The Knowledge Engineering Review, Vol. 18:1, 47–81. © 2003, Cambridge University Press
DOI: 10.1017/S026988890300064X Printed in the United Kingdom

Methodologies for the development of knowledge-based systems, 1982–2002*

ROBERT PLANT[1] and ROSE GAMBLE[2]

[1] *Department of Computer Information Systems, University of Miami, Coral Gables, Florida, USA*
[2] *Department of Mathematical and Computer Sciences, University of Tulsa, Tulsa, Oklahoma, USA*

Abstract

Knowledge-based systems have often been criticised for the limited theoretical base upon which they are constructed. This view asserts that systems are often developed in an ad hoc, individual way that leads to unmaintainable, unreliable and non-rigorous systems. The last decade, however, has seen an increased effort to produce methodologies to counter this view as well as continued research into validation and verification techniques. This paper presents a brief discussion of some of the important research in knowledge-based system life cycles and development methods. Methodologies are considered and are discussed in light of two sets of quality assurance criteria.

1 Introduction

The creation of any information system is an involved and complex process, leading to the development of design methodologies. These methodologies, models or system life cycles attempt to define a series of steps or processes through which a developer will be assured of producing a high-quality software system. Such a system would be reliable, efficient AND maintainable and match users' requirements.

The creation of traditional, procedural, algorithmic software has led to three major schools of development: (1) those who use a methodological approach that follow stage-based process models such as Boehm's spiral model (Boehm, 1988; Boehm & Hansen, 2001) or Royce's waterfall model (Royce, 1970), (2) those who advocate prototyping (Ribeiro Justo *et al.*, 1997), and (3) those who advocate the utilisation of formal methods of specification (Jones, 1980). More recent object-oriented methodologies rely on processes such as the unified process (Jacobson *et al.*, 1999). However, the major emphasis of these systems is the development of traditional software systems, and as such the area of pattern-directed inference systems, better known as knowledge-based systems (KBSs), does not automatically fall into this category. Consequently, KBSs do not benefit from the developmental experience behind these methodologies. KBSs have been used since the early 1980s and, as yet, the methodologies associated with developing these systems remain opaque. The aim of this paper is first to state the qualities needed in a successful methodology, followed by a discussion of the three major classes of existing methodology for KBS development: the stage-based approach, the 'industrial-strength' models and formal models. The paper concludes with an assessment of each methodology based upon two quality assessment frameworks and provides insights into the next generation of methodologies for KBS.

* The second author is supported in part by the AFOSR (F49620–98–1–0217) and the NSF (CCR-9988320).

2 Quality assessment frameworks

As with the development of any software system there are two key aspects to the quality assurance process: system validation (are we building the right product?) and system verification (are we building the product right?). Research in the area of validation and verification as applied to KBSs has progressed in parallel to that of KBS development methodologies, building up an extensive literature (Antoniou & Plant, 1997; Ayel & Rousset, 1995; Coenen *et al.*, 2000; Gamble & Landauer, 1995; O'Leary & Preece, 1998; Plant, 1994; Preece, 1993; Schmolze, 1996). The ability to ensure that the quality of the development matches the requirements of the task is very important, however not all tasks require the same level of quality. For example, systems utilised in areas where there is a risk to human life clearly require more rigorous validation and verification than those systems that are used as a 'proof of concept' demonstrator system from which a more formal system will later be developed. Thus quality, verification and validation are intrinsically related to the methodology utilised in systems construction. The literature and tools (Murrell & Plant, 1997) related to validation and verification primarily relate to specific individual techniques for pre- or post-system development scrutiny or alternatively for specific system aspects such as the verification of the rule base. This inhibits developers from achieving a broader quality assurance assessment. After an influential NASA report considered the application of software quality measures to AI software (Rushby, 1988), the road to quality assurance then took several directions. These included standards from organisations such as IEEE and AIAA, research in validation and verification through forums such as AAAI, and research in software quality assurance where, for example, a set of eight quality metrics were proposed by Carpenter (Carpenter & Murine, 1984) and applied to KBSs by Plant (1991).

A central issue in the discussion of quality is, what are the parameters that define quality? Preece (1995) cites five popular quality criteria and briefly discusses the issue. The criteria he considers are: understandability, modularity, verifiability, efficiency and complexity (Antoniou & Sperschneider, 1994; Ghezzi *et al.*, 1991). Another aspect of quality is the concept of risk in the development of expert systems. Cheng and Jamieson (1996) also consider the quality issue from a software engineering perspective. Cardenosa and Pastor (Cardenosa, 1994) go a stage further and propose a four-phase quality assurance plan for both system development and maintenance.

A more abstract quality framework has been proposed by Gaines (1996), which examines the separation of physical, mental and knowledge worlds. Gaines's proposition is that "each world has different criteria for truth and valid inference, and each system's interaction with each world may be evaluated using different criteria" (Gaines, 1996). This builds upon Gaines's earlier work that considers the management of quality assurance in integrated knowledge acquisition and performance systems (Gaines *et al.*, 1992).

The increased adoption of KBSs in areas that have stringent critical quality requirements has led to specialised literature and a proliferation of general software quality and development standards from multiple organisations.

- NATO: AQAP-150, Quality Assurance Requirements for Software Development, ANSI/IEEE:IEEE Std 730–1998,
- Standard for Software Quality Assurance Plans, IEEE Std 730.1–1995,
- Guide for Software Quality Assurance Planning, IEEE Std 828–1998,
- Standard for Software Configuration Management Plans, IEEE Std 1008–1987,
- Standard for Software Unit Testing (ANSI), IEEE Std 1012–1998,
- Standard for Software Verification and Validation, IEEE Std 1012a-1998,
- Supplement to Standard for Software Verification and Validation: Content Map to IEEE/EIA 12207.1–1997, IEEE Std 1028–1997;
- US DoD 8120.2 Automated Information System (AIS) Life-Cycle Management (LCM) Process, Review, and Milestone Approval Procedures,
- DoD 8120.2-M Automated Information System Life-Cycle Management Manual and
- NRC: ERPI NP-5236/1987, ERPI NP-5978/1987, NSAC-39/1981; NUREG/CR-4640/1987, NUREG-0653/1980 (Plant.1991).

Many of the early standards were not directly suitable to the KBS development life cycles as they were based upon stage-based methods of quality assurance aimed at traditional procedural software development. Many focused on deriving adequate documentation, in contrast to addressing the issues of completeness and correctness in the knowledge acquisition or knowledge representation stages. They also lacked the formal proof of correctness stages necessary for the rigorous development of complex heterogeneous code development. Further, as discussed by Livson, "Neither ANSI/IEEE Std 730–1984 nor MIL-S-52779A explicitly state that software quality assurance shall conduct verification, validation and testing" (Livson, 1988). This led to many organisations with the need for high degrees of reliability and quality to move towards developing their own standards and approaches. A lead organisation in this effort has been the Software Technology Branch of NASA. Their work includes techniques such as "evidence flow graph methods" (Becker *et al.*, 1989), "the use of meta-knowledge in the verification of KBS" (Morell, 1989), and real time KBS systems development (Culbert, 1990; Johnson, 1988). Several other organisations have produced significant research in this field, including the American Institute for Aeronautics and Astronautics (ANSI, 1992), the Federal Aviation Administration (Ibrahim *et al.*, 1997) and the Electrical Power Research Institute (EPRI), whose methodology is used in the creation of KBSs within the US nuclear power industry (Miller, 1990). All of these organisations adopted the MIL-STD-2167A standard as the foundation of their approaches.[1]

Miller, a lead researcher in the creation of the EPRI methodology, based its KBS development principles around seven major requirements (criteria 1 to 7 in Table 1) (Miller, 1990). These criteria are useful as macro-level indicators of a methodology's general applicability to the development of stable KBSs that can be mapped onto MIL-STD-2167A or successor standard. The KBSs developed by these criteria can be considered verifiable and adaptable when an analysis of criteria 1 to 9 yields positive values. The quality of the systems developed is enhanced when the methodology extends the

Table 1 Macro-level quality criteria for KBS methodologies

	Criteria	Possible values
1	Easily handles ill-formed or changing requirements	No, Yes
2	Suitable for stable system development	No, Yes
3	Maps well onto DoD-STD-2167(A) for consistent management	No, Yes
4	Appropriate for embedded, real-time, data-driven systems as well as stand-alone knowledge-based systems	No, Yes
5	Provides for plan-reviews and completion audits of requirements, design and implemented systems, with configuration management throughout	No, Yes
6	Supports competent validation and verification	No, Yes
7	Extends to maintenance activities	No, Yes
8	Mathematical techniques are used wherever possible or appropriate	No, Yes
9	Promotes implementation independent specification of system features/requirements	No, Yes
10	The documentation is adequate/rigorous	No, Adequate, Rigorous
11	Either a refinement process or a stage-based approach is adopted	Stage, Transformation model
12	Each step in the development can be traced back to the previous step and justified	No-Informal, Yes-Rigorous, Integrated

[1] MIL-STD-498 was superseded by MIL-STD-498, which was then superseded by ISO/IEC 12207.

Table 2 TRILLIUM$_K$ level 1 capability

TRILLIUM$_K$ level 1 capability	
Problem specification	No explicit statement of requirements. No test plan. No acceptance criteria
Conceptual model	No documented conceptual model
Design model	No documented design model for knowledge base. Typically a commercial shell is used; the reasons for choosing a shell should be documented
Implemented model	Implemented knowledge base is the only complete description of knowledge
Verification analyses	Verification performed by informal proofreading – no formal verification analysis conducted
Validation analyses	Validation performed by ad hoc testing and informal evaluations. No permanent recording of test suite

verification and validation through to the maintenance activities, developing rigorous documentation and utilising a refinement process (criteria 1 to 12). Taken together these twelve criteria allow KBS project managers and developers to quickly assess a methodology as suitable or unsuitable to their needs.

The use of these criteria in isolation, in order to assess the potential overall quality of a system development methodology, is extremely difficult without a framework that relates formal levels of quality achievement with specific criteria and metrics. One such framework, based upon the SEI process maturity model (Humphrey, 1989), is known as the TRILLIUM Model (TRILLIUM, 1992). This model is adapted for use with KBS development in the form of the TRILLIUM$_K$ Model. TRILLIUM$_K$ has three capability levels, ranging from Level 1, in which an informal development style is practised, to Level 3, where rigorous procedures are followed. Tables 2, 3 and 4 show the increasing formality of the criteria with respect to KBS development.

Each of the levels is broken down into six aspects of development: problem specification, conceptual model, design model, implemented system, verification analysis and validation analysis. The intent is three-fold. First, it allows developers a starting point from which to consider their own methodology and its requirements. Second, it provides a model by which the rigour of a systems progression might be considered, from the weakest at Level 1 to the most rigorous at Level 3. Lastly, the model provides

Table 3 TRILLIUM$_K$ level 2 capability

TRILLIUM$_K$ level 2 capability	
Problem specification	Informal statement of requirements, test plan and acceptance criteria
Conceptual model	“Paper model” stated semi formally. Separation of concerns achieved by isolating domain, task and cooperative knowledge components
Design model	Architectural design for system components. Semiformal or formal designs for procedural parts of knowledge base, and for inference engine
Implemented system	Implemented knowledge base and inference engine is traceable, where appropriate, to conceptual and design models. When a third-party shell is used, the behaviour conforms to that which is required
Verification analyses	Knowledge-base integrity and expression logic is checked automatically, with all detected anomalies fully documented and resolved
Validation analyses	Testing using documented test suite is performed according to problem specification. Semiformal evaluations of system usability are performed and documented

Table 4 TRILLIUM$_K$ level 3 capability

	TRILLIUM$_K$ level 3 capability
Problem specification	Semiformal statement of requirements, including minimum and desired functionality. Formal constraints should be associated with all possible minimum requirements. Test plan and acceptance criteria are associated with each functional requirement that cannot be verified formally
Conceptual model	Formal knowledge-level model (that is, with well-defined syntax and semantics). Appropriate representation languages used for domain, task and cooperative knowledge-base components
Design model	Formal architectural design, module interface specifications and internal module designs for all system components, including interface engine, domain knowledge modules, task knowledge modules, meta-level control knowledge modules and external interface components
Implemented system	Implemented system is fully traceable to conceptual and design models or is derived automatically from them using a correctness-preserving transformation procedure. A third-party tool may be used only if rigorous assurances are available of its correctness and reliability
Verification analyses	Full inference logic is checked and all anomalies documented and resolved. System compliance with all minimum constraints is verified and documented if possible
Validation analyses	Rigorous structural and functional testing is performed according to problem specification. Test suite is executed and maintained using support tools. Approved empirical methods are used for usability and utility evaluations, and results are fully documented

developers with a framework through which the standards of a customised or adapted methodology can be judged.

An important function associated with these capability models is the ability to justify or "trace" the development of a system from one stage in the to another. This may be done informally through the explosion or folding of diagrammatic representations or formally through the wrapping of mathematical descriptions around proofs of refinement. Without this functionality in a life cycle, a developer cannot adequately determine the correctness and validity of a system as it moves from stage to stage, from the conceptual to the concrete form. In the next section of the paper we will utilise both the macro-level quality criteria and the TRILLIUM$_K$ capability model to assess the characteristics of several KBS life cycles found in the literature and to determine the degree of development formality present.

Early KBSs, such as MYCIN (Shortliffe, 1976) and Dendral (Lindsay *et al.*, 1980), originated as research systems. As such, they were developed in an ongoing cycle of experiment-assess-implement, which led to systems that were highly complex, full of interesting technical features and ideas, yet had no methodology behind them. Therefore these systems fall into level 1 of the TRILLIUM$_K$ capability model. These early developers paved the way for current KBS development methodologies. It is understandable that these systems would fall short of quality standards. Because it is important for modern developers not to reinvent poor-quality KBSs, we discuss in detail the early KBSs in Appendix A.

3 "Industrial-strength" life cycle models

In this section, we discuss a category of models termed "industrial strength" methodologies that directly target pragmatic corporate usage at organisations such as NASA, AIAA, the US Department of Defense and the Nuclear Regulatory Committee.

3.1 Weitzel and Kerschberg's methodology (KBSDLC)

The proliferation of KBS development environments and prototyping led Weitzel and Kerschberg (1989) to create a methodology that addresses these issues. Their methodology adds rigour to the prototyping approach by introducing a series of "processes" through which the development is driven, in contrast to the stage-based models. Their life cycle comprises the eleven phases shown in Figure 1.

In this methodology, KBS prototype development proceeds through each of these processes. However, once a process has been "activated", the knowledge engineer either "deactivates" the process (proceeding to the next one) or returns to a previous process and "reactivates" it for further processing. In this way, the system evolves as a series of activated processes.

This developmental style is advantageous for prototyping, as the systems produced will be higher quality in relation to those factors described earlier. However, the methodology cannot overcome all the inherent problems associated with prototyping. For instance, the approach advocates the creation of small prototypes and their later integration, which for a sizeable or complex domain can be a difficult task when differing representations are used or hybrid representations are created. Furthermore, the approach is low on formality and does not exhibit many of the factors Miller expects

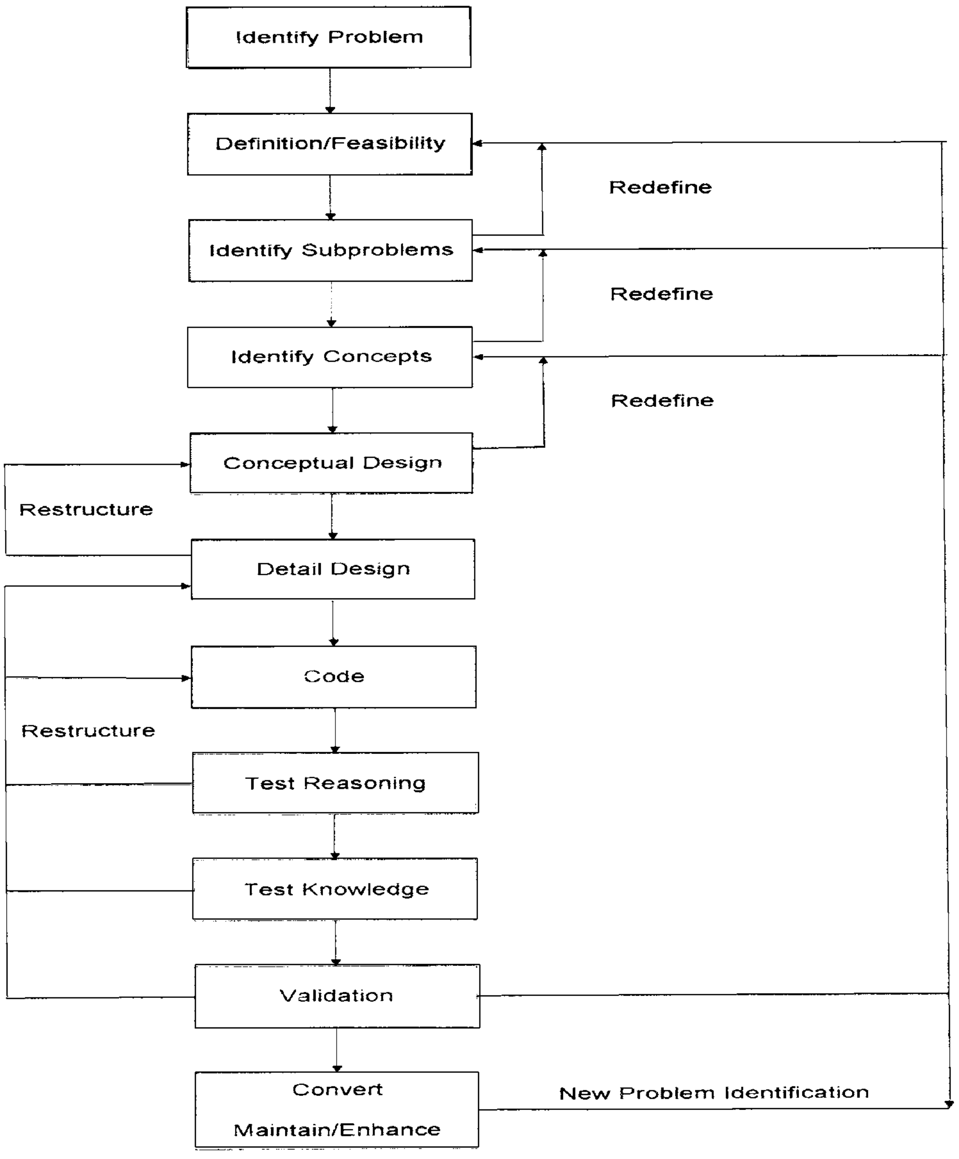


Figure 1 Weitzel AND Kerschbergs' methodology

from a real-world life cycle model, placing it at level 2 of the TRILLIUM$_K$ capability model for all but the verification process, which remains at Level 1. However, many of these deficiencies and the lack of detailed directions for the knowledge engineer can be easily rectified, leading to a powerful pragmatic prototyping methodology.

3.2 Miller's model

KBSs are accepted in industry, embedded in complex systems and no longer confined to the laboratory or specialist researcher. This is manifested by the emergence of methodologies that adhere to standards laid down by respected organisations such as IEEE, ANSI, AIAA and the Department of Defense. Miller proposes "a realistic industrial strength life cycle model for KBS development and testing" (Miller, 1990). This model aims to fulfil all of the requirements we quoted earlier in the paper.

Core to Miller's approach is its ability to fulfil and satisfy the original DoD-STD-2167.[2] The standard 2167A is intended to promote structured methods within the development of traditional systems through the use of detailed requirements specifications. Miller's method requires that all aspects of development be capable of being traced back to that specification. Miller comments "the standard 2167A life cycle thus seems ill suited for KBS development" (Miller, 1990). However, its philosophy is flawless and so Miller's adaptation was to introduce the three-part model shown in Figure 2.

A set of initial requirements are used to create an initial prototype in a rapid manner, even if this involves "unabashed hacking". The goal is to produce a stable set of requirements as early and as quickly as possible, such that all stakeholders can agree on exactly what it is the system is to do. Then a requirements specification is written to act as a baseline document. The resulting prototype undergoes a series of "incremental knowledge builds" in which the system is refined in accordance with any variances that have been found or are required. This deviates from 2167A in that the evolutionary approach allows all aspects to be updated or altered including the requirements document if necessary.

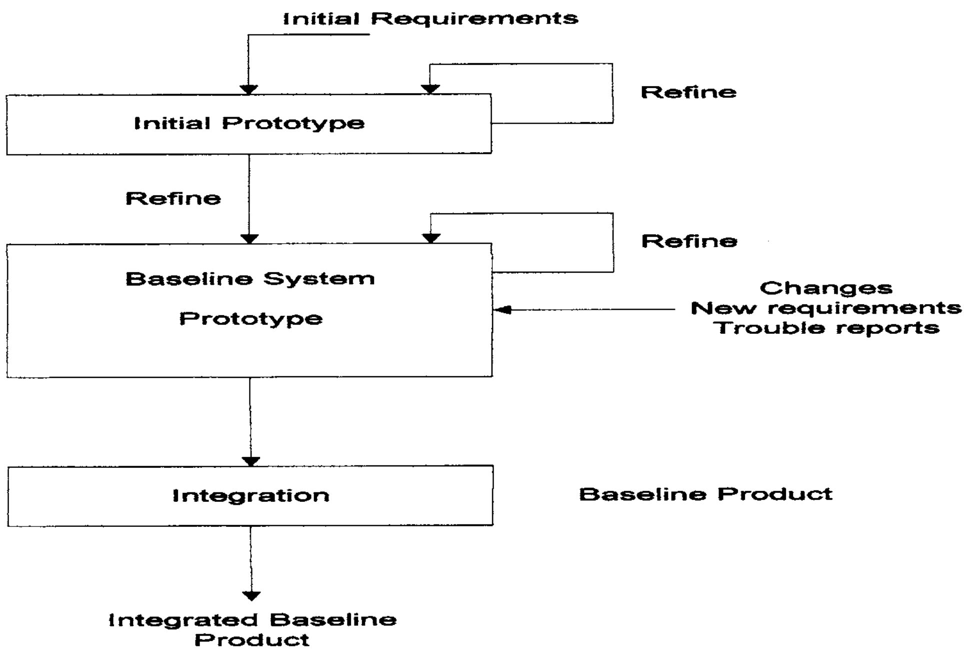


Figure 2 Miller's basic model

[2] Subsequently superseded by MIL-STD-498, issued on 5 December 1994 (`http://jcs.mil/htdocs/teinfo/directives/soft/ds2167a.html`).

When the development system has reached a stable state, the knowledge engineer can rewrite the system or upgrade it to a delivery system in the environment required. It should be noted that extensive validation and verification is performed through the incremental build, based upon three testing techniques: new function testing, critical defect testing and regression testing. This methodology is one of the first to fully realise the need for extensive validation and verification and is in part due to the nature of the systems created through it, which included systems for the Nuclear Regulatory Committee.

This approach to system development produces KBSs that more closely meet Miller's own quality and formality requirements. However, even with the level of documentation required for 2167(A), the approach lacks formality in certain areas, such as knowledge representation, elicitation and acquisition. In order to overcome these deficiencies, Miller introduced a refinement of the methodology as shown in Figure 3, which included multiple sub-stages and processes in developing the specification towards systems integration. This approach is still highly practical in nature and beneficial to real-world industrial-strength system developers.

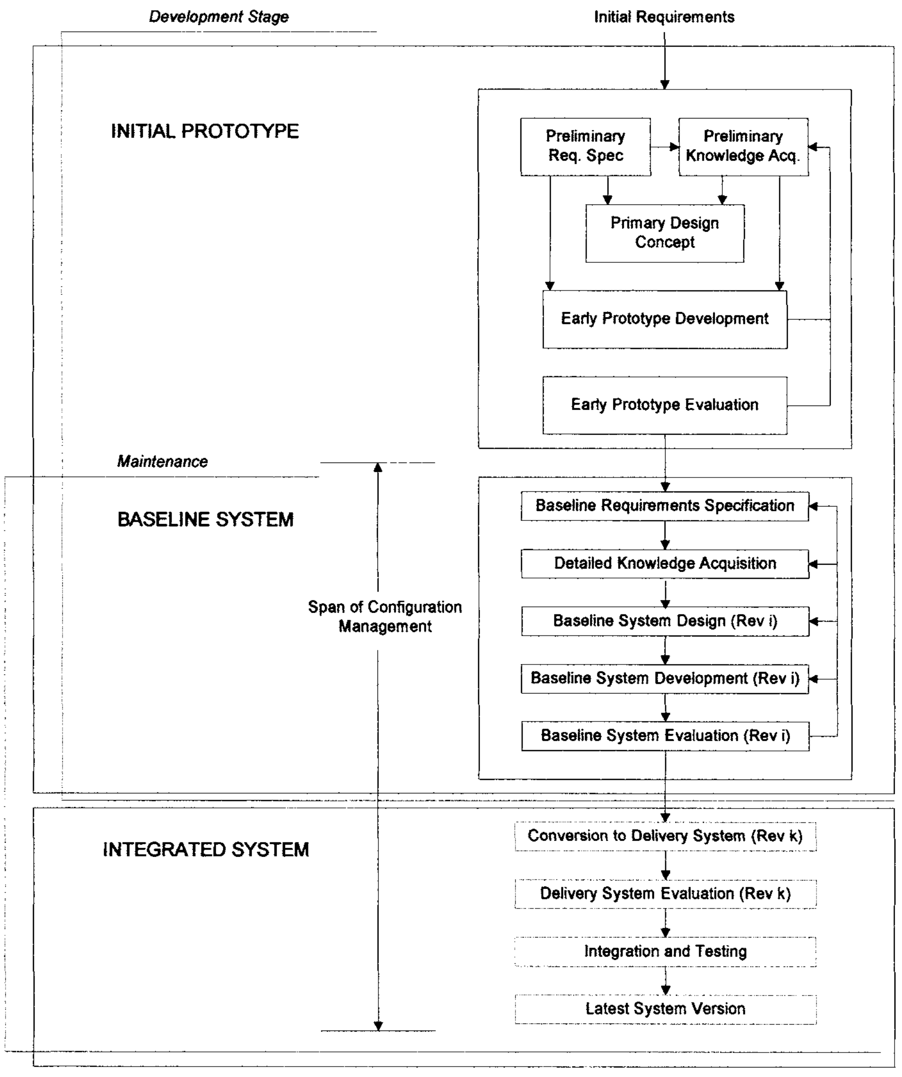

Development Stage
Initial Requirements
INITIAL PROTOTYPE
Preliminary Req. Spec
Preliminary Knowledge Acq.
Primary Design Concept
Early Prototype Development
Early Prototype Evaluation
Maintenance
BASELINE SYSTEM
Span of Configuration Management
Baseline Requirements Specification
Detailed Knowledge Acquisition
Baseline System Design (Rev i)
Baseline System Development (Rev i)
Baseline System Evaluation (Rev i)
INTEGRATED SYSTEM
Conversion to Delivery System (Rev k)
Delivery System Evaluation (Rev k)
Integration and Testing
Latest System Version

Figure 3 Miller's industrial-strength methodology

3.3 ANSI/AIAA AISE model

The advent of embedded KBS in critical aerospace environments led military software designers to consider the problem of producing KBSs under strict managerial constraints, such as those imposed by DoD-STD-2167A and its successor MIL-STD-498.[3] This task was not dissimilar to Miller's industrial-strength Model, which was motivated by the Nuclear Regulatory Committee and the Electrical Power Research Institute (EPRI). Building upon the early work of the CSC corporation's Expert Systems Development Methodology (CSC, 1989a; 1989b; 1989c), the American Institute of Aeronautics and Astronautics together with ANSI developed a model that could be mapped to the United States Department of Defense criteria. The basic model (ANSI, 1992) is illustrated in Figure 4.

It shows a five-phase model that links together the phases of application identification, prototyping, development and integration, and integrated test and maintenance. This stage-based approach is supplemented by the strict adherence to deliverables and review parameters, following a DoD-STD format. Its aim is to simplify the process of creating milestones and their associated deliverables.

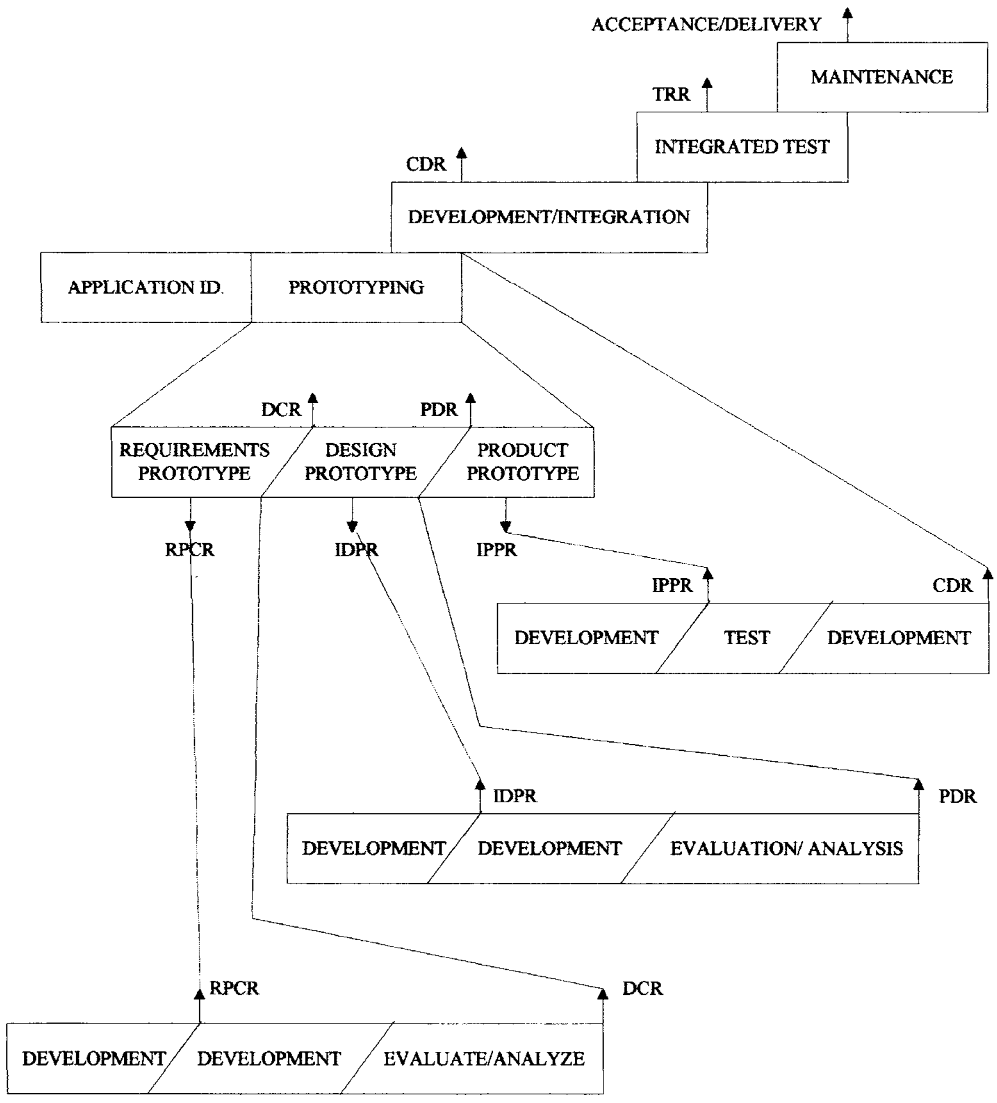


Figure 4 The AI Software Engineering (AISE) model for KBS development (AISE, 1992)

[3] `http://www.pogner.demon.co.uk/mil_498/`

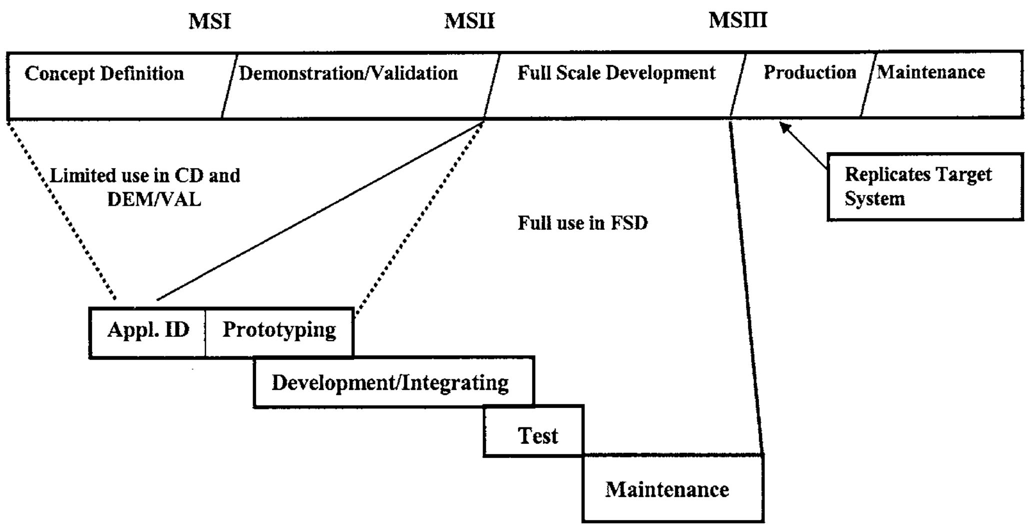


Figure 5 AISE tailoring for acquisition cycle phases (AISE, 1992)

The methodology is layered whereby one of the top-level processes, such as prototyping, can be expanded out to subsequent levels of detail (as illustrated in Figure 3) where the prototyping is shown to encompass three sub-stages: requirements prototype, design prototype and product prototype. These are again complimented with milestones and review stages, in the form of Requirements Prototype Concept Review (RPCR), Design Concept Review (DCR), Initial Design Prototype Review (IDPR), Preliminary Design Review (PDR) and the Initial Product Prototype Review (IPPR).

The expansion of the prototyping aspect continues to a third level where the requirements prototype, the design and the product prototypes are themselves expanded to a level of detail such that the milestones can be achieved. The requirements prototype is illustrated in Figure 5.

Figure 5 shows the main focus of the requirements prototype is to obtain an understanding of the "breadth" and "scope" of the knowledge base and the Computer Software Configuration Items (CSCI). Again, this throw-away prototype is ascertained with the assistance of reviews and milestones. This phase also attempts to establish the groundwork for selecting a target environment for development.

The design prototype builds upon the requirements prototype to incrementally create the full design as well as a working and functionally complete system. In parallel development, the interface issues are moved towards resolution with those issues surrounding the final target environment. The third prototype phase is the "product prototype" in which the system is refined into a final production. This necessitates the performance, computational and interface requirements to be fully satisfied.

The first review, the IPPR, is an assessment review to determine whether or not the KBS components (CSCI) meet requirements, while the second review, the Critical Design Review (CDR), assesses the final prototype system against the full functionality of the requirements.

The AISE methodology tackles the same problems as Miller's methodology in a similar way, utilising a series of prototyping stages and reviews that moves the initial requirements through a solidification process to full implementation and integration. Miller uses the term "operational concept" which AISE uses as evaluation criteria in the design concept review. Thus the two approaches can be seen as complementary, as well as being able to map onto the DoD-STD-2167A and MIL-STD-498 system life cycle models.

3.4 MOKA

MOKA (Methodology and tools Oriented to Knowledge-based engineering Applications) is the result of the Advanced Information Technology (AIT) pilot phase ESPRIT Project 7704, which subsequently became ESPRIT 25418, involving industrial users (Aerospatiale-Matra, BAE Systems, Daimler Chrysler and PSA Peugeot Citroen), IT vendors (Knowledge Technologies International & Decan

Consulting and Services) and academia (Coventry University) (Stokes, 2001). MOKA's goal was to produce a methodology well suited to Knowledge-Based Engineering (KBE) that builds upon and utilises the strengths of other approaches, including object-oriented approaches to modelling the KBE environment through the Unified Modelling Language (UML) (Booch *et al.*, 1998) widely adopted by industrial systems developers. The MOKA approach embodies the same set of principles as CommonKADS and the KADS expertise model to describe the engineering design process (Stokes, 2001). The industrial aspect of this method requires intra-operability and an open standards philosophy. Thus MOKA embraces the "Knowledge Interchange Format" (KIF) (Genesereth, 1991), Ontolingua (Fikes *et al.*, 1997), and CYC (Lenat, 1995) for its ontological basis, and the Standard for the Exchange of Product Model Data (STEP) ISO 10303 and the EXPRESS data modelling language to enable information exchange between participants.

MOKA uses an iterative life cycle model that has six stages: identify, justify, capture, formalise, package, activate (then repeat). The methodology primarily focuses upon the capture and formalise aspects of the knowledge engineering design process through a two-level model: an informal level and a formal model level. The informal level models the environment through templates or forms that store knowledge components in a standard format that can then be analysed for linkages and reference purposes during knowledge-base verification. Five types of template are used: Illustrations, Constraints, Activities, Rules and Entities (ICARE). Having created an informal model, the MOKA advocates the creation of a formal model through a refinement process. The formal notation is a graphical language based on UML known as MOKA Modelling Language (MML) and is used to create meta-models through views and class objects. The formal model itself is composed of two sub-models: the design process model, which "records the design rationale of an application" (Stokes, 2001), and the product model, which "contains all the knowledge that describes the product itself" (Stokes, 2001) (equivalent to the domain representation in the MK model and the domain layer in KADS).

The MOKA project is geared to be compatible and interrelated to as many other approaches and standards as possible within the industrial landscape. To that end the MOKA approach suggests the use of XML in the modelling and distribution of files created in developing the system model, giving their model greater reach in collaborative commerce applications or across the extended enterprise.

4 The formal models

The balance between speed and rigour in development is for some applications heavily focused upon rigour, such as systems developed by NASA (Robinson, 2002) and military systems,[4] prompting the need for several approaches of a more formal nature.

4.1 A task-based specification methodology

In the quest to create a formal approach to KBS development that was independent of the problem-solving architecture, Yen and Lee (1993) proposed a Task-Based Specification Methodology (TBSM) to allow the knowledge engineer to create a set of specifications of a system's functional units, or tasks, at any point in the development. They define a TBSM specification as "having two primary components: a model specification that describes the system's static properties and a process specification that describes its dynamic properties. TBSM uses a task/method/sub-task approach to capture these levels of specification at the appropriate level of abstraction, letting the knowledge engineer refine the overall specification by constructing more detailed model and process specifications" (Yen & Lee 1993). Figure 6 illustrates their model.

The TBSM uses a series of specifications, at a variety of levels of abstraction, to allow "the developer to identify inconsistent and incomplete specifications either within or between multiple

[4] `http://carlisle-www.army.mil/usacsl/divisions/std/branches/keg/cog-disciple.htm`.

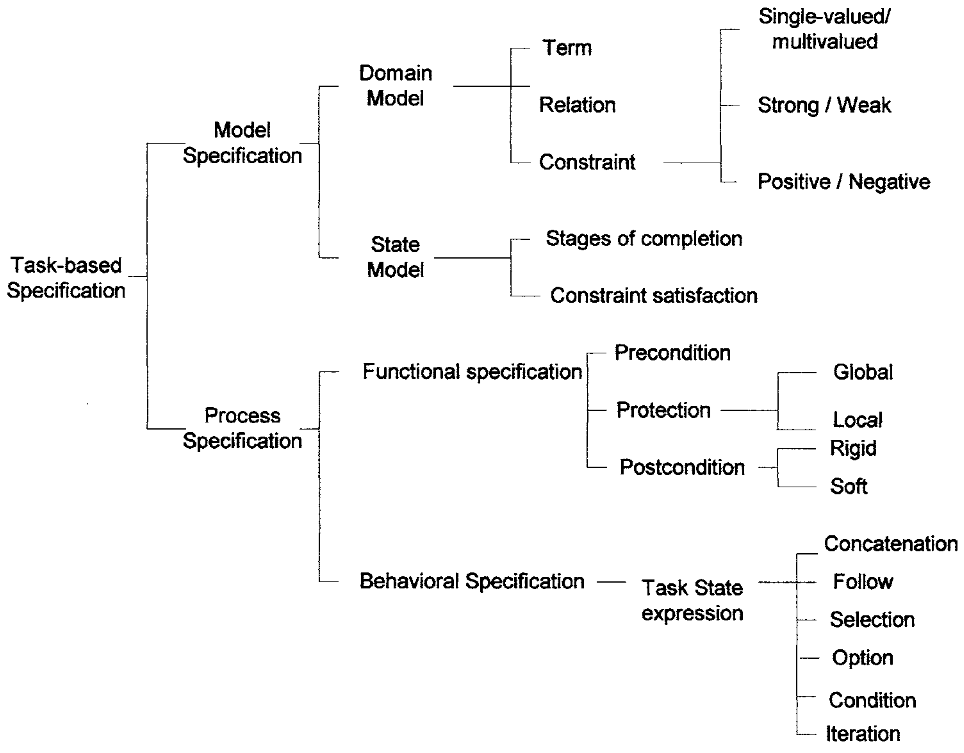


Figure 6 Components of the task-based specification methodology

levels, based upon their formal semantics" (Yen, 1993). TBSM utilises weak and strong inconsistency checking as well as consistency and completeness checking (Yen, 1993). The methodology is designed to be highly suitable to verification processes.

4.2 A meta-knowledge model

The framework defined by Plant and Gamble (Plant, 1987; 1997) uses a meta-knowledge model to link the "stages" of system creation adhering to Newell's knowledge-level construct (Newell, 1972). The motivation for this Meta-Knowledge (MK) approach is to create rigorous systems that emphasise formality, analysis and the justification of actions taken in the development. Included in this context are the consideration of corporate standards, the evolution of technology and encompassing of external factors such as total quality management or ISO 9000 standards inspections. This is similar to what Hilal and Soltan (1993) refer to as circumstantial occurrences.

As noted earlier, it is infeasible to create a rigorous, mathematically sound specification for the majority of KBSs prior to a thorough analysis of the domain space. In order to overcome the problem of weak specifications, the MK model combines several techniques that fit into a life cycle based on multiple levels of refinement. Initially the problem specification is captured as a rough "operational concept" borrowing from Miller. This conceptual design becomes refined by three "levels" or process stages, towards a set of specifications, each of which presents a different view of the domain space, e.g. a domain specification, a representation specification, a cognitive engineering specification. They can be combined to form a composite specification with maximum effectiveness.

The approach, depicted in Figure 7, begins with the initial "operational" specification of the system and through utilisation of the meta-knowledge surrounding the project aims to create a more definitive baseline specification. Repeating the process to a stable point constructs a sufficiently robust specification. This new stronger specification acts as the basis of the knowledge elicitation phase that again utilises meta-knowledge to drive the process. A difficulty frequently encountered at the elicitation stage is that the knowledge may well be in many formats, making the validation and

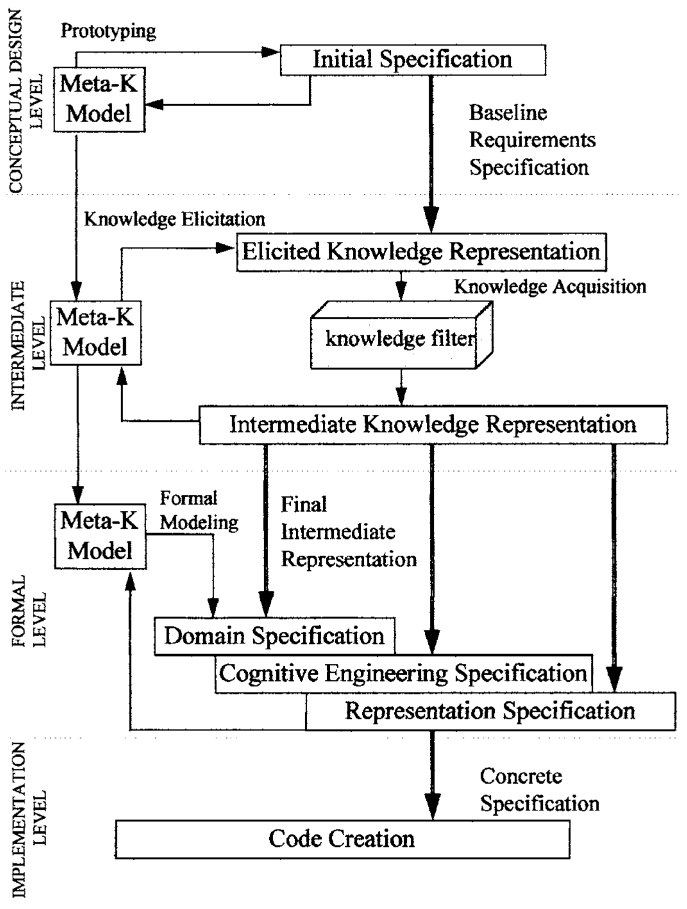


Figure 7 The MK model approach to knowledge-based system development (Plant, 1997)

verification process extremely difficult. These "elicited knowledge representations" need to be normalised into a semi-formal notation in the form of the intermediate knowledge representation. This is the role and function of the knowledge filter.

The knowledge filter, depicted in Figure 8, is an application of the knowledge level in which Newell proposes that knowledge and representation are intimately linked and that computational systems are really a set of levels such that "a level consists of a medium that is to be processed, components that provide primitive processing, laws of composition that permit components to be assembled into systems, and laws of behaviour that determine how system behaviour depends on the component behaviour and the structure of the system" (Newell, 1972). The knowledge filter concept acts according to his premise that "each computer system level is a specialisation of the class of systems being described at the next lower level" (Newell, 1972) in that the elicited representation is refined through a set of sub-processes, e.g. erotetic logic and conversational coherence, that act as specialist filters and refine the information, knowledge and data into a more robust form. This comprise results are captured in the semi-formal, intermediate knowledge representation, that concurrently provides meta-knowledge for the continued refinement process.

This process allows for gross validation issues to be recognised and the elicitation cycle to be repeated until the issues are clarified. As a critical mass of the system is defined, the information, data, knowledge and meta-knowledge are focused into the next knowledge level, the formal level, in which formal notations are utilised to normalise the knowledge set, upon which rigorous verification can then take place. These are the domain, cognitive and representation specifications. These separate specification categories allow the knowledge, the interface and representational issues to be focused upon individually, but aligned through the meta-knowledge and the refinement process. Subsequent to this, the knowledge is refined into a coded system.

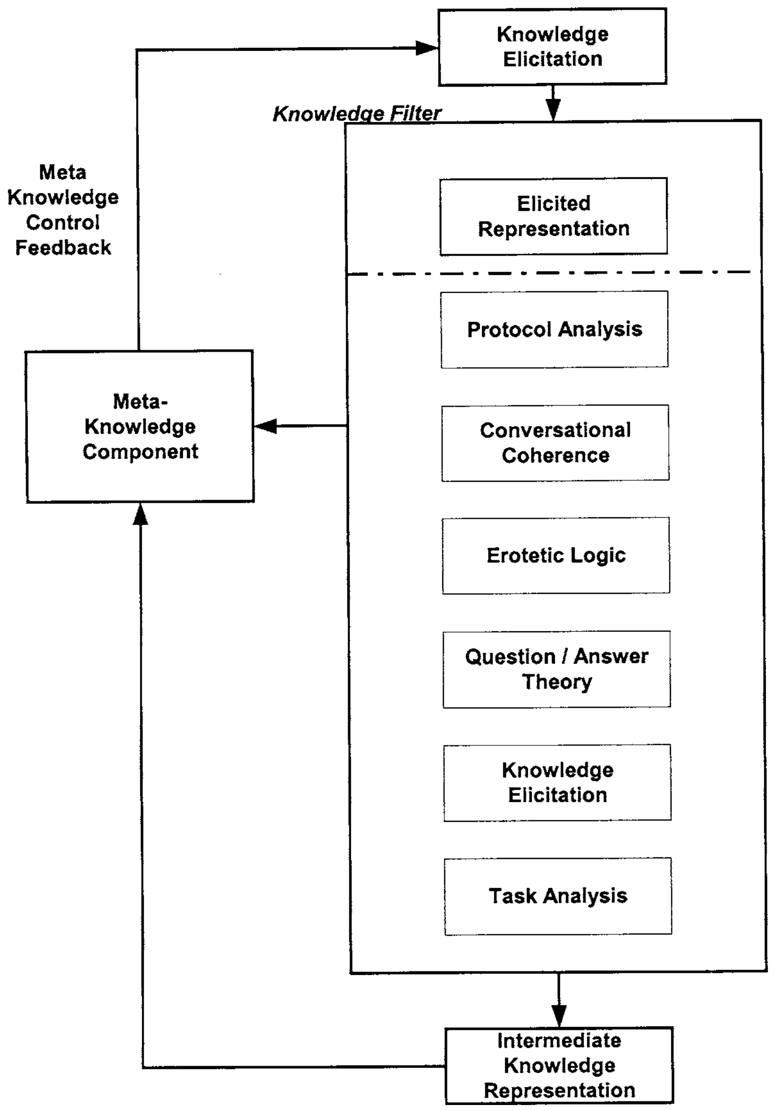


Figure 8 The intermediate level of development (Plant, 1997)

4.3 CommonKADS

Researchers at the University of Amsterdam working through the ESPRIT initiative defined the KADS methodology (Wielinga *et al.*, 1993a; Wielinga *et al.*, 1993b). Their approach, much like Buchanan's, is an analogy of methodologies used for the development of traditional software with the following six stages: analysis, specification, design, implementation, testing and validation. In KADS, the first of these stages is viewed as the most important – the analysis of the knowledge and the problem-solving methods relevant to a certain domain. Thus it requires as detailed an investigation as possible. The rationale is that analysis is fundamental to all software engineering problems: the earlier problems and errors are detected, the lower the cost of making amendments. KADS identifies five types of knowledge analysis that are designed to help define the problem area: domain analysis, task analysis, analysis of the task environment, analysing the user and analysing the expert. After the domain definition has taken place, the type of analysis changes and the domain knowledge acquired from elicitations, such as think-aloud protocols, can be examined using techniques like the "protocol analytic method" (Wielinga *et al.*, 1993b). On the basis of these analyses a specification of the system can be constructed detailing the constraints on the knowledge representation, inference mechanism, user interface and performance. In the third stage, that of design, the actual tools for knowledge representation and inference are chosen or constructed. The fourth and fifth stages are where implementation and testing produce cyclical and incremental development of the knowledge base which can then be validated. Overall, the approach develops an awareness and need for specifications as a rigorous philosophical base.

The original focus of KADS was on the knowledge acquisition aspects. However, this changed as it became apparent that knowledge engineering could not be considered only from the "capture" and "representation" of elicited knowledge, but necessitated a knowledge-level perspective (Newell, 1972) within a multi-viewpoint modelling paradigm. The state of the art in knowledge-based technologies also expanded rapidly from the late 1970s as KBSs increased in sophistication and complexity toward heterogeneous, embedded environments in contrast to earlier stand-alone, diagnostic systems. In order to accomplish the successful creation of this type of KBS, a methodology was needed to allow the developer access to, and control over, many more aspects of embedded design, such as interface issues and total system architectural considerations as well as managerial and planning issues. These issues, as well as problems pertaining to the original KADS development, are considered in detail within a second ESPRIT project (ESPRIT 5248), which led to the creation of KADS-II and ultimately to CommonKADS (Schreiber *et al.*, 1999).[5]

CommonKADS is a methodology to create a series of interconnected models, as illustrated in Figure 9. CommonKADS can be viewed from the same organisational perspective as the MK model discussed earlier in that it recognises the need for a larger organisational perspective to be obtained during the creation of the KBS. This perspective involves going beyond the technical objectives and creating a KBS that impacts the organisation, as well as improves it.

CommonKADS uses six models in Figure 9: organisation, task, agent, communication, expertise, design. Each of these models focuses upon a specific aspect of the problem and architectural environment. The six models serve several purposes. First, they act as a vehicle of communication among the stakeholders of the system, such as the knowledge engineer and the domain expert. This specification function attempts to overcome the long-standing criticism that KBSs cannot be adequately defined. It is similar to the MK model's use of a composite specification scheme. Both

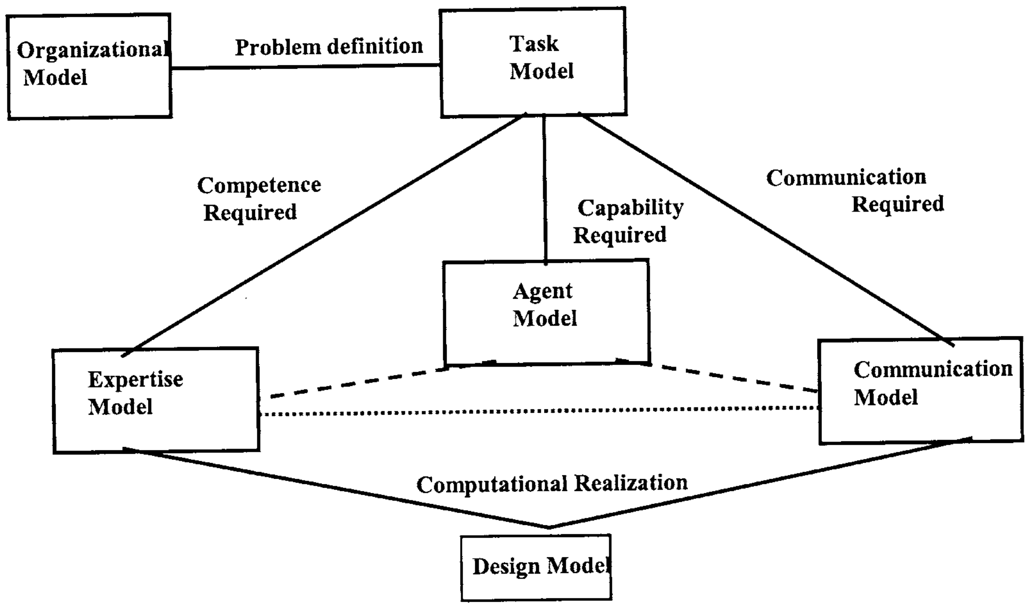


Figure 9 CommonKADS and the relations between its models (de Hoog *et al.*, 1994a)

[5] `http://www.swi.psy.uva.nl/projects/CommonKADS/home.html`, `http://www.commonkads.uva.nl/frameset-commonkads.html`.

CommonKADS and the MK model integrate the KBS specification into an environment where embedding the system is likely. The second purpose of the model is to facilitate risk management in KBS development. Finally, the model enables the creation of a reuse library. We can briefly consider the six models and their relationships.

The *organisation model* sets the context for the knowledge-based activities that the project is concerned with, such as functions, structure, process, power relations and resources (de Hoog *et al.*, 1994b). The objective is to identify and store pertinent data about organisational features. The organisational model template is illustrated in Figure 10, which shows the separate aspects of the model and their interrelationships.

The template is composed of eleven "components" below:

- the organisational context,
- the problem and opportunities of the organisation which need to be addressed,
- the current problem the organisation is working/focused on,
- the solutions identified for the current problem,
- the function component that identifies the functions of the organisation,
- the process component that describes inter-organisational dependencies and how functions are carried out within the organisation.,
- the structure component defines the organisational structure of the company,
- the people and their roles are defined,
- the organisational knowledge at different levels is defined,
- the computing resources of the organisation,
- other resources pertinent to a particular organisation that need to be defined and
- the power of the organisational members.

Creating a template for these components and considering the relationships between them places the problem-solving process in its organisational context.

The *task model* contains the tasks that realise the organisational functions in the form of a task structure, where each task is characterised by its input, output, control, various features, environmental constraints and required capabilities (Duursma *et al.*, 1993). The process by which this template is filled out is known as "task analysis". The task template has eight slots as shown in Figure 11: name, goal, description, performance, control structure, performance time, frequency pattern, and decomposition pattern.

This template is supported by four other entity templates: a feature template, an ingredients template, a capability template and an environment template, in conjunction with relationships that link it with the organisational, expertise, communication, agent and design models.

The task model is utilised to identify the organisation in terms of tasks. However, these need to be defined formally, and this is the role of the agent model. The *agent model* (Waern, 1993a) defines all of the properties of a task, its capabilities and constraints, as well as any reasoning capabilities a task may need or have. Thus the agent model maintains a collection of relevant properties of the agents (user, a KBS or another software system) for different tasks. These properties and interrelationships are shown in Figure 12.

A philosophical foundation upon which CommonKADS is based is that of providing a framework for embedded systems to be created. This type of system requires the specification of its interfaces, which is achieved through the *communication model* (Waern, 1993). In the communication model transactions (transaction plans, ingredients and initiatives) between agents are represented as additional tasks. These are needed to accommodate the assignment of task information needs to a variety of different agents. This model, as illustrated in Figure 13, captures the user models, the communication tasks and the transfer tasks.

The rationale for the communication model is:

> The task distribution defines communication tasks in order to achieve the overall goal of the task. The agent model and the expertise model give constraints on those communication tasks as well as

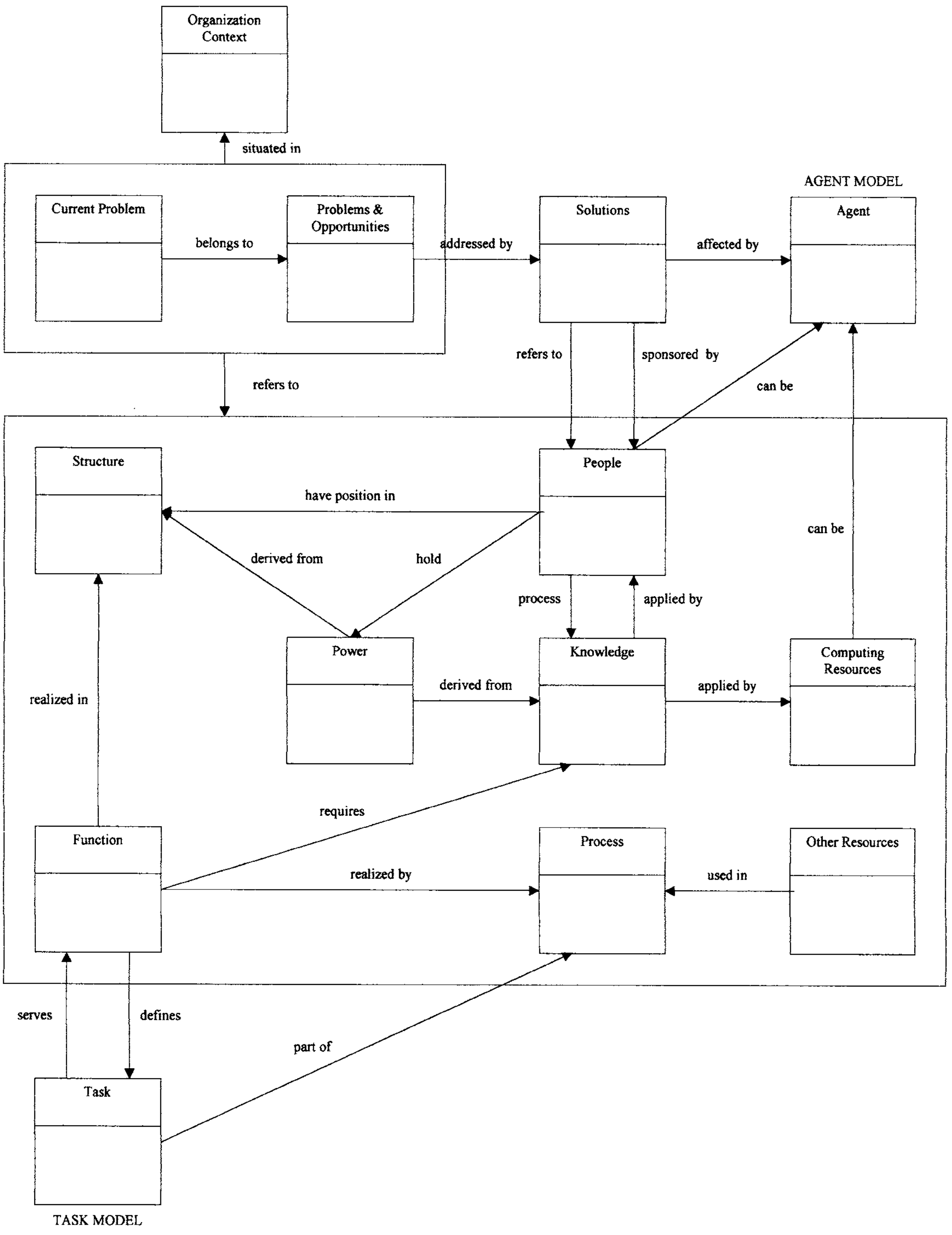


Figure 10 The organisation model template (de Hoog, 1994)

on additional tasks. The communication model provides the concepts and mechanisms for a communication based on the competence of the communicating agents.

(Waern, 1993)

The fifth model focuses upon the knowledge that is to be incorporated into the system. The creation of an expertise model is a complex task, and a complete description of this process is provided in the extensive CommonKADS literature (Wielinga *et al.*, 1993b). In this model, the knowledge and use-specific structure of an agent relevant to a particular task that is described as domain knowledge, task

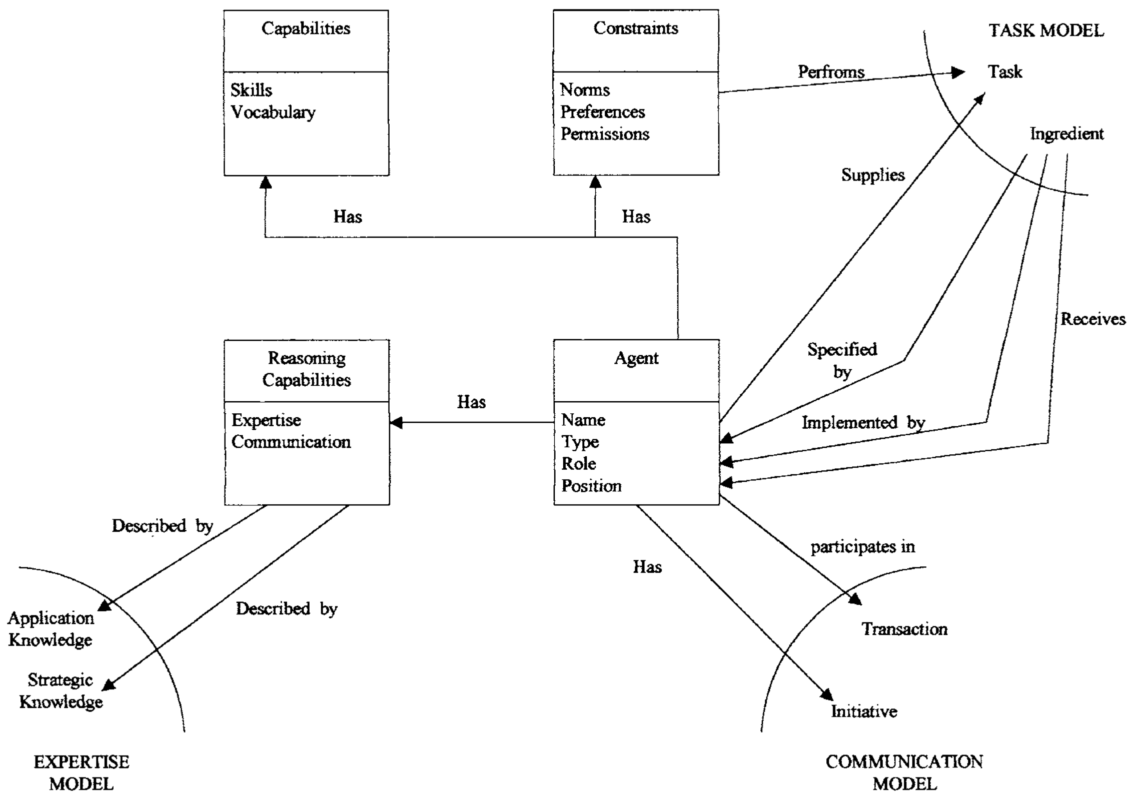


Figure 11 ER-diagram of the task model of the CommonKADS methodology (Duursma *et al.*, 1993)

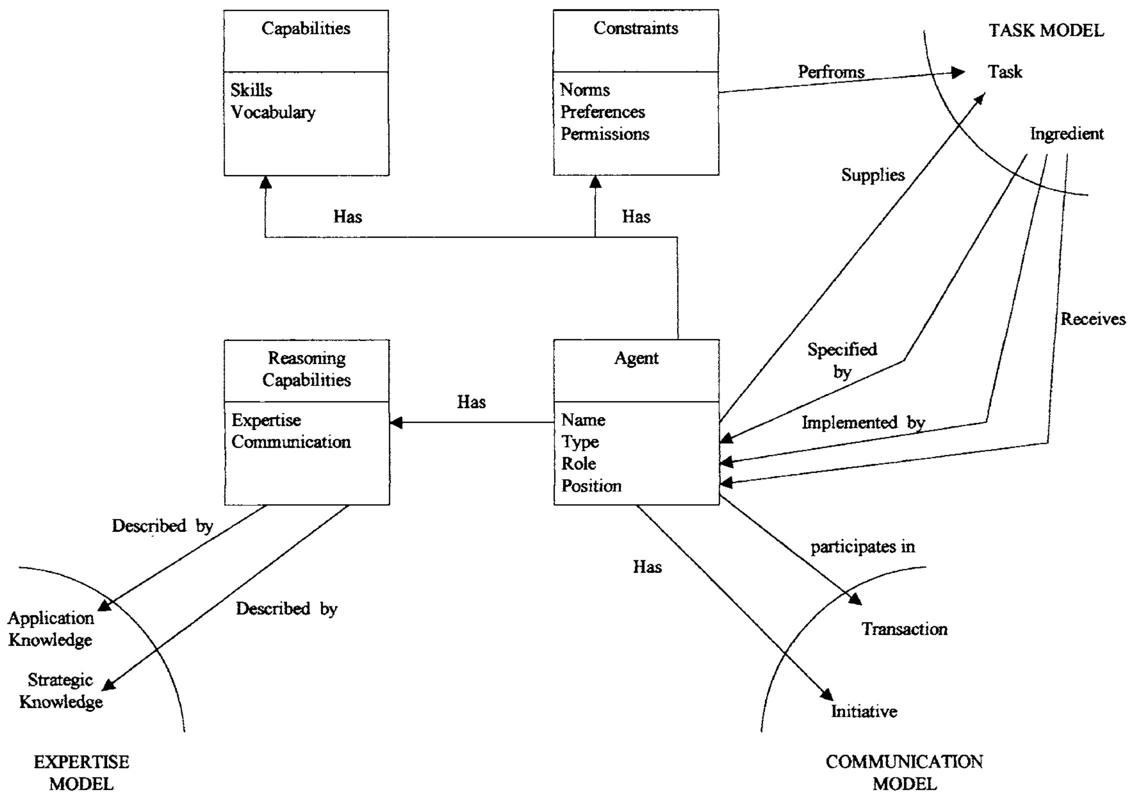


Figure 12 Agent model objects (Waern, 1993a)

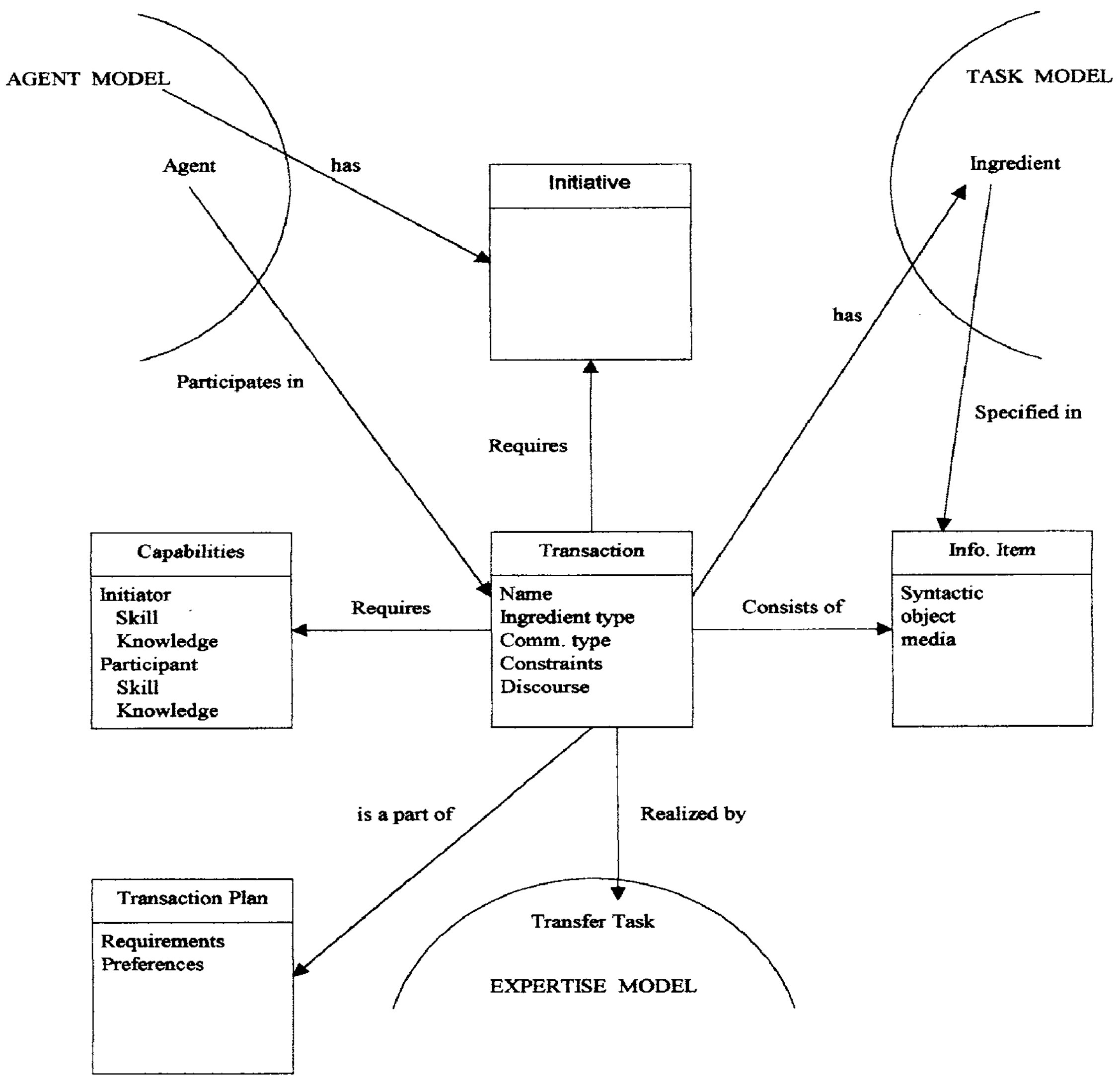


Figure 13 Communication model objects (Waern, 1993)

knowledge and inference knowledge, and various mappings between them. An overview of the model is provided in Figure 14.

The two main, high-level epistemological categories of knowledge, *application knowledge* and *problem-solving knowledge*, are defined in Tables 5 and 6. The application knowledge category is composed of domain, inference and task knowledge, while the problem-solving knowledge is composed of problem-solving methods and strategic knowledge. The aim here is similar to that of the domain specification – a cognitive engineering specification and the representation specifications of the MK model approach to knowledge-base system design. Individual aspects of the expertise are captured in specialised formats that allow validation processes to be performed, prior to unification of the knowledge in the final system.

In the framework of the expertise model, CommonKADS identifies four approaches to constructing the model:

1. Data-driven expertise modelling (Wells, 1994), where a model is created through an elicitation, acquisition and formalisation process with little reuse or library facilities utilised. This is, in essence, Buchanan's approach to KBS creation.
2. Select-and-modify approach (Orsvarn, 1993), in which a library reuse approach is utilised and stems from the work on KADS-I.

Figure 14 Major components of the expertise model (de Hoog; 1994a; Wielinga *et al.*, 1992)

Table 5 CommonKADS: application knowledge types

Knowledge type	Definition (de Hoog, 1994a)
Domain	This expresses what is known about the application domain of a task. It refers both to the knowledge about the specific systems that are the subject of problem solving and to the general knowledge about them. This knowledge consists of the domain ontology, which is the way an application sees the world, and a domain model, which captures groups of statements about the domain that can be generated by the domain ontology. Domain knowledge may further be specified with the help of some meta-descriptions, model ontologies and schemata, which specify the type and structure of domain models
Task	Knowledge about a task relates to the goal of the task, as well as to the activities that contribute to the achievement of the goal. The goal and the activity aspects of a task are specified in respectively the task definition and the task body.
Inference	This category specifies basic inferences that can be made using the domain knowledge. These can be linked to form inference structures

3. Compositional modelling from library elements (Schreiber *et al.*, 1994), whereby a set of low-level generic library components are utilised and modified to create the customised model.
4. Refinement approaches to knowledge modelling (Akkermans *et al.*, 1994), which are used when no predefined library components are available for the construction of an expertise model in a new domain area. Therefore, the generic components are refined into a form and framework to create a model for the application.

CommonKADS also provides a semi-formal notation for the specification of CommonKADS expertise models, known as the Conceptual Modelling Language (CML). This language has a similar role to that of the formal mathematical modelling notations utilised in the MK methodology. In order to place the sixth and final model in perspective, it is necessary to return to Wielinga's definition of the CommonKADS philosophy:

> The development of a KBS is seen as the construction of a set of models of problem-solving behavior, seen in its concrete organization and application context. A KBS is a computational realization associated with these models.
>
> (Wielinga *et al.*, 1992)

Table 6 CommonKADS: problem-solving knowledge types

Knowledge type	Definition (deHoog, 1994a)
Problem-solving methods	Since the domain of strategic knowledge is problem-solving, the associated domain knowledge consists of various problem-solving methods. These problem-solving methods are dynamic "model construction methods" in the sense that they dynamically generate structures that are comparable to the original KADS-I models (minus the strategic layer). For example, they can generate a task body from a task definition, given certain features about the task environment or structure of available domain knowledge.
Competence theory	The competence theory is a theory about the required competence of the application problem-solver. As such it can be seen as meta-domain knowledge.
Strategic	The inference and task aspects of problem-solving knowledge together are called strategic knowledge. They can be seen as the operationalisation of rationality principles employed by a problem-solving agent. Currently, there are no predefined components from which one can build the task and inference structures for strategic reasoning.

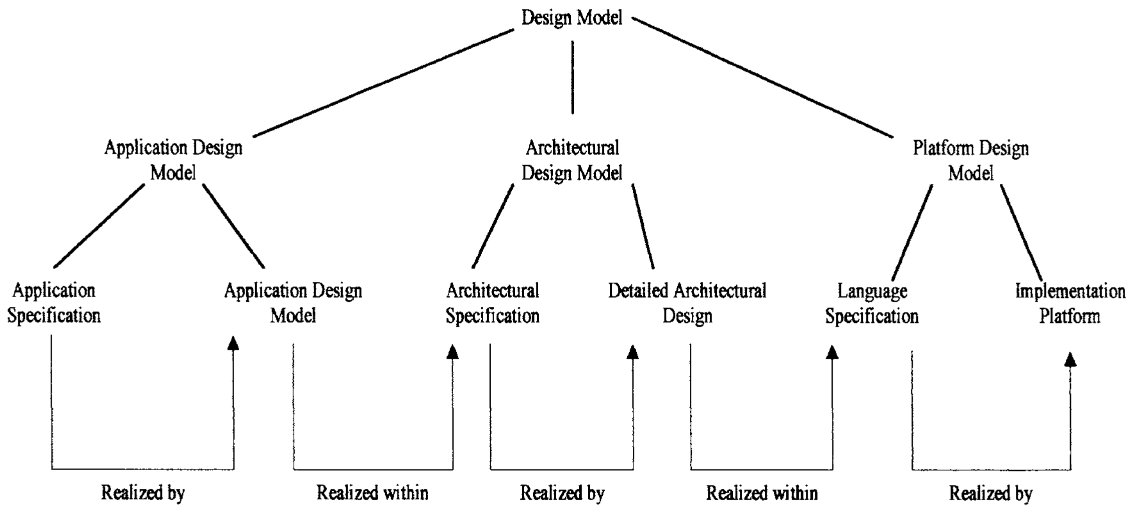


Figure 15 The top-level decomposition of the design model (Van de Velde *et al.*, 1993)

Hence the *design model* is the realisation of the problem-solving behaviours described in the expertise and communication model in computational and representational terms (Van de Velde *et al.*, 1993). This model is shown in detail in Figure 15, where three functional design areas are indicated: *application design*, *architecture model* and *platform design*.

The *application design* documents the application specification of the application. This is performed at the conceptual level, in which the application specification is created through a High-Level Application Design Language (HADL) that is specific to CommonKADS.

The *architectural mode* details the high-level "computational infrastructure" in which the application design, in the form of the detailed design, will ultimately be implemented. This model covers two types of architecture: the computational architecture and the interface architecture. The *computational architecture specification* is "the abstract machinery that is used to implement the reasoning capabilities of an application. It consists of computational objects and computational methods to describe an architecture within a certain paradigm. The commands specification describes the external interface to the architecture" (Van de Velde *et al.*, 1993). The Interface Architecture Specification is "the abstract machinery that is used to implement the interaction facilities of an application. It consists of interface activities to describe an interface architecture within a certain paradigm. The events specification describes the external interfaces to the architecture" (Van de Velde *et al.*, 1993).

The *platform design* details the target language for hardware and software. The platform design also interacts with the agent and organisational models in specifying the computing resource requests of both the organisation and user environments.

Having created a model set of a given problem domain the second aim of the CommonKADS philosophy is to facilitate both model reuse as well as the representation of models from other design systems and frameworks. These are approached through the CommonKADS expertise modelling library (Breuker & Van de Velde, 1994). This library provides a framework within which developmental expertise can be captured for future use. The process balances the scope and detail of the reused characteristics and the representation framework, so that the model information is as complete as possible yet represented in a wide variety of potential future models. This is a step forward from the intent of the KADS-I library that only stored "skeletal" models or problem-solving techniques (Breuker *et al.*, 1987).

The inherent complexity of KBS creation, through the CommonKADS process model, necessitates the utilisation of a project management component that is "risk-driven" to minimise the developmental failure aspects. This is pursued through a result-oriented, cyclic approach to system management. The basic premise is based upon Boehm's spiral model (1988) of project life cycle management, where the spiral has been made cyclic in a style reminiscent of the multi-level feedback cycle utilised in the MK model as the basis of its quality and management developmental principles. Similarly, CommonKADS utilises a review, risk, plan, monitor four-phase project management activity cycle (Wells, 1994).

Research has been performed to extend, transform and validate the KADS conceptual model towards an operational one that is formally defined through Petri nets and executed through simulation modelling (Le Goc *et al.*, 2002). John Kingston and a research group at the AI Applications Institute define several extensions to the KADS methodology, including a framework for cooperative working (Kingston, 1993) and a "Pragmatic KADS" for commercial KBS systems that cannot absorb the traditional large overheads associated with the full KADS version (Kingston, 1992). Moreover, they demonstrate the use of KADS to small applications (Price & Kingston, 1993).

5 A comparision of industrial and formal approaches

The concept of describing a model as "industrial strength" stems from their applicability to large-scale projects that require the production of a manageable, modifiable, stable, verifiable resultant system. The industrial strength systems described in Section 3 have been utilised by corporations, governments and entities that require a system development methodology that fits into a larger structure both from a developmental perspective and from an organisational perspective, for example where a KBS has to be embedded in another system and operate under a military development quality assurance framework and perhaps also a language framework, such as the XD ADA MIL-STD-1750A Emulator Support Option for OpenVMS Systems. The formal approaches were developed explicitly to solve large complex knowledge-based systems implementations and are comprehensive in nature. The complexity of the approaches, the need to perform extra work to map the solution to other standards such as the US MIL-STD-499B, and their utilisation of formal methods, are often initially perceived by organisations as barriers to their adoption. However, the formal models are extremely powerful as they are specifically designed for complex KBSs and with effort they can be incorporated into other standards. The size, intricacies and extent of formal methodologies are inhibitors for their utilisation in small, one-off KBSs or in systems development for inexperienced users who have a short development time window. In such a case, it is probably that the conceptual basis for system development and its subsequent verification criteria can be matched with an industrial strength model to create a successful development environment with known constraints and quality levels.

6 Methodology quality assessment

In Sections 3 and 4, we consider eight methodologies for the development of KBSs. This section uses research in methodology assessment, drawing upon the literature from the validation and verification community (Hilal & Soltan, 1991; Howard *et al.*, 1999; Miller, 1990; Mitev, 1994; Nandhakumar & Avison, 1999; Plant, 1997; Preece, 1995; Wasserman & Freman, 1983).[6] We make an empirical evaluation of the rigour and quality of the systems that may be derived from the discussed approaches. Not all assessment models are applicable for all the methodologies. Thus the comments and results are not exhaustive.

Table 7 uses the seven criteria proposed by Miller in conjunction with the five additional criteria proposed in this paper to give results for the industrial strength and formal model categories. (The models in the knowledge acquisition and early prototyping category are not assessed, as their primary intent is not the whole methodology. Hence it is unfair to judge them against these criteria.) The table acts only as an indicator of the presence of the assessment properties due to the variability inherent in the models' adoption and usage. The values associated with the criteria of assessment are also not meant to be exhaustive and differing degrees of grain size could be used.

Table 7 shows a movement from stage-based methodologies (see Appendix A) with minimal validation and verification, adequate documentation or formal justification towards more rigorous life

[6] `http://www.csd.abdn.ac.uk/ apreece/Research/vavpage.html.`

Table 7 Quality assessment matrix

	METHODS AND VALUES							
Miller's extended criteria	Weitzel	Line/OCO	Miller	ANSE	MOKA	TBSM	MK	CKADS
1. Easily handles ill-formed or changing requirements	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
2. Suitable for stable system development	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
3. Maps well onto military type standards for consistency	Yes	Yes	Yes	Yes	No	No	No	No
4. Appropriate for embedded, real-time, data-driven systems as well as stand-alone knowledge-based systems	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
5. Provides for plan reviews and completion audits of requirements, design and implemented systems, with configuration management throughout	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes
6. Supports minimally competent validation and verification	No	No	Yes	Yes	Yes	Yes	Yes	Yes
7. Extends to maintenance activities	Minimal	Minimal	Yes	Yes	Yes	Yes	Yes	Yes
8. Mathematical techniques are used wherever possible or appropriate	No	No	Yes	No	Yes	Yes	Yes	Yes
9. Either a refinement process or a task-based or a stage-based approach is adopted	Staged	Staged	Transform	Staged	Refinement	Task-based	Transform	Model
10. Each step in the development can be traced back to the previous step and justified	Informal	Informal	Yes	Yes	Yes Rigorous	Yes Rigorous	Yes Rigorous	Yes Integrated
11. The documentation is adequate/rigorous	Adequate	Adequate	Rigorous	Adequate	Rigorous	Rigorous	Rigorous	Rigorous
12. Promotes independent specification of system features /requirements	No	No	No	No	Yes	Yes	Yes	Yes

cycle models that either promote transformation principles between rigorous stages or integrate them, such as CommonKADS. An interesting note is the influence of the military standard form of the life cycle, which is prevalent in the AISE, Miller and Weitzel methods. However, the movement towards methods that need to support complex embedded systems has made the strict adherence of this type

Table 8 TRILLIUM$_K$ quality assessment matrix

	Methods and values											
Phase	Buchanan	Davis	Grover	Alexander ADS	Weitzel	Line	Miller IS	ANSE	MOKA	TBSM	MK model	CKADS
Problem Specification	2	2	2	3	2	2	3	3	3	3	3	3
Conceptual model	2	1	2	3	2	2	2	3	3	3	3	3
Design model	1	1	2	-	2	2	3	3	3	3	3	3
Implemented system	1	1	2	-	2	-	3	3	3	3	3	3
Verification analyses	1	1	1	-	1	-	2	1	2	2	2	2
Validation analyses	1	1	2	-	2	-	2	2	2	2	2	3

of standard difficult and CommonKADS can be viewed as an approach to system creation that would make adherence to a military type standard difficult to adhere to. This is overcome through the enforcement of extensive quality standards in the CommonKADS environment

The result of a comparative evaluation for twelve of the methodologies against the TRILLIUM$_K$ scale is shown in Table 8, the MEDESS, the ideographic, and POMES are not included as they are prototype, proof of concept models while the ESPRIT 1098 project has been superceded by the KADS initiative. The methodologies progressively become more rigorous in a chronological fashion. They follow the research tradition of building upon earlier results and improving upon the weaknesses of the earlier models. This progression matches the demands placed upon the technology; as KBSs move from the research laboratory towards commercial deployment – at first as stand alone systems, then progressing towards larger environments, and finally becoming another embedded technique in complex systems. The methodologies continue to improve in terms of their validation and verification basis, which in conjunction with the application of formal methods to aspects of their development ensures that future methodologies will be even more rigorous. The TRILLIUM$_K$ scale is a useful mechanism through which a comparative evaluation of methodologies can be performed.

7 Comments and conclusions

The survey of KBS life cycle methodologies shows that techniques are maturing but have not yet reached a point at which developers can create systems following a totally rigorous framework based upon solid theoretical principles. This is due in part to the epistemological nature of the problem being modeled and in part to the lack of focused formal mathematical research into knowledge-based design principles. Current research is focused on modelling KBSs through multiple viewpoints, creating an understanding of validation and verification such that they can be used to re-engineer the life cycle models. Research is also ongoing in the use of "problem-solving methods" (Fensel, 1998) through which reasoning components are reused across applications. This area can be seen as benefiting from the current research in the use of formal techniques to model and specify KBSs. This research includes development of software architectural styles that pertain to knowledge-based system structure and behaviour (Gamble *et al.*, 1999). The overall aim of KBS research should enable developers to create robust, rigorous and provably correct systems that can be integrated with other systems such that they become a part of "conventional" software development (Howard *et al.*, 1999; Mitev, 1994). However, it is envisioned that the development of methodologies that facilitate flexible, robust and verifiable systems will continue to be a significant research challenge as the breadth and nature of software technologies continues to diversify, each of which may interface with or embed a knowledge-based component.

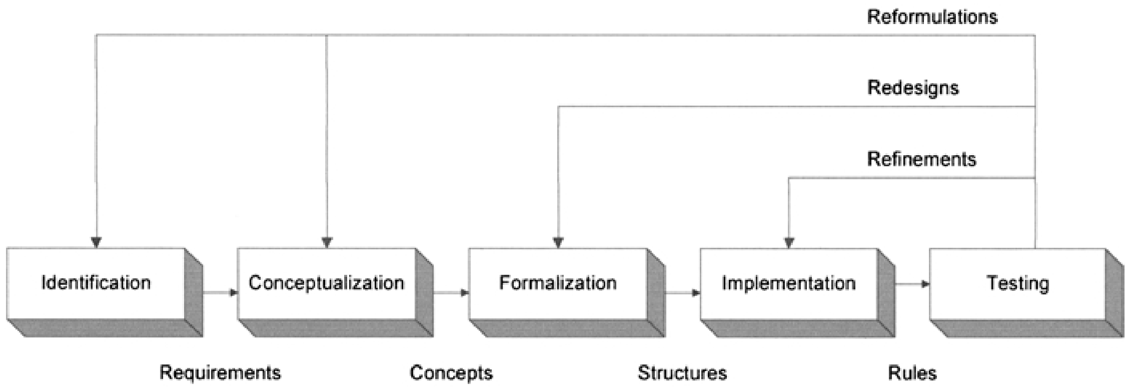
Appendix A: Stage-based approaches

In this section, we describe early approaches to KBS development and lessons learned.

A.1 Buchanan's methodology

A consequence of these early, experimental systems was a great rush of enthusiasm in industry for KBSs, or "expert systems" as they became known. Yet, even with the advent of shells, such as OPS5 and CLIPS, there was little in the way of methodological leadership until a group of researchers at the 1983 Palo Alto workshop on expert systems devised what is referred to as "Buchanan's methodology" (Buchanan, 1983). This methodology is characterised by the five-stage model shown in Figure A.1.

Buchanan's methodology for developing KBSs is based on the waterfall life cycle as advocated by Royce (1970). It is a useful first contribution to the literature in that it identifies several crucial stages that any system is going to pass through. For example, the aim of the first stage is to identify and characterise the important aspects of developing a system of this type, breaking the problem down into four sub-stages: i) participant identification and roles, ii) resource identification, iii) goal identification



Identification:	Identify problem characteristics
Conceptualization:	Find concepts to represent knowledge
Formalization:	Design structure to organize knowledge
Implementation:	Formulate rules to embody knowledge
Testing:	Validate rules to organize knowledge

Figure A.1 Buchanan's methodology

and iv) problem identification (Buchanan, 1983). Thus its use identifies and guides the resolution of the significant problems that face knowledge engineers, e.g. the selection of an appropriate knowledge representation formalism. The methodology also identifies opportunities for the development of systems using a prototyping approach.

Unfortunately, Buchanan's methodology raises more questions than it provides answers for. For example, the methodology fails to: i) indicate how knowledge is to be elicited, ii) what techniques are necessary to analyse the elicited knowledge, iii) how that knowledge is to be represented in a suitable form, iv) how to determine which form is, in fact, most suitable, and v) how the knowledge will be prevented from change in semantic meaning over the stages. Additionally, the methodology fails to differentiate between the stages adequately, describing the whole process of developing the system as knowledge acquisition.

If contrasted against either the twelve requirements in Table 1 (henceforth called Miller's *extended* criteria) or the TRILLIUM$_K$ capability model's criteria for rigour, Buchanan's methodology achieves low scores (see Table 8). This is in part due to the fact that the primary focus of these early methods[7] is the development of the knowledge acquisition component of the life cycle. Thus it is difficult to judge fairly the method as a whole against the quality criteria. However, even with these limitations the methodology provides a worthwhile initial contribution to the literature of KBS development.

A.2 Davis and Lenat's nine-point paradigm

At the same time as Buchanan's method was being developed, Davis and Lenat (1982) presented a general paradigm for KBS development through prototyping. The paradigm is based on the following nine points:

1. *System conception.* In this stage the problem domain is selected in order to undertake a feasibility study of the proposed system. The selection criteria are suitable for symbolic reasoning and a limited vocabulary.

[7] The methodologies attributed to Buchanan (Shortliffe, 1976), Davis & Lenat (1982), Grover (Grover, 1983), and Alexander (Alexander *et al.*, 1986) primarily focus upon the knowledge acquisition component of the lifecycle.

2. *System design*. A knowledge representation formalism is chosen for the problem along with the control architecture and the specification of the user interface.
3. *Knowledge acquisition*. The domain knowledge is extracted from an expert and encoded into the knowledge representation formalism.
4. *Prototype construction*. A basic set of elements is encoded into an evolutionary prototype capable of solving a number of typical problems in the domain.
5. *Prototype testing*. The domain experts are again utilised to help test the prototype and then expand the system to cover a wider range of problems and increase performance.
6. *Formal evaluation*. The performance and acceptance of the system is evaluated within a realistic environment.
7. *Extended use and enhancement*. More extensive testing of the system is performed and the system is enhanced where necessary to accommodate the environment.
8. *Transfer*. When the system has reached an acceptable level of performance, the system is placed in its actual working environment.
9. *Maintenance and documentation*. Since knowledge is often in a continual state of flux, the documentation is probably more important in knowledge-based software than conventional software as maintenance may well be higher.

This methodology differs from Buchanan's in that it is a more forceful proponent of prototyping, the process through which experimental programs are developed to consider options and alternatives within a set of parameters. Prototyping allows the developer to test the feasibility of the proposed system given a minimal amount of time with little resources (Luqi *et al.*, 1998). This is because the systems are developed in a *run–debug–edit* or *construct and test* environment, which allows less than fully developed ideas to be expanded and implemented in a creative manner. The developer must be aware of, and resist, the temptation to perform what Kowalski (1983) describes as "the trial and error approach" to software development. The use of prototyping is not without its place, such as in developing the user interface or experimenting with control architectures. However, the developer should be aware of the problems associated with the validation and verification of KBSs developed through prototyping as well as the difficulty in maintaining the knowledge-based component of such systems.

Like Buchanan's methodology (Buchanan *et al.*, 1983), Davis's approach to system development does provide some useful conceptual guidelines for knowledge engineers but leaves out the execution-related aspects. The approach has a major flaw in that the representational form and the control architecture are selected prior to the knowledge being extracted from the domain expert. This can lead to a poor selection of a representation formalism, which in turn could fail to allow the system to reach its potential. Further, Davis states that the knowledge representation formalism typically used is that of production rules, without justifying why that representation is preferred.

Overall, both Miller's and the TRILLIUM$_K$ requirements criteria indicate that a refined version of the methodology would make a satisfactory general approach to the prototype style of development. However, as it stands it contains deficiencies and is ill suited to non-prototype KBS development.

A.3 A pragmatic knowledge acquisition methodology

A *pragmatic knowledge acquisition methodology* was suggested by Grover (1983), based on the assumption that when access to domain experts is limited, the knowledge engineer can still use development time (both with and without the domain expert) efficiently to construct a valid model of the domain. Grover suggests a three-phase methodology that aims towards the production of a *knowledge acquisition document series*. The three phases he advocates are: i) domain definition, ii) fundamental knowledge, and iii) basal knowledge consideration. These phases are shown in Figure A.2. The result of passing through these three phases is a series of documents that allow users, experts and system designers to possess consistent, organised and up-to-date domain knowledge upon which the system can be based. Grover gives details of the documents he expects for each of the phases. This

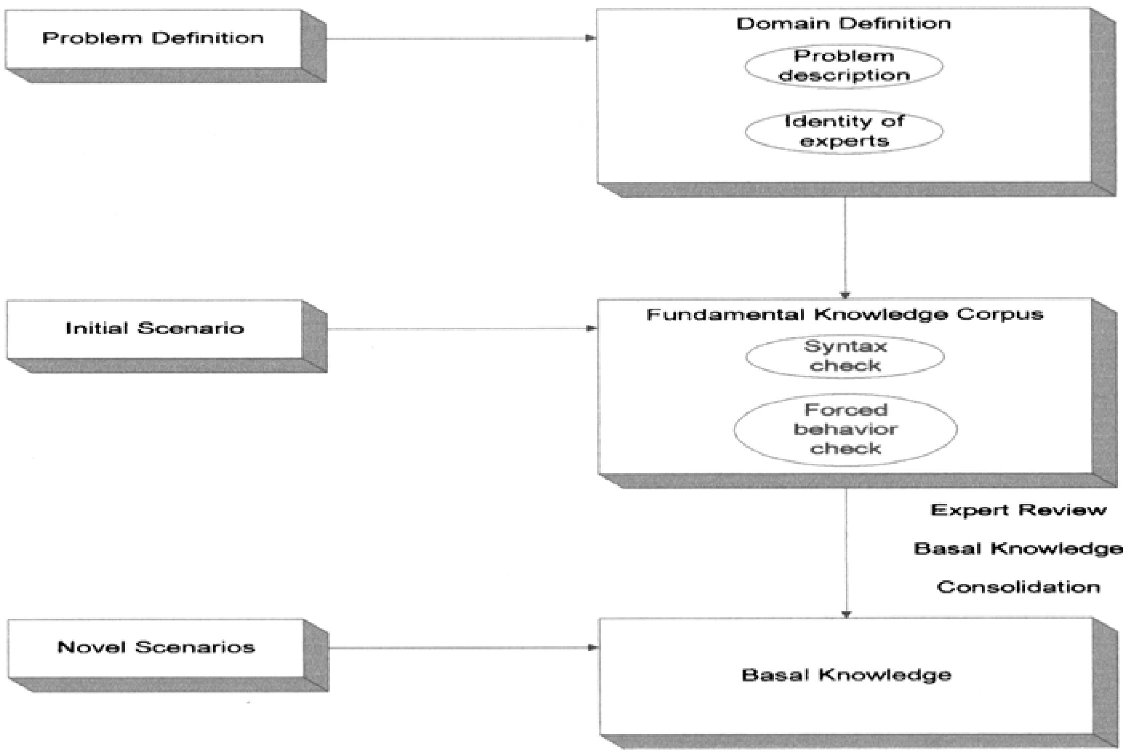


Figure A.2 Grover's methodology

very much ties in with several of Miller's extended criteria.[8] For example, many of the documents that Grover requires are similar to those required under the military standard 2167(A) for system development. Grover's methodology also facilitates changing requirements, stable system development, maintenance and, to some extent, validation and verification.

Grover's approach does not go into detail on reaching the various aspects of the development. It is useful in identifying three stages in a systems life cycle and suggesting points that need to be achieved in order for the stage to be of use (their basal state). The approach would benefit from further detailed development, especially in the areas of representation selection, acquisition and elicitation. The methodology achieves low scores in both Miller's and the TRILLIUM_K requirements criteria.

A.4 Ontological analysis

Growing out of the early stage-based methodologies, Alexander proposes a methodology based upon "ontological analysis" (Alexander, 1986), a variant of Newell's "knowledge level" that aims to produce a formal specification of knowledge elements in a task domain (Newell, 1972). Alexander defines ontology as:

> A collection of abstract objects, relations and transformations that represent the physical entities necessary to accomplish some task.
>
> (Alexander, 1986)

He states that experience has shown the process of constructing a complex ontology to consist of three steps:

1. Analysis of the (static) physical objects and relations.
2. Analysis of the (dynamic) operations that can change the task world.

[8] Again it must be noted that Miller's criteria are intended to be for the whole life cycle and Grover's methodology is primarily a knowledge acquisition methodology.

3. Analysis of the (epistemic) knowledge structures, the selection and use of these operations.

The research of Alexander indicates that several different formal tools are useful for extracting and defining ontologies. Consequently, a family of languages collectively called SPOONS (SPecification of ONtological Structure) is defined to encompass tools based on domain equations, equational logic and semantic grammars. He states that the

> most useful and concise of these languages is SUPESPOONS (SUPErstructure SPOONS), which is based on the domain equations of denotational semantics (Stoy, 1977) and algebraic specification (Guttag, 1977). Because of the rich ontologies found in most knowledge engineering problems, domain equations provide a concise and reasonably abstract characterization of the necessary knowledge structures.
>
> (Alexander, 1986)

The first analysis in building the whole ontology is an analysis at the static ontological level, where the physical objectives in the problem domain and their inherent properties and relations are identified. At this level the analysis performed is quite similar to the entity-relationship model of Chen (1976).

The second analysis, the dynamic ontological analysis, serves two roles: i) it identifies the problem space in terms of configurations of elements (defined in the static ontology) and defines the problem operators that transform the domain of problem states, and ii) it defines which knowledge is unchanged and which knowledge changes as the problem is solved.

The third ontology is the epistemic ontology. While the dynamic ontology defines the operations available to perform a task from a given state, the epistemic ontology defines the knowledge structures that guide the selection and use of these operations. The epistemic ontology contains two types of knowledge structures: one is used to select which operations should be performed and the other controls the actual performance of the operations.

Alexander defines "principles of practice" to aid the application of the ontologies to KBS development. These principles are as follows:

1. Begin with the physical entities, assessing their properties and relationships from there.
2. The static, dynamic and epistemic ontologies are not strict boundaries – use them loosely.
3. Clearly establish the distinction between objects and what they are intended to represent.
4. Understand and separate intentional and extensional entities.
5. Build relative abstractions through the use of generalisation and aggregation.
6. Encode rules as simple associations and heuristic steps as mappings between domains.
7. Ensure the compositionality of elements.

Alexander's methodology is useful for conceptualising knowledge engineering problems. The methodology is advantageous in that it provides an approach backed with theoretical foundations. However, because the method is intended only to give a formal specification of the domain, there is little consideration for developing the specification towards an implementation. Hence it is difficult to use the approach to derive an implementation for a specified domain. A possible option is to combine this methodology with another one, say Buchanan's, to arrive at a more formal development and higher-quality system.

A.5 A line model of development

A model that utilises the concepts of both stages and continuous refinement is proposed by Hilal and Soltan (1993). The model is based upon the principal of "circumstantial occurrences" termed "COC", which defines special circumstances of a project. This philosophy identifies that no two developments are ever the same due to external factors, such as the nature of work practices at an organisation and changes in individual and organisational learning. Thus a project history forms a continuous line of change, where "each point along the line represents the COC of a project. The closer the points are the more similar they are. Such alignment might allow the classification of project's COCs into classes, where each class might have its best suitable strategy" (Hilal & Soltan, 1993). Hilal uses this concept

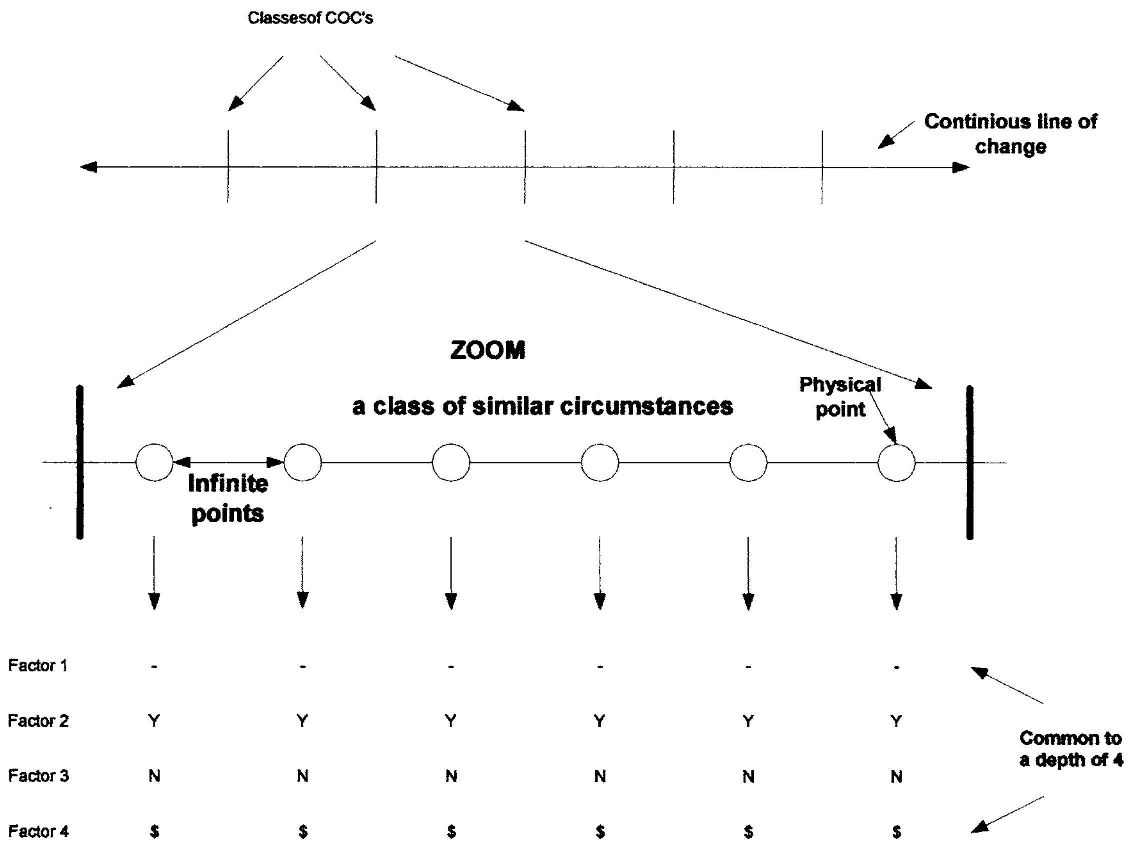


Figure A.3 The COC model's continuous line of change (Hilal & Soltan, 1993)

to develop a second line that allows mappings from the points on the first line to the strategic change classes or options given a change in the circumstance of the development as depicted in Figure A.3.

The concept that Hilal utilises is to give the knowledge engineer a set of options at each point in the development to improve the "state" of the project "with respect to other possible states" (Hilal & Soltan, 1993) along a spectrum of infinite possibilities. The strength of this model is in its philosophical base; however, as the methodology lacks details regarding development from concept to implementation it would be best to adopt this approach with the pragmatism of a more detailed model.

A.6 Other knowledge-based models

Several other researchers have presented methodologies in various areas that are related to knowledge-based systems, including MEDESS (van Weelderen, 1991), for the design of knowledge-based support systems. This methodology determines that an effective expert support system should be created by understanding the current situation and possible alternatives (similar to Hilal). The design process is considered from three perspectives: the experts', the organisational and the interorganisational. These perspectives are also considered on three levels: a skill-based level, a rule-based level and a knowledge-based level. Four problem-solving drivers are considered: the "what", the "why", the "how" and the "with what", through which the information is elicited and transformed into a model management system (van Weelderen, 1993).

A multi-aspect 'ideographic' model (a knowledge map for knowledge) is primarily related to the knowledge acquisition portion of KBS development (Wainwrite, 2001). The knowledge map can be thought of as a three-dimensional "molecular structure" that describes the characteristics of an entity *E*. The mapping uses a similar approach to vanWeelderen in that it considers the following:

1. "why E" experiences,
2. "what does it feel like to be E" classifications,
3. the distinctive characteristics "which make E what it is",
4. "how E" fits together, and
5. events "when E does something".

Wainwrite's methodology examines internal and external interdependencies, develops a taxonomic approach to knowledge types, uses time-structuring associations and considers the explanatory issues associated with knowledge use.

The research literature also includes the results of two early research-council-funded projects.

- a "People Oriented Methodology for Expert Systems" (POMES) that was the result of the British government's Alvey AI initiative of the 1980s (Daiper, 1987; 1988) and
- an early ESPRIT project (1098), "Methodology for IKBS systems" (Hayward, 1987), which developed a methodology very similar to the stage-based model of Buchanan.

References

Akkermans, JM, Schreiber, AT and Wielinga, BJ, 1994, "Steps in constructing problem solving methods" *Proceedings of the 8th Banff Knowledge Acquisition for KBS Workshop, Volume 2: Shareable and Reusable Problem Solving Methods* 29-1-29-21.

Alexander, JH, Freiling, MJ, Shulman, SJ, Staley, JL, Rehfuss, S and Messick, SL, 1986, "Knowledge level engineering: ontological analysis" *AAAI 5* 963–968.

ANSI, 1992, *Life Cycle Development of Knowledge-Based Systems Using DoD-Std 2167A* ANSI/AIAA G-031–1992.

Antoniou, G and Sperschneider, V, 1994, "On the verification of modular knowledge bases" *EUROVAV '93: Proceedings of the European Symposium on the Validation and Verification of Knowledge-Based Systems* 117–129.

Antoniou, G and Plant, R, 1997, *AAAI Workshop Notes on Validation and Verification, Fifteenth National Conference on AI.*

Ayel, M and Rousset, M-C, 1995, *EUROVAV-95: European Symposium on the Validation of KBS.*

Becker, LA, Green. PF and Bhatnagar, J, 1989, "Evidence flow graph methods for validation and verification of expert systems" Nasa Contractor Report 181810, Langley Research Center, VA.

Boehm, B, 1988, "A spiral model of software development and enhancement" *IEEE Computer* **21**(5) 61–72.

Boehm, B and Hansen, W, 2001, "The Spiral Model as a Tool for Evolutionary Acquisition," Cross Talk, May 2001.

Booch, G, Jacobson, I, Rumbaugh, J, 1998, *The Unified Modeling Language User Guide* Addison-Wesley.

Breuker, J and Van de Velde, W (eds), 1994, *Expertise Model Document Part II: The CommonKADS Library* ESPRIT Project P5248 KADS-II/I/VUB/TR/054/3.0.

Breuker, JA, Wielinga, BJ, Van Someren, M, de Hoog, R, Schriber. AT, de Greef, P, Bredweg., B, Wielmaker, J, Billault, JP, Davoodo, M and Hayward, SA, 1987, *Model Driven Knowledge Acquisition: Interpretation Models* ESPRIT Project P1098 Deliverable D1 (task A1), University of Amsterdam and STL Ltd.

Buchanan, BG, Barstow, D, Bechtal, R, Bennett. J, Clancy, C, Kulikowski. C, Mitchell. T and Waterman, 1983, "Constructing an expert system" in F Hayes-Roth, DA Waterman and DG Lenart (eds) *Building Expert Systems* Addison-Wesley.

Cardenosa, J, 1994, *EUROVAV '93: Proceedings of the European Symposium on the Validation and Verification of Knowledge-Based Systems.*

Carpenter, CL and Murine, GE, 1984, "Measuring software product quality" *Quality Progress* May 1984 Vol 7(5), pp. 16–20.

Chen, P, 1976, "The entity relationship model – towards a unified view of data", *ACM Trans. Database Systems* **1**(1) 9–36.

Cheng, B and Jamieson, R, 1996, "A software engineering view on expert systems quality & evaluation criteria" *Workshop Notes, Verification, Validation and refinement of Knowledge-based Systems, at the Pacific Rim International Conference on Artificial Intelligence, Cairnes, Australia* 17–26.

Coenen, FP, Bench-Capon, T, Boswell, R, Dibie-Barthelemy, J, Eaglestone, B, Gerrits, R, Gregoire, E, Ligeza, A Laita, L, Owoc, M, Sellini, F, Spreeuwenberg, S, Vanthienen, J, Vermesan, A and Wiratunga, N (2000). "Validation and verification of knowledge-based systems: report on EUROVAV99" *Knowledge Engineering Review* **5**(2) 187–196.

CSC, 1989a, "Expert system development methodology reference manual" Computer Sciences Corporation, System Sciences Division, Report CSC/TM-88/6148.

CSC, 1989b, “Expert system development methodology standard” Computer Sciences Corporation, System Sciences Division, Report CSC/TM-88/6147.

CSC, 1989c, “Expert system development methodology user guide” Computer Sciences Corporation, System Sciences Division, Report CSC/TM-88/6146.

Culbert, C, 1990, *AAAI-90 Workshop Notes on Knowledge-Based Systems Verification Validation and Testing*.

Davis, R and Lenat, D, 1982, *Knowledge-Based Systems in AI* McGraw-Hill.

de Hoog, R, Martil, R, Wielinga, Taylor, R, Bright, C and van de Velde, W, 1994a, *The Common KADS Model Set*, ESPRIT Project P5248 KADS-II/DM1.1b/UvA/018/6.0/FINAL.

de Hoog, R, Benus, B, Metselaar, C, Vogler, A and Menezes, W, 1994, *Organisational Model: Model Definition Document* ESPRIT Project P5248 KADS-II/M6/UvA/041/3.0/June 1994b.

Diaper, D, 1987, “POMESS: a people orientated methodology for expert system specification” *Proceedings of the First European Workshop on Knowledge Acquisition for Knowledge-Based Systems* pp. D4.1–14.

Duursma, C, Olsson, O and Sundin, U, 1993, *Task Model definition and Task Analysis Process*, ESPRIT Project P5248 KADS-II/M5/VUB/RR/004/2.0.

Fikes, R, Farquhar, A, Rice, J, 1997, “Tools for assembling modular ontologies in ontolingua” *Knowledge Systems Laboratory* April 1997 Technical report KSL-97-03.

Gaines, BR, Shaw, MLG and Linster, M, 1992, “Managing quality assurance in integrated knowledge acquisition and performance systems” *AAAI-Workshop Notes, Verification and Validation of Expert Systems* San Jose, CA, 16 July, 1992.

Gaines, BR, 1996, “Fundamentals of validation” *AAAI-96 Workshop Notes on Verification and Verification of Knowledge-Based Systems* 1–6.

Gamble, R and Landauer, C, 1995, *Workshop Notes, Verification and Validation of Knowledge-Based Systems, Fourteenth IJCAI*.

Gamble, RF, Stiger, PR and Plant, RT, 1999, “Rule-based system formalized within a software architectural style” *Knowledge Based System Journal* **12**(1) 13–26.

Genesereth, M, 1991 “Knowledge interchange format: principles of knowledge representation and reasoning” *Proceedings of the Second International Conference (on what)* 599–600.

Ghezzi, C, Jazayeri, M and Mandrioli, D, 1991, *Fundamentals of Software Engineering* Prentice Hall.

Grover, MD, 1983, “A pragmatic knowledge acquisition methodology” *IJCAI 8* 436–438.

Guttag, JV, 1977, “Algebraic data types and the development of data structures” *CACM* **20**(6) 396–404.

Hilal, DK and Soltan, H, 1991, “A suggested descriptive framework for the comparison of knowledge-based systems methodologies” *Expert Systems* **8**(2) 107–114.

Hilal, DK and Soltan, H, 1993, “Towards a comprehensive methodology for KBS Development” *Expert Systems* **10**(2) 75–91.

Hayward, S, 1987, “Methodology for IKBS system” Research Report: User Centred Systems, STC Technology Ltd, Harlow, Essex, Great Britain.

Howard, GS, Bodnovich, T, Janiciki, T, Liegle, J, Klein, S, Albert, P and Cannon, D, 1999, “The efficacy of matching information systems development methodologies with application characteristics – an empirical study” *The Journal of Systems and Software* **45** 177–195.

Humphrey, W, 1989, *Managing the Software Process* Addison-Wesley.

Ibrahim L, Deloney R, Gantzer D, LaBruyere L, Laws B, Malpass P, Marciniak J, Reed N, Ridgeway R, Scott, A and Sheard S, 1997, *The Federal Aviation Administration Integrated Capability Maturity Models, Version 1.0, An Integrated Capability Maturity Model for the Acquisition of Software Intensive Systems* The Federal Aviation Administration, Washington, DC.

Jacobson, I, Booch, G and Rumbaugh, J, 1999, *The Unified Software Development Process* Addison-Wesley.

Johnson, SC, 1988, “Validation of highly reliable real-time knowledge-based systems” *SOAR 88 Workshop on Automation and Robotics* Dayton, Ohio, July 20–23, 1988 pp. 16–22.

Jones, CB, 1980, *Software Development – A Rigorous Approach* Prentice Hall.

Kingston, J, 1992, “Pragmatic KADS: a methodological approach to a small knowledge-based systems project” *International Journal of Knowledge Engineering* **9**(4) 171–180.

Kingston, J, 1993, “KBS methodology as a framework for co-operative working” Artificial Intelligence Applications Instute, University of Edinburgh, AIAI-TR-130.

Kowalski, RA, 1983, “Logic for expert systems” *Expert Systems 83: Proceedings of BCS SGES Conference* 133–145.

Le Goc, K, Frydman, C and Torres, L, 2002, “Verification and validation of the SACHEM conceptual model” *International Journal of Human-Computer Studies* **56**(2) 199–223.

Lenat, DB, 1995, “Cyc: a large-scale investment in knowledge infrastructure” *Communications of the ACM 38* 11 30–49.

Lindsay, R, Buchanan, BG, Feigenbaum, EA and Lederberg, J, 1980, *DENDRAL* McGraw-Hill.

Livson, BU, 1988, “A practical approach to software quality assurance Livson” *ACM SIGSOFT Software Engineering Notes* **13**(3) 45–48.

Luqi, C, Chang K and Hong Zhu, 1998, "Specifications in software prototyping" *Journal of Systems and Software* **42**(2) 125–140.

Miller, L, 1990, "A realistic industrial strength life cycle model for knowledge-based system development and testing" *AAAI Workshop Notes: Validation and Verification.*

Mitev, NN, 1994, "The business failure of knowledge-based systems: linking knowledge-based systems and information systems methodologies for strategic planning" *Journal of Information technology* **9** 173–184.

Morell, LJ, 1989, "Use of metaknowledge in the verification of knowledge-based systems" NASA Contractor Report 181821. Langley Research Center, VA.

Murrell, S and Plant, R, "A survey of tools for validation and verification 1985–1995" *Decision Support Systems* **21**(4) 307–323.

Nadhakumar, J and Avison, DE, 1999, "The fiction of methodological development: a field study of information systems development" *Information Technology and People* **12**(2) 176–189.

Newell, A, 1972, "The knowledge level" *Artificial Intelligence* **18** 87–127.

O'Leary, DE and Preece, A, 1998, *AAAI Workshop Notes on Validation and Verification, Sixteenth National Conference on AI.*

Orsvarn, K, 1993, "Technical diagnosis by top-down fault location" ESPRIT Project P5248 KADS-II/T1.3/WP/SICS/003/0.5, SICS.

Price, S and Kingston, J, 1993, "The KADESS Knowledge-Based System: Employing the KADS methodology in an engineering application" *Sixth International Conference on Industrial and Engineering Applications of AI and Expert Systems* 188–196.

Plant, RT, 1987 "A methodology for knowledge acquisition in the development of expert systems" Ph.D. thesis, Department of Computer Science, University of Liverpool, England.

Plant, RT, 1991, "Utilising formal specifications in the development of knowledge-based systems" in D Partridge (ed.) *Artificial Intelligence and Software Engineering* Ablex Press.

Plant, RT, 1994, *AAAI Workshop Notes on Validation and Verification, Twelfth National Conference on AI.*

Plant, RT and Gamble, R, 1997, "A multilevel framework for KBS development" *International Journal of Human-Computer Studies* **46** 523–547.

Preece, A, 1993, *AAAI Workshop Notes on Validation and Verification, Eleventh National Conference on AI.*

Preece, AD, 1995, "Towards a quality assessment framework for knowledge-based systems" *Journal of Systems and Software* **29** 219–234.

Preece, AD, 2002, "A history of verification and validation references" available at `http://www.csd.abdn.ac.uk/~apreece/Research/vavpage.html`.

Ribeiro Justo, G R, Vekariya, P, Delaitre, T, Zemerly, J and Winter, S, 1997, "Prototype-oriented development of high-performance systems" *Proceedings of the 2nd International Workshop on Software Engineering for Parallel and Distributed Systems (PDSE'97)* IEEE CS Press, May 17–18, 1997, pp. 74–83.

Robinson, B, 2002, "All systems go for nasa tech" *Federal Computer Week* available at `http://www.fcw.com/fcw/articles/2002/0218/tec-nasa-02-18-02.asp`.

Royce, WW, 1970 "Managing the development of large software systems" *Proceedings of IEEE WESTCON* 1–9.

Rushby, J, 1988, "Quality measures and assurance for AI software" NASA Contractor Report 4187. Scientific and Technical Information Division.

Schreiber, AT, Terpstra, P, Magni, P and Van Velzen., M, 1994, "Analysing and implementing VT using CommonKADS" *Proceedings of the 8th Banff Knowledge Acquisition for KBS Workshop, Volume 3: "Sisyphus II – VT Elevator Design Problem* pp 44-1–44-29.

Schreiber, G, Akkermans, H, Aniewierden, A, de Hoog, R, Shadbolt., N, Van de Velde, W and Weilinger, B, 1999, *Knowledge Engineering and Management: The CommonKADS Methodology* MIT Press.

Schmolze, J and Vermesan, A, 1996, *AAAI Workshop Notes on Validation and Verification, Thirteenth National Conference on AI.*

Shaw, M and Garlan, D, 1996, *Software Architecture: An Emerging Discipline* Prentice Hall.

Shortliffe, EH, 1976, *Computer-Based Medical Consultations: MYCIN* North Holland.

Stokes, M (ed.), 2001, *Managing Engineering Knowledge, MOKA: Methodology for Knowledge Based Engineering Applications* ASME Press.

Stoy, JE, 1977, *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory* MIT Press.

TRILLIUM, 1992, "TRILLIUM: Telecom Software Product Development Capability Assessment Model" Bell Canada Quality, Technical Report Draft 2.2.

Van de Velde, Duursma, C, Schreiber, G, Terpstra, P, Schrooten, R, Golfinopoulos, V, Olsson, O, Sundin, U and Gustavsson, M, 1993, "Design model and process" ESPRIT Project P5248 KADS-II/M7/VUB/RR/064/2.0.

Van Weelderen, JA, 1991, "MEDESS: a methodology for designing expert support systems" Ph.D. thesis, Delft University of Technology, Delft, The Netherlands.

Van Weelderen, JA and Sol. HG, 1993, "MEDESS: *A Methodology for Designing Expert Support Systems" Interfaces* **23**(3) pp 51–61.

Waern, A, 1993, "The communication model" ESPRIT Project P5248 KADS II/M3/SICS/RR/002/0.2.

Waern, A, 1993a, "The CommonKADS agent model" ESPRIT Project P5248 KADS II/M4/TR/SICS/001/V1.0.

Wainwrite, C, 2001, "Knowledge management" *Management Services* **45**(11) 16–19.

Wasserman, AI and Freman, P, 1983, "Characteristics of software development methodologies" in TW Olle, HG Sol and CJ Tully (eds) *Information Systems Design Methodologies: A Feature Analysis* North Holland.

Weitzel, JR and Kershberg, L, 1989, "Developing knowledge-based systems: reorganizing the system development life cycle" *Communications of the ACM* **32**(4) 482–490.

Wells, S, 1994, "CommonKADS development method guidelines volume one: the commonkads life-cycle model (LCM)" ESPRIT Project P5248 KADS-II/TR/M10a/LR/0109/1.0.

Wielinga, B, van de Velde, W, Schreiber, G and Akkermans H (eds), 1992, "Towards a unification of knowledge modeling approaches" ESPRIT Project KADS-II/T1.1/UvA/RR/004/4.0.

Wielinga, B, Van de Velde, W, Schreiber, G and Akkermans, H, 1993a, "Expertise model definition document" ESPRIT Project P5248 KADS-II/M2/UvA/026/1.1.

Wielinga, BJ, Van de Velde, W. Schreiber, A and Akkermans. JM, 1993b, "Towards a unification of knowledge modeling approaches" in Jean-Marc David, Jean-Paul Krivine and Reid Simmons (eds) *Second Generation Expert Systems* Springer-Verlag.

Yen, J and Lee, J, 1993, "A task-based methodology for specifying expert systems" *IEEE Intelligent Systems* **8**(1) 8–15.