Interaction and Feedback in Automatically Assessed Algorithm Simulation Exercises

Ari Korhonen, Lauri Malmi, Jussi Nikander and Petri Tenhunen
Helsinki University of Technology, Helsinki, Finland

archie@cs.hut.fi lma@cs.hut.fi jtn@cs.hut.fi ptenhune@cs.hut.fi

Executive Summary

Feedback is an essential element of learning. Students need feedback on their work and their solutions to assignments both when they work manually and while they use a computer. A number of tools have been implemented to automatically assess and give feedback, for example, on programming exercises and algorithmic exercises. However, one problem of the provided feedback is that in most cases its scope is too narrow to support the needs of different types of learners. For example, many systems provide purely verbal feedback.

In this paper we consider how exercises with automatic feedback should be designed to support a broader scope of learners. We discuss the Felder-Silverman learning model, which we use as the framework for our discussion. The model categorizes learners with four different axes: sensing vs. intuitive learners, visual vs. verbal learners, active vs. reflective learners, and sequential vs. global learners. We discuss how all dimensions of the model can be taken into account when designing assignments and automatic feedback. Moreover, we use two modern automatic assessment systems, PILOT and TRAKLA2, as example systems to demonstrate our ideas.

We strongly believe that incorporating analysis of learners' preferences into design of courses, automatic feedback systems, and learning environments leads to better learning. As teachers, we should better support the needs of our students, and also train their skills to process information in more versatile ways.

Our discussion concentrates on algorithmic assignments. However, in the conclusion we briefly illuminate how similar approach could be used to design better assignments and feedback for programming exercises, as well.

Keywords: computer science education (CSE), algorithm animation, algorithm simulation, automatic assessment, learning models, visualization

Introduction

What I hear, I forget. *What I see, I remember.* *What I do, I understand.*
Confucius

In this paper we discuss different learning styles in the context of studying in virtual learning environments. The key idea of such environments is that learners construct their own mental models of knowledge (Norman, 1983) by studying course material, by interacting with the teacher, other learners or the system only, and by solving exercises. An essential aid for carrying out such a construction process successfully is getting feedback while the mental model is applied to solve prob-

Material published as part of this journal, either on-line or in print, is copyrighted by the publisher of the Journal of Information Technology Education. Permission to make digital or paper copy of part or all of these works for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage AND that copies 1) bear this notice in full and 2) give the full citation on the first page. It is permissible to abstract these works so long as credit is given. To copy in all other cases or to republish or to post on a server or to redistribute to lists requires specific permission and payment of a fee. Contact Editor@JITE.org to request redistribution permission.

Special Series Editor: Salvatore Valenti

lems. Based on the feedback the learner can verify the correctness of his or her mental model or tune it to better match the observations.

Feedback is a concept with a wide range of interpretations, ranging from simple confirmation of successful commands to critical comments and explanations given by an expert human tutor. In this paper, however, our focus lies on giving automatic feedback on non-trivial assignments. In many institutions basic computing courses have hundreds of students, and providing feedback on exercises that support the mental model construction is a very laborious process. Therefore many automatic assessment systems have been developed during the last 10 years to aid assessing exercises in large courses. Areas of interest include checking programming exercises (Vihtonen & Ageenko, 2002; Benford, Burke, Foxley, Gutteridge, & Zin, 1993; Jackson & Usher, 1997; Saikkonen, Malmi, & Korhonen, 2001), assessment of algorithm simulation exercises (Bridgeman, Goodrich, Kobourov, & Tamassia, 2000; Hyvönen & Malmi, 1993; Korhonen & Malmi, 2000), and analyzing object-oriented designs and flowcharts (Higgins, Symeonidis, & Tsintsifas, 2002). All the example systems can provide non-trivial specialized assignments. Therefore general purpose virtual learning environments and online training systems that can only provide structurally simple exercises such as fill-in forms or multiple choice questions are left out of the scope of this study.

Compared to human instructors, however, the feedback provided by automatic assessment systems can be very limited. In its simplest form, the feedback may include only textual descriptions. For example, in program analysis systems such as Ceilidh (Benford et al., 1993) and SchemeRobo (Saikkonen et al., 2001), the systems test submitted programs against test data, and report correct or incorrect functioning of the program. In addition, an evaluation of the program structure may follow, again in textual form. Another example is the TRAKLA system (Hyvönen & Malmi, 1993; Korhonen & Malmi, 2000) that is used for solving algorithm simulation exercises by showing how the given algorithm changes the given data structure. The system compares the solution to the correct model solution and tells how many points the student got from the exercise. Moreover, even though the students are able to simulate the algorithms in graphical form, most of the feedback received, including the model solutions, is still in textual form.

As pointed out by Felder and Silverman (1988), most people are able to grasp information in graphical form better than in textual form. Therefore in many occasions, graphical feedback supported by more advanced systems, is more useful than textual feedback. For example, the VIOPE (Vihtonen & Ageenko, 2002) system analyses C/C++/Java programs and gives graphical hints of the problematic areas in the target program. PILOT (Bridgeman et al., 2000) allows the user to simulate graph algorithms by clicking the nodes and edges on the screen. Incorrect selections are highlighted in a different color. TRAKLA2, which is the successor of TRAKLA, can provide model solutions also in a graphical form (see http://www.cs.hut.fi/Research/TRAKLA2/). The model solution for each simulation exercise can be viewed as an algorithm animation.

Different students are different types of learners, and we should design the exercises and feedback to match the variety of learners. In this paper we approach this problem using a rigorous learning model as the framework for our discussion. Felder and Silverman (1988) presented a model in which they categorized the different learning styles with expressions like visual, verbal, active, reflective, sensing, intuitive, sequential and global. We discuss each of these categories separately and point out by examples and reasoning how we could design assignments and feedback to support people with different learning styles. We limit our discussion to the context of algorithmic exercises. However, the principles covered can easily be extended to programming exercises.

As a case example we present the TRAKLA2 system, which is based on the Matrix application framework (Korhonen & Malmi, 2002). TRAKLA2 is a learning environment that supports both algorithm animation and algorithm simulation tasks where the user directly manipulates data structures through a

graphical user interface. The old TRAKLA environment has been in production use over ten years (Malmi, Korhonen, & Saikkonen, 2002). The new TRAKLA2 was launched in spring 2003 on our course of data structures and algorithms, where some 500 students used it to solve 15 different automatically assessed exercises. Both systems are capable of assessing simulation sequences and giving several different kinds of feedback for the learners. Our prior experimental studies (Korhonen, Malmi, Myllyselkä, & Scheinin, 2002a; Malmi et al., 2002) show that such learning environments based on automatically assessed exercises can be valuable tools for large-scale courses. One of the main results in the research was that there was no significant difference in learning results between groups exercising on the Web and groups solving the same exercises in a classroom situation with human tutors giving feedback. This is a significant result, since for large courses classroom learning requires a lot more resources than web-based learning environments.

As another example system we use PILOT. Even though the current system covers only graph algorithms, the system demonstrates several interesting dimensions of learning styles that are relevant here. We also discuss some programming assessment systems briefly.

We start by presenting different learning styles in the second section. In the third section we briefly analyze some of the automatic assessment systems in the light of the Felder-Silverman learning model. In the fourth section we more thoroughly discuss assignments and feedback that support different types of learners in the context of learning algorithms and data structures. The final section concludes the observations, suggests briefly how to apply our ideas to programming exercises, and points out future research in the area.

Learning Models

The purpose of learning models is to identify and classify different learning styles. A learning model can be used for designing a course to meet the needs of different students better. For example, some students prefer facts and hard data before theories. Others prefer visual information, i.e., pictures and animation, before written or spoken information. Some like individual studying and others might prefer interactive learning in groups.

Students could be categorized in numerous ways. A learning model provides a framework for categorizing different learners. The model attempts to represent the whole spectrum of actual learning styles with a specified multidimensional space with named axis. Two well-known learning models are Kolb's experiential learning (Kolb, 1984) and the Felder-Silverman learning model (Felder & Silverman, 1988).

Ideally a learning environment should support students in all categories. Therefore, we face the question of which model we should choose as the basis of our design. Fortunately this is not a real problem, because the results are very similar regardless of the model used (Felder, 1996). We have decided to use the Felder and Silverman (1988) model, since it has dimensions, which can easily be adapted to evaluating science education, including automatic assessment.

Felder - Silverman Learning Model

The Felder-Silverman model characterizes students' learning styles by using the following dimensions, each of which has two extremes.

1. Sensory vs. Intuitive, i.e., what type of information does the student preferentially perceive.

   Sensing learners are practical and oriented toward facts. They typically like straightforward things like working with details or memorizing data. Methodologically they like experimentation and problem solving by standard methods. Assignments supporting these things include simple questions, fill-in forms, multiple-choice questions or connecting related pairs. Adequate feedback of such questions is typically just correct/incorrect. All of these are easy to automate and

the key problem is designing good questions, not analyzing the answers. Experimentation, on the other hand, requires working with real gadgets or at least simulation tools, which allow the user to manipulate data in a realistic way.

Intuitive learners like conceptualization. They often prefer theories, principles, innovations, complications and grasping new concepts. All these issues include assignments, analysis and feedback of solutions such that require complicated reasoning.

2. Visual vs. Verbal, i.e., what sensory information is most effectively perceived.

   Visual learners prefer visual information like pictures before verbal information. In the other extreme *verbal learners* like written or spoken language.

   This dimension is fairly obvious for everybody but still interesting. Some students can most effectively perceive information presented in verbal form - as written text, speech, or mathematical formulas, while others are more comfortable with pictures, animations or diagrams. In automatic assessment it is desirable to have feedback designated toward both visual and verbal learners.

3. Active vs. Reflective, i.e., how does the student prefer to process information.

   Active learners are group workers who learn by trying things out and prefer continuous interaction. *Reflective learners* first think things through by themselves, i.e. they do reflective observation. The interaction should drive them to rethink their solution anew promoting their need for theoretical understanding.

   Simulation with continuous feedback fits active learners. The user can actively observe and directly manipulate the view. For example, the user can modify a data structure and the representation is changed immediately as a result. Reflective feedback, on the other hand, can be given as a summarizing analysis of simulation results. The system could also give descriptive feedback on mistakes during the simulation process.

4. Sequential vs. Global, i.e., how does the student progress towards understanding.

   Sequential learners proceed linearly with small steps. They can be supported by designing sequences of tasks, analyzing results and giving directions to the next assignments based on the results. A typical example would be a questionnaire or a set of tasks, which has many different branches depending on how the student proceeds.

   Finally, *global learners* learn holistically in large steps. They like to get a holistic view of acquired knowledge. This can be supported by open questions, creative exercises and interdisciplinary assignments. In most cases, only human tutors can manage to give appropriate feedback, although simulation tools can be of great value, since they allow exploration of complex models. The global learner can compare the overall dynamic behavior of the model with his or her own mental model.

Initially, Felder had also a fifth dimension, the inductive/deductive dimension. Inductive learners prefer presentation from the specific to the general and deductive learners prefer vice versa. He, however, decided to drop out this dimension (Felder, 2002), since most undergraduate students prefer inductive approach while teachers prefer deductive approach. If the model is used for designing courses and assignments, it should not promote this conflict.

Finally we note that all the dimensions are continua - not discrete categories. A student's preference on a given scale may also change with time and may vary from one subject matter to another.

Automatic Assessment

By automatic assessment systems we denote software that is capable of analyzing and giving meaningful feedback on complicated static or dynamic structures or sequences of operations. We deliberately distinguish algorithmic exercises from programming exercises. The goal of programming exercises is to teach the fundamentals of programming in general or on a specific programming language. Data structures and algorithms, on the other hand, are generally higher-level concepts than most concepts related to introductory programming. Thus, it is possible to design algorithmic exercises that work on a conceptual level, manipulating concepts instead of any specific implementation. We call such exercises algorithmic exercises. They may, of course, include pseudo code or real code as an element, but this is not necessary.

We will briefly introduce programming exercises; however, the main focus of this paper is on algorithmic exercises discussed after the programming exercises.

Assessing Programming Exercises

In many institutions introductory programming classes are very large with hundreds of students. It is thus obvious that a number of systems have been developed to aid the grading process and give feedback for the students. Such systems are, for example, Ceilidh (Benford et al., 1993), its later version CourseMaster (Higgins et al., 2002), ASSYST (Jackson & Usher, 1997), SchemeRobo (Saikkonen et al., 2001), and the VIOPE system (Vihtonen & Ageenko, 2002). ASSYST is a tutor's tool to aid the grading process. It does not give direct feedback for the student, and is therefore left out of this discussion.

Ceilidh is able to analyze programs written in different languages. It executes the target program with test cases and compares the program output to model output. The student receives information about the tests passed, and number of points received. Additional language dependent program style and structural analyses can be added. The CourseMaster is a successor of Ceilidh. It has options to analyze also other kinds of objects than programs, such as flowcharts, OO designs and logical circuits. The feedback given, however, is quite similar to that of Ceilidh: points, reports of passed tests, and possibly additional teacher's comments. The level of given feedback can be regulated to match the purposes of the teacher.

Scheme-Robo analyzes Scheme functions. Instead of comparing the actual output to model output, the system directly evaluates the Scheme expression returned by the target function. In addition, some structural analysis and coarse run time analysis are possible. The VIOPE system evaluates C, C++ or Java programs or SQL statements using quite similar methods as above. However, in syntax analysis it can point out problem areas in the code by highlighting them.

If we consider these systems using the Felder-Silverman model, we observe that their feedback is almost solely textual and thus supports verbal learners better than visual learners. The other common characteristic is that they analyze only complete solutions submitted by students, such as final programs, designs, or circuits. Little aid is given during the process itself. Such mode of working suits intuitive learners who already master, at least partially, the concepts and processes involved. Moreover, the summary feedback, such as points and reports of passed tests need reflective processing. The systems support global learners in the sense that the solution process is left open. The global learner can choose the solution method freely.

In summary, we observe that these systems do not give much support for sensory, visual, and active learners, although such students form the majority of engineering students (Felder & Silverman, 1988). We must, however, recognize that programming is inherently an activity that requires intuitive, verbal, reflective and global approach. For example, there is no sequence of steps to write a program. Writing

program code, of course, is an active process but without a strong intuitive understanding of the goals and concepts used and an ability to reflect the results, it is very hard to solve a given programming exercise.

Assessing Algorithmic Exercises

Algorithms and data structures are inherently abstract concepts. Most people grasp their idea better from a visual representation rather than from a verbal description or a formal definition. For this reason, there is a long tradition of research in algorithm animation. One common type of algorithmic assignment is to create algorithm animations from source code using an algorithm visualization system or using a specialized drawing program (Hundhausen & Douglas, 2002). Another often-used type of assignment is to view an algorithm animation and, possibly, to interact with it in some fashion, for example, by making predictions on how it evolves during a time period (Byrne, Catrambone, & Stasko, 1999).

A fundamentally different approach is *algorithm simulation*. By this we mean an activity where the user directly investigates and manipulates the conceptual view of a data structure. He thus simulates operations a real algorithm would do, and the system records the actions performed and, possibly, gives adequate feedback of the operations. Actual code is not necessarily needed at all. This approach provides an opportunity to automatically assess a user's actions on the conceptual level. However, very few systems exist for this purpose.

PILOT (Bridgeman et al., 2000) and TRAKLA2 are modern algorithm simulation systems. They both provide automatic feedback for algorithmic assignments where the learner must manipulate data structures in order to simulate the operations an algorithm would perform. Both systems support the following features:

1. *Individually tailored exercises* - In both systems it is possible to generate random instances of a problem. Therefore it is possible to give each student a unique instance of the problem, preventing plagiarism.
2. *Direct manipulation* - Using a graphical user interface it is possible to create a solution for the problem by manipulating graphical objects on the screen.
3. *Automatic assessment* - It is possible to submit a solution to the system for assessment, and to get immediate feedback on the correctness of the solution.
4. *Model solution* - The system is capable of creating a model solution for each problem instance. The solution is in the form of algorithm animation.

The main focus of PILOT is on graph algorithms, such as breadth first search or Prim's algorithm. Moreover, the system allows for immediate visual and textual feedback. After each operation on the data structure, the system highlights changes and can tell the learner whether his action was correct. In the case of incorrect action, the system can explain what went wrong. It is also possible to use PILOT without immediate feedback after each action. In this mode, the learner can submit his solution for automatic assessment, after which he is given summarizing feedback: grading, explanatory text, and the model answer.

TRAKLA2, on the other hand, supports a variety of different data structures, like arrays, trees and graphs. The system also contains implementations for abstract data types, such as dictionaries and priority queues. There are, for example, AVL-tree and binary heap and corresponding exercises implemented in the system. However, TRAKLA2 does not support immediate textual feedback. The system visualizes the changes in the data structure after each operation without printing any explanatory text. Furthermore, it is capable of grading each solution as well as creating the model solution.

Figure 1 is an example sequence from an algorithm simulation assignment. The assignment was to simulate the insertions of the keys from the input stream into an initially empty AVL-tree. Between the first three frames keys A, K and Q have been inserted. Finally, between the last two frames a rotation has been performed at the root. All operations were performed through the GUI.

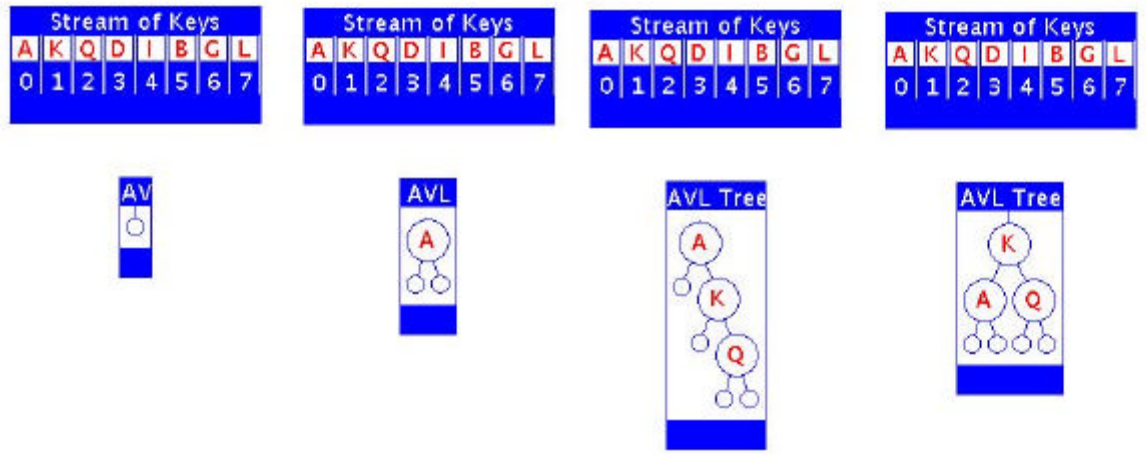

Figure 1: AVL Tree Simulation

While considering systems assessing algorithmic exercises in terms of the Felder-Silverman model, we can see that they provide much more versatile feedback than programming assessment systems. Both PILOT and TRAKLA2 can create a visual model answer, and provide textual feedback. Furthermore, algorithm simulation through direct manipulation provides a way for visual learners to work in the way they prefer. They can manipulate the graphical objects to create the solution. The assessment systems also support both active and reflective learners. An active learner gets immediate feedback on his actions as the visualization changes according to his actions. A reflective learner, on the other hand, gets summarizing feedback when she submits her solution.

Designing Algorithmic Assignments for Different Learners

In this section we discuss more thoroughly the dimensions of the Felder-Silverman learning model, and how they could be incorporated in designing algorithmic exercises. We will use PILOT and TRAKLA2 as examples for demonstrating our ideas.

Characterizing Exercises

The Felder-Silverman model suggests that we could distinguish four independent axes for characterizing exercises. Since we are concentrating on computer-supported exercises, the classification is based not only on how the assignment is presented or solved, but also on the automatically gained feedback. The four axes are

1. sensory vs. intuitive exercises
2. visual vs. verbal exercises
3. active vs. reflective exercises
4. sequential vs. global exercises

The problem in this classification is, however, that students can solve the same exercise in different ways depending on what kinds of learners they are. The following example clarifies this. One exercise in

TRAKLA2 deals with the Boyer-Moore-Horspool string matching algorithm (Korhonen, Sutinen, & Tarhio, 2002b). The user is given a fixed source text and a pattern presented in an array. The user can change the contents of the array containing the pattern. Moreover, he or she can see the skip table for characters and the pseudo code of the algorithm. When the pattern is drag&dropped on the source text, the algorithm starts its execution and each line of code is colored when it is covered during the execution. The assignment is to find such a pattern that all code lines are executed at least once. The user can try different patterns and browse the execution of the algorithm as an animation.

We observed how a small number of test users, including students and teachers, solved the problem. At first we explained the assignment to each user and demonstrated how the system operates. Thereafter they were supposed to solve the problem independently. We observed very different approaches. Some test users did not use the system at all for a while. Instead, they remained pondering on the algorithm, even making some notes on a paper. After a while they had constructed a possibly correct pattern. At this point they executed the algorithm on the data. Usually the test users using this approach found a solution after one or two iterations. One test user, on the other hand, immediately started exploring the behavior of the algorithm by testing different patterns and by observing the results. Pondering aloud he tried to infer how the algorithm (code) works. Finally, after several tests, he understood its working and produced the correct solution. Obviously in the first case people worked intuitively, whereas in the latter case the test person worked both using active experimentation and reflective observation. As a conclusion, we can see that the same assignment applied well to different kinds of learners.

We claim that in general we should prepare assignments that fit to different learners. However, we also recognize the need for training properties, which are weaker in our students. Such assignments should require certain kind of exercising. In the following text we discuss more closely each Felder-Silverman dimension, and give examples of exercises and feedback that support different types of learning in the given dimension.

Sensory vs. Intuitive exercises

Recall from the section on learning models that sensors are patient with details, data, and memorizing facts. They also like solving problems by standard methods and experimentation. Intuitors, on the other hand, prefer theories, principles, innovations, and complications. They are good at grasping new concepts, and they are comfortable with symbols.

A typical set of sensory assignments includes simple multiple-choice questions, fill-in forms or connecting related pairs. Such exercises can be trivially assessed by any well equipped general purpose learning environment, and the feedback about them is simple.

Consider the following exercise presented in TRAKLA2 (see Figures 2 and 3). The student is requested to drag the keys from the tree in preorder into the list below and thus simulate the preorder traversal algorithm. Sensing learners have a practical approach to the exercise. They might apply a rule of thumb to solve the exercise without even looking at the actual algorithm. They could apply the simple mnemonic in which an outline of the tree is drawn (starting from the left side of the root) and each time the outline passes a node from its left side the value is printed. The visual feedback of the drag&drop operations, as well as the changing list, supports their working. More support is provided by the animation controller because they can browse their solution backwards and forwards to review and check it.

For intuitors, this exercise may seem trivial and dull. However, we can easily transform the same exercise into a new form that better suits for intuitors. We simply replace the text "*Traverse the following binary tree in preorder*" by text "*Traverse the following binary tree using the algorithm below ...*", and give the pseudo code of preorder traversal (see Figure 3). Now the student has to understand the algorithm properly in order to solve the problem. Executing the code in one's mind is a task suitable for intuitors, whereas sensors could find this difficult. The feedback provided by TRAKLA2 is a summary of

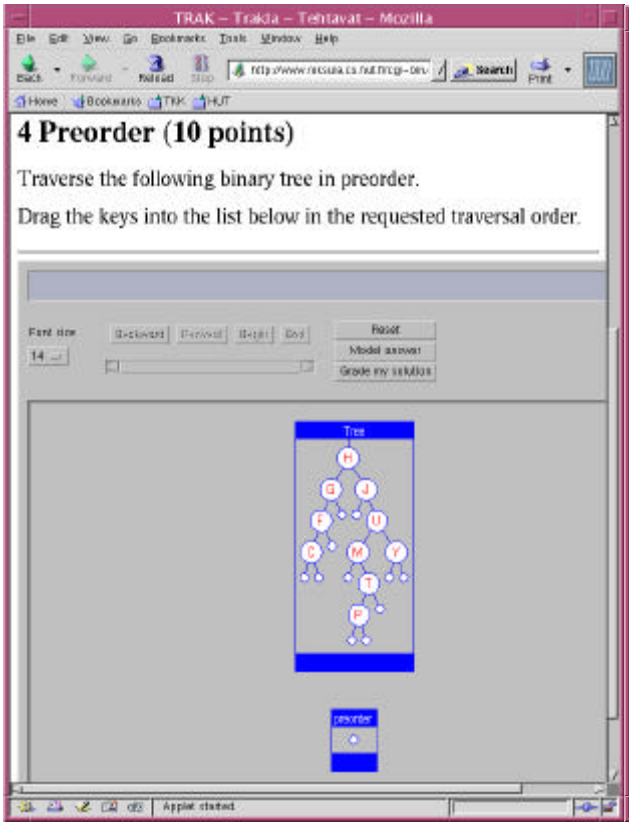

Figure 2: Example exercise supporting sensory learners

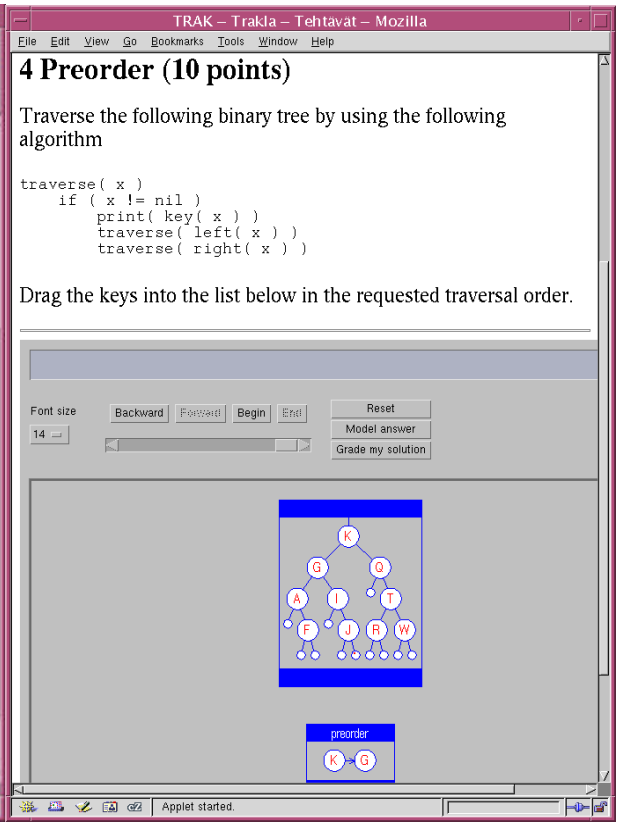

Figure 3: Example exercise supporting intuitive learners

the correct steps. This suits for intuitors confirming the correctness of their thinking. If the answer is incorrect, the information encourages them to reflect where they had made the mistake.

Algorithm simulation provides an interesting method of experimentation for sensors. For example, in PILOT they can exercise how the algorithm works by a simple trial-and-error method with immediate visual feedback on how they proceed. TRAKLA2, on the other hand, supports experimentation by allowing students to work with abstract data types (ADT). They can drag keys into a dictionary ADT and see visually how its structure is changed after each ADT operation (e.g. insert, delete). This allows "what if?" type questions and exploration on how the structure works in different cases. Automatic assessment to gain feedback, however, is not needed here because the visual feedback itself aids learning. For intuitors, this approach allows setting and testing hypothesis, as we already mentioned in the case of Boyer-Moore-Horspool algorithm.

From the student's perspective there is a fundamental difference between the preorder traversal exercise and exploring the behavior of a dictionary ADT. In the first case, the student is allowed to drag the keys in any order to the target list. TRAKLA2 deliberately does not give any hint on what went wrong; only the summarizing result is provided. This forces students to work reflectively, since active exploration to find the correct solution is simply too laborious. However, when we allow ADT operations we can set up exercises where active exploration may lead to a solution. Consider the following assignment: "*Insert the following keys A B C D E F G H I J K L in some order into an initially empty AVL tree so that the height of the resulting tree is 5*". Here the solution can be reached by applying a simple trial-and-error method although intuitive approach is also possible. Actually it could be even faster (one solution is to create a Fibonacci tree).

Visual vs. Verbal exercises

A pure verbal exercise is commonly given in textual form and solved by paper and pencil or by typing in the answer. Finally, the feedback on performance is given in written or audio form. The role of the feedback is to provide general and/or specific context sensitive information about the evaluated submission. Typical examples are explanations of tests that were passed or failed, feedback on observed errors, and summaries on results. As verbal feedback is already used in most assessment systems, we should emphasize work to produce visual feedback in forms such as charts of results, visualizations of processed data, and animations.

A visual exercise in its extreme is presented without any textual additions using only elements like pictures and diagrams. Visual exercises and feedback apply well to working with conceptual understanding since they provide the opportunity to present concepts and their relations as graphical objects. Here the learner can compare his or her own mental model to this visual representation. Moreover, in such a representation less important details can often be hidden.

Let us reconsider our example of preorder traversal. The same assignment can be given and solved either verbally or visually. In the TRAKLA system the preorder assignment is presented verbally. The student is given the adjacency list representation of the tree to be traversed, and he reports a list of traversed nodes. Feedback is sent by email and includes the grading points. In TRAKLA2 the same exercise is purely visual, as could be seen from Fig. 2. The feedback is partly visual (the GUI response to user's operations and the model solution presented as an animation) and partly textual (summary of points).

Even though similar feedback on exercises can be given both visually and verbally, these forms have roles where each of them works better. Verbal feedback applies well to presenting numeric values (e.g. grading), explaining details, their meaning or reasoning behind them. Visual feedback, on the other hand, is good at providing conceptual overviews, highlighting changes and pointing out errors. Moreover, in interactive working such as algorithm simulation, the student can observe the effects of her manipulations on the data structure while solving the exercise. Both PILOT and TRAKLA2 use this method to support learning. Furthermore, because they actually understand the underlying concepts, they are capable of giving feedback also in the conceptual level. Thus, if the user performs operations that are not allowed for the structure, the GUI can alert the user about the misconception.

In many cases, the best method is to combine both visual and verbal feedback. An error can be highlighted and its reason can be explained in words, as is the case in PILOT exercises. Or, a summary statistics is provided in a visual form (chart) added with numerical values of data.

Active vs. Reflective exercises

Active students prefer learning by trying things out, doing active experimentation, while reflective learners prefer examining and manipulating information introspectively. Their working can be described as reflective observation.

For the first ones algorithm simulation is a natural way to delve into the world of algorithms. The user can actively observe and directly manipulate the view, i.e. the student can modify a data structure and the representation is changed immediately as a result. Explorative simulation using ADT operations extends his possibilities in a natural way. If we consider the feedback best supporting such working, we could characterize it as continuous feedback, i.e., every time the student does something, he can observe the results. Both PILOT and TRAKLA2 employ this approach.

For reflective students we should provide forms of interaction that drive them to rethink their solution anew. As an example, the learning environment could provide summaries (as in TRAKLA2) or textual descriptions of the mistakes made by the learner during the exercise session (as in PILOT). However, it

might be better to refrain from pointing out the exact mistakes, and describe them on a more general level, instead. At least the teacher should be able to control the detail level of the feedback.

A very important feature of most automatic assessment systems is that they allow resubmission of exercises (Korhonen et al., 2002a; Malmi et al., 2002). The student can submit a solution several times and get feedback each time. This feature promotes learning both with active and reflective learners. Active learners get feedback on how they proceed, and they can experiment on things. However, if the problem is too simple or the solution space is too small, this may lead to using trial-and-error method without any actual purpose of understanding the problem. With more complicated exercises, resubmission with immediate summative feedback strongly promotes reflection. Most programming assignments, as well as algorithmic exercises have this property. Our experience is that a small number of allowed submissions is better than large. If the student feels that there is no "cost" of using a submission, she will more likely do a small change and resubmit the solution in the hope of fixing the error. No actual reflection takes place compared to the cases where the student has, for example, only 3 or 4 submissions allowed. Loosing one submission for a bad guess has then a high cost for her.

Use of random data to personalize assignments is an important feature in TRAKLA2. It is used in two different ways. In some assignments, the student can resubmit the assignment with the same data for a small number of times. They can continue working with the exactly same assignment after getting the feedback. However, due to the tailored data they cannot copy any answers from their fellow students. The model solution is not shown before the deadline for submissions has passed.

In other exercises randomization of data is used in a different way and the number of resubmissions is unlimited. The student can request the grading of his exercise or view the model solution at any time. However, he cannot continue the exercise, but has to start the whole assignment anew with *new random data*.

We could characterize these two types of exercises in the following way. In the exercises with the same data the student reflects to identify the error in the particular assignment. In the exercises with always new data he reflects more on understanding the algorithm in general. So far, we do not have enough results to say, which method is better for learning, since the latter form of exercises were in use for the first time in spring 2003.

Finally, we note that one important point promoting reflection is the comparison of user's solution and the model solution. If possible, this should be carried out side by side, optionally added with descriptions or summaries about the differences such that force the learner to rethink his solution anew.

Sequential vs. Global exercises

The final dimension concerns sequential vs. global learners. Sequential learners progress linearly step-by-step to the solution and each phase is logically connected to the previous and next phase. For global learners we should emphasize the big picture of the topic and allow the learner to jump between the phases. This applies also to the whole course. Extremely sequential courses follow strict program in which each subtopic is cumulatively dependent on the previous one. On the other extreme, there is no connection at all between different subtopics.

Let us reconsider our binary tree traversal example again. For sequential learners we should provide a meaningful way to process one traversing algorithm after another and for each algorithm a way to solve it step by step until the whole task is completed. For global learners, however, the assignment could emphasize the generalized depth-first search algorithm that visits the nodes before the first recursive call (preorder), between the two recursive calls (inorder) and after the last recursive call (postorder).

In a larger context, a learning environment should provide a way to proceed with the exercises either sequentially or globally. For example, in TRAKLA2 both approaches are supported. There exists a pre-

defined order for the exercises a sequential learner may follow. However, not all students prefer to follow such a strict path. Therefore we allow all the exercises to be fully solvable from the very beginning of the course. The learner may proceed with the exercises in her own order. She may, for example, start by focusing on the exercises concerning linear data structures even though they do not appear consecutively in the other course material.

Conclusion

Learning style models should be taken into account when designing virtual learning environments. In a classroom situation students with different learning styles can get almost immediate feedback from the teachers and from their fellow students. A distance learner working with a Web-based environment, however, is facing a very different learning situation. He lacks such multidimensional support of human tutors, and even if we count opportunities such as email, newsgroups and chat, the feedback is almost totally verbal. Moreover, the delay for getting the feedback is in most cases long. Therefore, we should deliberately design the environments so that they support multidimensional features of learning. We have found that the Felder-Silverman learning model is a useful framework for such a design process.

We observed that many of the current automatic assessment systems provide feedback only for a narrow scope of learners. They seem to support verbal, intuitive, and reflective students best. Visual feedback is provided in some cases, but support for active and sensing learners is rare. Such systems, however, exist and we have presented PILOT and TRAKLA2 as example systems that support multiple learning dimensions. We hope that the examples and the related discussion promote the reader to think how to best support different learners. We should recognize, however, that these issues cannot be solved solely by developing more advanced assessment systems. The key question is how to design activities that promote learning, and thereafter design the feedback in such a way that it best supports these activities.

Suggestions for Learning Programming

As an example of applying our approach to a new subject, let us briefly consider programming courses. Several automatic assessment tools have been developed to support the learning process and speed up grading in such courses. However, programming is a skill that seems to have some inherent properties that match to certain kinds of learners. First, in practice, it is almost totally a verbal activity. Second, the run time execution of a program is a black box, which offers very little insight for sensors into what is going on. Thus understanding the execution requires strong intuitive skills. Third, debugging errors is a highly reflective activity, where active experimentation is a bad guideline. For sensing and active students such a working method easily leads to dummy trial-and-error type testing, with no aim of truly understanding the program execution. Intuitive and reflective learners, on the other hand, can use experimentation by setting hypothesis and interpreting the results to correct the program. Finally, programming skill essentially needs global learning. There is no sequence of operations, or some standard procedure that leads to preparing a working program.

The problem we face is that how could we support the other dimensions in learning programming. We suggest that program visualization tools, as well as visual debuggers should be used much more. They support sensing and visual learners by giving a better insight into what happens while the program is executed. In addition, we should design exercises where active experimentation plays a key role. These could include, for example, studying a working program and preparing data that produces the required output, added with a requirement to reason how the student proceeded to the solution. For sequential learners we could device a sequence of exercises, where they gradually develop a working program by preparing small incremental changes and additions. Automatic assessment tools can be used to support the process by giving the feedback on whether the program is working correctly. Even though such feedback would be totally summative and verbal, the learning process itself has been designed to support different learning styles.

More Aspects on Applying the Model

We can use the Felder-Silverman learning style model to characterize different features of learning environments. The discovery of different learning styles immediately raises the question whether we should emphasize the extremes that are peculiar to a learner or should we try to do just the opposite? A creative solution for this is to do both of them. On one hand, we could develop the environments to support a learner to choose the learning style because the assignments enable different approaches. On the other hand, we could design exercises that force the learner to behave in a way that is typical for students stressing one extreme of learning style. In any case, we have to make sure that both extremes are supported.

Algorithm simulation seems to offer a valuable tool for allowing different kinds of algorithmic exercises. We could identify different aspects of algorithm simulation environments, for example, by looking at the four categories implied by the two dimensions: visual-verbal and active-reflective. These issues can be summarized as a two-dimensional plane with visual-verbal and active-reflective axis, as illustrated in Figure 4. Active learners are supported by immediate feedback and continuous interaction with the GUI in which both visual and verbal dimensions are represented. On the other hand, reflective learners are supported by providing more precise and detailed feedback on misconceptions. In addition, the detailed feedback can be offered automatically either verbally or even visually in terms of algorithm animation.

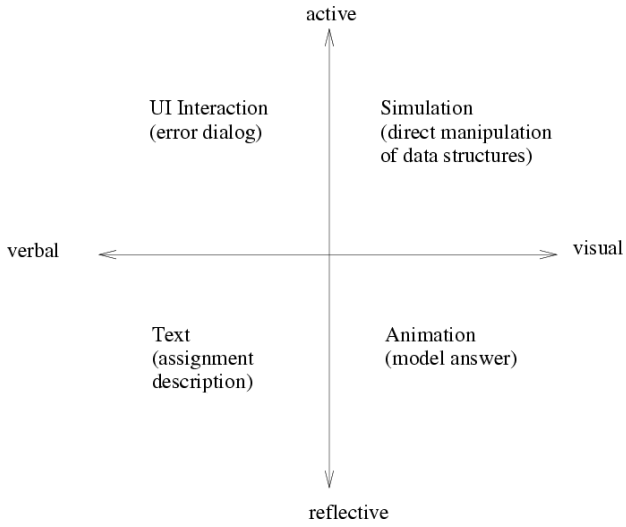

Figure 4: Examples of Visual-Verbal x Active-Reflective Categories

Final notes

According to Felder (1993) the teaching style on most lecture courses tilts heavily towards the few students who are, at the same time, intuitive, verbal, deductive, reflective and sequential learners. Of course, the tools and methods available have also a role to play here. For example, the amount of accurate feedback traditional teaching methods can provide simultaneously for a large number of learners may be very limited, thus forcing to design more reflective than active forms of exercises. However, the design of virtual learning environments should provide a broader vision, since we can use automatic assessment tools to broaden the scope of activities.

One step toward this direction is the work carried out in this paper. Many interesting examples have been identified such that can better support the different learning styles. However, we need further research and results from the area of educational sciences to identify how the different modes of feedback actually affect the learning results. Further work should be based on such analysis.

References

Benford, S., Burke, E., Foxley, E., Gutteridge, N., & Zin, A. M. (1993). Ceilidh: A course administration and marking system. *Proceedings of the International Conference of Computer Based Learning*. Vienna, Austria.

Bridgeman, S., Goodrich, M. T., Kobourov, S. G., & Tamassia, R. (2000). PILOT: An interactive tool for learning and grading. *Proceedings of the 31st SIGCSE Technical Symposium on Computer Science Education* (pp. 139-143). ACM. Available online at http://citeseer.nj.nec.com/bridgeman00pilot.html

Byrne, M., Catrambone, R., & Stasko, J. (1999). Evaluating animations as student aids in learning computer algorithms. *Computers & Education*, *33* (4), 253-278.

Felder, R. M. (1993). Reaching the second tier. *Journal of College Science Teaching*, *23* (5), 280-290.

Felder, R. M. (1996). Matters of style. *ASEE Prism*, *6* (4), 18-23.

Felder, R. M. (2002). Authors preface - June 2002. http://www.ncsu.edu/felder-public/Papers/LS-1988.pdf

Felder, R. M., & Silverman, L. K. (1988). Learning styles and teaching styles in engineering education. *Engineering Education*, *78* (7), 674-681.

Higgins, C., Symeonidis, P., & Tsintsifas, A. (2002). The marking system for CourseMaster. *Proceedings of the 7th annual conference on Innovation and technology in computer science education* (pp. 46-50). ACM Press.

Hundhausen, C. D. & Douglas, S. A. (2002, October). Low-fidelity algorithm visualization. *Journal of Visual Languages & Computing*, *13* (5), 449-470.

Hyvönen, J. & Malmi, L. (1993). TRAKLA - a system for teaching algorithms using email and a graphical editor. *Proceedings of HYPERMEDIA in Vaasa* (pp. 141-147).

Jackson, D. & Usher, M. (1997). Grading student programs using ASSYST. *Proceedings of 28th ACM SIGCSE Tech. Symposium on Computer Science Education* (pp. 335-339). ACM, San Jose, California, USA.

Kolb, D. A. (Ed.) (1984). Experiential learning: experience as the source of learning and development. New Jersey, USA: Prentice-Hall.

Korhonen, A. & Malmi, L. (2000). Algorithm simulation with automatic assessment. *Proceedings of the 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education* (pp. 160-163). ACM, Helsinki, Finland.

Korhonen, A. & Malmi, L. (2002, May). Matrix - concept animation and algorithm simulation system. *Proceedings of the Working Conference on Advanced Visual Interfaces* (pp. 109-114). ACM, Trento, Italy.

Korhonen, A., Malmi, L., Myllyselkä, P., & Scheinin, P. (2002a). Does it make a difference if students exercise on the web or in the classroom? *Proceedings of the 7th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education*, ITiCSE'02. ACM, Aarhus, Denmark.

Korhonen, A., Sutinen, E., & Tarhio, J. (2002b). Understanding algorithms by means of visualized path testing. In Diehl, S. (Ed.), *Software Visualization: International Seminar* (pp. 256-268). Springer, Dagstuhl, Germany.

Malmi, L., Korhonen, A., & Saikkonen, R. (2002). Experiences in automatic assessment on mass courses and issues for designing virtual courses. *Proceedings of the 7th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education* (pp. 55-59), ITiCSE'02. ACM, Aarhus, Denmark.

Norman, D. A. (1983). Some observations on mental models. In D. Gentner & A. Stevens (Eds.), *Mental Models* (pp. 7-14). Lawrence Erlbaum Associates.

Saikkonen, R., Malmi, L., & Korhonen, A. (2001). Fully automatic assessment of programming exercises. *Proceedings of the 6th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education* (pp. 133-136), ITiCSE'01. ACM, Canterbury, United Kingdom.

Vihtonen, E. & Ageenko, E. (2002). Viope-computer supported environment for learning programming languages. *Proceedings of Int. Symposium on Technologies of Information and Communication in Education for Engineering and Industry* (pp. 371-372) (TICE2002). Lyon, France.

Biographies

Lich. Tech **Ari Korhonen** is a researcher at Helsinki University of Technology (HUT). He received his M.Sc (Computer Science) in 1997, Lich. Tech in 2000 and currently he is a PhD student at HUT. His research includes data structures and algorithms in software visualization, various applications of computer aided learning environments, and automatic assessment in computer science education.

Lauri Malmi is a professor of Computer Science in Helsinki University of Technology (HUT). He received his Doctor of Technology diploma in HUT in 1997. His main research area is Computer Science Education including software visualization, automatic assessment, new educational methods, and evaluating how they improve learning.

Jussi Nikander is a M.Sc student at Helsinki University of Technology. He has worked as a research assistant in the Matrix project for over two years, and has been assisting on introductory data structures and algorithms course during this time. His research interests are in data structures and algorithm visualization.

Petri Tenhunen is a M.Sc student at Helsinki University of Technology. He has been for over two years as a research assistant in the Matrix project developing algorithm visualization and automatic assessment systems. He has also been assisting different introductory courses like data structures and algorithms, mathematics and programming during and before this time. His research interests are in Computer Science Education and algorithm visualization.