QUEST: Quality of Expert Systems

Michael Perre
TNO Physics and Electronics Laboratory
Information Technology Division
Command and Control Systems Group
P.O. Box 96864
2509 JG The Hague
The Netherlands

1 IN QUEST OF QUALITY

TNO Physics and Electronics Laboratory, in collaboration with the University of Limburg and the Research Institute for Knowledge Systems, worked on a technology project named 'QUEST: Quality of Expert Systems' [FEL90]. QUEST was carried out under commission of the Dutch Ministry of Defence.

A strong motivation for this research project is the fact that more and more conventional systems contain intelligent modules, without the assurance that these modules satisfy the same rigorous quality measures as the conventional ones do. In comparison with conventional software systems the quality of expert systems is viewed as being not very satisfactory. Some of the more problematical aspects are knowledge acquisition, testing, evaluation and the maintenance of the knowledgebase. As yet there is not much unanimity with regard to the ways in which these problems can be tackled. This is an objectionable state of affairs, especially when you are dealing with critical applications, e.g. process control systems in oil refineries or nuclear power plants. The same argument is also valid for military Command, Control, Communications and Intelligence (C^3I) systems . A characteristic of these systems is that they consist of large databases with which the deployment of men and material is coordinated.

It needs no argument that statements about quality of software can only be made when the concept 'quality' has been defined and made measurable. Whether a piece of code has 'a good quality' is difficult to establish. The absence of quality, on the other hand, is much more obvious. The purpose of this paper is to set up a framework in which the quality of expert systems can be captured. There are three important aspects regarding the quality control problem:

- Development process of an expert system;
- Analysis of an object system resulting in specifications;
- The expert system as a product.

The relation between these three aspects is reflected in the following formula [FEL90]:

PRODUCT = f{DEVELOPMENT PROCESS(SPECIFICATIONS)}.

Applying the development process on the specifications results in the product 'expert system'. Quality can be controlled in three ways:

- A structured development process;
- Validation and verification of the system specifications;
- Testing of the product.

Reliability and maintainability criteria should be emphasized.

This paper attempts to present a survey of methods and techniques available to control the quality of expert systems and focuses on that part of system development that sets expert systems substantially apart from other software: the specification and implementation of the knowledgebase. Starting point of this discussion is the thesis that a knowledgebase can be viewed as a collection of facts which can be manipulated with intelligent rules. These rules are also stored as objects in the knowledgebase [Nijssen86].

By considering a knowledgebase as a special kind of database, various facilities of database management systems can be used in knowledgebase management systems. Examples are recovery (to restore a knowledgebase after a calamity, fault or power failure), concurrency (simultaneous utilization of a knowledgebase by different users), distribution (physically distribute a knowledgebase over different locations), security (protect a knowledgebase against unauthorized usage) and integrity (guard against inconsistencies of the knowledgebase) [Date83]. Especially this last point enables a direct relation with analysis and design methods of databases. Consequently, there is a need to build a conceptual model of a knowledge domain [Nijssen89].

2 THE LIFE-CYCLE OF AN EXPERT SYTEM

Most development methodologies for expert systems emphasize the knowledge acquisition phase. This is no wonder considering the degree of complexity intrinsic to this activity. However, the development of an expert system is not limited to this phase only. The necessity of a structured development process has long been acknowledged, an example is System Development Methodology (SDM) [Turner87]. Since 1987 there exists a version of SDM which is used primarily for expert system development: Structured Knowledge Engineering (SKE) [SKE88]. In this methodology more specific expert system activities like knowledge acquisition are worked out in detail. Another promising approach is that of Weitzel and Kerschberg, who are also proponents of so called expert database systems [Weitzel89].

SDM is a phasing methodology for the management of system development projects. SDM deals with the whole of a system's life-cycle, from information planning to its use and maintenance. The SDM manual is a guideline for managing projects, arranging the documentation and the development and management of information systems. For every development phase, the activities to be performed, are included. For every activity are registered: a description, one or more analysis and design methods, a number of deliverables or documents and literature references. Especially SKE makes a (concise) statement, based on SDM, about the arrangement of the other phases, such as technical design, implementation, testing, acceptance and maintenance. The advantage of the SKE approach is that it embraces a familiar, conventional method of development, which is SDM; this enhances the chances of actual application.

It is expected that a development method for expert systems incorporates a good phasing, leads to clearly defined intermediate and final products, can be applied iteratively, allows for prototyping and provides a well-defined testing and evaluation phase. Though this list is rather ambitious, Kerschberg's method appears a good candidate. Important is that this author sees 'systems based on knowledge' as an evolutionary development of conventional systems. It is no wonder therefore, that he is a staunch proponent of a fusion of database and expert systems.

Kerschberg's development method relies considerably upon techniques such as prototyping, without ignoring the development of a conceptual model of the problem domain. In order not to become stuck in sequential development, the individual phases of this method consist of 'processes' that can be activated, deactivated and reactivated. A great deal of attention is paid to

Contrary to black box testing, white box testing uses the structure of the source code. The thought behind the white box techniques presented by Myers, is that the ideal set of test cases will literally cover every nook and cranny of a program. This ideal can best be approached using path coverage. In path coverage, every possible path of execution in the program is covered by (at least) one test case. In general, the demand for a path coverage leads to unacceptable costs (it must be kept in mind that a path that runs through one or other repetitive loop differs from the path that runs through one and the same loop for three times). Therefore, one has established a set of coverage criteria, that will be discussed in order of their ascending demands:

- Statement coverage: every statement must be executed during testing at least once;
- Decision coverage: every branch (the result of a decision) must be run through during testing at least once;
- Condition coverage: every 'condition' (term in a logical expression) must at least yield one instance of true and one of false during testing;
- Decision/Condition coverage: both decision coverage as well as condition coverage must be satisfied;
- Multiple condition coverage: during testing, the tuple of 'conditions' of 'decision' must run through every possible value at least once.

It is reasonable to assume that non-nested decisions can be examined independently from each other. This is not true, however, in the case of nested decisions, especially when multiple condition coverage is demanded. [Myers79] does not give a clear statement on this. Nevertheless, it is repeatedly indicated that multiple condition coverage is a necessary prerequisite for proper testing. When you consider that nested decisions may be seen as a single decision, a maximum number of conditions of about 20 would not seem impossible for extensive applications. This would come down to a magnitude of about a million test cases. It is therefore safe to assume that Myers concludes that conditions in nested decisions do not influence each other or at least not always.

4.2 Applicability of conventional testing methodologies

Considering the statements in various publications about the intricate flow of control, impairing proper testing of expert systems, and the 'complications during the testing of data-dominated systems', it is concluded that the specific problems inherent to testing expert system belong to the declarative part (the knowledgebase(s)) and not to the inference engine (shell) [Llinas87 Myers79, Deutsch82]. It is held, in general, that the advantage of explicit (declarative) knowledge representation over implicit (procedural) lies in the fact that shell and knowledgebase may be tested separately. During the discussion of the applicability of the aforementioned white and black box testing methods, it is assumed that the shell will contain no bugs (anymore).

Equivalence classes, the leading concept in black box testing can only be useful in a specification that is either detailed or will be (almost) identically approached by different persons. In black box testing, it is also desirable to that the specifications of the boxes to be tested, is not too large, meaning that the number of test cases necessary is not too large. [Doyle88] remarks that 'in the AI approach (...) the interpreted specification functions as an inexpensive, but complete prototype'. Here, the knowledgebase is considered to be a (sub)specification that together with the (conventional) specification of the shell, forms a specification of the expert system. A demand that must be part of every specification is that its validity can be checked without problems. Should the knowledgebase be disorderly, however, then this will not be possible. Main question now is of course whether in such a case a specification of the knowledgebase can be formulated that can alleviate the testing and maintainability problems observed, or eliminate them altogether. Doyle's

elementary assertions that are no facts; they define which facts can be part of communicable knowledge and which cannot. All facts are incorporated in the database, whereas the grammar contains a description of all the rules.

An important type of rules are the constraints. These are rules that are no specification of object types or factual types; they restrict the facts that are allowed on the basis of the definition of object types and factual types. Constraints, in their turn, can be subdivided in static rules and transition rules. Static rules define which facts may be present in the database at any given time; they describe states. Transition rules define which changes are allowed in the database; they describe transitions from one state to another.

In general, the information/knowledge analysis with the help of ENIAM is conducted along the following lines. Reality must first be described in words. Then, this description must be unwound into elementary assertions. Subsequently, these elementary assertions are divided in facts and rules. The latter can be formally described.

4 TESTING AN EXPERT SYSTEM

4.1 Black and white box testing

Methods to guarantee the reliability of software can be subdivided into methods that 'embed' (enhanced) reliability (e.g. structured analysis, design and implementation methods) and those that test reliability afterwards. The overlapping notions testing, evaluating, validating and verifying belong to this last group. Over the years, an extensive range of methods, paradigms and hints has been developed for testing software. Starting principle of this paragraph is a testing method as described in [Myers79]. Whereas this method is very useful for testing conventional programs like interpreters and operating systems, it seems to be inadequate for expert systems and (to a somewhat lesser extent) other data-governed systems such as databases [Deutsch82].

Testing can be done mentally and by executing a piece of software. To the mental techniques belong, among others, inspection and walkthrough. Walkthrough comes down to investigate how the interpreted program will react to certain test cases using the source code. As in every form of testing, the expected/desired output must be specified before actual testing. Inspection assumes two different forms: a verbal paraphrase of the programmer of what the program does (using the source code) and running down a check-list with likely errors.

Testing through execution also consists of two types, i.e. black box and white box tests. The first type considers the system or 'module' to be a black box: only the result (the output) is important when feeding certain input. How this software establishes its output is of no importance to black box tests. Because it is virtually impossible for most software to try all possible input values, it is attempted to compile a set of test cases that is as efficient as possible. Efficient here means: small and with a large probability of errors manifesting themselves. Experience teaches that the value range of input can be divided into so-called 'equivalence classes', characterised by the fact that two test cases from the same equivalence class will give rise to the same error (if resulting in an error at all). Because of the heuristic character of the use of equivalence classes and boundary values it cannot be determined objectively, what exactly is the 'best' set of test cases. Furthermore, an optimum state is always a balance between enhancing the error-finding probability and limiting the time involved in testing.

derivation and inference rules) in what is called E(xtended)NIAM. ENIAM not only offers the possibility (as does NIAM) to define 'set-oriented' constraints graphically, but also 'member-oriented' ones. Examples of constraints based on sets are unicity, exception, and totality rules. A member-oriented constraint makes a statement about the occurrence or nonoccurrence of a member within a certain set. The extension is based upon 'existential graphs' [Sowa84], a graphic representation of logical statements. A small number of symbols is used that have the same expressive power as first order predicate logic.

ENIAM offers possibilities of a more extensive integrity control, because the representation of constraints makes it 'simple' to generate executable Prolog programs. These programs (being the exemplification of the constraints) can be, with the help of algorithms that will be dealt with later in this paper, tested for consistency and completeness. This aspect has already been the subject of investigation by [Creasy88].

3.2.2 Conceptual modelling with ENIAM

Vital to NIAM (and ENIAM) is the principle of describing reality with the help of a natural language. In the development of knowledge systems, use is made of knowledge/information that has been recorded in writing (e.g. manuals) on the one hand, and on the other hand, knowledge that is based more on the experience of an expert. The intention of knowledge acquisition is to add a structure to this jumble; a conceptual model of the knowledge domain must be designed.

Reality consists of a set of objects that are relevant within a certain knowledge domain. This thesis, based on [Wintraecken85], implies that information or communicable knowledge is nothing more than making a statement or an assertion about objects within this reality. A fact is the elementary assertion that establishes a relationship between objects in this reality. A feature of an elementary assertion is that it only contains a single fact. Every non-elementary assertion can broken down into elementary assertions.

The objects in an elementary assertion can be divided into lexical and non-lexical ones. Lexical objects can somehow be represented in the form of letters, numbers and/or other symbols and may be seen as names or references to other objects. For a non-lexical object it holds true that it cannot be represented. The elementary assertion: ' Klaassen is the family name of an employee' contains the name 'Klaassen' as a lexical object and 'employee' as a non-lexical object to which reference is made by means of the name.

ENIAM substantiates reality with the help of binary facts, i.e. facts in which two objects from reality always play a role. A distinction is made between two types of binary facts: bridges and ideas. A bridge is a fact that relates to a lexical and a non-lexical object; it forms, as it were, a bridge between two 'worlds'. An idea is a fact that relates to two non-lexical objects; the name idea was chosen because non-lexical objects are 'discussed' without the lexical objects that determine them. The example in the last paragraph is a bridge.

One can also speak of types: object types (lexical and non-lexical) and factual types (idea and bridge types). Introducing this abstraction, it becomes possible to refer to a set or class of equivalent elements. The assertion: 'There are family names' is an example of the specification of a lexical object type. The assertion 'There are employees' specifies a non-lexical object type.

ENIAM is based on the assumption that knowledge can be described with facts and rules. Facts can be seen as knowledge concerning a certain reality that may change in time. Rules are

does not say anything about the order in which these sources of knowledge must be activated, nor the conditions under which this happens.

The universal structure that KADS imposes on all conceptual models is a four-layer organization. The domain layer contains knowledge about objects (concepts) of the actual domain and their mutual relationships. The inference layer specifies what inferences are possible. The task layer specifies in what situation and order inferences must be executed. The strategy layer, finally, contains the knowledge that is necessary to switch to another solution strategy if the present is unsatisfactory. Breuker and Wielinga remark that this layer in reality is not often used [Breuker88a].

3.2 ENIAM

3.2.1 Extended Nijssens Information Analysis Methodology

Nijssens Information Analysis Methodology (NIAM) is an example of a data-oriented method [Wintraecken85]. NIAM uses three notions:

- Knowledge is everything someone knows about a certain subject;
- Information is knowledge that can be transferred;
- Communication is the exchange or transfer of information.

That part of the real or abstract world to which the information exchanged through communication relates, is called 'reality' or 'Universe of Discourse'. When we limit ourselves to communicative processes that use an information processing system, then the information has to be represented in the communication medium. Communication between humans using an information processing system is only practical if the data for the information to be transferred is predefined. These matters are described in a so-called conceptual grammar. The information analysis process may now be seen as an activity 'where information, exchanged during communication processes, is analysed and a (conceptual) grammar for these communication processes is formulated, based on this analysis [Wintraecken85].

In this way, NIAM offers the possibilities to define the specifications in a formal manner. The analysis phase yields a conceptual grammar that is transformed during the design and implementation phase into a computer based information system. The conceptual grammar describes three aspects:

- Communication, or information flows;
- Transformation, or functions that influence data elements;
- Logical data storage method.

An important activity during knowledge system development is the definition of a conceptual model. The conceptual model must be a consistent representation of the knowledge domain in which, just like in database applications, a distinction is made between a knowledge scheme (the definitions of all existing facts and relationships) and the actual knowledgebase. The advantage is that now the knowledgebase can be discussed in abstract terms. Consistency and completeness can be controlled by means of constraints that are imposed upon the actual knowledgebase. A conceptual model must be made using a method that explicitly defines the distinction between facts and their definition.

Meanwhile, colleagues of Nijssen, the developer of NIAM, have added extensions to this method [Creasy89]. The core of these is formed by the possibility to formally represent constraints (e.g.

activities such as refining, redefining and restructuring. The Kerschberg method incorporates the possibility to integrate the development of expert systems into a larger, comprehensive system.

3 KNOWLEDGEBASE SPECIFICATION METHODOLOGIES

The purpose of this paragraph is the introduction of two methodologies to consistently and completely specificy the knowledgebase of an expert system, KADS and ENIAM. The manner in which a certain knowledge domain of an expert system is specified does not really differ from the way it is done in the development of a database system. This means that the knowledgebase, being the central part of an expert system, is a replication of a knowledge domain in terms of a model. The semantics of the knowledge contained in the knowledgebase must be specified unambiguously. In other words: the conceptual model is a semantic definition of the knowledge domain.

3.1 KADS

The adherents of KADS [Breuker87] oppose the traditional AI approach to knowledge acquisition prototyping. Breuker and Wielinga maintain that prototyping, because it forcibly imposes abstraction, yields a unstructured knowledgebase without the conceptual insights, necessary for explanation and maintainability [Breuker88a]. They claim that this holds true especially for knowledge domains of non-trivial complexity or size.

The innovative angle of KADS lies in the fact that it promotes choosing an epistemological framework (from a list of available frameworks used earlier, or parts thereof) as a first step in knowledge acquisition. Subsequently, one tries to place the accumulated knowledge within the framework. The framework chosen (in KADS this is called interpretation model) is a domain-independent abstraction of domain-dependent (but implementation-independent) conceptual models.

For this purpose, a typology of generic tasks has been established. A generic task is a (domain-independent) abstraction of an inference process that may be seen as a conceptual entity. Clancy's heuristic classification is an example of an elementary, generic task [Clancey85]. KADS task typology sees heuristic classification as a form of singular diagnosis, that in its turn is a form of diagnosis etc. An assumption of KADS therefore is that inference processes can be abstracted within a specific domain (i.e. stripped from its domain-specific notions) that can still be described clearly enough to recognize an equivalent as such in another domain. The 'subjective impressions' of those that have used KADS in practice, partially support this but there are no (more) substantial indications [Breuker88a].

Ideally, an interpretation model consists of an inference structure and a task structure. The first defines what the 'primitive' inferences are (in terms of name, inputs, outputs and a very broad description). The task structure defines how these basic inferences should be linked together to execute the task in question. Unfortunately, KADS has failed to work out the task structures of interpretation models. The task structures used in actual applications appeared to be too diverse. In point of fact this means that KADS interpretation models purely consist of inference structures with concise textual information added. Primarily, they have a declarative (non-procedural) character. An interpretation model purely indicates what type of objects ('meta-classes') function as input or output of the appropriate types of primitive inferences (sources of knowledge) and

statement cited previously seems to hold true for the rule part of an expert system. In such a situation black and white box methods coincide. The only way then, to enhance the orderliness of product alias specification is to reformulate the rules, using, for instance, other domain concepts or a totally different approach producing more modularity. It goes without mention that generally speaking, this will not be a simple task.

The use of boundary values in test cases will, in general, be less fruitful in expert systems, simply because the variables are more often symbolic (and thus discrete). Illegal values, however, can play a very useful role, especially there where making test cases gets complicated because the maker is not able to supply the desired (correct) output to the input.

When an expert system is subjected to multiple condition coverage (about nested conditions), then this will soon lead to an unacceptably high number of test cases. The fact that an expert will usually have to assist in connecting the desired output to the input of the test cases makes the case even worse. After all, the availability of experts is a familiar bottle-neck in the development of expert systems.

Holding on to limited multiple condition coverage (i.e. within a decision) is, however, dubious in many cases, especially in the case of a decision tree. A subdivision (preferably hierarchically) of the knowledgebase in small modules is therefore desired. Multiple condition coverage can then be demanded in every individual module. If these modules are not too large and legitimately claim the title of module, in other words, form conceptual units without too much interaction, then such an approach is undoubtedly optimal. Which demonstrates the vital importance of structuring and modularity to testability. Needless to say perhaps, is the remark that the aforementioned list of demands has been written down without taking into account its feasibility. It is merely a sketch of the structure that would be ideal for testing and maintenance. In practice things would be too rigid (in hierarchical structuring, for instance) to serve as an effective guideline.

4.3 Structure and modularity

In a traditional sense, a software design is properly structured, when the modules possess a certain independence from each other and every module is coherent from a functional point of view, i.e. actions within the module are related to each other. The first aspect is called 'coupling', the second is called 'cohesion'.

At least as far as the definitions in [Gane86] and [Yourdon79] go, coupling and cohesion are characteristic of procedural software. This does not mean, however, that these cannot relate to knowledgebases: when a knowledgebase is subdivided in modules it is very well possible that a change in one module causes changes in other modules, to keep the knowledgebase consistent as a whole, for instance. In fact, [Nguyen87]'s algorithm for consistency control uses a modular subdivision of the knowledgebase to perform a check more efficiently, be it that the subdivision into modules is done afterwards (i.e. after the development of the system). Because this is done purely on the basis of dependency between predicates without taking into account the conceptual unity of modules, there is no procedural cohesion involved. It cannot be determined whether the coupling of the modules in the context tree is data or control-coupled. This depends on whether during the inference process, new (derived) facts are being stored.

In blackboard architectures with a single (not divided in panels) blackboard, the coupling between the separate knowledgebases is in principle a form of 'common environment' coupling.

Usually, blackboards are divided in panels and for every knowledgebase it is specified what panels may be read and which may be written on. In that case, the coupling may be better described as object-oriented. In both cases, this basically involves data-coupling in the sense that a change in a knowledgebase produces no unwanted side-effects in another knowledgebase, at least not if the design of the knowledgebases has not taken the functionality of those other knowledgebases into consideration. An exception to this is changing the access rights (read/write) of a knowledgebase on the various blackboard panels. Assuming that the access rights form a neatly arranged unit, this will not give any problems. Such a change might, however, be considered as a change of the goal specification, something that would not occur in regular maintenance. In a blackboard system such as Hearsay-II, the various knowledgebases are distinct from each other, not only procedurally but also conceptually. Therefore, no functional cohesion is involved. Neither is module hierarchy in 'normal' blackboard systems. Distributed blackboard systems possess a limited (two-layer) module hierarchy [Durfee88].

5 KNOWLEDGEBASE INTEGRITY CONTROL

The integrity control of knowledgebases is mainly concerned with detecting errors in 'executable specifications'. It is assumed here that the specification of a knowledge-domain has been established with the help of ENIAM. The resulting conceptual model can be transformed into a formal representation in Prolog that can be executed on the computer. The rules form an important part of the knowledge in a knowledgebase and may be compared to production rules. The diagnostic part of Shapiro's algorithm can check the integrity of this Prolog code [Shapiro84].

Shapiro's algorithm was primarily aimed at program debugging; the location and correction of errors in the program code [Hamdan89]. These errors can have different causes, such as inadequate functional design or a failing system development method. According to Shapiro, however, a more fundamental cause can be pointed out. He sees a program as the epitomization of a complex set of assumptions. The behaviour of a program is a derivation of these assumptions, which makes it difficult to predit its behaviour completely and exhaustively. The considerations were an incentive for Shapiro to develop a theory that must produce an answer to two questions:

- How can an error be detected in a program that does not function properly?
- How can this error be corrected?

This theory has led to the construction of algorithms for each of these two problems. With the help of a diagnosis-algorithm, errors must be detected, after which improvements can be made using the error-correction algorithm. These tow algorithms are both included in a so-called 'Model Inference System' (MIS) that takes a program that must be debugged for its input and a list of input and output examples, that partly determine the behaviour of the program.

It is necessary that during the debugging process it is clear what the expected behaviour of the program will be. It is always assumed that there an 'instance' that can give an answer to the questions that are posed by the system about the 'Universe of Discourse'. The answer can be given by the developer of the program, or by another program. When, for instance, a properly functioning version of a program will be altered, questions that are generated during debugging can be 'answered' by the old version. It is even possible to build a (simple) program from scratch by starting with an 'empty' program. MIS as well as the programs to be debugged are written in Prolog.

By applying Shapiro's algorithm on 'pure' Prolog programs, three types of errors can be detected; termination of a program with incorrect results, termination with missing results and non-termination. It is designed to detect three possible errors in programs: non-termination, incorrect solutions and missing solutions. Considering the descriptions of consistency and completeness it can be said that Shapiro's interpretation of these notions covers various aspects. As far as consistency is concerned, the conflict and circularity aspects analysed with the help of algorithms for the detection of incorrect solutions and non-termination. The algorithm for detecting a missing solution indicates where there is a missing line in the program in order to generate this solution; this covers the completeness aspect. Non-termination is detected by Shapiro by a preset maximum depth of recursion is reached by the program, without having reached a solution. When compared to other algorithms, e.g. [Decker88; Ngyuen85], this approach is very pragmatic, but, nevertheless, very effective. In this case, conflict has a slightly different definition. Shapiro's use of the term correctness lends a more formal interpretation to this problem.

Shapiro's algorithm offers good possibilities for the integration of conceptual modelling (ENIAM) and integrity checking. In determining non-termination, it is felt that the algorithm lacks more support in further diagnosing the proof tree that is produced. Adding other techniques as is suggested by Nguyen is worth considering. From the analysis of correctness and completeness it has become evident that the user plays an important (interactive) role. By selecting variants with the most favourable query complexity and extending the algorithms to 'full-fledged' Prolog, an integrated AI development environment can be realised. The possibilities offered by the 'Model Inference System', including automatic 'repair' of incorrect procedures, can certainly play a role in this.

The aim of Shapiro's algorithm was to design a formal theory for the debugging process. The use of Prolog was instigated by the circumstance that there is a direct relationship between the syntax (structure) and the semantics (meaning) of Prolog. This makes it simples to diagnose and correct faulty programs. This characteristic is very welcome in the analysis of knowledgebase specifications, because a Prolog program may be seen as a formalisation of these specifications. So it is more than just a 'program'. A Prolog program embodies a formalisation of knowledgebase specifications. An extension in the direction of 'full-fledged' Prolog is very well feasible according to Shapiro [Shapiro84].

6 CONCLUSIONS

The quality framework presented at the beginning of this paper is a good guideline for tackling the quality problem of expert systems. The concept quality is viewed from three perspectives: the system specifications, the structuring of the development process and the expert system as a product. The knowledgebase is identified as the most important part of an expert system. Inference mechanism, man-machine interface and explanation facilities contain such an amount of 'conventional' components, that quality control can be achieved with well known methods and techniques. Reliability and maintainability of the knowledgebase are recognized as the principal quality criteria.

The development process of expert systems cannot easily be controlled. As opposed to 'conventional' life-cycle methodologies, there is no accepted version for expert systems. Experience accumulated in the application of SKE and the Kerschberg-Weitzel model could eventually lead to an easing of the problem.

For the moment there is no suitable (and tested) method that can be used in making clear specifications of expert systems. The development of KADS qualifies as an incentive. ENIAM provides good opportunities for specifying knowledgebases. Especially the way in which completeness and consistency of the knowledge model is guaranteed, makes ENIAM a far better choice than many other AI-specification methods, like KADS.

Testing of expert systems is an underdeveloped field of study. Often it is not possible to test the advice given by the system against an objective standard. Methods focussed on a structured generation of expert system test cases are not yet available. This is why the use of 'conventional' test methodologies is advocated.

7 REFERENCES

[Breuker87] Breuker, J. (ed.), "Model-Driven Knowledge Acquisition: Interpretation Models", University of Amsterdam, 1987.

[Breuker88] Breuker, J., Wielinga, B., "Models of expertise in Knowledge Acquisition", In: Guida, G., Tass, C, (eds.), "Topics in Expert System Design", North Holland, 1988.

[Clancey85] Clancey, W.J., "Heuristic Classification", In: "Artificial Intelligence", Vol. 27, 1985, pp. 289-350.

[Creasy88] Creasy, P.N., "Extending Graphical Conceptual Schema Languages", University of Queensland, 1988.

[Creasy89] Creasy, P.N., "ENIAM: A More Complete Conceptual Schema Language", In: "Proceedings of the Fifteenth International Conference on Very Large Databases", 1989, pp. 107-114.

[Date83] Date, C.J., "An introduction to database systems: Volume II", Addison-Wesley, 1983.

[Decker88] Decker, H., "Integrity Enforcement on Deductive Databases", In: Kerschberg, L (ed.), "Expert Database Systems, Proceedings of the First International Conference on Expert Database Systems", 1988, pp. 381-395.

[Deutsch82] Deutsch, M.S., "Software verification and validation: Realistic project approaches", 1982.

[Doyle88] Doyle, J., "Expert Systems and the 'Myth' of Symbolic Reasoning", In: "IEEE Transactions on Software Engineering", Vol. SE-11, Nr. 11, November 1988.

[Durfee88] Durfee, E.H., "Coordination of Distributed Problem Solvers", Kluwer Academic Publishers, 1988.

[FEL90] Lenting, J.H.J., M. Perre, "QUEST: Kwaliteit van Expertsystemen", FEL-90-A012, 1990.

[Gane79] Gane, C., T. Sarson, "Structured Systems Analysis", Prentice-Hall, 1979.

[Hamdan89] Hamdan, A.R., C.J. Hinde, "Fault Detection Algorithm for Logic Programs", In: "Knowledge-Based Systems", Vol. 2, Nr. 3, 1989.

[Llinas87] Llinas, J., Rizzi, S., "The Test and Evaluation Process for Knowledge Based Systems", Technical Report F30602-85-C-0313, Calspan Corporation, June, 1987.

[Myers79] Myers, G.J., "The Art of Software Testing", Wiley, 1979.

[Nguyen87] Nguyen, T.A., "Verifying Consistency of Production Systems", In: "Proceedings of the IEEE 3rd Conference on AI", 1987, pp. 4-8.

[Nijssen86] Nijssen, G.M., "Knowledge Engineering, Conceptual Schemas, SQL and Expert Systems: A Unifying Point of View", In: "Relationele Database Software, 5e Generatie Expertsystemen en Informatie-analyse", Congres-syllabus NOVI, 1986, pp. 1-38.

[Nijssen89] Nijssen, G.M., Halpin, T.A., "Conceptual Schema and Relational Database Design: A Fact Oriented Approach", Prentice-Hall, 1989.

[Shapiro84] Shapiro, E.Y., "Algorithmic Program Debugging", The MIT Press, 1984.

[Sowa84] Sowa, J.F., "Conceptual Structures: Information Processing in Mind and Machine", Addison Wesley, 1984.

[Turner87] Turner, W.S. (et al.), "System development methodology", 1987.

[Weitzel89] Weitzel, J.R., L. Kerschberg, "Developing Knowledge-Based Systems: Reorganizing the System Development Life Cycle", In: "Communications of the ACM", Vol. 32, Nr. 4, 1989, pp. 482-488.

[Wintraecken85] Wintraecken, J.J.V.R., "Informatie-Analyse volgens NIAM", Academic Service, 1985.

[Yourdon79] Yourdon, A., Constantine L.L., "Structured Design", Yourdon Press, 1979.

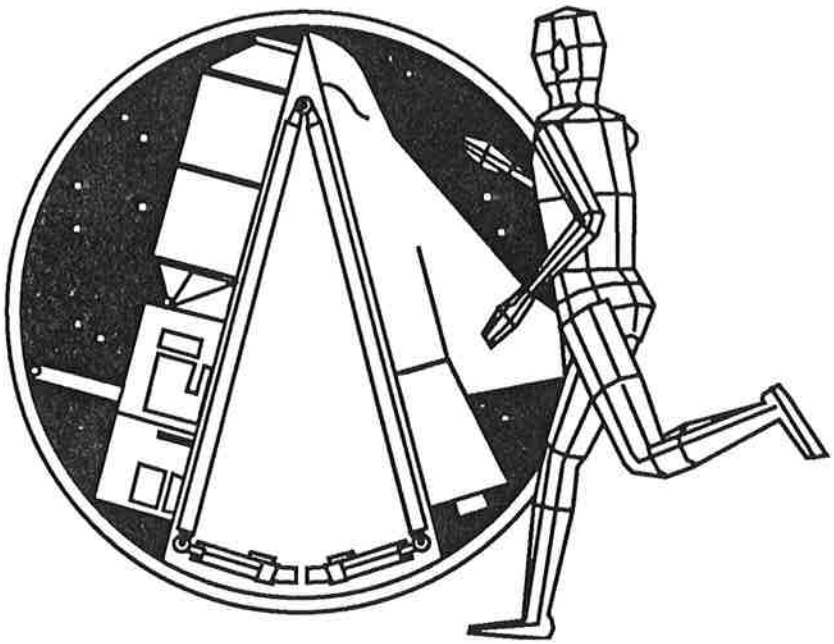
PROCEEDINGS

ARTIFICIAL INTELLIGENCE AND KNOWLEDGE-BASED SYSTEMS FOR SPACE

- Table of Contents -

22-23-24 May 1991

ESTEC, Noordwijk, The Netherlands



Organised by:

THE EUROPEAN SPACE AGENCY Simulation & Electrical Facilities Division - Technical Directorate