

Apprenticeship-Based Learning Environments: A Principled Approach to Providing Software-Realized Scaffolding through Hypermedia

Mark Guzdial and Colleen Kehoe
GVU Center and EduTech Institute
College of Computing
801 Atlantic Dr.
Atlanta, GA 30332-0280
{guzdial, colleen}@cc.gatech.edu

Abstract

Becoming a skilled practitioner means learning both conceptual and process knowledge. We propose the use of hypermedia to provide components of an apprenticeship learning model for students, in which process and conceptual knowledge learning is integrated. Our model for this use of hypermedia is called an Apprenticeship Based Learning Environment (ABLE). We present seven design principles for an ABLE based on learning research, and show how these are met in our first ABLE, STABLE. The formative evaluation of STABLE suggests that it supports learning and performance, but leaves the students unsatisfied. We suggest an eighth principle for iterative design of ABLE tools that meets students' needs.

Helping Students Become Skilled Practitioners

Becoming a skilled practitioner involves gaining deep conceptual knowledge as well as process knowledge in order to solve problems. Conceptual knowledge is the information that one most often thinks about as propositions about a field: Definitions, patterns, facts about relationships (e.g., " $F=ma$ "). Process knowledge is the information about how one goes about activities: Heuristics for how to decide the next step, how to use the tools and techniques of a domain, approaches that work well with given kinds of problems. Skilled practitioners have an ability to undertake a complex process in the domain of their expertise and to change their process to suit a given problem in the domain (Schon, 1982). Doctors,

for example, must use an enormous amount of knowledge about anatomy and more specialized fields and apply it to design treatments to solve problems. While there is certainly deep conceptual knowledge that needs to be learned in these kinds of domains, the conceptual knowledge is related to or even in service of the process knowledge, e.g., a mechanical engineer may learn about gears in order to design artifacts that use gears. In general, there is a need for students to learn about process as well as concepts.

The most common way across the ages in which students have learned process on the way to becoming skilled practitioners is through *apprenticeship* (Collins, Brown, & Newman, 1989; Rogoff, 1990). Apprenticeship has students learn process through active participation in the task. Student participation may be very limited at first while students gain an understanding of the process through observation and making small contributions (what Lave and Wenger call *legitimate peripheral participation* (Lave & Wenger, 1991)) but the involvement develops into full participation and eventually task ownership.

Apprenticeship learning has several characteristics which makes it effective:

- *Scaffolding*: Students are supported to both (a) be successful in their process and (b) learn the process. This kind of support is called *scaffolding*. Scaffolding is faded over time as the student gains in competency and can take on more of the process without support. Scaffolding has three components to it (Guzdial, 1995):
 - *Communicating Process*: The process is shown or demonstrated to the student, often in a structured and simplified form.
 - *Coaching*: Support is provided in response to student activity – often in response to student failure.
 - *Eliciting Articulation*: Encouraging the student to talk about and reflect on the process in an explicit way.

- *Authenticity*: Because all the process learning is in terms of active participation in the task, the authenticity of the learning and the relationship between daily activity and learning goals is more obvious than it is in a traditional classroom (Farnham-Diggory, 1990). Authenticity can help to improve transfer of the learned knowledge and student motivation (Vanderbilt, 1990).
- *Sequenced Tasks*: While not all traditional apprenticeships offered well sequenced tasks (i.e., in a real working shop, the next job to do depends on the next customer to walk through the door), effective apprenticeship environments provided students with the right task to undertake when the right kind of process (and concept) learning was needed (Collins et al., 1989). Too complicated of a task can reduce student motivation, while too simple of a task would not encourage learning (Blumenfeld et al., 1991).
- *Collaboration Across A Range of Abilities*: Traditional apprenticeship environments involved multiple apprentices at various levels working on a variety of tasks. Such an environment provided ample opportunity for just-in-time scaffolding as more senior students were available to more novice students to answer questions and provide demonstrations (coaching). In addition to supporting novice students, the opportunity to work as teacher helps more senior students articulate and refine their learning, as in a reciprocal teaching setting (Palincsar, 1986; Palincsar & Brown, 1984).

School reform efforts have attempted to build on the strengths of an apprenticeship approach as a means of improving learning in domains not typically associated with apprenticeship, like reading, mathematics, and science learning. Becoming a skilled practitioner may not be an explicit goal (e.g., not all students need to become expert scientists and mathematicians), but the learning outcomes in an apprenticeship match the goals of many school reform efforts. Apprenticeship is a model that results in students

gaining the conceptual and process knowledge of a skilled practitioner, which includes outcomes such as problem-solving skills and transfer of knowledge that school reform efforts seek to develop. The *cognitive apprenticeship* approach explicitly borrows from the apprenticeship model (Collins et al., 1989), while the *project-based learning* approach emphasizes some characteristics of an apprenticeship (such as scaffolding and authenticity) (Blumenfeld et al., 1991).

However, creating an apprenticeship learning experience is hard in traditional classrooms. In elementary and middle school classrooms, with one teacher and perhaps thirty students, trying to undertake project-based learning requires a significant effort on the part of the teacher as well as classroom and curricular support (Krajcik, Blumenfeld, Soloway, & Marx, 1992; Krajcik, Blumenfeld, Marx, & Soloway, 1994). Supporting the individual efforts of all of those students in 50 minute daily time periods is a herculean task. In undergraduate education, where becoming a skilled practitioner in the classic apprenticeship model *is* often an explicit goal (Augustine & Vest, 1994; Coleman, 1996; Dixon, 1991), the even greater numbers of students (e.g., 75 to 200 or more students) and even smaller time periods (e.g., two or three meetings per week) make a traditional apprenticeship model seem impossible.

It is not surprising that education researchers have looked to technology as a potential solution to the problem of gaining the advantages of an apprenticeship model in traditional classroom settings (Collins, 1990). Technology can help to reduce unnecessary details while scaffolding students' process during practice (Hmelo & Guzdial, 1996), at both the elementary and middle school (Blumenfeld et al., 1991) as well as undergraduate education (Kolodner, 1995; Kolodner, Hmelo, & Narayanan, 1996).

We are using technology to create an apprenticeship-based learning environment (ABLE) for undergraduate education. Our strategy is to use a principled approach to designing hypermedia that provides *software-realized scaffolding* (Guzdial, 1995). Software-realized scaffolding is providing the same support that a master provides an (Rogoff, 1990; Wood, Bruner, & Ross, 1975), but in software.

Our use of hypermedia in ABLEs is unique in that the focus in an ABLE is to present process knowledge and the conceptual knowledge that supports a process. Most use of hypermedia is to present declarative knowledge. While conceptual knowledge does appear in an ABLE, the media in an ABLE are structured around process, with conceptual knowledge serving to explain strategies, introduce abstract patterns, and generally supporting process learning.

- Use of hypermedia to be used concurrently with activity or to support other activity is relatively rare. Bottino has developed hypermedia where the media is integrated with the task activity (Bottino, Chiappini, & Ferrari, 1994). In the ABLE approach, we mean for the tool to be used concurrently with other activity, but without modifying other activity. Our approach is somewhat less expensive than modifying whatever tools are needed in the activity.
- Hsi and Agogino have developed hypermedia to support design learning by presenting a design case with reflection activities, but it is not designed to be used while the student is actually trying to design (Hsi & Agogino, 1994). Students are meant to use an ABLE literally side-by-side with some other activity, from programming to cooking.
- There are high functionality hypermedia case bases, such as the Case Based Design Aids (CBDA) (Domeshek & Kolodner, 1992), which provide multi-faceted perspectives on designs and serve as resources to new designs. However, these CBDA focus on describing product (i.e., the result of the design process)

primarily. While intermediate artifacts may appear in the case base, communicating process is not the primary goal. ABLEs are uniquely designed to communicate process, with conceptual knowledge in support of that process, and are meant to be used in conjunction with other activity.

We have designed the ABLE structure around a set of six principles that are based on findings about how students learn and how to support student learning. These principles define the kind of information to be included in a case, how it should be structured, and what the students' interface to the case should include. In this paper, we present the principles of the ABLE approach and an instance of that approach, *STABLE* (SmallTalk Apprenticeship-Based Learning Environment).

We have conducted a formative evaluation of *STABLE* with promising and some surprising results. In general, we have found that *STABLE* supported student performance and learning beyond what similar students not using *STABLE* were able to achieve. However, *STABLE* did not feel friendly and satisfying to the students. In our analyses, we found that there is an additional design principle that emerged from actual practice with the tool. While we believe in a principled design approach, the reality is that a new kind of educational software creates a new context not adequately described by previous research. For example, we found that we had assumed that students would investigate cases in depth, when the dominant use strategy was to view several comparable cases and contrast them. An iterative design approach is required, even when starting from well-founded principles.

Technological Approaches to Learning Skilled Practice

Educational technologists have created a variety of environments to support students in learning the conceptual and process knowledge for skilled practice. We describe three

efforts: Intelligent Tutoring Systems, Goal Based Scenarios, and Design Support Environments. Though not all of these explicitly use apprenticeship as a model, there are clearly characteristics in common between the approaches used here and apprenticeship.

Intelligent tutoring systems (ITS's)¹, the classic instances of which are those created by John Anderson of Carnegie-Mellon University, help students develop skills in domains such as generating geometry proofs, programming in LISP, or solving algebra word problems (Anderson, Boyle, Corbett, & Lewis, 1990; Anderson, Boyle, & Reiser, 1985; Anderson, Conrad, & Corbett, 1989; Anderson, Corbett, Koedinger, & Pelletier, 1995). ITS's guide students through undertaking a process and provide them with a well-selected sequence of tasks to undertake. The students' process is traced against a model of a successful process, and students are held to that model. A modern ITS uses a variety of representations to present process, such as the innovative tree representations of proofs in the Geometry Tutor (Anderson et al., 1990) or in GIL (Merrill, Reiser, Beekelaar, & Hamid, 1992).

The greatest difference between ITS's and most instantiations of an apprenticeship model of learning is the authenticity of the knowledge and task. Early ITS's used relatively small and well-defined problems that could be easily modeled. Recent efforts in ITS have emphasized lifelike settings for problems (particularly in the PAT tutor developed in conjunction with Pittsburgh schools (Koedinger & Sucker, 1996)), but the process of problem-solving itself is still not lifelike in most ITS's. Students do not face complexity and uncertainty that characterizes problem-solving in many domains (Spiro, Coulson, Feltovich, & Anderson, 1988). While reducing the complexity improves the efficiency of

¹We are focusing particularly here on K-12 and post-secondary applications of ITS, as opposed to professional uses which tend to be much more based on authentic tasks.

learning, it may also remove key elements of skilled practice that are difficult to model in an ITS approach. For example, students never debug programs in the Lisp Tutor—programs are not executed until they are already correct (Anderson et al., 1989), while debugging is a critical element of real computer science practice (Spohrer, 1989). Without this complexity, there is also little need in an ITS setting for the collaboration seen in apprenticeships.

Rather, we see more of a competitive social setting developing (Schofield, Britt, & Eurich-Fulcer, 1991), where performance in the ITS is the goal, rather than use of the developing skills.

Goal-Based Scenarios (GBS) are developed at the Institute for the Learning Sciences as authentic environments for students to gain process skills and requisite conceptual knowledge (Schank, 1992; Schank, Fano, Bell, & Jona, 1994). Students using GBS are provided with an interesting situation in which they have goals to achieve. Students are provided the opportunity and resources to achieve these goals. Students' progress is compared against a model of a successful process, and when students fail, they are provided conceptual and process information (usually presented in story form) to allow them to understand and correct their failure.

While GBS are closer to an apprenticeship environment than ITS, they lack the ability to support students through truly complex processes where a detailed model for tracing successful process cannot be provided. In many fields, such as engineering design, there is no single correct process at the appropriate level of detail, and perhaps no way to test for failure until much later in the process. For example, if a student decides to search in a textbook for a piece of information before doing a Web search, is her process wrong? If a GBS cannot interpret whether an action was on the path of a successful process or not, it cannot provide feedback appropriate to the action. While the GBS may provide other

support that might help with learning in a more complex domain, it would not be in the form of stories that explain failure.

Design Support Environments (DSE) are explicitly aimed at supporting students engaged in design of complex artifacts, such as software, instruction, and static body diagrams (Guzdial, Weingrad, Boyle, & Soloway, 1992; Soloway et al., 1993). DSE provide scaffolding in the software (hence, *software-realized* scaffolding) by simplifying the task environment through

- changing the process and providing a large library of cases (Kolodner, 1993; Kolodner, 1995) (often in the form of working components),
- providing coaching that is appropriate to the students general process (e.g., guiding students through a top-down design process in programming),
- frequently eliciting articulation (e.g., goals, journal entries, predictions), and occasionally,
- providing a mechanism for fading the scaffolding (Guzdial, 1995).

DSE's typically do not present problems or situations to students (though they can be aimed at particular classes of problems or situations by placing the right kinds of examples in their libraries) and instead try to provide scaffolding for the general task of design. The authenticity and sequence of tasks is left to the curriculum, though typically, DSE's have been used with relatively authentic tasks in a sequence that gradually increases in complexity. No DSE has explicitly supported collaboration, though some have been used in a collaborative setting (Urduan, Blumenfeld, Brade, & Soloway, 1993), and there is no philosophical reason why collaboration could not be used with DSEs.

Probably the biggest drawback to DSEs is that they are expensive to produce.

- Large libraries of cases (components or examples) are challenging to assemble. The tough questions are how many cases are needed to be useful and which cases are the ones most useful to the students (especially difficult in ill-defined design domains).
- Designing software that structures and perhaps changes the students' task is complicated because it often demands changing the interface to computer-based tools (like language interpreters and compilers). DSE approaches have been to re-implement existing tools (Hohmann, Guzdial, & Soloway, 1992) or provide a whole new interface to existing tools (Guzdial, 1995).
- Scaffolding (coaching and articulation, especially) is difficult to design, and there are not yet enough examples for a methodology to exist of how to design scaffolding (Jackson, Stratford, Krajcik, & Soloway, 1995; Soloway, Guzdial, & Hay, 1994).

A Principled Approach to Designing Software-Realized Scaffolding as Hypermedia

The Apprenticeship-Based Learning Environment (ABLE) approach is to provide software-realized scaffolding, much as in DSE tools, but through hypermedia structured around process. The structuring of the hypermedia is based on principles of learning support in the educational literature. The ABLE approach seeks to provide an apprenticeship model of learning in a relatively low-cost but high-benefit manner. We focus on supporting design tasks in an ABLE because of its educational benefits (Carver, Lehrer, Connell, & Erickson, 1992; Hmelo, Holton, Allen, & Kolodner, 1997; Lehrer, 1992; Puntambekar & Kolodner, 1997) and the kinds of classes we have been working with, though other kinds of processes (such as cooking) could also be supported with an ABLE.

We begin by characterizing what an apprenticeship in a design class might be like. We characterize the core components as:

A Master who provides authentic tasks to the apprentices (students) and scaffolds the process in a community of apprentices working in a design setting.

The ABLE approach maps to our model of a design apprenticeship in this way (Table 1):

- The Teacher takes the role of the Master, and a well-sequenced curriculum that integrates with the ABLE approach.
- The core of the ABLE approach is a hyperlinked case base of exemplary design projects. These projects model good process. Where foreseeable, potential mistakes are pointed out with links back to the step in the project where the mistake may have been made. Multiple representations of the project are provided, i.e., students can look at a process and the products of that process in different ways. Links are created between (a) concrete projects and (b) the abstract concepts and procedures which are exemplified in the projects
- The ABLE case library is meant to be used in conjunction with a computer-supported collaborative learning environment that supports the peer-to-peer and student-to-master interaction that is an important component of an apprenticeship. We currently use CaMILE [Guzdial, 1997 #727; Guzdial, 1996 #697] as a forum for students to communicate during the apprenticeship. It is also explicitly linked to the ABLE case library to provide (a) a sense of the community in which the original project was written and (b) a discussion area for talking about the cases themselves.
- The ABLE approach has mostly been used when students are engaged in design activities.

Apprenticeship Model	ABLE Approach
A Master who provides authentic tasks, and	Teacher who encourages use of an

	apprenticeship approach with a curriculum that takes advantage of ABLE case base.
Scaffolds the process	A case base of projects offering modeling and coaching, integration between conceptual and process knowledge, multiple representations, and libraries of generic procedures.
In a community of apprentices	Linked to a collaborative environment that is integrated into case-base and course.
Working in a design setting.	Working in a design setting.

Table 1: Comparing Apprenticeship Model and ABLE Approach

In an ABLE projects case base, each project is broken into a hierarchy of steps.

Surrounding those steps are a variety of resources (Figure 1):

- Links to representations of the process (e.g., a picture of the step hierarchy) and of the product (i.e., a picture or diagram representing the artifact being designed in this process).
- Links to the CaMILE forum of where the designers who did this project discussed it, and a link to a CaMILE forum where current ABLE users might talk about the project.
- Links to more abstract information, like related concepts and generic forms of steps.

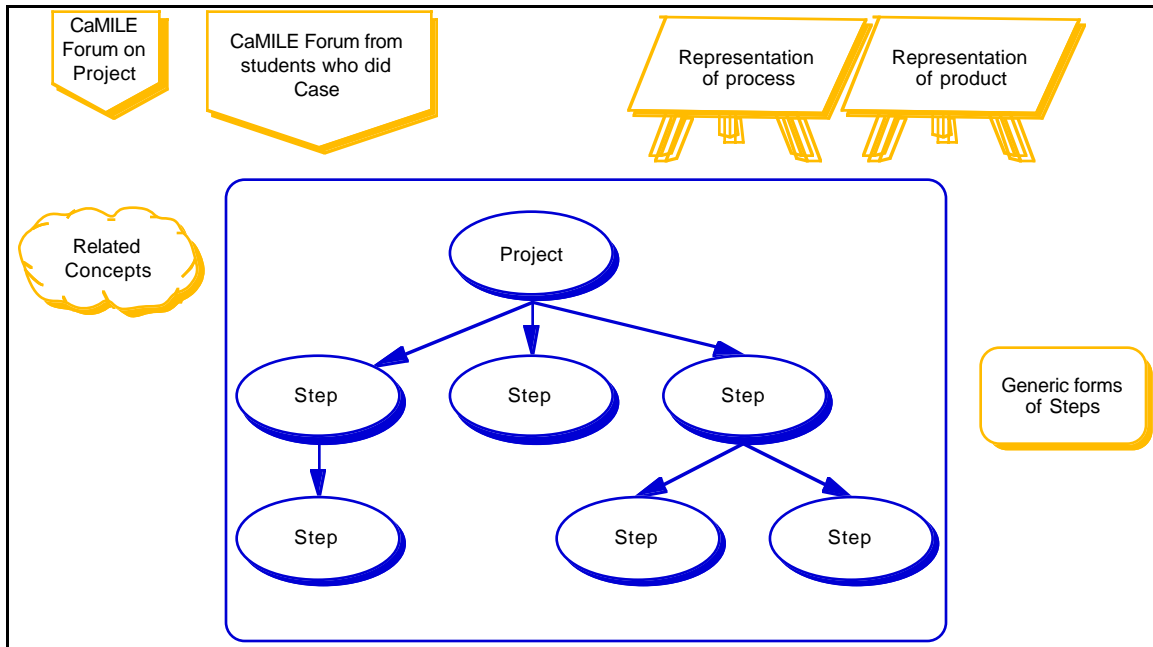


Figure 1: Structure of ABLE project cases

The contrast between the ABLE approach and other technology-based approaches to supporting the learning of skilled practice emphasizes a loose, inexpensive, and flexible support.

- An ABLE approach cannot provide the detail and quality of feedback that an ITS can. It is simply a hypermedia, with no attempt to model the student, nor the task (Bareiss & Osgood, 1993), nor adapt the presentation for the student or the task (Hekmatpour, 1995). Not having such models reduces the cost of an ABLE. Instead, the ABLE approach relies on having example projects that relate to students' design efforts. We have typically gathered projects for ABLE's from high-ability students. Gathering cases from a more senior peer group helps to make the cases more accessible: The context in which the case was built is familiar, and the language in which the original student expressed the case is comfortable to case-using student. Feedback comes from the student's own efforts and is facilitated by (a) easy comparison between the cases and the student's work and (b) comparison

with others in the collaborative forum. But because no models of the student are created, the ABLE project cases can support a higher level of complexity than a traditional ITS or GBS approach – a level of complexity more in line with the work of Sophomores in Computer Science or other intermediate undergraduate students.

- An ABLE does not attempt to track student progress, unlike either an ITS or a GBS. It makes no judgement of success or failure. An ABLE case library can contain scenarios about successes or failures, but it is up to the student to recognize the value or applicability of the scenario.
- An ABLE case library is similar to a DSE component library. The critical difference is that an ABLE case library is a collection of whole projects, not individual components. While identifying useful components is hard, identifying good student work is relatively easy. In a single term of a class with 75 students working on three design projects each, nearly a dozen good student projects were identified for our first ABLE.
- Unlike a DSE, an ABLE does not attempt to change the process nor modify any external tool. Instead, the ABLE case library describes in great detail how the process actually unfolded, including how to use whatever design tools may be involved. While this is not the best approach if the design tools are very difficult to use, our tradeoff is supporting learning versus changing the task being learned. This is similar to the tradeoff taken in the minimal manual approach, which was able to gain rapid student learning and performance (Carroll, Smith-Kerker, Ford, & Mazur, 1986).

The key to the ABLE approach is the design of the case library, since it is the main source for scaffolding. We structured the case library to follow principles of supporting learning that we have derived from research in the educational technology, cognitive science, and education communities. Below are the key principles of our approach.

Principle I: Provide adaptable scaffolding through levels of detail

Scaffolding, in a sense, is about “how much of the answer you give away.” There is no single “answer” in a design context, but the issue in a design learning context is how much support to provide to a student and how much to ask the student to do herself. Scaffolding is inherently about the tradeoff in task responsibility between the master and the apprentice (Rogoff, 1990). The more of the task that is the responsibility of the apprentice, the more the scaffolding is said to have “faded.” The scaffolding provided in an ABLE is about process and content. The question is how much process and content knowledge to “give away” to the student, and how much to hold back. A skilled teacher provides enough to support student’s success without impinging on the student’s learning, that is, enough to remain in the Zone of Proximal Development, which is that range of activity where a student is challenged (but not overwhelmed) and can succeed with help (Rogoff, 1990).

The fading of scaffolding is more complicated to do in software-realized scaffolding. It is difficult for the software to sense how much a student does or does not know. Computer scientists refer to software that senses the users needs and automatically changes to meet those needs as being *adaptive* (Neal, 1991; Oppermann & Simm, 1994). The option that we prefer is having the software being *adaptable*—having the student change how the system supports her activity.

There have been several examples of software-realized scaffolding in educational technology, e.g., (Carroll & Carrithers, 1984a; Carroll & Carrithers, 1984b; Merrill & Reiser, 1993), but few have actually supported adaptive or adaptable scaffolding. Emile (Guzdial, 1995) did provide an adaptable form of software-realized scaffolding for constructing physics simulations. Students could turn on or off various scaffolds such as guides for the students’ process, enforced prompts for articulation (such as predictions about the outcomes of physics simulation experiments), and supports for entering in

program code. One of the observations from the Emile experience was that students who attempted their task with the least amount of scaffolding seemed to learn the most. This may be just a correlation, not a causal relationship. Our conjecture is that students should be provided with as little support as possible, but with the option of adapting their scaffolding for more support.

Our first principle for an ABLE case library is to present adaptable scaffolding through multiple levels of detail, with the least amount of detail offered first (Figure 2). A student might ask for more detail on a given step, thus adapting the scaffolding to increase the amount of support provided. We have not developed a firm principle for how many levels ought to be provided. In our original ABLE implementation, we provided three levels of scaffolding (least detail, more detail, most or all detail), and our current implementations sometimes offer four.

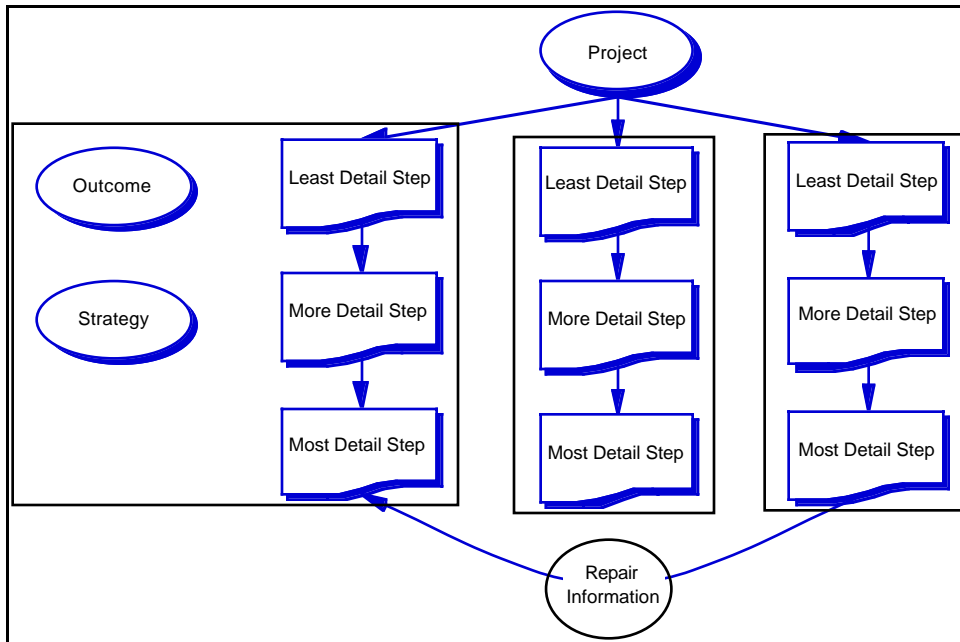


Figure 2: Scaffolded steps in ABLE project case library

Principle II: Strategy information is available but not immediate

Michael Redmond's thesis work on CELIA was an explicit attempt to model an ideal apprentice (Kolodner, 1993; Redmond, 1992). CELIA developed explicit causal models of the domain and of processes in the domain. The below quote is from Kolodner's description of CELIA:

Celia is an *active intentional learner*. As it watches and listens to a teacher performing some troubleshooting task, it *predicts* what the teacher will do or say next. It then *observes* or listens to the teacher. When predictions don't match what the teacher says or does, it attempts to explain the discrepancy and learn from it. If it can't explain a discrepancy, CELIA sets itself up with a goal of acquiring whatever knowledge it thinks is necessary to explain it. (p. 127)

The idea for ABLE's is to be support for such an apprentice, with the working hypothesis that students are or can become an ideal apprentice as Redmond described. An ABLE should provide the opportunities to learn that the CELIA model tells us that an apprentice is looking for.

- *Watching the teacher perform some task.* The ABLE case projects represent and serve as a presentation of a task description.
- *Predicting what the teacher will do or say next.* ABLE's present the task description at varying levels of detail and completeness. When a task step (*process step* below) is presented to the student (as the teacher/master/expert performed the step), it is first presented at a high level of detail (e.g., "Create a button that can be dragged around"). As the student requests, more detail can be provided, but by *not* offering it immediately, the student is encouraged to predict and fill in the blanks for how and why the teacher did a particular step.

- *Attempts to explain the discrepancy:* ABLE's have a great deal of information about the task: explanations, steps, outcome checks, repair procedures, and related concepts (arranged using a cognitive media types structure (Ram, Recker, & Stasko, 1995)). Thus, ABLE's can provide a lot of suggestions (outcome checks and repair procedures), explanations, and related information to help a student understand a discrepancy.

The second principle of ABLE's is that strategy information should not be immediately accessible. If requested (e.g., through a click), strategy information should be available to help students understand why a step was undertaken. But by initially hiding the strategy information, students are given the opportunity to think of their own strategy for the given step.

Principle III: Outcome information is available but not immediate

Marcia Linn and her colleagues on the Computer as Lab Partner project have shown the important benefits of prediction in science learning (Lewis & Linn, 1992; Linn & Songer, 1991). In their research, students used simulation to explore questions about thermodynamics. But before running the simulation, students were asked to make a prediction about what they thought was going to happen in the simulation. By making a prediction, students articulate their conceptualizations, which may not be accurate with respect to the simulation and the view of scientists. Thus, there is a greater potential for creating the kind of impasse which may lead to addressing the misconception and the learning of a new, more robust conceptualization (Schank et al., 1994).

The outcome of a step, that is, what the student should see if the step were to be executed, is similar to the result of a simulation. An expert practitioner should be able to predict the outcome of a step. Students need to be provided the opportunity to make that prediction.

The third principle of ABLE's is that the outcome of a step should not be immediately available. By pushing it one click away, the user is provided the opportunity to predict the outcome and compare the prediction to the recorded one.

Principle IV: Where possible, suggest potential problems and solutions

One of the ways in which model-tracing tutors improve the efficiency of learning by (a) creating a model of possible errors and misconceptions then (b) tracking the user's performance against this model (Anderson et al., 1995). Whenever the user varies from the path, the student's error is pointed out, and the student is provided an opportunity to correct the error. Anderson and his colleagues have explored a range of variations on when the student is alerted to the error. In general, identifying errors and encouraging correction led to more learning in less time.

The model-tracing approach is difficult to use in the broad class of problems where (a) multiple paths and solutions are appropriate and (b) not all student errors or failures are predictable or identifiable. Most software design problems fit into this class, for example. On simple program assignments, the majority of student errors are predictable and detectable, but not all. The semantic error checker Proust achieved approximately 85% accuracy identifying errors on a small program to compute an average of a set of positive integers (Johnson & Soloway, 1985). On larger problems, errors are much more difficult to detect.

The fourth principle of ABLE's is to recognize the difficulty of identifying problems and solutions in all domains, but wherever possible, common problems and solutions should be identified. The benefits in learning efficiency are clearly demonstrated.

Principle V: Use multiple representations

There is an enormous literature on how to create a good representation of information (Tufte, 1983), how we understand representations of information (Kosslyn, 1989), and how to get computers to generate good representations of information (Mackinlay, 1986). There is now a small but growing body of literature that shows the benefits of *multiple* representations for information. For example, Kozma's studies of expert chemists show that they frequently move between multiple representations in multiple media: Talking about balance equations at one point, gesturing in the air about molecular structure at another, and pointing to spectrometer readings at a third (Kozma, Chin, Russell, & Marx, 1997). There is reason to believe that use of multiple and linked (i.e., parts of one representation map directly to parts of another) representations is an important part of understanding complex domains (Spiro, Feltovich, Jacobson, & Coulson, 1991).

Further, there is evidence that multiple linked representations can play an important role in the success of students undertaking complex activities. Multiple, linked representations in chemistry (e.g., equilibrium equations, spectrographic information, molecular models) can help students to more robust understanding of equilibrium that can be applied in chemical experimentation (Kozma, Russell, Jones, Marx, & Davis, 1996). The GPCeditor was a programming and design environment organized around multiple, linked representations (e.g., program code, structure diagrams, lists of variables and other components), and it was shown to be successful in helping students to solve more and more complex programs than is typical in a first programming course in high school (Guzdial, Konneman, Walton, Hohmann, & Soloway, 1998).

The fifth principle of ABLE's is to use multiple, linked representations to help describe the project. Through use of multiple representations, the student can be shown different

aspects of the case. By linking the representations, the student can be led to see connections in the different aspects of the same case.

Principle VI: Design for use in practice

One of the most important aspects of apprenticeship is that the learning in an apprenticeship was contextualized or situated. The learning of concepts and skills that made up practice occurred (a) in a place where the concepts and skills were being used and (b) as the student tried to apply the concepts and skills in real practice (Lave & Wenger, 1991). There are several benefits to this situatedness. From a situated learning perspective, our intelligent behavior is deeply related to the spaces in which we practice and the tools which we use in our practice. By separating learning from the contexts in which it will be used, schools do students a disservice by making it more challenging to apply the learning. From a more traditional cognitive science perspective, transfer of learning from one context to another is facilitated when there are cues and reminders between the two contexts that support the transfer (Bruer, 1993). By placing the learning of design in the context of doing design, the opportunity for students is increased to note where the learning applies and to see a need for the learning in their practice.

Practically, providing an ABLE for use during practice can be a challenging problem. For computer-based tasks using some tool (e.g., programming, designing using a computer-aided design tool, etc.), the challenge is to provide the ABLE hypermedia on the same screen at the same time that a student is undertaking the task. A greater challenge is to somehow make the computer tool and the ABLE aware of one another, so that case issues might be presented that are relevant to the current tool activity. For non-computer-based tasks (e.g., cooking, manipulating the glassware of a chemistry experiment, etc.), the challenge is to make the ABLE hypermedia available in the context at all, let alone having the ABLE be aware of the activity outside of the computer.

We have not yet attempted an ABLE implementation outside of a traditional desktop computer, but we have been preparing our tools to address this challenge. Our ABLE implementations thus far have been generated from a database of case elements. The database is then traversed by an ABLE Generator, which creates the hypermedia for the user. Currently, our ABLE Generators produce HTML for providing an ABLE on the Web. We envision producing ABLE Generators that produce, say, the DOC format which is available on Palm Pilot and other handheld or palmtop devices².

The sixth principle of ABLE's is to make feasible the use of the case material while the student is undertaking the project. By integrating the ABLE with the task context, we create a better opportunity for transferable learning.

Principle VII: Support the sense of community

Lave and Wenger have pointed out that an important aspect of an apprenticeship is the integration of the student into the community of practitioners (Lave & Wenger, 1991). Much of what an apprentice is learning is how practitioners in the community think, approach problems, and conduct their practice. Thus, a critical aspect of an apprenticeship, and perhaps any form of higher education (Brown & Duguid, 1996), is the association with a community and the interaction with the community. While students sometimes *do* create a community around a class and a set of assignments, not all students may be involved, or may feel part of the community, or may be sharing with the community (Newstetter & Hmelo, 1996). There is a role for support to create a student community.

² <http://www.memoware.com>

Further, there is benefit for more senior members of the community to work with the younger members. The claim that “one of the best ways to learn is through teaching” is both often heard and is well-supported in the research literature, e.g., (Harel, 1988; Palincsar & Brown, 1984). It is beneficial for the more expert members of the community to learn through articulating their understanding for the more novice members of the community. The mutual benefit of this kind of interaction is one of the hallmarks of an apprenticeship community.

Creating community interaction within hypermedia calls for a multi-strategy approach. Where possible, we connect the case material with the authors who originally wrote the case, e.g., by pointing out the author’s names, the dates when the project was undertaken, the transcript of design discussions which surrounded the creation of this case. Perhaps surprisingly, such a simple-minded approach can work surprisingly well. Students in different undergraduate cohorts know one another, and we have heard stories of students approaching older students whose cases were visited by the younger students. While recognizing a community with older students is not the same as establishing a community with professional practitioners, we believe that this is a first step toward an apprenticeship style community. While the masters are not present (other than through the teachers and teaching assistants), the younger apprentices are learning from the older apprentices, which is part of the model. We also integrate our ABLE with computer-supported collaborative learning (CSCL) tools, e.g., CaMILE (Guzdial et al., 1996; Guzdial & Turns, 1998; Hmelo, Guzdial, & Turns, 1998), so that students can discuss the cases and other projects with their current peers.

The seventh principle of ABLE’s is to support the sense of community in which the cases are placed. The goal is to integrate the students into a community of practitioners, and to

provide both more senior and more junior students the benefits of an apprenticeship community.

STABLE: An Instance of ABLE

STABLE (SmallTalk ABLE) is our first instance of an ABLE³. It was created to support a course at Georgia Institute of Technology's College of Computing, *CS 2390 Modeling and Design*. The ten week course focuses on the object-oriented paradigm of software construction. Course topics include object-oriented analysis, design, and programming, as well as simulation and graphical user interfaces (which were among the first application domains of the object-oriented approach). The programming language Smalltalk is the focus of the course, though the programming language C++ is typically introduced near the end of the course as a contrast. Smalltalk is an older language with great expressive power (Kay, 1993), but it can be challenging for students to learn, e.g., (Carroll & Rosson, 1991; Carroll, Singer, Bellamy, & Alpert, 1990). Students in CS 2390 at Georgia Tech are faced with learning both a new programming paradigm as well as learning a complex programming language. STABLE was seen as a means of easing the difficulties in the course by providing models of both the object-oriented analysis and design process, and also the ways in which Smalltalk was used in object-oriented programming.

The cases for STABLE were gathered during the Winter 1996 quarter, which we will hereafter refer to as the Case Gathering Offering of the course. The teaching assistants who did most of the grading in the course were asked to help with the collection of cases. They were asked:

- To identify the best student work on each of the assignments in the course, and
- To note the most common mistakes on those assignments.

³ http://www.cc.gatech.edu/classes/cs2390_96_spring/stable/stable.html

We then asked each student if they would be willing to let us use their project as a case in our library. If students agreed, we asked them for (a) their own description of their process as it appeared in their assignment documentation and (b) a copy of their program. These materials were then broken into individual components and loaded into a database.

Wherever possible, the students' own words were used, so that the language and level of discussion would be appropriate for future students in the class. We generated new information for concepts, levels of detail of steps, and other cross-project information which no student would have created in their descriptions. A Generator gathered the case components from the database and created HTML pages with appropriate linkages between the pages.

STABLE now consists of some 13 cases and typically is generated as some 1200 web pages. Over offerings of the course, different user interfaces have been tried, which involved modifying the Generator and recreating the Web pages. Cases have been added and some mistakes fixed over time.

In the rest of this section, STABLE is described with sample screenshots. The organization of this section mirrors the previous section. STABLE is introduced in the ways in which it meets the seven principles of an ABLE.

Provide adaptable scaffolding through levels of detail

A STABLE case is created as a hierarchy of steps, where each step can be displayed at multiple levels of detail. Figure 1 presents a single step from one of the cases available in STABLE. The sample step is the "Least Detail" version of the step. The user is basically told that, in this step, the "DivisorCount class" (a software construct in Smalltalk) is built. The student is not told what the DivisorCount class consists of, nor what the software

construct looks like. But if the student already understands much of the detail of this project, more detail may not be necessary.

If the student wishes “More Detail,” she may click on that button and receive a new page which describes the components of a DivisorCount class (Figure 2). Again, this may be enough for a more learned student to gain the information needed from this step of the class. However, if the full detail is desired, the student may ask for another level of detail, where the detailed description of the software associated with the DivisorCount class is presented (Figure 3).

There are additional perspectives on detail that STABLE provides to the student. For example, at the bottom of Figure 1 is a link to “Concepts Related to this Step.” A concept page describes the abstract lesson that this step is an example of. A concept page also links to all other steps that exemplify this concept. In this sample step, the concept is “What is a part-whole relationship?” which is a term used for describing how different software classes are related. The “What is a part-whole relationship?” page links to all pages throughout STABLE where a part-whole relationship is described or implemented. A concept that applies to many steps is an abstract level of detail which is available to the student, as the student chooses.

On other pages, “Concepts Related to this Step” are related to more concrete programming concepts, such as how to enter code into Smalltalk and how to find useful software constructs in Smalltalk. Thus, the “Concepts” links presented both high-level, abstract issues, and also practical how-to information, as a master does in face-to-face apprenticeships (Schon, 1985).

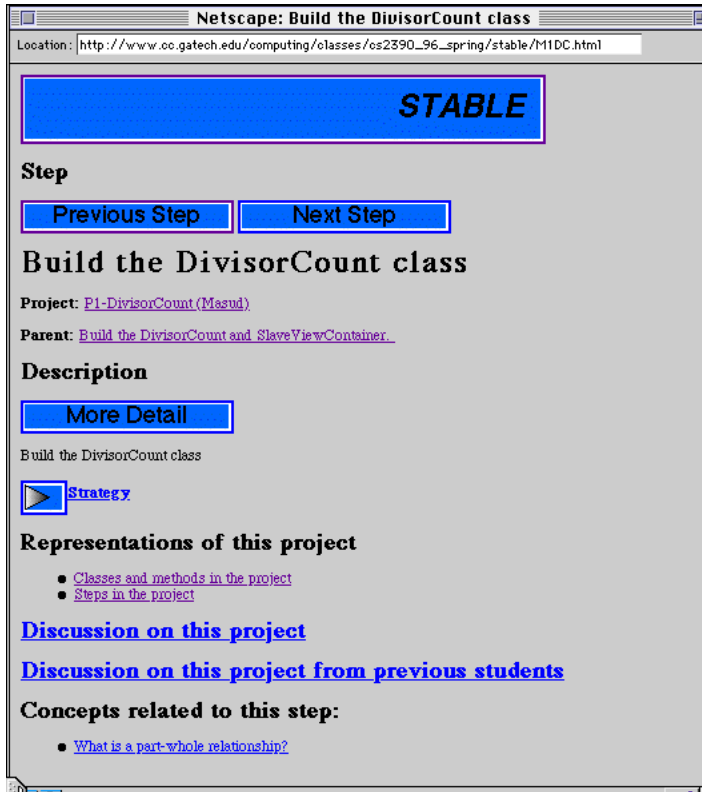


Figure 1: A Sample Step “Least Detail” Page in STABLE

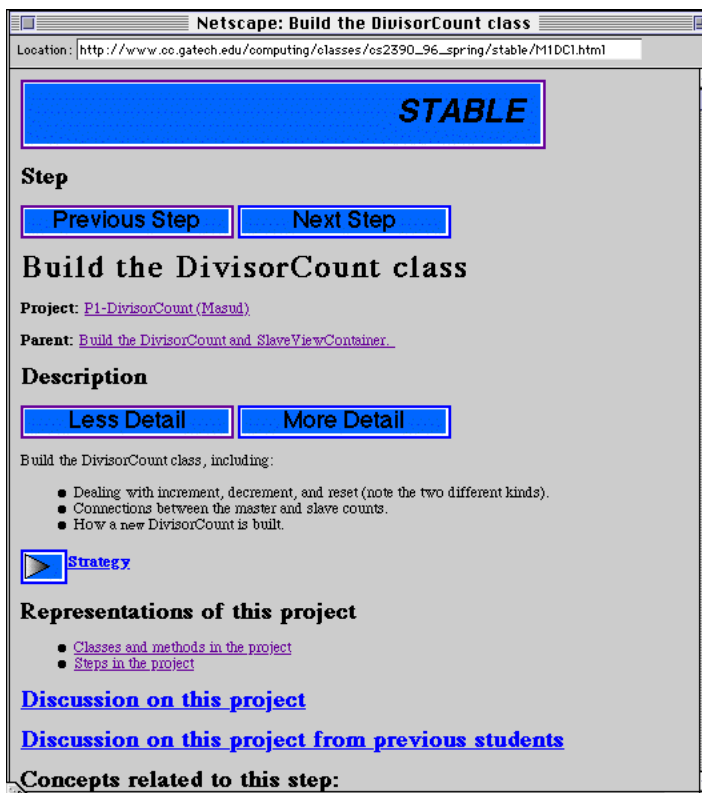


Figure 2: “More Detail” Version of Sample Step

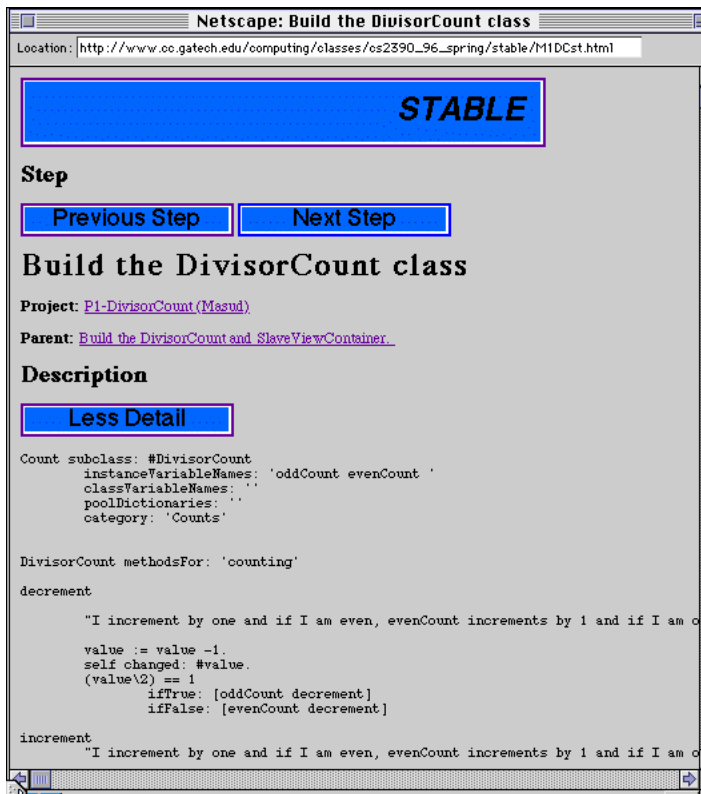


Figure 3: “Most Detail” Version of Sample Step

Strategy information is available but not immediate

STABLE uses user interface clues to convey that strategy information is available, is relevant and is integrated into this page, but is not immediately visible. In Figure 1, we see a link for Strategy information. The use of the right-turned arrow next to the Strategy link is chosen to convey to students the existence of additional information, like the arrows used in the List view in the Apple Macintosh Finder. When the link is selected, the page is redrawn with the arrow pointing down and the strategy information appearing next to the Strategy heading. Thus, when the strategy information is present, it appears integrated into the text of the step.

Outcome information is available but not immediate

Outcome information was presented with the same kind of user interface structure: an up-turned arrow to indicate the presence of additional information, and the integrated view with a down-turned arrow (Figure 4). Outcome information presented clues as to how the student could tell if the step was completed correctly. Not all steps contained Outcome links (as the above did not). For many steps in a process, especially cognitive ones (such as deciding on a name for a software class, or writing the code for a class), there may not be a visible outcome to check for success. However, when Outcome links are possible, they sometimes include figures to show how the screen should look or what the design representation might look like.

Where possible, suggest potential problems and solutions

If an Outcome is available, it is possible for students who are duplicating the step (or the entire process) to tell immediately if the step is unsuccessful. At that point, the student could use Repair Information (Figure 4): Suggestions of possible problems and solutions.

Repair information in STABLE is a set of possible symptoms and the step in which the problem that related to that symptom probably occurred (Figure 5). We explicitly chose not to state what the mistake was in duplicating the process or step, nor do we explicitly tell students how to make the repair. Rather, we identify common mistakes (based on what other students in the Case Gathering Offering of the course faced) and where those mistakes commonly were made. Our goal is to make the learning process more efficient, but the philosophy behind STABLE is still constructivist in that we believe that informing the students of their mistake would be removing the opportunity for the students to learn their own debugging and repair skills.

We also chose not to attempt greater integration with the students' design environment, Smalltalk. There were two large challenges that made the integration complicated. The first is the challenge of recognizing what the student is doing in the general Smalltalk programming environment and providing reasonable help at the time. It would be difficult to distinguish an innovative solution from a completely wrong one (Johnson & Soloway, 1985). The second challenge is the technical one of getting Smalltalk to communicate with a hypermedia database. While we can integrate a Web browser with other desktop applications (Guzdial, 1997), it is difficult to do this on the multiple platforms that students use at Georgia Tech.



Figure 4: A Step with Outcome and Repair Information

STABLE

Repair Information

Repair Suggestions

If you find this symptom, **then** check this step:

- **IF** Complaint that time did not know how to + **THEN** [Where time is initialized: Create the OceanQueue class and its methods](#)
- **IF** Complaint that did not know how to add: **THEN** [Where occupants is set to a new OrderedCollection: Create the Ocean class and its methods](#)
- **IF** Syntax errors or complaint that Ocean does not know if something is food **THEN** [Where the reference to ocean is made: Create the mainline \(System Workspace\) code that will run the system](#)

Hints: Check the code carefully. Use inspectors on anything that the system says does not understand something -- the inspector will tell you what class something is.

Figure 5: Repair Information Page

Use multiple representations

Most cases in STABLE have at least two graphical representations associated with them.

- One representation is the hierarchy of steps into which the process has been decomposed (Figure 6). This representation is generated during the process of creating the case, not by the student originator of the case. The process representation is an active user interface element. Clicking on any step in the process representation will navigate directly to the step page (least detail first).
- A second representation is a depiction of the software product, as a diagram in a standard object-oriented notation used in the class. This is not a screenshot or other representation of running the student's software, but is instead a depiction of the focus of the software design and implementation process—that is, the software structure itself. While the product representations are sometimes produced during creation of the case, more often the originating students' actual diagrams (sometimes hand-drawn as in Figure 7, sometimes machine generated) are used for this representation. The software product representations are also active user interface elements. Clicking on any structure in the diagram navigates to a new page that lists all the steps in which that structure is defined or modified.

In some cases, a third representation of the case is presented. Typically, this is a second representation of the software product emphasizing different aspects of the software structure. For example, one software product representation may describe the static structure of the software, while another may describe the interaction between software components during execution of the software.

All of these representations are linked in that each representation serves as a navigation structure into the process via the individual step pages. The software process representation is clearly indicating steps in the process. The software product representation indicates pieces of the software which emerge from analysis, are defined in design, and are implemented in the programming phase of the process. The analysis, design, and programming subprocesses are depicted as one or more steps in the process, so it is relatively easy to point out the exact steps where the software components are referenced.

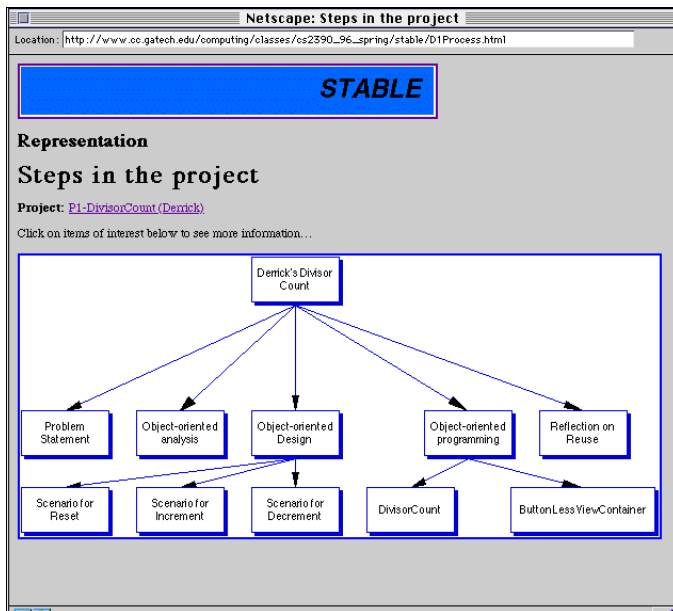


Figure 6: Representation of Hierarchy of Steps in Project

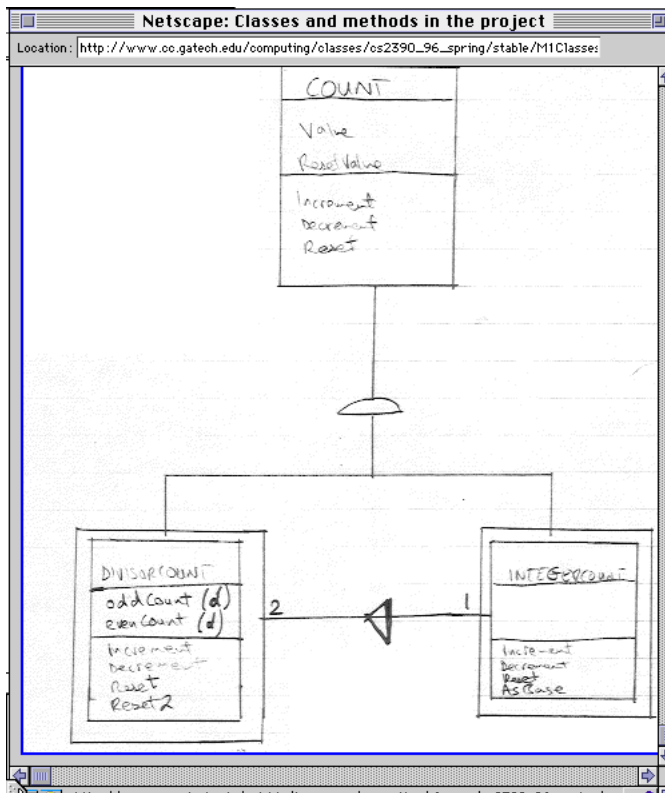


Figure 7: Representation of Product in a Project

Design for use in practice

STABLE is served to students as a set of HTML pages, which can be viewed with any HTML 2.0 or better web browser. We do not use advanced HTML features (such as Java applets, JavaScript, nor XML), and we have only recently started using frames in STABLE. Because STABLE is designed for low-powered browsers, there is little impediment to opening a Web browser on STABLE while the student is also using the Smalltalk programming environment. Technologically, there is nothing to prevent the use of STABLE alongside real practice in the course.

Further, the material in STABLE is organized to encourage use in practice. Students are not just permitted, but actively encouraged to reuse anything that they find in STABLE. Thus,

it is not at all unusual for students to copy source code out of a STABLE project case and simply paste it into their Smalltalk code.

We have not attempted any kind of deep integration between STABLE and Smalltalk, though it is an area of active interest for us. For example, it should be possible for some interface between STABLE and the Smalltalk environment to suggest project cases that are relevant to the students' current work, based on pattern matching with the kinds of software components that the student is currently using. Integration that actively connects the student's work to STABLE would make it more valuable for students to use STABLE while engaged in practice.

Support the sense of community

We use several strategies to situate the project cases within a community of practice. As can be seen in Figure 1 (under the Project name), each case is named for the student who wrote the case. The originating student's full name appears on the opening page for every project case. Students have told us that they do start using cases first that are originated by peer students whom they recognize. Simply naming does play an important role in situating the project case in the community.

An important part of establishing a sense of community is our use of a Web-based computer-supported collaboration tool called CaMILE (Guzdial et al., 1997; Guzdial & Turns, 1998). CaMILE allows for the creation of hyperlinks to individual notes or threads of discussion. (Technically, each note has its own unique URL.) We have used CaMILE for classroom discussions in this course for several years (Guzdial, 1998). We integrate CaMILE with STABLE in two ways:

- Each case is linked to the CaMILE discussion space where the originating author and her peers were discussing this project (bottom of Figure 1). In this way, new

students can review the capture of the design issues that were discussed at the time that the original case was written.

- Each case is also linked to a CaMILE discussion space set aside for comments on the STABLE cases themselves. Students are encouraged to talk about the usefulness of the case (or lack thereof). For example, in actual use, students will point out bugs in the case's code or issues that were not well resolved in the case (e.g., input conditions that the project did not deal with).

Formative Evaluation of STABLE

We conducted a formative evaluation of STABLE during the Spring 1996 quarter, which we will refer to as the STABLE Evaluation offering of the course. A variety of measurements were taken during this STABLE Evaluation quarter to get a sense of whether STABLE was being successful in supporting the students' performance and learning as well as whether the design of STABLE was appropriate. Where a comparison group was useful, the students in the STABLE Evaluation offering was compared with the Case Gathering offering students. We hesitate to call this a "control" group because we took no measurements to help us control for variance in student abilities or other variables between the two course offerings. Nevertheless, we believe that the assumption that consecutive offerings of a large course (approximately 75 students each offering) are largely alike is a reasonable assumption for a formative evaluation.

At issue during the STABLE Evaluation quarter was more than simply the tool itself. We recognized that how STABLE was integrated into the course was a significant aspect of the evaluation. Poor integration might mean that STABLE would not get used by the students, or would be used improperly (e.g., outside of the context of actual practice). Our first subsection below describes how we integrated STABLE into the CS2390 course

curriculum. The subsections beyond address the research questions of this formative evaluation.

- Couldn't we just put all the cases in a book?
- Do the students use STABLE?
- Do the students perform better with STABLE?
- Do the students learn better with STABLE?
- Do the students like STABLE?

Integrating STABLE in a Curriculum

Most offerings of CS2390 at Georgia Tech have a similar structure.

- There are twice weekly 90 minute lecture and discussion periods.
- Students are required to complete eight laboratory assignments, which are typically completed in a computer cluster with a Teaching Assistant available to answer questions.
- Concurrently, students are required to complete four sizable homework assignments. The first is an analysis and design problem with no programming component. The second through fourth homework assignments typically involve programming in Smalltalk.
- There are midterm and final examinations.

During the Case Gathering offering, these were instantiated with fairly standard projects:

- The first laboratory assignment was a simple exercise in programming called the "Muppet" lab in which students create objects to represent the muppets, Kermit and Oscar.
- The second and third laboratory assignments were on Smalltalk programming, while students concurrently attempted their first homework assignment (an elevator control system for a Mile High Skyscraper).

- The midterm then followed these assignments.
- The first programming assignment was to do a modification of an example discussed in class: To create a “DivisorCount” class.
- The second programming assignment was a simple spreadsheet mechanism. Students were asked to implement some of the internals of the spreadsheet, but no formulas and no user interface.
- The third programming assignment (and last for the class) was a simulation of bus routes around an Olympic Village.

Eight cases were gathered from this set of assignments. (An additional four were added from laboratory assignments and lecture examples.) Two different solutions to the DivisorCount problem were placed into STABLE, and three different solutions of each of the spreadsheet and bus route simulation problem were also added into STABLE. These cases were available to students during the STABLE Evaluation offering.

The curricular goals for the STABLE Evaluation quarter were to (a) encourage students to make use of STABLE and (b) gain the benefits of STABLE in terms of supporting student performance and learning. The standard CS2390 curriculum was modified in these ways.

- The first lab was again the Muppet lab, but this time, the Muppet lab was already described in STABLE. Students were asked to follow the process and replicate it in Smalltalk.
- The second lab was for students to replicate either of the DivisorCount project cases, but to review both. During the following lecture period, a discussion was encouraged comparing the two project cases.
- The third lab was a programming assignment for which there was a relevant STABLE case to draw upon and reuse components from.

- During the second and third labs, students were working on their first assignment which was to extend and improve a design of a project case from STABLE.
- The first programming assignment was to extend and generalize the spreadsheet problem which had been the second programming assignment in the Case Gathering offering. Specifically, students were to add the ability to evaluate formulas in spreadsheet cells. Students were not required to use the three spreadsheet problem STABLE cases, though it was certainly in their best interest to do so.
- The second programming assignment was a simulation of a factory floor, which had some elements in common with the bus route simulation. Note that, again, students in the STABLE Evaluation offering were attempting a problem similar to one that was in the Case Gathering offering, but earlier in the quarter.
- The third programming assignment was completely different than any in the Case Gathering offering: An implementation of a Smalltalk example from the text, but in C++.

Our hope with this structure was that the students would use STABLE, would recognize its value, and would willingly use it for their programming assignments. By using these on their programming assignments, we hoped that they would be able to complete the problems successfully, despite the fact that these were more complicated problems than the students in the Case Gathering offering had and that they were offered earlier in the course.

Couldn't we just put this all in a book?

Our first formative evaluation question was whether the implementation we had generated for STABLE really made sense. An obvious alternative implementation would be to have all the same pieces (e.g., steps at different levels, strategy information, outcome information, repair information, etc.) but in a static paper book. Was it really better to have

the information initially hidden as we had it implemented? Further, a book may be more costly or more difficult to peruse, but perhaps integrating the information into a single, long page-per-case on the Web would be easier for students than all the clicking necessary to walk the case in the hyperlinked structure.

To answer this question, we conducted a comparative evaluation during the first laboratory assignment in the course, where students were attempting to replicate the Muppets project from STABLE. We re-generated STABLE where each case (all steps, all relevant concepts, all repair information) was presented in a single, long Web page. We then printed several of these pages (some five cases, including the Muppets case on top).

Each of the three sections of the course were then given a different version of the STABLE case material.

- Section A was given the Web address (URL) to the hyperlinked version of STABLE, described in the previous section.
- Section B was given the Web address (URL) to the “flat” one-page-per-case version of STABLE.
- Each student in Section C received an inch thick pile of paper containing the printed version of the “flat” cases that Section B had access to.

We did not take any measurements of whether students in Section C accessed the Web-based STABLE, but we did ask students in Section A and B not to share their URLs.

We used three different outcome measures for this simple comparison.

- We counted the number of steps completed in the final program that the students turned in, out of the seven steps in our decomposition of the Muppets project.
- We asked students what they thought of STABLE. We asked them to rate how well they liked STABLE, on a scale from one (“not at all”) to five (“very much”).

- Finally, we administered a post-test in lecture on the day after the laboratory. There were seven questions on the post-test, where only three were relevant to the laboratory. The other four questions were on material available in the STABLE case base, but not directly appearing in the Muppets project case. We were wondering if the hyperlinked students would explore some of the other concept and project information. The post-test was simply graded correct or not for each problem.

The results of the evaluation of the three different kinds of STABLE appears in Table 1. In general, the students using the hyperlinked STABLE completed more of the problem, liked STABLE the most, and performed best on the post-test. A t-test on the post-test results between Section A and Section C were significant at the $p < 0.10$ level ($p = 0.06$). While these results are clearly formative, they do suggest that our system design was not a bad one—it did not seem to be hindering student performance or learning, and could actually be enhancing those impacts.

	Steps completed on problem (out of 7)	How well students liked STABLE (1 to 5)	Number right on post-test (out of 7)
Section A: Linked STABLE (n=27)	2.6 (stdev 1.4)	3.4 (0.9)	5.7 (1.4)
Section B: Flat STABLE (n=24)	2.0 (1.5)	3.2 (0.9)	5.5 (1.5)
Section C: Flat Paper STABLE (n=26)	1.7 (1.5)	2.9 (0.9)	5.3 (1.7)

Table 1: Results of STABLE Structure Evaluation

Do the students use STABLE?

Other than on the first two laboratory assignments, we did not require use of STABLE in the course. Use was entirely voluntary. The obvious question was *whether* students did actually use STABLE after those first two labs. Further, we were interested to see *how*

they used STABLE: When it was used, whether different kinds of pages got different kinds of visits (e.g., less detail as time went on), etc.

We have done detailed analyses of the log files, that is, the recordings from the Web server hosting STABLE of when STABLE was “hit.” As with any Web server recordings, the numbers do not actually reflect the number of page visits. If a page is visited once in a session, visiting the same page again typically does not generate a new request to the Web server as the request is already cached by the user’s browser on her disk. Nonetheless, such measurements do give a snapshot of the kind of use and when it occurred.

Figure 8 is a graph showing the number of page “hits” (a visit to the page that is recorded on the Web server) over the dates of the STABLE Evaluation course. We have also added markers at the 1000 hits level for when labs occurred and when STABLE-relevant homework assignments were due in the course. (Recall that the last homework assignment was in C++, not in Smalltalk, and there were no C++ cases in STABLE.)

In general, the answer to the question is that students made extensive use of STABLE even after the first two laboratory assignments. There are significant peaks of STABLE use during the third, fifth, sixth, and seventh laboratory assignments. These peaks are on the order of hundreds of page hits, which are unlikely to be generated by only a few students. More likely, use of STABLE was distributed across the class. There is also a peak around the homework assignment due just before the eighth laboratory assignment, which suggests that students did use STABLE for homework as well as laboratory assignments. It is also notable that there is STABLE usage several days before the dates of a laboratory or homework assignment due date. While due dates drove usage, usage occurred outside of the laboratory sessions, presumably from computer labs or home computers where the students could get Web access. We note relatively little STABLE use during the end of the

quarter (when students were working on their C++ homework assignment), though there are some peaks at the very end of the quarter which may suggest the use of STABLE in studying for the final exam.

We also note that students did not seem to use STABLE differently over the course of the quarter. We were somewhat surprised at this finding. We anticipated relatively little use of the “Most Detail” steps as students became more expert in Smalltalk and thus were more capable of anticipating the details of the steps. In our results, however, we have found that the relative ratio of least to most detail visits remained relatively constant throughout the STABLE Evaluation offering.

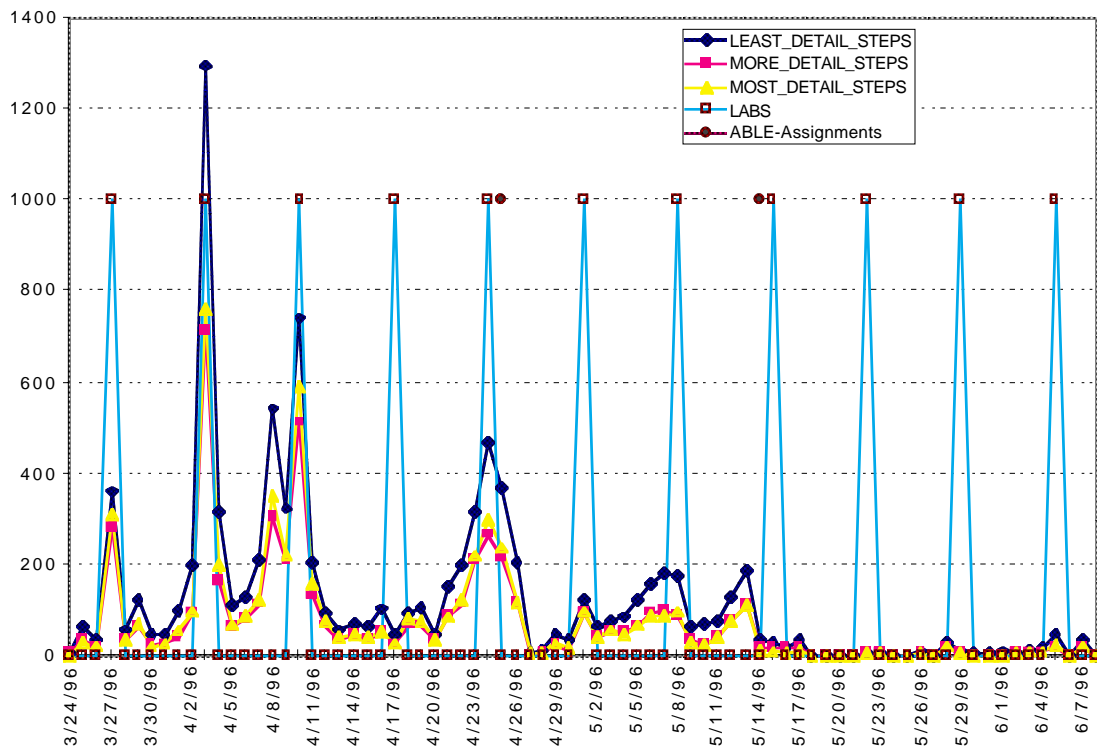


Figure 8: Graph of “Hits” within STABLE over the STABLE Evaluation Offering of the Class

Do the students perform better with STABLE?

While Figure 8 suggests that students will actually use STABLE, there was still the question of whether STABLE would help them. We might have made STABLE too confusing to be of significant use to the students, or we may have put the wrong kinds of information in STABLE. In any case, we wanted a measure of whether STABLE actually supported student performance.

Recall that the STABLE Evaluation students were assigned as their first homework assignment a similar problem to one that the Case Gathering students had for their second homework assignment. If students were using STABLE successfully to improve performance, one would expect that the STABLE Evaluation students would be at least as successful at their problem (using the same grading criteria and same graders), since they would be building on the Case Gathering students' work (even though they were getting a problem of greater complexity earlier in the quarter). However, if they were not using STABLE, one would expect lower performance due to the complexity of the assignment.

The results in Table 2 suggest that students using STABLE performed with a better overall grade than did the original Case-Gathering class. Using mostly the same graders (two of the three Teaching Assistants were the same in each quarter) in both quarters and essentially the same grading criteria, the students in the STABLE Evaluation Quarter had a higher average grade and a lower standard deviation. A t-test comparing the two classes was significant at $p < 0.05$ ($p = 0.03$).

Comparing grades is not a robust experimental measure. What these formative results do suggest is that STABLE was not hurting performance and may have been enhancing performance. These results provide evidence that students were doing more than simply

browsing STABLE. They were using it to enable them to perform complex problems successfully.

	Case Gathering Offering	STABLE Evaluation Offering
Grades: Case Gathering Second Assignment vs. STABLE Evaluation First Assignment	75.6 (stdev 31.8)	85.8 (18.3)

Table 2: Performance Comparison between Case Gathering class and STABLE Evaluation offering

Do the students learn better with STABLE?

An important question is whether students actually learned with STABLE, or preferably, learned better than students who went without STABLE. There is ample evidence in the research on project-based learning that it is certainly possible to improve students' performance on projects while keeping them too busy to do the reflection which may lead to learning (Turns, Guzdial, Mistree, Allen, & Rosen, 1995). Use of STABLE, even successful use of STABLE, does not automatically imply learning.

To gain some measure of learning, we used several isomorphs of problems which had been on the Case Gathering class' final examination when creating the final examination for the STABLE Evaluation class. An isomorph of a problem is a similar but not identical problem, where the main difference is in details rather than concepts. For example, one problem was to repair a badly designed hierarchy of software components (called "classes"). The original problem involved repair to a Payroll program class hierarchy with several problems, such as TemporaryWorkers who had an hourly wage mistakenly also receiving a salary because they were misclassified as being a kind of SalariedWorker. The transformed problem was set in a bookstore where new items, such as audio CD's, accidentally had page numbers because they were defined as being a kind of Book.

In all, four isomorphic problems appeared on the STABLE Evaluation offering final examination.

- One was the design problem isomorph described above.
- Another was on C++. Since STABLE actually did not contain any cases from C++, this problem served as a kind of control. If the STABLE Evaluation students did better on the C++ problem as well as the others, it may indicate that the STABLE Evaluation students had were more high-ability, and that results had nothing to do with the tool itself.
- A third problem was on Simulations.
- A fourth problem was on the user interface programming structures used in Smalltalk.

Again, since most of the Teaching Assistants were the same, the graders were mostly the same. I used exactly the same grading criteria for both final examinations, since the problems differed only in details. I also compared each problem using a t-test. The results appear in Table 3.

- Students in the STABLE Evaluation class did significantly better than the Case Gathering class on the Design problem. Even if STABLE was badly designed, one would hope that students looking over many more detailed cases than they might normally see in a traditional textbook would improve their performance on design problems.
- Students in the STABLE Evaluation class did worse on C++ problems than the Case Gathering class, significantly at the $p < 0.10$ level. Since students in general did well on this problem (97% and 93%), it is probably not a practically significant difference. Though it may be that the emphasis on STABLE led to a de-emphasis

on C++, the results suggest that the STABLE Evaluation class was no more gifted than the Case Gathering class.

- Students in the STABLE Evaluation class did better, but not significantly, on the Simulation and User Interface Programming problems.

Again, these are formative results with many caveats: We're using grades to compare the two classes, who are probably not comparable, and we're comparing on isomorphic problems which may differ accidentally in some important issue (e.g., students may understand bookstore issues better than payroll issues). The results are promising, however, and suggest that STABLE may be improving learning in the areas that it addresses.

	Case Gathering Final Examination Results	STABLE Evaluation Final Examination Results
Design Problem	0.83 (stdev 0.21)	0.91 (0.14) $p=0.02$
C++ Problem	0.97 (0.10)	0.93 (0.12) $p=0.06$
Simulations Problem	0.88 (0.17)	0.91 (0.16) $p=0.35$
User Interface Programming Problem	0.68 (0.42)	0.74 (0.31) $p=0.43$

Table 3: Results on Isomorphic Problems on Final Examinations in Case Gathering and STABLE Evaluation classes.

Do the students like STABLE?

Finally, we were interested in whether the students liked STABLE. Certainly their frequent use of STABLE suggested that they liked it well enough to continue using it, but use is not really a measure of user satisfaction. They may have simply had no better alternative.

We used a multiple-strategy approach to finding out how students liked STABLE. We measured student satisfaction with surveys and with interviews with the students. We also

kept track of quotes in CaMILE for comments on STABLE and its usefulness (or lack thereof).

The survey data was not overwhelmingly positive. Students generally agreed that STABLE was useful: average 1.25 on a scale of 1 to 3 where 1 was agreement with the statement “The information I found [on STABLE] was useful” and 3 was disagreement. Less strongly they agreed that “I learned more because of the information that was in STABLE” (average 1.7). They were neutral (average approximately 2) on the statement “It was easy to find the information I needed.”

Interviews with the students supported some of our concerns. Students complained that “STABLE was the only place to find out some stuff.” They reported frequently being lost in STABLE with all the links, and that the second level of detail was “useless.” They found the class and the cases difficult, but valuable. They said that it was difficult to find the pieces that they wanted in STABLE. Students reported seeing “other” students simply copying and pasting code without truly understanding it. But they also reported themselves reviewing code in STABLE and using STABLE to inform their own work rather than wholesale copying. Students did not find the use of CaMILE from within STABLE useful at all, neither to look through previous discussions nor to talk about STABLE cases.

However, students did use CaMILE to talk about their current projects with other students, and they did discuss STABLE and their concerns about it. Perhaps most damning and most articulate were the unsolicited quotes from students when STABLE was discussed in CaMILE.

I don't know if anyone else feels this way, but I think that STABLE is not organized very well. The overabundance of links on each and every page makes it difficult to find the information that one is looking for.

... I tried to find out exactly what the point of the project was. I read the overview of the project. Hmm, it didn't really tell me what the final product should do. Instead, it's broken down into step to show me how to accomplish the task which would take forever to go through just to figure out what the whole point was. A one shot, here it is, page that lays out the problem and the constraints would make STABLE a lot easier to use.

STABLE has also become a crutch. There are 3 detail levels loosely categorized as such:

- 1) Can be done by a Smalltalk master, i.e. one who's worked with it for about a year.
- 2) Can be done by someone who has a solid exposure to the language, say, has passed this class with a good grade.
- 3) Giving all the source code needed that can be filled in, or cut and pasted into the right place if someone has some clue about how Smalltalk works.

This is useless. If there was a category between 2 and 3, something along the lines of pointers to useful functions in Smalltalk, ideas about how blocks might be set up, and some pointers on interesting problems of syntax, now that would be extraordinarily useful.

The final result was that, no, students did not like STABLE very much. They found it useful, even necessary at times. Further, there were concerns that STABLE made the class too easy in some ways.

Discussion: Further Use, Emerging Principles, and Moving Beyond Software

STABLE and ABLE's in general have continued to evolve since the formative evaluation. We have not yet conducted a summative evaluation of STABLE because we have considered the basic design not yet fully successful given student dissatisfaction with it.

Meanwhile, the results at the formative evaluation stage were successful enough that we have moved on to begin development of an ABLE in a new domain, chemical engineering.

Changes During Further Use

In the last two years, we have continued to use STABLE in CS2390 which is offered three times a year at Georgia Tech. We have made several changes to address concerns raised in the formative evaluation.

The first set of changes was to the curriculum. Our initial strategy was to begin with assignments for which STABLE was an enormous help, perhaps even necessary, and slowly move to assignments that stood apart more from STABLE's content. We decided that the general structure of the strategy was good, but the speed of separation needed to be greater. Students needed to feel that STABLE was a resource, but not a crutch. Copying STABLE examples alone could not solve the assignments.

In successive offerings of the course, the early assignments are still informed by STABLE, but as early as the second laboratory assignment (second week of the class) students are writing significant pieces of Smalltalk code that is unlike anything in STABLE. Students no longer complain that STABLE is a crutch, and students do say that they use STABLE to look over designs even without using any of the programming code in a project.

The second set of changes have been addressed to the user interface and structure problems in STABLE. We have tried two different approaches to address those problems:

- First, we have modified the Generator to produce frames-based pages. The page is split into two parts: The top 2/3 presents essentially the same page as before, and the bottom 1/3 presents specific information. Requesting more detail on a step page results in a change in the bottom frame, rather than sending the user to an entirely

new page. Similarly, strategy and outcome information is presented in the lower frame, rather than causing a new page to load.

- Second, additional levels have been added between “More Detail” and “Most Detail.” For example, on all pages where the “Most Detail” is program code, there is an intervening level called “Outline Detail” where the code is described in outline form.

While these changes have improved user satisfaction with STABLE, satisfaction is still relatively low. Students still complain that they can’t find the information that they want and that they get lost. While we still believe that our approach is well-founded and has successful results, we recognize that there must be an additional design principle to be addressed that will help assure student satisfaction.

An Eighth Principle

The HCI community has come to understand the need for emergent principles in software design. No matter how carefully and how principled the design is, the results are not what was originally predicted. The complexity is that a problem or user situation is analyzed in terms of the way it was before the software arrived. When the software is available, we are in a new problem or situation—the software changes things, and new kinds of software can create new problems and new situations.

This kind of shift has occurred in our understanding of student use of STABLE. Through a great deal of log file analysis, we finally found what we believe to be the cause of user dissatisfaction. We analyzed use of STABLE at the level of individual sessions with STABLE: From the first page that they access, to the last page accessed by a single student from a single computer within minutes of the last page accessed. First, we found that students access STABLE at very large volumes—typically, 30 pages or more per session.

If we discount sessions where less than 5 pages are visited (as being an unusually brief visit to find a known piece of information), we find that students visit an average of 1.62 *different* cases in a single session.

We realized that the structure of STABLE was entirely wrong for a very frequent activity of students. Students are very often trying to *compare* different cases. STABLE, on the other hand, is designed to support detailed exploration of different cases. Follow up interviews with students have borne out this hypothesis. Students do not want to just understand one way of doing something. They want to see different ways of doing the same thing. In order to compare two different cases, the student has to walk the hierarchy of steps for one case, then walk another hierarchy of steps for another case. It is no wonder that students visit so many pages and are so frustrated with STABLE.

While useful in its explanatory power, this finding is also confounding for the design of the tool. It is difficult to determine *a priori* what aspects of two cases that students will want to compare. Students may want to compare the similar parts of two simulations, but they may also want to compare how a spreadsheet user interface is implemented and how a simulation user interface is implemented. We have not yet implemented an interface that provides this facility.

We now add a new, eighth principle of ABLE's: Provide supports to enable students to navigate the hypermedia in a comfortable manner. The challenge in applying this principle is that it is not an initial design principle but an *iterative* design principle. We simply could not have known that students would want to compare different cases before we built STABLE because, before STABLE, students did not have access to a large corpus of cases to compare!

Creating an ABLE for Chemical Engineering

Working with Chemical Engineering faculty at Georgia Tech, Matthew Realff and Pete Ludovice, we have begun creating CABLE, a Chemical engineering ABLE. CABLE presents projects in analyzing chemical engineering problems and solving them with computer-based numeric methods. CABLE was designed and implemented by J. Nicole Pinkerton.

One of the challenges in implementing CABLE was to determine a stepwise decomposition of a chemical engineering design project. In object-oriented programming, there is a natural decomposition: Analysis precedes design which precedes programming. Since programs are naturally hierarchically structured, the higher-level stages of analysis, design, and programming can be easily further decomposed into steps based on the program structure. Such a well-defined structure does not exist for chemical engineering.

We decided to use the structure of the modeling equations that chemical engineers use for describing their systems as the structure for the decomposition. For example, an equation that models the friction between a chemical and a pumping system involves terms for the friction between the chemical and the pipes, the chemical and the valve, and the chemical and the pump itself. Each of these terms can be further decomposed and solved. Similar decompositions based on models of pumping system structures can be created for other aspects of problems to add to CABLE.

Only three cases have been added to CABLE thus far. We have conducted pilot tests with three undergraduate chemical engineers using CABLE to solve complex problems. They found CABLE useful and easy to navigate. However, because we only have a few cases in the system thus far, we cannot yet apply the eighth principle of ABLE's since we don't know how students will use the system when they have access to a larger number of cases.

Conclusions

Our experiences with ABLE's, and STABLE in particular, lead us to several conclusions about the design of hypermedia, about producing ABLE's as software-realized scaffolding, and about STABLE itself.

- A principled approach to hypermedia design leads to a good deal of success with even the first iteration of a design. We are pleased that STABLE served students well in terms of learning and performance in even the first term of use.
- ABLE's do work well as software-realized scaffolding, but do not provide all the support that a real apprenticeship involves. ABLE's are static and cannot respond to specific questions about specific problems. We now see ABLE's as one part of a larger structure where there is support for various aspects of an apprenticeship. For example, an ABLE cannot take the place of students looking at one another's designs and talking about them.
- While we are very pleased with our formative evaluation results with STABLE, we are also careful to not underestimate the complexity of designing hypermedia like STABLE. While STABLE does seem to support student learning and performance, students do not see STABLE as a success. Dealing with the emergent issues of a new kind of technology like STABLE require careful analysis and multiple iterations to produce a truly successful tool from the perspective of all stakeholders.

In general, we consider ABLE's as a structure and a set of principles to be promising. The limited results of our formative evaluation suggest that this approach may have great power to support student performance and learning. We continue to explore ABLE's as a way of improving how students learn process.

Acknowledgements

The Educational Technology Research Group at Georgia Tech's College of Computing has been a frequent source of insight and feedback during our work with ABLE's, especially from Janet Kolodner and Michael McCracken. Thanks also to the other CS2390 instructors for their support and encouragement: Rich LeBlanc, Gregory Abowd, Colin Potts, and Noel Rappin. Funding for this work has come from the National Science Foundation grants RED-9550458 and CDA-9414227. This material is based upon work supported under a National Science Foundation Graduate Fellowship.

References

- Anderson, J. R., Boyle, C. F., Corbett, A. T., & Lewis, M. W. (1990). Cognitive modeling intelligent tutoring. *Artificial Intelligence*, 42, 7-49.
- Anderson, J. R., Boyle, C. F., & Reiser, B. J. (1985). Intelligent tutoring systems. *Science*, 228, 456-462.
- Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill acquisition and the LISP tutor. *Cognitive Science*, 13, 467-505.
- Anderson, J. R., Corbett, A. T., Koedinger, K. R., & Pelletier, R. (1995). Cognitive Tutors: Lessons Learned. *Journal of the Learning Sciences*, 4(2), 167-208.
- Augustine, N., & Vest, C. M. (Eds.). (1994). *Engineering Education for a Changing World*: American Society for Engineering Education.
- Bareiss, R., & Osgood, R. (1993). Applying AI models to the design of exploratory hypermedia systems, *Proceedings of the ACM Conference on Hypertext* (pp. 94-105).
- Blumenfeld, P. C., Soloway, E., Marx, R. W., Krajcik, J. S., Guzdial, M., & Palincsar, A. (1991). Motivating project-based learning: Sustaining the doing, supporting the learning. *Educational Psychologist*, 26(3 & 4), 369-398.

- Bottino, R. M., Chiappini, G., & Ferrari, P. L. (1994). A hypermedia system for interactive problem solving in arithmetic. *Journal of Educational Multimedia and Hypermedia*, 3(3/4), 302-326.
- Brown, J. S., & Duguid, P. (1996). Universities in the Digital Age. *Change*, July/August, 11-19.
- Bruer, J. T. (1993). *Schools for Thought: A Science of Learning in the Classroom*. Cambridge, MA: MIT Press.
- Carroll, J. M., & Carrithers, C. (1984a). Blocking learner error states in a training-wheels system. *Human Factors*, 26(4), 377-389.
- Carroll, J. M., & Carrithers, C. (1984b). Training wheels in a user interface. *Communications of the ACM*, 27(8), 800-806.
- Carroll, J. M., & Rosson, M. B. (1991). Deliberated evolution: Stalking the View Matcher in design space. *Human-Computer Interaction*, 6, 281-318.
- Carroll, J. M., Singer, J. A., Bellamy, R. K. E., & Alpert, S. R. (1990). A View Matcher for learning Smalltalk. In J. C. Chew & J. Whiteside (Eds.), *Proceedings of CHI'90: Human Factors in Computing Systems* (Vol. Seattle, April 1-5, pp. 431-437). New York: ACM Press.
- Carroll, J. M., Smith-Kerker, P. L., Ford, J. R., & Mazur, S. A. (1986). *The minimal manual* (Computer Science/Cognition): IBM Thomas J. Watson Research Center.
- Carver, S. M., Lehrer, R., Connell, T., & Erickson, J. (1992). Learning by hypermedia design: Issues of assessment and implementation. *Educational Psychologist*, 27(3), 385-404.
- Coleman, R. J. (1996). The Engineering Education Coalitions. *ASEE Prism*(September), 24-31.

- Collins, A. (1990). Cognitive apprenticeship and instructional technology. In B. F. Jones & L. Idol (Eds.), *Dimensions of Thinking and Cognitive Instruction* (pp. 121-138). Hillsdale, NJ: Erlbaum and Associates.
- Collins, A., Brown, J. S., & Newman, S. E. (1989). Cognitive apprenticeship: Teaching the craft of reading, writing, and mathematics. In L. B. Resnick (Ed.), *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser* (pp. 453-494). Hillsdale, NJ: Lawrence Erlbaum and Associates.
- Dixon, J. R. (1991). New goals for engineering education. *Mechanical Engineering*(March), 56-62.
- Domeshek, E. A., & Kolodner, J. L. (1992). A case-based design aid for architecture. In J. Gero (Ed.), *Proceedings of the Second International Conference on Artificial Intelligence and Design* .
- Farnham-Diggory, S. (1990). *Schooling*. Cambridge, MA: Harvard University Press.
- Guzdial, M. (1995). Software-realized scaffolding to facilitate programming for science learning. *Interactive Learning Environments*, 4(1), 1-44.
- Guzdial, M. (1997). A shared command line in a virtual space: The WorkingMan's MOO, *UIST'97 Conference Proceedings* (pp. 73-74). Banff, Alberta, Canada: ACM.
- Guzdial, M. (1998). Technological support for apprenticeship, *Proceedings of WebNet 1998* (pp. In press). Orlando, FL: American Association for Computers in Education.
- Guzdial, M., Hmelo, C., Hübscher, R., Nagel, K., Newstetter, W., Puntembakar, S., Shabo, A., Turns, J., & Kolodner, J. L. (1997). Integrating and Guiding Collaboration: Lessons learned in computer-supported collaboration learning research at Georgia Tech. In R. Hall, N. Miyake, & N. Enyedy (Eds.), *Proceedings of Computer-Supported Collaborative Learning'97* (pp. 91-100). Toronto, Ontario, CANADA.

Guzdial, M., Kolodner, J. L., Hmelo, C., Narayanan, H., Carlson, D., Rappin, N., Hübscher, R., Turns, J., & Newstetter, W. (1996). Computer support for learning through complex problem-solving. *Communications of the ACM*, 39(4), 43-45.

Guzdial, M., Konneman, M., Walton, C., Hohmann, L., & Soloway, E. (1998). Layering scaffolding and CAD on an integrated workbench: An effective design approach for project-based learning support. *Interactive Learning Environments*, *In press*.

Guzdial, M., & Turns, J. (1998). Supporting sustained discussion in computer-supported collaborative learning: The role of anchored collaboration. *Journal of the Learning Sciences*(Submitted).

Guzdial, M., Weingrad, P., Boyle, R., & Soloway, E. (1992). Design support environment for end users. In B. A. Myers (Ed.), *Languages for developing user interfaces* (pp. 57-78). Boston, MA: Jones and Bartlett.

Harel, I. (1988). *Software design for learning: Children's construction of meaning for fractions and LOGO programming*. Unpublished Ph.D. Dissertation, MIT.

Hekmatpour, A. (1995). An adaptive presentation model for hypermedia information systems. *Journal of Educational Multimedia and Hypermedia*, 4(2/3), 211-238.

Hmelo, C. E., & Guzdial, M. (1996,). *Of Black and Glass Boxes: Scaffolding for Doing and Learning*. Paper presented at the International Conference of the Learning Sciences, Evanston, IL.

Hmelo, C. E., Guzdial, M., & Turns, J. (1998). Computer-support for collaborative learning: Learning to Support Student Engagement. *Journal of Interactive Learning Research*, *In press*.

Hmelo, C. E., Holton, D. L., Allen, J. K., & Kolodner, J. L. (1997). Designing for understanding: Children's models of lungs, *To appear in Proceedings of the Nineteenth Annual Meeting of the Cognitive Science Society* . Mahwah, NJ: LEA.

- Hohmann, L., Guzdial, M., & Soloway, E. (1992). SODA: A computer-aided design environment for the doing and learning of software design, *Computer assisted learning: 4th international conference, ICCAL '92 proceedings* (pp. 307-319). Berlin: Springer-Verlag.
- Hsi, S., & Agogino, A. (1994). The impact and instructional benefit of using multimedia case studies to teach engineering design. *Journal of Educational Multimedia and Hypermedia*, 34(3/4), 351-376.
- Jackson, S. L., Stratford, S. J., Krajcik, J., & Soloway, E. (1995). Learner-centered software design to support students building models : Presented at the Annual Meeting of the American Educational Research Association.
- Johnson, W. L., & Soloway, E. (1985). PROUST: An automatic debugger for Pascal programs. *BYTE*, 10(4), 179-190.
- Kay, A. C. (1993). The early history of Smalltalk. In J. E. Sammet (Ed.), *History of Programming Languages (HOPL-II)* (pp. 69-95). New York: ACM.
- Koedinger, K. R., & Sucker, E. L. F. (1996, 1996). *PAT goes to college: Evaluating a cognitive tutor for developmental mathematics*. Paper presented at the International Conference on the Learning Sciences, Northwestern University.
- Kolodner, J. (1993). *Case Based Reasoning*. San Mateo, CA: Morgan Kaufmann Publishers.
- Kolodner, J. (1995,). *Design Education Across the Disciplines*. Paper presented at the Second Congress on Computing in Civil Engineering, Atlanta, GA.
- Kolodner, J. L., Hmelo, C. E., & Narayanan, N. H. (1996, 1996). *Problem-based learning meets case-based reasoning*. Paper presented at the International Conference on the Learning Sciences, Northwestern University.
- Kosslyn, S. (1989). Understanding charts and graphs. *Applied Cognitive Psychology*, 3, 185-226.

Kozma, R., Chin, E., Russell, J., & Marx, N. (1997). *The roles of representations and tools in the chemistry laboratory* (Technical Report): SRI International, Menlo Park, CA.

Kozma, R. B., Russell, J., Jones, T., Marx, N., & Davis, J. (1996). The use of multiple, linked representations to facilitate science understanding. In S. Vosniadou, R. Glaser, E. DeCorte, & H. Mandel (Eds.), *International perspective on the psychological foundations of technology-based learning environments* (Vol. 41-60,). Hillsdale, NJ: Erlbaum.

Krajcik, J. S., Blumenfeld, P., Soloway, E., & Marx, R. (1992). Enhancing the teaching of project-based science : Proposal to the National Science Foundation program in Teacher Preparation and Enhancement.

Krajcik, J. S., Blumenfeld, P. C., Marx, R. W., & Soloway, E. (1994). A collaborative model for helping teachers learn project-based instruction. *Elementary School Journal*, 94(5), 483-497.

Lave, J., & Wenger, E. (1991). *Situated Learning: Legitimate Peripheral Participation*. Cambridge, UK: Cambridge University Press.

Lehrer, R. (1992). Authors of knowledge: Patterns of hypermedia design. In S. Lajoie & S. Derry (Eds.), *Computers as Cognitive Tools* (pp. 197-227). Hillsdale, NJ: Lawrence Erlbaum Associates.

Lewis, E. L., & Linn, M. C. (1992). Heat energy and temperature concepts of adolescents, naive adults, and experts: Implications for curricular improvements : Paper presented at the National Association for Research in Science Teaching Annual Meeting, San Francisco, CA.

Linn, M. C., & Songer, N. B. (1991). Cognitive research and instruction: Incorporating technology into the science curriculum : Paper presented at the American Educational Research Association meeting.

- Mackinlay, J. (1986). Automating the design of graphical presentations of relational information. *ACM Transactions on Graphics*, 5(2), 110-141.
- Merrill, D. C., & Reiser, B. J. (1993,). *Scaffolding the acquisition of complex skills with reasoning-congruent learning environments*. Paper presented at the Workshop in Graphical Representations, Reasoning and Communication from the World Conference on Artificial Intelligence in Education (AI-ED'93), The University of Edinburgh.
- Merrill, D. C., Reiser, B. J., Beekelaar, R., & Hamid, A. (1992). Making processes visible: Scaffolding learning with reasoning-congruent representations, *Intelligence Tutoring Systems: Second International Conference, ITS'92* (pp. 103-110). New York: Springer-Verlag.
- Neal, L. (1991). User modeling and tailorable, adaptable, and adaptive systems : Tutorial presented at CHI'91.
- Newstetter, W. C., & Hmelo, C. E. (1996). Distributing cognition or how they don't: An investigation of student collaborative learning. In D. C. Edelson & E. A. Domeshek (Eds.), *Proceedings of ICLS'96* (pp. 462-467). Evanston, IL: AACE.
- Oppermann, R., & Simm, H. (1994). Adaptability: User-initiated individualization. In R. Oppermann (Ed.), *Adaptive User Support: Ergonomic Design of Manually and Automatically Adaptable Software* (pp. 14-66). Hillsdale, NJ: Lawrence Erlbaum Associates.
- Palincsar, A. S. (1986). The role of dialogue in providing scaffolded instruction. *Educational Psychologist*, 21(1-2), 73-98.
- Palincsar, A. S., & Brown, A. L. (1984). Reciprocal teaching of comprehension-fostering and monitoring activities. *Cognition and Instruction*, 1, 117-175.
- Puntambekar, S., & Kolodner, J. (1997). Learning by Design: Developing children's understanding of science by engaging them in solving design problems : Symposium

presented at the American Educational Research Association Annual Meeting, San Diego, CA. April.

Ram, A., Recker, M., & Stasko, J. (1995). Cognitive Media Types. *Journal of Educational Multimedia and Hypermedia*, *In press*.

Redmond, M. (1992). *Learning by Observing and Understanding Expert Problem Solving*. Unpublished Unpublished Ph.D. Thesis, College of Computing, Georgia Institute of Technology.

Rogoff, B. (1990). *Apprenticeship in thinking: Cognitive development in social context*. New York: Oxford University Press.

Schank, R. C. (1992). *Goal-based scenarios* (Technical Report): The Institute for the Learning Science, Northwestern University.

Schank, R. C., Fano, A., Bell, B., & Jona, M. (1994). The design of goal-based scenarios. *Journal of the Learning Sciences*, *3*(4), 305-346.

Schofield, J. W., Britt, C. L., & Eurich-Fulcer, R. (1991). Computer-Tutors and Teachers: The Impact of Computer-Tutors on Classroom Social Processes : Paper presented at the annual meeting of the American Educational Research Association, Chicago, IL.

Schon, D. (1985). *The Design Studio: An Exploration of its Traditions and Potential*. London: RIBA Publications Limited.

Schon, D. A. (1982). *The Reflective Practitioner: How Professionals Think In Action*. New York: Basic Books.

Soloway, E., Guzdial, M., Brade, K., Hohmann, L., Tabak, I., Weingrad, P., & Blumenfeld, P. (1993). Technological support for the learning and doing of design. In M. Jones & P. H. Winne (Eds.), *Foundations and frontiers of adaptive learning environments* (pp. 173-200). New York: Springer-Verlag.

Soloway, E., Guzdial, M., & Hay, K. E. (1994). Learner-centered design: The challenge for HCI in the 21st century. *Interactions*, 1(2), 36-48.

Spiro, R. J., Coulson, R. L., Feltovich, P. J., & Anderson, D. K. (1988). Cognitive flexibility theory: Advanced knowledge acquisitions in ill-structured domains, *The Tenth Annual Conference of the Cognitive Science Society* (Vol. August, Montreal, pp. 375-383). Hillsdale, NJ: Lawrence Erlbaum Associates.

Spiro, R. J., Feltovich, P. J., Jacobson, M. J., & Coulson, R. L. (1991). Cognitive flexibility, constructivism, and hypertext: Random access instruction for advanced knowledge acquisition in ill-structured domains. *Educational Technology*, 31(5), 24-33.

Spohrer, J. C. (1989). *MARCEL: A Generate-Test-and-Debug (GTD) Impasse/Repair Model of Student Programmers*. Unpublished Ph.D., Yale University.

Tufte, E. (1983). *The Visual Display of Quantitative Information*. Cheshire, CT: Graphics Press.

Turns, J., Guzdial, M., Mistree, F., Allen, J. K., & Rosen, D. (1995). I wish I had understood this at the beginning: Dilemmas in research, teaching, and the introduction of technology in engineering design courses, *Proceedings of the Frontiers in Education Conference*. Atlanta, GA.

Urdu, T., Blumenfeld, P., Brade, K., & Soloway, E. (1993). IByD: Instruction By Design. In M. Jones & P. H. Winne (Eds.), *Adaptive Learning Environments*. Berlin: Springer-Verlag.

Vanderbilt, Cognition and Technology Group at (1990). Anchored instruction and its relationship to situated cognition. *Educational Researcher*, 2-10.

Wood, D., Bruner, J. S., & Ross, G. (1975). The role of tutoring in problem-solving. *Journal of Child Psychology and Psychiatry*, 17, 89-100.