynamic Memory: A Theory of Reminding and Learning in People and Computers. Cambridge University Press

[Scheer 1991] Scheer, A.-W., Architektur betrieblicher Informationssysteme, Springer-Verlag

[Schwanke 1991] Schwanke R., An intelligent tool for re-engineering software modularity, Siemens 1991

[Smolander et al. 1991] Smolander K., Lyytinen K., Tahvanainen V.-P. and Marttiin P.: MetaEdit - A Flexible Graphical Environment for Methodology Modelling, Proceedings CAiSE 1991, Trondheim, Norway, Springer-Verlag.

[Spivey 1988] Spivey, J.M.,:The Z Notation: A Reference Manual. Prentice-Hall International,

[Subramanian & Genesereth 1987] Subramanian D., Genesereth M., The relevance of irrelevance, Proc. 10th IJCAI, Morgan Kaufmann

[Sutcliffe 1991] Sutcliffe A.G., Object oriented systems analysis: the abstract question. Proc. IFIP WG 8.1 Conf. The Object Oriented Approach in Information Systems, Quebec City, Canada.

[Sutcliffe & Maiden 1990] Sutcliffe A.G., Maiden N.: Software reusability: delivering productivity gains or short cuts, Proc. INTERACT'90, North-Holland, 948-956.

[Sutcliffe & Maiden 1992] Sutcliffe A.G. and Maiden N.: Analysing the analyst: cognitive models of aoftware engineering, to appear in Intl. J. Man-Machine Studies

[Symonds 1988] Symonds A., Creating a software en–gineering knowledge base, IEEE Software, March 1988

[Vitalari & Dickson 1983] Vitalari N. and Dickson G.W.: Problem solving for effective systems analysis: an ex–perimental exploration, Comm. ACM 26, 11, 948-956

[Wegner 1987] Wegner P., The object-oriented classification paradigm, Research Directions in Object-Oriented Programming, Shriver Wegner ed., MIT Press