ABSTRACT

In recent years, software testing is becoming more popular and important in the software development industry. At the same time, more software testing courses are developed to fulfill the training needs for software testing professionals in the workforce as well as computer science/software engineering students studying in universities. However, it is generally agreed that software testing is a difficult subject to be taught within the duration of a course. Through real life teaching experience in a university software testing course, we found that students can learn the details of software testing principles and attain the testing techniques more effectively by including a practical project component in the software testing course curriculum, requiring them to plan, design, implement and test an automated software testing system.

KEY WORDS

Education, software engineering, software testing, software reliability

INTRODUCTION

This paper discusses the approach and describes some of our experiences in teaching the Software Testing and Reliability course (HIT3057/8057, hereafter referred as STR) to the undergraduate and master students studying in the School of Information Technology, Swinburne University of Technology in the second semester of 2001 [7].

Programs offered by the School of Information Technology, Swinburne University of Technology often address the needs of the contemporary IT industry, and hence program structures and course contents are continually under review in order to keep them in touch with the state-of-the-art technology. In the last few years, software testing has emerged as an important field in the software development industry. Recently, there are publications urging for more software testing to be taught to students at both the graduate and undergraduate levels [1,3,5]. To equip our students with sufficient software testing knowledge and skills so that they become highly employable in this growing area, a new course on software testing and reliability was developed and introduced into our undergraduate and postgraduate curricula since the second semester of 2001.

Apart from offering to the third and fourth year undergraduates in the computer science, software engineering and information technology disciplines, the STR course is also offered as an elective to postgraduate students undertaking the Master of Information Technology program in Swinburne. In 2001, there is a significant portion of master students from a wide range of overseas countries such as Norway, India and Malaysia.

STR serves as an advanced software engineering course for the students. Before starting the STR course, most undergraduate students have finished two courses in software engineering (at the introductory and intermediate levels), while the master students have completed at least one software engineering course or equivalent as prerequisite. Although software testing is often covered in their previous software engineering courses, none of the students has ever taken any previous formal software testing or reliability course as a single complete unit. Due to the program testing nature of the course, all STR students are also required to have completed at least one programming language course (in Java or C++). Some undergraduate students return to study from the industry...
after finishing one year of Industry-Based Learning (IBL). Hence, the group is of mixed background with various levels of experience in software development. In 2001, there are around 50 undergraduates and 30 master students taking the STR course.

The STR course covers the following topics in a series of 12 lectures of two hours each followed by one hour of tutorial running weekly within one semester:

- Selection of test cases
- Program instrumentation
- Data flow analysis
- Domain testing strategy
- Reliability theory

During the course, guest speakers are invited from the software testing industry to introduce some real life aspects of software testing such as automated system and test management to the students.

Before the STR course was introduced in 2001, students in Swinburne were taking the Advanced Software Engineering (ASE) course instead. The ASE course covered the areas of maintenance, metrics, reliability, software quality, testing, validation and verification with about 50% weighting on testing [6]. In general, the students found the course very useful for their future works. During a subject review session conducted in 2000, we found that there was a general consensus from students that more hand-on experiences on software testing will improve their understandings on the testing methodologies covered in the lectures. Bearing this valuable comment in mind, the STR course has been designed to include a major practical assignment of planning, designing, implementing and testing an automated system for software testing. In view of the lack of software testing tools available, if students devote sufficient efforts in producing and testing their own software testing system, they can apply their knowledge to a more real life environment and learn the practical experiences of being software testing practitioners. In essence, we hope that after completing the course, students are provided with sufficient training to develop their own automated software testing tools for their needs.

DEVELOPMENT OF PROGRAM INSTRUMENTATION SYSTEM

In 2001, the STR students are required to complete an assignment in groups of five to six between week 5 and week 11 of the course as part of the course assessment. Aiming at providing more hand-on experience to the students, the assignment requires them to design and implement a program instrumentation system that supports dynamic data flow analysis. The system should be able to accept a program source code in either Java or C++ as an input file and insert software probes into it for the detection of data flow anomalies. The instrumented program is then compiled and executed instead of the original program. In addition to the normal output of the original program, a report of the detected anomalies would also be produced by the system (Figure 1).

![Diagram of Program Instrumentation System](Figure 1)
Depending on the choice of programming language, students can choose between the following four different ways to tackle the assignment:

1. Language X = Java, Language Y = Java
2. Language X = C++, Language Y = C++
3. Language X = Java, Language Y = C++
4. Language X = C++, Language Y = Java

Naturally, it is anticipated that most groups will choose option 1 or option 2 listed above according to their preferred programming language.

Due to the time constraint and the workload involved, students were asked to implement a limited system that
includes programs with the following common features as a minimum requirement of the assignment:

- Data type: integer, float
- Data structure: one-dimensional array
- Input/Output: cin and cout for C++, keyboard class and system.out.println for Java
- Assignment statement: arithmetic assignment statements
  - Arithmetic operator: +, -, *, /
  - Equality operator: ==, !=
  - Relational operator: >, >=, <, <=
  - Logical operator: !, &&, ||
- Control statement: IF-ELSE, WHILE, FOR

In addition, data flow analysis across module boundaries (function call) is listed as an extension work for the more capable students due to its higher demand of programming efforts. Students are encouraged to include extra functionalities depending on their design and progress. Bonus marks (up to 10% of the maximum score) will be allocated to groups with outstanding achievement.

Before the project description was issued to the class in the beginning of week 5, all students have just completed six lecture hours of program instrumentation and data flow analysis that mainly covered the materials from the papers by J.C. Huang [4] as well as Chan and Chen [2]. All underlying theories and principles required for completing the assignment have been covered and explained to the students before they began to explore the project.

In addition to the six hours of lecture, the assignment is also supported by a series of tutorials running between weeks 5 and 10. During these tutorial sessions, students have been briefed in small groups on the planning, requirements, design, testing, documentation of the system, and some implementation issues of the state transition function and the state anomaly reporter.

The academic staff of the course play the role of clients for the project groups, and the students are required to elicitate user/system requirements through interviews and emails. They then proceed to the design and implementation phases after documenting the System Requirement Specification (SRS).

To develop the required system, students are free to adopt any development process model that they learnt in previous software engineering courses. Even so, most groups realized that the requirements for this system will remain fairly stable throughout the development cycle and hence basic waterfall model became the best choice for this project. Being a software testing exercise, it was strongly emphasized that students should apply all appropriate testing methodologies as they see fit to verify and validate their instrumentation system. As expected, none of the groups includes the maintenance phase as part of the development process.

To allow students to verify the basic functionalities of their developed system, sample test cases have been released to them before assignment submission. However, it was also emphasized that students should design and select their own test cases according to their test objectives. In addition, all groups were advised to create extra test cases for more thorough testing of the system and any extended features that were implemented. Test plan and test reports therefore form an essential component of the assignment, and hence ensuring the exercise matches with the objectives of this course.

Assessment criteria of the assignment are based on the areas of team management, requirements and specifications, testing, system correctness, efficiency, coding quality and documentation. After submission, every individual group is required to demonstrate their final delivered system in the last week (i.e. week 12) of the semester as part of their assignment requirement. During the demonstration, new test cases are employed to check the correctness and completeness of the instrumentation system developed by the groups. Apart from fulfilling the assessment purpose, the demonstration also serves as an informal user acceptance testing of the system in the client's environment.

**OBSERVATION**

Since Java is the dominant programming language in Swinburne courses, twelve out of fifteen groups implement the instrumentation system in Java to test Java source codes. Two groups implement the system in C++ to instrument C++ source codes. One group takes up the challenge of using Java to implement the instrumentation system that tests C++ source codes. In terms of the system qualities of completeness and correctness, the undergraduate students generally performed better than the master students. It was surprising to find some master students found the programming part of the system development is too hard for them and two groups even have difficulties to implement a fully working system.
Despite the fact that many groups found it difficult to start the project in the very beginning, most groups managed to deliver a functional system that successfully detects data flow anomalies in programs written within the specified requirements on or before the assignment submission date. It is very encouraging to see that about half of the groups could build their systems to properly handle extra features such as comments in the source codes, extra arithmetic operators like ++ and --, variable of data types other than int and float (e.g. char). Though not specifically required for the assignment, five groups could manage to report also the position of the relevant action for the anomalies (by referring to the line number of the input source code). Out of all groups, only one has successfully implemented the system to handle simple function calls and hence capable of detecting anomaly across module boundaries.

It was interesting to find that a few groups have developed a GUI environment for the instrumentation system to improve its usability. To make the system even more flexible, some groups have also developed a few extra utilities such as generating anomaly report in various details according to the user’s choice, and exporting the final anomaly results to Excel spreadsheets for further analysis.

When the assignment was initially designed, it was intended that the students can use the instrumentation system they developed to test the system itself. However, this turned out to be far too difficult than expected for most students to implement such a system that can provide all features to perform dynamic data flow analysis for the source code of the instrumentation system. After careful consideration of the realistic time constraint, we decide to leave it as an open option for individual project groups.

A few observations from the students’ work are worth mentioning:

• Students admit that they learn most of the program instrumentation techniques through the implementation of the instrumentation system in the assignment, as many of these details cannot be covered in the lectures due to its short time frame and the complexity involved.
• Students learn by exploring in the assignment how to determine the appropriate position where the software probes should be inserted into the source codes to be tested.
• Through actual implementation of the instrumentation system, students understand that for some control structures (e.g. while loop), more than one software probe are usually required to properly keep track of the state transition of a variable.
• Students can fully appreciate the complexity and creativity of testing a software system by developing and testing an automated testing system.

• Though quite demanding, most groups enjoyed the work and found the project very fruitful after all the hard work.

CONCLUSION

The practical assignment in the STR course is regarded as a success as most students felt that they can understand the concepts of program instrumentation and data flow analysis much better after completing the project. They also agreed that they could now relate the software testing principles more closely to practical testing processes after involving directly in the planning, design, implementation and testing of an automated system for software testing.

ACKNOWLEDGEMENT

We take this opportunity to express our acknowledgement to all Swinburne students taking the HIT3057/8057 Software Testing and Reliability course in Semester 2, 2001.

REFERENCES


