Tools for a Software Engineering Course

J. M. Verner
Department of Information Systems
City University of Hong Kong

Abstract

There are a number of major factors that heavily impact on the tools that we are likely to find useful in a software engineering programme. These factors include how we define software engineering, the background of the students, whether the programme is a graduate or undergraduate one, whether the students are part-time or full-time, and the number of semester hours we are able to devote to software engineering courses. In this short paper we discuss the kind of software engineering courses offered within an undergraduate programme that is part of an Information Systems degree, and the tools that we have successfully used within this programme.

1 Introduction

How we define software engineering, the research interests and expertise of the academic staff involved in teaching a software engineering programme (and given that the area is so large), what specific aspects of software engineering we consider should be emphasized and supported by practical work, all impact on the choice of tools that we use, or wish to use, in our software engineering courses. Different national cultures, attitudes and expectations in governmental, commercial and industrial sectors, and the software development approaches currently used, will also affect the focus of the course and hence the tools we may wish to consider. Hong Kong has a major banking and financial sector and the economy depends heavily on financial and data communications software. Other important business sectors in Hong Kong include transportation, manufacturing, government and tourism.

Other major factors that impact on our choice of tools include the background of the students, whether the programme is a graduate or undergraduate one, whether the students are part-time or full-time, and the number of semester hours we are able to devote to software engineering courses. In the following sections we consider these factors in more detail and then describe the practical work and tools that we believe are appropriate. We conclude with a brief look at possible future directions.

1.1 Background of the students

Two practical considerations heavily influence what we can reasonably address in a software engineering programme. The first consideration is whether the programme is to be taught to graduate or undergraduate students. The second consideration takes into account the minimum background that we can safely assume for the students in the course.

If the students are graduates we may demand as entry criteria an undergraduate degree in computer science or information systems, and a number of years of industrial experience. Even if we can select from hundreds of applicants, and even if we only select those with very high undergraduate grades we may not be fully cognizant of either the content or the quality of their undergraduate programme. A Master's student in Hong Kong, is very likely to have an undergraduate degree from a university outside Hong Kong because the number of university places in Hong Kong, until very recently, was severely limited by the Hong Kong Government. For example, this year, in one of our Master's programmes (MA (IS) in software project management), we had approximately 320 applicants for 30 places. Of the students selected, one third have degrees from the UK, one third from the U.S., Canada, or Australia, and the remaining one third have graduated from Hong Kong Universities.

In many places, and Hong Kong is no exception, Master's level software engineering teaching implies evening classes with part-time students experienced in software development. They are (almost) all employed in some capacity in the software industry. In this case we expect the students will have familiarity with some commercial development tools and techniques. Unfortunately they are usually different tools and frequently different techniques. A serious problem is that the students do not have vast amounts of time to spend in both travelling to the university, and in learning and using the new tools

we may wish to provide. In many cases, we have to select project work that allows them to use their employers' tools or else to apply some techniques we wish them to investigate on applications that have been, or are in the process of being, developed by their organisation.

If the students are undergraduates we are in a slightly better situation in some respects. They are normally full time students so are unlikely to have the same (after work) travel problems as the graduates. We also have a more consistent though lower base from which to begin our software engineering courses as we have more control over required prerequisite courses. The difficulty here is that most students have little experience of real world software development and this results in some important software engineering concepts being difficult to teach. In addition, in an undergraduate degree programme, in information systems or computer science, we are generally limited to a maximum of two software engineering courses. A question that perhaps should be asked here is "Is it really possible to teach software engineering to undergraduates?". However, this is not a question that we intend to answer here, other than to note that many of us try to do so [10].

The ideal situation would appear to be one similar to that attempted at the Wang Institute of Graduate Studies, experienced students in full-time Masters' programmes.

1.2 Definition of software engineering

The definition of the term "software engineering" by the academic staff involved can have an impact on our programmes. We take a process view of software and in this we agree with Tate [9], who regards it as a set of products (such as requirements, designs, source code modules, documentation, test files, etc.) produced by an instance of the software process. Hence we also agree with his definition of software engineering which is:

> "The application of engineering principles to the software process, including the formulation of standards, the use of scientific methods and adherence to agreed professional practices and codes of conduct, where possible."

Software engineering and allied courses are mainly taught within a computer science department although in some institutions (certainly in the Asia-Pacific region anyway) an information systems department may teach the same or very similar courses [10]. This often depends on the orientation of the Computer Science and Information Systems departments and their staffs' research interests. In some institutions the software engineering courses are taught jointly by both departments.

Software engineering is sometimes treated as a small part of a computer science course and then what is offered can be very selective, sometimes concentrating almost exclusively on formal methods and related matters, such as proof of correctness, or on object-oriented development and the concepts of reuse [9]. When offered within an information systems department, the emphasis is typically on the management of software development and the course is usually approached from a more traditional business-oriented point of view under titles such as *Project Management and Quality Assurance*.

We consider the following to be some of the key areas of software engineering that we need to address in any such course: software process and its improvement, project management and software development paradigms. However, given the process view of software defined earlier then we believe that the critical focus in software engineering education should be on:

systems - integration - evolution - quality - productivity - measurement and management

This view may be criticised for not including program correctness, tools, CASE, environments, and technologies. However, although these need to be taught somewhere, we do not consider them the major part of software engineering. Programs are not systems and if the student cannot understand where and how tools, environments and technologies should be used and how they affect our systems then we are probably wasting or time. A program does not scale up into a system and the problems encountered in developing a program are quite different from those encountered in developing a system [9].

1.3 Number of semester hours

There are a number of problems inherent in designing software engineering courses. A major problem is in selecting and prioritizing the material for inclusion in the course. How much can be included depends on whether we are talking about a single course, several courses or a complete degree

programme. If we are talking about a complete degree programme we can offer a much broader coverage of the area and include a number of different and innovative tools. If we have only a limited number of courses and/or semester hours then an overriding factor may be the learning curve for a particular tool as this will have a significant impact on our educational objectives. We do not want to train our students in the use of a particular tool but to use the tools to show how such aids impact on our software development (or maintenance), or to illustrate key principles.

1.4 Research, education and training

We also need to carefully consider what our educational objectives are, how much we can achieve and how we can best achieve the objectives. We must find a suitable balance between research and practice in what we teach and need to consider the gap between research and current practice. Our courses cannot reflect the state of the research, nor can we take too narrow a view as our graduates are then likely to be of little use in the real world [9]. Comments, made by one of the large employers in Hong Kong, have suggested that some of the earlier computing graduates were in this category.

The function of a university is education rather than training so we also cannot teach courses that reflect the state of the practice or the skills of our students will be out-of-date by the time they graduate. Graduates need the ability to adapt to new technologies and techniques when they arise and to be able to change as they rise to more senior positions [9].

2 Project work

To provide the required level of software engineering knowledge and understanding we must include in our course software development project work, project management, and a careful end-of-project analysis that identifies the successes and failures of the development exercise. This can best be done if we include a major piece of systems development, that is, a group project in our course.

If we follow Bjorner [4] then we also need to make sure we address requirements conformance tracking, version control and configuration management, test case generation and validation, change request recording, monitoring and control, design decision recording and tracking, software creation installation and operation, and quality assurance and planning monitoring and control. A group software development project will also provide some practical experience for much of this material. Bjorner's discussion of software engineering also fits in well with the CMM level 1 key process areas [6].

Of course group project work can be difficult from the lecturer's point of view. The projects are hard to assess, are time consuming, and the project itself must be large enough to be meaningful. However, we also must make sure that there is not so much programming that our real educational objectives are lost (for example, not enough time to properly address testing and quality issues). The programming problem may be overcome if we either use tools that reduce programming effort, or we cut down the required effort in other ways, such as supplying a suitable library of domain objects.

A course such as the one that we are describing here is frequently offered as a final year undergraduate course. It can be quite difficult to teach such a course in the Hong Kong context as most undergraduate degrees here are of three years' duration. The University Grants Committee, which is the major source of university funding in Hong Kong, has been unwilling to change this.

In order to gain the most from such a software engineering course the students need the following prerequisites.

- Programming - enough programming to be both competent and comfortable in several programming languages. This means that they have learned at least two 3GLs (including C), and a 4GL. Our students currently have the opportunity to use COBOL, C, Visual Basic, Prolog, Foxpro and SQL, though not all are compulsory. They have written programs on their own in some cases up to about 2000 lines of code (LOC);
- A systems analysis and design course with hands-on experience of a suitable CASE tool to support the methodology used (currently we use SSADM, a structured analysis methodology);
- An OO course;
- Database - both design and implementation;
- Data and file structures course;
- Data communications.

Some students may have already taken an optional user interface design module. This can be useful for the whole group as we feel that it is important that the project selected for the implementation is interactive. We believe that a serious consideration of the characteristics of the user and his work habits is important for everyone involved in applications development.

As we discussed earlier our course must concern itself with systems, their integration, evolution, quality, productivity, measurement and management and the tools we require are to support these processes. We are concerned with the implementation and management of systems larger than those that single student can build on their own in the time available. We believe that by being involved in the development of a group project the students are exposed to problems that they do not encounter if their experience is limited to single developer exercises as, as was noted earlier, single programs do not scale up to systems.

For the group project the students should (in groups of 5 or 6) design, develop, test, implement and document, on time and within an effort budget that they have set for themselves, a major system in a suitable language. In general what language is considered suitable will depend on the students' backgrounds, skills and knowledge of the available languages. In the past we have used 4GLs (such as Oracle and ALL), and this has worked very well. The students must be given tight specifications, with absolutely no room for gold plating. We usually prefer to assign students to groups ourselves and have found that this has worked better than allowing them to form their own groups [10]. With good knowledge of the student's background it is often possible to assign the class into teams that have a greater variety of skills, for example, a good coverage of systems analysis, programming, database, data communications, user interface design, etc. We have found that by allowing students to form groups with their friends that they often have too limited a range of skills and this can lead unsatisfactory group experience [9].

3 Tools

In addition to the programming language used other tools are required. Groups must estimate their own effort, schedule, staffing levels, and milestones, in addition to monitoring the project throughout the development. For this they need project management tools. We believe that, by insisting that groups set their own schedules, the students learn far more about real world development and its problems than if we set schedules for them. At this stage, and depending on the programming language used, a group may decide that the project cannot be completed in the available time and may decide to implement only part of the specifications; or they may decide to use an incremental development approach. Whatever approach they take, incremental development, partial development or the entire development, the group's choice must be justified and documented.

We have successfully used the estimation package Before You Leap (BYL) [3] to help with the effort and schedule estimates, and MS Project [5] to provide PERT and Gantt charts. The PERT charts allow a student group to determine the impact of slippages on delivery date [7]. BYL is made up of several components including a size and effort estimation feature that uses Function Point Analysis [1] (MarkII version [8]) and the cost estimation model COCOMO [2] (several different versions). Students are encouraged make a number of different kinds of estimates and to use these in setting their project plans. We have found that by using Function Point Analysis to help plan their project students have a much better understanding of when this technique is appropriate and the differences between the standard function point count, and the MarkII version [8]. By using COCOMO they must deal with lines of code estimation and, because they are developing their system in a 4GL, they get a better understanding of both the usefulness and the difficulties inherent in lines of code counting and estimating. In addition COCOMO allows a form of game playing so the students can permute the project deadline and examine its impact on the overall cost [7]. This helps them better understand and control the factors that impact project effort and schedule.

Actually getting a good set of specifications for the students can be both difficult and time consuming. We have found that it can work quite well if students use the specifications they have developed in an earlier systems analysis course which uses a CASE tool, System Engineer (a LBMS product). This circumvents some of the time that can be spent in coming to grips with the specifications if we give them a project that they have not seen before and/or the specifications are to be developed for an unfamiliar applications environment. Another way of overcoming this problem is to present the class with a poor but working prototype of the system that they are required to develop [10]. If a suitable 4GL is not available it is possible to use a 3GL so long as a properly documented library of components is available.

Other techniques and methods must be used with the implementation of the project and its quality assurance. However, we have not used any tools to support them so will not go into detail here. We also expect a good set of documentation to accompany the completed project. To help with documentation and

other requirements a word processor, spreadsheets, statistical packages, etc. are available as and when required.

4 Future tools

We are currently investigating the use of an integrated set of CASE tools suitable for client server applications (these products are integrated with our analysis and design tools). A number of integrated tools including reverse engineering, a program generator (Visual Basic), configuration management tools, database design tools, and an integrated project management tool set are available. We have not at this stage made a final decision whether or not to adopt this development environment. Our decision will be based on both our estimation of the learning curve involved and if we can introduce the parts of the tool set into in our other prerequisite courses. If we decide to use this tool we will not use it as the only tool for the software engineering course. We will continue to use BYL to provide additional project estimates. We are also investigating the use of some risk analysis tools and hope that we can obtain a toolset for use in the next academic year.

5 Conclusion

The choice of tools used in our software engineering courses is based on our belief that software engineering is the application of engineering principles to the software process and that the management of this development process is critical. To achieve our goals we believe that the students need to be involved with a project that is large enough to require the serious and sustained use of good management practices. Any tools used must not have an excessive learning overhead as this is likely to detract from the course's major educational objectives. We have not found a single toolset that is appropriate for everything we wish to teach and think that it is highly unlikely that a single tool will cover our current or future requirements.

Note: These comments are based on software engineering courses that the author has been associated with in both New Zealand, and Australia, and recent discussions regarding changes to the information systems programme at City University of Hong Kong.

References

[1] A. Albrecht and J. E. Gaffney, "Software Function, Source Lines of Code and Development Effort Prediction", IEEE Transactions on Software Engineering, Vol. SE-9 no. 6 November 1983, pp. 639-648.

[2] B. Boehm, Software Engineering Economics, Prentice Hall Englewood Cliffs NJ, 1981

[3] Before You Leap, Strategic Systems Technology Ltd., London, UK.

[4] D. Bjorner, "University Curricula in Software Technology", IFIP Transactions on Software Engineering Education, (eds. B. Z. Barta, S. L. Hung and B. Cox, December 1993, pp. 5-16

[5] MSProject, Microsoft, California, USA.

[6] M. Paulk, B. Curtis, M. B. Chrissis, Capability Maturity Model for Software, CMU/SEI-91-TR-24 , Software Engineering Institute, Carnegie-Mellon University, Pittsburgh, PA 15213, 1991.

[7] Roger S. Pressman, Software Engineering, A Practitioner's Approach, McGraw Hill, Third Edition, 1992.

[8] Charles R. Symons, "Function Point Analysis: Difficulties and Improvements", IEEE Transactions on Software Engineering, Vol. 14, no 1, January 1988 PP. 2-10.

[9] G. Tate, "Software Engineering Research and Practice - Implications for Education", IFIP Transactions on Software Engineering Education, (eds. B. Z. Barta, S. L. Hung and B. Cox), December 1993, pp. 163-174.

[10] J. M. Verner and E. G. Todd, "Experiences with the Organisation and Assessment of Group Software Development Projects", IFIP Transactions on Software Engineering Education, (eds. B. Z. Barta, S. L. Hung and B. Cox), December 1993, pp. 309-315.