AN ADVANCED COURSE ON SOFTWARE DEVELOPMENT ENVIRONMENTS

ANTHONY FINKELSTEIN

Imperial College, Department of Computing, 180 Queens Gate, London SW7 2BZ
acwf@doc.ic.ac.uk

This paper will describe an advanced course on Software Development Environments. The paper will detail the content of the course and will discuss issues in the presentation of advanced and specialist courses in software engineering.

1. Introduction

There is an increasing literature on software engineering education. This literature has however concentrated on overall curriculum issues, introductory courses and basic project work. Relatively little attention, with the notable exception of the work of the SEI Software Engineering Curriculum Project (Ford 1991), has been paid to advanced and specialist courses in software engineering suitable for graduate students.

This paper describes, in detail, the course on Software Development Environments at Imperial College. It considers the particular issues that arise in the presentation of advanced and specialist software engineering courses.

The Software Development Environments course is a 30 hour course consisting of 20 lectures and 10 accompanying problem classes. It is available to MEng degree students at Imperial College as part of the Integrated Engineering Study Scheme described in Finkelstein (1991). The course is required of all students completing the MEng Computing (Software Engineering) and its predecessor MEng Software Engineering. The course is offered during Year 4 of their studies. The course is in addition available as an option to students following other MEng programmes of study and to appropriately qualified graduate students as part of the Imperial College Advanced MSc programme (Foundations of Advanced Information Technology).

The course has also been available to all students in the University of London through the LiveNet interactive video conferencing system. The course forms the basis of an industrial short course.

Below we describe the background of students following the course. We outline the objectives of the course and the means by which student performance with respect to those objectives is evaluated. We consider in some detail the content of the course and explain its structure. We discuss the material required to support the course. We review the use of interactive video-conferencing, its benefits and difficulties. We discuss how advanced and specialist courses can be adapted for use as industrial short courses.

2. Background

The bulk of students taking the Software Development Environments course are those following MEng courses with a specialisation in Software Engineering. The course is also popular with Doctoral students working in the area of software engineering and programming languages who are seeking to extend their knowledge of automated support for software development. The Advanced MSc programme concentrates largely on computing theory and relatively few students have the necessary background knowledge of software engineering to follow the course, these students are admitted to the course based on a short interview.

All students on MEng programmes will already have a solid background in the science and engineering of computation. They will also have completed 4 required software engineering courses, 2 in Year 1 and 1 in each of Years 2 & 3. These courses contain essential prerequisite material.

Year 1

Software Engineering - Programming [I]

> This course introduces "programming-in-the-small", in particular the implementation of programs in both functional and imperative languages. The objectives of the course are: to develop the ability to plan and carry out small program development projects; to use program design techniques; to implement programs using both functional and imperative programming languages; to test and evolve those programs. The course develops an understanding of the programming process. It introduces topics such as higher-order programming and assessing quality of program design. The course is the primary carrier of: informal specification; program design; implementation in functional and imperative languages; testing and debugging.

Software Engineering - Programming [II]

> This course includes an exploration in depth of the abstract data type and its extension to file structures held on storage devices. It seeks to develop the ability to abstract the principal requirements of data structures as appropriate for the required program, from their implementation and aims to give an understanding of standard algorithms for handling them and of their performance. This course is the primary carrier of: file handling; design of data types and algorithms.

Other Year 1 courses introduce predicate logic and discrete mathematics and their use in specifying and reasoning about programs.

Year 2

Software Engineering - Design

> This course introduces software development “in-the-large” in particular software specification and design and develops a familiarity with basic aspects of software development practice, techniques and tools. It aims to develop: the ability to plan and coordinate a small team software development project from statement of need through to implementation and maintenance; the ability to use formatted systems analysis and design techniques; the ability to identify, select and evaluate software development tools. The course builds on students knowledge of “programming-in-the-small” and develops an understanding of specification and design techniques and how they can be deployed as part of a disciplined software development process. The course is the primary carrier of the following: software process; software evolution; system modelling.

Year 3

Software Engineering - Methods

> This course presents a collection of methods, techniques and tools from which students may select when trying to solve problems that are less well defined and larger than they have previously encountered. The aims of the course are: to give insight into the size and complexity of some current software systems; to present methodologies for planning and managing software projects. This course builds on the concepts of programming-in-the-small (Software Engineering - Programming [I]) and software design (Software Engineering - Design). It provides practical knowledge and experience for planning, managing and performing software projects. The course integrates and develops previous concepts notably those of system modelling and illustrates their role in the software engineering process.

Students will have completed a significant group software engineering project in each of Years 2 & 3. The Year 2 group project will include a substantial analysis and design task with appropriate documentation. The Year 3 project, which includes a large implementation, generally requires the use of standard software engineering tools in the Unix environment.

All students in Year 4 will have completed a period of industrial placement between Easter of Year 3 and the start of Year 4. The placement provides training in accordance with the requirements of the Institution of Electrical Engineers (IEE Membership Brief M5). It includes experience of: product and/or service specification; design and development; documentation; procurement, implementation, testing and quality assurance; application (system) engineering; installation, commissioning, operation and maintenance. Students obtain practical knowledge and experience by participating in useful work, in particular through involvement in team-based projects. The placement element of their programme of study and their exposure to “real-world” software engineering practices and problems provides students with a significant motivation to pursue the study of automated support for software development.

3. Objectives

The primary objective of the course is to develop an awareness of the state-of-the-art in software development environments and automated support for software development. The course presents the issues, challenges and practical concerns that underlie the construction and use of such environments. Students completing the course should have the ability to evaluate and deploy appropriate automated support for software development in an industrial setting.

A supporting, but nevertheless important, objective is that students gain an appreciation of the intellectual challenges which the study of software engineering presents. It is essential that students are exposed to the “research edge” of software engineering and are shown problems of equivalent substance to those they encounter in, for example, the theory of computation or parallel computing.

4. Assessment Methods

Assessment is one of the most difficult issues in software engineering education. The assessment of student performance on the course is by examination and coursework. Students sit a 2 hour examination paper and complete coursework equivalent to 3 nominal hours. The weighting of examination to coursework is approximately 80:20.

During the examination students must answer 3 questions which require both a knowledge of the content of the course and a good understanding of the issues which underlie it.
A typical question which students might be asked is:

> Give a short description of the architecture and principles of operation of Popart/Paddle & Marvel respectively. They have some common principles and structural features what are they?

Such questions generally require an “essay-style” answer though “note-form” answers are acceptable. Computing students find this very difficult. They lack the experience of producing written accounts, under examination conditions, which students reading humanities or social sciences have. This significantly limits the effectiveness of examinations as an assessment method for this type of course.

The coursework component particularly tests the skill component of the course objectives. The assignment varies each year but basically involves the preparation of a “strategy” for providing automated support for a small software development organisation which is described in the coursework brief. Students are expected to demonstrate an appreciation of the key factors that determine what type of automated support is appropriate in what circumstances. Particular attention is paid, when assessing the submissions, to whether the students have asked the “right questions” rather than produced the “right answers”.

5. Course Organisation and Structure

The course has an unusual and innovative organisation. It is entirely based round examples and case studies drawn from research and industrial practice. It is divided into units addressing a selected example or narrowly focused set of issues. Units vary in length but are generally about 50 minutes of lecture time. General principles and models are provided through a set of supporting readings.

This “artefact-centred” approach (Carroll 1990) is motivated by some specific concerns about advanced and specialist education in software engineering. Students find abstract models and generalised architectures difficult to understand without solid examples. In any case the state-of-the-art in software development environments does not provide any generally accepted analytical or taxonomic framework. The approach is supported by some more general observations on student difficulties in the study of software engineering which we shall not rehearse here but are considered in some detail in Finkelstein (1991).

The units are interspersed with problem classes which involve a short discussion of the supporting reading and an exercise. There is a large extended exercise which spans a significant part of the course.

6. Course Content

6.1 Units

The course is composed of 15 units. They are preceded by a short introduction to the course and a statement of what is required from students following it. The units are as follows:

Unit 1: Software Engineering Revision

> This unit reviews basic software engineering concepts and aims to establish a shared software engineering vocabulary. The unit links the course to the background courses discussed above.

Unit 2: Overview of Software Development Automation

> This unit gives an overview of software development automation it distinguishes between tools, work benches and environments. The unit gives a very simple reference architecture which students can use during the early part of the course.

Unit 3: The Analyst and IEF

This unit gives a detailed discussion of two example “CASE tools” - The Analyst and IEF. The unit shows how they are used to support software development and considers their architecture, strengths and shortcomings.

Unit 4: Unix

This unit examines the most widely used general-purpose, industrial-strength development environment - Unix. The unit shifts the focus from looking at Unix as an operating system towards looking at it as a software development environment. The unit reviews the advantages and disadvantages of Unix and considers its role in providing operating system support and supplementary tools for other software development environments.

Unit 5: Smalltalk-80

This unit examines the paradigm case of an exploratory programming environment - Smalltalk-80 - and examines its role in the evolution of programming tools, methods and environments. Smalltalk-80 is compared and contrasted with conventional software development environments discussed in the preceding units.

Unit 6: PCTE

This unit looks at PCTE (Portable Common Tool Environment). In particular it examines in depth the PCTE object storage and considers the role of object storage in environment software frameworks.

Unit 7: Pecan, Garden & Field

This unit examines the development of programming environments since Smalltalk-80 through a discussion of the Brown University family of programming environments. The unit to looks in some detail at Pecan, Garden and Field and the approach which underlies them.

Unit 8: Programmers Apprentice

This unit looks at AI applied to support for software engineering in a principled manner in particular the storage and use of “programming knowledge”. The unit examines the assistant approach and its relation to software development environments.

Unit 9: Arcadia & Marvel

This unit examines Arcadia & Marvel which employ the "process programming" approach to software development support. The simple reference architecture introduced in Unit 1 is extended to take account of this approach

Unit 10: Aspect & IPSE 2.5

This unit reinforces and builds on the examination of the process modelling centred architectures of Arcadia & Marvel (Unit 9) by looking in detail at two UK software development environments Aspect & IPSE 2.5.

Unit 11: Popart /Paddle

This unit, through an examination of Popart/Paddle, considers formal models of the development process and their use in software development environments. It discusses support for formal (transformational) software development.

Unit 12: ISTAR

This unit examines ISTAR, a "federated environment" based on an abstract model of software development and of task sharing.

Unit 13: ESF

This unit considers the ESF (Eureka Software Factory) technical architecture for software factories. The unit looks at the software factory concept and how it is being realised within ESF.

Unit 14: User Interface Issues

This unit analyses the user interface issues which are important for software development environments and examines approaches to handling these issues. UIMs (User Interface Management Systems) are considered and some examples reviewed.

Unit 15: Commercial Issues

This unit examines the commercial issues which influence the architecture and use of software development environments.

The course concludes with a revision session which revisits the Units and considers their key points.

Students receive printed notes summarising the material covered in the units. They are expected to supplement these with their own notes.

The units are carefully organised in sequence. They start with “environments” which the students have are already been introduced to: CASE tools, Unix, integrated programming environments and look at their architecture and the principles which underlie their construction. The evolution of these types of environments are considered by looking at PCTE, which gives scope for discussing object management, and the Brown University family of programming environments, which gives scope for discussing environment generation and extension. The idea of basing automated support on a systematic understanding of the development process is introduced through Programmers Apprentice and fully detailed in the analysis of process modelling in Arcadia, Marvel, Aspect and IPSE 2.5. The idea of development knowledge is used to relate transformational environments, such as Popart/Paddle, to software development environments in general. The issues of scale and distribution of an environment are considered through an examination of ISTAR and ESF. Lastly two key sets of issues which determine the shape and success of software development environments are considered.

6.2 Problem Classes

There are 10 problem classes associated with the course. These problem classes are used for: discussion and questions arising from the units; exercises completed individually and in small groups; directed discussion of the readings; small demonstrations and hands-on sessions.

Exercises

There are 3 main exercises, two of which are extended over several problem classes:

Exercise 1: Revision Debate

> Students are asked to debate the merits of different software engineering paradigms (for example, “This House Believes that the ‘Waterfall Approach’ is the Best Way to Build Large Software Systems”). A formal debate structure is established with speeches on each side and questions from the floor. A vote is taken. Particular emphasis is placed on the implications of the different paradigms for automated support.

Exercise 2: Evaluation

> Students are asked to work in small groups (3 students) and to prepare a form which can be used to help inexperienced software engineers to select a CASE tool for their work. The groups exchange forms and, using packs of publicity material describing commercial CASE tools, prepare evaluations based on the form they are given. The students review progress and discuss the strengths and weaknesses of their own forms and those of others. They also discuss the respective merits of the various tools.

Exercise 3: Process Modelling

Students are asked to work in small groups (3 students) and are given a description of a small, but realistic software process. It focuses on the designing, coding, unit testing, and management of a rather localized change to a software system. The problem is taken from the literature (Kellner et al. 1991). Students are asked to identify the tools that might be involved in carrying out this process. Students are then asked to produce a data flow diagram of the process. Once this is completed students are required to produce a data model of this software process. Lastly students are asked produce a formal specification of the process using a simple pre-condition [action] post-condition scheme (similar to that used in Marvel).

Readings

There are 9 readings that accompany the course and which form the required preparation for the problem classes. In addition particular papers may be recommended to amplify specific units (for example, Wile (1983) to support Unit 11).

The readings are:

[1] The Byte Staff (1989); Making a Case for CASE; Byte; December 1989, pp 155-171.

[2] Sheil, B. (1983); Power Tools for Programmers; Datamation; February 1983, pp 131-144.

[3] Rich, C. & Waters, R. (1988); Automatic Programming: myths & prospects; IEEE Computer; August 1988, pp 40-51

[4] Osterweil, L. (1987); Software Processes are Software Too; Proc. 9th International Conference on Software Engineering; pp 2-13, IEEE CS Press.

[5] Finkelstein, A. (1989); Not Waving but Drowning: representation schemes for modelling software development; Proc. 11th International Conference on Software Engineering; pp 402-404, IEEE CS Press.

[6] Green, C.; Luckham, D.; Balzer, R.; Cheatham, T. & Rich, C. (1983); Report on a Knowledge-Based Software Assistant; Kestrel Institute Report.

[7] Scacci, W. (1987); The USC System Factory Project; USC Technical Report CRI-87-67.

[8] Myers, B (1989); User-Interface Tools: introduction & survey; IEEE Software; January 1989, pp 15-23.

[9] Perry, D & Kaiser, G. (1989); Models of Software Development Environments; Proc.

10th International Conference on Software Engineering; pp 60-68, IEEE CS Press.

A guided discussion of each reading is held which attempts to bring out the key issues and tests students understanding of the material.

Demonstrations

Where practically possible we have sought to provide demonstrations of, and hands on sessions with, some of the environments discussed in the course. These have included The Analyst, PCTE, Smalltalk-80, Marvel, and ISTAR. This is an area in which we hope to extend and improve the course.

Video-tapes

An early version of this course was based round seminars given by key developers of software development environments. These seminars were video taped and form an additional resource which students on the course are encouraged to take advantage of. The tapes include: Frank McCabe on MacProlog; Bob Snowden and Clive Roberts on IPSE2.5; John Darlington on Transformational Environments; Peter Hitchcock on Aspect; Vic Stenning on ISTAR.

7. Interactive Video Conferencing

Providing a range of specialist and advanced courses is always difficult. It is sometimes difficult to justify courses with relatively low take-up and software engineering staff are in demand to teach the important introductory courses and supervise projects. We have been experimenting with the use of interactive video-conferencing to make the provision of such specialist material more economic and to share scarce expertise.

The Software Development Environments course was made available on LiveNet, the University of London video conferencing facility which links the major schools of the University. This facility employs wide-band fibre optic links to give interactive real-time video with all participants at each of the six available sites able to see and hear each other. Some lectures on the course were attended by students from University College London and Royal Holloway & Bedford New College. The full course was attended by students from Queen Mary & Westfield College. Notes for the course were distributed in advance to designated coordinators at each school. Electronic mail was used for questions and discussions. So far we have not used LiveNet to support problem classes though we see no problem in doing so.

Students adjusted quickly to the new medium and we regard the experiment as successful. The only limits are in the use of complex graphics and diagrams - careful use of notes can circumvent this problem.

8. Industrial Short Course

This course has been adapted for use as a short course in the Imperial College Computing Forum, a "club" of industrial and commercial organisations who support research and education in computing at Imperial College. This "dual-use" is an important benefit of advanced and specialist courses in software engineering. Exposing experienced software engineering practitioners to course material results in improvements to the course and its presentation which benefits students. In particular students taking the short course variant of the Software Development Environments course showed particular interest in the issues of standardisation and management control of software development. This has been more heavily emphasised in revised versions of the Software Development course.

In adapting and giving the course we have learned: particular care must be paid to ensuring that attendees have a shared software engineering vocabulary; presenting in a concentrated period allows a much quicker coverage of material; there is limited scope in an industrial short course for exercises and discussions, more emphasis needs to be placed on the units; it is of considerable benefit to use the experience of attendees by being open to questions and allowing attendees to determine the emphasis to be placed on different parts of the course.

9. Advanced and Specialist Courses in Software Engineering

The Software Development Environments course described above demonstrates that advanced and specialist courses in software engineering can provide an intellectually demanding and useful addition to both specialist software engineering students and as a component of a general computing education. There are obvious benefits to industry of advanced education in software engineering of this type. Other examples of courses which could be organised include: software testing; requirements engineering; software economics; software quality.

Experience has taught us that prerequisite to advanced and specialist courses in software engineering is a solid introductory programme and that some industrial experience on the part of the students is vital.

An effective way of introducing the material is through an example-based approach. This must be complemented by an appropriate scheme of readings and discussions. Students gain from being given readings from the literature - it enhances their knowledge gathering and critical skills. Particular attention should be paid to the development of exercises and problems which is the hardest part of the design of advanced and specialist courses. A problem in the assessment of advanced and specialist courses in software engineering is students poor writing skills. This issue needs general attention in the development of computing curricula.

The economics of specialist course provision can be made more favourable by adapting advanced and specialist courses to build industrial short courses and by multi-institutional course sharing.

Video conferencing shows promise in this area.

The health of software engineering as a field of research depends on students selecting it as a topic for study at a doctoral level. Students will only do this if they have seen that there are open problems and difficult issues worthy of their attention. They will not get this from introductory courses alone.

Acknowledgements

I would like to thank the students who have taken the Software Development Environments course for their ideas, participation and stimulating questions. I wish to thank my colleagues Jeff Kramer, Manny Lehman & Jeff Magee for their suggestions and comments.

References

Carroll, J. (1990); Towards an Emulation-based Design Theory; Interact ‘90; North Holland.

Finkelstein, A. (1991); Student Problems in Software Engineering Education; IEE Colloquium on The Teaching of Software Engineering; Digest No: 1991/034.

Finkelstein, A. & Kramer, J. (1991); An MEng Programme of Study in Software Engineering; Proc. 1st National Conference on Software Engineering in Higher Education, pp128-151, SIHE.

Ford, G. (1991); 1991 SEI Report on Graduate Software Engineering Education; Software Engineering Institute, Carnegie Mellon University; Technical Report CMU/SEI-91-TR-2.

Kellner, M.; Feiler, P.; Finkelstein, A.; Katayama, T.; Osterweil, T.; Penedo, M. & Rombach, H. (1991); Sofware Process Example; Proceedings of the 1st International Conference on the Software Process; pp176-1187, IEEE CS Press. (Also 6th International Software Process Workshop)

Wile, D. (1983); Program Developments: formal explanations of implementations; CACM, 26(11), pp 902-911.