AnnAnn and AnnAnn.Net : Tools for Teaching Programming

Clare J Hooper, Leslie A Carr, Hugh C Davis, David E Millard, Su A White, and Gary B Wills
University of Southampton, Southampton, UK
Email: (lac, hcd, dem, saw, gbw} @ ecs.soton.ac.uk

***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, making it difficult for students to understand how it was developed, or where to start. This paper presents AnnAnn.Net, an animated code annotator which makes it possible to present the incremental development of code to large groups or for self study. The tool is designed for ease of use by both lecturers and students. The implementation and the ration for which are described in detail. The design of the system is underpinned by a sound pedagogical approach and these are discussed, along with the educational benefits of this approach are examined.**

***Index Terms*— Learning to Program, literate programming, cognitive apprenticeship, scaffolding, constructivist learning**

I. 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]. Achieving 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 in lecture halls, or we must ask them to conduct their studies alone.

Presenting programs to large groups is difficult. 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 and AnnAnn.Net – successive versions of 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.

II. 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.

A. 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. E.g. to show:

(i) the class outline including constructor
(ii) how main method creates an instance of this class
(iii) how events are delegated to the event handler

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

B. 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.

Figure 1 PowerPoint Slide

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

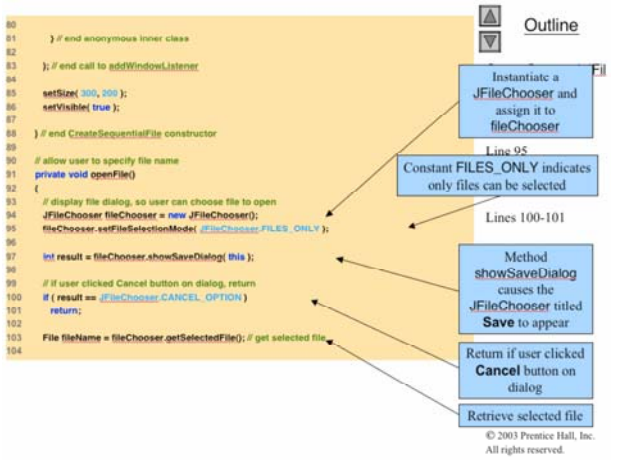


Figure 2 Text Book Figure

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

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.

C. Textbook Layout

A related approach is one commonly used in textbooks, reproducing the listing as a figure (as in, 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.

D. Literate Programming

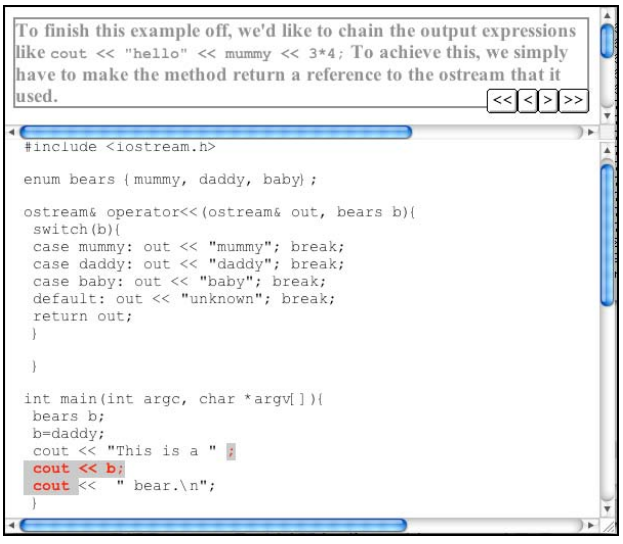
Knuth developed Literate Programming [6] as a way of mixing documentation and code. It allows the programmer to develop very sophisticated explanations which break up the standard program ordering and interleave it with TeX or troff documentation commands (the source program and document are derived 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.

III. AnnAnn

AnnAnn is a simple documentation system that embodies a constructive explanation paradigm. The lecturer may work from a familiar starting point, and show (and explain) 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. The rather terse syntax (similar to the UNIX patch command) allowed the author to create blocks of micro-explanation.

The AnnAnn compiler takes an original file and a list of changes to be applied over time, and produces a Web presentation in Dynamic HTML. An extract from an AnnAnn presentation is shown in Figure 3.

Figure 3: The Original AnnAnn in use



The aim of an AnnAnn explanation is to start with a familiar program (typically a Hello World style program, applet or JFrame) and by applying successive small changes (adding and initializing an array, fleshing out a for loop, creating a user interface object, etc.) 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, and so on.

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 3 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. 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.

IV. ANNANN.NET

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 its complexities: it helped students learn, but hindered lecturers preparing the learning material.

It was for these reasons that a new version was proposed, to keep the motivation of the original system, but otherwise be a complete redesign. This project became known as AnnAnn.Net, due to the network-centric nature that the redevelopment took.

A. Simplification

In order to help the AnnAnn format reach its full potential, the development of AnnAnn.Net took the approach of capturing the strengths of the original system, and incorporating them into a completely different solution. The criticisms of the AnnAnn system were drawn upon to shape the new platform, helping motivate and shape its development.

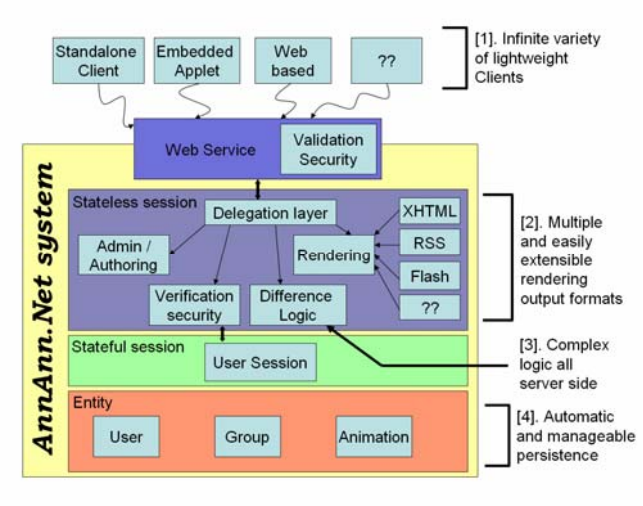
The first important decision was to relieve the user of a technical burden. Lecturers simply don't have the time, or necessarily the technical skill, to learn a new scripting language to create slides. Tools such as PowerPoint are successful due to the simplicity and speed by which material can be constructed. Although an experienced user can rapidly assemble AnnAnn animations, the learning curve is very steep, and even at the summit the cognitive load for creating an animation is considerable. Therefore the scripting language was abstracted away from the user, being replaced by a more intuitive and higher level method of interaction and input.

A further lesson drawn from the evaluation of the AnnAnn system was about the tool's operation. The AnnAnn system requires an original text file, a 'changes' script file and a compiler. This then generates a large series of HTML documents (more than 100 for a simple for loop), in the same directory as the AnnAnn compiler. It also has to be run from the command line, with a varying range of parameters and flags, which can complicate matters. Given the wide, and expanding, range of presentation formats AnnAnn needs to be extensible, but it also has to be more streamlined (generating fewer files) and easier to use.

B. Web orientated Structure

AnnAnn.Net was designed to alleviate criticisms of the original AnnAnn system, by being designed with the user as principle concern. In order to achieve this, the system had to become widely accessible, easy to use, fast, and customizable. The logical solution to accessibility was a web based solution. To make it simple for the user, as well as customizable, the functionality had to be server side, and ideally modular.

Figure 4: Layered Design of AnnAnn.Net

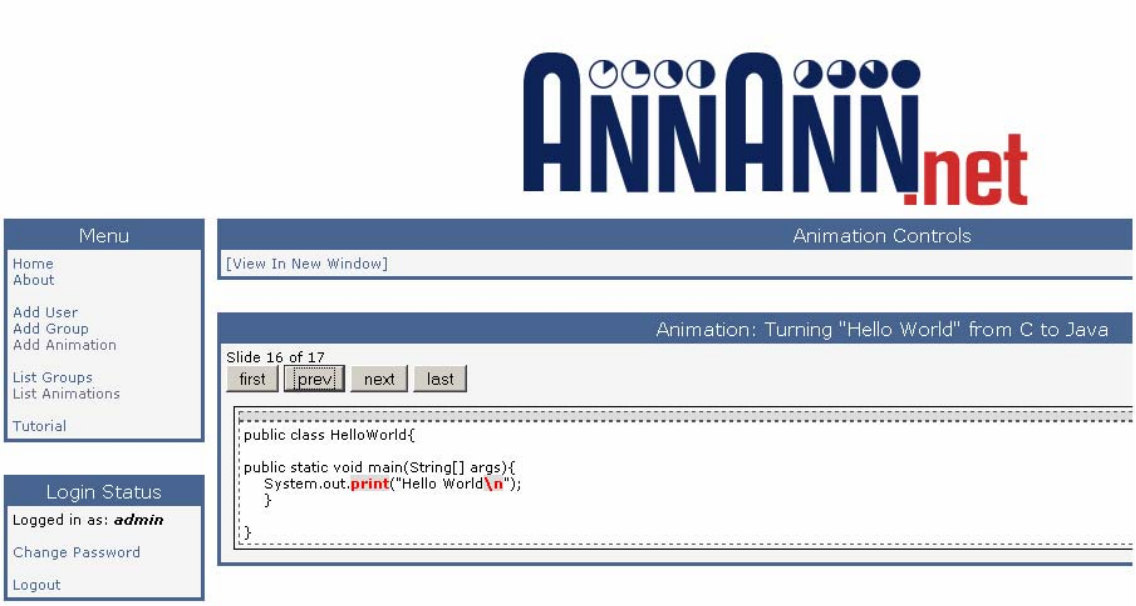


Thus the AnnAnn.Net system is well defined via a web service API, offering authoring, security and rendering of animations, in addition to handling the document differencing which was originally performed by the changes script. As shown in Figure 4, AnnAnn.Net is structured around the principle of extensible server side components (especially the rendering engine and exporting technology), and the customizable nature of the clients, made possible due to limited functional requirements.

The rendering engine currently supports XHTML and RSS output formats. The system was designed such that it can rapidly be extended with new rendering components. Minimal development would be required to develop renders for Flash, PowerPoint or any other popular or future presentation format.

The change script was rewritten into a server-side differencing algorithm, loosely based on the UNIX 'diff' algorithm; this moved traditionally client-based functionality to the server, facilitating the concept of a lightweight thin client structure

Figure 5 Example client, developed in PHP



C. A Client for Every Purpose

AnnAnn is designed not just for 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, such that client design may be fully focused on HCI issues for the target group.

Figure 6 The first slide of the animation

Figure 7 Slide three: the inserted text is highlighted

Figure 8 The final slide of the animation

This means that, by using the AnnAnn.Net 'webservice' API, any developer can rapidly construct a front end client for the system, with layout, format and platform functionality tailored to the exact target group.

Figure 5 shows one such example, developed for the original testing of the AnnAnn.Net system.

The client supports creation, editing and viewing of animations. Figure 6Figure 8 show a simple animation in the client that demonstrates how a link can be added to a HTML page.

D. LOM and Reuse

A feature being built into AnnAnn.Net is the Learning Object Metadata (LOM) standard [17]. This annotates animations, allowing sharing in a global learning resource pool, with accurate information on applicability, difficulty and required skills as well as ownership, authoring and usage.

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, granting authors some flexibility in their mark-up. It is essential to ensure that these packaged LOs are optimized for reuse and repurposing [18]. 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 is a step towards a more open, yet still secure, teaching domain designed to encourage utilization of the best features of web based communication to allow students to benefit from a global wealth of knowledge in their instruction.

V. AnnAnn AND AnnAnn.NET COMPARISON

The AnnAnn.Net tool aimed to bring the advantages of AnnAnn style animations to a much wider range of users, through a simpler and more accessible design than the original. To evaluate the tool, it was decided to compare it with AnnAnn in a controlled experiment, where the only independent variable was the tool being used.

Twenty students from various disciplines and year groups were evenly split into two random groups. Each user was given a set of ten animations to later develop, each taking up to one A4 side of paper. After an initial five minute familiarization for subjects to look through the animations and ask questions, ten minutes were given for each group to get acquainted with their first development tool. Over twenty minutes, subjects worked on completing as many animations as possible using the first tool. Next, the order of the animations was changed, and the groups swapped tools, with a second ten minute familiarization session, and a second test. Finally, a questionnaire was given out. The results are summarized in Table 1.

Clearly, the AnnAnn.Net tool was found to be easier to use, with usage compared to simple data entry, as opposed to usage of AnnAnn, which was compared to programming (as one participant remarked, 'typing speeding had a lot to do with how fast you could do the tasks with AnnAnn.Net'). Only six AnnAnn animations were completed over the two 20-minute sessions. Given this, it is unsurprising that all users selected AnnAnn.Net as the preferred system; its simpler nature appealed to the whole domain of technical expertise.

Number animations created in…	**AnnAnn**	**AnnAnn.Net**
<= 10 minutes	1	33
> 10 minutes	5	9

Table 1 Time taken to complete the tasks

We found that 25% of users expressed frustration with the AnnAnn system. Our concern was that the documentation for that system was so poor as to bias the experiment towards AnnAnn.Net (one suggested the AnnAnn tutorial was aimed at those with 'prior knowledge and experience in programming'). In terms of a real world context, however, the data is valid, as the two tutorials in question gave very similar instruction to the relevant help available online for each tool. The fact that so many users struggled with AnnAnn reflects the low uptake, probably due to the technology being difficult to use and poorly documented for people who aren't highly technical.

During the experiment the AnnAnn.Net system was subject to ten users constantly developing at any one time; no comments were made about speed in any questionnaire, or indeed during the experiment at all: this shows that the system scales effectively to this level, paving the way for a full scalability test.

VI. 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.

VII. Comparison: AnnAnn.Net and Alternatives

University professors are continuing seeking ways to make teaching programming easier; this is usually involving a graphical approach [26]. It is commonly acknowledged that it is difficult to learn how to program [25] [29]: an approach is required that is flexible to the students needs. AnnAnn.Net, like AnnAnn embodies a constructivist explanation paradigm. Both are built as scaffolding tools: artifacts to hide some of the problem complexities, so that students may be aware of the big picture and achieve the major goal of the exercise [24]. This is done by allowing lecturers to present each section of the solution only as it is needed. Chalk observed that scaffolding would appear to be necessary when teaching programming, due to the complexity of the task [21].

There are similar tools used to teach programming. One example is JEWL [23]: however, this tool is used to help students actively build code (by providing a GUI library), rather than to help students follow a particular explanation of a concept. Codewitz [20]] is another example of software which allows the sharing of learning objects. It differs from AnnAnn.Net in that it is used to create worksheets and exercises, more than to create explanatory animations.

By contrast, the work of Culwin, Campbell and Adeboye [23] developed a tool which teaches Java programming through 'scaffolding'. However, as the student cannot work without the tool, it may be considered a 'skeleton' and not a 'scaffold', as AnnAnn.Net is. Additionally, Culwin's tool is limited to teaching the Java language only. There's no agreement on how to best teach any specific coding language [[30]], and AnnAnn.Net leaves the specifics of language and program type to the lecturer.

OOP-Anim [27] is a system designed to support the teaching of basic object oriented (OO) programming concepts; again, an e-learning tool, but different from AnnAnn.Net in two ways. Firstly, it is specific to OO programming, and secondly, it is not a lecture-based tool, but rather a tool for students to use directly.

As shown by Price, Baecker and Small's taxonomy [28], there exist many approaches to the problem of software visualization, and many problems remain, including scope and scalability: AnnAnn.Net has not been tested with large (hundreds-of-lines) programs, because it's intended usage as a lecturer's tool: a 45 minute lecture cannot cover a program of such magnitude. Future uses of AnnAnn.Net may be larger scale, but for the immediate future, efforts were best expended elsewhere.

AnnAnn.Net makes use of XML and Web Services. The power and ease of use of XML have been noted by Coyle, as has the importance of Web Services [22]: use of XML allows the Web Services to be utilized, and also provides future compatibility with semantic applications. Web services provide services to software, and thus AnnAnn.Net can be used by anyone with an internet connection.

VIII. Concluding Remarks

We have described the AnnAnn tools, which assist students in understanding programs, and we have described their use. We have explained 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.

ACKNOWLEDGMENTS

Our thanks go to Ben Dowling, Timothy Griffith, 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 May 2007.

[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.

[20] Kujansuu E., “Codewitz – An International Project for Better Programming Skills”, *Learning Objects for Computing workshop*, London Metropolitan University, UK, Learning Technology Research Institute (2004)

[21] Chalk P., “Community of Practice: learning the craft of programming”, in *Proceedings of the 2nd LTSN-ICS one day conference on the Teaching of Programming*, Wolverhampton, UK (2002)

[22] Coyle F., “Breathing Life into Legacy”, *IT Professional*, Volume 3 Issue 5 (2001) pp 17 - 24

[23] Culwin F., Campbell P., Adeboye K., “A Bridging, Scaffolding or Skeletal Initial OOSD Learning Object”, *LTSN-ICS 5th Annual Conference*, Ulster, UK (2004)

[24] Hogan, P., *Scaffolding student learning instructional approaches and issues*, Cambridge Brookline Books (1997)

[25] Jenkins, T. “On the Difficulty of Learning to Program”, in *Proceedings of 3rd Annual Conference of the LTSN-ICS*, Loughborough, UK (2002)

[26] Jones, R., Boyle, T. and Pickard, P., “Object World: Helping Novice Programmers to Succeed Through a Graphical Object-First Approach”, in *Proceedings for the 4th Annual Conference of the LTSN-ICS*, Galway, Ireland (2003)

[27] Esteves M., Mendes A., “OOP-Anim, a System to Support Learning of Basic Object Oriented Programming Concepts”, in *Proceedings of the 4th International Conference Conference on Computer Systems and Technologies: e-Learning*, Rousse, Bulgaria (2003) pp 573 - 579.

[28] Price B., Baecker R., Small I., “A Principled Taxonomy of Software Visualization”, *Journal of Visual Languages and Computing* 4 (1993) pp 211 - 266

[29] Sayers H., Nicell M., Hagan S., “Teaching Java Programming: Determining the Needs of First Year Students”, in *Proceedings for the 4th Annual Conference of the LTSN-ICS*, Galway, Ireland (2003)

[30] Lemos R., “Teaching programming languages: A survey of approaches”, ACM SIGCSE Bulletin, *Proceedings of the 10th SIGCSE technical symposium on Computer Science Education*, Volume 11 Issue 1 (Jan. 1979)

Clare Hooper (UK, 1983) completed an MEng Computer Science at the University of Southampton, UK, in 2006. She is now pursuing an EngD, also at the University of Southampton, looking at the pervasive technologies for learning and accessibility.

Previous work includes roles as an Extreme Blue Intern and as a Research Assistant. Publications cover undergraduate work, including her third year dissertation (on narrative pace in hyperfiction) and fourth year work on AnnAnn.Net, and computer mediated augmentation of workplace 'break' spaces.

Ms. Hooper is a student member of the BCS. Details of her publications may be found at http://eprints.ecs.soton.ac.uk/

Leslie A Carr is a Senior Lecturer in the Intelligence, Agents, Multimedia at the University of Southampton. Since the 1980's he has experimented with multimedia information systems, novel ways of constructing hypertexts, digital libraries and knowledge management systems.

David E Millard (UK, 1976) obtained a Bachelor of Science BSc (Hons) in Computer Science in 1997, and a PhD in Contextual Information Systems in 2000, both from the School of Electronics and Computer Science at the University of Southampton. He is now a Lecturer in Computer Science within the Learning Societies Lab at the University. His research interests include adaptive, contextual and narrative information systems, pervasive and mobile learning, social computing and the human knowledge interface.

A full listing of his publications can be obtained from http://eprints.ecs.soton.ac.uk/view/person/1615.html

Hugh C Davis is Director of e-Learning and Head of the Learning Societies Lab at the University of Southampton. His research interests lie in the ways in which technology can improve the learning experience, particularly in a research-led learning and teaching environment.

Su A White is Learning and Teaching Co-ordinator for The Faculty of Engineering and Applied Science at the University of Southampton. She is a graduate of the London School of Economics, a qualified journalist, lecturer and computer scientist. Her research interests concern the ways of using technologies in learning and teaching and also in the inter-relationship between organisational structures and change

Gary B Wills (UK, 1962) obtained a Bachelor of Engineering BEng (Hons) from the University of Southampton, UK in 1995 and a PhD also from Southampton in 2000 for his research into the Design and Evaluation of Industrial Hypermedia systems.

He is a Lecturer in Computer Science at the University of Southampton. His main research interests are Personal Information Environments (PIEs) and their application (usability) in industry, medicine and education. PIE systems are underpinned by distributed adaptive hypermedia, and advanced knowledge technologies. Recent projects include a Virtual Research Environments for Orthopaedics using a distributed architecture to allow collaboration, discussion and dissemination of results for surgeons conducting clinical trials and who are not co-located. Also in the industrial domain, a project allowing engineers to view the same federate knowledge from different perspectives. This uses advance knowledge technologies and ontologies to extract the knowledge and appropriately present this to the users.

Dr Wills is a Chartered Engineer and a member of the IET. A full listing of his publication can be obtained from http://eprints.ecs.soton.ac.uk/