Learning, Teaching and Designing Problem Solving: An Assessment

Wellesley R. ("Rob") Foshay[1]

Andrew Gibbons[2]

Three previous papers (Foshay 1995; Foshay 1997; Foshay and Kirkley 1998) examined the current cognitive research on problem solving, and asked what instructional strategies based on that work might be used to improve the then-standard (behaviorally based) recommendations for teaching of problem solving. In the past decade, there has been continued progress in the understanding of what problem-solving skills are for novices and experts. There has been some advancement in our understanding of how problem-solving is learned, and in how to teach certain kinds of problem-solving skills. The focus of inquiry has shifted somewhat, from descriptive analysis of high-level problem solving to the use of problems in instruction. But there are still many unanswered questions about instructional strategies and their corresponding design techniques. For background, we will first provide a brief sample of some current descriptive thinking on problem solving. Then, we will reframe the inquiry around problems in teaching. Finally, we will comment on current lines of investigation surrounding design of instructional problems. We will conclude with a discussion of key questions which, in our view, are critical to advance our understanding of problem solving instruction and measurement.

What is Problem Solving? A Sample of Current Views

Anderson (Anderson 1980) defines problem solving as "any goal-directed sequence of cognitive operations." Jonassen's recent review (Jonassen 2000) quotes this definition, and adds a number of elaborations. Many of his points are important for our discussion, and will be reviewed in this section.

For much of the 20th century, educators have devoted their attention to trying to define and teach problem solving skills. In the early 1900s, problem solving was viewed as a mechanical, systematic, and often abstract (decontextualized) set of skills, such as those used to solve riddles or mathematical equations. These problems often have correct answers that are based on logical solutions with a single correct answer (convergent reasoning). This view persisted through the 1970's work on general problem solving, and was an influence on the "inquiry-oriented" curricula of the time, such as "new math" and "new science," as well as commercial training courses on problem solving. Curriculum objectives and courses on general problem solving persist in some schools, post-secondary institutions, and industrial training programs.

---

[1] PLATO Learning, Inc. E-mail: rfoshay@plato.com

[2] Utah State University. E-mail:

Cognitive research done in the last 20 years has led to a different model of problem solving. Today we view problem solving as a complex set of cognitive, behavioral, and attitudinal components. In 1983, Mayer defined problem solving as a multiple step process where the problem solver must find relationships between past experiences (schema) and the problem at hand and then act upon a solution. Mayer suggested three characteristics of problem solving:

1. Problem solving is cognitive but is inferred from behavior.
2. Problem solving results in behavior that leads to a solution.
3. Problem solving is a process that involves manipulation of or operations on previous knowledge (Funkhouser and Dennis 1992).

Bruning, Shraw and Ronning (1999) survey current problem solving models, and observe that they generally include these activities:

1. Identifying the problem.
2. Representing the problem.
3. Selecting an appropriate strategy.
4. Implementing the strategy.
5. Evaluating solutions.

Of course, many problems are too complex to be solved with a single iteration of this process. In these cases, the learner breaks the problem down into *intermediate goals* and solves each one in turn, using this process. This switching between smaller, intermediate goals and a larger, final goal is an example of a higher order thinking skill called a *cognitive strategy*. Gagne's (1985) definition of problem solving reflects this principle, and positions problem solving as one kind of higher order thinking skills. He defined problem solving as the "synthesis of other rules and concepts into higher order rules which can be applied to a constrained situation." Scandura (1977) treats cognitive strategies as higher-order rules within his Structural Learning Theory.

Note that there is no assumption that problem solving must involve inductive reasoning. Early on, observations of expert problem-solvers in a variety of fields, (such as Elstein, Shulman et al. 1978), demonstrated that that if the problem solver recognizes that he or she has solved a similar problem before, then all that is needed is to recall how it was solved last time, and do it again. In fact, a common error is to mis-apply such recalled solution algorithms to new problems with superficial similarities to familiar ones. Thus, inductive problem-solving models are likely to be used only to the degree that the given problem is novel to the learner.

Note also the central role of prior knowledge (manifested as a schema) in problem representation. The role of prior knowledge is central to problem representation and formulation of an appropriate problem solving strategy. This is the key reason that problem-solving is context-bound. It also means that far transfer of problem solving skills is problematical, and the conditions under which it occurs are not well understood.

Since far transfer is a central goal of general education at all levels, we presume that a range of experiences in various contexts is needed for far transfer. However, it is not yet clear how to determine the range of contextualized problem solving experiences needed to achieve the purposes of education. Nor is the role of abstract (decontextualized) problem representation in supporting far transfer well understood. In professions education and training contexts, however, the implication is clear: the context-bound nature of problem solving requires that training in problem solving occur in every context in which it is possible to predict the need.

Problem solving also includes attitudinal as well as cognitive components. To solve problems, learners have to want to do so, and they have to believe they can. Motivation and attitudinal aspects such as effort, confidence, anxiety and stress, fatigue, persistence and knowledge about self are important to the problem solving process (Jonassen 2000). These factors are strongly influenced by the performance context, thus reinforcing the important role of contextualization in problem solving, and the importance of collaborative learning in the teaching of problem-solving.

Jonassen's review contrasts this understanding of problem-solving with the general problem solving methods developed in the 1960's. He concurs with the argument that problem solving is not a general, context-free skill. Instead, it is clear that problem-solving is a context-bound skill, in which experts synthesize their rich declarative knowledge to generate a dynamically changing, personal, working mental model of the system (problem space) suitable for solving a particular class of problems; they draw on an extensive reservoir of past experience solving analogous problems in the same domain, and they can set intermediate goals and switch between their corresponding sub-problems according to strategies appropriate for problems of a given type. Novices don't know as much, and therefore have mental models which are less complete, poorly structured and even misleading. Therefore, novices can't simply imitate what experts do, because it doesn't make sense to them. The instructional challenge is to help novices develop initial mental models which are not misleading, and then to help them enrich their mental models to an expert's level. Simply recalling (or retrieving) well-structured solution algorithms is not enough.

Cognitive scientists such as J.R. Anderson have long distinguished between *declarative* and *procedural* knowledge (Anderson 1995). The former includes knowledge structures containing facts ("knowing what"), concepts ("knowing that") and principles ("knowing why"). The latter includes problem-solving ("knowing how"). Newell and Simon (1972) further described a continuum of cognitive problem types differentiated by degree of structure inherent in the problem. At one extreme are invariant, step-by-step procedures which can be recalled and applied to *well-structured* problems such as balancing a checkbook. At the other extreme are *loosely-structured* problems, which have no single well-defined and agreed-upon solution; there may not be a fully satisfactory solution at all. An example of a loosely-structured problem might be a design problem such as composing a painting. To solve this problem, the artist might need to define his or her intent (e.g., by defining the desired effect or the "message" of the work). The artist would probably also use materials and a style with which he or she is familiar, and perhaps recognize the client's wishes (if it is a commission). From all this would come a definition of the intended painting (the proposed solution). Then, the artist would break

the problem in to a series of more structured intermediate tasks intended to reach the ultimate goal of the completed painting the artist had in mind.

The distinction between declarative and procedural knowledge seems to fit well with what is understood about the importance of prior knowledge and mental modeling. It suggests that declarative knowledge, however it is acquired, must be synthesized into mental models (or schemata) for a specific intended purpose of problem solving. This has suggested to instructional designers (most notably van Merrienboer 1997) that declarative and procedural knowledge are best addressed with different instructional strategies, presumably in separate (but closely linked) events. By contrast, accounts of constructivist learning environments rarely describe learning events explicitly for the purpose of acquisition of declarative knowledge. Thus, they appear to prefer treating it as an incidental part of problem solving activity.

Newell and Simon's continuum of structure in problem solving also has endured, and is one basis of Jonassen's recent typology of problems. Problem-solving typically encountered in the world of work includes examples from throughout the continuum of cognitive problem solving types. All jobs have many well-structured problems; most jobs have moderately-structured problems; many jobs have at least a few loosely-structured problems. Particularly with the move to self-managing teams and flat management structures, the trend clearly is to include moderately- and even loosely-structured problems in jobs in almost any work environment. Thus, we can observe that the only part of problem solving which is ill-structured is the goal definition and the strategy; the actual performance of step-by-step tasks is well-structured. The implications for task decomposition in analysis, and for teaching of problem solving, are important.

For design purposes, another point about the continuum of structure is useful. If we think of problem solving in terms of input, process and output, then:

- *Well structured* problems typically have well-defined inputs, processes and outputs.
- *Moderately structured* problems typically have well-defined inputs, a range of alternative processes, and a range of outputs. The processes and outputs typically include some that are clearly "wrong" and a number which are potentially "right" solutions.
- *Loosely structured* problems typically have little definition of inputs, processes and outputs.

This distinction is especially important in technology-based teaching of problem solving, because each point in the continuum of structure carries with it different implications for technologies which are feasible for construction of appropriate learning environments, such as instructional games, simulations and projects. For example, this distinction was key to design of the PLATO Problem Solving Activity architecture.

We previously noted that meaningful job tasks often include a mixture of problems at various points on the problem structure continuum. For example, a team designing an advertising campaign would deal with the loosely-structured task of deciding the main

message of the campaign. Once that is done, then there are many moderately-structured tasks such as selecting the media mix and designing the ads. Producing the ad involves many well-structured procedures for using the tools and stylistic "tricks" of the medium. The implications for task analysis are significant: any analysis of real-world problem solving is likely to include a variety of problem solving tasks at varying levels of structure.

Solving moderately- and loosely-structured problems is inherently a far transfer task, since the performer works from prior knowledge and experience to the new problem. Even for workers who solve mostly well-structured problems (such as operating or troubleshooting common faults in any technological device), it is important to generalize to previously unencountered problems, and to adapt quickly to changes (e.g., new equipment models or software releases) without formal retraining. This is another reason why moderately- and loosely-structured problem solving is becoming an increasing component of even apparently well-structured jobs, and why the need for training in moderately-structured and loosely structured problem solving is growing in the work place as well as in schools. Significantly, Jonassen questions the common research assumption that well-structured problem solving skills chain together for transfer to moderate- to ill-structured problem solving. Instead, he argues that problems require different problem solving skills depending on their degree of structure. Thus it does not follow that transfer from well- to moderate or ill-structured problem solving is to be expected: they must be taught separately. Note that this view is consistent with van Merrienboer's 4C/ID model.

We noted above that recall of solution algorithms for previously encountered problems is a common strategy. Jonassen further notes that degree of apparent structure in a problem depends on its familiarity to the learner as well as its inherent qualities. The first time a learner encounters a new problem, it appears to be ill-structured, even if objectively it is not. When the learner has learned to solve the problem and remembers the solution algorithm, it no longer appears ill-structured. Each time the learner recalls and applies the solution algorithm, it becomes more completely elaborated, and thus more structured for that learner, regardless of the problem's objective degree of structure. Thus, degree of structure in a problem depends on both its inherent structure (which can be determined by analysis), and the learner's subjective experience with the problem. This suggests that a novice problem solver may tend to view as ill-structured, problems which for an expert are moderate- to well-structured. The implications for planning and assessing a progression of problem solving instruction are significant.

Problem solving experience also affects mental model development and manipulation. Underlying problem solving is a mental model (schema) of the performer's view of the system being manipulated. A mental model is formed by experience in problem solving. John Anderson (*cited above)* believes the mental model is synthesis of declarative knowledge into a structure which is optimized for solving a certain class of problems. Solving problems typically requires performers to dynamically restructure and "run" their mental models of the system in order to predict the effect of a proposed action on the system or to explain an observed system behavior. The complexity and structure of the user's mental model depend on what problems the performer has experience with solving. For example, the mental model needed to make a domestic telephone call

includes only the visible parts of the phone, a little about the syntax of phone numbers and dialing prefixes, and almost nothing about the telephone network beyond the wall jack. At the other extreme, the strategic problems a CEO must solve are fairly loosely structured, so the mental model the CEO needs of the company and its environment is quite elaborate and dynamic. While mental models are elaborated and are manipulated with growing skill commensurate with experience in problem solving, we do not yet understand whether explicit learning of a mental model and its manipulation has a role, or whether this kind of learning can occur only incidentally within the context of problem solving experience.

Troubleshooting is a special case worthy of discussion. A common presumption is that troubleshooting is always a well-structured problem for which the solution is recalled or retrieved from a job aid. But cognitive researchers (Tenney and Kurland 1988; Lesgold, Lajoie et al. 1992) have found that expert troubleshooters, especially when faced with a fault new to them, treat it as a moderately structured problem. They generate the specific troubleshooting algorithm for the problem "on the fly" using their mental model of the system, experience with analogous cases, and ability to use the strategies for troubleshooting circuits of that type. Experts may even prefer this strategy to "looking it up," and they often criticize published job aids as inefficient because they often don't rely on expert-level knowledge of factors such as component reliability, history of the unit's use, previous repairs, etc. By contrast, this kind of information is neither known, nor can its significance be understood, by a novice troubleshooter. Furthermore, the moderately structured strategy help the expert troubleshooter adapt to new equipment models and changes in the technology. The novice using well-structured strategies cannot adapt so easily and more often requires re-training.

Jonassen's recent review recognizes troubleshooting as one type of problem. He goes on to further distinguishes 11 additional problem types and differentiates them by structural characteristics, including the learning activity, types of inputs and outputs, success criteria, type of context, degree of structure, and degree of abstraction. Jonassen's analysis does not include issues of instruction, but potentially, this typology may lead to differentiation of instructional strategy by problem type.

How Should We Teach Problem Solving?

The significant progress in our understanding of what problem solving is, reviewed above, has enabled a reconsideration of how problem solving is learned, and how to teach it. In comparison, however, research on learning and teaching of problem solving is much less advanced.

Working from the understanding of problem solving summarized in the previous section, and particularly from Jonassen's review (Jonassen 1997), Foshay and Kirkley's review (Foshay and Kirkley 1998) proposed these 15 instructional strategy principles for teaching problem solving:

- For any "real-world" job or work skill, identify both the declarative and procedural knowledge components. Give each appropriate instructional emphasis.

- First introduce a problem solving context, then either alternate between teaching declarative and procedural knowledge, or integrate the two.
- When teaching declarative knowledge, emphasize mental models appropriate to the problem solving to come, by explaining knowledge structures and asking learners to predict what will happen or explain why something happened.
- Emphasize moderately- and ill-structured problem solving when far transfer is a goal of instruction.
- Teach problem solving skills in the context in which they will be used. Use *authentic* problems in explanations, practice and assessments, with scenario-based simulations, games and projects. Do not teach problem solving as an independent, abstract, decontextualized skill.
- Use direct (deductive) teaching strategies for declarative knowledge and well structured problem solving.
- Use inductive teaching strategies to encourage synthesis of mental models and for moderately and ill-structured problem solving.
- Within a problem exercise, help the learners understand (or define) the goal, then help them to break it down into intermediate goals.
- Use the errors learners make in problem solving as evidence of misconceptions, not just carelessness or random guessing. If possible, determine the probable misconception and correct it.
- Ask questions and make suggestions about strategy to encourage learners to reflect on the problem solving strategies they use. Do this either before or after the learner takes action. (This is sometimes called *cognitive coaching).*
- Give practice of similar problem solving strategies across multiple contexts to encourage generalization.
- Ask questions which encourage the learner to encourage the learner to grasp the generalizable part of the skill, across many similar problems in different contexts.
- Use contexts, problems and teaching styles which will build interest, motivation, confidence, persistence and knowledge about self, and reduce anxiety.
- Plan a series of lessons which grow in sophistication from novice-level to expert-level understanding of the knowledge structures used.
- When teaching well-structured problem solving, allow learners to retrieve the algorithm (e.g., from a reference card). If the procedure is frequently used, encourage memorization of the procedure and practice until it is automatic.
- When teaching moderately-structured problem solving, encourage the learners to use their declarative (context) knowledge to invent a strategy which suits the context and the problem. Allow many "right" strategies to reach the solution, and compare them for efficiency and effectiveness.

- When teaching ill-structured problem solving, encourage the learners to use their declarative (context) knowledge to define the goal (properties of an acceptable solution), then invent a solution. Allow many "right" strategies and solutions, and compare them for efficiency and effectiveness.

Van Merrienboer's (van Merrienboer 1997) 4C/ID model echoes the distinction between prerequisite (declarative) knowledge, algorithmic methods (well-structured procedural knowledge), supportive knowledge (mental modeling) and heuristic methods (procedural knowledge for moderate- to ill-structured problem solving). In his view, each knowledge type has its own selection of instructional strategy alternatives. For algorithmic methods, he suggests part-task practice strategies (gradually chaining together the full algorithm). For prerequisite (declarative) knowledge, he suggests just-in-time presentation strategies. For heuristic methods, he suggests whole-task practice, using instructional simulations and related techniques, by default in an inductive-expository strategy. For supportive knowledge, he suggests methods which will promote elaboration and understanding, including case studies with a guided discovery strategy, and modeling of fully-worked examples with an inductive-expository strategy. For each of the four general instructional strategies, he adds prescriptive principles which appear to be generally consistent with the Foshay and Kirkley ones.

Underlying both the Foshay and Kirkley principles and van Merrienboer's model are a number of key issues for definition of instructional strategies for teaching problem solving. Among the most important are:

- How should knowledge structures be represented to the learner?
- How should the teaching problems be selected?
- How should the teaching problems be sequenced?
- How should we do cognitive coaching?
- How should we teach mental modeling and cognitive strategies?
- How can we measure proficiency in problem solving?
- Does direct (tutorial) instruction have a place in teaching problem solving?

We will discuss each of these questions in turn. In discussing them, we will review current work on problem-solving teaching by a representative selection of current cognitive and constructivist theories.

1. How should knowledge be represented?

Traditional (behaviorally-based) task analysis methods for analyzing well-structured problems are well developed and understood by most designers trained in job task analysis (for example, see (Mager 1982)). There have been a few attempts to develop and document practical and generalizable cognitive task analysis (CTA) methods for problem solving, such as the PARI technique (Hall, Gott et al. 1995). However, like most task/content analysis techniques, CTA methods generally suffer from subjectivity: no two analysts would produce a substantially similar analysis if presented with the same

problem. Furthermore, Gibbons, Nelson and Richards (Gibbons, Nelson et al. 1999) argue that the representations are often difficult to interpret, do not directly suggest designs, and are incomprehensible to the learner. Thus, they are inefficient at best, and ineffective at worst.

Rule-based vs. Case-based analysis. The influence of work in expert systems and intelligent tutoring led to an early emphasis on analyzing problem spaces into structures of logical decision rules (e.g., "IF you receive a busy signal, THEN hang up and redial.") Schank and his associates have argued that this kind of knowledge structure has many inherent limitations which can be overcome by using case-based analysis (Schank, Kass et al. 1994). Instead, they argue for use of cases, which are stories of actual problems and how they were solved (a ready source is "war stories" experts exchange, which they often find valuable because they illustrate some key heuristic principle or insight into the relevant schema). Case-based analysis produces a rule set which is more "superficial" than rule-based analysis, and seeks to model descriptively only the behavior of the system in a particular circumstance (e.g., "When Ann used her office phone, this is how she made a long-distance call…"). Schank argues that this approach has two advantages over the more common rule-based CTA:

- It is more applicable to "real" problems, which are typically large, and "messy" (have irrational or inconsistent elements).
- It is a better model of how experts really think.

The second claimed advantage is particularly of interest here, since it goes to the heart of how one should teach problem solving. Schank is among those who believe experts solve novel problems by reasoning analogically from previous experience with similar problems. Thus, to teach problem solving, Schank believes one should directly teach "case studies," with enough attention to the underlying problem structure so the case and its solution are stored (indexed) in a way which will facilitate future recognition of the relevance of the case to others like it.

Scandura's Flex Form analysis (Scandura 2001), used in a wide variety of commercial applications as well as structural learning theory, is an excellent example of the power of rule-based analysis. Rule-based analysis is familiar to instructional designers using traditional task analysis of well-structured procedures. Using the principles of SLT, Scandura has shown how it is possible to derive higher-order rules in a bottom-up fashion. Working top-down, higher-order rules are of generative use in ill-structured problem solving. Case-based analysis is treated as simply a special subset of the general analysis system. This approach is at the basis of breakthroughs in structured design of large-scale software systems, and may well be of use in design of instructional simulations.

Merrill's Process, Entity and Activity Network (PEA-Net) (Merrill and 1 1993) is another rule-based knowledge representation technique which can be applied to problem solving. His Instructional Transaction Theory calls for mapping of an Elaborated Frame Network involving three types of elaboration: components (events, parts and steps in a process), abstractions (which define classes and subclasses of components), and associations (which are the meaningful links between processes, entities and activities). The resulting

network thus incorporates both declarative and procedural knowledge. The network may be close to a mental model representation, and can be used to generate instructional transactions of many types. Presumably, the growth of the mental model from novice to expert level would be shown by adding components to the PEA-Net.

The claimed advantages of case-based analysis may make it the preferred approach for training on moderately- and loosely-structured problems (Ohlsson 1995). However, there remains the key issue of how to perform the analysis and how to represent the result for design purposes and to the learner. Schank apparently goes no further than an intuitive gathering of "war stories" and construction of idiosyncratic prototypical case problems. A systematic analysis technique (which presumably would be "bottom up") is not described. A review of current methods (Gibbons, Nelson et al. 1999) found significant shortcomings with all current CTA methods.

In this context, Gibbons, Nelson and Richards (Gibbons, Nelson et al. 1999) have proposed a Model-Centered Analysis Process (MCAP) as a more systematic and generalizable solution (discussed more fully in the next section). MCAP provides four views of the problem: a problem structure view, an environment view, a systems view, and an expert performance view. MCAP is essentially problem-based (or case-based) analysis. While the result of an SLT analysis is a hierarchy of progressively more abstract procedural rules, and a PEA-Net is multi-node map, the result of an MCAP analysis is a hierarchical instantiation of the semantic string:

In <environment> one or more <actor> executes <performance> using <tool> affecting <system process> to produce <artifact> having <qualities>.

Thus, an MCAP analysis appears to capture procedural rules, but in a case-based, context-specific way. Given our current understanding of the importance of context in problem solving, this is may be an important advantage.

While PEA-Net, SLT analysis and MCAP are promising approaches for cognitive task analysis and design, a number of questions remain over their utility to learners. For example, it is not clear whether it is useful to dialog with the learner about the structure of the problem space or schema. It also is not clear if there is any benefit in presenting a structural map of the problem space or schema to the learner; many constructivist treatments have no such activity, or leave the structure's construction entirely to the learners (often as a collaborative learning activity). Finally, it is still not clear how to derive from the analysis a view of the problem space or schema which will advance the learner's understanding. If it turns out that the best method of representing the problem space or schema to the learner is specific to each context and each learner, it will be a major barrier to generalizability of problem solving teaching methods.

2. How should the teaching problems be selected and sequenced?

Given the severe limits on far transfer, it is obvious that learners need to work with a wide selection of problems. Given what we know about novice and expert competency in problem solving, it seems clear that novices should work with a progressively more complex and realistic sequence of problems. In the previous section, we referred to Jonassen's and van Merrienboer's skepticism over the current theoretical assumption that

novices can progress from well- to ill-structured problems; instead, it appears that different skill sets are involved and therefore it does not follow that learning well-structured problem solving is prerequisite to learning moderate- to ill-structured problem solving. However, we do not have a design method for specifying the needed set of problems, or for sequencing them.

Gibbons (Gibbons, Nelson et al. 1999) proposes that MCAP analysis provides a way to remove the complexity from real-world problems while preserving their structural properties, the authentic essence of problems. This design process, called "denaturing" (discussed further in the next section), calls for creation of novice-level problems which are structurally complete but artificially simplified through elimination of various elaborations and embedded well-structured procedures. These problems are intentionally unrealistic in a specified way; they are not merely fractional subsets of real-world problems.

However, there is the additional complication that degree of structure in a problem depends on what the learner already knows. Therefore, to effectively prescribe a problem at the appropriate level of denaturing for a particular learner, we will need a way to predict how the learner will perceive the structure of a given problem. Essentially, this is an issue of familiarity: the more similar a problem is to ones the learner has already completed successfully, the more structured the problem will appear. Therefore, Gibbons suggests that by using problems of structural similarity and comparable levels of denaturing, it should be possible to plan a sequence of problems suited to the experience of each learner – but only if we know the history of the learner with past similar problems, or if we can measure the learner's current state of mastery. In a computer-based simulation/game learning environment, this is quite feasible. In other environments, it is more difficult.

Note also that this principle of personalized prescription for problem solving may be an operationalization of Vygotsky's "zone of proximal development" (Vygotsky 1978). This interpretation places Vygotsky's concept in the context of contextualized problem solving, and thus implies that the zone of proximal development is an issue of individual experience with problems within a given domain (thus making it state-dependent). It is not clear if there are also general developmental issues (trait dependency) underlying the zone's definition for a given learner.

3. How should we do cognitive coaching?

Feedback to the learner in problem solving instruction can be at the tactical level of each well-structured step, or at the strategic level concerned with higher-order rules and problem space structures. Most examples of problem solving teaching, such as intelligent tutoring systems, seem to do both. There is some evidence (Lesgold, Lajoie et al. 1992) that simulations which include a "cognitive coach" to talk to the learner about strategy do improve learning efficiency considerably, when compared to conventional "dumb" simulations and lab exercises which practice defined procedures.

In intelligent tutoring jargon, a "coach" is the part of the system which provides feedback comments and suggestions, hints, helps and other supporting information to the learner,

beyond the realistic responses of the simulation as the learner manipulates it (Orey and Nelson 1993). The intended impression is that of a "master problem solver's" dialog with the learner about the problem solving which the learner is doing. In classroom settings, researchers are developing collaborative learning techniques to encourage teachers and small groups of learners to converse at this level, but it has proven to be a difficult skill to impart to teachers and learners (Blumenfeld, Fishman et al. 2000). As with cognitive task analysis, design practitioners need much more experience with coaching designs over a much wider range of content and training contexts to develop prescriptions for what techniques work best under what circumstances.

A related question concerns how useful it is to show the learner the procedure structure in advance, as opposed to having the learner discover it as the problem-solving exercise proceeds. A generation ago, this was a key issue in the debate over "discovery learning." Now, this is one issue in the quasi-ideological debate between constructivists and cognitivists, yet there is very little research specifically on this issue. Van Merrienboer suggests that the preferred method for most learners is an inductive-expository approach, in which case problems are presented first and presentation of structure abstractions and discussion of similarities among case problems occurs afterward: in effect, a "bottom-up" approach to cognitive coaching.

Van Merrienboer's preferred inductive-expository sequence would have the effect of progressing the dialog with the learner from concrete-operational issues with the context-specific problem, to more abstract-symbolic and context-independent representations of the problem. Presumably, the learner then would be better prepared to attack structurally similar problems in a new context, so the dialog would then revert to the concrete-operational, but in the new context with the new problem.

Use of Gibbons' "denatured" problems (discussed further below) presents a third alternative: the "denaturing" should have the effect of making the high-level problem structure more visible and understandable to novice problem solvers. This, in turn, could be expected to accelerate mastery of progressively more realistic and complex forms of similar problems.

4. How should we teach mental modeling?

In the previous section, we argued that mental modeling is a key skill in problem solving, yet it is often not addressed explicitly in problem solving instruction. This raises the question of whether mental modeling should be taught as a prerequisite and scaffolded as a co-requisite skill for problem solving. A variety of methods and technologies for this purpose have been developed (beginning, perhaps, with early experimentation with *Hypercard* and continuing today with tools such as *Inspiration*). However, many experimental treatments in problem solving studies neither elicit nor scaffold mental modeling, in effect treating it as incidental. So, while it is likely that explicitly teaching and scaffolding mental modeling is an important part of teaching problem-solving, research clarifying its position in complete problem-solving solutions is yet to be done.

The challenge here is to get learners to think structurally, rather than just procedurally, about the problems they are solving. A recent review (Atkinson, Derry et al. 2000) notes

how difficult it is to get learners to engage in this kind of thought. Most learners do not spontaneously do so, and scaffolding of the dialogs within the context of solving a problem has met with only limited success.

Atkinson, et. al. suggest that a powerful solution is pairing cognitive modeling of worked examples with closely related problem solving activities using scaffolded dialog. In particular, they suggest that:

- Worked examples and matched problems be closely interspersed.
- Multiple examples be presented for each problem type.
- Prompting and other surface features of the problems be incorporated which will emphasize the underlying structure of the problem.
- Presentation of the example problem, solution actions, and discussion of the problem structure and solution strategies be closely integrated, using multiple modalities. This is easily done in computer-based multimedia (CBT) formats (an example of this recommendation is the PLATO® *Reading Strategies* curriculum, which makes extensive use of think-aloud protocol-style worked examples of reading comprehension strategies).
- The explanation make clear each component subgoal and underlying declarative knowledge.
- A useful interaction is to provide incomplete explanations, and ask learners to supply the missing pieces.
- The learning environment should include incentives to explain examples.
- Worked example instruction be accompanied by direct instruction and prompting on how to self-explain examples.

In the context of worked examples, it may be useful to use structural modeling tools (such as Merrill's tools for building PEA-Nets, Scandura's FlexForm or *Inspiration,* or symbolic representations specific to the expert's profession) as a prompt to help the learner see the problem structure. However, the recommendations above lead us to expect that learner use of such tools for mental modeling will require prerequisite direct instruction. We believe incidental use of these tools for modeling is not likely to succeed, even with scaffolding.

We would also predict that use of a progression of "denatured" problems, as discussed above, would be a useful means of combining use of worked examples and cognitive coaching to help learners focus on the underlying problem structures and cognitive strategies.

5. How can we measure problem solving skill?

It is clear that measuring problem solving skill requires a way to judge both whether the learner found a "correct" solution, but also how efficiently it was found, using what kind of knowledge structure and cognitive strategy. Doing this with a high degree of validity

and reliability is a current topic of psychometric research(Goldman, Zech et al. 1999). Designers need prescriptive principles for design of problem solving measures and processes. Jonassen's typology (cited above) suggests a number of measurement strategies which could reveal various characteristics of the learner's knowledge structure.

However, guidelines for design of high-stakes tests for problem-solving are only now being developed. Guidelines for design of rubrics for judging school classroom-based problem-solving activities are widely published, but the techniques are still relatively simplistic applications of ethnographic techniques. They are labor-intensive, require considerable instructor training, and often are not grounded in a thorough cognitive task analysis. This leaves the design practitioner with very few standard measurement procedures for problem solving skill, other than reporting the learner's progress through a simulation, game or other classroom activity, and some observations from a "think aloud" dialog or other statement of rationale. As Goldman et al report, interpreting these traces is subjective and difficult. In turn, this makes it very difficult to state in generalizable terms just what problem-solving skills the learner has mastered.

We view the measurement problem as complementary to the analysis problem, discussed above. In our view, measurement has been an intractable issue precisely because prescriptively useful, efficient and reproducible cognitive task analysis techniques have not yet evolved. Until we understand better how to analyze problem solving behavior in a way which is efficient and easily interpreted, we will be unable to deal efficiently and effectively with the measurement issue. In this light, we believe PEA-Net, MCAP and Structural Learning analysis merit examination.

6. When should we include direct instruction on declarative knowledge and mental modeling?

The discussion of problem solving explained the important role of declarative knowledge. However, many constructivist solutions leave teaching of declarative knowledge implicit in their problem-based learning activities, and categorically oppose use of direct instruction. At best, there may be some support of the learner's exploration of information the problem domain.

By contrast, van Merrienboer's 4C/ID model calls for explicit provision of declarative knowledge on a just-in-time basis. He argues that direct instruction is both effective and efficient for this purpose, though he recognizes that an inquiry approach can work if efficiency is not a consideration. Depending on the complexity and difficulty of the declarative knowledge for the learner, this could be done through a combination of just-in-time reference and tutorial methods. In computer-based work environments, this is precisely the intent of the Electronic Performance Support System (EPSS). In training and education environments, EPSS are virtually unknown. However, the strategic trend in PLATO curricula (Foshay 1998) is to modularize heavily both tutorial and informational components, with the intent that they be used in a just-in-time basis during problem-solving. This strategy is described in detail for PLATO as the "Problem-Based Instruction" strategy (Foshay and Kirkley 1998). Consistent with van Merrienboer, this model advocates using tutorials within a context established by presentation of problems. It also requires that the tutorials and information presentations use examples and a frame

of reference consistent with real-world application of the declarative knowledge, so that the context of problem-based application is maintained.

There is clearly a need for additional research, from a "neutral" theoretical perspective, on the question of when and how to teach declarative knowledge in the context of problem solving. Specifically, tutorial strategies need to be evolved for this purpose, and we need to investigate issues of both efficiency and effectiveness of these strategies in the context of an overall instructional design for teaching problem solving.

How Should We Design Problem Solving Instruction?

At the start of this paper we posed the question "What is problem solving?" The discussion of teaching problem solving in the previous section makes it clear that learners must solve problems, but these problems must have certain characteristics which enhance their instructional utility. Thus, as we begin to discuss the design of problems for instructional use, the appropriate questions become "What is a problem?" and "How can problems be used instructionally?"

Building on Jonassen's definition of problem solving as "any goal-directed sequence of cognitive operations," we suggest that a problem can be defined as a requirement to reach a goal by carrying out a sequence of cognitive operations in which at least one of the following is unknown:

- The beginning state
- One or more of the required cognitive operations
- A satisficing sequence of operations
- The goal itself (end state)

The sense of problem solving has always been that the problem solver must synthesize one or more of these in order to reach a solution. If the learner already knows the beginning state, the operations, their sequence, and the goal, then the behavior required can be performed from implicit (automatized) or explicit (retrieved or reconstructed) memory.

It is, therefore, a non-trivial step to go from asking, "How do we teach problem solving?" to "How should we design instruction that uses problem solving as its vehicle?" In the former, problem solving is the target behavior; in the latter, problem solving is a means of instruction for some defined target behavior and some degree of problem solving ability is assumed as an entry skill. The target behavior often involves problem solving, but not necessarily: Duffy and Cunningham (Duffy and Cunningham 1996) identify "five strategies for using problems [instructionally] that reflect different assumptions about either what is to be learned or how learning occurs." The five strategies may use problems as:

- a guide or stimulus for independent thinking
- an integrator or test
- an example
- a vehicle for process, and

- a stimulus for authentic activity.

Basing instruction on a problem is one characteristic of a new instructional paradigm reviewed by Gibbons and Fairweather (Gibbons and Fairweather 2000), and by Hannafin and his associates (Hannafin 1992);(Hannafin, Hannafin et al. 1997).[3] The assumption of virtually all problem-based instructional forms is that the problem will be used as the central organizing element of instruction. Examples of a wide range of problem-based instructional approaches can be found in the *Handbook of Research for Educational Communications and Technology*(Jonassen 1996). Section III of that collection of papers, edited by Robert Tennyson, shows that the trend toward the new paradigm has multiple sources, including research in cognition and learning, intelligent tutoring systems, and constructivist philosophy. Articles by Grabinger (Grabinger 1996), Wilson and Cole (Wilson and Cole 1996) and Shute (Shute and Psotka 1996) describe a wide range of laboratory and public systems that exemplify this paradigm.

Most "standard" instructional designs structure and organize instructional products around an inner framework of message and strategy constructs for the purpose of direct instruction[4]. By contrast, the shared characteristics of problem-based products in general are:

- Presentation of models of knowledge and behavior
- Requirement that the learner form and/or solve problems
- Augmentation of problem solving with tailored support

Problem-based instruction is centered on problems and models of environments, cause-effect systems, and expert performance (Gibbons 1998). Learners are normally given a problem to solve within a problem-solving environment that provides problem manipulables and tools, resources, and expert instructional augmentations. Problem manipulables may be a realistic model of a work environment or merely a neutral work surface that provides access to the tools and resources and a workspace on which to arrange and rearrange problem and solution elements until they fall into place.

Designing instruction using the problem as the central organizing construct poses several questions to the designer:

- What is instruction like when problems are used as the basis for event definition?
- How can a person be instructed and solve problems at the same time?

---

[3] *Paradigm* as it is used here differs from Reigeluth's usage in *Instructional-Design Theories and Models, Volume II Reigeluth, C. M. (1999). Instructional-design theories and models: A new paradigm of instructional theory. Mahwah, NJ, Lawrence Erlbaum Associates.*

. Reigeluth refers to a new paradigm of *research* and *theorizing*. The use of the term here refers to a new style of *instruction* that makes unique assumptions about the structure and conduct of the instructional event, and therefore about the architecture and design of the product.

[4] Our use of the term "direct instruction" is intended to be generic, and not specific to Engelmann's Direct Instruction.

- How does problem-based instruction interact with common instructional design conceptions?
- How are problems generated, selected, and sequenced?
- How are problem environments designed?
- What are the implications for tools?

We discuss these issues separately in the sections that follow.

1. What is instruction like when problems are used as the basis for event definition?

The surface forms of problem-based instruction vary widely. Sometimes the variations from more common forms of instruction are so great that familiar elements of instruction—such as set presentations, lesson boundaries, and session boundaries—disappear. This can be disquieting to designers used to using those components as orientation points or indicators of good design, as well as instructors using the instructional materials. The examples of problem-based instruction described below illustrate this diversity:

Problem-Based Learning (Barrows 1988) – Problems are presented by a live tutor to a group of learners. Problem resource books contain the problem data. Group members are assigned responsibilities for keeping public charts that help the group record progress toward a solution and plan its solution activities. A range of information resources is made available to the group. Activity cycles between group work and individual information-gathering assignments.

Anchored Instruction (Barrows 1988) – Problem-posing stories that contain all relevant problem data are presented using a medium that permits later random access to the elements of that data. Problem solving is performed by groups of learners organized by different structures under different initiative-taking conditions, but usually under the leadership of a teacher. Direct instruction, which can take different forms, is delivered on selected subjects as determined by need and at the time of need.

Goal-Based Scenarios (Schank 1998) – Problems are presented to small groups or individual learners. The problem solving environment, normally computerized, provides controls for the learner to request information that is delivered as if by story characters having different perspectives on the problem.

Anderson's Tutors (Anderson 1993) – Problems that can be solved through the construction of an algorithm are presented in a computer-based problem solving environment that accepts the learner's algorithmic operations. An expert problem solver checks the operation applied and responds with corrective feedback for errors. Acceptable operations update the display to show solving progress and make ready to accept the next operation from the learner.

Model-It (Jackson, Stratford et al. 1996) – Learners are presented with an environmental data base and a set of tools for analyzing, manipulating, and representing the data. One or more problems are chosen, and the learners examine the data base for answers using the tools to spot patterns and trends in the data.

Reciprocal Teaching (Brown and Palincsar 1989) – Learners are assigned specific question-asking responsibilities within a problem solving group. A comprehension problem is posed to the group, and they proceed to model problem solving behavior by performing their individual question-asking and answering responsibilities. By this means, a model of problem solving is made visible to the group over repeated experiences.

The PLATO® Problem Solving Activity (PSA): After an opening scenario, the learner enters an "adventure game" space with various tools and information available. As the learner moves to a problem solution, a learner log is maintained for later comparison to an expert path. An intelligent coach subsystem can provide full modeling (in "show me" mode), strategy- and tactical-level coaching (in "assist me" mode), or no coaching (in "leave me alone" mode). The architecture is designed for collaborative learning, but can be used by solo learners.

All of these examples and more that could be cited illustrate the variations in surface form that typify problem-based instruction. These examples differ in their use of media, modes of expression, manners of interacting, and the distribution of responsibility for learning, but all of them share the characteristic that they "involve pushing the student into a mode of problem solving on their own" (Collins, Brown et al. 1989), p. 483).

2) How can a person be instructed and solve problems at the same time?

The idea of learning by solving problems—which includes solving never-before encountered problems—is quite old, and occurs in every learning theory popular in the last century. It is somewhat ironic that a prominent *new* theory of learning-by-doing called Cognitive Apprenticeship should contain in its own name the name of an instructional method as old as human memory can recall—apprenticeship.

However, the notion of learning by doing or learning by solving problems appears on its face to be paradoxical because to solve a problem seems to require information that must be taught or conveyed before the problem can be solved. Direct instructional models often resolve the paradox by using a *tell-show-do* instructional sequence. By contrast, in the cognitive apprenticeship model (Collins, Brown et al. 1989), several instructional techniques are intended to resolve the paradox, including modeling, coaching, scaffolding, progressions of problems, and emphasis on problem solving heuristics and control, as well as learning to learn. These techniques combine to support the learner's own efforts to solve problems, and in the process the mental steps of discovery that are required of the learner become small enough that deduction, inference, search, and pattern matching can fill in the missing links between the problem and a solution before the learner's patience and interest expire.

As an example of the how the problem-based approach resolves the paradox, let us look further at the structure of a typical cognitive apprenticeship. Two event structures are implied by the theory: the modeling event and the problem posing. Instruction alternates between these two: providing exemplar performance that can be observed, studied, reverse engineered, and mimicked; and providing performance challenges that require the learner to apply to new situations the information, tactics, skills, and principles extracted

from models. Where models don't help, the learner can invent them at the moment of need. In a cognitive apprenticeship, both recall-application and invention can be used to explain how learning takes place during problem solving.

These event structures (modeling and problem posing) can serve as temporal centers for organizing the other learning-enhancing activities of cognitive apprenticeship (Gibbons, in preparation). Different arrangements of enhancement activities define different instructional patterns; for instance, coaching before, during, or after problem solving (feedback). Sequences of these events (including both modeling and problem posing) in turn become the problem sequences that make up the backbone of a problem-based curriculum.

Although Schank (Schank 1998) makes a good argument for the value of errors to learning, we believe the positive support principles provided by cognitive apprenticeship probably provide the most efficient and effective foundation on which to build design prescriptions for problem-based instruction in the majority of circumstances, especially where entry level knowledge is relatively low when compared to the desired learning outcome. Thus, we consider cognitive apprenticeship to be a particularly promising strategy for problem-based instruction. The alert reader will recognize the PLATO PSA as broadly consistent with the cognitive apprenticeship model, especially when in a collaborative learning context the discussion enhances the modeling and problem posing provided by the intelligent coach.

3. How does problem-based instruction interact with common instructional design conceptions?

Problem-based instruction requires the use of different design languages. The new design languages express structures for the:

- design of problems
- sequencing of problems
- construction of environments
- modeling of cause-effect systems
- modeling of expert performance, and the
- creation of instructional augmentations to support problem solving.

Among the terms of the new design languages are "problem," "control action," coaching," and "model representation." In contrast, the language of more common designs includes terms such as "task," "objective," "presentation," and "lesson." The terms of both languages contain implicit and subtle assumptions about the division of instructional content, the partitioning and cumulation of instructional events, and the structure of instructional strategies. They also are heavily biased by past practice in the manner of provision of instructional event control systems, the assignment of responsibility and initiative during instruction, and the structure of messaging elements.

The standard for technology-based instruction has become—by usage and by tool structure—fixed in the form of the tutorial, consisting of blocks of presentation, demonstration and practice, often in a *tell-show-do* sequence. These same instructional

components are found in a problem-based instructional sequence, but the learner is likely to encounter them in an event-driven sequence. The event-driven design requires the new language of design constructs.

Gibbons *et al* (Gibbons, Nelson et al. 2000) shows how it is useful to see the design of problem-based instructional products in terms of design layers, analogous to the layers of a building's design—inner structure, outer skin, services, inner space partitions, and so forth (Brand 1994), Chapter 2). We can characterize the several layers of an instructional design in terms of:

- Building-block structures used at each layer
- Principles that guide building block arrangement, alignment, and articulation
- Layer-specific prescriptive theories
- Layer-specific design decision-making processes
- Layer-specific design skills
- Layer-specific standards or styles and traditions
- Layer-specific design and development tools

The following list might exemplify a typical set of design layers for an instructional product. After each, we have included in parentheses the analogous step in a standard instructional design process for CBT development:

- Content/model layer—The layer where partitioning and organization of content structures takes place. This layer expresses the assumptions of the designer about the forms and parceling of subject-matter elements.
- Strategy layer—The layer where partitioning and organization of strategic structures takes place. Also the rules for the application of strategy elements are defined here. This layer is controlled by the instructional-theoretic bias or style of the designer. Problems are one of the main design building blocks at this layer.
- Control layer—The layer where controls and their actions are identified. This potentially includes controls over the instructional event, the content, the message, and the representation.
- Message layer—The layer where the messaging implications of both content and instructional strategy are embodied in a set of message tokens and the rules governing their assembly.
- Representation layer—The layer where message tokens are channelized and mapped to representation tokens and where representation tokens are assigned specific representation resource values.
- Data management layer—The layer where structures and rules for data collection and management are specified. Data collected in an adaptive instructional system are used by the strategy structures.
- Media-Logic layer—The layer where the elements of media delivery are described and organized into structures. This layer is where abstract designed structures from other layers are matched with the constructs provided by a particular design/development tool.

At each layer of an instructional design a different set of constructs and principles is brought to bear. Structures within each layer must articulate with those of each of the other layers, and all layers must align with development tool constructs (Gibbons, Lawless et al. 2001).

It is the lack of a tightly articulated, layered design vocabulary that makes it so difficult to implement problem-based instruction using conventional CBT instructional design practices and tools. For example, the event-driven structures of problem-based instruction, which allow the learner to use any sequence of actions while solving the problem, are extremely difficult to implement in a sequence-driven authoring tool. In another example, management of adaptive instructional strategies requires some degree of expert decision-making, but existing CBT authoring tools do not provide this capability to the designer. Therefore, our tools have tended to solidify the existing instructional paradigm and make the new one harder to choose (Gibbons and Fairweather 2000).

Viewing a design as many layers rather than in monolithic terms also leads to a new perception of the design process itself. Rather than following a process, design becomes tracing the implications of a decision across layers, and each layer's design unfolds uniquely as decisions are made as a result of work in other layers. There is no inviolable sequence in which to design the layers: design constraints and design decisions both modify the proper order of design. The designer determines the approach to each layer of a design and is motivated by the theoretical orientation (often self-originated) to that layer. For instance, several approaches to the instructional strategy layer exist, each possessing its own set of constructs from which whole strategies are put together.

Not all layers are required for every design, and for a given design/development project, the layers involved in the design may vary. Decisions made within one layer place constraints on other layers and so influence their structures and organization. A design approached with a particular instructional strategy in mind will differ from one in which a particular instructional medium is specified by the statement of the design problem.

Different priorities are placed on the different layers by different designers, which probably accounts for a great deal of variation in design styles. Some designers prefer to design for a particular medium, some for a particular strategy vision, and some for a particular structuring of the message or representation. Each of these priorities influences the order of the designer's decisions.

Different designers choose different layers as the "backbone"—or primary structural element—of their designs, either by preference or by constraint. A designer with strong tool background but little design training tends to favor media-logic structures as the primary organizational factor of designs. A designer strong in strategy will use the structure of the instructional strategy as the controlling pattern of the product. Problem-centered designers think of course structures in terms of sequences of problem units, and most often those problems are the events that bring learners into interactive contact with one or more dynamic models. Therefore, the designer with this commitment must also think in terms of the model units that stand behind the problems. This is the key design priority of Model-Centered Instruction (Gibbons, Nelson et al. 2000). In the future the

key skills of the instructional designer may well shift to emphasize the selection, sequencing, and posing of problems and the selection, sequencing, and presentation of dynamic models for interaction. Note also that the layered approach is generally consistent with the principles of object-oriented programming. The implications of this design approach for CBT authoring in general, and instructional simulations in particular, are substantial.

4. How are problems generated, selected, and sequenced?

Though there is a general trend in favor of problem-based instruction, there are few systematic methods for identifying problems and forming them into curricular sequences. In general practice, problems are derived mostly by individualistic rules, often on the basis of informed guesses. A disciplined technology of problem-based instruction, however, requires problem generation, selection, and sequencing to be replicable according to public, shareable principles. Useful problems are those that can attract and maintain learner engagement in problem solving.

Useful problems have several characteristics:

- Their scope corresponds with the structure of the larger curriculum plan, in terms of issues such as correspondence to the problem-solving skill being taught, and stage of expertise development.
- Their length corresponds with time segmenting constraints
- Requisite skills and knowledge match the target learner's level of expertise
- They can be expressed in terms within the learner's frame of reference
- Minimum modeling and representation requirements match media capabilities
- Solving resources and tools are available or can be built
- Problem development and delivery is affordable

Problem sets are constructed through a process that involves generation, selection, and sequencing. Each of these is discussed in turn below.

Generation. The designer must generate a pool of candidate problems from which problems used in instruction may be selected according to the criteria above. At present, generation usually takes place manually, but generation of problem sets is a concept that holds the key to scalable problem-based instruction systems. Just as we now use random number generators to introduce variable data into problems, we need to systematically vary other problem parameters in an automated fashion.

Detailed pre-design analysis increases in importance when problem-based designs are used. It is the first step in generating problems. In the previous section, we examined the potential advantages of two approaches: Scandura's FlexForm, and Gibbons' MCAP, over current cognitive task analysis methods, when applied to problem-based design.

Analysis, however, may take different forms and is not strictly limited to familiar forms of traditional task and objectives analysis. For projects with limited time or budget for analysis, more direct approaches can be used that avoid detailing of individual tasks:

- *Typical work cycle (time focus)* – A typical work cycle is examined to identify major activities carried out during one cycle. These activities become the basis for specifying problems.
- *Use cases (user-goal focus)* – Activity patterns of typical performers are examined to identify major activities. Since performers often work within the same environment to achieve different goals, this analysis describes different kinds of performance problems that are solved from the perspective of different users.
- *Critical incident (high priority task focus)* – For projects highly constrained in design, development, and delivery time, critical incidents that represent difficult, frequent, or especially challenging problems can be identified quickly.
- *Failure analysis (error focus)* – For high-risk performance environments, analysis can identify typical failure scenarios and use them as the basis for specifying instructional problems.

Note that these methods deal only with conglomerated tasks. The designer cannot make assumptions about completeness of such analyses: problem components that occur infrequently in real settings yet are critical to competent performance may be left out inadvertently.

In many cases, detailed task analysis is mandated or routinely performed due to high risk or cost factors connected with performance. Aviation and nuclear power training are examples of fields of practice that regulate training and trace training activities back to task-expressed training requirements. When detailed task analysis is performed, additional approaches to problem generation become available:

- *Hierarchical task grouping* – In addition to the use of individual tasks as problem bases, hierarchical branching-point tasks can also be used.
- *Combinatorial generation with filtering* – Problems can be generated through repeated substitutions of analyzed elements in different, computed groupings into a tokenized problem string. Generation by this means produces an enormous list of potential problems, so filtering rules are used to guide creation of only problems that might represent reasonable and useful combinations. Generation with filtering is then followed by hand-selection.

Instructional problems do not map one-to-one onto traditional task and objective hierarchies. Problems within a set will often overlap, repeating tasks or solving activities. Research has defined intermediary structures that generate elaboration sequences (Reigeluth and Nelson 1998) and Work Models (Bunderson, Gibbons et al. 1981; Gibbons, Bunderson et al. 1995). In the case of work models, an intermediary is created that translates directly into a problem structure. Carefully defined processes for

generating problem candidates are necessary if sequences are to be controlled with greater precision by designers in the future.

Selection and Sequencing. Problem selection principles should attempt to:

- Maximize and maintain learner engagement
- Regulate problem difficulty
- Regulate desirable levels of experience duplication and review
- Maximize the rate of progress through the body of intended learning
- Maximize the value of problem order through calculated or empirically determined problem-to-problem transfer.

The means for achieving such a balancing act are largely unexplored, and the issues listed above suggest questions for research. The readiness of a learner for a particular problem changes moment by moment and depends on problems and experiences which have gone before. The notion of transfer, always a difficult concept to operationalize, becomes even more complicated when it is seen in this way as a moving target.

Burton, Brown, and Fischer (Burton, Brown et al. 1984), in a chapter titled "Skiing as A Model of Instruction," describe the concept of *increasingly complex microworlds.* This is the arrangement of learning problems into flexible orders according to their degree of challenge in a way that takes into account learner readiness and the conditions of instruction—in their example, on the ski slopes. Problem difficulty is judged by a live instructor in terms of:

- The difficulty level of a complex of tasks
- The conditions available for task execution (slope, snow conditions, etc.)
- The degree of proficiency demanded
- The readiness of the learner to perform

The instructor must include in this latter judgment the momentary emotional (confidence) state of the learner, the learner's ability, and the size of challenge the learner is likely to enjoy. A live instructor is involved in making these judgments in a semi-directive negotiating posture with the learner. These are difficult criteria for computerized decision-making at present. One of the four major sets of principles for cognitive apprenticeship—based in part on the increasingly complex microworlds idea—advises designers to increase problem complexity while increasing problem diversity, at the same time balancing global and local views of the body of expertise. The analogy to Vygotsky's Zone of Proximal Development was noted above.

The existing standard for problem selection, therefore, is designer judgment, and that judgment is normally applied alike for all learners by a set sequence of problems. Contributing factors to this level of practice probably include cost as well as the difficulty of the decisions involved.

Some systems have been developed that focus on the automatic selection of problem sequences tailored for individuals. Anderson (Anderson 1993) in a series of tutors that instruct highly structured subject-matter, bases the selection of successive problems on the rules shown to be mastered by performance on previous problems. Because the subject-matters of Anderson's tutors are highly structured (mathematics), and because certain rules are prerequisite to the use of other rules temporally and inclusively, the reasonable number of sequences is limited.

White and Frederiksen (White and Frederiksen 1990) have researched progressions of mental models that facilitated the learning of electrical circuit principles. They provide principles for identifying models underlying problems and provide principles for sequencing both models and problems. Loftin, Baffes, and Hua (Loftin, Wang et al. 1989) also describe a system capable of selecting a sequence of problems for individual learners based on prior problem performance which they constructed for NASA for the training of flight dynamics officers. Newman, Grignetti, and Massey (Newman, Grignetti et al. 1989) accomplished a similar system for guided missile technicians. Their system is built around a core of simple problems whose complexity can be regulated mid-problem by introducing competing and interfering tasks.

Sequencing decisions interact highly with generation and selection. A problem set can be a dynamic flow of problems rather than static edifice of them. For this reason, the goal of sequencing problems dynamically at the time of instruction, though a complex problem itself, is desirable in principle.

Gibbons, Nelson and Richards (Gibbons, Nelson et al. 2000) identify several organizing principles for problem sequencing:

- Maximum coverage in limited time
- Cognitive load management
- Integration of complexes of prior learning
- Decontextualization of skills
- Practice to automaticity
- Maximum transfer

Each of these sequences maximizes one outcome, and each consists of potentially a different set of problems but definitely a different ordering of problems.

Problem ordering discussions take place from the learner's point of view, but the implications can be great also for the designer. The designer must create either a set of problems and an ordering algorithm or a problem generator and a set of rules to guide problem generation. The generation option as a practical approach to design faces considerable difficulties. Generation of problems means that a closed set of problems cannot be used as a design framework, so instructional strategy, message, and representation cannot be planned and compiled in advance but must also be generated. In the early days of computer-based instruction, Suppes (Suppes, Jerman et al. 1968)

pioneered problem generation, but the subject-matter consisted of highly structured mathematics problems, and the expectations for strategy, message, and surface representation at that time were comparatively simple. In today's multimedia environment the challenge accepted by Suppes would be much more difficult. But in the light of today's increased software capabilities, we consider generation a reachable goal, at least in many content areas.

If problem (and strategy and message and representation) generation is possible, the designer faces a new aspect of the problem sequencing question. When a problem set is limited in its number of problems by selection, the sequencing algorithm for problems is relatively simple. When the restraint on the number of problems is lifted and an unlimited number is possible, then the sequencing algorithm must be considerably more detailed and complex, controlling and deciding a large variety of problem variables. The curriculum in this case, rather than being an ordered set of problems of increasing difficulty, becomes a field of problems which, in order to cross, no two students need step on exactly the same sequence of stones.

The ability to generate problem sets and problems at the time of instruction would resolve many scalability issues for problem-based instruction. The Power Law of Practice, proposed by Newell and Rosenburg (Newell and Rosenbloom 1981), suggests that opportunities to practice reduce the time of performance—in essence helping with the automatization of the performance. It seems reasonable to believe that this law applies as well to the automatization of performances that involve problem solving in the original learning. If so, the ability to create numbers of similar but not identical problems without linear cost increases has benefit for learning of higher-order, complex cognitive skills in the same way additional practice can leverage automatization of common skills. Problem sequences that offer rapid-fire presentation of problems with smaller increments of difficulty may make possible sequences involving one longer problem followed by many shorter problems to maximize both learning momentum and time. Reducing the incremental difficulty between problems also suggests the possibility of less learning by direct instruction and more learning by the act of problem solving itself. Efficiency and effectiveness tradeoffs between direct instruction and this kind of intensive problem based instruction have yet to be researched.

Researching basic questions of problem generation, selection and sequencing requires much work, however. Progress will be slow until we have better typologies for problems and better metrics for describing between-problem relationships.

5. How are problem environments designed?

Problem-based instruction replaces the familiar central structures of message and strategy with a problem solving environment and its auxiliaries. Within this environment problems are posed and the learner is given access to models, resources, and tools to use during solution. Figure 1 below illustrates the main elements of this environment.

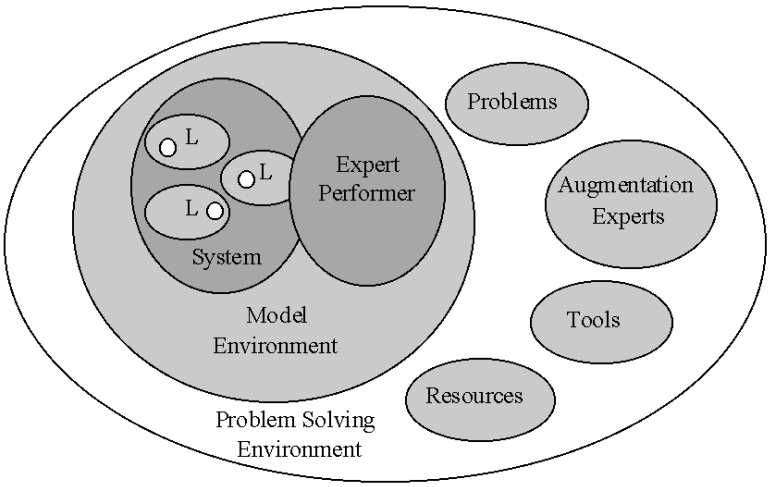

Figure 1. Elements of a problem solving environment

(from Gibbons & Fairweather, 2000).

The figure shows a main division between the *model environment* and four auxiliary functions: *problem administration*, *instructional augmentation*, *resource access*, and *tool access*. The elements of the model environment provide interaction with models of three kinds: environments, cause-effect systems, and expert performance. The *system models* model the behaviors of cause-effect systems, whether natural (photosynthesis) or human-made (a computer). *Expert performance models* model the behavior of expert performers with respect to those systems: either to observe them in operation or to manipulate their values and note the resultant changes. The *environment model*, the most subtle of the three, models the interaction of the systems with the environment they normally inhabit. This environment model contains one or more information-bearing locations (marked with an "L" in the figure). These locations are points at which the learner may see indications telling the current state of the system models. Also at these locations the learner may act on controls that convey new states to the models, resulting in observable reaction to the learner's action.

A flight simulation program found everywhere in software stores exhibits models of these three types. The user is placed in control of an aircraft and its systems. An expert pilot may be invoked to fly the aircraft to provide a demonstration of expert actions and thought patterns. A mostly-invisible environment of air surrounds the aircraft, containing winds and weather that influence aircraft flight and place demands on the user's problem

solving. The main information-bearing location in the average flight simulation is the view of the aircraft cockpit. From there, indicators of all the aircraft's systems can be observed, and controls can be operated. Some simulations allow other perspectives that provide useful information, including views from the ground, or from another aircraft flying beside the one operated by the learner.

Together, these three types of model constitute a *model environment* that operates within the larger *problem solving environment*. The problem solving environment provides additional supports that give interactions with the model environment their instructional value. These supports include one or more *problems* that can be posed with respect to the models. Students solve the problems by interacting with the models. *Augmentation experts* observe the interactions with the models and guide, judge, and assist with appropriate messaging and commentary. *Tools* placed within the problem solving environment are used by problem solvers to manipulate or represent problem data as decided by the learner. Tools can include spreadsheets, graphical tools, data analysis tools, report-building tools, or any other specialized tools that are normally used by a performer within the environment being modeled. *Resources* are also placed within the problem solving environment to supply information for problem solving or materials from which to build the solution or a response.

There are benefits to viewing the architecture of a problem-based instructional product in this way. First, each of the functional units described may be considered independently for media assignment. The architecture described makes no assumptions about media. Different elements can be executed using any appropriate medium, including live persons, physical structures, or virtual software structures.

Second, the elements of this architecture can be created as independent, portable modules capable of being reused. An analogy is to a spreadsheet. This mathematical tool can be executed using paper and pencil or computer, and the tool itself can be reused in problem solving environments other than the one it was specifically designed for. The same is true of the elements of the model environment as well: they may be designed to be portable across problem solving environments. This suggests several uses for the independent instructional object (Wiley 2000).

Third, this architecture separates the problem from the problem solving environment. The problem itself can be studied as an instructional device in its own right. The dimensions and properties of problems can be determined; the principles for problem construction and generation can be tested; and more detailed and general principles for problem sequencing can be derived because the problem can be considered apart from its content.

Designing interactive models and problems assumes that the models will differ in one or more ways from real objects, settings, and situations. It is the nature of models to be less than real. It is appropriate to say that every model experience is *denatured* in some sense. Therefore the experience learners get from interactions with the model during problem solving will be less than full-fidelity. The principle of denaturing is critically important to the construction of problem-based instructional products because costs normally decrease for more denatured models and increase for less denatured ones. The designer

of problem-based instruction can design with more precision and less cost if it is known what elements of models and problems require fidelity and which have low marginal contribution in terms of learning. Furthermore, as mentioned above, it may well be that a denatured problem will help the learner perceive the underlying problem structure early in the sequence of problems. Research in this area is badly needed.

6. What are the implications for tools?

What may have seemed at first as a simple shift to include problems more centrally into instruction, on closer examination appears to have profound implications for basic design assumptions, design processes, and product architectures. These implications ripple outward also to include needs for new or modified development tool capabilities. The requirements of the tutorial structure have long dominated tool structures, leading to programming-eliminating frames, menus, interfaces, and authoring systems. With the emergence of problem-based instruction as a major trend, development tools will now be called upon to provide capabilities that are required not for efficiency in delivering message or executing structured strategies but for efficiency in developing interactive models, problem sets, and instructional augmentations to problem solving such as coaching and feedback systems.

A projection of these new demands on tools was attempted by Gibbons and Fairweather (Gibbons and Fairweather 2000), who suggested the following areas either for new capabilities or strengthening and facilitating of existing capabilities:

- *Facilitation for environment and location building* – The locations within an environment are the points of interface through which the learner interacts with the controls and indications of a model during problem solving. Future tools should provide for the construction of environmental locations as unitized objects, and it should be possible to link individual locations into larger, integrated environment suites with linking paths between the locations. This should include the construction of both the underlying and the surface properties of locations.

- *Facilitation of model building and integration* – Tools for model building exist, but these tools must become accessible to the average designer and be capable of constructing a range of model types without specialized training or programming skills. Moreover, building suites of models that represent progressions from simple to more complex must be facilitated. Integration of these models with problem data sets must be made as non-technical as possible.

- *Facilitation of expert system building and integration* – Providing expert performance models and expert augmentation systems for coaching and feedback are only two areas of problem-based instructional products that require the construction and integration of expert systems. Expert systems will also be useful in the construction of problem sequencers. Development tools do not now provide for easy creation or integration of expert systems with developed products.

- *Facilitation of thread building* – During problem-based instruction that features expert augmentation it is often necessary to simultaneously provide interaction with one or more models while at the same time providing expert augmentation services. These parallel functions represent multiple threads of execution. Tools must in the future provide access to the designer for the creation and management of these threads.

- *Facilitation of integration* – Unification of models, problem managers, and instructional managers into functioning products that orchestrate and coordinate their individual functions will require the ability to plan and develop higher-level architectural structures. These must be made accessible to the designer.

- *Facilitation of data management* – Problem solving interactions produce copious amounts of potentially useful data. Difficult-to-use variable-keeping systems available in development tools are a hindrance to the creation of systems that can handle this large amount of data and supply it in a timely fashion to decision-making processes during instruction. This capability is a requirement if adaptivity during problem solving is to be achieved.

- *Facilitation of in-product tool building* – Tools to be used during problem solving must be made easy to attach to problem solving environments as standard interface overlays or as integrated elements of displays. The means for the creation of new tools and their attachment to problem solving environments will be a growing area of need.

- *Facilitation of multisource message/display management* – Not only must the integration of individual functional modules be possible, but also the designer must be able to integrate the various messaging and information contributions of each to a common, shared display surface. Interface-building functions that manage multiple source inputs must be made available to the designer.

- *Tools for model constraint* – Greater use of individual models can be achieved if models can be masked or constrained and then revealed as the complexity of problem challenges increases.

- *Support for group processes* – Both design and training functions are becoming group-oriented. Tools must in the future provide collaborative surfaces for both the design and delivery of instructional events.

Advances in top-down design technologies for large-scale object-oriented software systems hold promise for meeting many of these needs. One step in this direction is the *WebPLATO* authoring technology. This technology is built on a carefully structured class library of proprietary objects, with only the foundation classes provided by a commercially-available web authoring tool set. Thus, rather than working directly in the web tool set, a series of editors provides for automated input of the properties for each of the high-level objects. A future step in the direction of meeting these requirements may be through application of next-generation general software design tools (Scandura, personal communication).

Conclusion

This review demonstrates that while there has been considerable progress in our understanding of problem-solving behavior and how it is learned, the current corresponding theories of instruction for problem-solving generally lack the precision needed for generalizable instructional design practices. We have examined some promising new sources of instructional theory, and discussed key requirements for problem-based instructional design. We have argued that problem-based design represents an alternative to conventional message-based design. And, we have outlined the requirements for a new generation of authoring tools and technologies needed to support problem-based instructional design.

We believe the problem-based design system proposed here, when it has matured into a stable and reproducible technology with a complete tool set, will offer a number of advantages over current practice in problem-based instruction:

- *Close linkage of analysis, design and development.* Current practice isolates analysis, design and development, which leads at best to considerable wasted effort as the work products of one phase are reworked in the next. At worst, this disconnect allows the developer to ignore key analysis and design insights. We believe the layered design system outlined here holds promise for a much more efficient and effective system.

  Furthermore, because of the denaturing issue we discussed, and because degree of structure is in the eye of the learner, there is a tight linkage between analysis technique, assumptions about the learner, and intended design strategy. The kind of layered analysis and design system described here supports this tight linkage.

- *Increased precision of analysis and design.* In current (constructivist) practice, analysis and design procedures and standards typically lack precision. This makes the work products very idiosyncratic: no two designers would produce similar designs for a given topic.

- *Better measurement of learning outcomes.* The lack of precision in current practice also has frustrated efforts at precise description and measurement of learning outcomes. This, in turn, has led researchers to advocate ethnographic methods of assessment which are costly in large scale and difficult to scale up. We believe progress in analysis and design is the key prerequisite to improved measurement of learning outcomes.

- *Improvement in cognitive apprenticeship.* Cognitive apprenticeship has generally been descriptive, not prescriptive, therefore is not a design theory. We believe the kind of prescriptive theory for analysis and design outlined here will greatly strengthen the cognitive apprenticeship model. Specifically, it will provide a systematic way to design problems with maximum instructional utility, to scale and sequence those problems, and to describe the learning outcomes precisely. Every increment in systematizing what is now a largely intuitive process will help reduce the cost of problem-based instruction and improve its scalability.

An additional implication concerns the distinction we made between tutorial (message-based) and simulation (problem-based) instructional strategies. The distinction is certainly common in the literature, often accompanied by an explicit or implicit assumption (from a constructivist perspective) that tutorials are inferior. The problem-based instructional design perspective offers a different view. Using problem-based design, a tutorial is just a highly structured and (usually) denatured case of simulation with coaching and feedback. Thus, the tutorial is one end of the continuum, and fully realistic simulation is at the other. Varying stages of denatured instructional simulation are in between. There are critical implications of cost, technology and instructional efficiency associated with each point on this continuum. Thus, the decision of where to place a given instructional event on this continuum is essentially a cost-benefit decision. We therefore reject attempts to categorically dismiss any instructional strategy on the continuum on the basis of learning theory.

We argued that multiple analysis techniques will be needed for multiple purposes, and any given analysis and design will necessarily be relatively purpose-specific. As noted in our discussion, this is quite consistent with the principle of layered design. Thus, problem-based, layered analysis and design is likely to evolve as a family of design principles and techniques suitable for various contexts.

There are also implications for the feasibility of reusable learning objects (as in the *IMS* project). Work so far on reusability has largely been content- and message-centric. The design layers discussed here generally are left implicit, and presumably are embedded in the reusable learning objects. This inevitably makes the learning objects much more context- and learner-specific than would be desirable, and limits their reusability. We believe it may more desirable to use a problem-based design with a layered design approach as described here. This approach should allow for reuse of certain aspects of the design across contexts, thus strengthening reusability of objects.

Ultimately, our ability to articulate a practical, stable, reproducible and cost-efficient system for problem-based analysis and design will determine the success of the method, especially in computer-based instructional environments. Current practice, in both live instruction and in computer-based environments, is intuitive and labor-intensive. Each evolutionary step in development of the proposed layered design system will engender improvements in instruction, especially in automated instruction.

References

References

Anderson, J. R. (1980). Cognitive psychology and its implications. New York, Freeman.

Anderson, J. R. (1993). Rules of the mind. Hillsdale, NJ, Lawrence Erlbaum Associates.

Anderson, J. R. (1995). Learning and memory: an integrated approach. New York, Wiley.

Atkinson, R. K., S. Derry, et al. (2000). "Learning from examples: instructional principles frm the worked examples research." Review of Educational Research **70**(2): 181-214.

Barrows, H. S. (1988). The tutorial process. Springfield, IL, Southern Illinois University School of Medicine.

Blumenfeld, P., B. J. Fishman, et al. (2000). "Creating Usable Innovations in Systemic Reform: Scaling Up Technology-Embedded Project-Based Science in Urban Schools." Educational Psychologist **35**(3): 149-164.

Brand, S. (1994). How buildings learn: What happens after they're built. New York, Penguin Books.

Brown, A. L. and A. S. Palincsar (1989). Guided, cooperative learning and individual knowledge acquisition. Knowing, learning, and instruction: Essays in honor of Robert Glaser. L. Resnick. Hillsdale, NJ, Lawrence Erlbaum Associates.

Bruning, R. H., G. J. Schraw, et al. (1999). Cognitive Psychology and Instruction. Columbus, OH, Merrill.

Bunderson, C. V., A. S. Gibbons, et al. (1981). "Work models: Beyond instructional objectives." Instructional Science **10**: 205-215.

Burton, R. R., J. S. Brown, et al. (1984). Skiing as a model of instruction. Everyday cognition: Its development in social context. B. Rogoff and J. Lave. Cambridge, MA, Harvard University Press.

Collins, A., J. S. Brown, et al. (1989). Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. Knowing, learning, and instruction: Essays in honor of Robert Glaser. L. Resnick. Hillsdale, NJ, Lawrence Erlbaum Associates.

Duffy, T. M. and D. J. Cunningham (1996). Constructivism: Implications for the design and delivery of instruction. Handbook of Research for Educational Communications and Technology. D. H. Jonassen. New York, Association for Educational Communications and Technology, Macmillan Library Reference USA.

Elstein, A. S., L. S. Shulman, et al. (1978). Medical problem solving: an analysis of clinical reasoning. Cambridge, MA, Harvard University Press.

Foshay, D. R. (1995). What We Know (And What We Don't Know) About Training of Cognitive Strategies for Technical Problem-Solving". Educational Technology: Past, Present and Future. G. Anglin. Englewood, CO, Libraries Unlimited**:** 374-383.

Foshay, W. R. (1997). "What We Know (And What We Don't Know) About Training for Problem Solving: An Update." Performance Improvement **36**(3): 40-45.

Foshay, W. R. (1998). Instructional philosophy and strategic direction of the PLATO system. Bloomington, MN, PLATO Learning, Inc.

Foshay, W. R. and J. Kirkley (1998). Principles for Teaching Problem Solving. Bloomington, MN, PLATO Learning, Inc.

Funkhouser, C. and J. Dennis (1992). "The effects of problem-solving software on problem-solving ability." Journal of Research on Computing in Education **24**(3): 338-347.

Gagne, R. (1985). The conditions of learning. New York, Holt, Rinehard and Winston.

Gibbons, A. S. (1998). Model-centered instruction. Annual Meeting of the American Educational Research Association, San Diego, CA.

Gibbons, A. S., C. V. Bunderson, et al. (1995). "Work models: Still beyond instructional objectives." Machine-Mediated Learning **5**(3&4): 221-236.

Gibbons, A. S. and P. G. Fairweather (2000). Computer-based instruction. Training and Retraining: A Handbook for Business, INdustry, Government, and the Military. S. Tobias and J. D. Fletcher. New York, Macmillan Reference USA.

Gibbons, A. S., K. A. Lawless, et al. (2001). The web and model-centered instruction. Web-based training. B. R. Khan. Englewood Cliffs, NJ, Educational Technology Publications.

Gibbons, A. S., J. Nelson, et al. (2000). Model-centered analysis process (MCAP): A pre-design analysis methodology. Idaho Falls, ID, Center for Human-Systems Simulation, Idaho National Engineering and Environmental Laboratory (DOE).

Gibbons, A. S., J. Nelson, et al. (2000). Theoretical and practical requirements for a system of pre-design analysis: State of the art of pre-design analysis. Idaho Falls, ID, Center for Human-Systems Simulation, Idaho National Engineering and Environmental Laboratory (DOE).

Gibbons, A. S., J. S. Nelson, et al. (1999). Model-Centered Analysis Process (MCAP): a pre-design analysis methodology. Whte Paper. Logan, UT**:** 38.

Gibbons, A. S., J. S. Nelson, et al. (1999). Theoretical and practical requirements for a system of pre-design analysis: state-of-the-art pre-design analysis. Idaho Falls, ID, Idaho National Engineering and Environmental Laboratory**:** 79.

Goldman, S. R., L. K. Zech, et al. (1999). "Computer technology and complex problem solving: Issues in the study of complex cognitive activity." Instructional Science **27**: 235-268.

Grabinger, R. S. (1996). Rich environments for active learning. Handbook of research for educational communications and technology. D. H. Jonassen. New York, Association for Educational Communications and Technology, Macmillan Library Reference USA.

Hall, E. P., S. P. Gott, et al. (1995). A Procedural Guide To Cognitive Task Analysis: The PARI Methodology.

Hannafin, M. J. (1992). "Emerging technologies, ISD, and learning environments: Critical perspectives." Educational Technology Research and Development **40**(1): 49-63.

Hannafin, M. J., K. M. Hannafin, et al. (1997). "Grounded practice and the design of constructivist learning environments." Educational Technology Research and Development **45**(3): 101-117.

Jackson, S. L., S. J. Stratford, et al. (1996). "A learner centered tool for students building models." Communications of the ACM **39**(4): 48-49.

Jonassen, D. H., Ed. (1996). Handbook of Research on Educational Communications and Technology: A Project of the Association for Educational Communications and Technology. New York, Simon & Schuster Macmillan.

Jonassen, D. H. (1997). "Instructional Design Models for Well-Structured and Ill-Structured Problem-Solving Learning Outcomes." Educational Technology Research and Development **45**(1): 65-94.

Jonassen, D. H. (2000). "Toward a Design Theory of Problem Solving." Educational Technology Research and Development **48**(4): 63-85.

Lesgold, A., S. Lajoie, et al. (1992). SHERLOCK: a cogached practice environment for an electronics troubleshooting job. Computer-assisted instruction and intelligent tutoring systems: shared goals and complementary approaches. J. H. Larkin and R. W. Chabay. Hillsdale, NJ, Lawrence Erlbaum Associates**:** 201-238.

Loftin, R. B., L. Wang, et al. (1989). "An intelligent system for training space shuttle flight controllers in sattelite deployment procedures." Machine-Mediated Learning **3**: 41-51.

Mager, R. (1982). Troubleshooting the troubleshooting course, or debug d'bugs. Belmont, CA, Pitman Learning.

Merrill, M. D. and I. R. T. 1 (1993). "Instructional Transaction Theory: Knowledge Relationships Among Processes, Entities, and Activities." Educational Technology(April, 1993): 5-16.

Newell, A. and P. S. Rosenbloom (1981). Mechanisms of skill acquisition and the law of practice. Cognitive skills and their acquisition. J. R. Anderson. Hillsdale, NJ, Lawrence Erlbaum Associates.

Newell, A. and H. Simon (1972). Human problem solving. Englewood Cliffs, NJ, Prentice Hall.

Newman, D., M. Grignetti, et al. (1989). "Intelligent conduct of fire trainer: Intelligent technology applied to simulator-based training." Machine-Mediated Learning **3**: 29-39.

Ohlsson, S. (1995). Learning to do and learning to understand: a lesson and a challenge for cogntive modeling. Learnintg in humans and machines: towards an interdisciplinary learning science. P. Reimann and H. Spada. Terrytown, NY, Elsevier.

Orey, M. A. and W. A. Nelson (1993). "Development principles for intelligent tutoring systems: integrating cognitive theory into the development of computer-based instruction." Educational Technology Research and Development **41**(1): 59-72.

Reigeluth, C. M. (1999). Instructional-design theories and models: A new paradigm of instructional theory. Mahwah, NJ, Lawrence Erlbaum Associates.

Reigeluth, C. M. and L. M. Nelson (1998). Instructional Strategies for Problem-Solving Tasks. AECT 1998.

Scandura, J. M. (1977). Problem solving: a structural/process approach with instructional implications. New York, Academic Press.

Scandura, J. M. (2001). Structural learning theory: current status and new perspectives. American Educational Research Association, Seattle, WA.

Schank, R. C. (1998). Inside multi-media case based instruction. Mahwah, NJ, Lawrence Erlbaum Associates.

Schank, R. C., A. Kass, et al., Eds. (1994). Inside case-based explanation. Hillsdale, NJ, Erlbaum Associates.

Shute, V. J. and J. Psotka (1996). Intelligent tutoring systems: Past, present, and future. Handbook of Research for Educational Communications and Technology. D. H.

Jonassen. New York, Association for Educational Communications and Technology, Macmillan Library Reference USA.

Suppes, P., M. Jerman, et al. (1968). Computer-Assisted Instruction at Stanford: The 1965-66 Arithmetic Drill-and-Practice Program. New York, Academic Press.

Tenney, Y. J. and L. C. Kurland (1988). The Development of Troubleshooting Expertise in Radar Mechanics. Intelligent Tutoring Systems - Lessons Learned. J. Psotka, M. L.D. and S. A. Mutter. Hillsdale, NJ, Erlbaum Associates**:** 59-83.

van Merrienboer, J. J. G. (1997). Training complex cognitive skills: a four-component instructional design model for technical training. Englewood Cliffs, NJ, Educational Technology Publications.

Vygotsky, L. S. (1978). Mind in society: The development of higher psychological processes. Cambridge, MA, Harvard University Press.

White, B. Y. and J. Frederiksen (1990). "Causal model progressions as a foundation for intelligent learning environments." Artificial Intelligence **24**: 99-157.

Wiley, D. (2000). The instructional use of learning objects. Bloomington, IN, Association for Educational Communications and Technology (http://reusability.org/read).

Wilson, B. G. and P. Cole (1996). Cognitive teaching models. Handbook of research for educational communications and technology. D. H. Jonassen. New York, Association for Educational Communications and Technology, Macmillan Reference USA.