

Testing for tomorrow's professionals

V. Petruv, M. Ross, G. Staples

*Southampton Institute, East Park Terrace,
Southampton, SO14 0YN, UK*

ABSTRACT

This paper discussed ways in which students can be taught to test software effectively. It is based on the authors' experience of teaching software engineering on range of MSc, BSc, HND and professional courses, both full-time and part-time. The approaches used by the authors are described and are used to raise discussion points on possible alternative approaches.

Software Quality is used as a theme in the design of courses, with the emphasis on building quality into the software throughout the development cycle. This raises issues of how best to give students an understanding of the place of testing in this context. The languages that the students will meet first are third generation languages, and their early experience of testing is done with relatively small programs. Much of the literature on testing, and most of the available software tools also relate to third generation languages. Effective testing of large programs in these languages must also be considered. The transfer of 3GL skills to the testing of 4GL programs is also considered and the testing of programs relating to different programming paradigms, such as object-oriented programming and logic programming is discussed.

1 Introduction

The contents of this paper stem from discussions between the authors, and others, on the teaching of software testing. The analysis, design and coding parts of the life cycle, and related topics such as the choice of methodologies and programming languages, had been the subject of many recent discussions but we felt that our teaching of the important topic of software testing had not been critically examined for some years. The discussion is based on the authors' teaching experience on a group of related Degree courses in

218 Software Engineering in Higher Education

Information Systems, Computer Studies and Business Information Technology and HND courses in Computing. All of these courses have a Business Computing orientation. Our initial impression was that the topic had been covered more thoroughly in the past, so we started by considering our past practice, then looked at the changes that had taken place and how these had affected our teaching of testing and finally we evaluated our current teaching of the topic. The rest of this paper is structured in the same way. Of course, testing cannot be taught, or practised, in a vacuum and is affected by the programming language used, the life cycle approach, the type of software being developed and by economic and other factors and so these issues are also considered in this paper.

2 The Past

In the first year Systems Analysis and Design Unit, a simplified Waterfall life cycle model was used, in which testing was something that was done after coding. The first year programming units were third-generation language orientated, typically using Pascal, although some courses included Cobol as well. Some practical work in assembly code would also be done in the Computer Architecture Unit. No previous programming knowledge was required of entrants, so programs were generally small. Even where the programs contained a number of procedures, the testing approach was 'big bang' Test data design was done informally; students were required to design test data sets for their programs, after testing several programs for which the lecturer had supplied test data sets without discussing the design of these test data sets. The students were also asked to explain how they chose their test data and their answers indicated that most students had an intuitive understanding of the need for functional testing. Students were also asked to test each others programs; many used the same functional approach, others invented what may be called 'dysfunctional' testing, inputting irrational values to try to break the system.

In the second year Systems Analysis and Design Unit, the V life cycle model was introduced. The testing phase was split into unit testing, integration testing and acceptance testing and the related test planning linked across the arms of the V to coding, design and analysis respectively. In the second year, programming was still using the procedural paradigm, introducing data structures and pointer types in Pascal. The programs were larger and structured as a number of procedures.

Integrative assignments, involving a number of units, were then in fashion and it was possible to set assignments that involved analysis, design and coding, together with the relevant test planning and testing. Lectures included a full discussion of testing, based on the texts of Myers [1], Beizer [2], and Hetzel [3]. These included both Black Box (functional) and White Box (Structural) approaches to test-data design, and top-down, bottom up and other

approaches to integration testing. There was workshop time available for students to design test data sets using black box methods and white box methods with different levels of coverage, and software tools were available to give information on the coverage achieved. There was also time for groups of students to try out the various approaches to integration testing, and to discuss the advantages and disadvantages of these. A symbolic debugger was used to facilitate these experiments without the need to spend time developing test drivers or stubs.

3 The Changes

There have been many changes in recent years that have affected the teaching of testing. These can be crudely summarised as "more in the syllabus, with less time to teach it". In line with our Institute's mission statement, our Computing courses are vocationally orientated and the content is influenced by local employers and students, as well as by the course team. Although third generation languages are still used in the first year, a number of other paradigms are met in later years. These include the declarative programming paradigm, using Oracle, and object orientated programming using C++ or Smalltalk. The artificial intelligence units also include logic programming in prolog and the use of expert system shells. The introduction of first year units on the use of pc-based packages such as spreadsheets and databases is also relevant, as the need to test applications developed using these must be covered.

The Systems Analysis and Design units have also changed, to include structured methodologies such as SSADM. More complex life cycle models, that consider prototyping approaches and evolutionary development, have also been introduced. Object-orientated Analysis and Design methods complement the object orientated programming units. Soft systems methods are also considered, although these have little direct effect on the way testing is taught. The application of formal methods to the design of software is discussed in these courses and the possibility of developing provably correct software has major implications on testing.

More generally, the design of the courses listed above has used software quality as underlying theme. This design approach is discussed more fully in Ross and Staples, 1992 [4], and Monger et al, 1994 [5]. It is the view of the course teams that the software engineering approach to systems development requires the use of development methods at each stage that, as far as possible, avoid the introduction of errors, and the quality assurance of the deliverables from each stage to remove errors as early as possible in the life cycle, rather than relying on testing to remove errors after the software has been developed. The courses aim to cover good practice in all stages of the life cycle, and in the supporting activities such as project management. Students experience the main quality assurance techniques, such as inspections

220 Software Engineering in Higher Education

and walk-throughs, at all stages of the life cycle. Methodologies such as Prince and SSADM that emphasize these QA requirements are covered in detail and are related to quality standards such as ISO 9000. By integrating this quality approach into all units, the students get a deeper understanding than would be given by a single 'quality' unit in the final year. Recently, more attention has been given on these courses to the discussion of social, economic and ethical issues than in the past and software quality issues contribute to many of these discussions.

There have been major changes in course structures, with unitization and semesterization both working against integration, and in course organisation, with major reductions in contact hours, larger classes and an open-access policy that has increased the variation of knowledge and skills of the first year students.

As the courses are vocationally oriented, the major changes that have taken place in the use of computers by business are also relevant. In the past, the first employment of most of the students was in programming. Currently, a wide range of jobs is recorded, mainly in support areas, with programming as a minority employment.

4 The effect of these changes on the teaching of testing

The major approaches to unit testing and integration testing discussed above are still covered in connection with third generation languages. The critical evaluation of these traditional testing approaches in the context of new programming paradigms develops useful skills. A brief description of some of our Oracle assignments will be used to illustrate this. A fuller discussion is given in Ross and Staples, 1991 [6]. Two related assignments are based on the same case study material. In the first, the students develop a prototype using SQLPLUS and SQL FORMS and work with lecturers role-playing as users to improve the prototype. In the second, the student builds a final version of the system using information gained during the prototyping sessions; and writes a user guide for the system. These exercises can be individual or group-based and the changes requested during the prototyping sessions can be varied from simple changes to the user interface to major changes in the underlying data model. Most students successfully apply functional testing, and structural testing in SQL statements with WHERE clauses, without much prompting. However, a few students raise more subtle points for discussion; for example, how to test changes to programs in a production database system. The need for testing of the prototype before showing it to users is often raised. Students generally agree that the prototype should be tested to ensure that it functions as specified, with a minority who feel that this is a waste of time, as the users are going to change everything. The inclusion of a user guide for the system raises the issues that software is more than just a program, and that user guides also need testing.

The discussion of testing expert system raises different issues, relating both to the inference engine/user interface and to the rule set. Ethical issues such as possible errors in medical expert systems are easily raised in this context.

The testing of object oriented systems raises further issues, particularly when reusable objects are considered. These issues are discussed in Barbey and Strohmeier, 1994 [7] and Reed et al, 1994 [8].

5 Conclusions

It is no surprise that fewer hours are now spent on the topic of the testing of 3GL programs in view of the major changes discussed in the previous section, particularly the change in total class contact hours. However, a smaller proportion of the contact hours are spent on the topic than in the past, and this does need examination. We consider that our coverage is about right, for the following reasons:-

- i) Time is spent considering testing in the context of other systems development programming paradigms.
- ii) Few of our students will have a career as programmers in third generation languages or indeed as programmers in any language.
- iii) It is better to develop the students' skills in avoiding the introduction of errors, and in removing these that do creep in as early as possible in the life cycle, than to develop testing skills.

We feel that these arguments would apply to any Business Computing course that used a Software Engineering approach. Different conclusions may have been reached if the courses had a more technical orientation. Indeed, the Division offers an MSc in Computing (Software Engineering), which concentrates on procedural and object-oriented languages, such as C and C++. This course has a half-unit on validation, verification and testing. As most of the students get jobs as software engineers using these languages, the authors consider that the emphasis on testing is entirely appropriate. Chen and Mak [9] also give a different view on the teaching of testing and debugging in Computer Science courses, which supports the view that various approaches and emphasis are needed on this important topic for different courses.

References

1. Myers, G.J., *The Art of Software Testing*, Wiley-Interscience, New York, 1979.
2. Beizer, B., *Software Testing Techniques*, Van Nostrand Reinhold, New York, 1983.
3. Hetzel, W., *The Complete Guide to Software Testing*, Collins, London, 1985.
4. Ross, M. & Staples, G., Quality as a Theme in Computing Courses, in SEHE 92 (ed G.King) pp 183 to 190, in *Proceedings of second national conference on Software Engineering in Higher Education*, Southampton, SIHE, Southampton, 1992.
5. Monger, A., Petruv, V., Ross, M., Staples, G. & Tromans, I., Education in Quality for Computing, in ISD94 (ed J.Eupanac & S.Wrycza) pp 511 to 520, in *Proceedings of fourth international conference on Information Systems Development*, Slovenia, Moderna Organizacija, Kranj, 1994.
6. Ross, M. & Staples, G., Developing Transferable Skills via Prototyping using 4 GLs, in SEHE91 (ed G.King) pp 236 to pp 243, *Proceedings of first international conference on Software Engineering in Higher Education*, Southampton, SIHE, Southampton, 1991.
7. Reed, H.G., Turner, C.D., Aibel, J.B. & Dalton, J.T., Practical object-oriented state-based unit testing, in SQM94 (ed Brebbia, C.A., Ross, M., Staples, G. & Stapleton, J.) pp 2-363 to 2-378, *Proceedings of second international conference on Software Quality Management*, Edinburgh, Computational Mechanics Publications, Southampton, 1994.
8. Barbey, S. & Strohmeier, A., of The problematics of testing object-oriented software, in SQM94 (ed Brebbia, C.A., Ross, M., Staples, G. & Stapleton, J.) pp2 - 411 to 2-426, *Proceedings of the second international conference on Software Quality Management*, Edinburgh, Computational Mechanics Publication, Southampton, 1994.
9. Chen, T.Y. & Mak, I.K., Experience of teaching debugging and testing, in SQM94 (ed Brebbia, C.A., Ross, M., Staples, G. & Stapleton, J.) pp2-349 to 2-362, *Proceedings of the second international conference on Software Quality Management*, Edinburgh, Computational Mechanics Publications, Southampton, 1994.