The Knowledge Engineering Review, No. 4: 3, 1989, 189–204

Life cycles in software and knowledge engineering: a comparative review

MICHAEL WILSON*, DAVID DUCE* AND DAN SIMPSON**

**Informatics Dept, Rutherford Appleton Laboratory, Chilton, Didcot, Oxon OX11 0QX, UK*
***Dept of Computing, Brighton Polytechnic*

Abstract

Progress in software engineering has led to system development following models of the system life cycle. These models incorporate the use of prototyping and formal methods of program verification. They are becoming supported by integrated project support environments and permit the planning and monitoring of software development projects.

In contrast, knowledge based systems (KBS) are developed using informal views of the system life cycle. Tools have been developed to support some stages of the life cycle in an undisciplined manner. The commercial use of KBS needs development projects to be planned and monitored. This requires methods and tools based on systematic life cycle models to be established for KBS.

This paper reviews the current state of life cycle approaches to software engineering and KBS development projects in order to provide a direction for the development of methodical KBS life cycle models.

1 Overview and introduction

KBS have now penetrated the commercial market not only in that tools are being bought to enable industry to explore their potential, but also to develop practical systems which are now being used. Knowledge engineering and the production of knowledge based systems (KBS) are now practical commercial activities. To continue the commercial application of KBS it is necessary to establish methods for managing and conducting development projects so that resources, time and effort can be estimated, and quality can be monitored within a contractual framework. There are methods available for the development of conventional software systems which address these issues based upon life cycles of the software. A software product life cycle is a description of the stages through which the software passes from the initial identification of a need, through design and implementation, to final obsolescence. Alternative life cycle descriptions identify different stages and emphasise different decision points during product development. If a project can be divided into stages, then deliverables can be described for each stage rather than just for the whole. Consequently projects can be monitored and evaluated at points other than the end point. Life cycle based development methods describe how the software development can be performed and managed at each stage of the life cycle. Over the past 20 years it has become accepted in the software engineering community that software development should be undertaken using a life cycle based method. The benefits of such an approach include:

- The ability to plan the project
- The ability to estimate resource requirements for the development
- The ability to size the likely software product
- The ability to estimate hardware requirements
- The ability to update estimates on the basis of real figures during monitoring
- The availability of documents for monitoring and control

- The ability to fit the development process into a quality management system
- A development structure which may be audited for quality and cost

The result of using life cycle approaches is that the development process is made visible to the project management, project controller, quality controller, the project sponsor and users of the system. The major benefit is that potential problems can be caught early within the development with substantial savings in effort and resources used in rework and correction.

The traditional life cycle model used in software engineering has provided some of the above advantages, although it has been shown not to apply to all styles of product development; particularly where the program requirements are initially ill-specified. Since this is nearly always the case in KBS developments, the life cycles used in conventional software engineering cannot be used without modification for KBS development. However, life cycle based approaches will have to be developed for KBS development in order to provide the benefits listed above, and allow commercial KBS development to become commonplace.

Many knowledge engineers involved in KBS development do not have a software engineering background and are unfamiliar with conventional software development. Therefore, this paper introduces many issues which arise in software development projects which may be unfamiliar to knowledge engineers, by reviewing the state-of-the-art in life cycle based development methods in software engineering and KBS. It is hoped that this will enable knowledge engineers to consider more easily the best structure for KBS development methods. The paper consists of three major sections. Section 1 describes an inventory of the current best practice in KBS development outlining the stages and the decision points which appear to have been identified in previous projects. Section 2 reviews life cycle based development methods in software engineering which could be used as guides for a life cycle based KBS development method. The third section summarizes three recent projects which have suggested KBS development methods based on software engineering practice.

It is, of course, not possible in a paper of this length to consider methods for the whole of artificial intelligence. Similarly, although the development approaches described and issues raised may be applicable to the whole range of KBS developments, the examples cited mostly arise from expert systems development. This is because the area of expert systems is the most commercially advanced sector of KBS production, and the limitations applied there make it most immediately amenable to the introduction of agreed development methods. However, where possible, the issues raised should be considered for the full breadth of KBS developments.

Despite these limitations, the area covered in this paper remains very large. Consequently it is not possible to describe in depth each issue raised. Therefore, many topics are merely highlighted for consideration along with a reference where further details may be found. In particular, it is assumed that most readers will be familiar with the description of the stages of development of most current KBS, and description is only given in order to remind readers of the lack of consensus on the structure of KBS development; rather than describing it in detail to those unfamiliar with it. Many aspects of the structure of software development have been addressed by software engineers, and in reviewing the current state of software engineering methods, previously identified resolutions may be seen. Some of these are incorporated in the KBS development methods based on software engineering principles outlined in section 3. However, in places these appear to have adopted software engineering principles at the cost of KBS ones. Further progress is still required to produce KBS life cycle based development methods, and this paper is intended to offer a context from which knowledge engineers may make it.

2 Current commercial practice and KBS life cycles

Some individual organizations have now accumulated considerable experience in the development of KBS, however, there is no consensus in industrial practice on a development method for them.

Most KBS currently developed are stand-alone expert systems that ask users questions and reason through a hierarchical classification in order to explain solutions. The problems addressed

involve the diagnosis of symptoms or fault finding. A majority of these systems are small and are based as much on the regulations and theory that describe how decisions should be reached, as on human expertise. The development of these systems in industrial practice is led by a colloquial life cycle model based on early experience with KBS development which has been outlined, but not formalized (e.g. Hayes-Roth, Waterman and Lenat, 1983; Frieling et al. 1985). This generally follows the seven stages described below. Since the role of prototyping is contentious and also one of the features that distinguishes KBS development from conventional software development, it is mentioned in each of the stages at which it could be applied.

2.a Problem description

This usually divides into three subparts:

a. The identification of problems where the use of KBS within an organization will provide the leverage to overcome them cost-effectively.
b. Initial problem analysis. A description of the problem will be assessed as to whether it really is amenable to a KBS solution (e.g. after Prerau, 1985).
c. Description of problem requirements. The intended role of the system and the requirements it should meet are identified and a user model is established to guide future development.

For larger problems there may be an embedded version of the whole life cycle here, applied to some subset of the problem as a feasibility study. In practice, prototypes are often produced at this early stage to present clients with concrete evidence of the capabilities of KBS. Consequently, a sequence of successive prototypes is often developed.

2.b Knowledge acquisition

Knowledge acquisition is not a clear stage since it is involved in the problem requirement description and continues through later stages. A commonly used definition of knowledge acquisition is (cited by Anjewierden, 1987):

"Knowledge acquisition consists of the elicitation and interpretation of data on the functioning of expertise in some domain, in order to design, build, extend, adapt or modify a knowledge based (expert) system (KBS). In this view, knowledge acquisition is a permanent and crucial activity throughout all stages of designing, implementing and maintaining an expert system."

Having located domain experts with justifiable expertise, the knowledge acquisition may involve either a single or multiple experts. It is argued in some cases that a single domain expert cannot provide knowledge over the whole range of the problem, so the knowledge from several experts should be combined. The counter argument is also made: that the knowledge acquired from multiple experts cannot be unified into a cognitive model of the class from which KBS are derived; therefore only single experts should be used (see McGraw and Seale, 1988 for a review of the issues motivating the choice of how many experts to use).

There are many techniques available for knowledge acquisition, including: interviews, case description, critical incident description, performance analysis, protocol analysis, twenty questions, laddered grids, card sorting, multidimensional scaling, repertory grid, machine induction etc. This battery of techniques and the arguments for using them in different circumstances have recently been thoroughly reviewed by Neale (1988). In general, performance analysis, induction or psychological distance measures are normally used for fast manual or skilled tasks where experts can rarely describe their knowledge adequately. Where knowledge can be described, interviews are the most commonly used technique and the psychological distance measures based on concept sorting or association are rarely applied. This is not because the distance measures are normally inappropriate, but because of the lack of credibility experts attribute to card sorting and laddering tasks, and the amount of the expert's time required to perform them. However, tools to reduce the time required

are commercially available to support the psychological distance measure of repertory grid elicitation and analysis (e.g. RepGrid, see Shaw and Gaines, 1987), although these have not yet become standard tools. Similarly, the analysis of transcribed verbal protocols or interviews can be eased by research tools such as KEATS (Motta et al. 1986; Rajan et al. 1989). However, these are not easily available for commercial development projects yet, and are of no help in reducing the considerable time required by knowledge engineers to analyze videotapes of performance.

The knowledge acquired must be fed back to the domain expert so that it can be validated. For this to happen, it must be represented in some form. The type of representation chosen will interact with the structure of the life cycle. Firstly, the knowledge acquired may be presented in the form of a paper knowledge base intermediate between the expert's representation and that in the implementation vehicle. This offers the possible advantage of using a representation familiar to the domain expert and appropriate to the domain knowledge structure, thus saving the expert effort in learning a new representation, for example, a standard software, design or fault representation may be selected (see Peters, 1980 for a review). If no domain specific representation is available, a specifically designed mediating representation (see Young, 1987 for a review) could be used which is easily learned. The use of a paper knowledge base also prevents the knowledge engineer from making inappropriate implementational trade-offs at this stage of the development. Although paper knowledge bases aid in the validation of the static knowledge, only very simple inference strategies can be followed on them.

The alternative knowledge representation the expert can be presented with in order to validate it, is through a prototype. If a prototype is used, there will be an iteration between this stage and the next with the result that the representation chosen will probably follow through to implementation. If a paper knowledge base is used, several different representations may be used through the life cycle where each has advantages.

There is no formalized method used to select techniques appropriate to particular problem and knowledge types. The usual motivation for the choice of technique is familiarity to a member of the project team, or the constraints of the software tools available. Unless the representation chosen is solely used for mediating, it will result from a trade-off between expressive power and computational tractability. The representation chosen should have the power to express the problem as understood, however, as expressive power increases, computational tractability reduces (see Williams and Lambert, 1988 for a review of this trade-off).

2.c Prototype design and development

Acquired knowledge may be presented to the expert in the form of a prototype system. Unlike an intermediate representation, this will allow the expert to check the inference processes working with the knowledge quite easily, and both to correct errors and to identify and correct omissions. However, a knowledge engineer will be limited to using representations available in the prototyping tool, none of which may be compatible with the natural representation of the domain knowledge. Such tools usually employ general knowledge representation techniques such as frames, production rules, logic or semantic nets (see Cercone and McCalla, 1987 for a review) which the domain expert is required to understand if the prototype is to be modified successfully. This may involve considerable learning effort.

If a prototype is not used to validate knowledge, one might still be developed. There are two reasons proposed for the development of prototypes in the KBS development cycle:

1 The user requirements cannot be specified adequately at the outset.
2 The technical aspects of system design or implementation cannot be specified adequately.

These arguments apply more to KBS development than to conventional software development since KBS requirements specifications are often non-existent, imprecise, or rapidly changing. For a summary of the issues concerning whether prototypes should be discarded after being transformed into

paper specifications, incrementally developed into the deliverable implementation or iteratively developed into the deliverable implementation.

Whichever style of mediating representation is used for knowledge elicitation, a final decision will have to be made at this stage about both the representation and the inference strategy to be used in this and later stages of development.

2.d Knowledge base implementation and refinement

When the knowledge acquired, and the specification, have reached an acceptable level of validity and completeness, the full system is developed. The decision must be made here as to what the delivery vehicle for the system will be. This may be a progressive development of a prototype or a completely new design and implementation. Often the delivery vehicle for this implementation will be different from that used for the prototype. If prototypes are developed on workstations using AI toolkits, they will frequently be redesigned and recoded in conventional languages to be used on cheaper and more generally available delivery vehicles.

There is usually iteration between this stage and the next as there was between stages 2.b and 2.c.

2.e Testing

A partially implemented system will often be tested in the target environment or an environment simulating it in order to assess how it meets the requirements. A problem often found at this stage is to establish a database of cases against which to test the developing system. This may cause the developers to run the KBS in tandem with a present system involving human expertise in order to compare the results and verify the KBS.

The testing procedure is usually performed informally, often by the expert who acts as a knowledge source using the system and either notes down disagreements with the system or makes changes to it directly. In those cases where more thorough testing is performed, (e.g. Wyatt, 1987) changes are often required to the delivery vehicle to meet the user's requirements.

2.f Installation and system integration

Many large KBS will use pre-existing software to provide them with data. They may be linked to large mainframe systems from which much of the information they require can be retrieved. This can require considerable integration into existing systems particularly if the whole resulting system runs under time constraints. All KBS will have to be integrated into a pre-existing organizational environment. This will include not only educating users in the use of KBS and the modification of existing organizational systems to incorporate them, but also the establishment of procedures to maintain the KBS.

2.g Use and maintenance

A KBS can be maintained either by the customer/user or by the developer. Since the knowledge can change frequently and quickly, the KBS must change quickly too. For example, a KBS in the financial domain may have to change within hours of a national budget. Many conventional databases are put under the same requirement, though the problem here is comparatively easy. For the KBS as for the conventional databases, the solution is not for the customer to have a lifetime maintenance contract which would make them dependent on a developing company that may stop trading. The customers must be able to maintain the KBS themselves. Therefore, they must understand the knowledge base structure and the inference procedures applied. The problem which has been identified with many AI tookits is that users find it difficult to develop a consistent model of them. This is likely to be particularly so for domain experts involved only in maintaining systems.

2.h General features of the current commercial KBS life cycle

This brief description of a life cycle does not describe how all systems have been developed, or specifically how many research tools could be used, but it is consistent with commercial practice. Commercial KBS developers are aware of the need for documentation in order to monitor the progress of projects, but since there is no accepted set of documenting points, none have been included here.

Tools are used where available, but tool support has been provided on a stand alone basis motivated by local difficulties with the production of KBS. Existing tools (e.g. shells and AI toolkits; see Laurent et al. 1986 for a review) only help the knowledge engineer in actually coding a prototype or the implementation of a KBS; there are very few tools to support other parts of the cycle. Further, these tools only aid the actual coding task itself and do not offer control structures for code version management or documentation control of the type used in conventional software development projects. Thus it is difficult to select appropriate tools for different phases in the development process because of compatibility problems between the output provided by one tool and the input required by the next. Equally, since the tools have not been designed to be used as alternatives there are few grounds on which to choose those appropriate to a particular project.

The life cycle illustrates where research effort has been devoted in the past (e.g. development tools, knowledge acquisition techniques). However, there still remain many sections of the life cycle which are presently addressed in unsystematic ways based on the undocumented experience of the individuals involved. For example, those with experience in developing expert systems do make planning decisions about cost, effort and time, but no large scale studies have been performed to provide general formulae for these as they have in conventional programming (e.g. Boehm, 1981). To overcome this lack of data for estimating, one managerial technique which has been used is that of staged contracting, whereby a customer only agrees to the next stage in the development rather than committing himself to the full development cycle at any point.

A modified form of this informal stereotype life cycle has been used for systems which are larger than the usual, based on pre-existing knowledge bases, incorporated into existing software, or used to develop embedded or real-time systems, but there are only examples of the use of such modified life cycles and no formal description of them.

There are three major sources of information which can be drawn on in order to change present KBS life cycle management. These are, firstly, the careful appraisal of both successful and unsuccessful large scale KBS projects. Although there are many accounts of successful small KBS and some of large KBS, few accounts of unsuccessful KBS projects are available. Secondly, there are methods used in conventional software engineering which can be modified for the KBS development cycle. Thirdly, the output of research projects on the KBS life cycle or parts of it, which, if they scale up to large systems, can be incorporated into a life cycle oriented view.

3 Software engineering research and life cycles

This section describes the types of life cycles used in conventional software engineering; the development approaches used with them, and the management techniques that can be applied to them. The divisions in the life cycle outlined here, the increasingly formal methods of monitoring development, and the approach to providing tooling for these, cannot be applied to KBS development as they are, but they do offer guidance on how to structure KBS development projects.

3.a An early software development life cycle

The first life cycle model to have an impact on conventional software development was the waterfall model of the software development life cycle which is shown in Figure 1. It was recognized in the waterfall model that as a system progresses through the life cycle it becomes more complete and complex. Consequently, changes at later stages are more costly than those at early stages, with

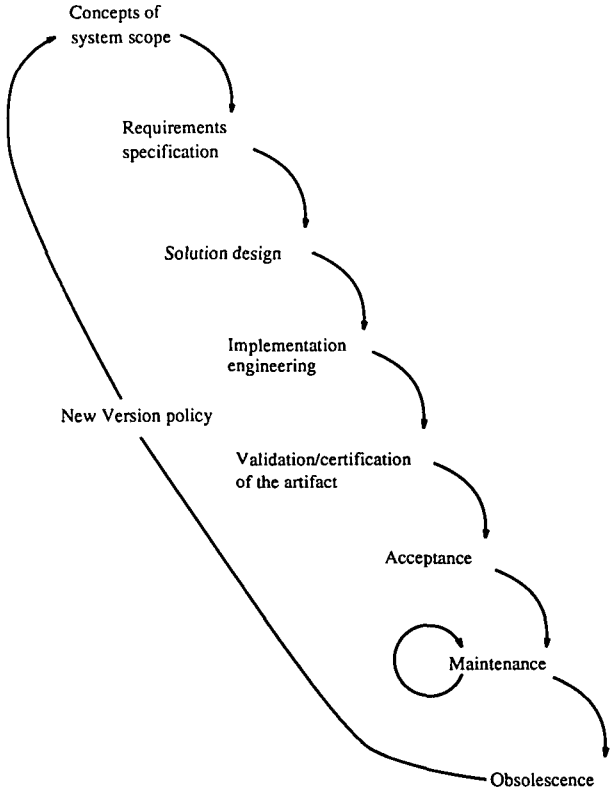

Figure 1 The classical waterfall software life cycle (after Macro and Buxton, 1987).

changes in the maintenance phase being extremely expensive compared to those in the requirements specification stage. Each stage in the development process is meant to produce a deliverable which can be verified against previous stages and validated against the client's requirements (the process of verification and validation is often referred to as "V&V"). This structuring has permitted the development of methods and tools to support development which aid both management and developers in estimating time, cost and manpower requirements through a project's life (e.g. Boehm, 1981).

In the waterfall model the development process is seen as proceeding sequentially through the identified phases, with each being completed before the next is started, thereby providing no easy possibility for the modification of work done at an earlier stage. In addition, single V&V techniques applied to developments using this life cycle may only find 30–70 % of the defects in a program (Myers, 1979).

3.b Recent software life cycle models

The waterfall model has been recently supplanted in software engineering by life cycle models (for example, that illustrated in Figure 2) in which the process is seen as a dynamic one in which exploratory or prototyping work into later stages in the development process is done to provide essential feedback to the current stage.

The functional role of the design documents, particularly the specification, in these life cycles should be noted. At each stage the documents provide the customer/client and the developer with a quality controllable and contractually binding view of the product being produced. This approach begs the question of whether the customer can specify what is required and whether the documents produced by the system builder can be seen to meet the requirements. Prototyping is often used to help the customer and producer reach agreement. The prototype may then be recoded into a paper specification which will form the contractual basis. The paper specification will then be analyzed for the correctness of non-functional requirements such as numerical accuracy or performance require-

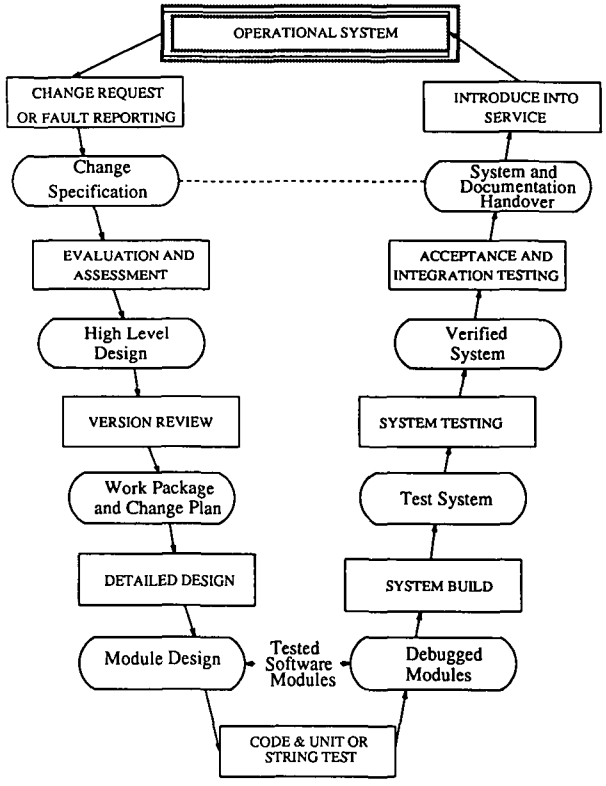

Figure 2 A recent delivered software development and maintenance life cycle (after STARTS, 1987). Quality assurance procedures will embrace the whole life cycle. Justification and authorization to proceed with changes will be reviewed at several points on the downward flow.

ments. If a formal specification is used it may be animated to produce the prototype of the system and also analyzed by computer tools for completeness, consistency and accuracy, but completeness can not be ascertained without discussions with the user. Where the agreed specification does not describe the customer's desires because the mismatch was not noticed, the development of an unsatisfactory product will become contractually binding. After the specification is agreed, the developer should be able to estimate, plan and monitor the development project with a clear view of both the end product and the process that will deliver it.

3.c Life cycle development approaches

It is useful to identify and define the components of development approaches. Current development approaches should have five components: notations, development rules, development guidelines, development processes and development methods. A notation comprises a syntax, a semantics and the relations between them which will allow a system to be described. A development rule determines which descriptions can be verified with respect to other descriptions. Development guidelines indicate how to use notations and development rules to best effect. A development process lays down the nature of, relations between, and order of development for, the descriptions to be constructed and verified. It thereby identifies, among other things, what test strategies are used and how quality assurance is performed. A development method lays down the notations, development rules and development guidelines to be adopted in a particular development process.

Using international standard ISO 9000/9001 the quality management requirements can be put in place for the life cycle described for the particular project. Particular software engineering standards for various stages of the life cycle have been published by the IEEE and are currently being considered by the ISO, such as standard 830–1984 for software requirements specification (ANSI/IEEE, 1984).

The software engineering community already has a set of methods and tools which can be used to instantiate these standards. Development methods such as JSD (Jackson, 1982), SSADM (Ashworth, 1988), Yourdon (Yourdon, 1972) and MASCOT (1979) are gaining reasonably wide acceptance in certain application domains. The acceptance stems largely from the existence of experience from which guidelines have been developed. The notations tend to be graphical and the introduction of tooling for these methods is an active development area. Few development rules are available for these methods.

In addition to these accepted methods, both more rigorous and formal methods have been developed at the research level. A rigorous development method is one in which the guidelines override the rules only when they provide sufficient confidence that the rules could be used to verify the properties assumed for a description. A formal development method is a development method in which the guidelines do not override the rules, in other words, the rules must always be applied to verify that a description has the properties assumed of it.

Many formal notations for system specification have emerged in recent years, for example VDM, Z, OBJ, CSP, Larch, Special, CIP (see Cohen, Harwood and Jackson, 1986 for a review). Some of these notations have associated formal rules (i.e. VDM) and are starting to find application in industry (at least for critical points in systems) as well as working towards international standardization. Prototyping tools to support the notations are becoming available, and case study work is beginning to build up in some application domains from which development guidelines can be extracted.

In the context of formal methods, work on development rules and guidelines is directed at the proof obligations which are imposed by formal methods. Much research remains to be done at the level of both theory and tools before industrial users will be able to apply such rules widely.

There are various techniques for verifying that new descriptions have the desired properties of earlier descriptions. Especially important techniques are:

1 Value testing, which can demonstrate that a new description satisfies test cases in which the input of particular values leads to the output of particular values; for example, values that a numerical addition program produces.
2 Conventional testing, which differs from value testing in that the inputs and outputs may be symbolic expressions instead of particular values.
3 Proof which within some theoretical framework can demonstrate with certainty that the new description is a correct refinement or implementation of the old description.
4 Transformation which produces the new description by modifying descriptions only in ways that are guaranteed to preserve the desired properties.

The current state in the verification of systems in software engineering is that it is only feasible to formally verify small modules of code (200 lines of Pascal or less). Verification therefore tends to be applied only to parts of a design which are critical in some sense. The remainder of the code is verified using a more informal technique.

There is much current work on the provison of tooling for the development process. Classically, tool support has been provided on a stand alone basis resulting in incompatibilities similar to those found in KBS development. Under ESPRIT funding, a common tool interface (PCTE; Cambell, 1986) has been defined which includes user interface and object storage components.

Tooling for formal notations and methods is slowly emerging from a number of projects. A VDM toolset providing structure editors and syntax checkers is available (Crispin, 1987) and generic proof tools are the subject of a part of the Alvey IPSE 2.5 project (Jones, 1987). Other notations and methods are supported by various degrees of tooling.

3.d Managing life cycle development approaches

In addition to the constructive aspects of the development process, life cycle approaches also address the managerial issues: how to manage the process to ensure that all the stages which should have been followed have been followed, to ascertain the state of the development at any point in

time, to provide assistance for planning development projects and in estimating costs, resource effort, and time scales.

There are several current methods available for estimating the cost of conventional software development, the time to complete projects, and the manpower and effort required by them. Most of these use measures derived from large numbers of case studies of system life cycles. If these methods are sufficiently accurate, they can be used to guide decisions about which path to take within the life cycle. Currently, the most commonly used method for estimating is the COCOMO model of Boehm (1981). When the estimators have a large body of experience of programming in a domain, or make estimates at the later stages of the development cycle, such current software cost estimation models can estimate conventional software development costs within 20% of actual costs, 70% of the time.

Computer tools exist to support both project management and estimation in stand alone form, however, methods and tooling for these managerial issues are addressed in an integrated form to varying degrees of sophistication by Integrated Project Support Environments (IPSE's). The key point about an IPSE is that it provides support for all aspects of a development project in an integrated manner, not just for the limited task of "writing the code". The more advanced research projects in this area (for example IPSE 2.5) are concerned with how to provide such support in a generic way; the idea is that an IPSE provides a language and framework within which particular development methods can be described and tooling provided. The development process is then controlled by effectively executing this process description, so that, for example, a user of the system will be told what activities can currently be performed.

For an IPSE to be driven for some project requires decisions to have been made as to which life cycle model to use. Depending on the power and generality of the IPSE in use, this decision can be more or less tentative.

This section has concentrated on describing work in software engineering which may act as a basis for improvements in the development of KBS. But it should be noted that the software engineering work is not complete. For example, there is still much research taking place on requirements capture, prototyping and the validation of the design against the customer's desires at points through the life cycle. Advances in these areas may lead to future developments in other parts of software engineering based life cycles.

4 KBS research advances and life cycles

Following the software engineering development approaches outlined above, three recent research projects have developed life cycle based development methods for KBS. These are complex methods which can only be outlined here, but they show the extent to which software engineering approaches are being incorporated into KBS research methods.

4.a The structured system approach

The KBS method which most closely follows a conventional software development method is presented by Keller (1987), based on Yourdon's structured system analysis methodology modified for KBS development. The life cycle supporting this methodology (illustrated in Figure 3) follows that of conventional structured systems analysis closely. A survey or feasibility study is used to decide if a project is worth doing. The output of this should not only recommend specific areas that can benefit from KBS, but also how KBS should be integrated into the customer's business, and the evolution of training policies. This is followed by "structured analysis" in which the user's needs are specified in terms of functions to be performed and the data relationships between them to produce a "structured specification". A "logical knowledge base description" is also produced to describe the logical information needs of the project. This description is used in the "physical KB design" to specify the implementation details of the knowledge base. The "structured specification" is used in the "design" phase to specify how the user's needs are to be implemented, and the "implementation" phase consists of the production of the system specified in the "packaged design". In parallel with these phases

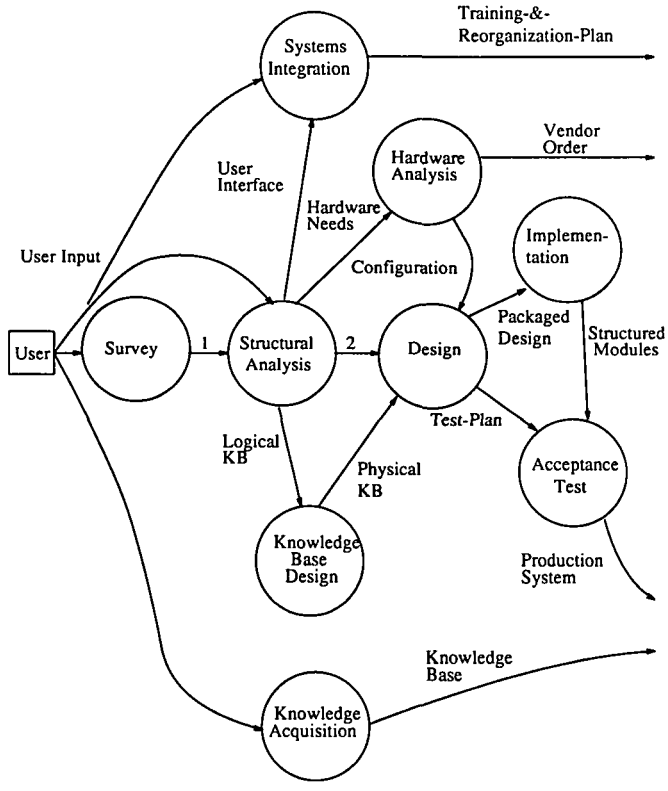

Figure 3 Structural systems KBS life cycle (after Keller, 1987). (1) Feasibility document, (2) Structured specification.

"knowledge acquisition" takes place providing information which is available to all the other phases.

4.b The KADS methodology

A second KBS life cycle developed from a conventional model is presented in the KADS methodology for KBS development resulting from ESPRIT project 1098 (Hayward, 1987). This describes a life cycle based on the waterfall model, the sequential development steps for a system, and the documents that are required to monitor the completion of these steps (see Figure 4).

This life cycle has been developed in detail for the initial analysis phase and will be developed for the design phase. It is unlikely that it will extend to the other stages. In the analysis phase a thorough description has been developed for types of task which provides inference strategies and metaclasses of knowledge compatible with them categorized by problem type (Breuker, 1987). These are provided as a knowledge acquisition method (Breuker and Wielinga, 1987; Wielinga and Breuker, 1986) and a specification of the documentation required for requirements analysis and specification. The analysis stage is supported by a computer based documentation system, a detailed handbook and tools to aid developers in its use (Anjewierden, 1987). One limitation within the analysis phase is that the knowledge acquisition method only applies to the capture and structuring of information obtained verbally and has not been developed to account for other knowledge sources.

The overall life cycle has many of the same limitations as the life cycle in current commercial practice in that:

a. it is described only from the point of view of the software engineer, and so, does not include estimating methods;
b. it only accounts for the "normal" KBS system, it does not describe the life cycles for KBS which

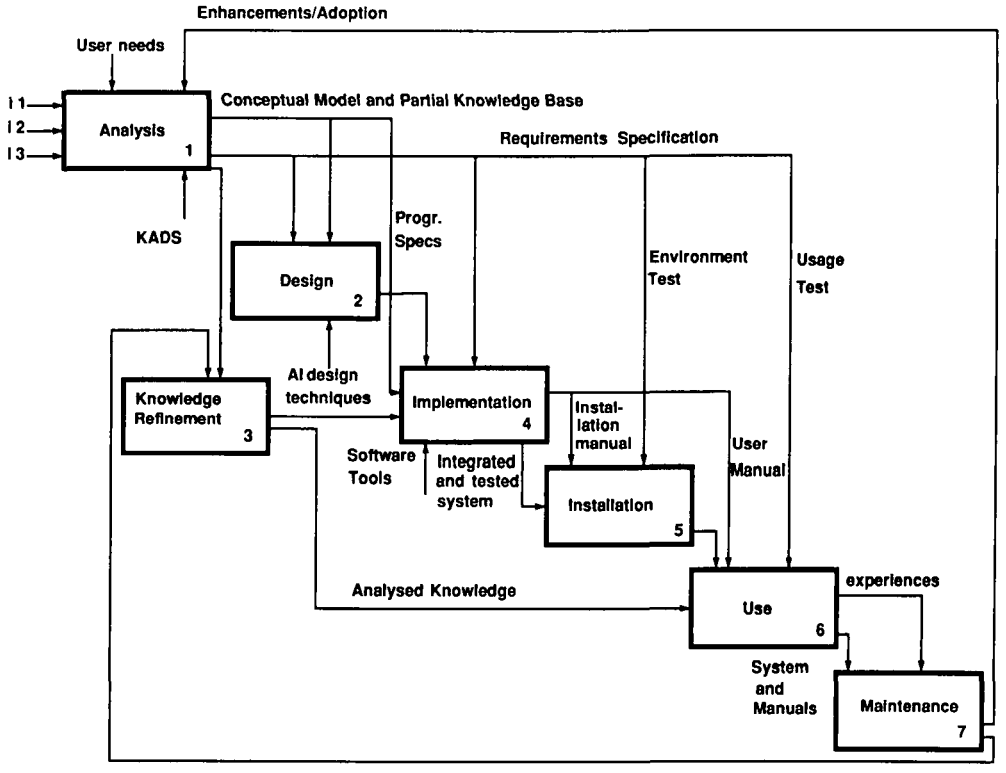

Figure 4 Esprit P1098 KBS life cycle (after Hayward, 1987).

interact with conventional software, either to access databases, as real-time systems, or as other forms of embedded system;

c. it does not incorporate software engineering methods for verification and validation across the whole, or stages of, the life cycle.
d. it does not incorporate prototyping, and follows a strictly incremental path.

The methodology has not yet become (either formally or de facto) an industrial standard, but it has been very influential in European research centres. Systems have been developed using the methodology for the parts of the life cycle where it applies and have shown it to be useful in those areas.

4.c The GEMINI project

The GEMINI project (Montgomery, 1988, 1989; Worden, 1988) was recently initiated in order to develop an expert systems development methodology which will integrate with the SSADM method for developing conventional software systems. The life cycle used here overcomes many of the problems of the previous two in that it both incorporates incremental prototyping as a form of validation and is linked to a method for conventional software development to allow it to address developments which mix both forms of system. The life cycle model from this project and the paper version of the method will be completed during 1989. At present, the life cycle model includes the six stages of SSADM (1, analysis of systems operations and current problems; 2, specification of requirements; 3, selection of technical options; 4, data design; 5, process design; 6, physical design), however, instead of a linear organization, these all receive their inputs and pass their outputs to a validation process. This process includes the assessment of the deliverables and the contingent choice on future action. In addition to the graphical representation techniques of SSADM and other techniques it employs, (relational data analysis; first cut rules; physical design control; quality assurance reviewing; project estimating) a battery of validation techniques are employed including prototyping. The KBS specific techniques such as knowledge acquisition are included as methods available within the relevant stages of the development method. Consequently, GEMINI can be

seen as a harness for validation and knowledge based development within which SSADM rests. The method and tooling will continue to be developed as the project progresses.

Although GEMINI is an advance on the alternative development methods, it is restricted in certain ways. It does not specifically address the problems associated with real-time KBS, and it seems at present that the close coupling of KBS with conventional components will not be a high priority. Rather, the emphasis is more on the compatibility of the KBS methodology with SSADM, and indeed one possibility for addressing the present mismatch between the two different kinds of life cycle is the partial decoupling of the development of the two components.

As well as the development of these overall life cycle approaches, there have been attempts to provide verification and validation (V&V) procedures following those used in software engineering to show that KBS meet user requirements (e.g. Martinez, Muro and Silva, 1987). Since V&V procedures do not yet exist for KBS while they are often mandated by procuring agencies for mission-critical computer programs, many organizations that might otherwise consider applying KBS are forced to employ more costly and less capable conventional software. Most V&V measures suggested for KBS have failed because KBS are typically developed from informal specifications through prototyping. This method has not provided adequate specifications for tracing requirements since the desired outputs are often emergent properties of the interaction between the knowledge base and the inference engine. It has been suggested that a specification of requirements could be developed after a prototype has been developed, and then these requirements could be used as the basis for V&V in the later development. The problems arising from pressure to conform the requirements to the software suggest this would only succeed if the V&V practitioner had sufficient organizational authority to set the requirements over the developer's objections. Of the four forms of verification described in section 3.c, value testing is the only type currently applicable to KBS. How ever, where the selection of test cases cannot be led by a requirements specification, they must be constructed from a domain expert's knowledge of boundary conditions and significant combinations of conditions.

Since requirements tracing appears inappropriate for most KBS, several engineering analyses have been suggested as possible evaluative measures for KBS before effort is expended on coding and testing (Green and Keyes, 1987). These include: criticality (the cost of failure of the KBS component on the remainder of the system); sensitivity and stability (the systems response to variations in the input, especially around operating limits); efficiency (can the system respond within time and resource limits); maintainability (is the software written to facilitate maintenance); interaction analysis (predicting the interaction of knowledge base modules and the inference engine); truth analysis (assessing the truth of facts, rules and other knowledge base components); uncertainty analysis (to check the consistency of uncertainties represented in the KBS with the expression of uncertainty in the domain). Unfortunately, there are not yet sufficient data available on KBS developments using these measures to assess their effectiveness.

4 Conclusion

Artificial intelligence covers a large area of research into "complex information processing" (the label preferred by Simon at Carnegie Mellon University). Partridge and Wilks (1987) argue that the problems of complex information processing do not permit specifications to be exact, and consequently even well verified systems based on such specifications will be inaccurate. They argue that the natural form of AI methodology is a "Run-Understand-Debug-Edit" (RUDE) cycle where the requirements can be continually altered. Although this can describe the method of code hacking which has always existed, it can also describe a form of iterative prototyping if some measure of convergence on a solution is applied. Despite the lack of an objective measure of convergence on a solution this is still the best practical method of development for experiments in many areas of AI. In this view, specifications can be seen as the products of AI developments.

First generation expert systems address a small part of complex information processing. Techniques and tools have been developed from early experiments which themselves used the RUDE

method. These have been summarized in the current commercial KBS development life cycle in the first section of this paper. There are still many disputes about the detailed structure of this life cycle, but there is also much agreement that its general shape cannot follow the RUDE method if practical systems are to be built according to contractual time scales and standards. Current KBS applications tend to have poor or non-existent design documents. In many current KBS projects, stages in development are not defined with the result that there is little management control, the work of individuals cannot be integrated and software is hard to validate and maintain. Many current KBS development projects are either bound to a single tool, or require considerable effort to transfer material between incompatible tools. Other areas of AI have not yet developed to produce commercial systems and structured development methods are not applicable to them. However, first generation expert systems and other KBS developments have reached the point where mechanisms to meet contractual obligations should and can be introduced.

Developments in software engineering have lead to structures and documentation for the life cycle for conventional systems which allow estimation, monitoring, verification and validation in order to meet contractual obligations. They have also introduced forms of prototyping into the life cycle for conventional systems so that initially ill-specified systems can be developed within it. It would not be appropriate to transfer software engineering life cycle methods to KBS which would prevent ill-specified problems from being addressed. However, aspects of software development control described in the second section of this paper can be introduced to the present KBS development cycle to allow estimation and monitoring without limiting the applicability of KBS techniques. Software engineering has described the stages and methods for controlled development and is now developing IPSE's to support the automatic control of the cycle in order to increase the speed and ease at which it can be performed whilst retaining product quality and management control. These will offer the ability to change program requirements more frequently than when greater manual effort is required during the cycle. They should perform the functions which the AI toolkits designed to support RUDE do, while encouraging planning and monitoring.

Three major research projects have addressed the problems of controlling and managing the KBS development cycle. These have incorporated aspects of software engineering into KBS development to describe life cycle stages, documentation, and management techniques. None of these is likely to be the perfect solution to developing all classes of KBS. Keller's approach is tightly tied to Yourdon's method and only applies to KBS developments related to database systems. The KADS method neither incorporates KBS systems linked to conventional software components, nor prototyping, although it has provided a thorough method for knowledge acquisition within the life cycle. This progress should lead to development projects which are easier to monitor and manage. Although other research has suggested methods for verification within KBS developments this aspect still requires further work, as does the integration of KBS tooling.

The largest problem which is not directly addressed by these projects is that of estimating the effort, time scale and manpower for KBS developments. Neither theoretical work based on software engineering, nor small scale research projects can provide a solution to this. Methods for estimating must be based upon data from both successful and unsuccessful large scale KBS projects. Although there are many accounts of successful small KBS and some of large KBS, few accounts of unsuccessful KBS projects are available. This data is required before managers can estimate the course of KBS developments in advance.

This paper does not provide a solution to this problem, nor does it provide a life cycle method of KBS development which will guide managers or KBS producers. It has provided an introduction to some of the problems in KBS development and an outline of the progress being made towards solutions to some of them. The articles referred to in this paper provide further reading which will allow both the problems and suggested solutions to be understood further.

References

The references are grouped by subject and annotated to enhance their tutorial value.

The best practice in current KBS development

Cercone, N and McCalla, G, 1987. "What is knowledge representation" In: *The Knowledge Frontier: Essays in the Representation of Knowledge* N Cercone and G McCalla (Eds.) New York: Springer-Verlag. A review of knowledge representation techniques.

Frieling, M, Alexander, J, Messick, S, Rehfuss, S and Shulman, S, 1985. "Starting a knowledge engineering project: a step-by-step approach" *AI Magazine* **6**(3) 150–165. A guide to the Tektronix knowledge based system development methodology.

Hayes-Roth, F, Waterman, D A, and Lenet, D B, 1983. *Building Expert Systems* Reading, Mass.: Addison-Wesley.

Laurent, J-P, Ayel, J, Thome, F and Ziebelin, D, 1986. "Comparative evaluation of three expert system development tools: KEE, Knowledge Craft, Art" *The Knowledge Engineering Review* **1**(4) 18–29. A review of the larger toolkits to support knowledge representation and KBS development.

McGraw, K L and Seale, M R, 1988. "Knowledge elicitation with multiple experts: considerations and techniques" *Artificial Intelligence Review* **2**(1) 31–44.

Motta, E, Eisenstadt, M, West, M, Pitman, K and Evertsz, R, 1986. "KEATS: the knowledge engineer's assistant" *HCRL Report No. 20*, Milton Keynes, U.K.: Open University. A description of one of the more advanced tools to support knowledge acquisition.

Neale, I M, 1988. "First generation expert systems: a review of knowledge acquisition methodologies" *The Knowledge Engineering Review* **3**(2) 105–145.

Peters, L J, 1980. "Software representation and composition techniques" *Proceedings of the IEEE* **68**(9) 1085–93.

Prerau, D S, 1985. "Selecting an appropriate domain for an expert system" *AI Magazine* **6**(2) Summer 26–30.

Rajan, T, Motta, E and Eisenstadt, M, 1989. "ACQUIST: a tool for knowledge acquisition" In: *Research and Development in Expert Systems V* B Kelly and A Rector (Eds.) 113–125, Cambridge University Press. This paper describes a part of the KEATS-2 suite of tools for KBS development.

Shaw, M L G and Gaines, B R, 1987. "Advances in interactive knowledge engineering" In: *Research and Development in Expert Systems III* M A Bramer (Ed.) 111–122, Cambridge: Cambridge University Press. This paper describes the interactive repertory grid elicitation and analysis tools for knowledge acquistion commercially available as RepGrid from the authors.

Williams, A and Lambert, S, 1988. "Expressive power and computability". In: *Approaches to Knowledge Representation: An Introduction*, G A Ringland and D A Duce (Eds.) Letchworth, England: Research Studies Press.

Wyatt, J, 1987. "The evaluation of clinical decision support systems: a discussion of the methodology in the ACORN project". In: *Proceedings of AIME '87*, J Fox, M Fieschi and R Engelbrecht (Eds.), Berlin: Springer-Verlag.

Young, R, 1987. "Intermediate representations". In: *Research and Development in Expert Systems IV* M A Bramer (Ed.), Cambridge: Cambridge University Press.

Software engineering methods

ANSI/IEEE, 1984. "ANSI/IEEE standard 830–1984" *IEEE Guide to Software Requirements Specifications*, New York, NY, USA: IEEE Standards Board.

Ashworth, C M, 1988. "Structured systems analysis and design method (SSADM)" *Information and Software Technology* **30**(3) 153–163. This paper outlines the stages of the standard software development method used for UK government projects.

Boehm, B W, 1981. *Software Engineering Economics* Englewood Cliffs, NJ, USA: Prentice Hall. This is the standard text on the prediction of the time, effort and resources required for a software development project.

Cambell, I, 1986. "PCTE proposal for a common tool interface". In: *Software Engineering Environments* I Sommerville (Ed.), London: Peter Peregrinus on behalf of the Institution of Electrical Engineers. This is a description of the EEC sponsored interface for software development tools.

Cohen, B, Harwood, W T and Jackson, M I, 1986. *The Specification of Complex Systems* Wokingham, UK: Addison-Wesley. This review describes several formal notations for the specification of systems, and the formal development approach which can follow from them.

Crispin, R J, 1987. "Experience using VDM in STC". In: *VDM '87: A Formal Method at Work, VDM-Europe Symposium* Berlin: Springer Verlag. VDM is the most advanced formal development method, which is described here in terms of its practical application and limitations.

Jackson, M A, 1982. *System Development* Englewood Cliffs, NJ.: Prentice-Hall International. The Jackson Structured Design method is one of the most popular rigorous development methods along with that described by Yourdon.

Jones, K D, 1987. "Support environments for VDM". In: *VDM '87: A Formal Method at Work, VDM-Europe Symposium* Springer Verlag: Berlin. A description of advanced integrated project support environments for formal software engineering methods.

Macro, A and Buxton, J, 1987. *The Craft of Software Engineering* Reading, Mass.: Addison-Wesley. This is one of the best introductions to the area of software engineering.

Mascot, 1979. *The Official Handbook of MASCOT* MASCOT Suppliers Association MASCOT is a rigorous development method for time critical software systems, used by for UK government projects.

Myers, G J, 1979. *The Art of Software Testing* John Wiley and Sons: New York, NY. Springer-Verlag (Pub.), 1984. *Approaches to Prototyping* Springer-Verlag: Berlin. The issues associated with prototyping within software engineering are addressed in depth in the papers in this collection.

STARTS Purchasers Groups, 1987. *Supplement to the STARTS Purchasers Handbook* Manchester: NCC Publications. The STARTS team recommended software tools for software development and the management of software projects in this text.

Yourdon, E, 1972. *Techniques of program structure and design* Englewood Cliffs, NJ.: Yourdon Press. One of the most popular structured systems methods is described in this book.

KBS development approaches applying software engineering techniques

Ford, L, 1987. "Artificial intelligence and software engineering: a tutorial introduction to their relationship" *Artificial Intelligence Review* **1**(4) 255–273. An introduction to the role of software engineering in KBS development.

Green, C J R and Keyes, M M, 1987. "Verification and validation of expert systems". In: *WESTEX 87—Proceedings of the Western Conference on Expert Systems* 38–43, Washington, D.C.: IEEE Comput. Soc. Press. A description of verification and validation procedures which could be applied to expert systems.

Martinez, J, Munro, P, Silva, M, 1987. "Modelling, validation and software implementation of production systems using high level Petri Nets", In: *Proceedings of the 1987 IEEE International Conference on Robotics and Automation* 1180–1185, Washington, D.C.: IEEE Computing Society Press. A technique for producing production rule systems which can be validated against user requirements.

Partridge, D and Wilks, Y, 1987. "Does AI have a methodology which is different from software engineering?" *Artificial Intelligence Review* **1**(2) 111–121. A controversial defence of the differences between AI and software engineering methods.

The KADS methodology

Anjewierden, A, 1987. "Knowledge acquisition tools" *AICOM* **0**(1) 29–38. This describes the tools being developed to support the KADS KBS development method.

Breuker, J (Ed.) 1987. *Model-Driven Knowledge Acquisition Interpretation Models* ESPRIT Project 1098, Deliverable Task A1, University of Amsterdam. A thorough description of the models used in the KADS knowledge acquisition phase.

Breuker, J and Wielinga, B, 1987. "Use of models in the interpretation of verbal data". In: *Knowledge Aquisition for Expert Systems: a Practical Handbook* A L Kidd (Ed.), New York: Plenum Press. A description of the KADS knowledge acquisition phase.

Hayward, S, 1987. "How to build knowledge based systems: techniques, tools, and case studies". In: *ESPRIT '87: Proceedings of the Esprit Conference* 665–687, Brussels: Commission of the European Economic Community. A brief overview of the complete KADS project.

Wielinga, B and Breuker, J, 1986. "Models of expertise". In: *Proceedings of ECAI 1986* Brighton. This paper provides an overview of the models used in the KADS knowledge acquisition phase.

Structured systems approach

Keller, R, 1987. *Expert System Technology: Development and Application.* Englewood Cliffs, NJ, USA: Yourdon Press. A description of the Yourdon structured system development method as applied to KBS.

The GEMINI project

Montgomery, A 1988. "GEMINI: government expert systems methodology initiative". In: *KBS in Government 88* P Duffin (Ed.), 75–88, London: Blenheim OnLine Ltd.

Montgomery, A, 1989. "GEMINI: government expert systems methodology initiative". In: *Research and Development in Expert Systems V* B Kelly and A Rector (Eds.), 14–24, Cambridge: Cambridge University Press.

Worden, R, 1988. "The GEMINI project: preliminary results". In: *KBS in Government 88* P Duffin (Ed.) 119–130, London: Blenheim OnLine Ltd.