Knowledge Based System Verification and Validation as Related to Automation of Space Station Subsystems: Rationale for a Knowledge Based System Lifecycle

Keith Richardson
Carla Wong

Systems Autonomy Demonstration Project Office
NASA/Ames Research Center
Moffett Field, CA 94035

Abstract

The role of Verification and Validation (V&V) in software has been to support and strengthen the software lifecycle and to ensure that the resultant code meets the standards of the requirements documents. Knowledge Based System (KBS) V&V should serve the same role, but the KBS lifecycle is ill-defined. The rationale of a simple form of the KBS lifecycle is presented, including accommodation to certain critical KBS differences from traditional software development.

Introduction

This paper discusses the rationale for using a KBS lifecycle to solve some problems of KBS V&V, describes the agents of this process, and presents three key aspects of variance from traditional software lifecycles. The KBS lifecycle is described in greater detail in a longer paper titled: "Workshop on Verification and Validation of Knowledge Based Systems: Recommendations for Procedures and Research."

V&V is the process of overseeing the construction and testing of a software program according to specifications. The purpose of V&V is to ensure that software is:

1) Designed to be testable,
2) Completed according to defined specifications which lead to testability,
3) Tested according to requirements.

The software lifecycle is used to structure this process. Since such a lifecycle has not been generally accepted for KBS it was natural that one of the major topics of conversation during the Ames KBS V&V workshop was definition of a KBS lifecycle. (Two proposals which outline some aspects of KBS lifecycle are discussed in the workshop summary.)

The KBS lifecycle presented here, however, addresses V&V issues in the context of a complete, high-level, lifecycle description. This lifecycle may be seen as the simplest form, applicable to a single KBS acting without interaction with other systems, but including the human interface. Some other complex situations will only need reasonably straightforward adaptations of the lifecycle, while others using new AI methodologies or with

interdependencies to external systems will not be accommodated by a KBS lifecycle without the benefit of further experience and research.

For the purposes of this paper the lifecycle is further simplified to the following structure:

- Requirements Phase
- Prototype Phase
- KBS Build Phase
- Test Phase
- Delivery and Monitor Phase

The basic format of the KBS lifecycle is derived from traditional software lifecycles, both for the reason that these lifecycles are better understood by the sponsoring organizations and software developers and because there are many aspects of KBS software methodology which are similar, or the same as, traditional software. Where software development methods are similar it is cost effective to capitalize on this extensive experience in traditional software.

Satisfying Agents of KBS Development

Among the critical problems which must be solved by a KBS lifecycle is the satisfaction of all the agents in the process. The major three agents are:

1) The agents who sponsor the project: the organization, managers, and investors.
2) The agents who build the KBS, the domain experts, knowledge engineers, test engineers, etc.
3) The end users of the KBS.

A person may serve as more than one agent in the KBS development.

The reason for satisfaction of the first set of agents is self-evident.

Satisfaction of the KBS builders is necessary because there are some aspects of the KBS development only they are familiar with - not every programming decision can be recorded. Some have suggested that independent testing and V&V agents will have a better understanding of the KBS than the builders; what they mean is that certain blind spots in the developer's perception are better discovered by people with different blind spots. Among the ultimate authorities for the integrity of a KBS must be the people who have been involved in the experimental process of prototyping - no other group will understand design decisions in such precise detail. For this reason the KBS lifecycle should support testability as defined by the KBS requirements, and support only secondarily independent code evaluators.

The third set of agents is the users. User acceptance is the final step in the KBS process before the KBS will be used and can be called successful. The importance of the users to the acceptance of a KBS can be more important than in acceptance of traditional software in three related ways:

1) Some users, e.g. astronauts, mission controllers, or doctors and lawyers, (and their professional organizations, the AMA, and ABA) have almost complete authority to decide whether software will be used in their domain - independent of their ability to substantiate their concerns.

2) A KBS may need to perform a range of functions, such as diagnose, monitor, control, educate, etc. Which of these functions are important, and in what manner they may be fulfilled is largely the decision of the users.

3) Users must evaluate how effectively they are able to use the KBS information in their real world domain.

The satisfaction of these three sets of agents, the sponsors, the builders, and the users, is the goal of the KBS lifecycle. This satisfaction should provide direction to all phases of the KBS lifecycle, and each stage must contribute to the end result of the process.

Three Aspects of the KBS Lifecycle

Among the important aspects of adaptation of traditional lifecycles to KBS development are:

I) Changes and improvements to requirements documents.
II) Addition of a reiterative prototyping loop.
III) Inclusion of the user in the development process.

I) Requirements Documents.

Some in the Ames workshop felt that the documents which are produced before prototyping in the KBS development process needed to be more completely specified - some went as far as to say that inadequate requirements were the primary cause of the failure of KBS development efforts. Others insistently pointed out that the result of the experimental process cannot be predicted before prototyping, and that precise specifications were unlikely or impossible.

Issues can be confused by this simple polemic distinction between requiring more or less thorough requirements documents. The suggestion made by this paper is that a different sort of requirements document is necessary which at once suits the experimental nature of KBS development, the pragmatics of funding and time constraints, and the necessity to be able to test function and assess the KBS success.

Experimental concerns dictate that the KBS development process be as unconstrained as possible and able to exploit fortuitous discoveries during the prototyping phase. Because neither the domain experts or the knowledge engineers have a precise idea of which experimental areas will be practicable at the beginning of the prototyping phase, enough time and resources must be available to pursue the certain number of investigations which will not contribute directly to the final KBS. Any requirements documents should be written to accommodate the prototyping process, and should specify money, time, and other resources available for prototyping.

Constraints dictated by KBS development sponsors, besides including restrictions on resources, should also specify as closely as possible criteria for success. In a "proof-of-concept" KBS development project, such as the Systems Autonomy Demonstration Project, the criteria for success will be the proof or dis-proof of the practicality of KBS application in an environment generally - no specific function may be required of the final stage of the KBS. A commercial KBS project intended for delivery and/or sale will be measured by utility functions such as: customer satisfaction, number of units sold, profit, repeat sales, maintenance costs, etc. KBS intended for NASA will use various criteria of success which will depend on program goals, intended use, hazard analysis, etc.

The requirements documents, where it measures KBS success in terms of utility, must take in account the uncertain process of the prototyping phase. It should include statements of goals as well as of resources. The format which is liable to be crucial to KBS development is the specification of a range of results, or a number of solutions. The sponsors of KBS development need to put considerable thought into the various combinations of constraint satisfaction which will prove satisfactory to them.

II) Reiterative Prototype Loop

Incremental refinement in prototyping is an inseparable feature of KBS development. Traditional software management is uncomfortable with this seemingly open-ended process, but costs and other resources can be controlled by astute specification of requirements.

The prototyping loop has also caused controversy in that the documentation style of a traditional software prototype seems to be overly cumbersome for KBS development. Traditionalists are uncomfortable with less than "complete" documentation, while KBS developers point out that design decisions early in the prototyping process which are later superceded are expensive to document and do not serve in a direct way to satisfy requirements. A goal should be that complete documentation be kept of any aspect of the project likely to be incorporated into a final version of the code. An experimental notebook may be an example of a format which satisfies all parties; decisions on what to include will be made partly by the prototype development team, and partly by system requirements.

The KBS prototype loop's duration and experimental extent will be limited by the requirements specification.

Where the requirements documents are intentionally incomplete one of the goals of the prototyping phase is to add further details to them. Thus a complete loop in the prototyping phase involves:

1) Knowledge Acquisition/Extraction.
2) Building of a Prototype.
3) Evaluation of Results.
4) Augmentation of Requirements Documents.
5) Decision on Direction of Next Prototype.

Decisions about project directions are made on the basis of newly revised requirements documents, and experimental results from the most recent prototype, and are used to determine whether: a) another prototype is needed, b) the project should be discontinued, being unable to satisfy requirements, c) the project should continue into the next lifecycle phase, which is to build a less experimentally oriented KBS which will serve as a platform for associated development efforts in interfaces, testing plans, user facilities, formal documentation, etc.

Following prototyping, changes to fundamental aspects of the KBS become less practical. During the prototyping, efforts to KBS development are restricted so that they rely on mostly inflexible assumptions, after prototyping, KBS-related efforts rely on the stability of the prototype itself. Major design decisions should be made during prototyping phase, later changes are possible, but at greater cost for associated efforts.

III) User in the Loop.

Another major difference between traditional software lifecycles and KBS lifecycles is the increased involvement of the user. It has been suggested that the end user of a KBS may have much discretionary power in approving use of a KBS. This hurdle can be anticipated and surmounted by including users in the development process.

In traditional software lifecycles precise user requirements can be included, or tacitly implied, in the requirements documents; but these documents can remain incomplete in the KBS lifecycle until a fairly late stage. At KBS requirements specification the incomplete user input is mitigated by the presence of the domain expert during prototyping, who in some ways will typify the concerns of the general user.

In traditional software another point of user input in the development process is the Alpha or Beta test, where a nearly finished version of the software is placed in an environment typical of expected use. In a KBS it is much more difficult to define "typical environment" and "expected user". One solution, indeed, is to limit KBS use to narrowly defined situations. An approach which is more appropriate for a KBS to be used in a variety of situations, and by users of varying skills, is to attempt to accommodate as many environments as possible. These accommodations will start by a direct user contribution to the requirements documents.

To the extent a KBS is liable to be used in situations which were unanticipated by the designers, or which change over the lifespan of KBS use, it is necessary to extend the KBS lifecycle to include user reaction and comment after KBS delivery. The overhead incurred by this additional burden should be designed to be of less cost than the combined benefit: to current users, to addition of new areas of utility, and of the value of direct user input for later versions of the KBS.

User interaction is liable to be least effective in the prototyping phase, where the state of the development process is complicated to explain , and the state of domain knowledge is non-coherent. The writing of requirements documents, however, provides an opportunity in that at that early point it is efficient to catch misconceptions about expectations of the user community, and relatively easy to explain project plans. The user will also have an important perspective on utility tradeoffs among various possible development options.

Conclusion

These changes, and others which can be found in our related papers, do not represent a complete plan for a KBS lifecycle. The KBS lifecycle will be improved with experience following these fundamentally important changes, especially in areas where there is not an analogous process in traditional software development. The KBS development process also will benefit from a significant amount of research over the next few years to improve predictability, costing, enhanceability, and so on. Research suggestions include improvements to supporting hardware and software, new KBS tools, automated testing facilities, and continued improvement to the KBS lifecycle.

Good V&V of KBS will follow improved specification and control of the KBS development process. Since the KBS V&V process is now being formalized it will be in the position to accommodate the latest software concerns; if the KBS V&V process seems difficult it is partly because more stringent demands are now being made on all software development. KBS V&V, then, in some ways will be the first to incorporate new standards of quality for software.

Acknowledgements

The technical assistance and comments of Dr. H. Lum, Dr. P. Friedland , and M. Dutton were much appreciated in the production of this paper.