Should software quality be a major issue when teaching first year programming to software engineers?

C.J. Burgess, a B.F. Jones b

aDepartment of Computer Science, University of Bristol, Bristol, BS8 1TR, UK
bDepartment of Computer Studies, University of Glamorgan, Pontypridd, CF37 1DL, UK

Abstract

Software quality is fundamental to the education of Software Engineers and therefore needs to be included as a major topic in every degree course involving Software Engineering, but it is far less clear how should it be incorporated during the various stages of the degree course. The foundation of the ideas underpinning software quality are many and varied since it is a complex idea and is difficult to measure. It demands a sound foundation based on a number of different issues such as specification, design, implementation, testing and documentation. The temptation is therefore to postpone discussion of quality until after the first year.

The authors argue that in the same way that quality cannot be "tested in" at the end of the software life cycle, so software quality cannot be "taught in" towards the end of a course. The motivation of generating high-quality software must be introduced from the beginning. This is essential to provide a good foundation for other quality topics, such as maintainability, portability and the importance of re-usability. The software crisis identified in the seventies is still alive and well, in spite of developments in the technology of design. The causes of the continuing crisis may rest in the failure to adopt professional approaches and attitudes as much as in the lack of use of technological innovation. It is this intangible ingredient for producing high quality software which it is difficult to teach and therefore must be inculcated right from the start of the course.

Introduction

Software Quality tends to mean different things to different people, some concentrating more on the process by which good software is produced, and others emphasising the quality of the actual product. Thus, before any discussion of the educational issues, it is very important that we define our terms. The authors will use the term Software Quality to refer to the
attributes of the product produced and Software Quality Management to the control of the process producing the software.

During the education of a Software Engineer, it is important that both Software Quality and Software Quality Management are taught as essential parts of the course. The authors maintain that Software Quality should be an integral part of most, if not all the courses that involve any aspect of programming, including the first-year course, and that some of the aspects of Software Quality Management should also be introduced during the first year, even although the topic is more likely to be dealt with in depth in later years, as part of a wider Project Management course. Teaching Software Quality has two important aspects. The first is giving the students sufficient technical expertise to produce a good product. The second is giving motivation and instilling a good professional attitude in the student, so that he or she appreciates the need for Software Quality and its implementation.

In this paper we will look at the ways in which many of the concepts of Software Quality can be introduced during a first-year course and thus provide a good basis upon which to build in later years. The first section concentrates on providing the technical expertise, followed by a section on encouraging a professional attitude, although the two aspects do overlap. Finally, it is shown how these together provide a good foundation for the later years of the degree course and the career that follows.

Technical Expertise

One of the problems of any degree course, and a first-year programming course in particular, is that the students are never involved in a large project, at least by commercial standards. In fact, unless involved in group projects, they are unlikely to be involved in programs longer than a few hundred lines. At this sort of size, many of the disciplines of software engineering may seem rather unnecessary, but it is essential that they are taught, and good habits are developed right from the start, as it is very hard to change them later. Thus it is important that the concepts of software engineering, including software quality, are taught and put into practice very early on in the course, even if this delays the start of the students coding any programs. These software engineering principles will be examined under various headings which are familiar from the normal software life-cycle.

Software Specification

In many large projects in the real world, too little time is spent on the specification and early design stages of a project, often due to the pressure
of the customer to get at least something working as soon as possible. This is a recipe for disaster for both large and small projects alike. Thus, starting even with the first programming exercise, the concept of a requirement's specification should be introduced, even though it may be extremely simple. This might be just the tutor's definition of the problem to be solved. The student should then be encouraged to elaborate this specification into something more precise, either by asking the tutor relevant questions or being encouraged to state assumptions that need to be made in order to solve the problem, e.g. the range of the data involved. Even though this may seem trivial for early problems, it is part of developing a good habit from the very early stage.

Irrespective of the actual specification method being used, whether informal or formal, it is very important to produce a good program specification. For some of the very simple problems, the requirement's specification may have specified or very strongly implied the algorithm to be used, but early in the course, the students should be encouraged to distinguish between the specification of the problem and the algorithm that will be used to solve it. There are a number of different approaches being used for exposing students to the more formal methods of specification. One of the most important aspects is that the students are exposed to a range of methodologies during their whole degree course. Assuming the students have some mathematical ability before they embark on the course, then it is good to encourage some more formal method of specification from the start. One possibility is to teach students to read formal specifications in a language such as VDM or Z and only have to write their own as part of a second-course. If this approach is adopted, than the relationship to the requirements specification must also be clearly taught. An alternative is to introduce algebraic specification of the data types as soon as data structures are introduced. This allows some degree of formalism without needing the same ability in discrete mathematics. All these provide a good basis for comparing the different available methodologies in later years, ranging from writing, and perhaps even refining, full formal specifications, to using structured methodologies such as JSD or SSADM.

Program Coding

Although the source of a program is only one aspect of many affecting software quality, the structure, presentation and commenting provided are often an accurate reflection of whether a professional attitude has been adopted during its development. It is essential that from the very early programming examples, students are taught the value of structured programming and modularisation. In the early programming examples, the use of the 'goto' statement should be avoided completely until the students
know enough to appreciate the few instances where the use of such statements might be legitimate, e.g. error exits from nested loops where no other suitable language construct is available. The sensible use of comments and meaningful identifier names should be encouraged from very early on. For example, encouraging students to include a comment box with every procedure or module which briefly states what that code is meant to do and the purpose of any import or export parameters.

The choice of an appropriate programming language for a first course is fundamental. The language should encourage all those aspects of good style that you are trying to teach, even if the language is not necessarily widely used. It is far more important to instill good habits and then learn a second or third language, rather than to let the market place influence the first course. It is also important that a good implementation of the language is provided with a user-friendly interface. In other words, the software being used to support the course should itself be of high quality. Otherwise students will soon be discouraged and frustrated. This is even more important if the course is shared by students whose main interest is not computing. The most suitable candidates amongst the well-known languages would seem to be Modula-2, ADA, or in the future ADA94. Pascal may also be suitable, if the language implemented has suitable extensions to support modularisation and data hiding, but this is second best since these would be non-standard. It is the authors’ opinion that the teaching of Basic, Fortran or C as a first language is very detrimental, and where this has already occurred at school, many bad programming practices have to be changed.

A more radical approach is to teach the software specialist a functional type of language as a first language, so that they are used to the more declarative type of style associated with formal specifications. In this case, the above discussion is more applicable to the second language they learn. The choice of programming language and its effect on quality is important in the real world as well as during their education [5].

Program testing

Unless taught otherwise students, and many practising programmers, only start to consider testing when they have coded a large part, if not the whole of their program. A far better practice is to properly plan the testing phase as an integral part of the project, even considering how the software will be tested during the requirements and specification stages. In addition, the student often considers a working program to be one that works on one or two data sets, or those provided by the tutor. It is essential that they be encouraged to view a working program as one that works correctly
on all valid data and rejects, with meaningful error messages, all incorrect
data. This can be encouraged, even in early programming examples in
several different ways. For example, students can be encouraged to include
software checks for valid ranges of data items, and produce error messages
for any violations. Students can devise test plans for their programs, based
on the source code, which ensure that every branch is obeyed at least once.
The use of dry running using very simple data and a pencil and paper can be
encouraged, even before the code is typed into the computer. In addition,
students should be encouraged to think in advance about what additional
output statements should be included in the code in order to verify that
the code is executing correctly. Like many aspects of software engineering,
these are items which might not be essential for small programs but become
increasingly important with size.

The concepts of unit, system and integration testing should be de-
scribed, even though they are difficult to appreciate on small problems.
Likewise the relative advantages and disadvantages of both structural and
functional testing, although the latter is harder to complement by examples.

Documentation

A first course should include a clear description of what is meant by
good documentation, including reference to both User’s Guides and Mainte-
nance Guides. It is also very helpful to students if they know exactly what
is expected of them in any given assignment. One way this can be done is
by issuing the marking schemes for any particular project in advance.

Professional Attitudes

A degree course with a large component of software engineering inevitably
represents a transition from writing fairly straightforward and short com-
puter programs to engineering larger and more complex software systems.
An important factor in making a successful transition is to enable the stu-
dents to cope with the increased complexity arising from the size of the
project, the need to collaborate with others, and from the demands of
thoroughness. Even superficially simple programs fail if an input value of
the wrong data type is entered, unless exception handlers, which are often
lengthy, are included. This point is addressed in introductory texts such as
Feldman and Koffman’s use of RobustInput in ADA [2]. Checking for in-
valid input is a relatively straightforward example of the trouble needed to
ensure the integrity of software. In general, exception handling is a difficult
topic, and although this is best dealt with later in the course, or in a second
course, it has a direct bearing on the quality of the software, and thus the
foundations for it should be laid in the first year.
The need for thoroughness may be "taught" at one level; however it goes beyond the normal demands that a student should understand what is being taught, and implies the need for an attitude which avoids cutting corners. This latter is probably best taught by the example of software engineers acting in a professional way but this is not easy in a first-year course. The need for it can probably be better appreciated if the students have to do at least a small group project in their first course, possibly without a lot of guidance so they learn by their mistakes. The pressure on the curriculum makes it unlikely that project management can be taught early on. Then in the second year, the students can embark on a major group project. This project has several aims: the students tackle a large project which integrates all the technical and managerial themes of the course, and any parallel courses, and it gives the student an important opportunity of working with others and trying to put project management, even on a small scale, into practice. This experience can be supplemented for students on Sandwich Degrees, by then spending the third year in supervised work experience. The group project also contains elements of self and peer appraisal. This appraisal is not intended to determine the grade for the group project module; experience shows that a significant proportion of students either over or under-estimate their abilities. Rather it is used to prompt the students to question their approach and to assess the effectiveness of their own and their colleagues’ contributions to the overall success or otherwise of the project. Before the project starts, the student cohort comes together to discuss the criteria by which they should judge their own performance and that of their colleagues, and to assign weightings to the relative importance of the different criteria. The tutor awards marks for the thoroughness of the analysis of the self and peer-appraisal, rather than for the success or failure of an individual’s contribution [3].

The sandwich year is of immense value to students. Working alongside a Chartered Engineer and taking responsibility for a part of a major industrial project is bound to enhance the student’s concept of Professionalism. In spite of the recession and the associated difficulties of placing students, great effort is expended to ensure that every student who wants to take advantage of the Sandwich year has the opportunity to do so. Sandwich students return to their Final Year with a much more mature attitude and with a much better idea of how to tackle the major individual project.

The major Professional Institutions covering the area of Software Engineering, the British Computer Society and the Institution of Electrical Engineers both visit Universities to accredit Degrees. They demand that Professional issues as well as technical issues are addressed in the courses. In its document "Course Accreditation" [7], the BCS expects that professionalism will be encouraged in legal and social aspects of computing as well
as purely technical methods of producing high-quality software [1, 6]. The need for the teaching of Software Quality Management in later years and for accreditation of the whole course has been covered in an earlier paper by the authors [4].

Conclusion

The teaching of quality must pervade the whole course both in terms of technical issues relating to software quality but also professional issues relating to software quality management. It is essential to teach quality from the first year, so that the right attitudes are developed for producing software of the highest quality, particularly for safety or mission critical software.

References


