The Development Process for Courseware Material: A Computing Methodology Approach

Andrew Hunter and **Ainslie Ellis**
School of Network Computing
Monash University, AUSTRALIA
Andrew.Hunter@infotech.monash.edu.au
Ainslie.Ellis@infotech.monash.edu.au

Abstract

At a university level, the conceptual stage of computer-facilitated learning (CFL) tends to be a faculty wide forum with ideas and inputs coming from many sources (e.g. academics, support staff, students). At the department level however, it is often a single academic who develops these ideas and attempts to satisfy the requirements. These requirements necessitate a full pedagogic understanding. Whilst it is suggested that it is the norm to utilise teams to develop programs and on line subjects, experience in the computing faculty shows that individuals often take, or are given responsibility for the development of their own subjects. It has been observed that computing faculty staff are very familiar with the technology and tend to adopt computing methodologies to develop these programs with pedagogical theory being added to the program as "user requirements". This paper investigates various methodologies currently employed for computing and some of their associated restrictions and how they differ with similar educational methodologies. It investigates ways in which a methodology could allow developers without a vast educational background to create pedagogically sound programs.

Keywords

Development, Software design, Computing, Education, Methodology, Team approach

Methodologies

A methodology is a set of methods that define the process and order of how something is to be achieved. Reeves (1992) believes there has been a paradigm shift from instructionist to constructivist and that this has, in turn, reflected a change in methodologies from breaking down the project into single step tasks to a more holistic approach. In computing, differing methodologies focus on different perspectives. These include:

- Data-oriented techniques that focus on the organisation's data and views the data independently of how it is to be used.
- Process-oriented techniques that are based on the "Input – Process – Output (IPO) concept. These techniques build models of systems based on studying the processes of a business.
- Object-oriented and component-oriented techniques.
- Other construction techniques that incorporate design patterns.

When trying to design educational programs using a traditional linear computing methodology such as the waterfall model the design process may become unclear. When designing programs for education a less sequential methodology allowing for more feedback at all stages of the project is needed. The preferred method for education is the prototype, or a hybrid mix of prototype and an iterative process.

What are the Various Computing Methodologies?

The Classic Waterfall Model

One of the first and well known methodologies is James Martin's (Beynon-Davies 1993) Waterfall model. It is usually seen as a natural progression of steps as shown in Figure 1. The Waterfall model attempts to separate the life cycle into discrete activities by creating a linear series of actions. One misconception is that the model always moves in a single direction. The modeling process does in fact tend to be iterative, looping back and forward between steps.

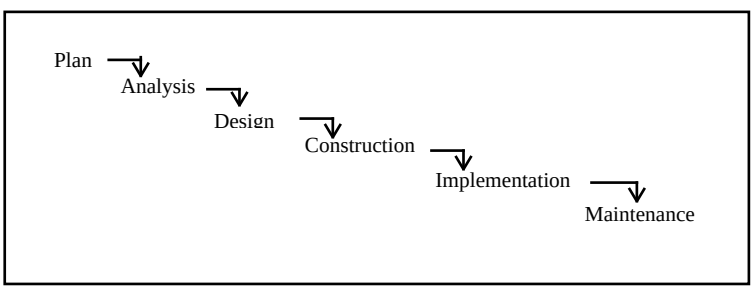

Figure 1: The Classic Waterfall Model

Pfleeger (1991) extended the Waterfall model by placing a larger emphasis on the testing component. In this model all steps are interconnected to allow a prototyping approach to be used in conjunction with the model. Unfortunately neither of these models address Human-Computer Interaction (HCI) issues.

Prototyping

The current literature on prototyping methods shows a disjoint between the way the methodology is currently employed across differing disciplines. This has led to confusion when trying to combine computing and educational facets into a single set of methods. Prototyping allows a sense of ownership for the end users but creates further problems of project creep. Often small changes with each iteration leads to a "chop and change" approach in the coding that may become hard to follow.

Prototyping (Computing)

The aim of prototyping is to enable input from the end user at an early stage by giving them the look and feel of the application. This is achieved by modeling the user interface whilst having little or no content behind that interface.

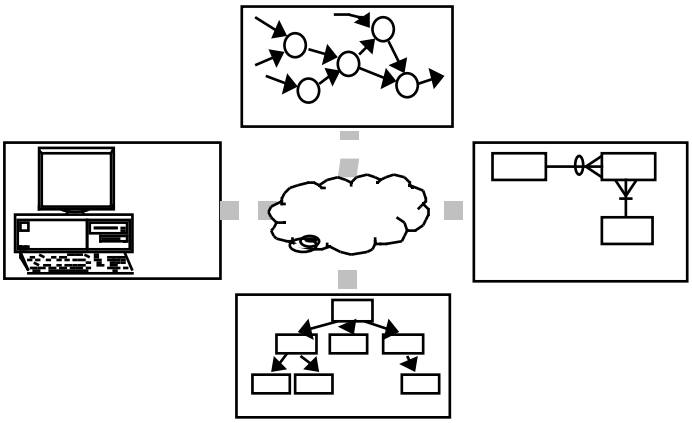

Figure 2 Four Dimensional Rapid Prototyping Model – (Connell & Shafer 1989, p.11)

Figure 2 shows the dimensions included in the rapid prototyping model. Early input from the user has the potential to highlight problems early in the design phase, thus enabling large cost savings (Freeman 1999, & Capper 1999). Rapid prototyping does not, however, dramatically shorten the development time of a project (Connell & Shafer1989) as the initial shell has to have the functionality added at a later time.

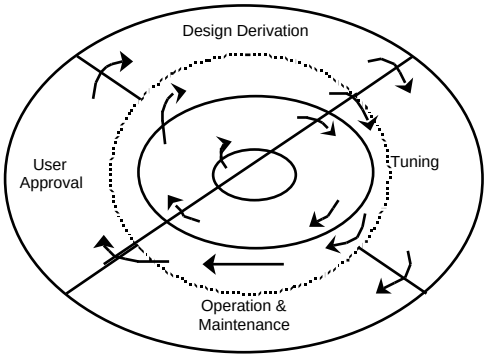

Figure 3 A non-evolutionary approach to prototyping (Connell & Shafer 1989, p. 272)

The non evolutionary model as shown in Figure 3 can be seen as less powerful, as little more than a menu structure is shown to the user. Software engineering regards the production of software as an engineering problem hence a rigid model such as this will allow a much more accurate and feasible design than might otherwise be possible. Within the computing industry the prototyping methodology uses little or no iteration. It is designed to present the user with what appears to be a working system. It is, however, only an interface with little functionality - typically a set of menus to ascertain the correctness of the design. This can be done either by working with the end user directly or by a series of iterations in this early design stage. Once this has been achieved and the user is happy with the design, the functionality is then coded. A problem here is that the user may not recognise that they are seeing a shell and have expectations that the project is near completion.

Prototyping (courseware)

There are many hybrid extensions of the prototyping model. Carroll (1995) believes the incorporation of scenario based design helps recognise the tasks that are involved. Iteration can also be introduced throughout the whole operation as shown in Fgure 4.

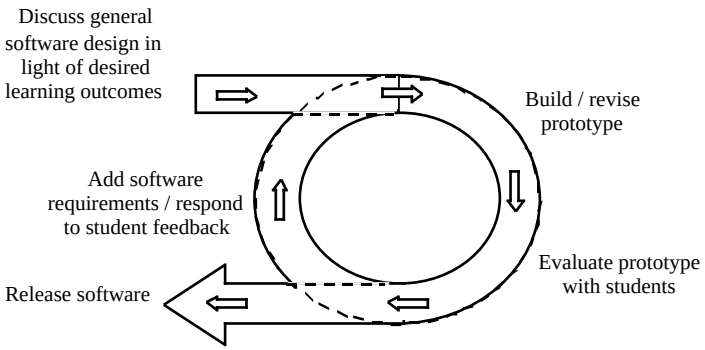

Figure 4. Educational Prototype with student involvement
(Pressman 1997, p. 3 as cited in Kennedy, 1998, p. 379)

This incremental development model involves developing the requirements and delivering the system incrementally. This allows the user to provide feedback for the latter parts of the program prior to their

implementation. This iterative model provides the advantages of evolutionary prototyping, which gives evaluative feedback throughout the development process whilst maintaining a control over the direction of the project. One of the dangers here is that unless the requirements are clearly defined they can tend to change direction with each iteration.

Other courseware methodologies

The Pragmatic model of Carswell and Murphy (1995) sets out a five stage courseware methodology as shown in Figure 4. This model was adapted from three other methodologies, that of Barker and Yeates (1984), Alessi's 8 stages of lesson design (Alessi 1991) and the evaluative features of Duschastel (1993 as cited in Carswell and Murphy 1995) It was designed primarily due to the lack of non-complex methodologies that were available for use for students at tertiary level.

Alessi & Trollip(1991) recommended a more detailed development scheme. Their model has ten steps for the development of a single lesson. They also believe it is important that the programmer understands the overall objectives, even if working in a team with others who know them. They stress the importance of sequencing events and recommend the use of storyboarding to facilitate pilot testing on learners. Evaluation and revision based on principles of cognitive psychology: perception and attention, memory, comprehension, active learning, motivation, locus of control, transfer of learning and individual differences should be conducted at several points throughout the development process. Bradler (1999) recommends a twelve stage approach similar to that of Alessi & Trollip but, in addition, their approach includes prototyping of key features and an attempt to identify reusable components.

Bostock and Drummond developed a model in 1995 that has since been modified by Bostock in 1996 (Bostock 1996). They also recommend the use of prototyping but feel it is important not to use technology for the sake of it and all alternatives should be looked at prior to making the decision to proceed. They stress the need to analyse the requirements and to describe the deliverables for each phase of the development cycle.

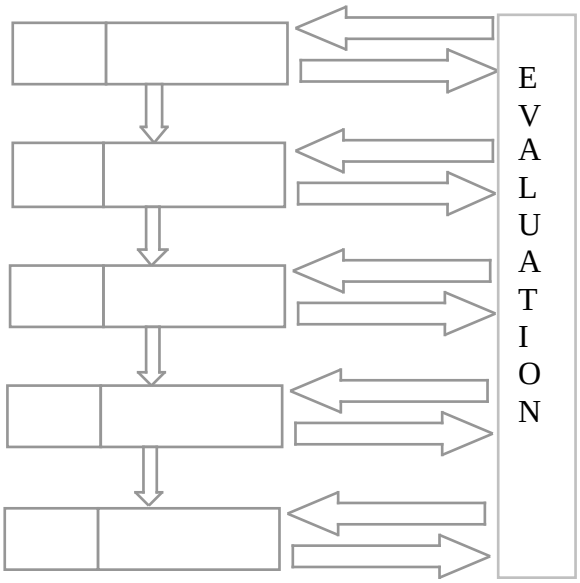

Figure 5 The Pragmatic Model (Carswell, L., Murphy, M. 1995, p 99).

Methodology Issues: Problems of Adaptation for Education

Computing methodologies tend to have an evaluation at the early analysis stage and testing as a final stage. Educational or courseware methodologies tend to have evaluation and feedback done at each stage. This is one possible reason for the differing views of prototyping with little iteration in the computing model, contrasted with the model used in education where a continual iterative approach is employed until all functionality has been achieved. Computing people tend to follow a methodology like a recipe. Most computing methodologies are very well established and documented. Education on the other hand is far more flexible and more often each step will depend on the result of the previous step. Theoretical concepts although well established require too much flexibility to become steps to be followed within a methodology.

Many of the courseware methodologies were adapted from the traditional Waterfall model. While Bostock (1996) includes prototyping, he does not incorporate it within a typical prototyping methodology but within the Waterfall model. This is perhaps because the Waterfall model is the easiest to follow and the steps are well defined, theoretically making it easy to incorporate educational steps. Unfortunately this model is a top down, sequential approach that is data driven. The analysis phase looks at 'what' will be done, with the design concerned with 'how'. As a result,

the instructional design component tends to focus only on the design phase where the process issues are addressed. This causes questions such as 'why we want the courseware' or 'what alternatives could be implemented' to be often overlooked.

Alternatively, Bostock (1998) describes courseware engineering as the mixture of two well established disciplines, software engineering as the process of developing business applications and instructional design as the process of developing instruction. The first question that needs to be answered is can we run the two methodologies in parallel or do we need to combine them into one coherent set of steps? Many developers (Riley 1995; Bostock 1996) believe that designing courseware is a team process and the software engineering component and the instructional design are separate tasks to be completed by different people each with their own expertise. Software developers believe understanding the motivations behind the request for software and the knowledge of how to successfully address the requirements to satisfy these motivations would require an understanding of learning processes and their outcomes (Pfleeger, 1991; Alessi & Trollip, 1991).

Muller, Wildman and White (1993). advocate a participatory design approach and see the participation and communication as more important than the methodology itself, while yet another group (Kensing et.al.1993, Carroll and Moran 1991) believes that using methodologies create many problems, with Carroll proposing a scenario based method to aid design. Brown (1997) concludes that current methodologies are inadequate for the use of courseware, or any highly HCI intensive projects.

Problems with adapting data driven versus process driven systems

Shah and Bonner (2000) believe the effectiveness of methodologies needs to be determined through metrics and benchmarking. It is important to remember that a methodology that has been used to create one successful application may not always be appropriate for another. The software engineering approach is a top down data driven approach, with the analysis phase deciding what is to be done and the design phase determining how. It is in the transition from the requirements analysis to that of design where the IT focus shifts from 'what' to 'how'. When we merge computing and educational methodologies, the computing part of the methodology is spelt out but the education part is usually skimmed over as though a presumption is made that the reader has an in-depth knowledge of pedagogy and learning issues. An example of this is the

pragmatic model of Carswell and Murphy (1995) that sets out defined formal steps for naïve users to follow. The model relies heavily on quality feedback and is designed to iterate within each stage. It assumes, however, that, on completion of a stage, the user will proceed to the next stage. Once again the model sets in concrete the processes of the computing side and leaves the more complex issues of the learning and educational theories to an assumed knowledge.

Some object oriented methodologies focus on the process driven perspective by looking at objects along with their actions as a single perspective rather than data and the processes as separate entities (Eriksson and Penker 1998). This may explain why Bostock (1998) believes the translation of analysed-needs to design-functions is the hardest part of instructional development. Bottom up or middle out approaches such as iterative or evolving prototyping methods are the newer generation of methodologies and are generally seen to add a quality component by raising satisfaction with evaluative feedback. They also allow changes to be introduced as more functionality is added to the project.

Findings

Marrying traditional computing methodologies with educational theory is somewhat like trying to fit a square peg in a round hole. The current linear and sequential models that predominate are the least able to cope with emerging constructivist theories. This is primarily due to the lack of evaluation and feedback throughout the development cycle. Iterative models that encourage continual user feedback and evaluation offer a pedagogical perspective for software development well in line with a constructivist perspective (Kennedy 1998)

Prototyping is especially valuable where requirements cannot be specified clearly. Its use is ideal for the development of interactive multimedia. Witt and Wager (1994) recommend prototyping for the development of Electronic Performance Support Systems (just-in-time training), but warn the term "prototyping" should not be misused to mean client evaluation

Whilst a prototype can give end-users a view of the system capabilities when establishing and validating system requirements, there are issues that need to be carefully monitored. With evolutionary prototyping the system may become corrupted by constant change. When using an incremental approach, latter changes to the system become increasingly difficult to make, as the first part of the system to be developed is the least evolved at development time.
The main disadvantage is that it is harder to control the scope of the project as early feedback often leads to changes in the design of the system and this may happen several times through iteration.

Conclusion: Where to Now?

It is important to be aware of the multiple range of skills required when choosing a methodology. One that models the processes rather than the data should be chosen. Rapid prototyping is developing as the favorite for educational purposes and it is one that can be conducted in parallel between education and computing; however, the definitions and techniques need to be carefully defined so that teams working on a project know exactly what is required of them.

There are many inherent complexities involved in the development of educational products. It is generally accepted that the former systematic approach based on instructional design principles employed in the instructionist approach have been replaced with constructivism views. Boyle (1997) believes this is partially due to advancements in technology (e.g. hypertext) and educational design. The constructivism view incorporates knowledge and skills across many disciplines such as cognitive and perception psychology, evaluation, communications, project management, media. Computer science and systems engineering is a continually changing environment (Nicholson & Ngai 1996).

While those in the computing field are knowledgeable about cutting edge technologies we must remember not to use new technology for the sake of it and the methodology used caters for evolving, or growing and must incorporate education and evaluation at each stage. Most would agree that developing multimedia projects needs a diverse range of expertise, including professionals in education and technology. When proposing to write a program, or whole subject, expertise needs to be sought from disciplines other than computing.

References

Allessi, S. M.and Trollip, S. R. (1991). *Computer-based Instruction*. 2nd ed. Prentice Hall.

Barker, P. and Yates, H. (1985). *Introducing Computer Assisted Learning*, Prentice Hall.

Beynon-Davies, P. (1993), Information System Development (Second ed) Macmillan Press, Ltd. London.

Bostock, S. (1996). *Courseware Engineering: An overview of the Courseware development process.*
[http://www.keele.ac.uk/depts/cs/Stephen_Bostock/docs/atceng.html]

Boyle, T. (1997). *Design for Multimedia Learning.* Prentice-Hall Europe. Hertfordshire

Bradler, J. (1999) Developing On-line Learning Materials for Higher Education: An Overview of Current Issues. *Educational Technology & Society* Vol 2(2)

Brown, J. (1997). HCI and Requirements Engineering- Exploring Human Computer Interaction and Software Engineering Methodologies for the Creation of Interactive Software. *Technical Report* CS-TR-97/1.

Carroll, J. and Moran, M. (1991). Introduction to this Special Issue on Design Rationale. *Human Computer Interaction* Vol 6, 197-200.

Caroll, J. (1995*). The Scenario based Design: Envisioning work and technology in system development (pp.1-18). New York: John Wiley and Sons, Inc.*

Carswell, L., Murphy, M. (1995). A Pragmatic Methodology for Educational Courseware Development, The 2nd All Ireland Conference in the Teaching of Computing, 5th-7th September 1994, Dublin City University, Dublin, Ireland [http://www.ulst.ac.uk/cticomp/papers/carswell.html]

Connell, J. and Shafer, L. (1989) Structured Rapid Prototyping: An evolutionary Approach to Software Development. Prentice-Hall, Inc. New Jersey.

Duchastel, P., (1993). Towards methodologies for building Knowledge-based Instructional Systems, Instructional Science, vol. 20, pp 49-58.

Eriksson, H. and Penker, M. (1998*) UML Toolkit: Unified Modeling Language* John Wiley and Sons, Canada.

Freeman, M and Capper, J. (1999). Exploiting the web for education: An anonymous asynchronous role simulation. *Australian Journal of Educational Technology*, Vol 15(1), p. 100. [http://cleo.murdoch.edu.au/ajet/ajet15/freeman.html]

Kennedy, D. (1998). Software Development Teams in Higher Education: An Educators View, Conference Proceedings ASCILITE '98 December 14 -16 Wollongong. (pp. 373)

Kensling, F. and Munk-Madsen, A. (1993). PD: A Structure in the Toolbox., Communications of ACM Vol 36(4),

Muller, M., Wildman, D. and White, E. (1993). *Taxonomy of PD Practices: A Brief Practioner's Guide*, Communications of ACM *Vol 36(4)*

Nicholson, A. and Ngai, J. (1996). Managing the Development and Production of Interactive Multimedia Courseware in Education. *Australian Journal of Educational Technology*, *Vol 12(1), 35-45.*

Pfleeger, L (1991) Software Engineering : The Production of Quality Software (2ed) *Macmillan Publishing Company, New York.*

Pressman, R. (1997) . Software Engineering: A practitioner's approach. (Fourth ed.) New York: McGraw-Hill Inc

Reeves, T. (1992). Effective Dimensions of Interactive Learning Systems. Finding the future: ITTE'92 Proceedings of the Information Technology for Training and Education Conference. Lucia, 1992 ,99-113.

Riley, F. (1995). Understanding IT: Developing Multimedia Courseware, ITTI, University of Hull.

Shah, J. and Bonner, D. (2000). *Findings of a Workshop by the National Science Foundation* [http://enws121.eas.asu.edu/events/NSF/report.html]

Sherman, L. W. (1995). A Postmodern, Constructivist and Cooperative Pedagogy for Teaching Educational Psychology, Assisted by Computer Mediated Communications. *In Proceedings of CSCL 95' Conference, Department of Educational Psychology* Miami University, Oxford, Ohio.

Sommerville, I. (1996). *Software Engineering.* Addison Wesley Essex – England.

Witt, C. L. and Wager, W. (1994). A Comparison of Instructional Systems Design and Electronic Performance Systems Design. *Educational Technology*, July-August, 20-24.