Int. J. Engng Ed. Vol. 14, No. 6, pp. 407–418, 1998
Printed in Great Britain.

0949-149X/91 $3.00+0.00

Managing Large-scale Multimedia Development Projects*

SIMON PRICE
Institute for Learning and Research Technology (ILRT), University of Bristol, 8 Woodland Road, Bristol BS8 1TN, UK. E-mail: simon.price@bristol.ac.uk

This paper presents generally applicable techniques drawn from the experience of managing the UK's Teaching and Learning Technology Programme (TLTP) Economics Consortium project to develop WinEcon—a computer-based package covering an entire first-year introductory economics degree course. The WinEcon project has been a highly successful, large-scale multimedia project. It has received multiple international awards, is site licensed by over 80% of UK universities and over 200 organisations world-wide. However, what really happens when you set out to develop the world's largest computer-based training package for economics with a team of 35 content experts and 17 programmers distributed across eight geographically separate sites is a far cry from the typical case study found in a software project management textbook. There are inherent characteristics of multimedia software which make its development difficult. Consequently any multimedia project carries a high risk of failing to deliver on time, quality or budget and the nature of large-scale development projects only serves to amplify the risk to such a degree that many such projects fail to deliver satisfactorily in any of these three areas. These management challenges encountered by the WinEcon project are independent of subject matter and must be addressed when managing any large-scale multimedia development.

INTRODUCTION

SINCE 1992 the UK Higher Education funding bodies has awarded £35 m to 76 discipline-based consortia under phases 1 and 2 of their Teaching and Learning Technology Programme (TLTP). Under this initiative, the Economics Consortium, a partnership of eight UK university economics departments, were awarded £640,000 to develop computer-based learning materials covering the whole first year economics degree course [1]. A comprehensive presentation of an electronic engineering design project within TLTP was given by Hicks *et al.* [2].

Five years on, after a total investment of over £1,000,000 and deals with six publishers, the resultant package, know as WinEcon, remains the world's largest and most widely used courseware package for teaching introductory economics. It has received multiple international awards, is site licensed by over 80% of UK universities and over 200 organisations world-wide.

However, in common with the experiences of other successful TLTP projects in disciplines apart from economics, the experience of the Economics Consortium has been an education in itself. This paper attempts to pass on the hard learnt lessons of this education, drawing on the experiences of the author as a member of the WinEcon management team and as a technical consultant to other TLTP projects.

* Accepted 10 October 1998.

PROJECT AIMS AND OBJECTIVES

The WinEcon project had two overarching aims:

1. to reduce the cost of delivering introductory economics courses, particularly for non-specialist students;
2. to maintain the quality of these courses in a climate of decreasing funding and increasing student numbers.

To achieve these aims the project targeted the traditional tutorial, where the staff to student ratio is relatively low and, consequently, where the cost of delivery is relatively high. This is in contrast to lectures where the ratio is high and cost relatively low. The consortium's key objective was to produce a software package capable of being used as at least a partial replacement for tutorials. Ultimately, it was hoped that the package would be capable of replacing up to 50% of tutorial time in any UK university which adopted it.

WinEcon, the resultant software package, was designed to cover the entire first-year economics degree course. It consists of over 1000 tutorial screens, a glossary, references to leading texts, economic databases, quizzes, exams, a course management system and a customisation interface.

PROJECT DEVELOPMENT MODEL

Although less than optimal in terms of technical effort, the Consortium opted to distribute the development work across the eight member universities with central co-ordination from a

small management team at Bristol. This model was deliberately chosen after a lengthy debate over the relative merits of centralised versus decentralised development. In all the arguments relating to effort and technical efficiency the centralised model was clearly superior. However, a decentralised model was chosen with quite a sizeable majority. The logic behind this seemingly bizarre decision being that there was serious concern that the project should build the right application, even if inefficiently, rather than the wrong one efficiently. In common with most multimedia projects, content rather than coding dominates development effort [3] and so it was deemed crucial that the content experts (i.e. the economists) be deeply involved in the development on a day-to-day basis.

Linked to this decision was the adoption of an iterative, evolutionary development model to facilitate a high degree of user (i.e. the economists) involvement at every stage in the development process. This, essentially, Rapid Application Development (RAD) model contrasts sharply with the classic waterfall model with its discrete stages and user input largely confined to the pre-development stages of the life cycle [4].

Organisational structure

Interestingly, the Economics Consortium, as is frequently the case with consortia, had a circular management structure. On day one the consortium consisted of eight Member Universities, each with equal representation on the Executive Committee. The Executive Committee appointed a project director and senior programmer as the management team to be based at the Centre for Computing in Economics in Bristol. The management team were charged with delivering WinEcon. So far this is analogous to a project control board and project management team in a hierarchical organisation. However, the developers undertaking the work were the Consortium Members themselves. Consequently there is a circle of 'advice' as illustrated in Fig. 1. Lines on the diagram indicate the flow of advice: heavy lines represent formal advice; light lines represent informal advice.

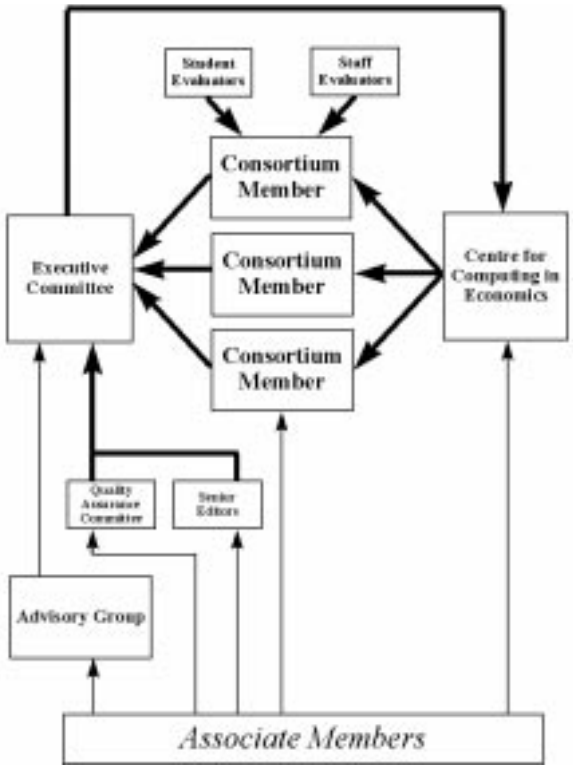

Fig. 1. Economics consortium structure.

The positive effect of this structure is that a product must be produced by consensus if it is to be produced at all. Therefore, such a structure is far more likely to result in a generally acceptable product, overcoming the oft cited 'not invented here' justification for rejecting off-the-shelf courseware. The incorporation of advice from around 100 associate member universities and their elected advisory group also safeguarded against the creation of an esoteric and unwanted product.

In arriving at this organisational structure the consortium made a judgement that the risk of failure resulting from unclear lines of authority and responsibility was outweighed by mitigating the risk of developing an unwanted product. Even so, a circular line of responsibility does carry a high risk of being unmanageable and one is unlikely to see it recommended in a software project management text.

INHERENT RISKS OF MULTIMEDIA PROJECTS

Non-multimedia, software development is typically undertaken by specialist programmers within specialist organisations using specialist tools and methods. By contrast, multimedia software development (especially in the educational sector) typically takes place under dramatically different circumstances. Multimedia, by its very nature, requires multidisciplinary skills for its development. It is characterised by the majority of the development effort being associated with content and user interface as opposed to code. In the WinEcon project a ratio of 2:2:1 was measured in review times which suggests that code accounted for only one fifth of the development effort. This fundamental shift in emphasis, coupled with novelty, tends to result in the following three characteristics which present inherent risks to any multimedia project:

1. *End-user programmers.* Content experts are often involved in development in a hands-on fashion and yet have no formal training in software engineering.
2. *Relatively inexperienced organisations.* Development teams are often drawn together from diverse organisations, cultures and

backgrounds to work together solely for the life of the project.
3. *Relatively new tools and methods.* Code-based development tools have been evolving for decades and are supported by analysis and test tools, methods and standards which are not yet available for multimedia [5].

Developing any type of software can be a risky business even for specialists. Due to the commercial and political sensitivity of data on project failure it is difficult to obtain an accurate estimate of project failure rates. However, some estimates have been made and about RAD projects in particular. Multimedia projects are usually RAD projects and according to Cambridge Market Intelligence, in 1995 over 70% of RAD/client server projects failed [6]. If we assume that multimedia projects are indeed unlikely to be more successful than conventional software projects then, all multimedia projects presently carry an inherently high risk of failing to deliver on quality, cost and time.

Additional risks in large-scale projects

In a small project with a handful of developers based at a single location, communication between the developers is relatively straightforward. In such circumstances, maintaining good team dynamics and a shared vision of the product is achievable and inexpensive. However, as the number of developers increases there is a nonlinear increase in the number of lines of communication and before long intraproject communication becomes difficult and expensive. The problem is exasperated when the developers are spread across multiple sites and organisations.

In the case of large-scale multimedia development projects there is also an additional effect. The inherent risks related to end-user programmers, relatively inexperienced organisations and new tools and methods are amplified so that previously benign issues become serious threats. For example, end-user programmers tend to know little about variable scope (e.g. local vs. global variables) which can conceal an integration time bomb when their work must eventually run alongside that of the other end-users. Another example, which the author has encountered in numerous multidisciplinary large-scale projects, is that there is no clear definition of when and how a job of work is deemed to be finished. A final example of how project size amplifies multimedia risks, in this case the risk of losing or overwriting each other's work, comes from the fact that today's software version control systems are poorly equipped to handle the data formats and volumes required for multimedia development.

Stacking the odds in favour of success

By identifying the inherent risks of large-scale multimedia projects early on in the life of the WinEcon project the Economics Consortium was able to take steps to avoid downstream problems. The following is a list of what turned out to be the five most efficient and effective of these measures, compiled with the benefit of hindsight:

- guidelines
- templates and model examples
- peopleware
- inspections and reviews
- customisation.

These are dealt with individually in the following sections.

GUIDELINES

Written guidelines setting out project procedures, conventions and standards are a basic requirement of any large project. Like the software, it is hard to produce them all at once and an evolutionary approach will be required. At some point before the main body of work starts these guidelines must be frozen.

Development work on a project should not start until the first iteration of the guidelines has been agreed. It has long been known that errors early on in the development process can ultimately be orders of magnitude more expensive to correct if carried through to later stages of development. Guidelines are a way of avoiding potentially serious early errors across a large project.

The WinEcon project's main guidelines started off as a supplementary manual for the ToolBook development environment but rapidly changed direction and title to become, *Authoring Guide—Standards for Courseware Authoring*. Over the first two years of the project it grew from an initial five pages to a 78-page document, at which point it was frozen. The final table of contents is listed below and shows the roughly sequential evolution of the document.

1. Development environment
2. Installation
3. Extensions to ToolBook
4. Custom authoring tools
5. Module structure
6. Programming style
7. Authoring style
8. Documentation
9. ToolBook hints and tips
10. WinEcon template error messages
11. Appendix I—E-mail Feedback Form

Although the authoring guide was the most substantial set of guidelines produced by the project, the following, smaller guidelines were also produced early in the project life cycle:

- *Data Security Plan—backup and anti-virus procedures*
- *Programmer's Logbook—an example*

Both of the above documents were concerned with risk reduction and, as events transpired, proved to be prudent measures. The data security plan facilitated a successful disaster recovery after thieves stole the NeXT FTP server which held master copies of two years of development work. Similarly, the logbook reduced the difficulty of continuing the work of a key developer who left before completing a software module on the critical path.

TEMPLATES AND MODEL EXAMPLES

Reusable templates (also described as patterns in some software engineering literature) can pin down most aspects of a screen or an interaction, leaving the developer to specify only what is different in each instance. Templates constrain creativity in many ways but in so doing can act as a rigorous method for enforcing project standards. At the same time, a template can provide a cost-effective way of making global changes to all instances of the template. Without templates, far-reaching late changes in response to user trials can be prohibitively expensive.

While written guidelines can be effective, some aspects of multimedia development cannot easily be captured by the written word and are more easily communicated through model examples or reusable components. It is useful to note that, when producing computer-assisted learning software, these examples may well be pedagogic exemplar rather than just technical models.

The WinEcon project developed a custom template layer on top of the Asymetrix ToolBook authoring tool. In its first version the template imposed a rigid screen format with predefined areas reserved for specific interactions. However, while highly efficient in technical terms, this proved too constraining in pedagogic terms and was soon replaced by a more flexible version. The cost of giving authors this extra flexibility was a template which was both more difficult to learn and more difficult to use. The benefit was a more educationally appropriate product.

The template went through nine major releases, although some of these releases actually removed features rather than added them. A notable example of feature removal was the reduction in the number of available font sizes and styles. This was to enforce a hitherto written guideline on the use of fonts that was being ignored by some authors.

Figure 2 shows a typical screen authored in the template. It was built around a number of standardised typographical elements and pre-programmed interactions. On top of these are a

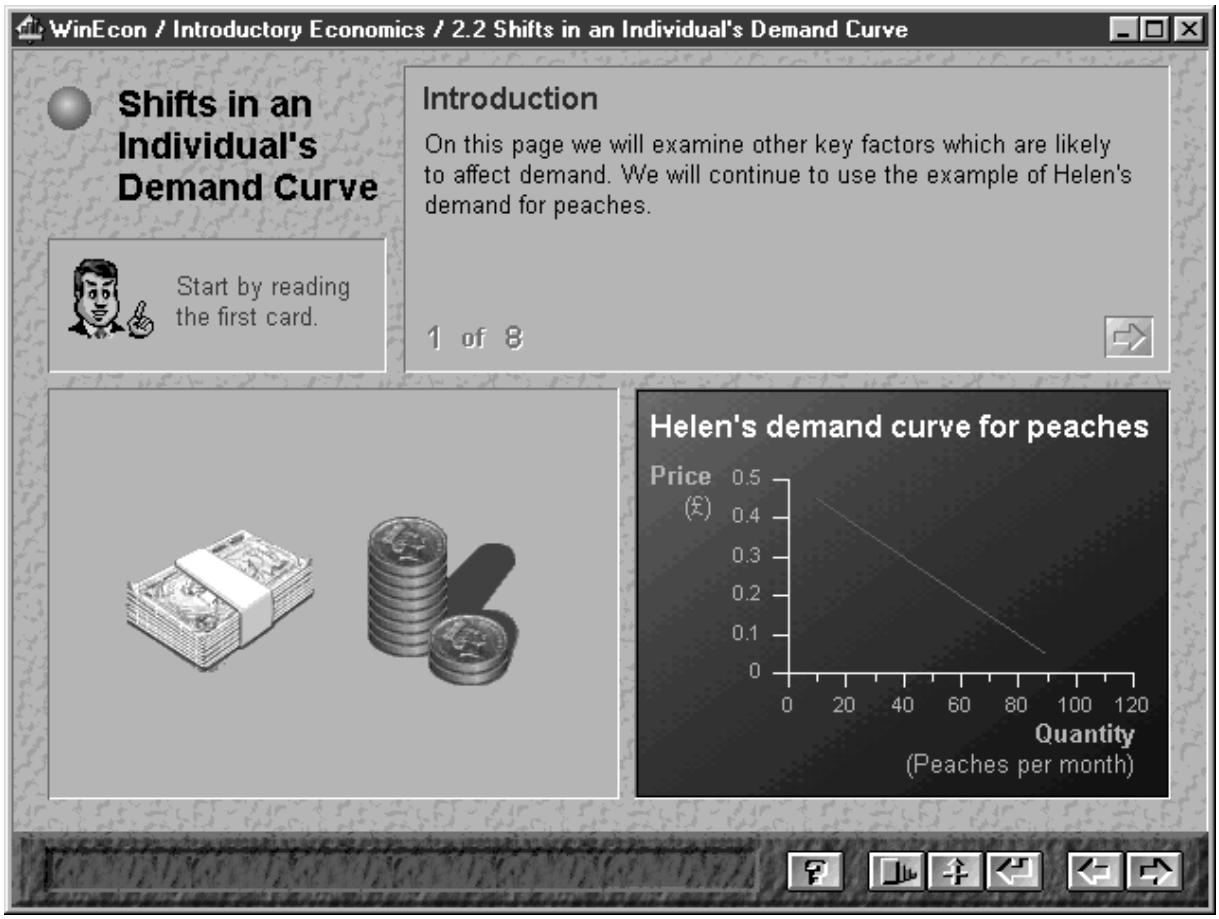

Fig. 2. Typical screen authored in the template.

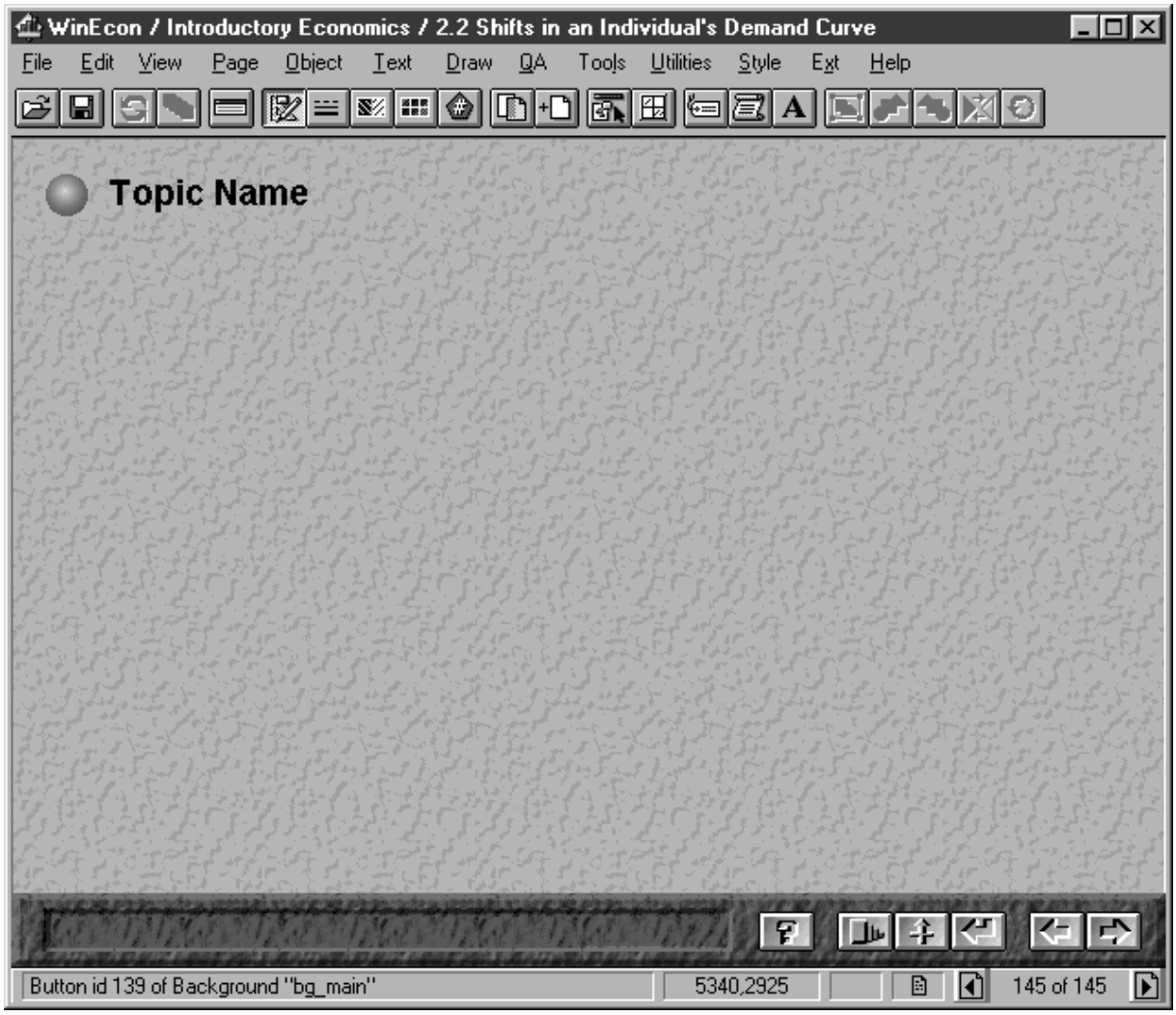

Fig. 3. A blank template screen.

number of layout properties and custom behaviours specific to this screen.

In order to create the screen shown in Fig. 2, the author would first create a new, blank template screen (Fig. 3). In keeping with most template-based systems, this blank screen encapsulates all the default control buttons in addition to a standard graphic design. Having created a blank canvas the author then needs to add each of the necessary user interface elements such as the graph, the 'step-by-step field', the 'professor field', the clipart and 3D objects.

The earliest versions of the template supported relatively few of these user interface elements and authors had to create them from native ToolBook objects and their own scripts. As it became clear what the most commonly needed elements were, standards for them were agreed and reusable components were created and added to a catalogue of clip objects (Fig. 4).

As the catalogue grew in size and the clip objects grew in complexity, alternative and more efficient methods of achieving standardisation were added to the template. Eventually, the role of the catalogue was refined to be solely a source of clip art and model examples; most interactive objects were transferred to the core template, accessible through pull-down menus added to ToolBook. The most frequently used pull-down menu was the Style menu which allowed authors to apply project standard styles to regular ToolBook objects.

Styles ranged from the purely typographical through to functional behaviours for objects. An example of the former was the 'professor field' which was used to convey advice to the student on how to use the current screen. By drawing a normal ToolBook text field (e.g. Fig. 5) and applying the professor field style to it, a graphic was attached to the field and font styles and borders were automatically set (Fig. 6).

When used for typographical standards, WinEcon template styles performed exactly the same role as styles in a word processor and had all the same benefits of consistency and ease of global update. However, by adding functionality to the style, as well as typography, far more powerful objects could be created. For instance, by drawing a normal ToolBook text field with text paginated by '@' characters (e.g. Fig. 7) and then applying the 'step-by-step field' style, a multiple 'card' text

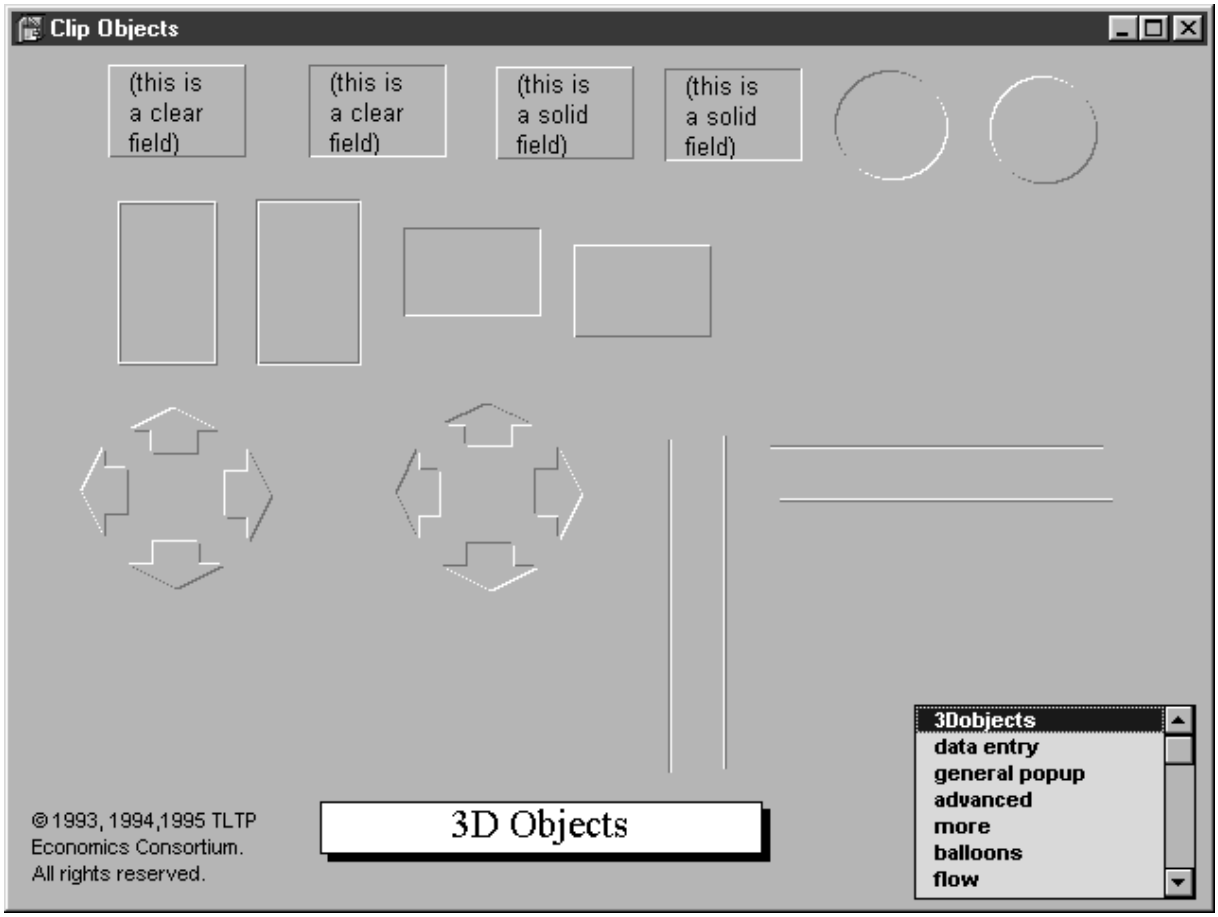

Fig. 4. Clip objects catalogue.

object could be created (Fig. 8). The template automatically split the text up into a number of cards and inserted left and right buttons to step backwards and forwards through the discrete blocks of text. These objects were used to drive the step-by-step build-up of explanations, graphs, tables and so on. Prior to the introduction of this object style, each author had invented their own convention for navigating backwards and forwards through a screen. Each author had come up with a slightly different look and feel and, in many cases, significantly different behaviours. The step-by-step style object provided a project standard and the effort involved

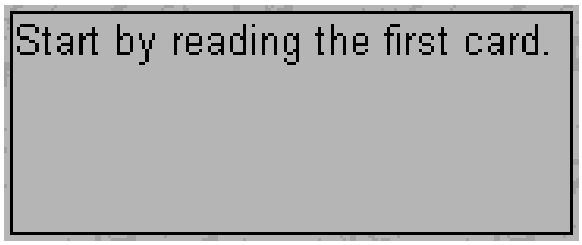

Fig. 5. Text field prior to applying 'professor field' style.

Fig. 6. Text field after applying 'professor field' style.

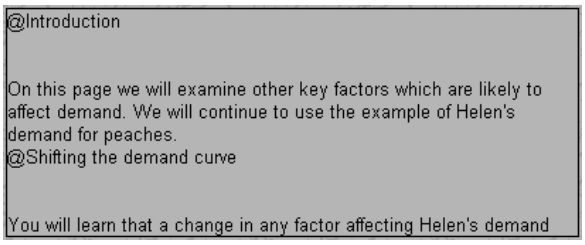

Fig. 7. Text field prior to applying 'step-by-step field' style.

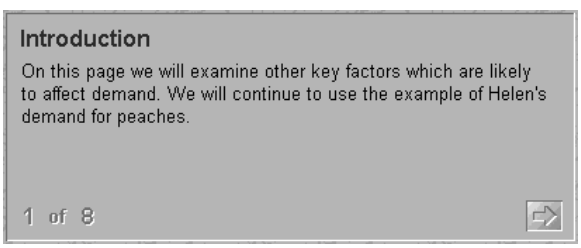

Fig. 8. Text field after applying 'step-by-step field' style.

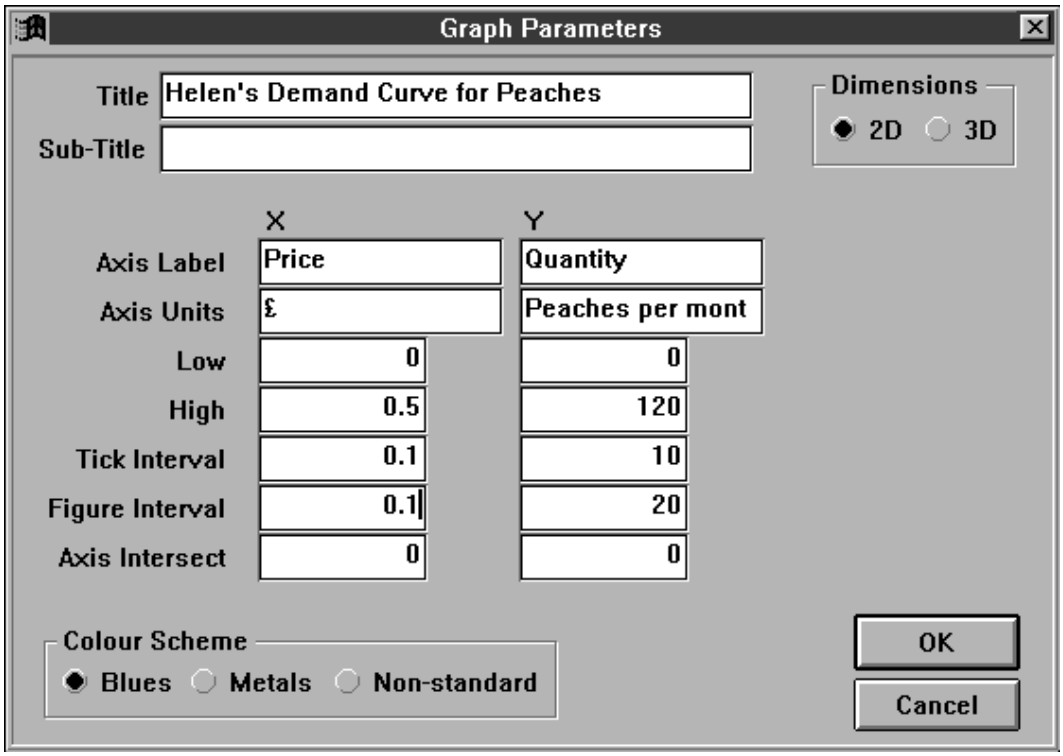

Fig. 9. 'Graph field' style's dialog box.

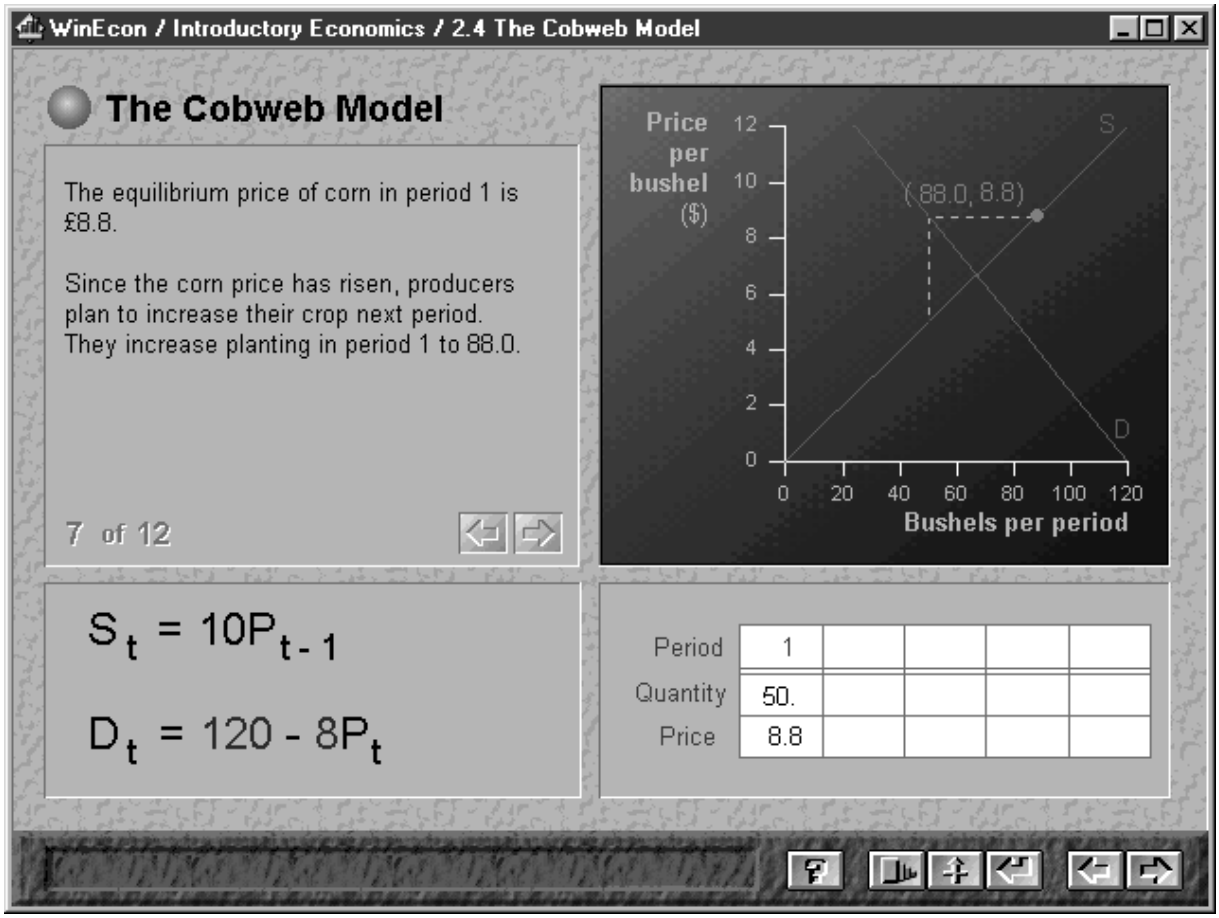

Fig. 10. A screen about to cause information overload.

in retrospectively replacing the diverse, home-grown mechanisms was worthwhile given the improved consistency across the courseware.

The graph which appears in Fig. 2 is a more extreme version of a style objects with built-in functionality. It was created by drawing a normal ToolBook text field and then applying the 'graph field' style. Doing so presented the author with a dialog box (Fig. 9) into which they would enter the details of the graph. Complimentary styles existed for lines, points and rectangles, each with their own dialog boxes. The need for a graph object had been anticipated from the outset. However, it had not been appreciated how difficult the non-programmer authors would find the programming associated with adding interactivity to graph objects. Fortunately the style system made it possible to retrospectively attach appropriate default functionality to graph objects to significantly reduce the burden on the non-programmer authors—allowing them to concentrate on the economic content rather than learning to program.

One example of where the template approach to developing a large-scale multimedia program had a clear pay back was when the project reached the alpha release stage. A serious problem arose which had not shown up in early pilots. Feedback from users of the alpha version was generally positive in all but one respect: the step-by-step mechanism for building up screens a piece at a time was very unpopular. The reason cited was that when one clicked on the next step button everything changed on screen at once, causing information overload and uncertainty about where to look first. The problem is illustrated in the transition from Fig. 10 to Fig. 11.

Clicking the next step button (i.e. the right arrow in the step-by-step field) simultaneously changed the text, the graph and the table. Users frequently thought that they had missed something and kept flicking back to the previous step.

Fortunately, by making a minor change to the template it was possible to globally fix the problem without having to undertake a major rework of the whole project. An extra button, the SHOW button was added to the step-by-step field style so that when a user clicked the next step arrow, only the text would change and a SHOW button would appear. Having read the text, the user clicked the SHOW button to reveal the changes elsewhere on the screen. Figure 12 shows this intermediate step between the screen states shown in Figs 10 and 11; only the text differs from Fig. 10 and clicking SHOW would then reveal the changes to the graph and table to arrive at the state shown in Fig. 11.

One final advantage of the template approach came when a new version of the ToolBook authoring

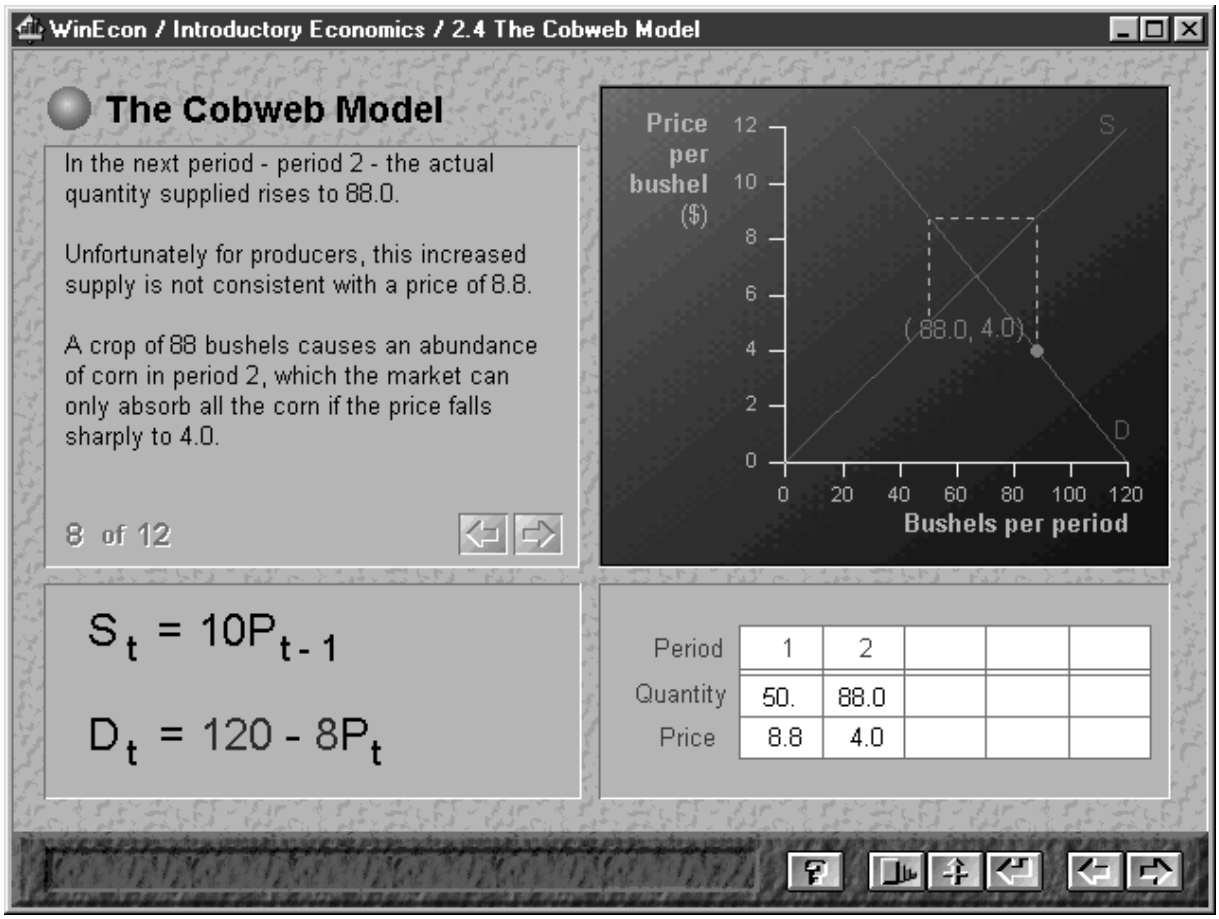

Fig. 11. The same screen after information overload.

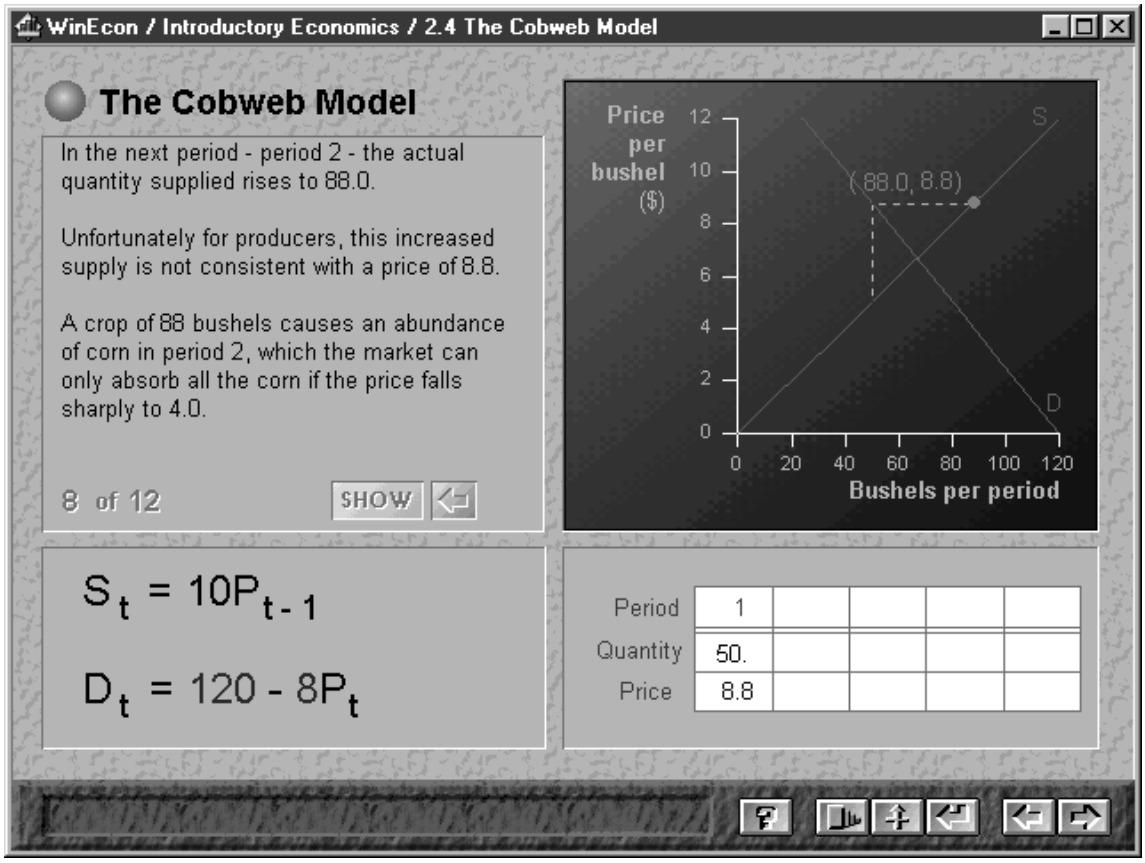

Fig. 12. Intermediate step using SHOW button.

tool was released mid-way through the project. By applying global changes to the template the project was able to upgrade to take advantage of the much improved authoring tool. The upgrade would have been prohibitively expensive otherwise.

PEOPLEWARE

'Peopleware' is a term borrowed from DeMarco and Lister [7] and refers to the role of people in software development teams.

However, before delving into the realms of team dynamics and psychology, it is important to note that one cannot understate the importance of good old-fashioned contractual obligation with financial penalties for failing to deliver. For all the peopleware issues discussed below, a clear and legally binding contract between partners is essential in any development partnership. In particular, it safeguards the time contribution of content experts for whom multimedia authoring is often only a minor part of their overall job function.

On the other hand, contractual obligation does not have much to contribute in terms of creating team spirit, rivalry and the shared vision which is so hard to achieve with large teams. In WinEcon these were consciously built up through regular newsletters, discussion lists and meetings. Other communication methods may be more appropriate in different projects but these worked well for WinEcon.

Best practice should be identified, shared and promoted. Those who contribute to the exchange should be rewarded financially or through some other incentive scheme. Peer respect and rivalry are also powerful motivators if individual or team successes are communicated across the project.

One effect it would be wise not to overlook is the pride developers can derive from external recognition of their work. On a small project this is immediately obvious whenever demonstrations are given to visitors or trials conducted. On larger projects, particularly multiple site projects, it is important to communicate this positive feedback to those members of the team who would otherwise not receive it.

INSPECTIONS AND REVIEWS

The mixed backgrounds of multimedia developers makes the application of the software quality management techniques particularly difficult. Even so, the WinEcon project achieved considerable success with a weak form of Fagan Inspections [8, 9] tailored to cover content and user interface as well as, more conventionally, code.

Full Fagan Inspections, while effective, can be prohibitively expensive to conduct over a large

volume of multimedia material. Consequently, the Economics Consortium adopted a simpler programme of structured reviews. Before the first round of reviews these were seen as an imposed bureaucracy and were strongly argued against by some of the team. By the third round, they were a popular event and the reviewers became a much sought after resource.

On the motivational level, simply knowing that poor quality would be detected put an end to a number of bad practices. Also, considerable personal satisfaction could be derived from passing a review.

The reviews produced perceptible improvements in the '-ilities' (e.g. reliability, maintainability, usability) while simultaneously having a positive educational side effect. The fine details identified in the ever evolving review checklist built up a common view of the product at a level which was not achievable through written guidelines or even through templates.

On the WinEcon project the checklist grew from a two to a nine page document over the first two years of the three year project. The checklist was divided into three sections covering each of the three main areas of multimedia courseware development: user interface, content and code. This is can be seen in the table of contents from the checklist document listed below.

1. Look and Feel
   1.1. Sections
   1.2. Pages
   1.3. Objects
2. Educational Style
   2.1. Interaction
   2.2. Learner-control
   2.3. Individualisation
   2.4. Educational Objectives and Prerequisites
   2.5. Assessment and Feedback
   2.6. Readability
   2.7. Cross-referencing
3. Technical Style
   3.1. Authoring Guide
   3.2. OpenScript Code

Each section in the checklist consisted of a series of questions under relevant sub-headings. The questions for the subsection on graph objects are listed below as an example of typical checklist items:

- Do graphs conform to the standard style?
- Do they have the 3D frame?
- Do they have the source of the data?
- Are they labelled and have keys?
- Are they sufficiently legible with consistent colours?
- Are they animated?
- Are blue and white grid lines used?
- Do these reveal and do so in the correct direction (^ <)
- Are arrows used to show the direction of animation?
- Have solid shading blocks been used?

Like the core template, the checklist reached a point where it was felt necessary to freeze it in order to provide a fixed frame of reference and target for developers. This occurred approximately two years into the three-year project by which point the checklist had largely stabilised anyway.

Inspections and reviews also offer a relatively objective measure of when a job of work is complete. Given that the checklist evolved with contributions from across the project team it then embodies the most detailed available specification of a 'complete' module. Were there no other benefits, this alone might be sufficient justification for the adoption of reviews or inspections in a large-scale project.

CUSTOMISATION

No matter how well managed a large-scale educational multimedia development project is, it may still be doomed to failure unless the end result is customisable. It simply is not possible to please all the people all the time and this is never more true than when developing software as a consortium of diverse organisations. During the early requirements analysis for WinEcon it became apparent that no single solution would satisfy all the Economics Consortium partners' expectations. Chiefly, there were differing and mutually exclusive opinions as to the order in which various economic topics should be taught. A technical solution was proposed by the management team to allow a compromise to be reached. It was decided that the courseware should be customisable so that the structure of courses could be redefined by lecturers to suit their own requirements.

Thus, the WinEcon Lecturer customisation and course management system was born out of a dispute over the structure of the ideal course structure. WinEcon Lecturer allows instructors to aggregate subsets of topics selected from the full range of around 1000 screens into sections, chapters and courses of their own choice. Similar customisation features in WinEcon Lecturer (Fig. 13) solved disputes over the correct definition of glossary terms, the right Fig. textbooks to refer to, the best wording of questions and a host of other subjective choices.

Although strictly speaking a software requirement, end-user customisation of software, as used in WinEcon, is a powerful managerial tool that facilitates compromise where it would otherwise be impossible. It was not part of the initial project specification submitted to the funders but was introduced to meet a managerial need for compromise.

CONCLUSION

There are strong arguments to support the view that multimedia projects carry a number of

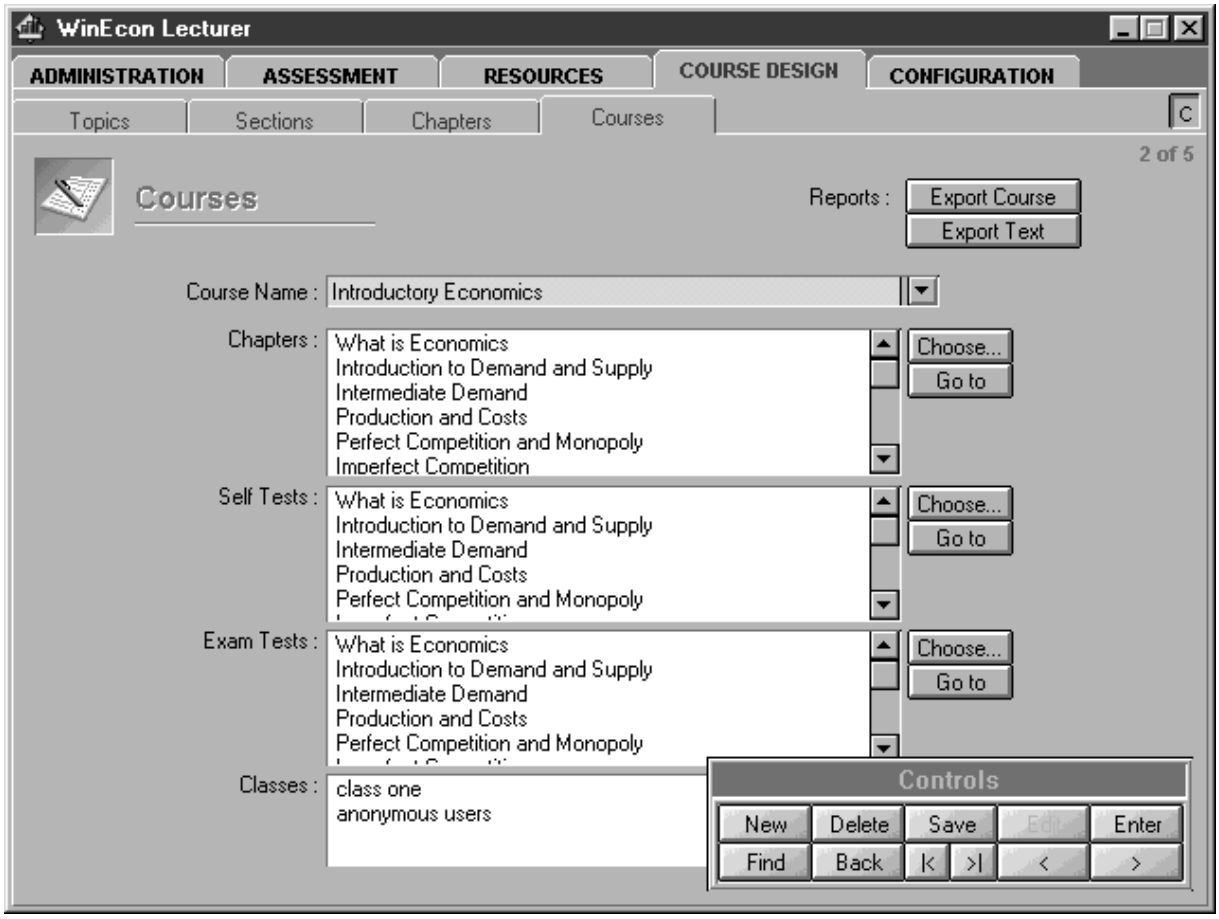

Fig. 13. WinEcon Lecturer customisation interface.

inherent risks which are amplified in large-scale developments. The greatest risk of all is that the project will be unable to specify sufficient detail at the outset and that the wrong product will be built. This risk can be offset by adopting an evolutionary development model with a high level of end-user involvement throughout all stages of the development process. While doing this results in a number of secondary risks, these can be addressed if the project takes the time to monitor and manage them. Guidelines, templates, peopleware, reviews and customisation are profitable areas to address in meeting the challenges these risks pose to large-scale multimedia development projects.

In 1998 the Economics Consortium was awarded a further £299,000 to develop WinEcon II or 'web.econ' as it will be known. This new project will extend the coverage of WinEcon to include business studies while at the same time adapting all the materials to a Web and Java implementation. Given the lessons learnt on the WinEcon I project, which aspects of the project will the Consortium be approaching differently and which will it carry forward unchanged? The RAD development approach and organisational structure will be carried forward largely unchanged—risks and all. However, three key changes are being proposed by the author:

1. *Instrumentation of the authoring tool.* WinEcon II will be using tools capable of recording development effort, duration and elapsed time. This information will be used to drive effort prediction models as the project progresses and will feed into tracking, planning and scheduling.
2. *Constant end-to-end product visibility.* Using Web technology, it should be possible for developers to either upload their work on a day-by-day basis or to actually author online so that the entire product will be visible to the entire development team at any point in time.
3. *Continuous online review and feedback.* By virtue of end-to-end visibility of the product it will be possible to distribute the review effort more evenly across time. More frequent, smaller reviews will be applied so that errors can be picked up early and correction costs reduced. In so doing, the benefits of inspections and reviews will be maintained and in some cases enhanced.

REFERENCES

1. J. Sloman, WinEcon, software review, *Economics Journal*, **432** (1995) pp. 1322–1346.
2. P. J. Hicks, E. L. Dagless, P. L. Jones, D. J. Kinniment, F. J. Lidgley, P. E. Massara, D. Taylor and L. T. Walczowski, A computer-based teaching system for electronic design engineering, *Int. J. Eng. Education*, **13** (1997) pp. 64–71.
3. S. Price, L. L. Cheah and P. Hobbs, Software quality management in end-user programming and object based rapid application development (RAD), *Software Quality Management IV—Improving Quality, Proceedings of the British Computer Society SQM'96 Conference*, Mechanical Engineering Publications, London (1996) pp. 485–493.
4. DSDM Consortium, *Dynamic Systems Development Method*, Tesseract Publishing, Ashford, Kent (1997).
5. I. M. Marshal, S. Price, P. I. Dugard, P. Hobbs and W. B. Samson, Code-based analysis of the development effort of a large scale courseware project, *Information and Software Technology*, **39** (1997) pp. 541–549.
6. P. Gerrard, Never mind the quality—here's Client/Server, *Software Quality Management Presentations*, SGES Publications and Mechanical Engineering Publications, London (1996).
7. T. DeMarco, T. Lister, *Peopleware*, Dorset House (1987).
8. T. Gilb, D. Graham, *Software Inspection*, Addison-Wesley (1993).
9. I. S. Strauss, R. Ebenau, *Software Inspection Process*, McGraw-Hill (1994).

Simon Price is director of the WinEcon II project. He was formerly senior programmer on the WinEcon project and prior to that worked for eight years in the computer games industry.