WHY AREN'T SE TECHNIQUES BEING USED BY KBS SYSTEMS DEVELOPERS ALREADY ?

Michael Wilson

In order to describe current Knowledge Based Systems (KBS) development, and explain the elements of software engineering it does, doesn't, could and couldn't use, it is necessary to informally characterise software engineering itself, then compare KBS development with this. In conclusion, this paper will then suggest the directions KBS research is taking, and should move in future.

Software engineering methods can be viewed as five layers which constrain simple hacking. The layers have been individually developed over the last twenty years, and the early layers are more developed than the later ones. The first layer of software engineering technology is the use of life cycle models to structure software development projects. These allow managerial decisions to be made, and emphasise the consequences for later stages of early design decisions; the extreme case being the cost in maintenance of failures to develop systems to meet the real requirements of users. The second layer focusses on the design and implementation core of the life cycle and states the documentation and control procedures for version control, quality control, easier maintenance and project management. The third layer uses data from previous development projects as the basis for predictions of the time to develop systems, the cost and the resources required. In order to make such decisions in any engineering discipline it is necessary to collect data from a large number of commercial projects as has been done for software metrics such as the COCOMO model (Boehm, 1981). The fourth layer stipulates that the representations developed at each stage of the life cycle must be subject to processes of verification against previous representations and validation against the user or client's need for a solution to their problem. With this stage comes the introduction of notations for the representations and guidelines on the assessments which should be made. Many sets of guidelines and notations are available in structured development methods from Yourdon, Jackson, Mascot, SSADM and others. The fifth layer requires that these guidelines for verification become formalised, firstly so that the verification can be proven to be correct, and secondly so that the development stage itself is replaced by a transformation process which operates on one representation to produce the second.

Throughout the growth of these layers two specific processes operate: the development of agreed standards and the development of tools to support the methods. Initially tools were developed for implementation, then for version control, documentation and management forecasting; with the introduction of notations for the representations earlier in the life cycle for analysis and design; then to support formal notations and verification methods. Today tooling is moving towards integration with Integrated Project Support Environments (IPSE's) incorporating support for all stages of the development life cycle and the roles of all those involved in it. The first generation of IPSE's were little more than tailored databases, whereas third generation IPSE's will themselves include KBS techniques.

The objective of the engineering of software is to reduce uncertainty in the final product by reducing uncertainty in the development process. Given the progression of software engineering through these layers, the later stages of the life cycle have become less uncertain and the doubts remain at the early stages of requirements capture and initial program specification. To address these areas software engineering has incorporated prototyping and animated specifications so that user or clients can validate whether specifications meet their needs. Unless the initial problem is correctly identified, even a completely certain development process will produce inappropriate software.

This superficial summary of software engineering is obviously incomplete, has omitted many aspects of the field and is biased in the presentation of those it includes. It does not serve as a tutorial and does not substitute as an introduction for a recent textbook on the subject (e.g.

Macro and Buxton, 1987). However it does allow a contrast to be drawn with KBS development.

To contrast SE and KBS developments, KBS will be divided into four classes. Firstly, artificial intelligence (AI) systems (which are arguably not KBS) that are developed in academic or research environments as experiments to test an approach to complex reasoning or to specify it in detail. Not only is there rarely a complete specification or set of requirements for such systems as the software engineering paridigm requires, but they are usually produced in order to act as a specification themselves for future uses of an approach. These systems are not intended to be used by clients and rarely exist long enough to require any maintenance. When they do persist, they are modified to test other approaches or theories. Such systems are not compatible with software engineering since the maintenance puts few demands on them, they have very ill formed specifications, the only users are usually the developers so the process of verification is continuous, and uncertainty in the finished product is only bounded by the wishes of those developer/users. There is no need for standards to be applied to these systems, and the tooling is required to support experimentation rather than development estimates, quality control or project management. This class may seem irrelevant to practical KBS but all KBS were in this class fifteen years ago, and it is the properties of this class which illustrate why the origins of KBS are contrary to the spirit of software engineering.

The second class of KBS are first generation expert systems which are mostly stand alone systems that are consultative in operation. This class has had great publicity in the last few years and many commercial organisations have experimented with them with the result that a few are now in common commercial use. Systems of this second class have mostly been developed within the company which will use them as feasabality studies of KBS technology. As feasabality studies they have generally used the KBS tools which support design and implementation through prototyping but they have not been concerned with maintenance. However where such systems have become established, the problems of maintenance have arisen and the lack of the first level of software engineering which imposes a life cycle structure has become evident. If this class of system is to continue to be developed and exploited it is necessary not only to incorporate a life cycle approach to avoid maintenance problems, but also to include the second level of software engineering to monitor and control large development projects, and the third level to allow estimates of cost, resources and time.

The third class of KBS includes many cases of KBS which are similar to the AI experiments of class one in that the requirements are not defined and the exercise is not one of developing a system as much as exploring a problem or person's expertise to develop a set of requirements for a system which need not be built with KBS technology. Before the project starts not only is it accepted that the problem is one which appears to require intelligence in the solution, but also that when the problem is studied in detail using the knowledge acquisition and representation techniques used in KBS development the apparent need for intelligence will be reducible to a need for a conventional program. In these cases SE development methods can not be applied since they would rely on requirements to be complete, and the purpose of the project has been to develop such requirements which may then be used in a project using SE methods to develop conventional software. However, these systems also require estimates of cost, resources and time to be made so that contractual and managerial structures may be established for them.

Although some practical systems have been developed which are purely stand alone, the fourth class of KBS includes many applications which have shown that access to other programs and data are commonly required to make a usable system. For example, medical diagnosis systems were among the earliest consultative KBS but few are used in practice since they require clinicians to type in answers to a large number of questions about a patient's medical history. Many of these questions would not be required if diagnostic systems could read a database containing the patient's medical history, and the systems would require less input from clinicians before coming to a conclusion. Similarly, many successful KBS monitor equipment for their main data input or take the output of a testing device and interpret it. Systems which require equipment to be monitored are time critical to the same extent as many conventional software systems; those that use databases become part of a larger

conventional software development project to which they are closely bound. For KBS which are closely bound to conventional programs in many application sectors (such as health, defence, transport, plant control) the safety critical requirements on the overall program will also apply to the KBS component, in addition to the time critical requirements and the need to link the KBS development to a software engineering method. For such KBS to be developed practically, not only are the first three levels of software engineering required as in stand alone KBS but also the fourth and fifth levels (to incorporate a structured development method and formal verification) since the KBS must be closely coupled with the development of conventional systems.

The demands for this class of system clearly pose a dilemma: the need for a KBS presupposes that the problem is not well understood but involves intelligent behaviour for which the requirements cannot be clearly expressed, whereas the contractual demand is that the system be built on schedule, to budget with assurable reliability according to a development method which relies upon clearly stated requirements. The lack of a resolution to this dilemma is the major reason why KBS developments of this promising class of system do not use SE methods.

Research on linking SE and KBS development methods.

The first class of KBS do not require SE methods, but rely on inspired hacking, or the Run-Understand-Debug-Edit cycle of development to explore problems (Partridge and Wilks, 1987).

The second class of KBS require the first level of SE to provide a life cycle approach to avoid maintenance problems, but also the second level to support the monitoring and control of large development projects, and the third level to allow estimates of cost, resources and time. However, this class of KBS also require the knowledge acquisition and representation components of KBS development. Therefore, for the introduction of software engineering techniques into the second class of KBS development we need to understand how the present KBS life cycle differs from that of conventional programming, what those differences offer as benefits, and how these benefits can be retained while introducing more structure to the process. The KADS (Hayward, 1987) project has developed a KBS development method incorporating constraints of documentation and version control procedures to aid project control along with the KBS methods and tooling for them. However, KADS does not include the third level of SE allowing estimation of cost, resources and time for projects since insufficient data is available so far on which to base predictions. Other drawbacks with the KADS approach are that it has lost one of the stereotypical aspects of KBS development in that it does not include prototyping, and that its use of knowledge acquisition techniques is limited to verbal data. However, KADS does provide a basic development method for this class of KBS.

The third class of KBS do not require SE techniques, but indeed could be incorporated into SE development methods themselves in order to address the problems of requirements capture and initial specification.

The fourth class of KBS require the all levels of SE development and would therefore also benefit from integrated tooling with conventional SE methods. Keller (1987) has adapted the Yourdon development method for conventional software to address the sub-class of these KBS which use databases. However he has merely added a strand of knowledge acquisition in parallel to the conventional database program development method. This technique has severally limited the aspects of KBS development included in the approach and the classes of problem it can address. The KADS method is not appropriate for this class of system because its knowledge acquisition is limited to verbalisable reasoning which is not likely to apply to many real time problems, and because it is not integrated with a conventional development method. Another recent project GEMINI (Montgomery, 1989) is intended to incorporate SSADM into a KBS development method and include prototyping as one form of verification. Since this project has recently begun it is not yet possible to predict what it will offer for tooling or the breadth of knowledge acquisition included. Obviously, no form of project estimating is possible within this approach yet, although this may come with further use of the method.

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

This is only a sample of the attempts to understand the details of the present KBS life cycle and how it could be combined with SE approaches (e.g. Wilson et al, 1988). It is clear that there are some classes of KBS development where SE techniques are inappropriate, but the most promising areas of KBS development could use SE methods if the reliance on initial specifications could be overcome. However, even if these succeed there will remain an outstanding need to provide systematic tool development to support the whole life cycle, and data on successful and failed large KBS developments to allow estimation of time, effort and manpower on projects before software engineering methods can be used for KBS development.

References

Boehm, B.W., 1981. *Software Engineering Economics* Englewood Cliffs, NJ, USA: Prentice Hall.

Green, C.J.R. and Keyes, M.M., 1987. "Verification and Validation of Expert Systems". In *WESTEX 87 - Proceedings of the Western Conference on Expert Systems,* 38-43. Washington, D.C: IEEE Comput. Soc. Press.

Hayward, S., 1987. "How to build knowledge based systems: techniques, tools, and case studies". In *ESPRIT '87: Proceedings of the Esprit Conference,* pp 665-687. Brussels: Commission of the European Economic Community.

Keller, R., 1987. *Expert System Technology: Development and Application.* Englewood Cliffs, NJ, USA: Yourdon Press.

Macro, A. and Buxton, J., 1987. *The Craft of Software Engineering* Reading, Mass.: Addison-Wesley.

Montgomery, A., 1989. "GEMINI: Government Expert Systems Methodology Initiative". In B. Kelly and A. Rector (Eds.) *Research and Development in Expert Systems V,* pp 14-24. Cambridge: Cambridge University Press.

Partridge, D. and Wilks, Y., 1987. "Does AI have a Methodology which is Different from Software Engineering?" *Artificial Intelligence Review* **1** (2), pp 111-121.

Wilson, M.D., Duce, D.A. and Simpson, D. (1988) Life cycles in Software and Knowledge Engineering: A comparative review. *Knowledge Engineering Review.* **3** (4)