ANNANN –A TOOL TO SCAFFOLD LEARNING ABOUT PROGRAMS

L.A. Carr
Intelligence Agents Multimedia, ECS
The University of Southampton
Southampton, UK
lac@ecs.soton.ac.uk
http://www.ecs.soton.ac.uk/~lac

H.C. Davis
Learning Technology Group, ECS
The University of Southampton
Southampton, UK
hcd@ecs.soton.ac.uk
http://www.ecs.soton.ac.uk/~hcd

Su White
Learning Technology Group, ECS
The University of Southampton
Southampton, UK
saw@ecs.soton.ac.uk
http://www.ecs.soton.ac.uk/~saw

ABSTRACT

It is difficult for a student to learn about programs and to understand the rational that went into the development of the parts that led to the whole. Tools for explaining this essentially dynamic process are limited and typically static in nature. This paper presents AnnAnn, an animated code annotator which makes it possible to present the development of code to large groups or for self study. The educational benefits of this approach are examined.

Keywords

Learning to Program, literate programming, Program Development, cognitive apprenticeship, scaffolding, constructivist learning.

1. INTRODUCTION

As part of our task in helping students to learn to program we often need to show them programs. A constructivist view of learning to program suggests that learning happens by iterative refinement of understanding [1], and that an important activity in this refinement involves the study of programs produced by experts [2].

In the ideal world we would have one-to-one tutorials with each student [3], where we could walk through the intricacies of designing a solution to a problem, and the students would gain instant feedback on their nascent understanding as it developed [4]. In practice we must often talk to large lecture halls full of students, or we must ask them to conduct their studies alone.

Presenting programs to large groups is difficult and the problem with working alone is that example program study materials are usually static in nature so that it is difficult for the student to see how the final program was developed, and programs often contain so much information that it is hard for a beginner to understand where to start.

This paper starts by reviewing the existing technologies used for presenting and annotating program evolution, then presents AnnAnn – an animated code annotator. It concludes by examining the benefits of using this tool from the point of view of both the teacher and the learner.

2. LANGUAGE

Learning to program is a difficult task, requiring engagement with a significant number of abstract concepts and with their realisation and embodiment in sample programs, in specific languages, solving particular problems. In teaching programming, a lecturer is frequently required to explain the workings of a number of non-trivial programs so that the students can build up an understanding of the simultaneous threads of:

(a) the language syntax

(b) the language constructs situated in context

(c) designing a program that solves a real problem

(d) constructing a complete program

A presentation that shows a program and explains how it works must concurrently deal with hundreds of lines of code, many methods and possibly multiple classes together with an explanation that addresses each of the above issues as they emerge.

2.1 Photocopied Acetates

The most direct way to lecture about a program is to photocopy the listing onto acetates. This is cheap to do and requires minimal resources, but puts an enormous burden on the lecturer for remembering the 'script' for what needs explaining in what order. For example*:*

(i) show the class outline including constructor;

(ii) show how its static main method creates an instance of this class

(iii) delegate the button's events to the event handler object..

A typical explanation may involve the elaboration of several dozen individual points.

2.2 Powerpoint Programming

Figure 1 shows an example from a typical Deitel and Deitel *Java How To Program lecturers'* slide set [5]. The restricted screen size means that only 24 (of the almost 200) lines can be displayed at a time. The blocks of explanatory text are displayed one at a time in the running slideshow; they variously explain variable declarations, named constants, method invocations, flow of control , and overall effects.

The sequential presentation of the program (through 8 slides) means that the explanation is constrained to be in program order. The main difficulty for the lecturer is that the explanatory texts must be placed at a particular position on the screen real-estate. Any alteration to the program, while developing or maintining this resource, invalidates the chunking of code, the position of the explanations and of the arrows which tie them to the program lines. It is this approach that renders the PowerPoint solution infeasible for anything but small, easily chunked codes samples.

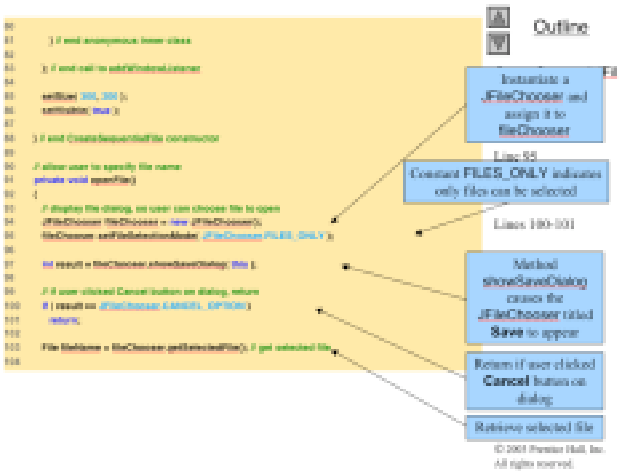


Figure 1: Powerpoint Slide

Deirel and Deitel: Java: How to Program [5]

2.3 Textbook Layout

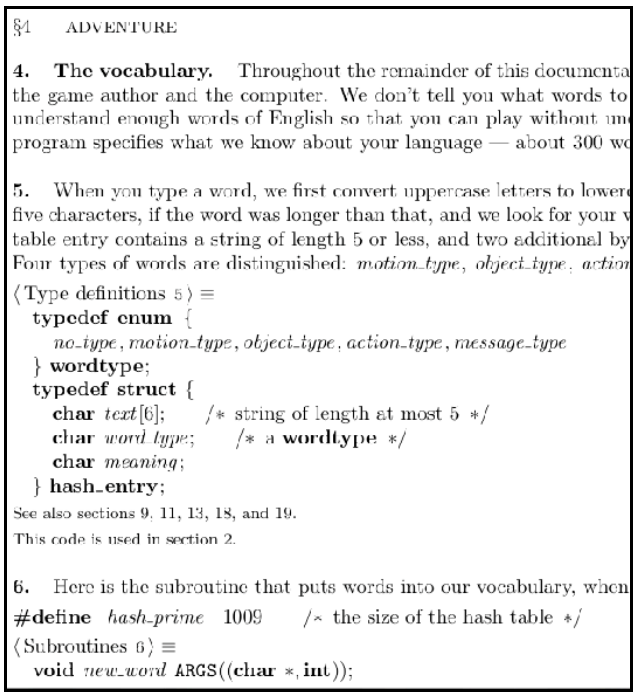
A related approach is one commonly used in textbooks, reproducing the listing as a figure (as in figure 2, shown with numbered lines and highlighted regions). Text in subsequent pages refers back to individual lines. Increased freedom with his format comes from the ability to give the explanation in any order in the main text and to refer back to the code out-of-sequence. The disadvantage with parallel texts is the reader's need to track backwards and forwards as reference is made to different regions of code. Bt contrast, some textbooks embed the code fragments into the text (as with Arnow and Weiss, Java: An Object Oriented Approach, Addsion Wesley). This maintains the freedom to discuss the program elements in the most appropriate order.

Figure 2: Text Book Figure

Deitel and Deitel: Java: How to Program [5]

Literate Programming

Knuth developed *Literate Programming [6]* as a way of mixing documentation and code which allows the programmer to develop very sophisticated explanations which break up the standard program ordering and interleave it with TeX or troff documentation commands (the source program and document are derived by programs called 'tangle' and 'weave'). Although it has been used in a teaching context [7], it is too complex for Introductory programming courses as it adds an extra layer of complexity in the programming task.


§4 ADVENTURE

4. **The vocabulary.** Throughout the remainder of this documenta the game author and the computer. We don't tell you what words to understand enough words of English so that you can play without un program specifies what we know about your language — about 300 wo

5. When you type a word, we first convert uppercase letters to lower five characters, if the word was longer than that, and we look for your v table entry contains a string of length 5 or less, and two additional by Four types of words are distinguished: *motion_type*, *object_type*, *action*

⟨ Type definitions 5 ⟩ ≡
```
typedef enum {
  no_type, motion_type, object_type, action_type, message_type
} wordtype;
typedef struct {
  char text[6];      /* string of length at most 5 */
  char word_type;    /* a wordtype */
  char meaning;
} hash_entry;
```
See also sections 9, 11, 13, 18, and 19.
This code is used in section 2.

6. Here is the subroutine that puts words into our vocabulary, when
```
#define hash_prime 1009      /* the size of the hash table */
```
⟨ Subroutines 6 ⟩ ≡
```
void new_word ARGS((char *, int));
```

Figure 3: Literate Programming

3. AnnAnn

AnnAnn is a simple documentation system that embodies a constructive explanation paradigm, allowing the lecturer to work from a familiar starting point by showing (and explaining) a small change to take the code one step closer to the final solution [8].

The AnnAnn compiler takes an original file, and a list of changes to be applied and produces a Web presentation in Dynamic HTML. An extract of an AnnAnn file is shown in figure 4; the rather terse syntax (similar to the UNIX patch command) allowing the author to create blocks of micro-explanation. We have recently developed a GUI based editor (figure 6) which allows the user to specify the changes directly and to annotate these changes.

The aim of an AnnAnn explanation is to start with a familiar program (in the most extreme case, a Hello World program, applet or JFrame) and by applying successive small changes (adding and initialising an array, fleshing out a for loop, creating a user interface object) to turn it into a different program for a different purpose. A *Hello World* program can be turned into a character-by-character file reading program in a dozen steps, three more steps will enable line-by-line reading, four more create a program which reads from pages on the Web *etc*.

```
# So far we've made the computer print out a static message.
# Instead, lets make it do a bit of work - I've always had
# trouble with my seven times table and I'd like to check
# what 5 * 7 is.
< "Hello World"
> 5 * 7

# Now that I think of it, I need to know 6 * 7 as well.
< 5
> 6

# And 7*7.
< 6
> 7

# There is one quantity here which is varying each time I
# try out the program. I could put it in a Java *variable*,
# which helps the program model the fact that I am
# looking for *something* times seven.
< 7
> <i>something</i>
```

Figure 4: AnnAnn explanation language

Each block in an AnnAnn file identifies a region of the program that needs to be altered, the altered text and a paragraph of explanation indicating to the students why the change needed to occur and how it achieves its goals.

Figure 5 shows AnnAnn in use. A code fragment is on display, and explanation of the next change to make is on display, and the highlighted lines are about to be replaced. The user can step backwards and forwards through all the steps between the initial code and the final code till they properly understand the reason for each addition.

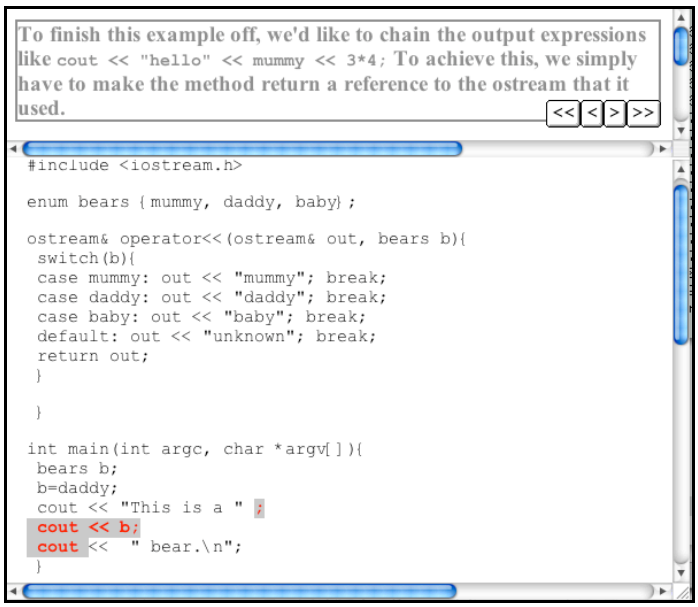


Figure 5: AnnAnn in use

AnnAnn takes a base program and a file of annotated changes and produces a family of HTML files

1. A simple set of HTML files that are backwards compatible with all browsers that support style sheets.
2. A compact, frames-based Dynamic HTML for modern browsers
3. A printable version that combines all the changes for each step onto a single slide.

Since AnnAnn displays through standard web browsers it is suitable for use in lectures are for students to study alone.

4. The Educational Perspective

When searching for an educational perspective on the pedagogic appropriateness of various approaches to programming it seems although there have been some examples of mapping approaches to educational theory, the predominant approach has moved little from Lemos' 1979 perspective that "most of the literature consists of subjective opinions on the most effective methods of instruction for a given programming language". [9]

We have shown that AnnAnn provides teachers with a way to explain the development of a program from some known and previously understood situation to a more complex program possibly using features a student may not have previously understood.

The end goal of designing good programs has always been that the student will learn how to decompose problems into appropriate classes with appropriate methods (or to make some other top down structured design). But some thought shows that it is unreasonable on teachers' parts to assume that this is a skill that students can be expected to pick up easily in the first instance before they have learned about programming "in the small" and the whole paradigm of programming and state machines. Failed attempts at teaching object first programming have lead some (e.g. [10]) to observe that this is an inappropriate way to learn programming.

The authors are firm supporters of the "object first" approach to learning programming, but after some years of taking this approach have come to understand, as have many others (e.g. [11][12]) the enormous cognitive leaps that we are asking our students to take. In the past when students were presented with a Basic interpreter and experimented initially at the command line they slowly built up a model of what the computer was doing, whereas when we teach programming in Java, they have an enormous number of new concepts to understand within typically a few weeks. We have observed that while students who have some previous understanding of programming can cope with our approach, students who have no previous experience of programming often struggle [13].

Anecdotally we are familiar with the student who turns up asking for help half way through the course saying they have just realised that "they just don't know where to start – they don't understand anything". This is typically at the point in the course when we ask the students to complete their first non-trivial assignment, and on investigation the problem turns out to be that while they have succeeded in getting a tenuous grasp of the concepts of class and methods, they do not yet have enough practice or confidence to design a program on their own.

From an educational point of view the thing to do when you ask students to make large cognitive leaps is to provide scaffolding– artefacts that hide some of the complexities of a problem so that the students may keep their eye on the big picture and achieve the major goal of the exercise [14]. Ideally such artefacts should be "fadeable", so that they may be incrementally removed as the student learns to work without the scaffolding.

A simple example of a scaffolding tool that we are familiar with in program development is the input line completion and formatting feature in many IDEs which, for example, give us hints as to the number and purpose of the parameters to a method as we are typing.

AnnAnn is a scaffolding tool in that it provides a way to explain to students the design process by dynamically presenting each part of the solution as it is needed. This feature may be used by a teacher in class to demonstrate to students how a program is designed, or how a particular programming principle may be applied, or it may it may be used by students wishing to study the problem in their own time (and possibly at a distance).

Another education perspective is to view AnnAnn as a tool to aid cognitive apprenticeship [15]. The structure of the tool is such that it easily supports the skilled practitioner demonstrating to the novice the methods they choose to use when building a program. As such it sits between the place where the 'master' builds the program in front of the novice using totally authentic tools; and where the novice is provided with an overly complex completed product. It may also be that the use of the tool directs the master into making explicit 'tacit knowledge' which they routinely draw upon to build a program.

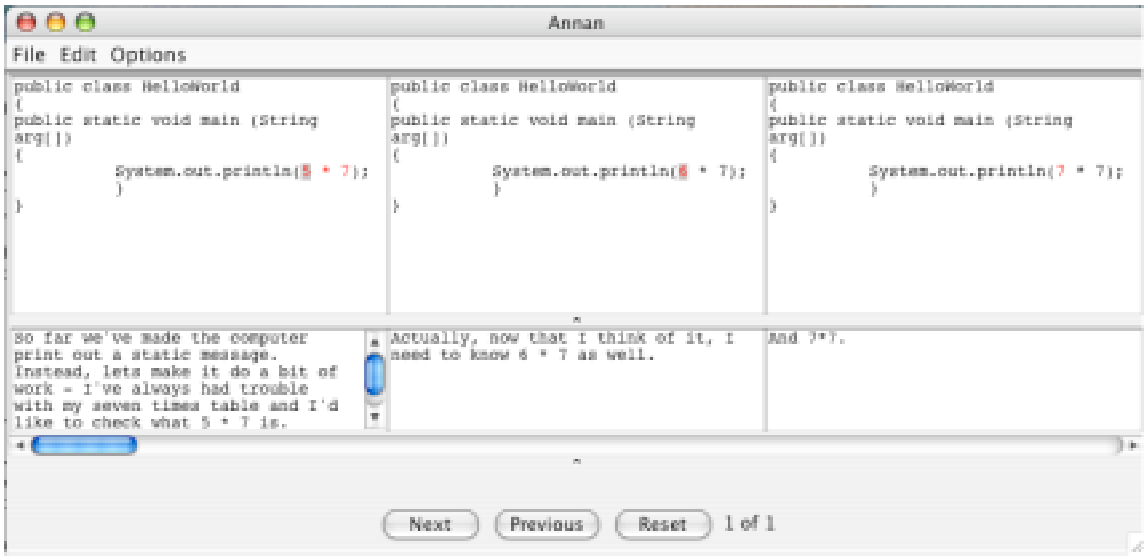


Figure 6: AnnAnn Authoring GUI

5. CONCLUDING REMARKS

We have described AnnAnn, a tool to assist students to understand programs and we have described its use. We have explained the reasons why we developed the tool, and justified the educational frameworks within which we believe it sits.

In practice we have found two distinct modes in which we use this tool. The first is to explain the application of new programming principles, constructs and patterns as the focus of a teaching event. We have also found it useful as a tool to document and explain some complicated template code prior to students being required to make alterations and additions as the basis of some coursework, saving contact time.

A visit to the AnnAnn website [8] will provide the reader with numerous examples of its use, and the first author can provide the tools to others on request. What AnnAnn now needs is community; we hope that others will contribute both to the on-line examples and to the development of the tools.

6. REFERENCES

[1] Mayes, J.T. Learning Technology and Groundhog Day. In W. Strang, V. Simpson, & D. Slater (Eds) Hypermedia at work: Practice and Theory in Higher Education, Canterbury, University of Kent Press. 1995. http://apu.gcal.ac.uk/clti/papers/Groundhog.html

[2] Said Hadjerrouit . A constructivist approach to object-oriented programming. In the Proceedings of the 4th annual SIGCSE/SIGCUE ITiCSE conference on Innovation and technology in computer science education pp 171—174. ACM Press 1999

[3] Bloom, B. S.. "The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring." Educational Researcher 13: 3-16. 1984

[4] Ben-Ari, M. Constructivism in computer science education.In the Proceedings of the twenty-ninth SIGCSE technical symposium on Computer science education, Atlanta, Georgia, pp 257-261, ACM Press, 1998

[5] Deitel, H.M. and Deitel, P.J. *Java How To Program.* Prentice Hall. 1997

[6] Donald E. Knuth, Literate Programming ((CSLI Lecture Notes, no. 27.) Stanford, California: Center for the Study of Language and Information, 1992), xvi+368pp.ISBN 0-937073-80-6

[7] Stephen Shum and Curtis Cook. Using literate programming to teach good programming practices, ACM SIGCSE Bulletin, v.26 n.1, pp. 66-70, March 1994

[8] http://www.annann.org/ last accessed 25 March 2004.

[9] Ronald S. Lemos, Teaching programming languages: A survey of approaches ACM SIGCSE Bulletin,Proceedings of the tenth SIGCSE technical symposium on Computer science education, Volume 11 Issue 1, Jan 1979

[10] David Callear: Teaching Programming: Some Lessons from Prolog: In the Proceedings of the LTSN-ICS 1st Annual Conference, Heriot-Watt, 2000

[11] Zhu H & Zhou M. Methodology First and Language Second: A Way to Teach Object Oriented Programming. ACM OOPSLA '03, Anaheim Ca. 2003

[12] Roger Duke, Eric Salzman, Jay Burmeister, Josiah Poon, Leesa Murray December (2000) Teaching programming to beginners - choosing the language is just the first step Proceedings of the Australasian conference on Computing education

[13] Davis, H.C., Carr, L.A., Cooke, E.C. & White, S.A.. *Managing Diversity: Experiences Teaching Programming Principles*. In the proceedings of the 2nd LTSN-ICS Annual Conference, London. 28 - 30 August 2001

[14] Hogan and Pressley. Scaffolding student learning instructional approaches and issues. Cambridge, Brookline Books. 1997

[15] Collins, A., J. S. Brown and S. Newman. Cognitive Apprenticeship: Teaching the Craft of Reading, Writing and Mathematics. Knowing, Learning and Instruction: Essays in Honor of Robert Glaser. L. B. Resnick. Hilldale, NJ, Erlbaum.