Educating Knowledge-Based Software Engineers

Paul D. Bailor
Department of Electrical and Computer Engineering
Air Force Institute of Technology
Wright-Patterson AFB, Ohio 45433

Abstract

This paper is an attempt at defining the educational requirements for the field of knowledge-based software engineering (KBSE). By analyzing an updated version of the program transformation system paradigm, ten topic areas reflecting KBSE educational requirements are identified. It is shown that many of the ten topic areas can be satisfied using courses already available in most graduate level computer science/engineering programs when examples and course projects are specifically directed at KBSE concepts. A comparison of the ten KBSE topic areas to the 21 software engineering education topic areas developed by the Software Engineering Institute (SEI) shows that few of the KBSE requirements are satisfied by the 21 SEI topic areas. Additionally, recent efforts at the Air Force Institute of Technology show that a KBSE program, including laboratory support, can be quickly started.

1 Introduction

Unfortunately, the field of software engineering is still not precisely defined, but the work that has been done to define the educational requirements of a software engineer has been an extremely beneficial first step. The Software Engineering Institute (SEI) has been working for the last five to six years on defining the educational requirements of a software engineer [1]. One of their results has been the definition and establishment of a masters degree program offered at Carnegie Mellon University [2]. Additionally, several other universities throughout the United States offer software engineering programs that generally cover the same requirements [3]. Shaw [4, 5], points out that software engineering will require explicit recognition of specialties that also require corresponding educational requirements and opportunities. The application of artificial intelligence (AI) technology to software engineering is one such specialty, and in fact, knowledge-based software engineering (KBSE) has been loosely defined as the application of AI technology to software engineering [6].

Over the past fifteen years, many researchers have investigated applying AI technology, especially in the area of knowledge-based systems, to the task of software engineering [6, 7]. One such effort is the Knowledge-Based Software Assistant Project (KBSA) [8, 9] being conducted by the United States Air Force's Rome Laboratory which has recently announced the availability of a working prototype [10]. The basis for the KBSA project is the program transformation system paradigm outlined by Balzer, Cheatham, and Green [11]. This paradigm proposed the use of formal specification languages and automated assistants to help the software engineer define and apply a sequence of behavior preserving transformations to a validated formal specification in order to obtain an effective, efficient, and consistent software system in some target language. This approach moves the use of formal languages and automated language transformation tools from the programming language implementation phase to the initial requirements analysis and specification phases of software development.

This paper uses the program transformation system paradigm proposed by Balzer, Cheatham, and Green [11] as the beginning basis for defining the educational requirements of KBSE. This basis is established in the next section, and a not too surprising discovery is the multi-disciplinary nature of KBSE. Because of space limitations, I am going to limit most of the discussion to the key practice area of software product engineering. That is, the technical issues associated with the development of software systems. There also exist software project management key practice areas such as configuration management and quality assurance that can also benefit from knowledge-based technology. For example, the project management assistant that is part of the KBSA project [9].

In developing the basis, two assumptions are made. First, it is assumed the reader is familiar with the
contents of the paper by Balzer, Cheatham, and Green [11]. Second, it is assumed the reader is already an advocate of KBSE and does not have to be convinced of its merits. Readers needing to be convinced should begin by reading the foreword, introduction, and Chapter 24 of the book Automating Software Design [7]. Also, it should be pointed out that a complete development of KBSE educational requirements is beyond the scope of this paper. What is specifically being attempted is the identification and at least the partial definition of the topic areas that make KBSE different from "regular" software engineering. Specific learning objectives and the experimental aspects of KBSE projects also need to be addressed.

After the basis has been established and a set of KBSE education requirements proposed, Section 3 compares the requirements to the core curriculum requirements established by the SEI for software engineering [1]. Section 4 discusses the successful implementation of the KBSE educational requirements at the Air Force Institute of Technology (AFIT). AFIT is a graduate education institution run by the United States Air Force, and enrollment is open to qualified military officers and Department of Defense civilians. Additionally, AFIT is fully accredited by the North Central Association of Colleges and Schools, and the appropriate engineering programs are accredited by the Accreditation Board for Engineering and Technology. The Department of Electrical and Computer Engineering is composed of two divisions, the Electrical Engineering Division and the Computer Science and Engineering Division. The Computer Science and Engineering Division is responsible for the software engineering program as well as several others, and it currently has 17 faculty members. The student population for the division is approximately 50 masters students and three to five doctoral students enrolled each year.

2 Educational Requirements Basis

As stated in the introduction, the program transformation system paradigm is used as the starting point for developing the basis. This seems a natural place to begin considering the impact this paradigm and the KBSA project have had on the KBSE field [7]. A diagram of the initial concept of the program transformation system paradigm is shown in Figure 1. However, over the last ten years, new ideas and concepts have been introduced which require Figure 1 to be modified. Figure 2 shows these modifications, and a discussion of the significant changes is provided below. The remaining parts of Figure 2 are left as originally specified.

1. A box labeled Domain Analysis has been added to reflect the recent work in this field [12]. Domain analysis is defined as "the process of identifying, collecting, organizing, and representing the relevant information in a domain based on the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within the domain" [13]. The major artifact produced by a domain analysis effort is the domain model which is defined as a formal representation of knowledge designed to achieve specific operational goals such as formal specification, automated software generation, reverse engineering, etc., [14]. Thus, a domain model contains knowledge about a specific problem domain in a form useful for some software engineering task. In terms of Figure 2, the domain model defines the vocabulary used to create a formal specification. An integral part of a domain model is its corresponding domain theory. Essentially, this is the set of all statements we wish to make about the domain, formally expressed in a language such as first-order predicate calculus along with a set of inference rules. The domain theory provides a deductive inference capability that allows general domain knowledge to be applied to specific applications within the domain. In general, the domain theory will import already developed theories such as set theory, sequence theory, and mathematical models of the domain. Additionally, the domain theory will contain a set of object and operation definitions along with the laws/rules that pertain to them and allow for an inference capability.

2. A box labeled Software Engineering Knowledge has been added to encompass more than just the decisions and rationale recording. This knowledge is used to assist in the development of the formal specification as well as the mechanical optimization process that applies behavior preserving transformations to the formal specification to transform it into some concrete source program. As an example of software engineering knowledge, consider the concept of using established engineering models as a basis for developing a software architecture for a problem domain [15]. The software architecture then serves as the basis for a corresponding software design generated by an-
Figure 1: Program Transformation System Paradigm

Figure 2: Updated Program Transformation System Paradigm
analyzing and describing a system in the context of the software architecture [15]. Additionally, knowledge about the composition rules used to guide system construction would reside here.

In the following sections, each box in Figure 2 is examined to determine what knowledge-based software engineers need to know to perform these tasks. The educational requirements of each task are then presented in a manner similar to that found in the 1991 SEI Report on Graduate Software Engineering Education [1], i.e., a set of topic areas and corresponding learning objectives (based on Bloom’s taxonomy [16]). Also, a general mapping to graduate courses that currently exist at AFIT is provided [17]. The course titles are provided merely for contextual reference. Even though an attempt is being made to map the requirements to courses, one can expect course content to vary from school to school even when the courses have identical names. Therefore, the educational requirements are more important than how a particular curriculum has been structured to satisfy these requirements.

2.1 Domain Analysis

This is the most difficult box to analyze because of its multi-disciplinary nature, and this problem is compounded by the fact the disciplines in question are not necessarily well defined. The disciplines that must interact in this step are those of domain engineering, knowledge engineering, and software engineering. Rather than make an attempt to precisely define the three disciplines, working definitions are provided, and this paper emphasizes their interactive nature rather than trying to precisely define their differences.

- Domain Engineering — “An encompassing process which includes domain analysis and the subsequent construction of components, methods, and tools that address the problems of system/subsystem development through the application of the domain analysis products” [13]. An important aspect of domain analysis is the capturing of knowledge from domain experts [19].

- Knowledge Engineering — “The acquisition of knowledge from a human expert or other source and its coding in the expert system” [18]. This process also follows a detailed life cycle model that encompasses knowledge definition, acquisition, analysis, extraction, design, coding, verification, etc.

- Software Engineering — In 1976 Boehm proposed the following definition for software engineering, “the practical application of scientific knowledge in the design and construction of computer programs and the associated documentation required to develop, operate, and maintain them” [19]. As pointed out by Shaw [4], many times the term software engineering is used “to refer to life cycle models, routine methodologies, cost estimation techniques, documentation frameworks, configuration management tools, quality assurance techniques, and other techniques for standardizing production activities.” Examples of definitions oriented in the latter way can be found in many of the current software engineering textbooks, such as those by Pressman [20] and Sommerville [21].

Two questions arise at this point: what are the similarities between the three types of engineering, and how does knowledge-based software engineering fit into the picture? The latter question can be answered by examining the definition of KBSE. One definition of KBSE is “the application of AI technology to software engineering” [6]. The premise behind this definition is that software engineering is a knowledge intensive activity. Therefore, it needs to evolve from the more traditional database centered approach to a knowledge-base centered approach that contains reasoning rules for the data and an inference engine that applies the rules [6]. Essentially, the difference between the traditional approach and the KBSE approach is now one of “reasoning about the semantics of a software system versus merely reasoning about the structure of a software system” [6]. Using essentially the same perspective, KBSE has also been defined as the use of knowledge-based technology in the form of intelligent assistants to help software engineers carry out their activities in a manner that does not require any creativity on the part of the assistant [22]. Perhaps the best definition is provided by simply taking a broader perspective on the term knowledge-based programming. Tyugu [23] defines knowledge-based programming as an activity where knowledge bases are used to accumulate useful concepts, programming is performed in terms of a problem domain, the computer is used in the whole problem-solving process beginning with the problem description, and automatic program synthesis is performed. Tyugu’s definition becomes even clearer when the five goals of knowledge-based software engineering are enumerated [6].

1. To formalize the artifacts of software development and the software engineering activities that produce these artifacts.
2. To use knowledge representation technology to record, organize, and retrieve the knowledge behind the design decisions that result in a software system.

3. To produce knowledge-based assistants to synthesize and validate source code from formal specifications.

4. To produce knowledge-based assistants to develop and validate specifications.

5. To produce knowledge-based assistants to manage large software projects.

The question concerning the similarities between domain, knowledge, and software engineering can now be addressed. To produce and effectively utilize a domain model, all three disciplines have to be taken into account. It is unlikely that one engineer is competent in all three areas even though domain and knowledge engineers would seem to have more in common with each other than they do with software engineers. What a software engineer needs to know in the areas of domain and knowledge engineering depends on the perspective taken on the five goals of KBSE. For example, the first two goals can be viewed from the perspective of software engineers who use knowledge-based technology to assist in the task of building application software systems. Alternately, goals three, four, and five can be viewed from the perspective of software engineers who construct the knowledge-based assistants used by other software engineers. In the first perspective, software engineers apply knowledge-based technology to assist with the development of application software. In the second perspective, software engineers develop the knowledge-based technology used to generate application software. In other words, the software engineer delivers the knowledge for generating software, rather than the software itself.

From an educational requirements perspective, two types of knowledge-based software engineers have been defined. In one case the software engineer needs to be able to apply knowledge-based technology and thus needs to achieve the knowledge (recall and recognition), comprehension (translate, interpret, and extrapolate), and application (use of generalizations in specific instances) levels of learning defined by Bloom [16]. Alternately, to be able to construct knowledge-based technology, a software engineer needs to achieve the analysis (determine relationships), synthesis (create new relationships), and evaluation (exercise of learned judgment) levels of learning defined by Bloom [16].

With this dual perspective of technology application vs. technology development in mind, each task shown in Figure 2 is examined to determine the educational requirements in the form of topic areas without necessarily defining specific level of learning objectives. Instead, course mappings are provided with the intent of conveying the type of information that should be covered in each topic area and why this information is important to a knowledge-based software engineer.

For the task of domain analysis, a knowledge-based software engineer needs to know the following things.

1. Topic Area: Domain modeling methods and languages, in particular, the concepts of object-oriented modeling.
   - AFIT Course Mapping: Few if any courses exist that concentrate on formal domain modeling and the development of domain theories. Informal elements of these are contained in the course CSCE 593, Software Analysis and Design I. Formal elements such as mathematics for formal specification and formal specification models are introduced in the course CSCE 594 - Software Analysis and Design II, and covered in detail in the course CSCE 793 - Formal-Based Methods in Software Engineering.

2. Topic Area: Representation and manipulation of knowledge bases.
   - AFIT Course Mapping: CSCE 624 - Knowledge-Based Systems.

3. Topic Area: Mathematics essential to constructing algebraic, model-based, and rule-based system perspectives.

4. Topic Area: Mathematics essential to the development of domain theories such as first-order predicate calculus and inference mechanisms.
2.2 Requirements Analysis and Formal Specification

The overall objective of these two boxes is to use the domain model, its corresponding domain theory, software engineering knowledge, and a statement of the users requirements to construct a formal specification of the behavior of some application system. Most software engineers are already familiar with the traditional methods used for requirements analysis; however, a knowledge-based software engineer must also be familiar with the techniques and languages of formal specification. Essentially, this means the knowledge-based software engineer must now have a different “mindset” than a software engineer who is more versed in traditional specification techniques and languages. The knowledge-based software engineer must be comfortable with specification techniques relying on algebraic, model-based, and rule-based system perspectives. Additionally, the knowledge-based software engineer must be comfortable with the concepts behind various models of computation used to define the semantics of the corresponding specification languages.

1. Topic Area: Formal Specification of Software

2.3 Validation of the Formal Specification

The objective of this box is to validate the formal specification against the stated requirements, and as much as possible, perform automated verification for consistency and whatever other proof obligations are deemed necessary. The proposed method for validation is to use executable, formal specification languages. In this way the end user and knowledge-based software engineer can determine if the specified behavior is consistent with the end user’s expectations. Automated theorem proving techniques are then used to satisfy proof obligations such as: specification consistency, applicability of stated operations, existence of an initial state, and desired properties of the set theoretic data structures used to model the system behavior. To perform such tasks, the knowledge-based software engineer must know:

1. Topic Area: Software validation and verification techniques.
   - AFIT Course Mapping: No single course, covered as appropriate in all six core software engineering courses.
2. Topic Area: Theorem proving and automated theorem proving techniques.

2.4 Software Engineering Knowledge

This box represents an encoding of the knowledge a software engineer uses not only to develop a formal specification but also to transform the effective, but not necessarily efficient, formal specification into an effective and efficient target language implementation. This is a very broad and large body of knowledge, and no further attempt will be made to define this in the limited space of this paper. However, the educational requirements associated with codifying and manipulating aspects of this knowledge can be identified.

1. Topic Area: Representation and manipulation of knowledge bases.
   - AFIT Course Mapping: CSCE 624 – Knowledge-Based Systems.
2. Topic Area: Mathematics essential to constructing algebraic, model-based, and rule-based system perspectives.
3. Topic Area: Mathematics essential to constructing software system designs. Note that this is still one of the major weaknesses of software engineering!
4. Mathematics essential to the development of inference systems such as first-order predicate calculus and automated theorem proving.


2.5 Mechanical Optimization and Tuning

The objective of these two boxes is to develop a software system in the desired target language that is consistent with its formal specification. This is done by applying a sequence of behavior preserving transformations beginning with the formal specification and terminating when an efficient implementation has been obtained. For this area, one must know:

1. Topic Area: Mathematics for formally defining behavior (semantics).


2. Topic Area: Defining and verifying correctness preserving transformations.


3. Topic Area: Theorem proving and automated theorem proving techniques.


4. Topic Area: Data structure and algorithm analysis techniques.


2.6 Maintenance

The objective of this box is to allow for the evolution of a software system via the activities of perfective, adaptive, and corrective maintenance. However, the maintenance process is now performed on the formal specification as opposed to the target program. This means the educational requirements for the requirements analysis, domain analysis, and formal specification boxes apply here as well.

2.7 Summary of the Educational Requirements

Ten major topic areas were identified and at least partially defined as a beginning set of core education requirements for knowledge-based software engineering. In general, the ten topic areas are: Domain modeling, knowledge bases, mathematics for formal specification, mathematics for theorem proving, automated theorem proving, validation and verification techniques, mathematics for software design, mathematics for defining behavior (semantics), definition and verification of correctness preserving transformations, and data structure and algorithm analysis techniques. Most of the ten topic areas can be covered in courses typically offered as electives in graduate level computer science and computer engineering programs. The course numbers and titles specific to the Air Force Institute of Technology were given as points of reference, but in general, courses in artificial intelligence, discrete mathematics, theory of computation, compiler theory, and algorithm analysis fulfill many of the educational requirements of knowledge-based software engineering. However, to make these courses really effective, their orientation in terms of examples and course projects needs to be directed at the concepts associated with KBSE. Therefore, the ideal solution may be to take the 10 topic areas and investigate the feasibility of developing three or four software engineering courses that successfully integrate these topics and provide the necessary KBSE orientation.

To fully explore the difference between KBSE and "regular" software engineering, the ten topics described above need to be compared to the topic areas typically covered in a graduate level software engineering program. This makes it possible to answer questions such as those posed below:

1. Is it reasonable to expect the education of a knowledge-based software engineer to include the educational requirements of a "regular" software engineer?
2. How does KBSE fit into a "big picture" of software engineering?

The next section attempts just such an analysis.

3 Analysis of the Proposed Requirements

The 1991 SEI Report on Graduate Software Engineering Education [1], describes 21 core curriculum topics for software engineering. In general, no software engineering program of reasonable length could be expected to adequately address all 21 topic areas; therefore, most curriculum designers select some subset of these based on their particular objectives and student population [1]. Without selecting a particular subset, it is reasonable to expect that the education of a knowledge-based software engineer must include many of the 21 topic areas. However, only a few of the topic areas clearly address the specialty knowledge required of knowledge-based software engineers. Specifically, these are the topic areas of software quality issues (formal verification), requirements analysis (system modeling), and specification (formal specification). It is possible that some of the other 21 areas do as well, but it is not clear from the descriptions exactly what aspects and in what depth the specialty knowledge is covered. Therefore, it appears as if some aspects of system modeling, formal specification, and formal verification are the only KBSE topic areas covered in any depth in a "regular" software engineering program that follows the SEI guidelines. As was seen in the preceding section, many of the remaining KBSE educational requirements can be covered in courses typically offered as electives in graduate level computer science and computer engineering programs. Thus, one way to establish a KBSE program is as an extension to an existing software engineering program, and this approach has worked well at AFIT.

The AFIT software engineering program has been in existence for several years; and it is documented in the Software Engineering Education Directory [3]. The program consists of six core courses.

1. AMGT 553 — Software Project Management
2. CSCE 593 — Software Analysis and Design I
3. CSCE 594 — Software Analysis and Design II
4. CSCE 595 — Software Generation and Maintenance
5. CSCE 693 — Principles of Embedded Software
6. CSCE 793 — Formal-Based Methods in Software Engineering

Additionally, the students are required to take a course in mathematics, two theory courses, and whatever electives are necessary to complete the program. The courses mentioned in the preceding section and used to support an initial and evolving KBSE program are:

1. CSCE 523 — Artificial Intelligence
2. CSCE 531 — Advanced Mathematics for Computer Scientists
3. CSCE 623 — Artificial Intelligence Systems Design
4. CSCE 624 — Knowledge-Based Systems
5. CSCE 631 — Mathematics of Computer Hardware and Software Design
6. CSCE 663 — Compiler Theory and Implementation
7. CSCE 686 — Advanced Algorithm Design
8. CSCE 786 — Theory of Computation

Even though AFIT has had a software engineering program for several years, a KBSE program is still in its early stages with some problems still to be resolved. The emergence of KBSE as important new technology dictates an increasing need for its presence in the curriculum. Additionally, over the past three years a KBSE research group that currently consists of four faculty members, seven masters students, and one PhD student has been formed. For the research group to effectively accomplish its objectives, the KBSE educational objectives must be defined and accomplished. The KBSE research group is currently investigating the areas of "bridging" the gap between informal and formal software modeling techniques, formalizing the concept of software architectures engineering, and using domain models to compose software systems from reusable components.

Not all AFIT software engineering students pursue the KBSE aspect. In general, all the software engineering students are exposed to elements of KBSE in the courses CSCE 594 — Software Analysis and Design II and CSCE 793 — Formal-Based Methods in Software Engineering. However, only those students working in the KBSE research group typically pursue KBSE oriented electives. Additionally, not all students in courses such as CSCE 624 — Knowledge-Based Systems and CSCE 663 — Compiler Theory and Implementation perform projects pertaining to KBSE.
Typically, just those students doing KBSE research do this. Lastly, the typical masters student has room for only three or four of the eight courses used to support a KBSE program. In most cases the students opt for the courses CSCE 523, CSCE 531, and CSCE 663. Occasionally, students who are well prepared at the undergraduate level also take courses such as CSCE 624 and CSCE 631. One of the biggest strengths of the KBSE program at AFIT has been our ability to obtain a network of Sun SparcStations as well as copies of the Software Refinery™ System marketed by Reasoning Systems. The capabilities provided by Software Refinery have allowed us to incorporate its use across several courses such as:

1. CSCE 623 Artificial Intelligence Systems Design
2. CSCE 624 Knowledge-Based Systems
3. CSCE 663 Compiler Theory and Implementation
4. CSCE 793 Formal-Based Methods in Software Engineering

The software engineering students are now able to use a single environment and language to complement their classroom experiences as well as perform KBSE research. Additionally, the use of Software Refinery has made it much easier to achieve higher level educational objectives such as application, analysis, and synthesis which are part of a student's masters thesis experience.

The biggest weakness in our evolving program has to do with the automated theorem proving educational requirement. The courses CSCE 631 - Mathematics of Computer Hardware and Software Design and CSCE 786 - Theory of Computation both address automated theorem proving. However, most of the masters students do not have room for either one of these courses in their schedules. Additionally, the software engineering laboratory currently lacks a general capability for experimenting with automated theorem proving. The Software Refinery environment is capable of supporting this requirement, but additional work needs to be done on our part to develop a generalized theorem proving capability within the environment.

Overall, AFIT has been able to take significant steps in a short amount of time towards developing an effective program for knowledge-based software engineering. Additionally, the initial success of this effort shows that many of the schools listed in the Software Engineering Education Directory [3] should also be capable of developing similar programs. That is, programs where KBSE is treated as an extension of the existing software engineering curriculum that allows for emphasis in KBSE through appropriate electives and course projects. It also seems appropriate to examine the feasibility of developing a set of three or four integrated courses specifically directed at KBSE rather than relying on elective courses. However, this should not be misinterpreted as a call for a separate curriculum for KBSE, i.e., separate from "regular" software engineering; it should be interpreted as a natural extension.

4 Summary and Conclusions

This paper analyzed an updated version of the program transformation system paradigm with the objective of determining the educational requirements needed to support the field of knowledge-based software engineering. As a result, ten topic areas reflecting the educational requirements of KBSE were identified and defined. These topic areas can now serve as the core requirements for educating knowledge-based software engineers, and they help to better define the field of knowledge-based software engineering. Many of the ten topic areas can be satisfied using courses already available in graduate level computer science and computer engineering programs; however, the best results are obtained when the examples and course projects are specifically directed at KBSE concepts. This paper is only a first step at defining the field of knowledge-based software engineering. More precise definitions of the ten topic areas still need to be developed as well as specific learning objectives. Additionally, the feasibility of developing three or four software engineering courses that successfully integrate the ten topic areas and provide the necessary KBSE experimental element through course projects needs to be investigated.

References


