Chapter 5
A Holistic Approach to Software Engineering Education

Tugrul Esendal
De Montfort University, UK

Simon Rogerson
De Montfort University, UK

ABSTRACT

This chapter introduces a final-year software engineering module that brings elements of software quality, professionalism, and ethics into one coherent teaching/learning unit.

The rationale for the module is simple. The evolution of Information Technology has led to software being pervasive in today's society. Everyone is either a direct user of software or a recipient of its services. This puts the spotlight on software engineers to deliver fit-for-purpose software that ensures beneficial outcomes for all. However, this is not so easy to do, as evidenced by the many software disasters of varying severity.

There is, consequently, a demand for professionalism of the highest order, which in turn demands a new approach to software engineering education. It is the authors' contention that a unified study of software quality, professionalism, and ethics is the right approach, and such a holistic approach is a crucial component in getting the best out of software engineering education.

These ideas were developed and refined in a compulsory 30-credit module for software engineering and computer science students. The module employs a number of novel techniques in delivery and assessment, as well as a number of online learning tools. These provide an exemplar environment of new educational experiences for those preparing for a career in software engineering. Also included in this chapter are summarised feedback ideas received from students and the experiences of the tutors delivering the module. This leads to a series of recommendations for future developments, which will be of interest to all involved in software engineering education.

DOI: 10.4018/978-1-60960-797-5.ch005

INTRODUCTION

Ubiquity of Software

How ubiquitous software has become! Not long ago, data processing departments were the exclusive users of software. They had large and expensive computer systems on which to run their applications, which were managed by teams of information technology professionals. Many people did not even know what a computer looked like.

The arrival of personal computing changed all this. First, the desktop computer brought information technology into the home; and then, the laptop computer made it an integral part of our everyday lives. In parallel with that, the availability of processor chips put computing power into everyday objects, from cars to mobile phones, turning analogue devices into digital ones.

Nowadays, many objects rely on software to deliver their services. For example, most music is recorded, disseminated, and listened to on digital equipment. In hospitals, specialised digital devices monitor patients and help doctors to diagnose ailments. Aeroplanes can now fly without pilots on board. The common denominator in all of these examples is software. National and regional governments are another example. They have various crucial responsibilities to their communities and need complex computing systems to meet those responsibilities. At the other end of the spectrum, small organisations, with no in-house software capabilities, use off-the-shelf commercial software applications to process the data that underlie their businesses.

Software Complexity and Developer Responsibilities

The complexity of the service provided is necessarily reflected in the complexity of the software itself. Word processing is, in principle, a relatively simple application; but, its complexity grows when graphics and *what you see is what you get* formatting capabilities are added. A system that connects crime fighting agencies or one that takes astronauts into space is far from simple. Moreover, any software that may have health or safety implications (such as that in a nuclear power plant), or one on which people rely for their well-being (like that managing social security payments), must be absolutely reliable, no matter how complex it is.

This places non-negotiable responsibilities on all software engineers, in two areas:

1. Software quality, to use the *best* tools and techniques to deliver the *bes*t possible products
2. Professionalism and ethics, to ensure awareness of all stakeholders in the deployment of software and to safeguard their rights.

Software Disasters

The logical chain is straightforward: as our reliance on software grows, so does the demand for *good* software, which, in turn, requires *good* software engineers, the responsibility for which falls on educational institutions and specialist training organisations.

However, over the last decade, there has been a catalogue of system failures throughout the business world that evidences the fact that software development is not always being approached in a holistic way. Many examples can be found readily on the Internet, in response to the search keywords *software disaster*. They are of varying degrees of severity. Four significant examples are summarised below:

1. The launch of Airbus A380 was delayed by a year or more in 2006 because of software incompatibility problems. The wiring in one part of the aircraft did not match the wiring in another because the two parts were built by different partners using different versions of their communications standards