Abstract

Problem-based and case-based learning are well-known inquiry-based pedagogies. We propose a new inquiry-based approach called systems-questions-solutions (SQS) that combines the two more general approaches. SQS is based on proven educational principles plus general systems theoretic approaches to systems. This approach naturally introduces verification and validation into the educational process.
The SQS Approach to Computational Science Education

D. E. Stevenson

315 McAdams Hall
Department of Computer Science
Clemson University
PO Box 341906
Clemson, SC 29634-1906
steve@cs.clemson.edu

July 19, 2006

1 Central Issues

In the 21st Century, computing capability will be widespread in applications giving rise
to computational science [25, 23]. Computational science is the essential use of computa-
tion to develop knowledge. Computational science is inherently interdisciplinary, leading
to a requirement that practitioners be efficient learners. Knowledge of computation must be
organized in order to make use of knowledge in problem solving. Computational science
applications are often classified as information technology (IT) applications. Workers in IT
in general and computational science in particular must be expert in multiple areas simulta-
neously and will be faced with multiple learning situations as members of interdisciplinary

*Partial support for this work was provided by the National Science Foundation’s National Science,
Technology, Engineering, and Mathematics Education Digital Library program under grant DUE-0435187
and DUE-0525474.
teams in which the members have different backgrounds many of whom are unfamiliar with
the particular problem at hand. This lack of common background means that a significant
effort is required to educate team members. Because these computational teams continually
face unfamiliar problems and changing membership, the team members are constantly in a
learning situation where unfamiliar principles must be rapidly assimilated and applied.

Computational science applications have two significant products: a model in the
problem discipline and a computer system in the computational domain. Computational
science practitioners must be able to understand the model in order to complete the com-
putational assignment. In order to be efficient learners, practitioners must be able to (1)
organize knowledge rapidly, (2) absorb modes of problem domain operations, and (3) convert
problem domain concepts into computational artifacts.

An efficient way to organize knowledge is through traditional epistemological schema
(Figure 1) and newer techniques of mind maps [4] and concept maps [17]. But for virtually
all IT situations, the central concept is a system [12, 13, 27, 29]. This suggests that our
pedagogical principles should be organized around systems, questions about systems, and
answers to those questions in the context of the systems involved just as is done for system
theory in science and engineering. We propose that computational science should be taught
in a manner to emphasize rapid learning of new systems in an approach called systems-
questions-solutions.

2 Fundamental Learning Criteria

Because computational science demands efficient learning, we are faced with two questions:
(1) how do people learn? and (2) what should people be learning?

How People Learn[2] details three fundamental components in learning: (1) deep
factual knowledge (facts) of the subject, (2) deep knowledge of pragmatic issues (schemata),
and (3) metacognitive exercises. The pedagogy that best supports learning is an active
learning environment that engages the learner [2]. The assessment-obsessive K–12 systems
<table>
<thead>
<tr>
<th>Declarative</th>
<th>Procedural</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Focus Questions:</strong></td>
<td>Questions that serve to focus the inquiry about events and/or objects to be studied</td>
</tr>
<tr>
<td>World View</td>
<td>Value Claims</td>
</tr>
<tr>
<td>Philosophy/</td>
<td>Knowledge Claims</td>
</tr>
<tr>
<td>Epistemology</td>
<td></td>
</tr>
<tr>
<td>Theory</td>
<td></td>
</tr>
<tr>
<td>Principles</td>
<td>Transformations</td>
</tr>
<tr>
<td>Constructs</td>
<td>Records</td>
</tr>
<tr>
<td>Concepts</td>
<td></td>
</tr>
</tbody>
</table>

**Events and/or Objects**

Figure 1: General Knowledge Vee

1 focuses on (1) to the detriment of (2) and (3). (3) can be effectively used if the instructor chooses to [24], but for the professional metacognitive exercises are called “coding.” The author has used two popular pedagogies: problem-based learning (PBL) and case-based methods (CBM). PBL is based on evidence that learners learn best when presented with a realistic, preferably fuzzy, focus problem while CBM use collaborative analysis leading to judgments when there is no single, correct answer.

Many computer science course are presented outside a problem-based context that causes students to only solve half the problem. A simple diagnostic test can be given to students at the end of a course: name five problems outside computer science that the material in this course is used to solve. The authors experience is that the students cannot name five such problems. Computational science, and with it computer science, is about systems, questions about those systems, and solutions to problems posed by these systems. While popularized by Simon [22], systems thinking actually has ancient roots. General systems theory [14, 22, 27, 28, 29] is an approach to understanding individual systems through general principles that apply to all systems.

A system is any object with behaviors. There are alternative methods to describe these behaviors such as (1) states and transitions, (2) components and communications or (3) time-invariant functions and relations. A complex system is a system with multiple
identifiable components that are coupled, so that the definition is truly recursive. The state-
transition model of computation is a subtheory of dynamical systems, which accounts for the
large number of characterization of computing using engineering artifacts.

3 SQS Concepts

Modern mathematical treatments of systems theory are organized around problems. Prob-
lems, or more properly questions, can only be posed in the context of a system in which the
question and solution can be interpreted. Answers—meaning solutions—are developed in
the context of a specific system and question. The educational principle we want to develop
is exploration of the triple of (systems, questions, solutions) as SQS. Some examples of this
approach that can be seen in education today are

1. From (system, question), find one or more efficient solutions that satisfy the system
definition and the question.

2. From (systems, solutions), find one or more questions that are answered by the given
solution. Program debugging provides an example.

3. From (questions, solutions) synthesize one or more systems with given capabilities.
   Many social science problems are in this category.

3.1 Knowledge Concepts in SQS

Professional practice in computational science makes use of three types of knowledge: propo-
sitional, implicit, and algorithmic. Propositional and implicit are discussed in [2]. Algorithm-
ic knowledge [8] is central in the computational setting.

Knowledge representation is thought to be organized using schemata [16]. The au-
thor’s experience in teaching computer science indicates that algorithmic knowledge and
schemata are related but not synonymous. In pure computation, algorithms require both
representations and methods of manipulations and it is rare to find a situation in which there is exactly one appropriate representation or computational method.

Justified and coherent time-invariant details of behavior are the standard for knowledge in systems, from whence we can compute state information. Unfortunately, system definition based on state-transition relationships is often all one has, requiring a long, error-prone definition and development cycle. Many problems are stated nondeterministically so many questions have only probabilities for solutions. While testing is important in computer science applications, computational science applications require both verification (testing and proof) and validation (evidence of solving the problem in the users’ domain).

Practitioners must also be aware of Bloom’s taxonomy (Figure 2). System theoretic work involves the higher learning levels of analysis, synthesis, and evaluation. Analysis is the method by which we discover the components and behaviors in systems from state information, a crucial skill because most questions are posed on complex systems. Synthesis is the defining of a system, often with optimality constraints, with a specified behavior from a specified set of components. Much of computer science requires synthesis of optimal systems from both a given list of components and a list of behaviors. Evaluation of alternatives is often the reason the model was formed in the first place.

### 3.2 Implicit Knowledge

Western science is Cartesian, believing that only propositional (explicit) knowledge accounts as knowledge. Yet the most admired scientists are people who are able to solve problems outside the bounds of current knowledge using insights that are not propositional. Expertise is the ability to decompose a problem in novel ways and to apply novel solutions: propositional knowledge is not enough [6].
Category

**knowledge**: recall data or information

**comprehension**: understand the meaning, translation, interpolation, and interpretation of instructions and problems. state a problem in one’s own words.

**application**: use a concept in a new situation or unprompted use of an abstraction. applies what was learned in the classroom into novel situations in the work place.

**analysis**: separates material or concepts into component parts so that its organizational structure may be understood. distinguishes between facts and inferences.

**synthesis**: builds a structure or pattern from diverse elements. put parts together to form a whole, with emphasis on creating a new meaning or structure.

**evaluation**: make judgments about the value of ideas or materials. select the most effective solution. hire the most qualified candidate. explain and justify a new budget.

Figure 2: Bloom’s Taxonomy of Cognitive Processes

### 3.3 Schemata in Problem Solving

How are schemata combined with SQS? Let’s call the heart of SQS problem-solving process meaningful problem-solving (MPS) because the key step is transformation of the semantics of the problem to the semantics of the solution language. This approach combines a Deweyan problem-solving approach and “information mining” from the problem. Such a transfer is generally considered implicit but in CSE, due to its group orientation, it could also be explicit.

Marshall [16] posits that schemata have four parts:

1. Pattern matching and triggers that recognizes context, keywords, and other schemata.

2. Constraints and criteria that provide decision criteria for accepting the pattern as correctly invoking the schema.

3. Planning functions that prepare the environment.

4. Implementation functions that are the “actions” required to fulfill the schema.
3.4 Problem-Solving the SQS Way

The key insight is that computational solutions to problems always include a complex, schemata-mediated step of converting the semantics of the problems into the semantics of the solution in a computational way. This particular step is often missing in normal classroom science and mathematics classes because the problems are divorced from the application.

The SQS problem solving schema is shown below; it is the basis of the pedagogical approach.

1. Linguistic Phase. A problem is received as words and images. The words and images are from the problem posers vocabulary and context.

2. Concept Map Phase. During the concept map phase, the SQS-specific context is established by developing a lexicon, vocabulary, and concept maps [17].

3. Schematic Map Phase. The issues are understood semantically. The system-question-solution classification takes place. Schemata relevant to the missing information are accessed. The initial SQS elements are formulated.

4. Initial mapping phase. The semantics of the problem are mapped to the computational semantics, which are primarily data representation and algorithm “snippets” that will be used later in planning to suggest other coding schemata.

5. Completion. In order to produce a program, the semantics of the program from above are converted to programming language syntax. There are often a large number of judgment calls in program design, made concrete by performance constraints and logical criteria. Design decisions are often based on what can be safely ignored or approximated, again often a judgmentally, not a knowledge based move. Along the way, there are many judgment calls based on algorithm complexity, which is a measure of performance. Very large problems are often placed on networks of computers; some
networks being composed of thousands of processors simultaneously working on the
same problem. These networks are too complex to have a complete understanding of
state.

6. Iterate. A major difference between experts and novices is that experts will iterate or
perhaps attempt to solve the problem in a totally different way. Large, multprocessor
systems generally require several iterations just for the first version.

4 Assessment

From a straight learning perspective, assessment of student learning can be organized using
Bloom’s Taxonomy. While there are many assessment procedures, instruments, protocols,
etc, assessment of active learning requires that the students do things. Assessing proposi-
tional knowledge is relatively straightforward: what facts does the student recall. Implicit
knowledge is much harder to assess and therefore requires innovation.

In [24] the approach is relatively unsophisticated: understand the problem, code the
well-known algorithms, test the code, then explain the process. At the capstone level, the
explanations do not have enough detail to fully assess the knowledge of schemata.

The SQS approach was used in a beginning programming class in the Spring, 2006.
The students were all general engineering students. One approach to assessment was taken
from the K–12 reading procedures known as the cloze procedure\textsuperscript{1} that were developed as
follows. We selected a small C program that included a \texttt{struct} definition and a procedure
to delete an element from a linked list. One standard cloze procedure is to change every fifth
word to a numbered blank. The exercise is that the students must correctly replace the blank
with the correct symbol. As an experiment we presented this exercise to approximately 15
general engineering students and 20 computer science seniors. The results were to small for
statistical evaluation but were interesting: the results tracked with final grades in the senior

\textsuperscript{1}There are many web accessible explanations of cloze.
class, but all the general engineering students did well, regardless of final grade. A more formal assessment is planned.

5 Conclusion

The SQS pedagogy is based on the a well-research body of knowledge in cognitive psychology. The pedagogy is based on two fundamental principles: (1) computational science applications are always systems and (2) there are complex semantic mapping processes underlying computational solutions to systemic questions. This basic approach is used as outlined in [25, 23]

References


