Using quality principles to teach software quality techniques

C.S. Johnson, D.N. Wilson

School of Computing Sciences, University of Technology, Sydney, P.O. Box 123, NSW 2007, Australia

Abstract

This paper describes a methodology used to teach the concepts of software walkthroughs to students in an undergraduate computing science and information systems degree. The methodology resulted from the introduction of a practical component to enhance the theoretical base of the subject Software Quality Assurance. An additional and equal objective was for students to embrace one of the main concepts of quality, i.e. "right first time". The main teaching goal was to change the students’ previously held concepts of programming to one of "zero defects" using the techniques of Software Quality. The aim was for students to accept the concept of quality as a personal ideal, rather than an imposed standard. This notion of a personal ideal is the real challenge for any organisation adopting a quality approach regardless of application.

This paper also discusses the reward philosophy developed as a corollary to the methodology which provided students with a personal goal outside the formal marking scheme. This personal goal, discussed by students as part of the methodology, uncovers student motivations and acceptance of the value of a quality approach to software. This, in our opinion, is a key issue in using quality techniques within a teaching framework.
Finally, this paper will discuss extending the methodology to other disciplines, the advantages to the students of adopting a quality approach, the improvement in the learning process and the subsequent reduction to academic input in the teaching process.

1.0 Introduction

This paper describes the teaching of software engineering in the form of an undergraduate subject based on quality (Wilson & Johnson[1]). A simple but radical approach was adopted which used a quality approach to improving subject outcomes by demonstrating the effectiveness of quality techniques. By incorporating an integrated review process throughout the subject, continual improvement has been able to be applied. The results have been satisfying to the students and pleasing to the academics concerned.

Benefits of this methodology are raised which we feel apply across the software engineering teaching spectrum.

2.0 Background

The undergraduate subject Software Quality Assurance is a three hour elective in the Bachelor of Science (Computing) degree at UTS. It can be taken by either second year or fourth year students. The third year of the degree is spent in industry gaining practical experience. In general, there is an even mix between second and fourth year students and full-time and part-time students.

The authors have co-ordinated the subject since its inception in 1988. The first co-ordinator, Wilson, investigated and designed the subject before handing over to Johnson in 1992. The purpose in changing co-ordinators was to review the subject and its objectives, and to instil some new ideas and perspectives into the content and delivery.

The early versions of the subject had covered the material in a fairly thorough manner but the subject was theory based and fairly dry from the students' perspective. That is, assignments were in the form of essays and tutorials consisted of videos and/or discussions. The commitment to quality principles was only gained by a few students as a by-product of the material covered. A few part-time students were exposed to software quality in their work and these students tended to be more disposed to quality ideals than their full-time colleagues or other part-timers who been employed in Information System departments which had no quality programme.

The subject consisted of two essays worth 25% and 15% and a final exam worth 60%.
3.0 Requirements

It was felt that if we were teaching a subject on quality then students should on completion of the subject have or at least understand the commitment required to practice quality. The subject was reviewed in terms of students' comments gained in informal discussions, observation of the behaviour of students during the subject and in later subjects where one would expect some of these techniques or attitudinal changes to be present.

Teaching, at least in the authors' opinion, can always be improved with a practical content, if the practical work is cognisant with the subject and does not overload the student for the particular subject or in total for all concurrent subjects. Thus the decision was made to introduce a practical content to the subject which would not only demonstrate some of the quality practices but demonstrate their value to the students. The chief objective was to increase the number of students who understood the level of commitment a quality approach requires, and hopefully provide them with that commitment.

4.0 Changes Implemented

The changes were to introduce a practical content with a programming assignment and to introduce an essay to reflect on the effectiveness of the programming exercise. Additionally, two techniques lectures were moved to the early weeks of the subject.

The first of these lectures covered Walkthroughs and Inspections (Fagan[2][3], Yorurdon[4]) which included a "live" walkthrough in the lecture. A "live" walkthrough comprised a role play by three academics "walking through" a module of code. The second lecture covered Problem Exploration techniques such as Brainstorming, Nominal Group Technique, Cause and Effect Analysis (Fishbone Diagram), Affinity Diagram, etc. The students practiced the techniques introduced in these lectures during the course of the lecture (i.e. lecture and tutorial sections are interspersed). It was expected that students would then use the techniques in the programming assignment effectively.

The programming assignment was the principal change introduced. This programming assignment was to be an individual effort using Walkthroughs as the key Software Quality Technique. This involved the students working in a group. The problem of students undertaking an individual programming assignment in groups is that participation in a group generally either encourages cheating or the answers tend to be homogeneous. Neither of these outcomes was wanted. The solution was to give an assignment with four equal but different problems to be solved. The students formed into groups of four
and then decided between themselves who would do each part. Thus walkthroughs would allow students to check each other's work, but would not result in either cheating or a homogenous solution.

Students were required to perform two walkthroughs: one at the design stage and one at the code stage. These had to be documented using appropriate forms. An example of a walkthrough form was given out in a lecture but students could use any form they desired.

While the assignment now gave students practical experience of a software quality technique, i.e. walkthroughs, we could not see the students gaining any understanding from the exercise. Students would merely complain about the problem of scheduling meetings, the overhead of the paperwork and would therefore probably only pay "lip service" to the walkthroughs.

It was at this point we realised that the solution to demonstrating the benefits of adopting a quality approach was to set the students the task of producing "zero defect" software. In other words the code would be "right first time". The students were told that they would be allowed only one compilation and one execution with all test data. To ensure this, the program was to be entered by another member of the group, compiled once and run with the test data. This person was to sign a piece of paper attesting to the results of the run. The tester would stop on any failure, except typing errors made by the tester.

4.0 Assessment

The problem of marking the assignments was that assigning marks for "zero defects" would provide too great a temptation for the students to resist "cheating". This was the difficult part, to give a fair assessment for this assignment. Knowing students' "mark oriented behaviour" we realised that giving marks for achieving "zero defects" would make "cheating" (i.e. compiling and testing the program secretly) a very tempting option.

Thus, we decided that assessment would not be on achieving "zero defects" and this was made clear in the assignment handout. In other words, achieving zero defects was to be a reward in itself. This idea seemed to fit in with the quality philosophy.

Students were to be assessed on their adherence to software quality techniques. To this extent quite explicit programming standards were given. Also metrics of time spent were to be kept and students were required to buy a small book to keep these records. The books were to be signed off by either the lecturer or tutors. No compulsion was made for the signing off, it was up to the students to approach the lecturer or tutor and get them to initial the metric records in
the book. All records of the students' walkthroughs were to be included in the final assignment. This included the work that was walked through and the signed forms, both for the students' own work and all walkthroughs of other students in which they participated.

A key factor in this assessment was not telling students exactly what would be assessed. We were attempting to change normal behaviour not create adaptive behaviour. For instance, many students and groups did not adhere to the programming standards even though they were explicitly stated. If students had been told that marks would be lost for not adhering to standards then they would have adhered to the standards for the marks. That is, standards had a purpose and were to be adhered to, not ignored or given "lip-service".

The standards, of course, made marking the assignment extremely easy as they corresponded to the lecturer's standards and thus the code could be understood quickly. Quality at work!

The aim of the assessment was to correlate in the students' mind Software Quality procedures and success both in marks obtained and in their personal objective of achieving zero defects.

Another side affect of marking according to documentation not results was that it quite obvious whether students had followed the processes or had simulated them. The walkthrough forms required signatures and often follow up work; the metrics books were filled in over a few weeks, thus one expected to see a variety of pens used. Students who "faked" the processes tended to use only one pen and the faked work had a homogeneous feel to it. Also, they could not fake our signatures in their metric books.

As the final task in the assignment students were asked to write approximately one page reflecting on the experience. This was quite revealing with comments like "This was the first time I ever ever ever ever ever ever ever got a program to compile first time".

The students who achieve zero defects has ranged from 15% to 40% over the time the subject has run in this form. Most students get very close, although some obviously "cheat". We catch many of these but some slip through; they of course receive little benefit.
5.0 The Essay Assignment

The essay assignment question was "Discuss the process of obtaining commitment to SQA; improving the process you experienced, problems with this methodology and its effectiveness in gaining commitment.", in which students had to interview other students in the subject about the assignment as well as expressing their own views. This assignment was given as late as possible to maximise the gap between the first assignment and this one, thus allowing students to be both objective and reflective.

Most students answer this essay honestly, even to the extent of commenting on members of their group who "cheated" and noting that they receive very little benefit. All students commented on the tediousness of the metrics, with a reasonable proportion making the observation that time spent on design reduces problems at the coding stage. This was the main objective of collecting metrics. They are taught this in early subjects but never really put it into practice. Marks are awarded for reflection on the method, the effort they have put into discussing the assignment with other students and suggestions for improvement, a number of which we have subsequently incorporated.

6.0 Results

We assessed the results in two ways, one using the feedback from teaching surveys, the other from an exam question:

You have been engaged to participate in a debate. The topic for the debate is: "The more important concept in SQA is 'zero defects' rather than 'meets requirements'?"

Unfortunately you have lost the note telling you which side of the debate you will be on! Prepare your arguments for both sides of the debate, then present your own personal opinion.

This question should be answered in three parts, one for each side of the debate, then your own opinion.

The answers to the above question were generally well argued with students able to give a very real perspective in their answer, especially in separating their arguments for both sides.

In the teaching surveys conducted during the second semester in 1992 student 15 of the 27 students made the comment that the SQA material and approach should be taught in earlier and core subjects e.g. "Aspects of a subject like this should be included in the 1st [sic] couple of weeks of all programming
subjects. (If not all subjects) If only I had begun programming in this way!!!"; "I think the subject should have much more importance than just an elective status..."; "The subject itself should be a core subject rather than an elective subject".

What was surprising about these comments is that most of them were in response to a question in the teaching feedback survey "Please feel free to make any other comments,..." which occurs at the end of the teaching survey and is very rarely answered.

7.0 Recommendations

Our recommendation from this research covers three areas; the students, quality issues and teaching resources as we think there are extensive benefits to be gained in each area.

The main recommendation from this experience is that the teaching of all software engineering subjects should adopt this approach, i.e. giving four (or more) equal but different assignments questions for every one issued previously and requiring students to perform walkthroughs. It reflects the teamwork that occurs in industry, it allows students to help each other (without worrying about cheating issues for either side), to learn from each other, to communicate, to document and to achieve a more satisfactory learning experience. We also feel that this approach could be applied in many aspects of software engineering apart from programming, e.g. system design, interface design, even essay questions.

The effect of this recommendation is to produce a better standard of work in that most students will complete the whole task with programs that are more understandable, more robust, better tested and definitely better documented. In this respect we think that this is exactly the standard we should be setting in universities.

Whether the task is performed in a shorter time is hard to judge at this stage, given lack of metrics on previous methods and the learning curve. One point that should be raised is that the metrics certainly allow course co-ordinators to know the time spent on assignments. This could perhaps be compared with overall marks and learning being more individually tailored. Our judgement would be that time spent on average is about equal to normal methods perhaps reducing with experience. The only administrative problem in adopting this method generally is providing both meeting times and meeting spaces for students.
The use of teaching resources especially terminal and CPU time certainly is reduced with its obvious ramifications of cost savings in the provision of equipment etc. As well, time spent on marking is reduced because of standards which are strongly enforced by the students themselves. The greatest resource saving, is that the amount of time spent in effective student learning increases dramatically with no more effect from academics as the students are learning from each other. This point was raised by most of the students in their comments on the method.

On a final note the authors think that this method really is a good example of using a quality philosophy to improve the process. The fact that the process is the teaching of software quality is a bonus.

References