Software Engineering 10

Teaching introductory and advanced courses in software engineering

IAN SOMMERVILLE

Every course is different and what's included depends on the interests of the instructor, the background of the students, the type of software being discussed and the resources available. There is no right or wrong way to present a software engineering course so I have designed the book so that it can be used in a variety of different ways in both introductory and more advanced courses.

The way that I think about designing a course is to think about three fundamental learning areas that I want to cover:

1. Principles - what do I want students to understand that is fundamental to a large area of the discipline and (largely) independent of technology.

   For example, the notion of configuration management (CM), what's involved, why it's essential is a software engineering principle. We need CM in all kinds of software engineering project and have done so since we started writing non-trivial software systems. There are lots of CM approaches and technologies that embody these fundamental principles.

2. Skills and technology - what students have to do to apply software engineering principles in practical software development.

   This covers things like programming, APIs etc. and to continue the CM example, say the use of git and github for practical configuration management. These need to be applied in the context of practical work where students complete requirements and design exercises and/or write programs.

3. Awareness - things students need to be aware of but which are, in practice, impossible, meaningless or artificial in a university environment

   Most project management issues fall into this category as do some more advanced technologies such as safety engineering. In a CM context, an example of awareness might be continuous integration and deployment - you could do this as a practical topic but it doesn't really mean much when you don't have dependent teams and customers.

The other key issue that you have to think about is the kind of software that you are considering. The engineering techniques that you use for software products are completely different from those used in projects to develop real-time control systems or large-scale enterprise systems. This is not well-understood by students and I have made a point in Chapter 1 of discussing the different types of software and the fact that different types of software engineering techniques are required. I think it's pretty important to make students aware that one size does not fit all and that we need a range of software technologies to cope with different kinds of system.

Too many people who are promoting their own techniques present a view of software engineering that suggests there are universally applicable techniques. This is complete nonsense and anyone who suggests that some technique is universal either doesn't understand the discipline or is a charlatan.

The book's focus is project-based software development where specialized systems are developed rather than product based applications. This reflects my background and experience in critical systems and large-scale systems. Some of the material is, of course, relevant to product development especially the sections dealing with agile development.

Introductory Courses

In introductory courses, I believe that most of the course should focus on principles and on skills and technology focused on applying these principles. There is little point in making students aware of more advanced issues if they are only just starting in the area.

The key issue in designing an introductory course is 'what do the students already know?' I don't think that software engineering courses work unless students have completed at least an introductory course in programming and have some experience of using a programming environment such as Eclipse. I think it much better if they have also completed courses in algorithms and data structures and have written several non-trivial programs themselves. They have then some understanding of the issues that can arise when developing software.

A difficulty that I have found in presenting introductory courses in software engineering is that students now have a view that equates software engineering with coding, especially coding apps. This has been exacerbated by political statements about the need for coders and teaching coding in school. Students find it hard to understand that there are many different types of software and, for large-scale systems and many real-time systems, programming is only a small part of the problem.

I try to get round this by asking students how many software systems they think they have at home – its usually well over 30, mostly real-time systems. I then talk about the key characteristics of these systems and how they differ from apps.

Topic- based courses

One approach that you can adopt to the teaching of software engineering is to take a topic based approach where you present an overview of topics such as requirements engineering, design, testing, etc. The first nine chapters of the book can support this approach, with all fundamental life-cycle topics covered in Part 1. In theory, you could fit all 9 chapters into a 12 week, 1 semester course but I think this would be pretty rushed.

The problem with trying to cover too much in a 1 semester course is that you spend all your time presenting material so that students acquire quite superficial knowledge of the course topics. Essentially, your focus is on awareness rather than principles or skills. There isn't enough time to discuss examples and case studies and I think that these are essential to bring a course to life and to demonstrate the importance of software engineering.

The conventional approach to a topic-based course is to base this around the fundamental software engineering activities of requirements, design, development, testing and software maintenance. You start the course by introducing software engineering (Chapter 1) and software processes (Chapter 2), then cover the other life-cycle activities with the following chapters in Part 1. You can pick and choose topics depending on your own background although I recommend that you always include requirements engineering, architectural design and, unless it has already been covered in a programming course, software testing.

An alternative approach to a life-cycle based course is to use an 'inverted' approach where you start with programming (which students already know) in the middle of the life cycle than move 'outwards' so that you end with requirements and maintenance. I haven't tried this personally but I think it might be more effective than a life-cycle based course.

Your starting point would be Chapter 7, where you might discuss issues such as host-target development, configuration management and open-source software. You could follow this with Chapter 3 (Agile methods) and Chapter 8 (Software testing), with a focus on test-driven development. I would then move from there to architectural design (Chapter 6) and finish the course with a discussion of requirements engineering and why it is important. I don't think that modeling and agile development go well together but if you are interested in this topic, this could be introduced after a discussion of agile development when you might cover approaches to the high-level design of software.

I think that examples and case studies are critically important in topic-based courses. The difficulty in all examples is that students do not understand the business or operational domain – they don't understand how business works, hospitals, etc. For this reason, many instructors fall back on university examples – student record systems, course selection systems, etc. I don't like these examples much as they are quite artificial and (in fact) most students don't understand the university domain either.

While I have provided some information about various types of system, I recommend that you chose examples for your teaching that you know and understand. You can then talk about the issues and problems that arose and where software engineering techniques did and did not work. I worked on a system like the Mentcare system and I developed the high-level architecture for the iLearn system so these are my preferred examples; my wife is diabetic, so I know a bit about insulin pumps.

Project-based courses

Project-based courses focus around the development of a software project, ideally a group project undertaken by groups of three or four students. The best type of project is a large project, such as the development of the iLearn digital learning environment that I cover in the book. This can be split into a number of sub-projects, with each of these sub-projects undertaken by a different group.

The key benefit from this approach is that students learn about how to work in a group and also the importance of inter-group collaboration. Each group has to design and publish an API so that other groups can use their software; there may have to be collaboration on the system specification (who does what) and there needs to be an understanding of configuration management and agreement on the tools to be used.

The basic difficulty that students have here is that they can spend far too much time arguing about collaboration and not enough time in development. Therefore, as an instructor, you need to provide some guidance and to think in advance how a major project can be split into sub-projects (leaving this to inexperienced students will not usually work). It helps a lot of students are used to using a development environment, such as Eclipse. If not, you will have to set up tools for use and provide clear instructions on how to use these.

Given the timescale involved, project-based courses are inevitably based around agile development. Relevant chapters of the book are therefore Chapter 3 (Agile software development), Chapter 4 (Requirements engineering), Chapter 6 (Architectural design), Chapter 7 (Design and implementation), Chapter 8 (Software testing) and Chapter 25 (Configuration management). You may also include some material from Chapter 23 (Project planning) as some instructors like to introduce notations such as bar charts at this stage.

The difficulty with agile development, of course, is that it requires close customer involvement with the development team. If you have graduate students to help you, then

they can be briefed on the project and can act as customers. Otherwise, it is up to the course instructor and this can be very demanding and time-consuming. You need to be very specific about times for student interaction and to make clear that you are not available 24/7 to answer students' questions.

Project-based courses, when they go well, can be very effective but they can be riskier than other kinds of course. Sometimes, groups simply do not gel in the time available, with some people in the group doing less work than others. In those circumstances, students do not enjoy the work and it is important to try and bring out emergent difficulties as soon as possible. For this reason, I suggest that some group meetings should be timetabled rather than left to the group themselves and that the instructor should sit in on group discussions, intervening when necessary.

An approach that I trialed (once) was in a 2 semester course where each group developed some software in the 1st semester than handed this over to another group for maintenance. I acted as a customer and invented a set of changes that were required. Students were very unhappy with this approach as some groups felt that the software they were asked to maintain was not as good as the software that they had developed and they felt that they were at a disadvantage to the group maintaining their software. This was a fair point and I abandoned the approach part-way through the semester. If you want happy students, I recommend that you don't do this.

Ideally, a project-based course should be taken immediately after (or alongside) a life-cycle based course so that students have some understanding of requirements, design and testing. In St Andrews, students took an introductory SE course in semester 1, with a group project course in semester 2. As I discuss in my use of video document, you could use ask students to watch videos as background before class discussions, about the project work.

Advanced courses

These are courses where students have already taken an introductory software engineering course so are usually delivered in the final year of an undergraduate computer science degree or in a Masters degree course.

In general, for an advanced course, you can use material in parts 2, 3 and 4 of the book as the core of the course but you should add some extra reading so that students can explore some topic or topics in more depth.

I like to think of an advanced course following a T-model. The horizontal part of the T introduces a number of related advanced topics then one or two of these topics are expanded with further reading, practical work, etc. (the vertical leg of the T).

You can construct your own advanced course by picking selected chapters and augmenting that with new material and addition reading (the further reading section in each chapter can be a start here). Some of the new material in the 10th edition has come from advanced courses in systems engineering and in critical systems that I have taught.

Over the past few years, I have taught three advanced software engineering courses using material from the book. These are reuse-based software engineering, systems engineering for large-scale complex IT systems and critical systems engineering.

Reuse-based software engineering

This was an advanced course that I taught where the focus of the course was software development with and for reuse. This was based on the 9th edition but the relevant

chapters of the 10th edition are Chapter 15 (Software reuse), Chapter 16 (Component-based software engineering), bits of Chapter 17 (distributed software engineering) and Chapter 18 (Service-oriented software engineering).

Within the course, I explored two topics in more depth. These were ERP systems and the construction of enterprise systems by configuring these systems and service-oriented software engineering.

Most students have no previous experience of ERP systems so much of that part the course was concerned with simply explaining what these systems were and how they were configured. I drew on my own experience of such systems to discuss the problems that frequently arise and why these problems occur.

The other topic that I explored was service-oriented software engineering where the practical work of the course involved implementing a set of services that could process and analyze e-mail messages. This required students to learn how to implement services in Java. At that time, the implementation was based around 'traditional' web services but if I was doing the same course now, I'd use RESTful-services.

I asked students to publish their services then created my own test data to run against these services. Very few managed to create services that successfully processed all of my test data.

Critical systems engineering

I have taught critical systems engineering courses for many years and have adopted two different approaches in these:

1. A life-cycle based approach where I covered topics such as critical systems specification, critical systems design, implementation technologies, verification and validation, etc. I used the example of the insulin pump system throughout this course. The area that I explored in more depth was safety-critical systems requirements engineering. Safety and security were mentioned throughout the course.

2. More recently, I changed the structure of the course to be topic based: a general introduction (Chapter 10), System reliability (Chapter 11), Safety engineering (Chapter 12), Security engineering (Chapter 13) and Resilience engineering (Chapter 14). Topics that I explored in more depth were the ethics of safety-critical systems development and cybersecurity.

I make extensive use of cases studies in this course (see link below) and have experimented with using video alongside the lectures in the course. I briefly discuss this in my document on using video in teaching;

Practical work in this area is quite difficult to set because students have little background knowledge of the area and because developing a critical system as part of a course is practically impossible. I have used a number of examples:

1. Survey based work where students are asked to explore some area (such as safety-related failures) and write a report on this.

2. Critical systems specification where I have used the Mentcare system case study and asked students to analyse the requirements and to find conflicts in these requirements. Previously, I have used the insulin pump case study and have asked students to develop a partial implementation and make arguments why it was safe.

   The website and teaching material for the course is:

http://iansommerville.com/systems-software-and-technology/courses/critical-systems-engineering/

Systems engineering for large-scale complex IT systems

I have taught part of a Master's course in systems engineering for a number of years and some of the new material in the 10th edition of the book is based on my experiences with this course. This course focused on large-scale IT systems, which are inevitably organizational systems of systems. It is a very good complement to programming-based courses as issues apart from programming dominate this type of systems engineering.

The key topics that I covered in this course were an introduction to systems engineering (Chapter 19), system requirements engineering (Chapter 4), , socio-technical systems (partly covered in Chapters 10 and 19 but with additional reading), systems of systems (Chapter 20) and resilience engineering (Chapter 14). My colleague covered topics such as architectural design (Chapter 6 is partly relevant), methods such as TOGAF (briefly introduced in Chapter 20), data management and integration plus a number of case studies.

Practical work for my part of the course involved developing a conceptual design and a set of outline requirements for a road tolling system where drivers were charged according to distance travelled on a road.

The website and teaching material for the course is:

http://iansommerville.com/systems-software-and-technology/courses/systems-engineering-for-lscits/