Pergamon

Expert Systems With Applications, Vol. 7, No. 2, pp. 259–271, 1994
Copyright © 1994 Elsevier Science Ltd
Printed in the USA. All rights reserved
0957-4174/94 $6.00 + .00

An Integrated Methodology for Knowledge-Based System Development

ROBERT T. PLANT AND PANAGIOTIS TSOUMPAS

Department of Computer Information Systems, University of Miami, Coral Gables, Florida

Abstract—*This article details a rigorous development methodology for knowledge-based systems. This rigorous methodology is itself embedded within a multilevel process model for software development. The rigorous methodology is designed to utilize a set of formal or rigorous specifications in a composite style. These specifications detail areas like the knowledge base, the human–computer interface, and the representation, linked together through a process of representation refinement. The rigorous methodology aims at combining the aspects of knowledge engineering, cognitive engineering, and software engineering as they relate to knowledge-based systems. The knowledge-based systems development methodology is embedded within a two-level life-cycle model. The two levels are termed the* macro- *and* microlevels. *The macrolevel is used to understand the impact that those factors external to the actual system development have upon the system's creation and life cycle. These external factors include such influences as changes in technology and corporate planning. The microlevel is a process model that utilizes techniques from total quality management, measurement theory, and cost estimation, among others, to assist the software developer in producing software through a process of never-ending quality improvement. All of these techniques are utilized and complimentary to each other. The aim of having two levels is to allow the developer to focus upon each item separately but to understand the factors upon which the factor's development rests and its impact upon the other subprocesses.*

1. INTRODUCTION

THE AIM OF THIS ARTICLE is to illustrate the integration of a two-level process model of software development with a rigorous methodology for the development of knowledge-based systems. The two-level process model is a methodology that separates the creation process into two distinct levels: the *microlevel* and the *macrolevel*. The macrolevel focuses upon those factors external to the software creation process itself, such as customer requirements, technology development, and the corporate business plan of the customer. The microlevel focuses upon the process of software development itself: feasibility studies, detailed requirements, system design, and so forth. In separating these two levels, we can focus upon the external factors in isolation, yet the developer can obtain a better understanding of their interaction, the influence of each, and the relative effect each has upon the others. The third element in this article is to take this two-level development life-cycle model and adapt it such that the special needs of expert systems development can be incorporated in the model. This is done by integrating a rigorous knowledge-based system development methodology into the two-level process model.

Requests for reprints should be sent to Robert T. Plant, Department of Computer Information Systems, University of Miami, Coral Gables, FL 33124.

2. OVERVIEW OF EXISTING SOFTWARE LIFE CYCLES

We can consider the development of knowledge-based systems from three perspectives:

- methodologies that apply only to knowledge-based system development
- methodologies that apply only to procedural systems development
- methodologies applicable to the development of knowledge-based systems, traditional systems, and embedded systems

The first category, methodologies that are specifically (or by default) designed to assist developers in creating traditional, procedural systems, has been acknowledged to be of little value to the developer of knowledge-based systems (Miller, 1990; Plant, 1993). Methodologies such as those proposed by Royce (1970), Boehm (1988), or the DOD-2167A (1988) do not generally facilitate the knowledge engineer in handling the problems of incompleteness and inconsistency of data and fuzzy values and do not have the ability to assist system construction from weak specifications.

The weakness of the traditional waterfall-based life-cycle models is not assisted by the utilization of formal techniques of specification, as these too require the creation of complete specifications in advance of system development such that the system can be constructed from the specification, by following rigorous refinement steps. Thus, this made it necessary for the knowledge-engineering community to build models of their own processes that lead to the creation of pattern-directed inference systems (expert systems). The early methodologies of Buchanan, Davis, Lenat, Grover and Wielinga (Buchanan et al., 1983; Davis & Lenat, 1982; Grover, 1983; Wielinga & Breuker, 1983) were based primarily upon adaptations of Royce's waterfall model (Royce, 1970) and were stage-based in nature. However, they were inherently weak, and even though they did attempt to incorporate knowledge-engineering phases into their life cycles, they still suffered from the same problems encountered when applying traditional life-cycle models to knowledge-based problems. These early knowledge-based life-cycle models gave way to more sophisticated models of knowledge-based system development that attempt to consider the problems associated with representation selection, domain knowledge completeness, determination of domain knowledge correctness, validation and verification issues, prototyping, iteration, integrity, and maintenance issues (Miller, 1990; Plant, 1993; Weitzel & Kerschberg, 1989). These we can classify as belonging to the third category, life-cycle models that address knowledge-based system issues that are associated with mature systems of industrial strength (Miller, 1990) as opposed to the immature (or trivial systems) with which previous methodologies dealt.

The remainder of this article considers a new methodology for knowledge-based system development—one that combines careful consideration of external factors pertinent to commercial and pragmatic system development (Plant & Hu, 1992; Plant & Salinas, 1992) with the factors that affect the process of system development, in this case, knowledge-based system development.

3. TENETS OF A METHODOLOGICAL DESIGN

As we noted earlier, knowledge-based system design can be considered from three basic perspectives. However, a set of common weaknesses can be identified in each of these approaches:

- the absence of process control
- an emphasis upon inspection as a mechanism for system acceptance
- low priority placed upon organizational behavior of development personnel

The methodology we present aims at addressing these issues. It is our proposition that the incorporation of rigorous techniques in development as well as changes in management practices inspired from quality management theories will be of significant benefit.

The model proposed in this article has been created to avoid these problems through the utilization of a two-level model that includes the external factors that influence development. The two levels, macro- and micro-, integrate with four tenets for development that the software engineer must follow:

- Utilize a philosophy of continuous quality improvement in all aspects of development.
- Integrate in the development process the techniques of formal methods, metrics, and quality variance.
- Utilize metrics to control, monitor, and understand the process.
- Utilize tools and resources to promote communication and improve the communication.

These tenets are discussed throughout this article and are brought together to form the MM-Level process model. Within the MM-Level model is a rigorous process model for the creation of knowledge-based systems. This model has the following tenets:

- Every step in the development from initial specification to implementation should be capable of justification.
- The process should have an implementation-independent representation of the domain knowledge.
- The correct representation should be chosen, and the decision should be justified.

The overall aim of this methodology is to allow the creation of knowledge-based systems that also can be embedded into a larger development environment upon completion, the system into which it is embedded having also been created through a version of the MM-Level process model.[1]

4. OVERVIEW: THE INTEGRATED MM-LEVEL PROCESS MODEL

The aim of this article is to describe a two-level software development life-cycle methodology. These levels, macro- and micro-, combine to compose our MM-Level model. The macro perspective places the software development process in its environment with respect to the external factors. Research in software development has focused primarily on the importance of customer, neglecting other environmental factors that affect the quality of the final product. The second level describes the software development process from a micro perspective, where the methods and practices of improving the quality of products and processes are of primary concern. Integrated into these two levels is a

[1] In a conventional system the knowledge-based component would be replaced by a more standardized systems analysis and design component.

knowledge-based system development methodology that focuses upon the special issues that affect knowledge-based system development, for example, representation selection, elicitation techniques, and so forth.

5. MACRO PERSPECTIVE

The proposed macro perspective views the software development process as a part of a greater system that includes factors external to the development process itself (see Fig. 1).

These external factors include:

- latest developments in the hardware and software areas
- traditional input from the customer about requirements of the system under study
- input from various customers, through customer support, about problems discovered during the operation stage, and from potential customers, through marketing research, about current needs
- input from top management, through the C.I.O., about corporate business plans

In addition, factors like the impacting economic conditions, the strategic plan of the organization, and the competitive environment of the organization need to be considered. The exact set of external factors will be unique to a large extent for each organization.

The influence of these external factors on the software development process takes place through various forms: *directly* in the form of user requirements, or *indirectly* by changes and developments in technology. Even though the indirect factors have great impact on the development process, their importance historically has been neglected or downplayed. The model we propose utilizes these external factors to increase the awareness of management about such factors. We now consider these external factors in more detail and their effect upon the development process.

5.1. The DeltaT (ΔT) Effect

The rapidity with which technology changes in the area of information technology means that management and software engineers cannot afford to isolate themselves from these changes. However, unless management and developers control their software processes through an understanding of the impact that hardware, software, practical, and theoretical developments have upon it, these technology changes may have a serious and detrimental effect upon the developers' software creation process. We call this the ΔT effect (change in Technology) and suggest that it is management's role to encourage positively the dissemination of new knowledge and techniques.

5.2. Customer Requirements

One of the most influential of the external factors is the determination of the customer requirements. The

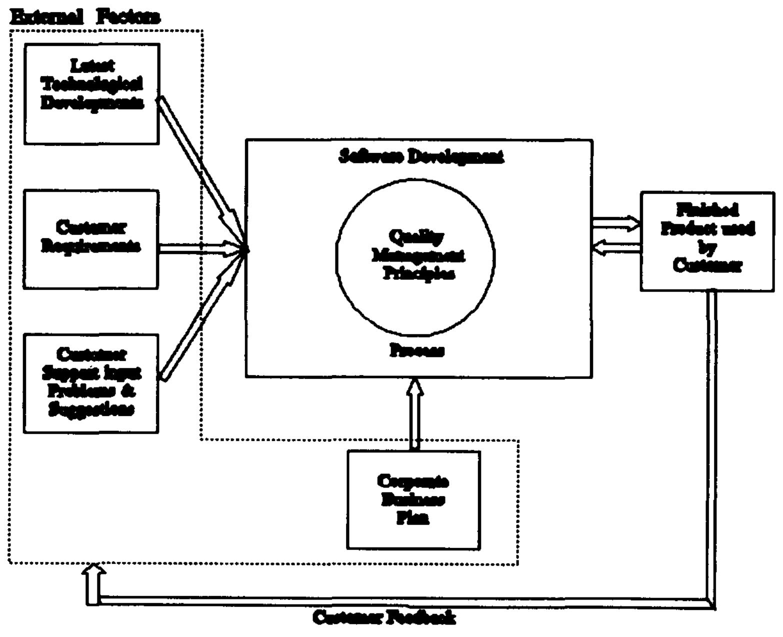
External Factors
Latest Technological Developments
Customer Requirements
Customer Support Input Problems & Suggestions
Software Development
Quality Management Principles
Process
Finished Product used by Customer
Corporate Business Plan
Customer Feedback

FIGURE 1. Macrolevel model.

external factors that weigh upon the determination of customer requirements in the macrolevel model are focused primarily upon the ability to use the appropriate level of rigor for the task and the customer environment that will produce a clear and unambiguous specifications document. This is open to factors like the customers' constraints in standards, laws pertaining to the area, and criticality of the domain.

5.3. Customer Support

Customer support is a function of a software development organization whose input either is not fully utilized or is neglected altogether. Customer support serves a dual purpose, (a) it supports customers during the implementation with training, and (b) it is the front line of the corporation for obtaining input from customers about discovered defects. Even though the first service provided by customer support is important for the image of the company, a discussion about it is out of the scope of this article. The second service provided by customer support though, is very important, and its significant input has been underutilized. Customer support provides input to software developers for newly discovered problems (defects), the only utilization of which is often for finding the defect and correcting it. We suggest that this input could prove to be very valuable if software developers were to trace the causes that create that defect (Tsoumpas, 1993). Such utilization, though, requires rigor and formality throughout the development process to be able to trace back such details. We discuss methods and practices that bring rigor and formality to the development process later in the microperspective part of our proposed model.

5.4. Corporate Plans

The corporate business plan is an external factor that has a special relationship with the end-product's quality. If there is a corporate plan for releasing a new product by a certain date and there are delays in the progress of the project, management's attitude will be decisive. Whether management's attitude is "meet the deadline regardless of defects," or, alternatively, "continue the high quality work and let's figure a way to work more efficiently so that we will not have any more delays," is instrumental in determining the final product's quality. Research by Weinberg and others (Weinberg & Schulman, 1974) has found that, given specific objectives, programmers can make the required choices to meet these objectives, provided the objectives do not conflict with each other. It has been suggested by all the quality advocates (Deming, 1986; Imai, 1986; Ishikawa & Lu, 1985; Juran, 1964) that management's commitment is of paramount importance for the success of a quality improvement program. Thus, we feel it is appropriate to include corporate plans as an influential factor in the software development process. Management commitment to the process of quality is therefore imperative; we can see that they have the ability to determine the pressure placed upon the software engineers to balance deadline dates against quality levels. Even though some research (King, 1978) has discussed the influence of corporate top management in information systems planning, there is little formal research on how the top management plans influence the quality of the end product. We believe that such influence exists and significantly affects the end product.

5.5. Summary of Macro Perspective

It has been the aim of this section to show the importance of the external factors in relation to the process of software development. Two major reasons for this are as follows. First, the identification of influential factors provides the manager with a better understanding of these factors and their role in the software development process. Second, by knowing about these factors, managers can incorporate them into their plans and control their influence over the software development process.

6. MICRO PERSPECTIVE

The micro perspective view of the software development process describes the way a number of quality management techniques and software development practices support the development of high-quality software. Because they support the software development life cycle, we call them *life support tools;* they are portrayed in the central component of Figure 2.

Boehm (1988) recognizes, indirectly, that the major problem of the software development process is the lack of adequate planning; therefore, he suggests that *risk analysis* is helpful in identifying problems that might occur. We strongly agree with this argument; however, we believe that risk analysis is only one of several techniques that management should utilize in its effort to monitor the software development process. In the MM-Level process model proposed here, we advocate that risk analysis be supplemented with some techniques and practices that, we believe, would enhance management's planning ability. These life support tools are:

- historical database
- software metrics program
- configuration management
- cost estimation model
- quality management practices

In the following sections we consider several of the tools in this life support system and identify their strengths and how they contribute to the improvement of the quality of the end product.

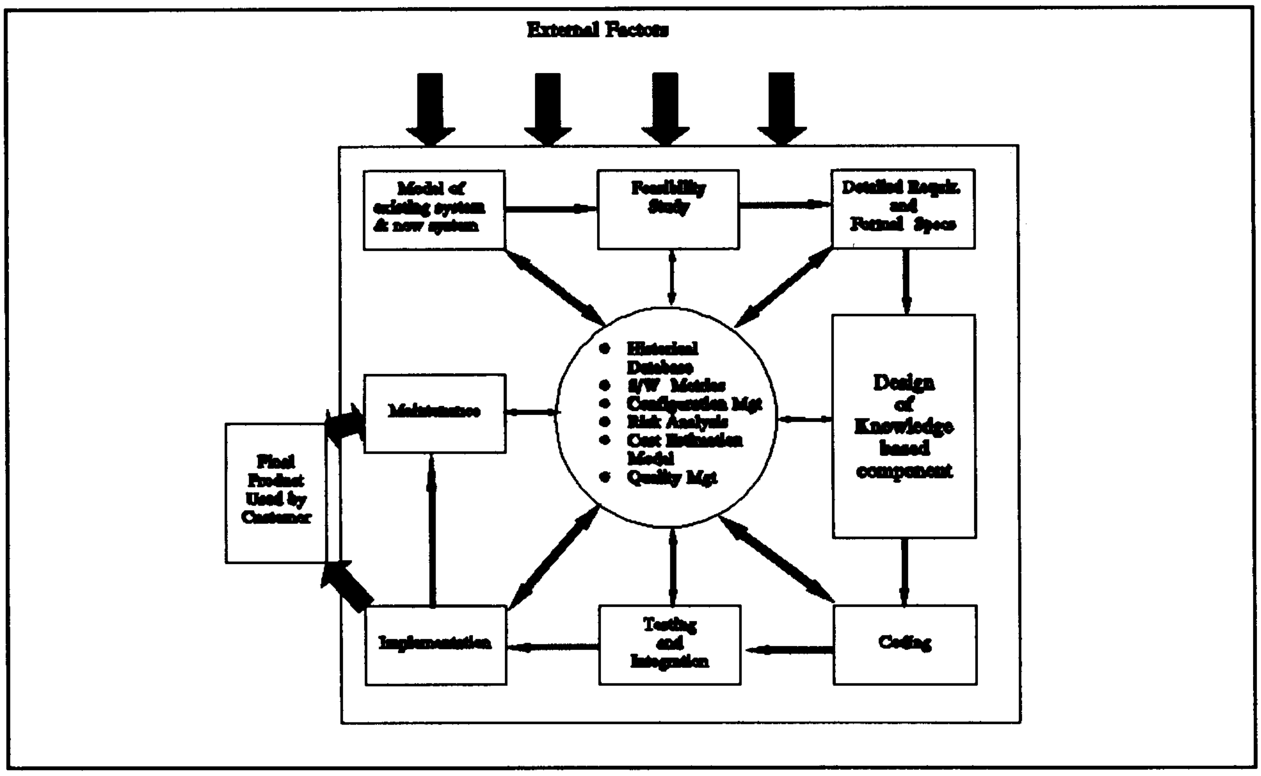

FIGURE 2. Microlevel model.

6.1. Measurement and Metrics

To assess the impact of differing methodologies upon development, software engineers need to increase their utilization of measurement theory and software metrics such that developers can more easily determine the impact and effect one parameter of a design has upon another. Research is active in this area, and workers consider such design issues as module length, optimal number of modules, and criteria of module separation (Conte, Dunsmore, & Shen, 1986; Sheppard, 1990).

The aim of software metrics is to assist the developer in understanding the relationship between the parameters of software design and its creation. These parameters are, however, not always easy to determine or measure, in addition to the difficulty of determining the consequences of the results. For example, a software engineer may use the whole-function criterion or an optimum-length criterion (Conte et al., 1986) to decide when to split a program in modules, or to determine the optimum length of a module, but other human-based parameters, such as measurement of a programmers experience or suitability of the programming language used, are not so easily measured.

This state of flux in software measurement necessitates that companies utilize a software metrics measurement program that will help each specific company to understand their own development parameters and their interrelations. Grady (Grady & Caswell, 1987) has pointed out that a program like this is difficult to implement, and serious commitment of all interested parties should be obtained before the implementation.

A comprehensive review of metrics and predictive models is presented by Conte et al. (1986) and Zuse (1992).

6.2. Cost Estimation Model Adoption

The first step in the development of a system within an organization is a managerial one, in that while the requirements for a system are being formulated, the system should be endorsed and backed by the top management of that organization. This necessitates that the system perform a useful function, be revenue-producing, and be developed in a cost-effective manner. Thus, it is necessary for the development team to adopt a software cost-estimation model such as COCOMO (Boehm, 1980), SOFTCOST (Tausworthe, 1981), or COSTMODL (NASA, 1991) prior to system development. The adoption of an appropriate cost model will enable the development and life-cycle cost centers to be identified early, thus the management will be confident in the systems return on investment prior to development. This will assist in obtaining a favorable management commitment to the project from the conception of the system.

6.3. Standards

As Hall (1990) suggested, formal methods could be used to guide the software developers in the design and programming phases and additionally serve as a clarifier of the requirement specifications document for the customer. A practice that enhances the robustness of the requirement specifications document is the use of standards that assist for uniform understanding of terms, techniques, and approaches. These standards are aimed at providing unambiguous definitions and act as a baseline document. Participation in this is a practice that can be adopted by all software development companies and establish industry-wide standards. This removes the ambiguity of such terms as: "user friendly," "easily maintainable," or "reliable." Terms such as these often have a different meaning for developers and customers, and both parties rely on their own personal judgment for translating them, resulting in a conflict over the final product.

By introducing these standardization practices, in conjunction with formal methods, management will be able to improve the communication between customers and developers. The employment of a quality management program guarantees that such effort will not be static, but that it will be improved by any newly acquired information.

6.4. Quality Management Practices

The quality management principle "each process is the customer of the previous process and the supplier of the subsequent one" (Gitlow & Gitlow, 1987) can be considered as a second tenet for the software company's process management. This can be seen when the specification of the system is passed from the requirements-formalization team to the software design team. The specification has to possess the ability to improve the communication between all members of the development process. Thus the key to achieving this process pipeline is *communication.*

It can be foreseen that one approach to achieving this level of communication and interaction is through Quality Circles, where professionals from both sides come together and discuss methods of improving the current practice by introducing new documents, changes in currently used documents, and additions in the utilized tools.

In the next two subsections we consider two important instruments through which quality process management can improve the product: quality circles and training.

6.5. Quality Control Circles

The introduction of external factors that influence the software development process had the purpose of emphasizing the need for communication in an efficient way that can be proven beneficial for the company. The introduction of quality control circles (Imai, 1986) as a forum of communicating ideas inside the process serves the need for exchange of information between project team members and members of other teams.

The aim of quality control circles is that people with different job assignments and different educational backgrounds come together and discuss their experiences, their everyday task problems, and that they voice their concerns about various subjects.

To maximize the return on the time invested in the quality circles it is useful to maximize the use of tools from quality management, such as Pareto charts and cause-and-effect diagrams, (Gitlow, Gitlow, Oppenheim, & Oppenheim, 1989; Ishikawa, 1982), which should be used by members of the quality circles. As Deming (1986, 1991) said in both his 14 points and his system of Profound Knowledge, general education is a very important aspect, and as such, subjects that are of common interest among the members of a quality control circle could be discussed, even if they are not related directly to software engineering. An extension of quality circles could include as members representatives from both customers and suppliers. Meetings with members of these entities in the business environment help formulating plans for future systems development.

An important aspect of quality circles is that each member chooses to participate in this kind of activities without management involvement. Managers of any level of hierarchy should also participate, without carrying with them into the quality circle environment their status within the company.

6.6. Training

A fundamental aspect of any quality program (Deming, 1986) is education and training, which focus upon the continuous and never-ending cycle of training, application, and quality improvement. This philosophy needs to be integrated and absorbed by the software development organizations for any true progress toward the production of quality software to be achieved.

6.7. Knowledge-Based Design Component

The micromodel of development offers a versatile approach to the software designer in that it allows aspects of the model to be adapted and amended to suit the system development needs, for example, real-time parallel processing. In the remainder of this article we show how the MM-Level model can be integrated with a development methodology for the creation of knowledge-based systems. The methodology is rigorous in nature and is intended to assist the knowledge engineer in isolating the descriptions of the knowledge-based component from the representational component (Plant, 1993). The design of the knowledge-based component replaces the standard system design and detailed design components in the MM-Level model.

6.7.1. *The Specification of Knowledge-based Systems.* The natural point from which to develop any software system is the creation of a specification. The specification ideally should detail every aspect of the system in unambiguous terms that all interested parties can consider. The creation of such a specification for knowledge-based systems is, however, a far from easy task for any but the most trivial of systems. In light of this problem, knowledge engineers often have been forced to proceed with only a minimal specification or no specification at all. This is a less than ideal situation and a source from which many subsequent developmental problems emanate. To overcome the problem of weak specifications in knowledge-based system development we advocate the use of two techniques: *prototyping* and *composite-specifications* through formal methods.

The first of these techniques, prototyping, is utilized to achieve the creation of a baseline document: the *initial specification.* Following Miller (1990), this phase utilizes prototyping to create an initial specification, and this phase does not end until all parties (customer, developer, user) agree that they finally understand what the system is intended to do, and in particular how it is supposed to do it—what Miller terms "The Operational Concept" (Miller, 1990).

The prototype process is primarily intended to establish the boundaries of the solution space. It is very important that the prototyping is used only to this end, as it is extremely detrimental to consider the more complex development issues at this stage, for example, representation, interface, and so forth, as these decisions would be made on incomplete knowledge of the domain and environment.

Embedded within the initial specification development process is an aspect of *cognitive engineering* known as *cognitive task analysis* (Roth & Woods, 1989), where:

Cognitive Task Analysis is used to derive a description of the cognitive demands imposed by a task and the sources of good and poor task performance. (p. 217)

The aim of cognitive task analysis then can be seen as an attempt by the knowledge engineer to:

Define what makes the domain problem hard, what errors domain practitioners typically make and how an intelligent machine can be used to reduce or mitigate those errors or performance bottlenecks. (p. 246)

The use of cognitive engineering techniques is not, however, limited to the creation of the initial specification. Wielinga, Breuker, and others have created a development methodology KADS (Breuker & Wielinga, 1987; Hesketh & Barrett, 1990) that also attempts to model expertise, such that it can be utilized in a knowledge-based software development project.

We shall utilize other cognitive engineering practices later in the methodology to assist in the assessment of validation and verification, quality assurance, in the selection of a representation as well as developing the system interfaces.

The creation of an initial specification provides the knowledge engineer with the first specification in the creation of the composite-specification of the system. A composite-specification is a set of specifications, each of which focuses upon an aspect of the development process: domain specification, representation specification, and so forth, the composite of which enables an approximation of a total specification for the system to be made. Figure 3 illustrates the six areas where specifications can be derived in a knowledge-based system, to varying degrees of formality.

From Figure 3 we can see that there are two distinct types of specification present: The *dynamic* specifications and the *static* specifications. Dynamic specifications refer to aspects of the system that are under constant change or for which the interaction of the components are undetermined due to their combinatorial complexity. Static specifications refer to those aspects that do not change, but rather remain consistent over a period of time. Thus, there are five static specifications:

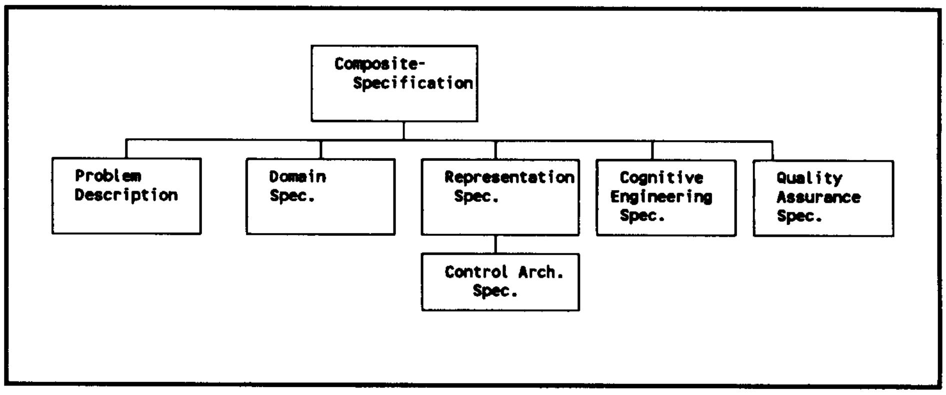

FIGURE 3. A composite specification.

6.7.1.1. *The specification of the domain knowledge.* We can easily model this aspect as the knowledge elicited from the domain expert(s) and knowledge source(s) is finite, and through the use of transformational processes, this can be specified formally in a language such as "Z" (Spivey, 1990). From a specification in a language such as this, we obtain several advantages. First, the Z notation in which it is written is clear, concise, unambiguous, and allows for both a technical and nontechnical readership. Second, the use of a formal notation has significant maintenance benefits, such as allowing knowledge engineers to keep a correct document of the domain information in an implementation-independent form, allowing the implementation language to vary if necessary.

6.7.1.2. *The specification of the representation.* The aim of this specification is to allow the knowledge engineer an opportunity to consider and identify those aspects of the knowledge representation language to be used and specify them in a formal manner. The selection of a representation is a difficult consideration that we shall discuss further in the next section; however, once a representational form has been selected, it is imperative that this be specified fully in terms of its denotational semantics and its syntax. For without these, it is extremely difficult to reason about a domain description/representation with any certainty. Included in this specification is the *specification of the control architecture* to be used in the system.

6.7.1.3. *Specification of the cognitive engineering aspects.* The cognitive engineering aspects of the system definition are those that involve:

- specification of the man–machine interface
- cognitive task analysis
- knowledge-encoding
- competence modeling
- performance modeling

The man–machine interface can be subjected to formal specification techniques, as demonstrated by Sufrin and He (1990), who use the Z notation to specify an interface, and Jacob, who formally specifies a man–machine interface (Jacob, 1983). The Z notation is again a superior form of specification to the pseudo-code, or natural language descriptions that are usually used.

As mentioned, cognitive engineering has an impact upon many aspects of system development, from the initial specification, through acquisition, elicitation, quality assurance to validation and verification. We shall consider each of these aspects later.

6.7.1.4. *Specification of quality assurance.* We can consider the quality assurance methods we wish to employ in the validation and verification of the system. These have to be defined both in terms of the techniques involved and the boundaries that are acceptable to the project.

In addition to the static specifications, there are those aspects that are dynamic in nature. The principal aspect of this type is the *specification of the problem description.* The nature of the techniques for formally specifying systems is, however, limited to static aspects of a system and does not facilitate full specifications of the dynamic aspects; thus we are unable to define the system in its entirety; this necessitated the utilization of prototyping in the creation of the initial specification in addition to the utilization of the composite-specification technique and the design tenets identified earlier.

Thus, we have identified the need for specifications in the creation of knowledge-based systems. We now discuss how these specifications can be brought together through the use of a rigorous development methodology.

The methodology as a whole can be introduced by considering Figure 4. These stages are now examined in greater detail.

As we have already seen, the initial specification is a document that can act as a baseline for the remainder of the systems development. Each of the resultant phases can be compared to the objectives and system specifications laid down in this document.

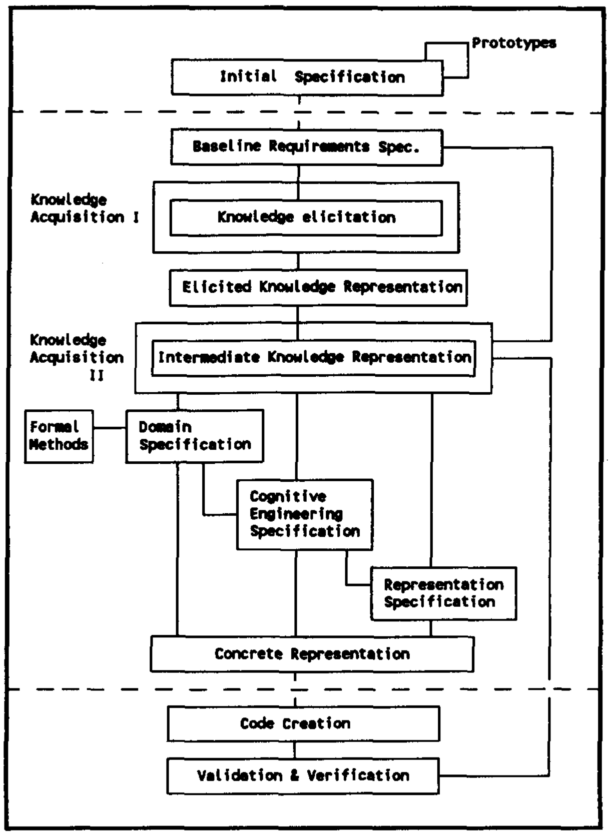
Prototypes
Initial Specification
Baseline Requirements Spec.
Knowledge Acquisition I
Knowledge elicitation
Elicited Knowledge Representation
Knowledge Acquisition II
Intermediate Knowledge Representation
Formal Methods
Domain Specification
Cognitive Engineering Specification
Representation Specification
Concrete Representation
Code Creation
Validation & Verification

FIGURE 4. Design of knowledge-based component.

6.7.2. *The Knowledge Elicitation Process.* The unique nature of knowledge-based systems is that they utilize domain-specific information that is "expert" in nature. This has several implications. The information may in itself be unique, scarce, or uncommon; however it is the way that the expert employs that information that makes the information valuable. Thus, one of the most important tasks befalling the knowledge engineer is to ensure that he or she elicits as much structural, control, and relational knowledge from the expert source as possible. Thus, this forms the basis of the knowledge elicitation task and the knowledge-based system development process itself. As later in the process, the knowledge engineer will have to consider specifying the static domain knowledge (facts, rules, heuristics etc.) and select a representation in which to manipulate this knowledge, which entails consideration of such factors as structural, control, and hierarchical knowledge types. Thus the knowledge engineers task in knowledge elicitation can be seen as falling into two categories:-

- elicitation of static knowledge
- elicitation of dynamic knowledge

The knowledge engineer has several different approaches to the knowledge elicitation process, (Roth & Woods, 1989; Welbank, 1983), for example:

- verbal transfer of knowledge, for example, interviewing—structured, focused and unstructured
- reporting techniques, for example, on-line, off-line, and hybrid
- psychological techniques, for example, repertory grid, critical incident, inference structure, goal decomposition, and distinguishing evidence
- knowledge engineer investigates literature

The choice of elicitation technique depends heavily upon the domain under consideration, the type of knowledge to be extracted, and the point the elicitation has reached. For example, the elicitation may commence with the knowledge engineer performing a series of unstructured interviews to extract high level conceptual knowledge. This may then be followed by structured interviews where the relationship of the domain, its structure, and more detailed information are obtained. This may then be followed by a series of focused interviews to fill in the low level information of a fine grain size. Several frameworks for the analysis of these techniques have been proposed (Burton, Shadbolt, Hedgecock, & Rugg, 1987; Dhaliwal & Benbasat, 1990), including *cognitive mapping and knowledge encoding,* two aspects of Woods's cognitive engineering paradigm (Roth & Woods, 1989; Woods & Roth, 1988).

The result of the elicitation process, depending upon the technique employed, will be what we have termed the elicited representation. This will be for example, a transcript in the case of an interview or an on-line report. The aim of this stage in the life cycle is to provide a permanent record of the knowledge in the form in which it was extracted. This will enable the knowledge engineer to follow a knowledge trail later in the process if necessary (e.g., maintenance phase).

The process of eliciting the different knowledge types, perhaps from different sources, with differing knowledge levels, using different techniques at different periods of time means that there will be a set of elicited representations that together form a historical data base of elicited knowledge.

6.7.3. *The Intermediate Representation.* Having created an elicited representation, we then have to analyze the knowledge that is embedded within it. The aim of this phase is to produce a new representation—the *intermediate representation*—the primary function of which is to provide a mechanism that is rigorous enough to allow several demanding analyses to take place upon it. One of these ultimately produces a formal specification of the domain knowledge, and another acts as the basis for the selection of the high level "classical" representation such as a production system, which ultimately will be used to represent the domain knowledge held in the formal specification.

The reason that the elicited representation is not used directly as the basis of these analyses is that the elicited representation may be in a form that is far too ambiguous in nature. Also, the elicited representation may be noisy, suffer from problems of continuity, lack modularity, or have poor linkage between areas of knowledge, problem areas that are not always apparent in their original form.

As stated, the aim of this phase is to produce from the elicited knowledge a more rigorous, intermediate representation (Scott, 1991). This representation will be structured in form, syntax, and semantics, such that the knowledge acquisition necessary to transform the elicited representation will identify the inconsistencies, incompleteness, and any incorrectness in the elicited representation. The knowledge engineer will use this intermediate representation to draw together the knowledge from the varying elicited forms: transcripts, repertory grids, questionnaires, and so forth. Intermediate representations are of the form: decision tables and/or graphs, decision trees, each of which encourage completeness, correctness, and consistency, and allow for refinement and reduction while having clean yet concise structures.

6.7.4. *Domain Specifications.* The intermediate representation provides us with a more rigorous form than the elicited representation with which to reason about the domain. This reasoning takes three directions: the creation of a domain specification, the creation of a cognitive-engineering specification, and the creation of the representation specification. The first of these specifications, the domain specification, is intended to provide a specification that focuses exclusively upon the domain knowledge, the static knowledge of rules, facts, and heuristics. The aim of this specification is to allow a knowledge-based system to have a repository from which the domain can be considered in isolation. This has several advantages, for example, in the course of maintenance or subsequent system updates the domain specification will be the unique location for the domain knowledge to be added, deleted, or modified. The knowledge engineer will be able to maintain the correctness, completeness, and consistency of the system as far as possible. These changes then can be traced throughout the remainder of the development process. The formalized procedures for updating the domain specification also can be specified for added rigor.

The domain specification therefore will have to have mathematics as its basis, and this leads to the adoption of the Z notation, a formal specification language. Specifications in Z consist of formal text and natural language text. The former provides a precise specification, whereas the latter is used to introduce and explain the formal parts. Specifications are developed via small pieces of mathematics that are built up using the schema language to allow specifications to be structured. This leads to formal specifications that are more readable than a specification presented in mathematics alone.

It is also advantageous to have a domain specification from the perspective of knowledge engineer-user-domain expert communication, as the specification can act as a medium for communication. Thus, it can be seen that the use of a formal language in the development of a knowledge base is very advantageous.

6.7.5. *The Cognitive Engineering Specification.* The cognitive engineering specification, as we have already noted, is composed of many aspects, including cognitive task analysis, knowledge-encoding, competence, and performance modelling. The combined effect of utilizing these cognitive components is very powerful and can be considered as a chief factor in maintaining the semantic correctness of the system as a whole, filling the gaps in the decision-making process. This can be seen as aspects of differing phases feed into each other. For example, the knowledge contained in the domain specification can be considered in light of the knowledge-encoding techniques, and this has an impact upon the choice of representation in the representation specification, which in turn will determine the systems' ability to manipulate domain knowledge.

The cognitive engineering specification also provides two models:

- The competence model that provides a model of the required competence expected from the model in the domain. (Roth & Woods, 1989)
- The performance model that describes the knowledge and strategies that characterize good and poor performance in the domain. (Roth & Woods, 1989)

The adoption of the cognitive engineering specification in these two roles then can act as the basis of a quality assurance mechanism for competence and performance.

6.7.6. *The Representation Specification.* The next step in the development methodology is to identify which (if any) classical or hybrid representation is the most suitable form around which to base the representation specification, where the classical representations are frames, production systems, semantic networks, and so forth. To find the most suitable form, several influencing factors have to be taken into account:

- information obtained from performing knowledge acquisition upon the intermediate representation
- information pertaining to representation selection that can be obtained from considering the composition of the domain specification
- information resulting from the cognitive engineering processes

Each of these information sources provides valuable insights on which representation would provide the best basis for the domain under consideration. The analysis of the intermediate representation will allow a coarse analysis of the underlying domain structure to be obtained. This is refined by considering the composition of the domain specification in terms of its knowledge and data types, their interrelationships, and structures. This is then enhanced by the cognitive mapping drawn from the cognitive engineering processes.

In order for a suitable match, the characteristics looked for in the intermediate representation, the domain specification, and the cognitive models have to be reengineered in the examination of the representation schemes themselves (rules, frames, etc). Once this is achieved, the match can then be made. The chosen representation is specified formally in terms of its semantics and syntax; this forms the representation specification (Craig, 1991).

6.7.7. *The Concrete Specification.* Having created the domain, cognitive, and representation specifications, we are now at a point at which these specifications can be combined into a form that will allow us to move toward implementation. This stage is known as the *concrete specification.*

The creation of the concrete specification is in stages. First, the domain knowledge is transformed from its Z specification into the form advocated by the representation specification, and second, a formal specification of the control architecture that is associated with the representation is created.

It should be noted that this is not the implementation as the representation is a hybrid between a high level version of what is to be implemented and a formal specification in the style suggested by the syntax and semantics of the representation specification (e.g., pseudo code), the aim being to produce an implementation-independent representation of the system. This will allow the knowledge engineer to have a simplified version (minus the complex syntax) with which to reason about the implementation later in the systems' life cycle, for example, maintenance.

6.7.8. *Coding.* Having created the concrete specification this is then used as the basis of the system implementation, with the interface issues resolved by referral to the cognitive engineering and man–machine interface specifications. The implementation of the system should be the most straightforward of all the stages, due to the high degree of structuring and refinement that has been performed upon the system in the previous phases. The mechanism through which the system is implemented is left open to the knowledge engineer as this is considered a trivial exercise once the specifications have been developed.

6.7.9. *Validation and Verification.* The area of validation and verification for knowledge-based systems is one of active research, and preliminary results have shown that the process of validation is an extremely difficult one. A discussion of the research in validation and verification is beyond the scope of this paper, see Liebowitz (1986), O'Leary (1987, 1993), Rushby (1988), and Culbert (1990). However, it should be noted that the representation refinement approach to development advocated by this methodology combined to the quality principles of the MM-Level model will strongly promote correctness.

6.8. Testing and Integration

Having completed the development of the knowledge-based component, the developer can now consider the integration of the system into any other system. An aim of this methodology is to minimize the amount of overhead involved in the process of embedding the knowledge-based component. This is the reason for the use of system-wide development standards, historical data bases, metrics, formal methods, and an adherence to integration throughout the system/process life cycle.

6.9. Institutionalization

An aspect of development that we have not yet addressed is that of institutionalization. This has been identified by Liebowitz (1991) and others as being the critical factor affecting system acceptance, usage, and ultimate success. The process of institutionalization can be broken down into three fundamental aspects: implementation, transitioning, and maintenance, all three of which have historically been weak when considered with respect to the case of expert system development. Liebowitz identifies four areas vital to the institutionalization process:

1. an awareness of expert systems for managers
2. user training strategies
3. user support service strategies
4. maintenance

All of these areas incorporate what Badiru (1988) terms Triple C—communication, cooperation, and coordination, important management aspects to the creation of an expert system. Thus, we see the process of institutionalization as the connecting link between the macro- and microlevels of our model, the influence that moves our methodology toward being a sociotechnical model (Dibble & Bostrom, 1987).

The institutionalization of system development can be considered as the *holistic* approach to development, where all levels of personnel, from users to managers, are involved in the development process, what Leonard-Barton terms "integrative innovation" (Leonard-Barton, 1987). The model we have proposed here attempts to overcome these problems of institutionalization through the participation of management in the macrolevel and instilling an awareness of the technology involved to them. Further, our model aims through the microlevel to actively involve the user/client in all aspects of the development. This is vital to the institutionalization process in that the technology transfer is greatly eased. This is apparent in many ways; the user becomes more understanding of the technology involved, the developer is relieved of the total obligation for system correctness as this is now shared with other members of the development team, and the probability of system success is increased if there is a continual involvement and thus continual feedback from all members and levels of the organization. Liebowitz has also identified other influencing factors that affect the institutionalization process, including the following:

- system migration
- standards
- configuration management
- testing
- user support services
- maintenance
- user training
- legal issues

We now briefly touch upon some of these areas as they relate to our model. However, for a fuller treatment the reader is referred to Liebowitz (1991).

The ability for the developers to perform system migrations is greatly eased by having a set of specifications from which to work. This facilitates the changes in platform that may occur over the system's life cycle. These specifications and the adoption of a rigorous development strategy also allow for close adherence to standards and subsequent changes in those standards. As the two-level model is also used for the development of conventional systems, the integration of standards is facilitated. This is also the case for configuration management; the methodology is intended to allow for the knowledge-based systems to be embedded into the conventional systems and as such minimize any special configuration management needs or criteria.

In following this two-level approach, in conjunction with the rigorous methodology, it has been our aim to maximize the system's correctness; however, a by-product of the representation refinement approach is to allow for, and support maintenance. By partitioning the specifications into their functional areas, the knowledge engineer can integrate any maintenance needs into the existing specifications in such a way as to assess the impact this will have upon the existing specification—thus maintaining integrity and correctness. As stated by Liebowitz (1991), "maintenance is a key issue in institutionalizing expert systems" (p. 98) and hence we have placed a heavy emphasis in designing our methodology to support this function.

The institutionalization process therefore can be seen as the link between the two levels of our model. It provides a basis for drawing together all the aspects of development on all levels to move toward the ultimate aim: successful deployment of the system.

6.10. Summary of Micro Perspective

The micro-perspective aims to promote the philosophy of continuous quality improvement through the utilization of life support tools and the utilization of TQM, metrics, modelling, and formal methods. The model does not explicitly include a formal quality assurance phase as the tenets of development have been integrated into all aspects of development. The model has also been designed to promote clarity of communication channels. The model is also able to accommodate differing development process needs, such as real-time, parallel, or as shown here a knowledge-based development component. This enables an integrated system to be developed through rigor and quality principles.

7. CONCLUSIONS

This article has aimed at showing the necessity to produce and follow a new philosophy for software development. We have shown how software development can no longer be considered from one perspective (that of the software development itself), but necessitates a two-level perspective that includes the external factors that influence development. If such an approach is followed, we feel that this will lead to a more complete framework from which systems can be built and monitored.

The two-level perspective, we argue, will be successful only if the developer follows certain tenets:

- utilize a philosophy of continuous quality improvement in all aspects of development through the Plan, Do, Check, Act cycle.
- integrate the *life support tools* into the development process, the basis of which are formal methods, metrics, quality variance
- utilize metrics to control, monitor, and understand the process
- utilize tools and resources to promote communication and improve the communication mediums
- utilization of an institutionalization policy throughout all aspects of development

Thus, we envisage the creation and development of a new philosophy of process-oriented software development, whose basis is the creation of quality software through multilevel, interdisciplinary models of the development environment, such as the MM-level process model we have described. These process models will be on several levels, linked together through strong institutionalization processes.

Finally, we have shown how the two-level model can be versatile in accommodating and integrating with other models such as the rigorous knowledge-based systems development methodology. This allows a consistent approach to be taken in developing complex systems that may be composed of several different component types, for example, real time, knowledge-based etc. This is, we feel, vital, as systems become increasingly hybrid in nature.

REFERENCES

Badiru, A.B. (1988). Successful initiation of expert systems projects. *IEEE Transactions on Engineering Management*, **35**, 186–190.

Boehm, B. (1980). Software engineering economics. Englewood Cliffs, NJ: Prentice-Hall.

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

Breuker, J., & Wielinga, B. (1987). Use of models in the interpretation of verbal data. In A.L. Kidd (Ed.), *Knowledge acquisition for expert systems: A practical handbook* (pp. 17–44). New York: Plenum.

Buchanan, B.G., Barstow, D., Bechtal, R., Bennett, J., Clancy, C., Kulikowski, C., Mitchell, T., & Waterman, D.A. (1983). Constructing an expert system. In F. Hayes-Roth, D.A. Waterman, & D.G. Lenat (Eds.), *Building Expert Systems* (pp. 127–168). Reading, MA: Addison-Wesley.

Burton, A.M., Shadbolt, N.R., Hedgecock, A.P., & Rugg, G. (1987). A formal evaluation of knowledge elicitation techniques for expert systems: Domain 1. In D.S. Moralee (Ed.), *Research and development in expert systems IV* (pp. 136–145). Cambridge: BCS Series, Cambridge University Press.

Conte, S.D., Dunsmore, H.E., & Shen, V.Y. (1986). *Software engineering metrics and models.* Menlo Park, CA: Benjamin/Cummings.

Craig, I.D. (1991). *Formal specification of advanced AI architectures.* Chichester, England: Ellis Horwood.

Culbert, C. (1990). Verification and validation of knowledge-based systems [Special Issue]. *Expert Systems with Applications*, **1**, (3).

Davis, R., & Lenat, D. (1982). *Knowledge-based systems in AI.* New York: McGraw-Hill.

Deming, W.E. (1986). *Out of the crisis.* Cambridge, MA: Massachusetts Institute of Technology, Center for Advanced Engineering Study.

Deming, W.E. (1991). System of profound knowledge. Notes from Deming Seminar 1992. Miami, FL.

Dhaliwal, J.S., & Benbasat, I. (1990). A framework for the comparative evaluation of knowledge acquisition tools and techniques. *Knowledge Acquisition*, **2**, 145–166.

Dibble, D., & Bostrom, R.P. (1987). Managing expert systems projects: Factors critical for successful implementation. *Proceedings* *of the 1987 ACM SIGBDP-SIGCPR Conference.* New York: ACM.

DOD-STD-2167A. (1988). *Military standard defense system software development.* Washington, DC: Department of Defense, SPA-WAR 3212.

Gitlow, H.S., & Gitlow, S.J. (1987). The Deming guide to quality and competitive position. Englewood Cliffs, NJ: Prentice-Hall.

Gitlow, H.S., Gitlow, S.J., Oppenheim, A., & Oppenheim, R. (1989). *Tools and methods for the improvement of quality.* Homewood, IL: Irwin.

Grady, R.B., & Caswell, D.L. (1987). *Software metrics: Establishing a company-wide program.* Englewood Cliffs, NJ: Prentice-Hall.

Grover, M.D. (1983). A pragmatic knowledge acquisition methodology. *IJCAI,* **8,** 436–438.

Hall, A. (1990). Seven myths of formal methods. *IEEE Software,* **7,** 11–19.

Hesketh, P., & Barett, T. (1990). *An introduction to the KADS methodology.* (Esprit Project P1098, Deliverable M1, EEC ESPRIT Project). Brussels, Belgium.

Imai, M. (1986). *Kaizen.* New York: Random House.

Ishikawa, K. (1982). *Guide to quality control.* Tokyo: Asian Productivity Organization.

Ishikawa, K., & Lu, D.J. (1985). *What is total quality control? The Japanese way.* Englewood Cliffs, NJ: Prentice-Hall.

Jacob, R.J.K. (1983). Using formal specifications in the design of a human-computer interface. *Communications of the ACM,* **26,** (4).

Juran, J.M. (1964). *Managerial breakthrough.* New York: McGraw-Hill.

King, W.R. (1978). Strategic planning for management information systems. *Management Information Systems Quarterly,* **2,** 27–37.

Leonard-Barton, D. (1987). The case for integrative innovation: An expert system at digital. *Sloan Management Review.* Cambridge, MA: MIT.

Liebowitz, J. (1986). Useful approach for evaluating expert systems. *Journal of Expert Systems,* **3**(2), 86–96.

Liebowitz, J. (1991). *Institutionalizing expert systems: A handbook for managers.* Englewood Cliffs, NJ: Prentice-Hall.

Miller, L. (1990). A realistic industrial strength life cycle model for knowledge-based system development and testing. *AAAI Workshop Notes: Validation and verification,* August 31st, Boston MA.

NASA COSTMODL. (1991). *COSTMODL: User's Guide.* NASA Johnson Space Center, Houston, TX.

O'Keefe, R.M., Balci, O., & Smith, E.P. (1987). Validating expert system performance. *IEEE Expert,* **2**(4), 81–90.

O'Leary, D.E. (1987). Validation of expert systems with applications to auditing and accounting expert systems. *Decision Sciences,* **18,** 468–486.

O'Leary, D.E. (1994). *Collected papers 1989–92: AAAI Workshops on Validation & Verification.* (In Preparation).

Plant, R.T. (1994). A rigorous development methodology for knowledge-based systems. *Communications of the ACM* (To Appear).

Plant, R.T., & Hu, Q. (1992). The development of a prototype DSS for the diagnosis of casting production defects. *Computers & Industrial Engineering, An International Journal,* **22,** 133–146.

Plant, R.T., & Salinas, J.P. (1992). CISEPO [City Selection Program]: A DSS for relocating companies within the U.S. *Computers, Environment and Urban Systems,* **16,**(2), 117–130.

Roth, E.M., & Woods, D.D. (1989). Cognitive task analysis: An approach to knowledge acquisition for intelligent system design. In G. Guida & C. Tasso (Eds.), *Topics in Expert System Design* (pp. 233–264). Amsterdam: Elsevier Science Publishers.

Royce, W.W. (1970). Managing the development of large software systems. *Proc. WESTCON, Ca.* USA.

Rushby, J. (1988). *Quality measures and assurance for AI software.* NASA Contractor Report 4187.

Scott, C. (1991). *A practical guide to knowledge acquisition.* Reading, MA: Addison-Wesley.

Sheppard, M. (1990). Design metrics: An empirical analysis. *Software Engineering Journal.* IEEE, 3–10.

Spivey, J.M. (1990). *The Z notation.* Englewood Cliffs, NJ: Prentice-Hall.

Sufrin, B.A., & He, J. (1990). Specification, analysis and refinement of interactive processes. In M. Harrison & H. Thimbleby (Eds.), *Formal methods in human-computer interaction* (pp. 153–200). Cambridge, MA: Cambridge University Press.

Tausworthe, R.C. (1981). Deep space network cost estimation model (Publication No. 81-7). Pasadena, CA: Jet Propulsion Laboratory.

Tsoumpas, P. (1994). Towards a methodology for quality software development dissertation (in preparation), Coral Gables, FL: Dept. of CIS, University of Miami.

Weinberg, G.M., & Schulman, E.L. (1974). Goals and performance in computer programming. *Human Factors,* **16**(1), 70–77.

Weitzel, J.R., & Kerschberg, L. (1989). Developing knowledge-based systems: Reorganizing the system development life cycle. *Communications of the ACM,* **32,** 482–488.

Welbank, M. (1983). A review of knowledge acquisition techniques for expert systems. British Telecom Research Labs. Martlesham Consultancy Services, Ipswich, England.

Wielinga, J.B., & Breuker, J.A. (1983). *Analysis techniques for knowledge-based systems: Part 1* (Report No. 1.1). Esprit Project 12, Amsterdam, Holland.

Woods, D.D., Potter, S.S., Johannesen, L., & Holloway, M. (1991). *Human interaction with intelligent systems: Trends, problems, new directions* (Technical Report). Cognitive Systems Engineering Lab, Columbus, OH. CSEL 91-TR-01.

Woods, D.D., & Roth, E.M. (1988). Cognitive systems engineering. In M. Helander (Ed.), *Handbook of human-computer interaction.* New York: North Holland.

Zuse, H. (1992). *Software complexity: Measures and methods.* Berlin: de Gruyter.