WHU-Forschungspapier Nr. 18 / Mai 1993

Maintenance of Knowledge Based Systems

Prof. Dr. Franz Lehner, Mag. Hubert F. Hofmann, Dr. Ralf Setzer,
WHU Koblenz, Lehrstuhl für Wirtschaftsinformatik und Universität Zürich, Institut für Informatik

Wissenschaftliche Hochschule für Unternehmensführung Koblenz
Haus d'Ester - Heerstraße 52
5414 Vallendar
Telefon 02 61 / 65 09 - 0
Telefax 02 61 / 65 09 - 1 11

Maintenance of Knowledge Based Systems

Franz Lehner, Hubert Hofmann, Ralf Setzer

Lehrstuhl für Wirtschaftsinformatik und Informationsmanagement
WHU Koblenz
Burgplatz 2
D-5414 Vallendar

Abstract

To date, more and more research institutes are involved in the design, development and utilization of knowledge-based systems. Little research has, however, been published on the maintenance situation of these systems. This paper deals with the specialities of the maintenance of knowledge bases. Exemplary error sources, requests for alteration and problems are described to highlight the complex nature of this field. The activities initiated and controlled by the knowledge base maintenance management are structured. Four base and two cross-sectional activities are identified and explained in detail.

Keywords

documentation, knowledge base, knowledge-based system, maintenance, maintenance activities, quality assurance, structuring of maintenance activities

Table of Contents

1 Introduction

1.1 Problem Definition and Goals

The necessity for maintenance originates from a continuously changing, largely unpredictable, and only partially knowable working environment (Winograd & Flores 1986). The knowledge based system's fit with the environment must be preserved and regularly validated. Therefore, the system requires permanent maintenance to ensure its usefulness and productivity. Thus, maintenance cannot be avoided and generally it does not point to any defect in the knowledge based system itself or in its development.

The term "software maintenance" denotes corrective, adaptive, and perfective maintenance (cf. Lehner 1991). Corrective maintenance is concerned with modifications to correct previously undiscovered errors in the software system. Adaptive maintenance comprises modifications required by changes in the working environment, and perfective maintenance is the improvement of system performance, e.g., through restructuring the system to reduce future maintenance.

Maintenance should not be regarded as being a necessary evil, and in general it is also not a sign of any defect in the product itself or in its development. Often it is the result of a changing environment (regarding tasks, technology etc.). Maintenance is an expression of the flexibility and capacity for innovation of an enterprise. The conformity of the application systems to the environment must be maintained and checked regularly if effectivity is to be guaranteed. When the environment changes, then usually the components of the application system which are affected by that change have also to be adapted to fit the new situation. Changes can also become necessary due to new or changed user requirements, errors in the software, performance problems (e.g. response-time) and for many other reasons. Errors and bad performance in particular can seriously hinder or make the use of the system impossible and so demotivate the user. Systems therefore require constant attention and further development to ensure that they do not lose their effectivity.

Conceptual modelling is to be particularly stressed as a special feature of the maintenance of knowledge based systems. When this aspect is taken into

consideration, the development of a knowledge based system can never really be regarded as ever being completed. Ultimately, this means that the (artificial) division into phases (e.g. design phase, development-phase and operational-phase) is of little use for maintenance purposes. At present, maintenance is still generally regarded as being a part of the operational phase in the life-cycle of application systems (for both conventional application systems and knowledge based systems). This is a simplification which does not provide a sufficient basis for a more thorough investigation of maintenance problematics. Implicit in this simplification is, for example, that the development of knowledge based systems or components can be carried out independently from later operations and the modifications which can be expected. Recognised procedures on the designing of knowledge based systems, e.g. KADS II (ESPRIT II Project 5248), have to some extent departed from strict phase-orientation by means of iterative development methods, prototyping-oriented procedure and the like. They do take maintenance into consideration (e.g. ensuring the possibility for expansion within the system design) but again only summarily.

The integration of maintenance into the process of the development of knowledge based systems ("design for maintenance") is, among other things, so difficult because very often the adaptability of human specialists is not taken into account and because the effect of knowledge based components on the work-situation is very difficult to predict (cf. e.g. Lamberts & Pfeifer 1992, Becker et al. 1992). The users are, however, "learning systems " and subject the contents of the knowledge base to a continuously new interpretation.

Only few and very specific investigations concerning the maintenance of knowledge based systems are available. The analysis described in this paper is a step toward a better understanding of the maintenance process of knowledge based systems and develops a technical and managerial framework of essential maintenance activities.

1.2 Knowledge Based Systems and Maintenance

The field of artificial intelligence constantly develops and many terms do not have accepted definitions. There are many definitions of knowledge based systems differing in their emphasis.

By knowledge based systems, we mean systems in the form of computer programs that support, or perform automatically, tasks that are usually carried out by domain specialists. The domain specialist performs these tasks by employing personal skills, expertise and judgement acquired and learnt over a period of time. A knowledge based system will generally consist of a knowledge base, a reasoning component (inference mechanisms), explanation and acquisition component, and a communication component.

Traditional artificial intelligence, especially research in the area of knowledge based systems, focuses on automated reasoning, i.e., to achieve the explicit goal of automated problem solvers. However, this view of knowledge based systems runs into severe problems (Clancey 1991; Hofmann, Pfeifer, Vinkhuyzen 1993). Since every representation will always be incomplete, given the properties of the real world, the system is bound to fail at some point.

Therefore, the (conceptual) models underlying the knowledge based system, the system's rationale, and the rationale of its development process has to be documented. This documentation describes the features of the application domain that are of interest to the client, but ignores details and relations that can be important for another purpose. This is an important point to realise in designing and maintaining knowledge based systems.

As one wants to reuse (components of) the knowledge based system, and as perception of problems will change over time, conceptual models and their implementation must change, too. A problem with many knowledge based systems is that they have a representation of the application domain that does not change over time (Hayes-Roth et al. 1983; Bobrow et al. 1986). Moreover, the reasoning (or inference) mechanisms of a knowledge based system freezes the interpretation of these representations. Therefore, knowledge based systems often do not allow the reinterpretation of data, which is necessary to cope with the changing environment.

The representation and storing of knowledge and knowledge-structures (Meta-Knowledge) cannot be separated from the representation formalism employed, from the programming language or from the development environment etc. Among other things, it is the fact that these links have to be taken into consideration - that they have a direct influence on maintenance activities - that

makes maintenance such a difficult task. The maintenance of knowledge based systems is, therefore, to be understood in a comprehensive sense. This will also take into account that there is a visible trend towards hybrid systems, i.e. application systems are composed more and more from components which have not only been developed with conventional technology, but also with the help of expert system technology. A further argument for the chosen perspective is the fact that in the expert system technology many elements of traditional programming are to be found and vice versa. Experience gained from the maintenance of conventional application systems can, therefore, probably in part be taken over.

There are various possibilities available for the representation of knowledge in a knowledge base. Most common is the use of rules, constraints, objects, or frames. The knowledge itself can be retained in very different forms (cf. VDI-Guideline 5006 1992): factual knowledge, causal knowledge, conceptual structures of the object area, knowledge in the form of rules, context and explanation of rules, procedural knowledge, reasons for the applications of the rules, strategies, weighting, proofs, probabilities among others. The inference component serves the linking of knowledge elements which are managed in the knowledge base. It must, therefore, be co-ordinated with the type and special characteristics of the knowledge representation. This fact provides a further argument as to why maintenance cannot be limited to an adapting of the knowledge representation.

2 Errors, Alteration Requests and Reported Problems

To date, a systematic structuring of important or typical error sources, requests for alteration and problems concerning knowledge based systems is not yet possible. The following description is, therefore, by way of being exemplary and should give an impression of the complexity of the maintenance of a knowledge based system. As a rule, it is hardly possible to obtain a general overview of potential error sources in knowledge based systems; neither can these sources be precisely or completely described. This is demonstrated by the fact that testing, the most usual form for discovering errors, usually uncovers no more than 50 per cent of the errors present in a system (cf. Jones 1983). The following descriptions, which are derived in part

from experience gained from personal development projects and in part from relevant literature, can therefore only give a general overview in the form of exemplary error sources, requests for alteration and problems. Some of the points mentioned can be observed generally during the maintenance of software systems, some apply specifically to knowledge based systems. In the following they are described together, because a clear division between them is for the most part neither possible nor useful.

2.1 Exemplary Error Sources

- One typical error source is the insufficient dimensioning of variables, the origins of which are to be found in the lack of specification. For example, parts of data records which are to be worked on are loaded into matrices which were originally designed for a smaller number of data records.

- The input checking routines for user inputs are another source for error. They are essential for assuring the integrity of the knowledge base. The conducting of wrong knowledge elements can lead to wrong results in the later knowledge processing.

- Particularly during the development of expert systems, a syntax error during the input of knowledge elements (e.g. when copying facts or rules with the object of keeping the input effort low) can result in the impossibility of a later "Pattern Matching", i.e. the effective checking of syntactical conformity with the aim of constructively linking the corresponding knowledge elements. This is particularly serious when the user has no detailed knowledge concerning the contents or structure of the knowledge base and so cannot check the result or can only partially check it.

- In the development systems (e.g. Shells, Programming Environments) of expert systems there may be programming errors which can be traced back to insufficient quality standards during their development. Furthermore, errors can arise caused by incompatibilities between the development systems and the respective operational system. Unless newer, co-ordinated versions of the development environment and the operational system are available, these errors can only be avoided by changing the implementation of the software system.

2.2 Exemplary Requests for Alteration

- A frequent request for alteration with regard to data storage is the alteration of the format in which the data is stored. An example for this is the introduction of the fifth digit in the postal codes of the Federal Republic of Germany, or the coming turn of the century which will lead to alterations in existing software systems.

- A similarly frequent request for alteration is the introduction of new objects, which are to be incorporated into an existing object category. If, for example, an object class CUSTOMER has hitherto been managed, it could become necessary due to altered requirements that this object class be split into two classes each with a different definition of a customer.

- Changes in the structure of knowledge bases which are due to changes in the environment of the software system are also a part of the maintenance process. An example for this is the combining of the contents of different data models in a common data model which is finally represented in a data base (e.g. relational, deductive, object-oriented).

- Frequent requests for alteration can be expected especially in the case of knowledge bases which are produced during the development of expert systems because of the use of an explorative construction paradigm. Maintenance also becomes more complex when in such knowledge bases not only knowledge is inserted into existing structures, but also new structures, thereby often making a revision of the existing structure for the recording of knowledge necessary.

2.3 Exemplary Problems

Some examples for the individual problem areas follow (cf. Becker et al. 1992).

- **Utilization Problems**: inputs or alterations going back a long way in the dialogue, linking of consultant and handbook components, checking efficiency of the knowledge based systems whether it is up-to-date.

- **Functional Problems**: indication of threshold values, indication of provisional values, filing memoranda, modification exclusively by developer, answering of why-not questions.

- **Integration Problems**: linking of consultant and handbook components; indication of provisional values;

- **Lack of Transparency**: transparency of contents (indication of contents, modification and completeness of the contents), functional transparency (operation of system functions, applicability of system functions).

Figure 1 shows the classifying and systematizing of problems which can occur when knowledge based systems are in use.

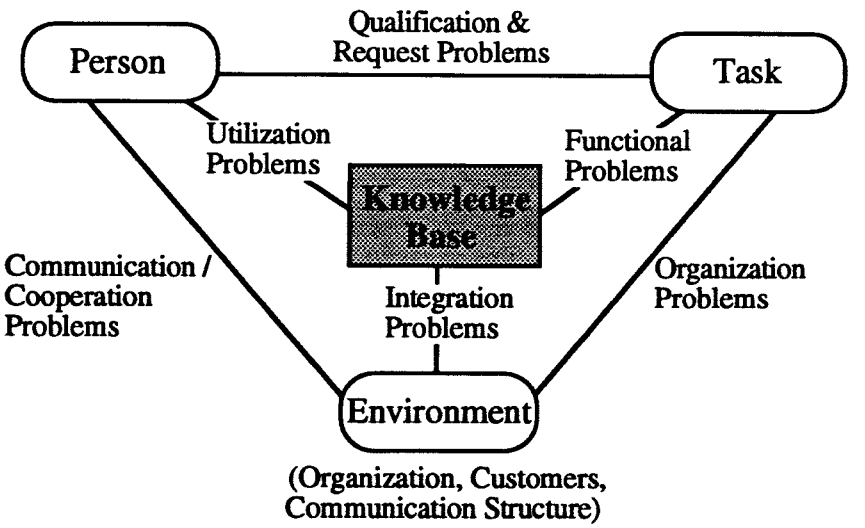

Fig. 1: Problem areas on the use of knowledge based systems (cf. Becker et al. 1992)

The fact that the **maintenance situation** concerning knowledge based systems which are in practical operation is a problem which must be taken seriously is borne out by the few reports in existence on this theme. For example, four years after the XCON/R1 system which supports the configuration of DEC-computers was put into operation, four employees were still working full-time on the extension and alteration of the knowledge. The maintenance effort involved had not decreased during this period of time, in fact it had increased. However, the benefits derived from the use of the system

were still valued as being higher than the costs of maintenance. SIRATAC, a support system for findings on the pest infestation of cotton plants, was replaced by a newly conceived system after several years in operation because the stored knowledge had increased to such an extent that maintenance difficulties arose (cf. Jansen & Compton 1989).

Another problem area which is frequently mentioned is the **documentation situation**. Very often no documentation at all exists for the knowledge based systems. In addition, it can occur that in certain programming languages, different types of knowledge can be mixed in one rule. The OPS5 language, for example, allows knowledge of the task area, inference knowledge, heuristics inter alia to be represented in one rule. Apart from the mixing of types of knowledge, there is also the problem that there is no explicit representation form for certain types of knowledge. Each specific solution depends on the language used. This could lead to, for example, the heuristic of problem solving finding expression in the application of rules sequence. The alteration of a single rule can result in unforeseeable consequences (cf. Angele et al. 1990). Together with insufficient documentation, or lack of the same, this is a particularly critical task for maintenance.

Development environments of knowledge based systems (e.g., shells) can also have programming errors, which can be traced back to insufficient quality standards during their development. Also errors can arise due to incompatibilities between the operational system supported by the development environment of the knowledge based system and the operational system in the working environment. As knowledge based systems have to be tested in the working environment, the described incompatibility leads to fault fixing in the implementation code. This causes the divergence of the knowledge based system's specification from its implementation, which is a major reason for maintenance difficulties (Hofmann 1993).

The **content of the knowledge base** is often neither transparent to the user nor to the maintenance personal, e.g., for rule-based systems the alteration of a single rule can often result in unforeseeable consequences. Thus, often only the developer (!) is allowed to change the knowledge base. This forces the user, who is the expert in the application domain, to work with a system that represents the application domain as perceived by the developer, who is a

novice concerning the application domain. With insufficient documentation, or lack of the same, this leads to enormous maintenance costs.

3 Process Model for Maintenance Activities

At present maintenance, both in "conventional" software systems and in knowledge based systems, begins when the system is accepted by the client, installed and put into practical use. This view does not provide an adequate basis for maintenance, because it implies that knowledge based systems can be designed independently from their practical use and expected modifications. We believe that maintenance is a process that accompanies the whole life cycle of a knowledge based system. The aim is to make maintenance and initial design as complementary as possible. If maintenance can be successfully carried out with the same methods, techniques and tools as the initial design of knowledge based systems, then organisations will benefit heavily from such an approach, e.g., uniformity and standardisation, facilitation of continued education and staff training, documentation support, introduction and back-up.

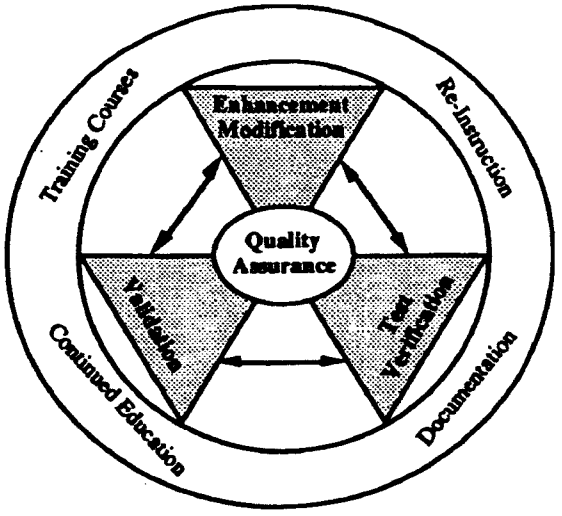

Fig. 2. Maintenance Process Model

We suggest the process-oriented division of work during maintenance, instead of component-oriented division of work. Component-oriented maintenance leads to division of work in the sense that different people maintain different

components (e.g., knowledge base and reasoning component). Process-oriented maintenance means the division of work based on different methods for which different people are responsible. This means that the people involved in maintaining the knowledge based system are assigned to working processes applying a particular method. For example, an individual or group performs enhancement or modification, verification and validation, whereas another individual or group performs quality assurance. Process-oriented maintenance helps all people involved in maintenance to understand thoroughly the knowledge based system. Moreover, the responsibility for a whole work process increases the motivation of maintenance personal. Figure 2 provides a simplified version of the maintenance process model, but shows the permanent possibility that activities recur to revise or elaborate insights in the perceived problem.

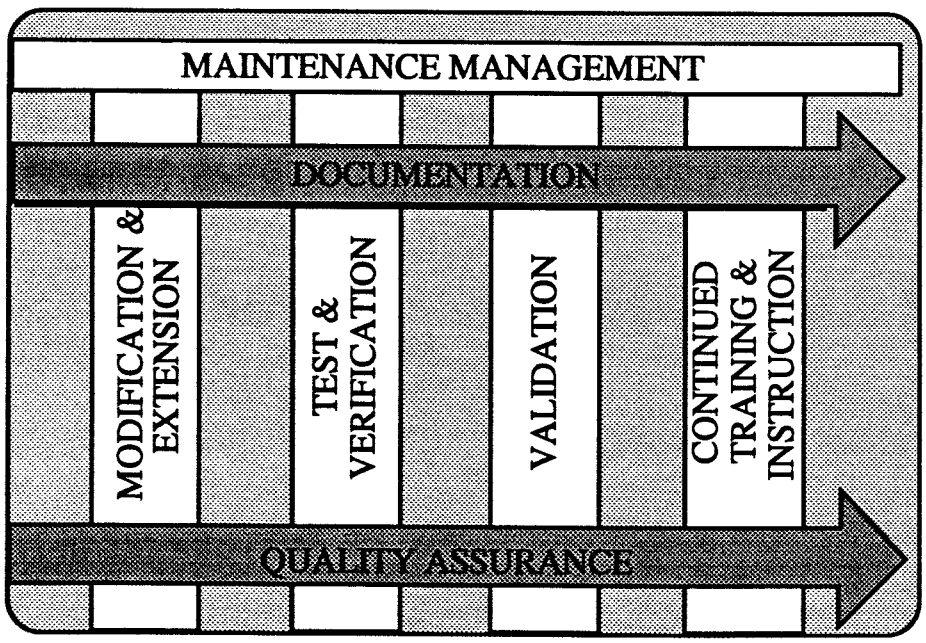

Fig. 3: Maintenance management

Thus, the clear, but artificial, separation of different phases in the software life cycle reveals its deficiency and is not conducive to maintenance. In the following, the maintenance which is to be initiated and controlled by the maintenance management is structured and presented in the form of activities. These activities are thought to be the most important maintenance domains,

although only few empirical findings have been presented in this field until now. The following activities are differentiated (cf. fig. 3):

- modification and extension
- test and verification
- validation
- re-introduction and training

These maintenance activities are supplemented by two cross-sectional functions, namely documentation and quality assurance. Both are to be taken into consideration during all maintenance activities. They are not classed with maintenance management, which spans all maintenance activities, because they involve the operational performance of the individual activities and not their consideration within the management processes.

In the following, documentation and quality assurance will be described first in order to emphasize their cross-sectional character. The individual maintenance activities will then be presented.

3.1 Documentation

The documentation of the existing knowledge based system is essential for effective and efficient maintenance. Ideally, this documentation will have been produced or continued parallel with the development of the knowledge base and possible previous maintenance processes. Within an actual maintenance process, the updating of documentation within the individual activity is also necessary. A sub-sequent documentation of alterations, i.e. a documentation after completion of the maintenance, results in an incomplete documentation or one that is not up-to-date. This can be disadvantageous to quality and efficiency. It can be expected, for example, that a test may be conducted far more accurately if, due to documentation, altered parts of the knowledge base are clearly identifiable. In addition, because of its different presentation form compared to the formal source code, the documentation accompanying the alteration can lead to an early questioning of the specification and its software implementation.

Ideally, the documentation is a reference containing everything necessary to develop and maintain a knowledge based system that is acceptable to the client. Only when the documentation is used as a reference throughout the knowledge based system's life cycle, it will be useful during maintenance. Otherwise one will run easily into a self-fulfilling prophecy (Parnas 1986). Documentation regarded as necessary evil written as an afterthought, because some bureaucrat requires it, will always be poor documentation.

Documentation within the maintenance process usually takes place by adapting or continuing the documents or data that have been provided by the original documentation. Aids are, e.g., word processing systems for flow text documentation, hypertext systems, data dictionaries with data description language for the formal description of data structure, and editors for the including of program commentaries. It can be expected, for example, that a test may be conducted far more accurately if, due to documentation, altered parts of the knowledge based system are clearly identifiable.

The documents used during maintenance are usually large. They may be several hundred pages long, each page containing many detailed items, each of which may have profound effect on the rest of the system. However there is a problem with such large documents. It is almost impossible to read a maintenance document of several hundred pages and be able to conceptualise the system. Once a document gets to a certain size and complexity, it is impossible for one person to hold the complete picture in his or her head. This inevitably leads to problems and errors. Consequently, conceptual models of the application domain are absolutely necessary to avoid the laborious transformation from maintenance requirements to code changes in the knowledge based system. This will cause the knowledge based system to diverge from its documentation.

In addition to the documentation of e.g. data structures, decision logic, interfaces and external references, a conceptional model of the application area is of crucial importance for the understanding and maintaining of knowledge based systems. A conceptual model is developed during the analysis of the application area, i.e. the inter-action of all concerned, e.g. software engineers and user, is essential. The user contributes above all his/her specialized knowledge of the application area and represents a partial, usually incomplete

model of the application area (cf. Newell 1981; Norman 1983). Conceptual models are, therefore, a reflection of expert knowledge on the task area, they do not, however, necessarily represent the cognitive processes of a task specialist (Winograd & Flores 1986). Customers, users, maintainers and developers investigate the knowledge based system and its connections to potential environments. They attempt to impose a global structure on acquired data to create a model of the application domain in terms of taxonomic hierarchies, causal networks, tables, flow diagrams, or whatever organisation is convenient for modeling the domain.

The results are documented as conceptual models. Conceptual models describe static and dynamic aspects of the problem resp. the application domain. Static structures of concepts are represented by networks of objects, which encapsulate attributes and exclusive services cf. (Rumbaugh et al. 1991; Dodd 1990). Dynamic structures describe the interface of the system and how the system interacts with its environment.

3.2 Quality Assurance

The aim of introducing principles, methods and procedure models of software engineering is to assure a certain standard of quality during the development of software. Software quality assurance is, consequently, part of the branch of software engineering and should therefore be a planned and systematically considered part of the maintenance of software products (cf. Wallmüller 1990, 6).

By (software) quality, those features of a (software) product are meant which ensure that the demands made of the product are met (cf. Heinrich & Roithmayr 1992; DIN 55350, Part 11; ANSI/ASQC A3 - 1978). In addition, in software production not only the features of the product itself, but also keeping to cost estimates and meeting deadlines are often seen as being part of quality. This requires a broader view of the concept of quality to include the process of software development. Quality goals are defined which are peculiar to the development process and which derive from the quality aims of the product (cf. Wallmüller 1990, 6).

The demand for inter-subjective testability and mechanical re-checking of software, or rather of the set quality features, led to the development of a field known as Software-Metrics. The aim of this field is to define and apply quality standards which allow quality features to be measured objectively and, if possible, mechanically. From the vast number of quality standards for software, the following are of particular interest from the point of view of maintenance (cf. Lehner 1991, 154 ff.).

- Design-Stability Standards (e.g. PDS Yau/Collofello),
- Complexity Standards (e.g. MacCabe, Halstead, Rechenberg),
- Alteration and Extension Standards (e.g. LSP Yau/Collofello, Modifiability Metric Yau/Chang),
- Reliability Standards (Error-Prediction Standard Ottenstein, Reliability Standard, Nelson),
- Availability Standards (Error-Day, MTBF, Standard Littlewood),
- Further Standards (e.g. Portability, Modularity metrics, Standards for measuring ease-of-use, clarity of code, testability and quality of documentation).

A specialized field of research is still involved in the development of metrics for the evaluation of knowledge based components (cf. O´Keefe et al. 1987; Ayel & Laurent 1991). Two kinds of metrics are differentiated here: Task-area-oriented metrics and metrics for knowledge based components (cf. Preece 1990). Task-area-oriented metrics measure the behaviour of knowledge based components and are analogous to the criteria applied by specialists to assess other specialists. Metrics of knowledge based components are independent of a specific software system and are concerned with the content and structure of the knowledge based components (cf. e.g. GATEWAY Project, IED Project 1751).

Experience in the field of software-metrics shows, however, that at present it is not possible to make up a complete list of quality characteristics (and thereby also quality standards), and that each organization, or rather each development environment has its own specific characteristics. This results in the fact that often only guidelines can be given for a metric. Also, it is often overlooked that the actual difficulty lies in the interpretation of the results of measuring; at present, no very well substantiated experience and results exist

in this area. Nevertheless, we may now proceed on the assumption that metrics will be increasingly employed to this end. They are at present already a permanent part of object-oriented design environments, which occupy a core position in the field of knowledge based systems.

Also we may now proceed with the assumption that metrics will be increasingly employed, a remark is in place. It is often overlooked that the actual difficulty lies in the interpretation of the results of measuring; at present, no substantiated experience and findings exist in this area. A related problem is that most of the measures concentrate on purely rule-based systems. However, many commercially available tools are hybrid, e.g., rules combined with object-oriented techniques and frame-mechanisms. So many measures are restricted in use. However, metrics are already part of object-oriented and logic-based development environments, which occupy a core position in the field of knowledge based systems.

3.3 Modification and Enhancement

New insights in the application domain, the modification of existing functions and data, and the correction of perceived errors, lead to continuous adaptation of the knowledge based system. Modification and enhancement is a permanent process of adjustment in which specialists, users, system designers, and maintainers as a group service, update and improve the knowledge based system.

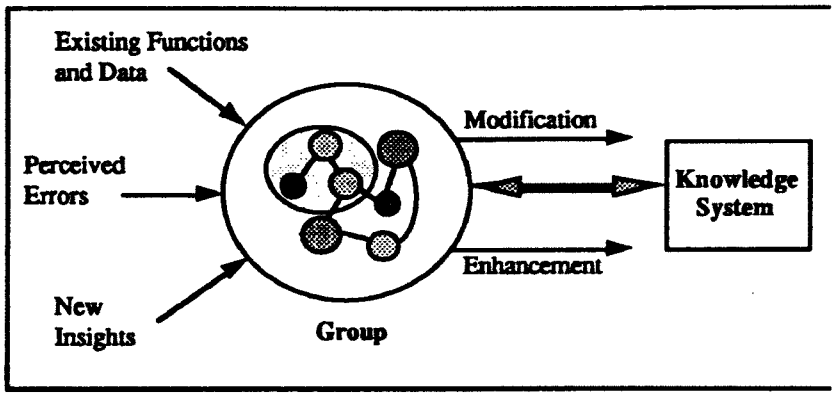

Fig. 4. Co-operative Modification and Enhancement

The modifications and enhancements of the knowledge based system must be documented in the knowledge based system itself (e.g., annotations) and they must be always traceable. We document the maintenance rationale, which outlines the reasoning and argumentation that make sense of a specific modification or enhancement. The rationale is important because a maintainer wants to change the knowledge based system to respond to new requirements and not to disturb its correctness.

Alterations in the user or technical documentation and changes in informational material (e.g., marketing documents, training documents, etc.) are also part of the modification and enhancement of the knowledge based system.

According to traditional concepts of maintenance, these activities should be carried out so that the design and implementation of the knowledge based system is preserved as far as possible. Maintenance activities are governed usually by the level of knowledge that prevailed at the time of initial design, i.e., out-dated methods are used to work on outdated knowledge based systems.

The envisaged maintenance concept has, however, a different starting point. Before modification or enhancement, the knowledge based system is brought up to the most recent level of technology. The maintenance personnel can then proceed with the modernised knowledge based system in the same way as with a newly developed knowledge based system, using the most up-to-date methods and aids. The modifications are specified, designed and implemented the same way as for newly-developed knowledge based systems.

Ideally, not the maintenance personnel, but the user would carry out the modification with the appropriate tools after having received the system. This kind of maintenance (tuning) must be considered during development (Greenbaum & Kyng 1991).

3.4 Test and Verification

When modification and expansion have been carried out, then the altered knowledge bases are to be tested and verified (cf. Pfeifer et al. 1992; Ayel & Laurent 1991).

Testing is an activity for verifying the correctness of results in the software design, e.g. experimental execution for determining pre-defined characteristics. In this context it is interesting to discover that testing is a far less effective means for finding errors than is generally supposed (cf. Lehner 1989). In comparison, formal design and code inspections carried out by small groups of experienced specialists are more effective at locating errors.

Maintenance test activities differ from testing during new development primarily in that a test environment is already in existence and representative test data is available. During maintenance, however, test environment and test data have to be adapted or extended.

Verification refers to activities which can be carried out purely on the "internal" representation of a knowledge base, that is, determining the consistency, freedom from redundancy and "correctness" within the framework of the chosen representation formalism. Verification treats the knowledge base as a "white box", i.e. what matters most is to show that the internal structure of the knowledge base corresponds to its formal specification.

Usually knowledge based systems are verified through testing, which is a confirmation technique ensuring that the specification of the knowledge based system follows pre-established criteria. Testing often means inspections, reviews, walkthroughs, and static techniques, cf. (Freedman & Weinberg 1990). For example, if a requirements specification is available for the knowledge based system, requirements tracing is just as practical as it is for conventional software. If there is no design document, then the requirements specification may be traced directly to the source code.

Other verification techniques especially important for knowledge based systems are: interaction analysis, truth analysis, and uncertainty analysis. Interaction analysis deals with the ability of the interacting components of the knowledge based system to produce the desired results. Truth analysis evaluates the truth of the knowledge base contents. Uncertainty analysis is applicable when the knowledge based system makes use of measures of uncertainty. It determines whether the uncertainties assigned and the method of combining uncertainties yield correct results.

The structure commonly employed for knowledge based systems makes interaction analysis important (Green & Eckert 1987). Knowledge based systems often comprise many small and quasi-independent components (whether rules or objects), and the intended behaviour of these systems results from the interaction of these components under the control of the inference mechanisms. Unlike conventional, procedural computer programs, the sequence and results of processing are not readily apparent. Interaction analysis strives to predict the results of component interaction, which particular attention paid to the possibility of unintended results arising.

When new facts or rules become known, then it should be possible to correct the knowledge base with only an acceptable amount of effort involved. Truth analysis is employed to evaluate the truth of elements (e.g., rules, objects) stored in the knowledge base. In the field of logical representation, truth analysis is supported by so-called truth-maintenance systems, (cf. Doyle 1979; Thuy & Schnupp 1989). At each extension of the knowledge base, these systems check all the contents for consistency and, if necessary, carry out corrections. The necessity for such systems arises when conclusions which have been deduced to be true and have been stored in the knowledge base later, through the addition of new knowledge, prove to be false. One speaks of monotonous logic when conclusions which have been deduced to be true and have been stored in the knowledge base can no longer prove to be false through the addition of new knowledge; at most, new, true facts can emerge. When this assumption (assumption of monotony) is not valid, then at each extension of the knowledge base all its contents must be tested for consistency and, if necessary, corrected. This procedure is called "truth maintenance" and the systems supporting it, "Truth Maintenance Systems". Without truth maintenance, non-monotonous logic leads to the intolerable situation that deduced facts and rules filed in the knowledge base can become false (Thuy & Schnupp 1989, 77).

3.5 Validation

Validation means the checks that (formal) specifications describe the right system, i.e., specifications are correct concerning the requirements. If the validation process is inadequate, misunderstandings and errors are propagated

to the implementation, and expensive modifications of the knowledge based system are required to correct them.

But validation is very difficult. The maintained knowledge based system not only changes the environment of the users, but also their perceptions of it. Therefore, it is important that knowledge based systems are always validated against the interpretations of the users. It follows, that the best place to validate software is the working environment, where the user depends on the software to do his or her work and is under time-pressure to perform tasks (Pfeifer et al. 1992).

Validating the knowledge based system is a recurrent and arduous activity, but it is made all the more difficult when the knowledge based system must be designed to be extremely flexible, because it is expected to be used under varying conditions. Consequently, if a user makes a recommendation requiring a change in the knowledge base, several outcomes are possible. Cochran and Hutchins (Cochran & Hutchins 1987) outlined the following example, when validating Mentor. Mentor is a knowledge based system that helps service representatives to maintain large refrigeration systems. Suppose a service representative makes the statement "never change the oil in the middle of winter", in response to the system recommendation to change oil in January. This could reflect either a difference of opinion (some experts might argue that the oil will collect contaminants all winter, while others might argue that it is better to have new oil in the machine all winter); or a correct statement for the region in which the user works, but incorrect for different regions in which the winter is shorter, contaminants less of a problem, etc.; or the service representative's misperception that it is best to change the oil in the winter; or a correct statement of the company's policy; or an altogether irrelevant criticism (as for Honolulu); or—last, but not least—a correct criticism that needs to be addressed.

Validation can be time consuming and expensive. Assessing the effort and amount of money that should be applied to validation is difficult. Curtailing the effort for validation can mean that clients run the risk of a poorly validated system. We believe that validation costs can be controlled by designing formal validation methods that are integrated within the development process.

It is important to have all means available for validation. Showing correctness of specified modifications and enhancements is extremely difficult if an abstract approach is adopted. Reading definitions and specifications, clients must picture the system in operation and imagine how that system fits into the working environment. This type of analysis is difficult to perform, if possible, for clients. Therefore, we suggest to perform validation by using the specification as an executable prototype, rather than validating already implemented systems that are difficult and expensive to change.

Executable specifications can contribute enormously to validation, because they allow users to experiment with stable portions of the knowledge based system and thus give them the necessary touch-and-feel experience (Fuchs 1993). So users can participate in the specification of modifications and enhancements and in their immediate validation. The involvement of users and the immediate reflection on consequences of maintained components of the knowledge based system are of utmost importance, because they contribute to the correctness of software, to the reduction of costs, and to the adherence to the schedules.

3.6 Re-Introduction and Training

The integration of the maintained knowledge based system into the working environment concentrates on the activities: re-introduction and training.

During integration, the conformity of system behaviour with required efficiency in the work environment is investigated by applying qualitative and quantitative criteria. An important assessment criterion is whether the re-introduction of the knowledge based system improves the user's work-situation or the supported business process. This investigation focuses on the quality, usability, and benefit of the re-introduced knowledge based system.

The quality assessment should be carried out by a group of experts who were not involved in other maintenance activities. Also (long-term) empirical tests of the system's usage are necessary. The benefit of the knowledge based system is determined by the users and the management of the supported business area. Usability is determined, for example, by criteria such as user acceptance, the ease with which the system's answers are understood, and the adaptivity of the knowledge based system.

The re-introduction of maintained knowledge based systems is backed-up by training activities. Due to the (technical) documentation resulting from the suggested maintenance process, the training is based on up-to-date material, which in turn reduces the necessity for user support.

4 Conclusions

Systematic surveys and data about knowledge based systems in productive use are not available. This is especially true for maintenance methods and tools. Although successful maintenance is a major factor for the efficient and effective use of knowledge based systems, only few publications are available. From the few reports available, it becomes clear that the maintenance of knowledge bases represents a decisive factor for success with regard to the applicability and efficiency of the system.

In so far as insufficient documentation, or the lack of documentation, is the cause of maintenance problems concerning knowledge bases, then the situation is comparable to conventional software maintenance and requires to some extent that similar measures be taken. The remaining problems discussed in this paper represent a combination of lack of experience and technical restrictions. An improvement of the situation is expected from new developments in the area of executable conceptional models as well as from Knowledge Dictionaries (cf. eg. Jansen & Compton 1989). Conceptional models, which are as similar to the functioning system as possible have, among others, the following advantages with regard to maintainability: explicability, self-documentation, and comprehensibility. Not only will they lead to fewer errors, but also to fewer important errors during maintenance. The contribution of Knowledge Dictionaries can be compared to the maintenance back-up of Data Dictionaries in conventional development environments. Particular effort is still required in the areas of quality assurance (e.g. programming conventions, program structures) and of quality assessment (e.g. Software-Metrics for object-oriented languages).

We believe that as long as maintenance is viewed strictly as a back-end activity, maintenance costs will continue to exceed development costs. We need to do more work on stopping misperceptions and errors getting into the knowledge

based system than detecting them afterward. During each activity of software development one must think about, e.g., its (long-term) implications, identify and prioritise the changeable aspects. Executable specifications and a formalisation of the validation process will help to improve the maintenance process of knowledge based systems.

Bibliography

Angele, J., Fensel, D., Studer, R.: What could the knowledge engineer learn from the software engineer? Ehrenberg, D. et al. (ed.): Wissensbasierte Systeme in der Betriebswirtschaft. Berlin 1990, 285-303

Ayel, M., Laurent, J.-P. (ed.): Validation, Verification and Test of Knowledge-based Systems. John Wiley & Sons, Chichester/New York 1991

Barker, V.E.; O'Connor, D.E.: Expert Systems for Configuration at Digital: XCON and Beyond. Communications of the ACM, vol. 34, no. 3, 1989, 298-310

Becker, B., Herrmann, Th., Steven, E. (ed.): Zur Diskrepanz zwischen Expertensystementwicklung und -einsatz. GMD Arbeitspapier Nr. 641, GMD Bonn/St. Augustin, April 1992

Bobrow, D.G.; Mittal, S.; Steffik, M.J.: Expert Systems: Perils and Promise. Communications of the ACM, vol. 29, no. 3, 1986, 880-894

Brown, B.J.: Assurance of Software Quality. Technical Report SEI-CM-1-1.1 (Preliminary), Software Engineering Institute, Carnegie Mellon University, 1987

Clancey, W.: The Frame of Reference Problem in the Design of Intelligent Machines. In: VanLehn, K. (ed.): Architectures for Intelligence, Erlbaum, Hillsdale, 1991

Cochran, E.L.; Hutchins, B.L.: Testing, Verifying, and Releasing an Expert System: The Case History of Mentor. In: Garcia, O.N.; Chien, Y.-T. (eds.): Knowledge-Based Systems: Fundamentals and Tools, EEE Computer Society Press, Los Alamitos, 1991

Dodd, T.: Prolog - A Logical Approach. Oxford Science Publications, Oxford, 1990

Doyle, J.: A Truth Maintenance System. Artificial Intelligence, vol. 12, no. 3, 1979, 231-272.

Freedman, D.P.; Weinberg, G.M.: Handbook of Walkthroughs, Inspections, and Technical Reviews. Dorset House, New York, 1990

Fuchs, N.E.: Compositional Software Development. Technical Report, Institute for Informatics, University of Zurich, Zurich, 1993 (in preparation)

Green, C.J.; Eckert, M.M.: Verification and Validation of Expert Systems. In: Second Western Conference on Expert Systems, IEEE Computer Society Press, Anaheim, 1987

Greenbaum, J., Kyng, M.: Design at work. Hillsdale, N.J.: Erlbaum, 1991

Hayes-Roth, F.; Waterman, D.A.; Lenat, D.B.: Building Expert Systems. Addison-Wesley: Reading, 1983

Heinrich, L. J., Roithmayr, F.: Wirtschaftsinformatik-Lexikon. 4th Ed., München/Wien, 1992

Hofmann, H.F.: Requirements Engineering: A Survey of Methods and Tools. Technical Report, Institute for Informatics, University of Zurich, Zurich, 1993

Hofmann, H.F.; Pfeifer, R.; Vinkhuyzen, E.: Situated Software Design. In: Ramamoorthy, C.V. (ed.): Fifth Intemational Conference on Software Engineering and Knowledge Engineering, Knowledge Systems Institute: San Francisco, 1993 (to appear)

Jansen, B., Compton, P.: Data dictionary approach to the maintenance of expert systems: The Knowledge Dictionary. Knowledge-Based Systems, Vol. 2, 1/1989, 14-26

Jones, T. C.: Verhütung und Beseitigung von Programmierfehlern. Elektrisches Nachrichtenwesen, 4/1983, 295-300

Lamberts, K., Pfeifer, R.: Computational models of expertise. Accounting for routine and adaptivity in skilled performance. Gilhooly, K., Keane, M. (ed.): Advances in psychology of thinking, New York: Simon & Schuster, 1993, in preparation

Lehner, F.: Nutzung und Wartung von Software. München, 1989

Lehner, F.: Softwarewartung. München, 1991

Lehner, F.; Hofmann, H.F.; Setzer, R.: Wartung und Pflege von Wissensbanken (in German). WHU-Forschungspapier Nr. 15, The Koblenz School of Corporate Management, November 1992; also as: Contribution to VDI (Verein Deutscher Ingenieure), Richtlinie VDI 5007, Wissensbanken in der Anwendung, 1993 (in print)

Newell, A.: The Knowledge Level. AI-Magazine, Vol. 2, No. 2, 1981, 1-20

Norman, D.: "Some Observations on Mental Models". In: Gentner, D. (ed.): Mental Models, Hillsdale, New Yersey, 1983, 72-113

O'Keefe, R. M., Balci, O., Smith, E. P.: Validating Expert System Performance. IEEE Expert, Winter 1987, 81-89

Parnas, D.L.: A Rational Design Process: How and Why to Fake It. IEEE Transactions on Software Engineering, vol. SE-12, no. 2, 1986, 251-257.

Pfeifer, R.; Hofmann, H.F.; Vinkhuyzen, E.; Rademakers, P.: Problemlösende Systeme: Implikationen für die Validierung (in German). KI, vol. 6, no. 4, 1992, 71-73.

Preece, A.: Towards a Methodology for Evaluating Expert Sytem. Expert Systems, Vol. 7, No. 4, 215-223

Rumbaugh, J.; Blaha, M.; Premerlani, W;; Eddy, F.; Lorensen, W.: Object-Oriented Modeling and Design. Prentice Hall: Englewood Cliffs, 1991

Thuy, N. H. Ch., Schnupp, P.: Wissensverarbeitung und Expertensysteme. München/Wien 1989

Verein Deutscher Ingenieure (ed.): VDI-Richtlinie 5006: Bürokomunikation - Expertensysteme in betriebswirtschaftlichen Anwendungen, VDI, Düsseldorf, September 1992

Wallmüller, E.: Software-Qualitätssicherung in der Praxis. München, 1990

Winograd, T., Flores, F.: Understanding Computers and Cognition. Ablex, New York, 1986

Forschungspapiere der Wissenschaftlichen Hochschule für Unternehmensführung Koblenz

Lfd. Nr.	Autor	Titel
1	Weber, Jürgen	Theoretische Herleitung eines Controlling in Software-Unternehmen (Juni 1991)
2	Heinzl, Armin	Spinning off the Information Systems Support Function (Juni 1991)
3	Setzer, Ralf	Ergebnisse einer Befragung zur Beschaffungsplanung für zentrale Rechnersysteme (August 1991)
4.	Pfähler, Wilhelm Lambert, Peter	Die Messung von Progressionswirkungen (Oktober 1991)
5.	Pfähler, Wilhelm Lambert, Peter	Income Tax Progression and Redistributive Effect: The Influence of Changes in the Pre-Tax Income Distribution
6.	Pfähler, Wilhelm Leder, Thomas	Operative Synergie - von der Theorie zur Unternehmenspraxis (z.Zt. nicht erhältlich, wird überarbeitet)
7.	Wiese, Harald	Network Effects and Learning Effects in a Heterogeneous Dyopoly (Dezember 1991)
8.	Heinzl, Armin Stoffel, Karl	Formen, Motive und Risiken der Auslagerung der betrieblichen Datenverarbeitung (Januar 1992)
9.	Bungenstock, C. Holzwarth, J. Weber, Jürgen	Wegfallkosten als Informationsbasis strategischer Entscheidungen (Januar 1992)
10.	Lehner, Franz	Messung der Software-Dokumentationsqualität (August 1992)
11.	Heinzl, Armin Sinß, Michael	Zwischenbetriebliche Kooperationen zur kollektiven Entwicklung von Anwendungssystemen (August 1992)
12.	Heinzl, Armin	Die Ausgliederung der betrieblichen Datenverarbeitung

13.	Lehner, Franz	Expertensysteme zur Unterstützung der strukturorganisatorischen Gestaltung von Unternehmen
14.	Lehner, Franz	Brauchen wir eine Theorie der Wirtschaftsinformatik?
15.	Lehner,Franz, Setzer, Ralf Hofmann, Hubert	Wartung und Pflege von Wissensbanken
16.	Müller, Klein	Grundzüge einer verhaltensorientierten Preistheorie im Dienstleistungsmarketing
17.	Lehner,Franz	Considerations on Information System Strategies Based on an Empirical Study
18.	Lehner, Franz Hofmann, Hubert Setzer, Ralf	Maintance of Knowledge Based Systems

Rektorat / Mai 1993