AnnAnn – Next Steps for Scaffolding Learning About Programs

SA White LA Carr HC Davis
CJ Hooper TP Griffith GB Wills
Learning Technologies Group
School of Electronic and Computer Science
University of Southampton
{saw, lac, hcd, cjh302, tpg102, gbw} @ ecs.soton.ac.uk

Abstract

It is difficult for a student to learn how to program and to build an understanding of the rationale which underpins the development of a program's component-parts. Conventional tools and approaches for explaining this essentially dynamic process are limited and typically static in nature. This paper presents AnnAnn, an animated code annotator, which can be used for self study and to present code development to large groups. The technical and educational benefits of this approach are examined.

1. Introduction

In helping students learn to program we often need to show them programs. A constructivist view of learning suggests that effective learning is enabled by the "iterative refinement of understanding" [1]. In programming 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 either take a didactic approach of talking formally to large groups of students lecture halls, 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.

One solution to this problem is the use of animations. First suggested by Baecker [16]; animations can reflect the temporal nature of code; show the sequence of changing events; demonstrate alternative views; and simplify the introduction of structure.

This paper starts by reviewing the existing technologies used for presenting and annotating program evolution, and 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. Understanding is tested and reinforced by the embodiment and realization of these concepts in sample programs, utilizing specific languages and programming constructs through the solving of 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) exploiting the language syntax
(b) using 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 explain show 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 maintaining 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.

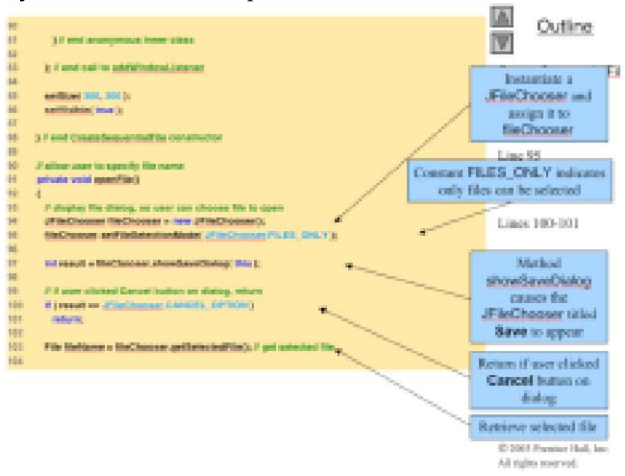

Outline
Instantiate a JFileChooser and assign it to fileChooser
Line 95
Constant FILES_ONLY indicates only files can be selected
Lines 100-101
Method showSaveDialog causes the JFileChooser titled Save to appear
Return if user clicked Cancel button on dialog
Retrieve selected file
© 2003 Prentice Hall, Inc. All rights reserved.

Figure 1 PowerPoint Slide
Deitel 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 this 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. By contrast, some textbooks embed the code fragments into the text (as with Arnow and Weiss, Java: An Object Oriented Approach, Addison Wesley). This maintains the freedom to discuss the program elements in the most appropriate order.

2.4 Literate Programming

Knuth developed Literate Programming [6] as a way of mixing documentation and code. It 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.

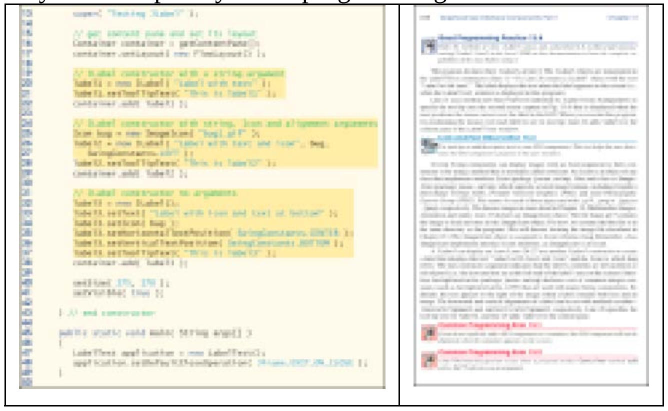
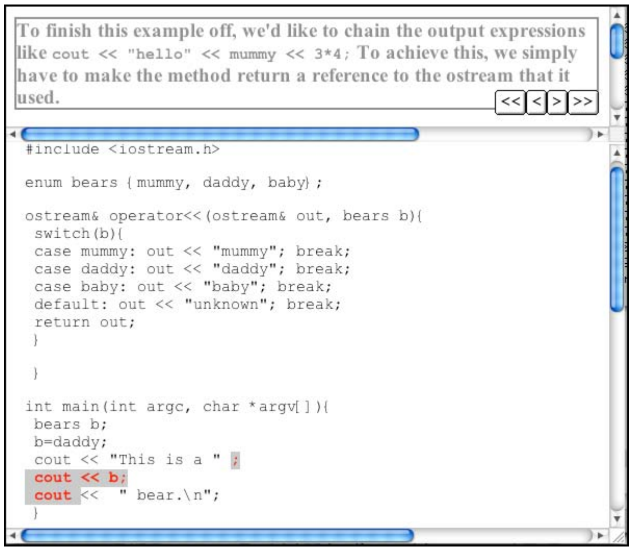


Figure 2 Text Book Figure
Deitel and Deitel: Java: How to Program [5]

3. The Evolution of 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 first version of AnnAnn took a starting file and a list of changes to be applied; from this, it produced a web presentation in dynamic HTML. **Figure 3** shows the original AnnAnn:



To finish this example off, we'd like to chain the output expressions like `cout << "hello" << mummy << 3*4;` To achieve this, we simply have to make the method return a reference to the ostream that it used.

```
#include <iostream.h>

enum bears {mummy, daddy, baby};

ostream& operator<<(ostream& out, bears b){
 switch(b){
 case mummy: out << "mummy"; break;
 case daddy: out << "daddy"; break;
 case baby: out << "baby"; break;
 default: out << "unknown"; break;
 return out;
 }

 }

int main(int argc, char *argv[]){
 bears b;
 b=daddy;
 cout << "This is a " ;
 cout << b;
 cout <<  " bear.\n";
 }
```

Figure 3 : The Original AnnAnn in use

AnnAnn proved an excellent tool and an intelligent approach to the difficult task of educating students in complex technical theory. However, its uptake was somewhat mired by complexities of the program itself. Ironically it helped the students learn, but hindered lecturers preparing of the learning material.

For these reasons, an evolution to AnnAnn was proposed, a version that kept the functionality of the original system, but was otherwise a redesign, from concept and content upwards. Due to the network centric nature of the redevelopment this implementation became known as AnnAnn.Net.

3.1 Simplification

AnnAnn.Net captured the strengths of the original system, and incorporated them into a completely different solution. The shortcomings of the AnnAnn system were used to shape the new platform, and identify the key design decisions of the platform.

The major differences between AnnAnn and AnnAnn.Net are:

- Production of a single output file, rather than a multitude of HTML documents.
- Users are relieved of the technical burden. AnnAnn.Net generates the change file itself, so users need not learn a new scripting language.
- A web-oriented structure: AnnAnn.Net is a simple, customizable web service, defined by a strict web service API.
- Support for multiple output formats: the rendering engine currently supports XHTML and RSS, and is designed such that it can rapidly be extended. Minimal development would be required to develop further renders for Flash, PowerPoint or any other conceivable popular or future presentation format.

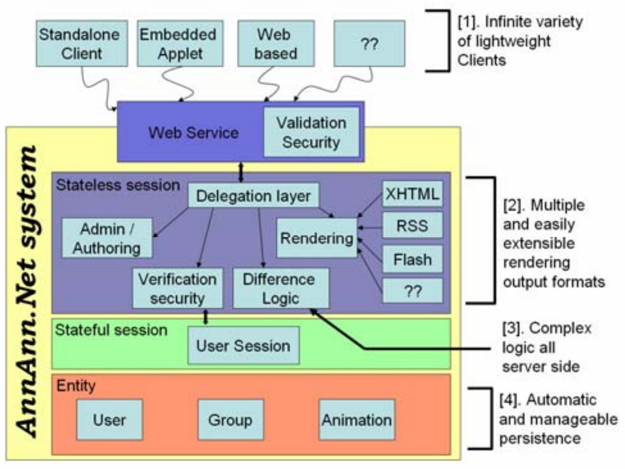


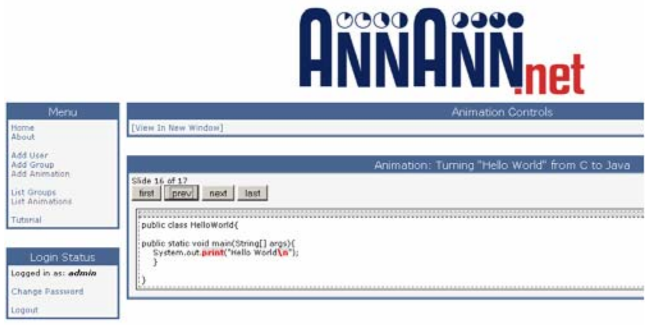
Figure 4: Layered Design of AnnAnn.Net

As can be seen in **Figure 4**, the AnnAnn.Net system is structured around the principle of extensibility of server side components (especially the rendering engine and exporting technology), and the customizable nature of the clients, made possible due to limited functional requirements.

3.2 A Client for Every Purpose

The AnnAnn system is designed not just for the purpose of teaching programming, but rather any subject matter that is well illustrated when broken into small steps. For this reason AnnAnn.Net is designed to fully support the lightweight client, offering all the complicated processing as server side functionality in order to allow the design of the client to be fully focused on HCI issues for the target group.

This means that, by using the 'web service' API for AnnAnn.Net, any developer can rapidly construct a front end client for the system, tailoring layout, format and platform functionality to the exact target group. **Figure 5** shows one such example, developed for the original testing of the AnnAnn.Net system.



Logged in as: *admin*

Animation: Turning "Hello World" from C to Java

Slide 16 of 17

```
public class HelloWorld{

public static void main(String[] args){
   System.out.print("Hello World\n");
   }

}
```

Figure 5: Example Client, developed on PHP

3.3 LOM and Reuse

Another feature being built into the AnnAnn.Net system is the IEEE's Learning Object Metadata (LOM) standard [17]. LOM allows the sharing of Learning Objects (LOs) on a far broader scale than departmental. AnnAnn.Net animations will be semantically enabled and part of a globally pooled learning resource. It is of note that LOs are particularly appropriate for the teaching of IT [18], as the field is both dynamic and young; as such, suitable textbooks are not always available. Online, accessible materials compete well.

Mark-up will provide accurate guidelines as to applicability, difficulty and required skills as well as ownership, authoring and usage.

It is a goal to automate collection of this metadata: for example, the author and creation date of a LO can be elicited by examining the logged in user and system timestamp. Fixed categories will be used, drawn from the IEEE LOM Standard [17]: these include 'description' tags, enabling authors some flexibility in their mark-up. It is essential to ensure that these packaged LOs are optimized for reuse and repurposing [19]. A related task is to build an AnnAnn.Net client which allows access to and editing of this metadata.

By building such standards into AnnAnn, replication of teaching materials can be reduced, whilst shared knowledge will promote a higher quality of teaching objects and reference materials. Eventually users may, in addition to contributing material, search for and customize animations (building sequences to create a program of study). AnnAnn.Net contributes towards a more open, yet still secured, teaching domain designed to encourage utilization of the best features of web based communication which will allow students to benefit in their instruction from a global wealth of knowledge.

4. The Educational Perspective

Taking an educational perspective on the pedagogic appropriateness of various approaches to programming we find some examples of approaches which map to educational theory. However, in the most part Lemos' 1979 observation that "most of the literature consists of subjective opinions on the most effective methods of instruction for a given programming language" [9] still holds true.

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 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 led some, for example Callear [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 realized 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– artifacts 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 artifacts 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 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.

5. Concluding Remarks

We have described AnnAnn, a series of tools to assist students to understand programs and we have described its use. We have explained the reasons why we developed the tools, and justified the educational frameworks within which we believe they sit.

In practice we have found two distinct modes in which we use these tools. 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 them to be useful as tools 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 online examples and to the development of the tools.

ACKNOWLEDGMENTS

Our thanks go to Ben Dowling, Andrew Lewis and Gavin Willingham for researching, designing, developing and testing the AnnAnn.Net platform, upon which one of the latter sections of this document is based.

REFERENCES

1. J.T. Mayes, “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.
2. S. 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 pp171—174. ACM Press 1999.
3. B.S. Bloom, "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. M. Ben-Ari. “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. H.M. Deitel and P.J. Deitel, *Java How to Program*. Prentice Hall. 1997.
6. D. E. Knuth, “Literate Programming (CSLI Lecture Notes, no. 27.)” Stanford, California: Center for the Study of Language and Information, 1992), xvi+368pp.ISBN 0-93707380-6.
7. S. Shum and C. Cook, “Using literate programming to teach good programming practices”, ACM SIGCSE Bulletin, v.26 n.1, pp.66-70, March 1994.
8. AnnAnn Web Site http://www.annann.org/ last accessed January 2006.
9. R.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. D. Callear, “Teaching Programming: Some Lessons from Prolog”, In the Proceedings of the LTSN-ICS 1st Annual Conference, Heriot Watt, 2000.
11. H. Zhu & M. Zhou, “Methodology First and Language Second: A Way to Teach Object Oriented Programming”, ACM OOPSLA ’03, Anaheim Ca. 2003.
12. R. Duke, E. Salzman, J. Burmeister, J. Poon, L. Murray, “Teaching programming to beginners – choosing the language is just the first step”, Proceedings of the Australasian conference on Computing education, December (2000).
13. H.C. Davis, L.A. Carr, E.C. Cooke, & S.A. White, “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. A. Collins, 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. Hillsdale, NJ, Erlbaum.
16. R. Baecker, “Two systems which produce animated representations of the execution of computer programs”, ACM SIGCSE Bulletin, 7 (1975), pp 158 - 167.
17. IEEE Learning Technology Standards Committee, Working Group 12 (Chair: Hodgins W.), Learning Object Metadata, http://ltsc.ieee.org/wg12/
18. K. Abernethy, K. Treu, G. Piegari, and H. Reichgelt. 2005. “A learning object repository in support of introductory IT courses”. In Proceedings of the 6th Conference on information Technology Education (Newark, NJ, USA, October 20 - 22, 2005). SIGITE '05. ACM Press, New York, NY, 223-227.
19. T. Boyle (2002). “Design principles for authoring dynamic, reusable learning objects”. In A. Williamson, C. Gunn, A. Young and T. Clear (Eds), Winds of Change in the Sea of Learning: Proceedings of the 19th Annual Conference of the Australasian Society for Computers in Learning in Tertiary Education, pp57-64. Auckland, New Zealand: UNITEC Institute of Technology.