A Hypertext-Based Approach to Computer Science Education Unifying Programming Principles

WENDY A. L. FOWLER AND RICHARD H. FOWLER
Department of Mathematics and Computer Science
University of Texas-Pan American, Edinburg, TX 78539-2999, USA

INTRODUCTION

Introductory computer science courses must provide students with both a theoretical foundation for study of the discipline and more concrete skills necessary to begin implementing programs. To achieve these goals, students must learn general problem solving skills, algorithm design, and an implementation language. Additionally, software engineering and program design techniques should be learned from the outset, and the beginning student must learn the basic software tools for programming and design. Maintaining a balance and perspective among these somewhat disparate skills is one of the principal challenges of the introductory courses.

A number of studies have investigated novice programming in general and the more specific issues of teaching programming in a computer science curriculum (Baile, 1991; Soloway & Spohrer, 1989; Koffman, Miller & Wardle, 1984). The Association for Computing Machinery's *Computing Curricula 1991* (Tucker, 1990) points out that programming encompasses all of the activities involved in the description, development, and implementation of algorithmic solutions to problems. The inherent difficulty in teaching program and algorithm design at the same time as program implementation is central to the issues of teaching programming. This difficulty is exacerbated by introductory computer science courses and texts that structure the presentation of design and programming principles around the introduction of a particular programming language. It is difficult to avoid this structuring, but for the student learning his or her first programming language there is an understandable tendency to loose sight of abstract concepts when trying to implement an algorithm in a particular language.

The challenge of teaching programming in the introductory courses lies in *simultaneously* teaching 1) general problem solving skills, 2) algorithm design, 3) program design, 4) a programming language in which to implement algorithms as programs, and 5) software tool use that supports design and implementation. Attempts to separate the teaching of these skills have typically focused on separating program design from the other facets. In general, this has resulted in students' acquiring better design skills (Baile, 1991).

The separation of program design and description from implementation is one of goals of computer aided software engineering (CASE) (Boehm, 1981). This suggests that commercially available CASE tools might play a role in the introductory computer science courses (Sidbury, Plishka & Beidler, 1989). Yet, commercially available CASE tools have some potential drawbacks: complexity of the CASE environment, an interaction style possibly different from other elements of the programming environment, documentation that is not appropriate for introductory students (Mynatt & Leventhal, 1990), and expense for both the software itself and the supporting hardware (Kiper, Lutz & Etlinger, 1992).

Student oriented software tools have been implemented that overcome some of these drawbacks. In a system designed for the beginning student the implicit enforcement of design methodology characteristic of CASE systems can be more closely matched to the needs of the student. Carrasquel, Roberts, and Pane (1989) describe a visually based system designed for students in introductory courses. The system allows students to enter and edit a structured design that specifies data and control flow among modules. The system can also interact with other programs for data display and code entry. Schweitzer and Teel (1989) developed a system for teaching structured design that automates several aspects of design and code generation. The SODA system (Hohmann, Guzdial, & Soloway, 1992) closely ties students to a computer aided design model of program design and development. Programs that support student's design and implementation tasks in the introductory courses are a natural extension of a tool based approach automating some aspects of programming.

AN ENVIRONMENT INTEGRATING DESIGN, PROGRAMMING LANGUAGE, AND HYPERTEXT

The environment we are using for our introductory programming courses provides students with a closely integrated suite of pedagogically oriented programs. The programs and the interrelationships among programs are designed to support the independent development of each of the skills necessary for programming, while keeping more abstract concepts of computer science in sight. The environment integrates design and programming tools with a hypertext of lecture material and laboratory exercises.

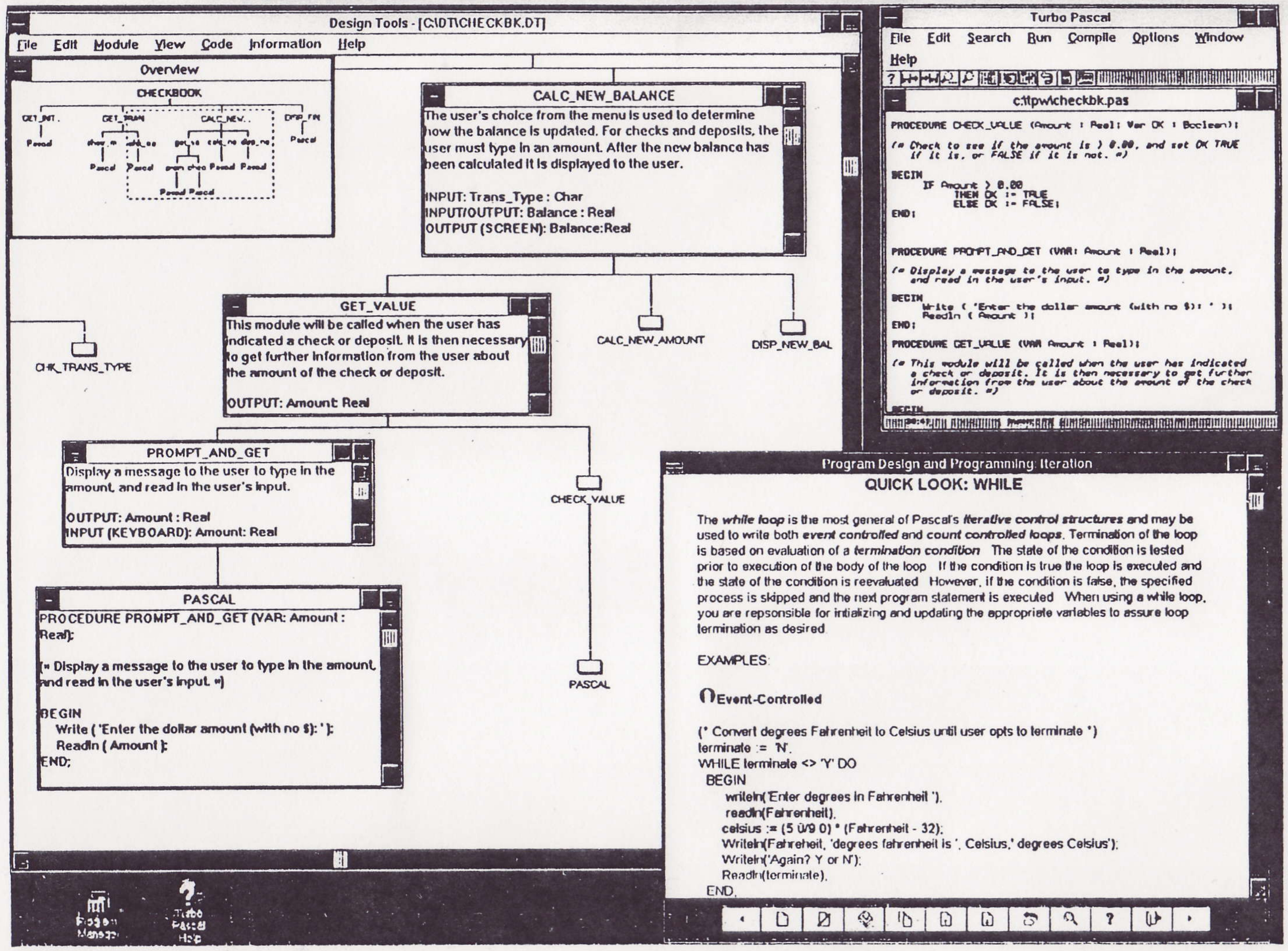
Figure 1 below shows the principal components of the environment. Design Tool, in the upper left corner, is a student oriented CASE tool focusing on visually based structured program design. It serves as a vehicle to develop skills in problem solving by problem decomposition and incorporates a pseudocode language designed for the beginning student.

The hypertext notebook is shown in the lower right of the screen. It includes all material presented in classroom lectures organized as hypertext. The notebook is used concurrently with Design Tool and the programming language environment and incorporates the materials used in student laboratory exercises. Each student has his or her own copy that can be annotated. The Pascal programming environment is shown in the upper right. Information can be easily transferred among each component of the environment.

Design Tool

Design Tool provides a visually based system for problem solving using problem decomposition, program design via structure charts, data flow checking, Pascal code generation, and report production. To facilitate separating the tasks students must master in learning to program, students' first experiences with the laboratory's environment consist of a series of exercises using Design Tool in problem decomposition. In completing design exercises, students are introduced to the diagramming and report writing facilities of Design Tool, as well as the general windowing and editing environment, before learning the Pascal language facilities.

Design Tool provides general tree display facilities for module creation and deletion, reordering and repositioning of modules, collapsing and expanding subtrees and windows, and various layouts of the tree structure. As shown in Figure 1, an overview diagram of the entire structure is always in view orienting the displayed modules to the complete program design. The overview also provides navigation facilities for moving within the module structure. The display facilities for panning, using the overview, and zooming in and out on parts of the structure are designed to facilitate the student's keeping track of the overall design during development.


Design Tools - [C:\DT\CHECKBK.DT]
File Edit Module View Code Information Help
Overview
CHECKBOOK
CHK_TRANS_TYPE
CALC_NEW_AMOUNT
DISP_NEW_BAL
CHECK_VALUE
PASCAL
Turbo Pascal
File Edit Search Run Compile Options Window Help
c:\tpw\checkbk.pas
Program Design and Programming: Iteration

CALC_NEW_BALANCE

The user's choice from the menu is used to determine how the balance is updated. For checks and deposits, the user must type in an amount. After the new balance has been calculated it is displayed to the user.

INPUT: Trans_Type : Char
INPUT/OUTPUT: Balance : Real
OUTPUT (SCREEN): Balance:Real

GET_VALUE

This module will be called when the user has indicated a check or deposit. It is then necessary to get further information from the user about the amount of the check or deposit.

OUTPUT: Amount: Real

PROMPT_AND_GET

Display a message to the user to type in the amount, and read in the user's input.

OUTPUT: Amount : Real
INPUT (KEYBOARD): Amount: Real

PASCAL

```
PROCEDURE PROMPT_AND_GET (VAR: Amount :
Real);

(* Display a message to the user to type in the amount,
and read in the user's input. *)

BEGIN
  Write ( 'Enter the dollar amount (with no $): ' );
  Readln ( Amount );
END;
```

```
PROCEDURE CHECK_VALUE (Amount : Real; Var OK : Boolean);

(* Check to see if the amount is > 0.00, and set OK TRUE
   if it is, or FALSE if it is not. *)

BEGIN
    IF Amount > 0.00
        THEN OK := TRUE
        ELSE OK := FALSE;
END;


PROCEDURE PROMPT_AND_GET (VAR: Amount : Real);

(* Display a message to the user to type in the amount,
   and read in the user's input. *)

BEGIN
    Write ( 'Enter the dollar amount (with no $): ' );
    Readln ( Amount );
END;

PROCEDURE GET_VALUE (VAR Amount : Real);

(* This module will be called when the user has indicated
   a check or deposit. It is then necessary to get further
   information from the user about the amount of the check
   or deposit. *)

BEGIN
```

QUICK LOOK: WHILE

The ***while loop*** is the most general of Pascal's ***iterative control structures*** and may be used to write both ***event controlled*** and ***count controlled loops***. Termination of the loop is based on evaluation of a ***termination condition*** The state of the condition is tested prior to execution of the body of the loop If the condition is true the loop is executed and the state of the condition is reevaluated However, if the condition is false, the specified process is skipped and the next program statement is executed When using a while loop, you are repsonsible for intializing and updating the appropriate variables to assure loop termination as desired

EXAMPLES:

Event-Controlled

```
(* Convert degrees Fahrenheit to Celsius until user opts to terminate *)
terminate := 'N'.
WHILE terminate <> 'Y' DO
 BEGIN
     writeln('Enter degrees in Fahrenheit ').
     readln(Fahrenheit).
    celsius := (5 0/9 0) * (Fahrenheit - 32);
    Writeln(Fahreheit, 'degrees fahrenheit is ', Celsius,' degrees Celsius');
    Writeln('Again? Y or N');
    Readln(terminate).
 END.
```

Figure 1. A typical screen showing the components of the student environment. The upper left window is the student oriented program design and problem solving tool showing an overview of the complete design and the student's arrangement of the design and code. The upper right window is the Pascal editor with code transferred from Design Tool. The hypertext in the lower right is opened to a section giving a tutorial overview of a programming construct.

A student oriented pseudocode language, centering on data flow and consistency, is available in Design Tool. A complete Pascal program module can be generated automatically from the pseudocode. Default values for language elements allow the student to design moderately complex problem solutions and generate most of the Pascal code very early. These early exercises serve to reinforce the lesson that programming focuses on problem solution and design and not the particular programming language or hardware. CASE software is typically designed to tie its user, more or less rigidly, to a particular design methodology (Vessey, Jarvenpaa & Tractinsky, 1992). In Design Tool several constraint and consistency checking options can be used to enforce a particular methodology. For example, during the early weeks of the course, modules can only be created using a top down methodology. The student is required to start at the top of the module hierarchy, only adding modules directly below existing modules. Relaxing various constraints enforced by the system, regarding the availability and consistency of data, gives the user more freedom in the design of programs. With no module creation or data flow constraints, the system can be used as a sketchpad in creating and connecting modules. Data flow and consistency for module sets can be checked at the request of the user. Finally, the student's design created in Design Tool can be printed and used as his or her written design document. Subsequently, the design is used as the basis of the implementation. The information in each module can be copied directly into the program file when code is generated, serving as documentation and a guide for the developing program.

Hypertext Notebook

The hypertext serves to integrate concepts introduced in the classroom and text with the design and programming environment. The notebook contains the course lecture material and supplements: course syllabus, schedule, laboratory exercises, programming assignments, and a large set of designs and example programs. The system in which the hypertext is implemented allows the direct transfer of information to both Design Tool and the programming environment. Laboratory exercises are based on transferring programs and designs from the notebook to the programming environment. Students are required to exploit the integrated nature of the learning environment using Design Tool to interactively examine program structure by executing example programs. Students are also encouraged to annotate their copies of the hypertext for study.

The hypertext is structured to reflect the sequence and scope of lectures. General lecture topics serve to organize the text as units. Within units, main lecture topics serve as organization points from which students can explore concepts in greater detail. Each unit provides numerous examples illustrating newly introduced concepts that build on previous examples and concepts. Students can experiment with the programs by copying them directly into the programming environment, and then compiling and executing the code. All examples used in the lecture are in the hypertext so that during lecture students can focus on a program's explanation and discussion and later use the hypertext to review and annotate code. One element of the hypertext is the Quick Look screen, as shown in Figure 1. A Quick Look provides a brief explication of a concept introduced in class and the text. The hypertext includes a series of these screens for the control and data structures presented in the course. These screens provide a readily available and easy to use help facility for programming language semantics and data structure concepts analogous to the programming language syntax help facilities included in most programming language environments. The screens can be displayed together with corresponding laboratory exercises to help students gain a greater understanding of the implementation of concepts. Key concepts on the Quick Look screens are linked to more detailed information in the hypertext.

The development of the hypertext was in part motivated by a curriculum restructuring following the suggestions of the Association for Computing Machinery's *Curricula '91*. During the course of authoring the hypertext, frequent referral to *Curricula '91* suggested that for our own use it might be useful to have that document on-line. To meet this need, the printed document was scanned and converted to hypertext. We maintain a separate instructor's copy of the course hypertext that incorporates links to the *Curricula '91* hypertext.

DISCUSSION

The structure of our introductory courses, based on an environment using a hypertext to integrate skills, is designed to address the challenges faced by beginning computer science students in developing programming expertise. The goal of the environment is to facilitate a pedagogy that allows instruction to focus on each of a set of somewhat disparate skills. The early introduction of program development tools as an integral part of program implementation serves as a foundation for a tool based approach used in more complex software development environments.

Design Tool is the first program students interact with, and the first weeks of the course focus on developing problem solving skills using problem decomposition. The use of this visually based environment for problem solving together with the hypertext is typically greeted with an enthusiasm that can help overcome some of the natural frustrations in learning any new software system. The repository of program designs in the hypertext notebook provides examples of complete designs for study in the early weeks. The first exposure to the programming language facilities entails transferring and executing programs from the student's hypertext notebook. This is designed to attenuate some of the difficulty in developing skills using the language facility, a "training wheels" approach.

The code generation facilities of Design Tool are designed to allow students to focus on algorithm development. Much of the potential for errors in data flow among modules is eliminated by constraint checking in Design Tool before moving to the implementation facilities. As students progress to the development of more complex designs, the strictly top down constraints of problem solving and module development are relaxed. This allows a design methodology reflecting both top-down and bottom-up program development appropriate for module reuse and a different design methodology. Finally, the hypertext notebook serves to physically and conceptually relate the more abstract concepts introduced in the course to the details of the design and implementation environment.

Over the past year we have used student questionnaires to assess student response to the tools and attitudes towards design and implementation concepts. Students have also completed pre-test and post-test exercises, and a standardized series of assignments and exams. Each semester students are required to complete three design assignments prior to engaging in any programming. Nine programming assignments are completed, each emphasizing specific control and data structures. Each programming assignment must include a preliminary design document. While we have not completed our evaluation, we present a brief review of our findings to date.

We have seen an improvement in modular design. The number of designs submitted has increased and the quality and completeness of each design has improved. Students using the suite of tools show an increase in the efficiency of their programming as measured by time to complete programs. On the first few programs, time spent designing the solution has increased by 20 to 30 minutes, but the time spent programming has decreased on average by one hour. For larger programming assignments, students have again increased their design time by 20 to 30 minutes, but have decreased the time to complete a program by two to five hours. Design Tool facilitates the design process and provides the students with a framework to organize their problem solution prior to entering the actual coding phase.

Student use of the hypertext has increased overall utilization of the laboratory. Though the time spent in programming has decreased, the time working in lab and experimenting with programming concepts has increased. Students spend a slightly greater number of hours in lab each week, and the distribution of the time on various activities has changed. Less time is spent on actual assignments and there is an increase in time spent investigating the concepts discussed in class, as well as other concepts not yet introduced. It is not uncommon for 10%-20% of the students include design or programming concepts not yet introduced in lecture in their programs. This exploration seems to be facilitated by the hypertext with its easy access to example programs and conceptual information. Students have indicated that they find the increasing levels of detail in concept presentation, the multiple examples of each programming concept, and the executable examples supporting each concept useful. Not only can they view the text of the program, but they can copy the programs into the programming environment and observe the results of the execution.

FUTURE DIRECTIONS

To further support students in the mastery of design and implementation skills, we are developing an algorithm animator which will be integrated with the hypertext, Design Tool, and the Pascal programming environment. The animator's visual display of data and control flow will provide an additional representation of the concepts introduced in the course. As with the other components of the learning environment, the animator is designed to supply a bridge between abstract concepts and more concrete representations.

References

Baile, F.K. (1991). Improving the modularization ability of novice programmers. *ACM SIGCSE Bulletin, 23*(1), 277-282.

Boehm, B.W. (1981). *Software Engineering Economics*. New York: McGraw-Hill.

Carrasquel, J., Roberts, J., & Pane, J. (1989). The Design Tree: A visual approach to top-down design and data flow. *ACM SIGCSE Bulletin, 21*(1), 17-21.

Hohmann, L., Guzdial, M., & Soloway, E. (1992). SODA: A computer-aided design environment for the doing and learning of software design. *Proceedings of the 4th International Conference, ICCAL '92: Computer Assisted Learning* (pp. 307-319). Berlin: Springer-Verlag.

Kiper, J., Lutz, M.J., & Etlinger, H.A. (1992). Undergraduate software engineering laboratories: A progress report from two universities. *ACM SIGCSE Bulletin, 24*(1), 57-62.

Koffman, E.P., Miller, P.L, & Wardle, C.E. (1984). Recommended curriculum for CS1: A report on the ACM curriculum task force for CS1. *Communications of the ACM, 27*(10) , 998-1001.

Mynatt, B.T., & Leventhal, L. M. (1990). An evaluation of a CASE-based approach to teaching undergraduate software engineering. *ACM SIGCSE Bulletin, 22*(1), 48-52.

Schweitzer, D., & Teel, S.C. (1989). AIDE: An automated tool for teaching design in an introductory programming course. *ACM SIGCSE Bulletin, 21*(1), 136-140.

Sidbury, J.R., Plishka, R., & Beidler, J. (1989). CASE and the undergraduate curriculum. *ACM SIGCSE Bulletin, 21*(1), 127-130.

Soloway, E. & Spohrer, J.C. (Eds.). (1989). *Studying the Novice Programmer.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Tucker, A.B. (Ed.). (1990). *Computing Curricula 1991: Report of the ACM/IEEE-CS Joint Curriculum Task Force.* New York: ACM Press.

Vessey, I., Jarvenpaa, S.I., & Tractinsky, N. (1992). Evaluation of vendor products: CASE tools as methodology companions. *Communications of the ACM, 35*(4), 90-105.

Acknowledgments

Intel Corporation donated equipment and support for the student laboratory. The graphic display program used in this work was developed under NASA grant NAG9-551 to the second author. Thanks to Iris Flores for writing and testing 210 programs and designs, and to Maybeth Shirah for help in converting Curricula '91 to hypertext.