N93-32156

The Knowledge-Based Software Assistant: Beyond CASE

Joseph A Carozzoni
Rome Laboratory: Knowledge Engineering Branch
Griffiss AFB, NY 13441-5700
carozzoni@aivax.rl.af.mil

Abstract

This paper will outline the similarities and differences between two paradigms of software development. Both support the whole software life cycle and provide automation for most of the software development process, but have different approaches. The CASE approach is based on a set of tools linked by a central data repository. This tool-based approach is a data driven and views software development as a series of sequential steps, each resulting in a product. The KBSA approach, a radical departure from existing software development practices, is knowledge driven and centers around a formalized software development process. KBSA views software development as an incremental, iterative, and evolutionary process with development occurring at the specification level.

1. Introduction

Attempts to solve the software crisis have varied from philosophies of management and disciplines of programming to new languages and tools. Many of these innovations have found their way into integrated environments and defined as Computer Aided Software Engineering (CASE). The outcome of these approaches were minor gains in productivity, reliability and maintainability. Although additional improvements may be achieved by continuing in this direction, the order of magnitude improvements needed to address the software crisis are not likely to be realized.

A major problem with present design and implementation activities is an informal development paradigm based on a series of sequential phases. The design rationale involved in the creation of software is lost once the initial implementation is completed. Additionally, modifications to completed systems are performed at the source code level where design information has been obscured by implementation and efficiency considerations.

Recognizing these problems, Rome Laboratory has undertook a program using technology from automatic programming and artificial intelligence to develop a knowledge based system that addresses the entire software life cycle. The Knowledge Based Software Assistant (KBSA), a retreat from pure automatic programming, is based on the belief that by retaining the human in the process, many of the unsolved problems encountered in automatic programming may be avoided. It proposes a new software development paradigm in which software activities are machine mediated and supported throughout the life cycle.

The underlying concept of KBSA is that the software development process will be formalized and automated. This will allow a knowledge base to evolve that will capture the history of the software development process and allow automated reasoning about the software under development. The impact on the software development process is that software will be algorithmically derived from requirements and specifications through a series of user guided formal transformations. Maintenance will occur at the requirements and specification level and the implementation process will be "replayed" as needed.

2. CASE: The Tool-Based Approach

Early efforts to automate software development focused on providing an environment of individual tools designed to handle particular activities. This early approach inspired by DoD 2167 requirements was data and product based. It ignored the software development process. A typical model of an early software development environment is the Unix operating system and a number of associated tools such as text editors, language compilers, the make facility, lint syntax checker, debugger, SCCR version control software, etc. Early tool-based approaches produced modest improvements in software productivity, but tended to ignore the growing crisis in software maintenance. The major drawbacks to this kind of environment were the lack of integration between the tools, and the requirement that the tools be used sequentially. Communication between the various tools relied on the host file system.

Efforts to improve the tool-based approach focused on creating a tighter integration of the individual tools. Reliance on the host file system gave way to more sophisticated "common repositories" such as relational databases. Representative of the improved tool-based approach is Integrated Case (ICASE), Portable Common Tool Environment (PCTE), and Software Engineering Environments (SEE). The primary goal of this evolving paradigm is standardizing the data exchange to allow inter-operability between differing tools and environments. Central to the evolution of the current CASE approach is increased sophistication of the central data repository. Still, mainstream CASE is data and product oriented.

3. KBSA: The Knowledge-Based Approach

While other technologies have readily adapted to production engineering techniques, software has resisted because the creative processes are informal and unspecified. Any solution to the software crisis must address the human intensive facets of software development such as conceptualization and reasoning. Traditional software technology has neglected to address issues dealing with the process of software development. Rather, it has been immersed in addressing the products (ie. written documents, program form, programming languages, management structure, metrics, etc.). Without a paradigm shift from product orientation to process orientation, software development and maintenance will never keep pace with increasing demands.

With this in mind, KBSA has broken away from the traditional tool-based approach in favor of a knowledge based approach to software development. Early attempts at pure automatic programming indicated that the knowledge of programming was still too immature to replace all of the human involvement. With pure automatic programming out of reach, KBSA sought near term relief by keeping the human in the decision making process. Figure 1 is the KBSA process model.

Under the KBSA paradigm, software activities are machine mediated and supported throughout the life cycle. By formalizing and automating the software development process, a knowledge base can evolve that will capture the history of the life cycle processes and support automated reasoning about the software under development. The payoff of this process formalization is that software can be derived from requirements and specifications through a series of human assisted formal transformations. In addition, requirements validation is automatic since the formal transformations have the property of being correctness preserving.

With this new paradigm, maintenance becomes analogous to development and will be performed at the requirements and specification level. Validation and verification are supported as it will be possible to "replay" the process of implementation as chronicled in the knowledge base. KBSA will provide a corporate memory of how objects are related, the reasoning that took place during design, the rationale behind decisions, the relationships among requirements, specifications, &

code, and an explanation of the development process. This assistance and design capture will be accomplished through a collection of life cycle activity facets, each tailored to its particular role. These facets will be highly integrated within a common environment.

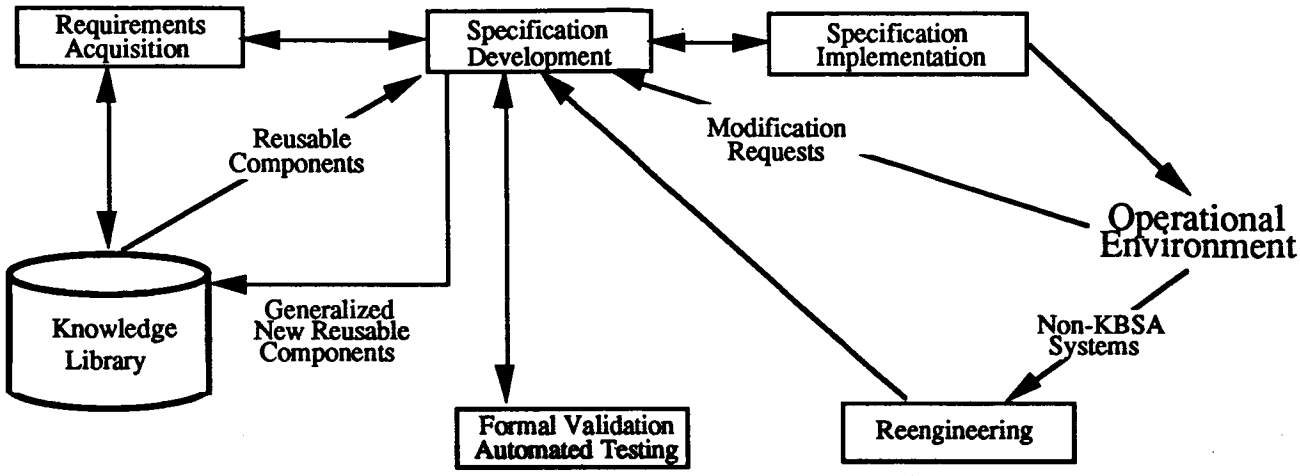

Figure 1: The KBSA Process Model

As envisioned, the KBSA environment will allow design to take place at a higher level of abstraction than is current practice. Knowledge based assistance mediates all activities and provides process coordination and guidance to users, assisting them in translating informal application domain representations into formal executable specifications. The majority of software development activities are moved to the specification level where early validation is provided through prototyping, symbolic evaluation, and simulation. Implementations are derived from formal specifications through a series of automated, meaning-preserving transformations, verifying that the implementation correctly represents the specification.

Post deployment support of the developed application is also concentrated at the requirements/specification level with subsequent implementations being efficiently generated through a largely automated "replay" process. This capability provides the additional benefit of design reuse as families of systems are spawned from the original application. Management policies are also formally stated, enabling machine assisted enforcement and structuring of the software life cycle processes.

The techniques for achieving these goals are:

- Formal representation and automatic recording of all the processes and objects associated with the software life cycle.
- Extensible knowledge based representation and inferencing to represent and use knowledge in the software development and application domains.
- A wide spectrum specification language in which high level constructs are freely mixed with implementation level constructs.
- Correctness preserving transformations that enable iterative refinement of high level constructs (specifications) into implementation level constructs (HOL or machine code) as KBSA carries out the design decisions of the developer.

4. Similarities between KBSA and CASE

Currently, software development is a human-intensive task. KBSA and CASE share the high level goal of facilitating software development by automating much of the work that is presently performed by humans. Both provide automated support for software development by encompassing the entire software development life cycle.
A key area in software development enjoying an increase in appreciation is reuse. If unaltered reuse is not possible, then the next step is assistance in taking something you already have in-hand and modifying it to fit your needs. Reuse has the advantage of being fast and the products mostly debugged. Both KBSA and CASE place significant emphasis on reuse.

Both address the life cycle from various view points which are connected by some form of centralized repository of information. Other areas of commonality of KBSA with some of the more robust and comprehensive CASE environments are automated assistance for:

- Documentation generation (e.g. DoD 2176A).
- Sharing and locking mechanisms for concurrent development.
- Expanded view beyond just software to address the development of systems.
- Graphical interfaces to ease development.
- Consistency checking (pre-defined and user-definable).
- Code and data generation.
- Configuration management.
- Prototyping tools.
- Project management facilities (planning, monitoring, and resources)
- Traceability.
- Testing.

KBSA and CASE have similar goals and attempt to enhance software development. Where they diverge is in their respective orientations (product versus process) and the level to which support is provided.

5. Differences between KBSA and CASE

The major differences between KBSA and CASE originate in the choice between product and process oriented development. Some CASE vendors mistakenly regard their approach as being process oriented, when in fact it is product oriented. Most CASE tools implement some derivative of the Waterfall model for software development This model was developed in the late 1960's in response to the growing software needs of DoD, and can be described by three basic characteristics:

- Software development is a sequence of several discrete stages.
- The product of each stage is documentation which in turn feeds the following stage.
- After each stage is completed it must be validated and verified against the prior stage.

There are many inherent problems in this model which impede software development. These problems occur because the Waterfall model is product based and fails to address the process of software development itself. The basic assumption that software development is a relatively uniform and orderly sequence of development steps is flawed. The extreme sensitivity to task sequence in this model does not provide direct support for modern methods of software development such as rapid prototyping and evolutionary development. Incremental, iterative, and evolutionary design is central to successful software development.

KBSA has put aside the product orientation and has placed focused instead on automating the software development process. A major feature of the KBSA approach is its formalized software development process. With the formalized life cycle, the machine captures all software decisions and provides knowledge-based reasoning assistance. This provides a "corporate memory" of the development history and knowledgeable assistance to humans throughout the life cycle. With this change in focus, KBSA has introduced a paradigm shift from addressing low-level programming problems to addressing high-level knowledge acquisition problems.

CASE has evolved from individual tools targeted for individual activities to an integrated set of tools targeted at those same activities. The accomplishment of ICASE has been to smooth out inter-tool communication. KBSA is a much deeper integration of the software development activities. While integration of CASE has occurred with data, KBSA has also integrated knowledge (about the software development process). The key to the flexibility and robustness offered by KBSA is this highly integrated environment.

KBSA offers a form of prototyping which is more robust than that offered by CASE. Present CASE prototyping is user-interface oriented opposed to the incremental, iterative, and executable design prototyping provided by KBSA. CASE typically has little semantic information which limits the sophistication of the analytic tools and has little automatic support for code generation and reuse. KBSA has rich semantic knowledge about the programming domain and the decisions made during system development which permits sophisticated analysis and has automated support for implementation and reuse.

The diagrammatic tools common in CASE environments use informal specifications and only represent the syntactics of a problem. In addition, the informal specifications used by the individual tools do not allow various views to be related and managed in a conceptually clean and consistent manner. The KBSA knowledge-base integrations varying views (different methodologies) into a single, unified formal representation. This single, unified formal representation captures both the syntactics and semantics of a specification. While CASE developers are typically restricted to a single methodology, KBSA developers can work in a mixed methodology environment. This also provides excellent support for concurrent developers.

Another difference is the manner in which reuse is addressed. CASE addresses reuse at the code level. During the coding phase, a programmer attempts to find a code module which most closely matches the design at hand. Finding a suitable code module for reuse can be time consuming. Even if one is found, modifications may have to be performed to fit the intended design. KBSA addresses at a higher level of abstraction and focuses on design reuse. By concentrating on design, reuse occurs during the requirements and specification phases. Design reuse is more direct and efficient than code module reuse. Code itself is not even part of the KBSA paradigm.

Post deployment maintenance now accounts for 80% of all software costs. The Waterfall model used by most CASE environments performs maintenance at the code level and as a separate process. In KBSA, maintenance is part of the evolutionary development process. Thus maintenance is performed at the requirements and specification level. In addition, this approach lends itself well to support reverse engineering. Legacy software systems can be analyzed and "feed" back into the KBSA process model as specification development. The specifications of legacy software can be recovered integrated with new software developments.

6. What Limits the Capability of a S/W Development Environment?

The KBSA approach to software development and maintenance greatly surpasses present state-of-the-art CASE tools. Some of the technological areas where KBSA excels are:

- The level of integration of the environment.
- The handling of informal requirements.
- The transformation of requirements to specifications.
- The types of specifications which may be used.
- If specifications are automatically maintained consistent with each other.
- The level to which specification consistency is addressed (syntactic versus semantic).
- The manner in which specifications are validated.
- The potential for optimization of the final system.
- Project management functions provided.
- The level to which project management functions are integrated and enforced.
- If the paradigm is product versus process driven.

Level of Integration: The major limiting factor in a software development environment is the level to which the system is integrated. The level of integration can be broken down into three primary levels. Early CASE tools represent the lowest level of integration where only data is shared via the host file system. The tools may or may not be stand alone products. The Unix operating system and its standard set of "bin tools" is a good example. The next level of integration encompasses a set of tools (i.e. ICASE) which communicate through a "common repository", normally a standard relational database. Common to both levels of integration is the reliance on "data" as the inter-tool communication medium. KBSA represents the highest level of integration where not only data, but knowledge (of the software process) is dealt with. Integrating both knowledge and data at the repository level requires a much more sophisticated knowledge representation schema than CASE repositories currently allow. KBSA provides robust inferencing and knowledge representation services.

Requirements Acquisition and Implementation: Eliciting requirements from users is difficult and imprecise. Discrepancies easily arise because the user and the developer communicate from different perspectives. Informal requirements are normally formed out of randomly ordered English text. This informal description of system behavior must be coordinated into more organized and structured requirements. The requirements must then be transformed into more precise and less ambiguous specifications. KBSA provides knowledge-based assistance for requirements acquisition (reference the following paper on ARIES in the proceedings).

Transformation of Requirements to Specifications: CASE informally transforms requirements to specifications using a human mediated conversion. This conversion is imprecise and looses requirements traceability links. KBSA uses formal transformations which have a sound mathematical basis. The transformation is machine mediated, correctness preserving, and does not loose requirements traceability links.

Specifications Types: Specifications may be informal, semi-formal, or formal. Informal and semi-formal specifications do not have a complete mathematical basis and cannot be handled with automated theorem proving techniques. Additionally, informal specifications only allow syntactical analysis and cannot offer semantical analysis. Examples of informal and semi-formal specifications are textual statements of work, pseudo-code, etc. Examples of formal specifications are the formal specification languages Z, VDM, CSP, and temporal logic. KBSA is based on formal specifications, and uses knowledge-based techniques to hide the complexity from the user. The use of formal specifications allow KBSA to perform both syntactical and semantical analysis

Specifications Consistency: The degree to which specifications can be maintained consistent with one another is limited by the underlying representation schema. Informal specifications require human mediated consistency analysis which is time consuming and error prone. The use of formal specifications in KBSA allow machine mediated consistency analysis which is automatic and

precise. Maintaining specification consistency is complicated if the environment allows the use of multiple views (mixing of methodologies). The use of formal specifications allow KBSA to integrate multiple views into a *single formal representation.*

Specification Validation. Specifications that were informally derived must be manually validated against requirements. Because the specifications in KBSA are derived by a formal transformation of the requirements, validation is automatically ensured.

Optimization Potential: Two aspects of optimization are program size and execution speed. The level to which optimization can be achieved is dependent on where and when the optimization effort originated. Normally, there are two opportunities where optimization can be performed. The first chance to optimize is with the specification and/or algorithm itself. The other occurs during coding. While current compiler technology is very good at optimization of code, the optimization of the specification and/or algorithm has not been integrated into CASE environments. KBSA has integrated facilities to address specification and/or algorithm optimization.

Project Management Functions Provided: Project management is often considered a separate activity. The result of using a separate project management system is inaccurate and untimely reporting of data. Additionally, it is easy for programmers to bypass project management efforts by inaccurately reporting data. In KBSA, project management is integrated into the knowledge base. This provides instantaneous and accurate reporting, and eliminates attempts to bypass the system. Also, KBSA is being extended to include knowledge-based estimation and forecasting.

Product versus Process Orientation: Software development has not been adaptable to standard production oriented techniques, thus the actual process of software development itself must be addressed. Product-oriented techniques focus on software development as an assembly line operation consisting of sequential phases. Process oriented techniques address software development as evolutionary and transformational. A common misnomer is to label the "process of generating the individual products" as being process oriented, while the actual process of software development itself remains assembly line oriented. KBSA is truly a process-oriented approach to software development.

7. Conclusions

Both KBSA and CASE are meant to improve software development, yet have significant differences in their approaches. The underpinnings of CASE is a central data repository for inter-tool communication and data sharing. In current tools, this repository is typically a relational database. Most CASE tools, whether following the waterfall, spiral, or comparable methodologies, use a phased approach to software development based on a series of sequential steps. This data driven model views the software development process as the generation of individual products (e.g. requirements document, design document, code, etc.). The central data repository in the present CASE model can store only meta-data (data about data), and facilitates control over access to that data. This product oriented model is insufficient and lacks the expressive power required to successfully attack the software crisis. KBSA has a much deeper and richer representation schema at the repository level. KBSA is a knowledge-based approach which includes not only data, but also contains knowledge of the software development process itself. Table 1 highlights significant differences between KBSA and CASE.

As envisioned, the KBSA environment will allow design to take place at a higher level of abstraction than is current practice. Knowledge-based assistance mediates all activities and provides process coordination and guidance to users, assisting them in translating informal

AREA	CASE	KBSA
Development	Phased	Transformational
Prototyping	User interaction; rapid mock up	Incremental, iterative & evolutionary; executable specifications
Validation	Code against intent	Specification against intent
Specification Implementation	Manual (AD HOC)	Automated (provably correct)
Verification	Structural and functional testing of code modules	Minimal; correctness preserving transformations
Documentation	Only products; design history lost	Automatic and current; history and rationale captured
Development Emphasis	Documentation and coding	Executable specification integrity and system evolution
Maintenance	Code patching	Specification revised, then development replayed

Table 1: CASE versus KBSA

application domain representations into formal executable specifications. The majority of software development activities are moved to the specification level as early validation is provided through prototyping, symbolic evaluation, and simulation. Implementations are derived from formal specifications through a series of automated meaning preserving transformations, insuring that the implementation correctly represents the specification. Post deployment support of the developed application is also concentrated at the requirements/specification level with subsequent implementations being efficiently generated through a largely automated "replay" process. This capability provides the additional benefit of reuse of designs as families of systems that can spawn from the original application. Management policies are also formally stated enabling machine assisted enforcement and structuring of the software life cycle processes.

References

1..Green, C. et al., "Report on a Knowledge-Based Software Assistant," RADC Tech. Report TR-83-195, RADC, Griffiss AFB, NY, Aug, 1983.
2..Jullig, R., et al., "KBSA Project Management Assistant," Final Technical Report, RADC, Griffiss AFB, NY, July 1987, TR-87-78 (two volumes).
3.."Knowledge-Based Specification Assistant," Final Technical Report, RADC Contract F30602-85-C-0221, June, 1988.
4..Larson, A. and Huseth, S., "KBSA Common Framework Implementation," RADC, 2nd Annual KBSA Conference, Utica, NY, Aug 18-20, 1987.
5..Sanders Associates, "Knowledge-Based Requirements Assistant," Final Technical Report, RADC Tech. Report, TR-88-205 (two volumes), Griffiss AFB, NY, Oct., 1988.

Epilogue: For more information, one may attend:

The 7th Knowledge-Based Software Engineering Conference
September 20-23, 1992
Sponsored by Rome Laboratory; In cooperation with ACM, IEEE, and AAAI
McLean Hilton at Tysons Corner; McLean, Virginia
Tel: 315-336-0937 (Barb Radzisz)
EMAIL: kbse7-request@cs.rpi.edu