Transactions on Information and Communications Technologies vol 14, © 1997 WIT Press, www.witpress.com, ISSN 1743-3517

Courseware development and software engineering

J. Traxler
School of Computing and Information Technology
University of Wolverhampton, Wulfruna St, Wolverhampton
WV1 1SB, Great Britain
E-Mail : cm1990@wlv.ac.uk

Abstract

This paper examines aspects of the notion that courseware can be successfully engineered in the way that, we are told, software is engineered.
Over the past twenty years software development has used the paradigm of engineering and the analogy with traditional engineering to give it focus and direction. Recently the engineering paradigm has been grafted onto the development of courseware. This paper looks at the validity of this approach and asks at what level should the comparison be made between software engineering and courseware engineering.
In particular the paper considers the current use of some software engineering ideas to enhance the quality of courseware products and to manage and control their development.
These are the areas where the engineering analogy is less useful. In software engineering terms, courseware engineering seems to lack clarity or consistency in defining its customers, and in defining the purpose, scope and boundary of its products, leading to problems in subsequent testing and evaluation. In managing the developmental process, courseware engineering often resorts to informal incremental prototyping. In software engineering terms, this lacks the visibility that would facilitate many aspects of management.
The paper concludes by offering a courseware lifecycle development and deployment model that addresses these issues and maps well into current practice in modularised higher education and the needs of undergraduate computer science education.

1 Introduction

Over the last decade there has been a renaissance of interest in all forms of Computer-Assisted Learning as courseware has been developed and introduced into many academic disciplines in much of higher education. The development and deployment of this technology seems to have been largely unstructured, uncoordinated and possibly wasteful. The renaissance has probably been driven

Transactions on Information and Communications Technologies vol 14, © 1997 WIT Press, www.witpress.com, ISSN 1743-3517

partly by the enthusiasm of individual lecturers and partly by central funding initiatives, and made possible by a wealth of technical advances.

It is clear that the sustainable, systematic and efficient use of courseware in higher education, specifically in undergraduate computing courses in modular high-volume institutions where it is currently under-represented, will only come about when there is a fuller analysis and critique of :

- the tools, techniques and processes by which courseware is developed
- the relationships between courseware developmental and deployment costs; and the educational and institutional benefits, quality and risks.
- the educational and institutional context in which courseware is used
- the teaching and learning situations in which courseware is deployed

The first of these items is the focus of the current paper, specifically asking whether the development of courseware is facilitated or undermined by models of systems analysis and design and developmental lifecycles adopted from software engineering.

2 Developing Courseware

In the early days of software development much was made of the analogy between what came to be called "software engineering" and traditional "hard" engineering. The case was made that traditional engineering was successful and was characterised by, for example, the use of tools, notations, diagrams, mathematics, project management and quality assurance and that therefore the development of software would benefit from adopting similar practices.

Traditional engineering, if viewed in possibly a rather idealised sort of way, attempted to deal with problems or situations based around projects characterised by large size, complexity, performance and boundaries of their requirements; by customer dimensions characterised by uncertainty, dispute, change, goals, preferences, priorities and deadlines; by legacy; by costs, personnel and resources and the need for forecasting and controlling them; by communications with people such customers, contractors and colleagues and by the cost and imperfections of raw materials

The current state of courseware development methods resembles the early days of software engineering. The case for courseware engineering is made implicitly by the very use of the term "courseware engineering" but also has its own explicit advocates [1] who argue that previous generations of educational software in England were clearly the flawed products of a "cottage industry" using inadequate tools, such as BBC Basic or Microtext, each developed by a single analyst/programmer/domain expert, using no clearly articulated development process or evaluation procedure and with no sustainable financial relation between customers, developers and consumers. Putting courseware

Transactions on Information and Communications Technologies vol 14, © 1997 WIT Press, www.witpress.com, ISSN 1743-3517

development on the rational basis implied by the analogy with software engineering will, it is being argued, prevent a repetition of these problems.

There are, of course, some alternative positions being taken : one of these [2] is that the introduction of computers into educational systems is analogous to the introduction of computers into clerical information systems and thus that general systems thinking is appropriate, tempered by educational purposes in the place of commercial ones. This may be true but the logic of the analogy does not necessarily lead inevitably to the tools of classical systems analysis and design since these are now felt to be inadequate for many contemporary information systems [3]. In any case, there has always been some debate as to whether systems analysis and design are part of software engineering or *vice versa* so we lose nothing by choosing to work with the software engineering analogy rather than the information systems one.

In order to determine how and where software engineering can legitimately contribute to courseware development we need to identify what developmental problems courseware development is trying to tackle rather than just adopting software engineering tools and techniques without identifying the underlying nature of the problems and difficulties that gave rise to them.

These problems, common to both course development and software development, will be the problems of large, complex, poorly bounded or rapidly changing requirements, the problems of project management and quality assurance with an abstract medium, the problems of communicating with customers and teams of developers about intangible requirements and the problems of systems with substantial "soft" components.

Any developmental approach intended to address these kinds of problems must attempt to **enhance or assure the quality of the product** and attempt to **manage and control the process** of development.

In looking at the parallels between software engineering and courseware engineering, the areas of courseware engineering where there is least clarity, definition and certainty are :

- defining appropriate customers, then
- defining the scope of the system and hence also
- defining appropriate testing and evaluation practices and finally
- defining a repertoire of development lifecycle models.

3 Enhancing the Quality of the Product

Drawing on the analogy with software engineering, the courseware product's quality attributes must include ideas of correctness, robustness, completeness and efficiency as well as more technical attributes such as maintainability,

Transactions on Information and Communications Technologies vol 14, © 1997 WIT Press, www.witpress.com, ISSN 1743-3517

portability, interoperability and reliability. We should look at what some of these ideas mean in the context of courseware development.

In courseware development, defining correctness must mean that we seek to capture and represent requirements (including identifying educational objectives and purpose, hence an appropriate environment and systems boundary, defining users and their interaction and navigation through the system) and we must identify the source of these requirements, a putative customer.

Software engineering's techniques for this requirements capture are designed to deal with requirements that are large and complex and potentially incomplete or inconsistent.

The software engineering techniques attempt to deal with this size and complexity by partitioning requirements into those that are high priority, mandatory or urgent and those that are low priority and optional. These techniques also partition the requirements on other grounds, for example into functional requirements and non-functional requirements, in order defer some of the difficulties and prioritise dealing with some of the others. Structured methods assume functional requirements to be the highest priority. Many methods of capturing and representing requirements will attempt to manage their size and complexity by decomposing the (functional) requirements into different views or aspects and then creating a model or representation of each aspect (for example a data model and a process model) in order to reason about them. They involve the use of graphical or formal notations and sometimes the structured or formalised use of prototypes.

In free-standing, software-based courseware products, the complexity and size is located in the interface and the interactions rather than in the algorithms, data structures and the computations and so the representation of requirements must focus on these attributes. In a broader sense, courseware may be a system of interlocking, interdependent courseware subsystems such as audio tapes and conventional textbooks designed to deliver a course and the complexity becomes more difficult to represent, or indeed to validate, especially if some of these components are human.

Another difficulty is that courseware, in either of the senses above, should be robust as well as correct : we could interpret this robustness as meaning that it must designed to be tolerant of a wide variety of students with a wide variety of learning styles working in a wide range of sub-optimal ("noisy") environments. Similarly, maintainability must be measured in terms of the product's or course's ability to respond to such inputs as changed curricula, tutors or assessment regimes.

There are other attributes of courseware product quality that will also need re-interpreting but correctness, robustness and maintainability should suffice to illustrate that the idea the correspondence between courseware and software is non-trivial and dependent on a clearer definition and understanding of the courseware product.

Having considered some of the desirable attributes of courseware products, we can consider two of the lessons of software engineering, or rather systems analysis. The first is that we should not draw the systems boundary too narrowly and not implement a new technology as pieces of disconnected subsystems or products. In terms of courseware development, this means we should attempt to engineer the wider learning experience system (or perhaps the course or the module) rather than attempting to engineer the multimedia component or subsystem in isolation.

Nevertheless, in looking at the literature, it is difficult to see what courseware engineering is actually trying to design : is it a casual episode of non-deterministic browsing, a piece of so-called edutainment, a set of interfaces and interactions, the formal teaching and assessing of a topic, a multimedia/hypermedia software system or an entire module ? As well as uncertainty on this issue, the literature also suggests that there is a spectrum of tutor involvement and control : at one extreme, Open University course teams [4] slot human tutors into clearly defined roles alongside all the other multimedia components whilst at the other extreme, multimedia software can play a subservient and possibly more conventional role *vis-a-vis* teaching staff.

The second lesson of systems analysis, or rather actually soft systems methods, is the need for an adequate, accurate and appropriate identification of the customer and/or user. The confusion in the system boundary may be just a consequence of ambiguity in the definition of the customer or procurer that does not arise so obviously in software engineering (though there may be an analogy between the IT manager/data-entry clerk dichotomy and the lecturer/student one) : are the customers the students; the tutors, teachers, lecturers and demonstrators; the module leaders; the course leaders and subject leaders; the budget holders; the external funders or commercial publishers ? Of course much courseware has been developed by the people who deploy it and it never gets any wider exposure than its original environment, further confusing the supposedly contractual relationship between developer and customer implicit in "engineering" but this category may be the least relevant one to a courseware engineering perspective as opposed to a courseware craft or cottage industry perspective.

Having looked at courseware engineering's lack of a common understanding about its scope and customers, we have seen two areas where the correspondence between courseware engineering and software engineering is poorest.

4 Controlling the Process

Looking at the process's attributes, if there were an analogy with software engineering and its projects then courseware developers should be seeking to manage the size and complexity of the development process, ensure its visibility,

organise its social and personnel aspects, establish the means for communication and maximise cost-effectiveness and resource utilisation.

This implies the need for some project management framework within which courseware development can take place. In reality, the number of people and parties involved in development can vary from one [5] to many [6]. In these latter cases, individuals and institutions have worked together [7] in ways that was largely temporary and constrained by funding and so clearly depart from any ideals of team size or composition [8]. These factors may account for the dearth of reflection and generalisation on the issue of project management in courseware development. There are though indirect remarks about project management difficulties being not uncommon [9] and this is hardly surprising.

Courseware engineering unfortunately relies largely an informal version of incremental or exploratory prototyping and so there is a considerable mismatch between the metrics and the process.

An important attribute of the development process is an explicit process or lifecycle model. Unfortunately many accounts of courseware development suggest that the confusion about the system boundary, the purpose of the system and the identity of the customer make such a model difficult to articulate and may account for the predominance of prototyping as the preferred developmental model. The Open University is unusual in formally articulating how it develops its multimedia courses (or courseware depending on one's definition) in a Course Production Handbook. Many new universities, of course, do document courses extensively after they are developed but have no formal model for the process by which they are developed. The Open University, on the other hand, does and has always adopted an iterative prototyping model [10] for course development adapted to subject needs. This prototyping is punctuated by peer review and by milestones such as design review, software review, integration testing and functionality acceptance and is monitored by a project management team, Project Control, and by a Quality Assurance section which defines testing, standards, scheduling and inventory control. This model is adopted irrespective of the detailed balance of components in the course (or courseware). The Open University views V-models and waterfall-models as inappropriate for educational software and uses the iterative model to tie together the development work on all the disparate components of a course (or courseware).

The confusion about processes and customers seems to spill over into the area of testing and evaluation. These terms are used to describe activities which range from "scientific" evaluation of interfaces to educational evaluation of outcomes. Furthermore "formative" evaluation seems to be actually a form of evolutionary prototyping. There are relatively few general frameworks or methodologies for evaluation [11], [12], [13], [14], [15] and in practice, practitioners use something local and one-off [16], [17], [18], [19] that even if valid is non-transferable and difficult to generalise from.

To return to the lifecycle of courseware development, in some accounts the technical and software-centred aspects are relegated to a small part of a much bigger lifecycle if the focus is on educational design or are driven purely by the nature of the interface and interactions so there is little consensus on the scope of a lifecycle model.

For the software-based kernel of courseware, aside from the use of prototyping, either expressed explicitly or just assumed, there seem to be few other process models. Nevertheless it has been suggested [20] that prototyping is actually under-represented as a courseware development because courseware projects are relatively small and do not have a conspicuous maintenance requirement. One practitioner has provided a different argument for his specific project [21], namely that the house-style for the look-and-feel of the interface and the house-style template for progress and navigation through the courseware, together with the expertise of the programmers, meant that no process model was required beyond informal incremental prototyping. The process model was basically *experts-in-a-blackbox*. The house-style template itself was apparently derived from *a priori* educational principles and not validated within the scope of the project [22].

There are some alternatives to prototyping, usually following a structured methods approach [1], [24], [2], [26] and as we have said, these are sometimes embedded in a larger framework with a more pedagogical bias that downsplays the developmental of the actual multimedia software [27], [19] and thus attempt to embed a conventional software analysis and design phase into their lifecycle models as specific phases after educational requirements or needs analysis.

Unfortunately, none of these structured lifecycle models mentions the need for a feasibility stage prior to analysis and design to evaluate the likely economic or pedagogic benefits; nor are any methods based around soft-systems approaches identified. Nor are any of these methods specifically tailored to the needs of computer science and software engineering education.

5 Discussion and Conclusions

As a paradigm for courseware development, courseware engineering modeled on the analogy with either "traditional" engineering or software engineering is clearly defective. The analogy between software engineering and traditional engineering is itself probably suspect and possibly based on an optimistic and idealistic view of the success and methods of traditional engineering.

It could argued that any analogy with either "traditional" engineering or software engineering is flawed since both are intended (however unsuccessfully in reality) to deliver one-off or bespoke project outcomes within clearly defined constraints of time and cost to very clearly defined customers.

Courseware might be more like a (mass-produced) product intended for an open albeit niche market of unknown customers at whatever costs the market

will bear. Whilst therefore a better analogy for courseware development might be retail software product development, there does not seem to be a paradigm for this either (in spite of its undoubted commercial success) though self-evidently successful companies must have such their own paradigm [29], albeit tacit.

Clearly then courseware development is in a quandary because on the one hand, it does not have a sound and justifiable paradigm, legitimised by economic and institutional success, that delivers desirable and marketable artifacts but on the other hand, the lack of such economic and institutional success means that there is not a body of work with sufficient mass to underpin the development of a durable and successful paradigm for development and deployment.

This is an argument for a strategy that exploits current work and momentum in order to break this vicious circle. In order to build towards success such a strategy must play to the strengths and opportunities of courseware and this involves identifying, in this instance in the context of high-volume computer science higher education, what those strengths and opportunities are.

One might at a high level search for other existing paradigms of creation and production such as software product development (already mentioned) or artisan/apprentice-based craft production.

At a lower level, it should be possible for a developmental paradigm to develop out of a more successful and viable practice. A starting point for this would be to address the deficiencies identified above.

Firstly, in the context of institutional modular higher education, the customer is clearly the module leader, directly accountable for the quality and outcomes of the students' learning experience. Hence, the broadest and most appropriate definition of the courseware product is the module and all the technologies within it that deliver learning, rather any free-standing constituent courseware product. The system boundary is coterminous with the module and its purpose is defined by module's aims and objectives and implicitly by prerequisites and post-requisites [30].

The arguments for an institution-wide look-and-feel and consistent navigational metaphors are self-evident and are attractive to developers, students and to institutional quality auditors and public relations personnel. Many projects [31] advocate standardising on a small number of authoring tools and platforms and again the virtues are apparent, in, for example, rationalising staff training and central technical support.

Two major deficiencies identified above in current courseware development are the interlocking issues of process model and evaluation.

The point of departure for the process model must be incremental prototyping. This can now be constrained by the module delivery, review and revalidation cycle stretched over say five years. The annual or bi-annual deliveries of the module become the prototyping increments. Compared to the norms for prototyping, the proposed process model must devote a greater proportion of resources to the delivery/maintenance phase and less to the initial

increment and must have a specifically educational but systemic evaluation in each delivery.

The cycle would start by taking an existing module as the first iteration of the prototype, embodying an inchoate version of the courseware requirements. The components, including any human components and implicit components such as the Web and the library, could then be identified along with ideas about their (actual) purposes, their functions and the relationships between them all. A courseware-style evaluation and institutional statistics would illuminate the module's current short-comings.

An incremental prototyping approach based around the annual module delivery cycle would allow for periodic maintenance and content revision, very necessary given the pace of change in the computing curriculum, as well as acting as a focus for staff development and the transformation of academics into "reflective practitioners" [32], "owning" what they use.

The proposed process model should obviously exploit the technical expertise of computer science lecturers in their ability to use new and more powerful tools and their ability to work confidently with various developmental process models. In these senses, computer science lecturers are not naive courseware developers. Conversely the proposed process model must recognise that computer science lecturers are pedagogically naive and find ways to transform their understanding of what they know into forms of expression and organisation accessible to all their learners.

What is known about courseware developmental cost factors, and what we know about the institutional metrics of module performance and quality, should encourage successive iterations of the module where at every opportunity the most expensive medium and method is progressively replaced by the one that is more attractive in terms of either costs, benefits or quality. The approach outlined would be responsive, create a sense of ownership amongst the users and create a growing resource grounded in practice and experience.

References

[6] Arnold, S et al (ed Donnelly, P J), Constructing and Implementing Multimedia Teaching Packages, University of Glasgow TILT, 1994

[8] Avner, R A, Is there any ideal size for courseware development teams?, Proc 30th Association for the Development of Computer Based Instruction Systems, Philadelphia, 1988

[14] Barker, P and King, T , Evaluating interactive multimedia courseware - a methodology, Computers Educ, 21, 4, 1993

[3] Bates, P E & Stephens, M A, Rapid Application Development - Concept, Methodology or What?, Proc NDISD, University of Wolverhampton, April, 1995

[13] Bates, T, Towards a Better Research Framework for Evaluating the Effectiveness of Educational Media, British Jnl of Educational Technology, 3, 12, pp215 - 233, 1981
[20] Black, T R, & Gray, D E, Prototyping of Computer-Based Training Material, Computers Educ, 22, 3, pp251 - 256, 1994
[1] Black, T R, & Hinton, T, Courseware design methodology : the message from software engineering, Aspects of Educational and Training Technology, Chap 4, pp25 - 32, 1988
[19] Boyle, T, Gray, J, Wendl, B, & Davies, M, Taking the Plunge with Clem : the Design and Evaluation of a Large Scale CAL System, Computers Educ, 22, 1/2, pp19 - 26, 1994
[29] Cusamano, M. A. & Selby, R. W.,Microsoft Secrets, 1996
[9] Gardner, N, & Darby, J, Using Computers in University Teaching : a Perspective on Key Issues, Computers Educ, 15, 1/3, pp27 - 32, 1990
[16] Garrud, P. et al, Non-verbal Communication : Evaluation of a Computer-Assisted Learning Package , Medical Education, 27, 6, pp474 - 8, 1993
[24] Howe, R F, & Williams, D O, A Methodology for Developing Computer-Based Teaching Programs, Computers Educ, 10, 3, pp347 - 352, 1986
[27] Keane, M M and Monaghan, P F, A Requirements Analysis/Educational Design for Courseware Relating to Heat Loss Through Building Walls, Microcomputers in Civil Engineering, 6, 2, pp311 - 28, 1991
[32] Laurillard, D.,The role of formative evaluation in the progress of multimedia, IFIP Trans A, A - 59, pp287 - 293, 1994
[17] Leathard, H L, & Dewhurst, D G, Comparison of the Cost-Effectiveness of a Computer-Assisted Learning Program with a Tutored Demonstration, Association for Learning Technology Journal, 3, 1, pp118 - 125, 1995
[11] Lefrere, P,Evaluation Methods and Procedures for Studying Learners' Use of Media, OU web-site (TELL/PLUM),iet/iet/PLUM
[31] McAteer, E & Shaw, R, Courseware in Higher Education - Evaluation 1 : Planning, Developing and Testing,The EMASHE Group, University of Glasgow,1994
[10] Moore, private communication, 1996
[21] Norcliffe, A., The TLTP W.I.S.D.E.N Project - a Progress Report,Proc. of 4th Annual Conference on the Teaching of Computing, Dublin, pp104 - 107, 1996
[5] Phillips, W A, Individual Author Prototyping : Desktop Development of Courseware, Computers Educ, 14, 1, pp9 - 15, 1990
[4] Sharp, H. et al, Integrating 'real' software project experiences into a postgraduate course using computer-based multimedia, Proc Inaugural Meeting of SIG in the Teaching of Software Engineering, 1996
[2] Stoner, G., A Systems Analysis and Design Approach to the Integration of Learning Technology, Proc of ALT-C 96, 1996
[22] Stubbs, G, Watkins, M, & Davies, R A, UDRIP, a model for CAL Specification and Design (unpublished), Proceedings of 2nd All-Ireland Conference on the Teaching of Computing, Dublin, 1995

Transactions on Information and Communications Technologies vol 14, © 1997 WIT Press, www.witpress.com, ISSN 1743-3517

[26] Thomas, R,Durable, Low-cost Educational Software, Computers Educ, 22, 1/2, pp65 - 72, 1994

[7] Trainor, R H, Implementing Computer-Based Teaching and Research : the Need for a Collaborative Approach , Computers Educ, 12, 1, pp37 - 41, 1988

[30] Traxler, J, The Nature and Impact of Current Trends in Higher Education on the Teaching of Computer Science and Software Engineering, Proceedings of First All-Ireland Conference on The Teaching of Computing, Dublin, pp140 - 46, 1993