Teaching “Data Flow Testing” in an Software Engineering Course

Chang Liu

School of Electrical Engineering and Computer Science
Russ College of Engineering and Technology
Ohio University
Athens, Ohio 45701, USA

http://ace.cs.ohiou.edu/~changliu
liuc@ohio.edu

Abstract

One common approach to structural testing of software programs is to design and select test cases according to control flows of software programs. Data flow testing not only does that but also pays attention to how a variable is defined and used at different places along control flows, which could lead to more efficient and targeted test suites. Therefore, it is important to cover the concept of data flow testing in undergraduate computer science programs.

We teach data flow testing in an undergraduate software engineering course. The challenge we face is to effectively teach this concept in a short period of time, typically thirty to fifty minutes. In this paper, we present how we currently teach this topic. We conclude this paper with our current results and a discussion of what can be done better.

1. Introduction

One common approach to structural testing of software programs is to design and select test cases according to control flows of software programs. Common control-flow-based test coverage criteria include the statement coverage criterion, the branch coverage criterion, and the path coverage criterion [1]. While it is not difficult for undergraduate students to understand these concepts, they often doubt the practicality of these coverage criteria because coverage criteria such as the path coverage criterion often (1) cannot be fully satisfied because of infeasible paths, and (2) require a large amount of test cases to achieve the highest possible coverage.

Data flow testing is a testing technique based on the observation that values associated with variables can effect program execution [2]. Data flow testing not only explores program control flows but also pays attention to how a variable is defined and used at different places along control flows, which could lead to more efficient and targeted test suites than pure control-flow-based test suites. An important insight that data flow testing can provide to students is that it shows a way to distinguish between the useful ones and the less useful ones among test cases generated from pure control-flow-based testing techniques and trim the number of required test cases without reducing the effectiveness of the test suite. Data flow testing can help remove students' doubts about the usefulness of control-flow-based test coverage criteria and stimulate their imagination of what else can be used to trim off less useful test cases. Therefore, it is important to cover the concepts of data flow testing in undergraduate computer science programs.

We attempted to cover data flow testing in a software engineering course. Section 2 describes the background information of this course. Section 3 presents our current approach to the instruction of this topic. Section 4 concludes the paper with a discussion of what can be done better.

2. Background

At Ohio University, our computer science program does not currently offer a technical course solely on software testing. We cover the topic of software testing in CS456/556 "Software Design and Development," a dual-listed software engineering course offered to both computer science and computer engineering senior undergraduate students and first-year graduate students. Two textbooks are required in CS456/556. The first one covers main technical content of the course, which is the Unified Software Development Process [3]. The second one covers communication topics, which are also important to software engineers [4].

A major component of this course is a term-long team project, which student teams are required to develop using the Unified Software Development Process they learn from the class. To provide a real world setting for students to apply their technical and communication skills, and to foster their senses of civic responsibility, we adopt service-learning projects in this course [10, 11]. Service learning projects are real-world projects that serve the needs of surrounding communities and that require the application of course contents [7]. We invite and select representatives from partner communities to serve as clients in the projects [12]. The clients typically come to the classroom in Week Two to present project requirements. They also participate the final project presentations and demonstrations to help judge the quality of student projects. In the past, Nelsonville Public Library, the Office of Code Enforcement and Community Development of the City of Athens, and other community partners had served as clients in CS456/556.

CS456/556 covers all five major phases of the Unified Software Development Process, include requirement, analysis, design, implementation, and test. Some phases, such as analysis, are covered in one week. Student teams are required to complete corresponding phases of their team projects also in one week. Other phases, such as implementation, takes several weeks. In this case, student teams are required to complete and submit corresponding phases of their projects in several stages, one stage every week. At the end of the course, student teams release the projects as open-source software packages [13] and turn them to the community partners.

The technical textbook [3] does not thoroughly cover testing techniques. When discussing unit test, we point students to supplementary materials such as an on-line document on code coverage at http://www.bullseye.com/coverage.html.

CS456/556 lasts for one quarter, including ten weeks of instruction and one exam week. The class typically meets twice a week. Each session last 110 minutes including a 10-minute break. About nine percent of the class time, or about 180 minutes (twelve fifteen-minute mini-sessions) out of a total of two thousand minutes class time (twenty one hundred-minute sessions), are devoted to communication topics, including foundations of communication, interpersonal communication, group communication, and public speaking [5, 6].

In addition, CS456/556 covers the topic of professional ethics using a case study about the Reagan-era "Star Wars" program [9]. Table 1 lists the main contents and the lesson plans of a typical offering of CS456/556.

Table 1. Typical course contents of CS456/556 sessions. (Note that from Week Three to Week Nine, most sessions also include a fifteen-minute mini-session on communication topics.)

Week	Session One (1 Hour 50 Minutes)	Session Two (1 Hour 50 Minutes)
1	Introduction / SourceForge	CVS – a source code version control system
2	Presentations by project clients	The Unified Process
3	Requirement	Requirement
4	Analysis	Analysis
5	Design	Design
6	Design / Professional Ethics	Professional Ethics ("Star Wars")
7	Implementation	Implementation
8	Unit test	Unit test
9	Midterm Exam	Test
10	Midterm Review / Test	More on the Unified Modeling Language
11	Exam (final presentation and project demonstration)	

In CS456/556, in addition to traditional style lectures that cover various topics, a variety of in-class activities are adopted to improve student learning on both technical and non-technical topics [6, 8].

3. Teaching "Data Flow Testing" in CS456/556

Because data flow testing is an important and effective testing technique, and because there are no other courses on software testing in our program, we decide to cover data flow testing in CS456/556. Our goal is that students fully understand the following concepts, know when they are applicable, and be able to apply them manually in unit tests.

- Definition occurrence
- Use occurrence
  - Computational use occurrence
  - Predicate use occurrence

- Def-use pair
- All definitions criterion
- All uses criterion
- All definition-use-paths criterion

These concepts can be explained by the following two paragraphs [1, 2].

There are two types of occurrences of variables in a program, namely definition occurrences and use occurrences. A definition occurrence of a variable is where a value is assigned to a variable. A use occurrence of a variable is where the value of a variable is used in the program. There are two types of use occurrences, namely computational use occurrences and predicate use occurrences. A computational use occurrence of a variable is where the value of the variable is used to compute the value of other variables or as an output value. A predicate use occurrence of a variable is where the value of the variable is used to determine the immediate execution path. A def-use pair is a pair of definition occurrence and use occurrence of a variable that can be linked by a path without passing any other definition occurrences of the same variable.

All definitions criterion is a test coverage criterion that requires that an adequate test set should cover all definition occurrences in the sense that, for each definition occurrence, the testing paths should cover a path through which the definition reaches a use of the definition. All use criterion requires that all uses of a definition should be covered. Clearly, all-uses criterion is stronger than all-definitions criterion. An even stronger criterion is all definition-use-paths criterion, which requires the coverage of all possible definition-use paths that either are cycle-free or have only simple cycles. A simple cycle is a path in which only the end node and the start node are the same.

We typically spend about thirty to fifty minutes on this topic. The course is about the entire software engineering lifecycle. The range of other topics to cover in this course, as described in Section 2, make it difficult to allocate more class time for this topic. In addition, the in-class activities that we do in the classroom are effective but often time-consuming, which make it even more difficult to allocate more regular class time for this topic. We intentionally omitted the concepts of the all-predicate-uses criterion, the all-computational-uses criterion, the all-c-uses/some-p-uses criterion, and the all-p-uses/some-c-uses criterion because of this time constraint.

We currently deliver this topic by a short lecture on the concepts, a walk-through on blackboard with an example, and an in-class paper-and-pencil exercise on a question such as the one listed in Table 2. We often follow up in the next session with a short quiz on a similar question.

Table 2. A sample question on definition occurrences and use occurrences.

Question

Assume that all variables are integers (of type int). In the following code:

```
1       i = get_a_number_from_console();
2       j = get_a_number_from_console();
3       if (a<30) {
4               i= get_a_number_from_console();
5       } else {
6               j=5;
7       }
8       if (a>b) {
9               i=8;
10              j=6;
11      } else {
12              printf("%d\n",j);
13      }
14      if (b<30) {
15              printf("%d\n",i);
16              j=6;
17      } else {
18              printf("%d\n",i);
19      }
```

(1) Which lines are definition occurrences of variable *i*?

(2) Which lines are use occurrences of variable *i*?

(3) For variable *i*, how many feasible def-use pairs are there? (Note that a "feasible" def-use pair can be covered by a test case.) Identify these def-use pairs using their line numbers.

Answer

(1) Lines 1, 4, and 9.

(2) Lines 15 and 18.

(3) There are six def-use pairs, only five of which are feasible.
1-18 (a=35, b=32)
1-15 (a=35, b=29)
4-15 (a=25, b=24)
4-18 (not feasible)
9-15 (a=29, b=25)
9-18 (a=35, b=34)

4. Discussion

Through several offerings of CS456/556, we found that it was difficult, using only lectures and paper-and-pencil exercises, to help students learn all relevant concepts, understand for example why there should not be a definition occurrence of the same variable in the path of a test case that covers a def-use pair, and figure out how to construct test cases according to data flow testing criteria. Based on quiz results and our observation during in-class exercises, less than half students had mastered this topic satisfactorily.

Most students understood the concepts of definition occurrences and use occurrences. However, some of them failed to understand the significance of the links between definition occurrences and use occurrences of the same variable. Therefore, they could out follow the three data flow testing criteria. Our speculation was that some of these students could not visualize program execution based on source code or simple flow charts that we displayed during the discussion of this topic because their learning styles [14]. Since we cannot devote more class time to this topic, we sense that we need teaching tools to help get the messages across more effectively. For example, a hands-on, step-by-step simulation or animation may help these students understand these concepts. However, we cannot find anything currently available that fits our need.

For those students who did master the topic of data flow testing, these concepts helped them reflect on control-flow testing that they learned earlier. They could observe that test suites generated according to pure control-flow-based test criteria can be trimmed in many ways, data-flow-testing being only one of them. Later on, when the last software testing concept covered in CS456/556, the concept of equivalent classes, was discussed, these students had more concrete, related examples of equivalent classes in mind and were able to understand and explain this concept faster.

Overall, we conclude that data flow testing is a worthwhile topic to cover in the tight schedule of the software engineering course CS456/556. However, to improve learning of this topic for students of all learning styles, including aural, visual, read/write, and kinesthetic learners, we need to develop innovative tool support and improved delivery methods so that this topic can be delivered in the limited amount of time.

References

[1] Hong Zhu, Patrick A. V. Hall, and John H. R. May, “Software unit test coverage and adequacy,” ACM Computing Surveys (CSUR), Volume 29, Issue 4, December 1997.

[2] P. G. Frankl, E. J. Weyuker, “An applicable family of data flow testing criteria,” IEEE Transactions on Software Engineering, Volume 14, Issue 10, Pages 1483 –1498, October 1988.

[3] Ivar Jacobson, Grady Booch, and James Rumbaugh, *The Unified Software Development Process,* Addison Wesley, 1999.

[4] R. F. Verderber and K.S. Verderber, *Communicate!*, 10th Edition, Wadsworth Publishing, 2002.

[5] Chang Liu, "Using Issue Tracking Tools to Facilitate Student Learning of Communication Skills in Software Engineering Courses," accepted by the 18th Conference on Software Engineering Education and Training (CSEE&T), Ottawa, Canada, April 18-20, 2005.

[6] Chang Liu, Karin Sandell, and Lonnie Welch, "Teaching Communication Skills in Software Engineering Courses," abstract accepted by the 2005 American Society for Engineering Education (ASEE) Annual Conference and Exposition, Portland, Oregon, June 12-15, 2005.

[7] Edmund Tsang, Projects That Matter – Concepts and Models for Service-Learning in Engineering, AAHE, 2000.

[8] Lonnie R. Welch, Sherrie Gradin, and Karin Sandell, "Enhancing Engineering Education with Writing-to-learn and Cooperative Learning: Experiences from a Software Engineering Course," Proceedings of the 2002 American Society for Engineering Education Annual Conference & Exposition, Session 2558, 2002.

[9] Kevin W. Bowyer, "`Star Wars' revisited - A continuing case study in ethics and safety-critical software," IEEE Technology and Society, 21 (1), pages 13-26, Spring 2002.

[10] Chang Liu, "Enriching Software Engineering Courses with Service-Learning Projects and the Open-Source Approach," Accepted by the 2005 International Conference on Software Engineering (ICSE) Education Track.

[11] Chang Liu, "Supporting Cross-Term, Cross-Team Projects in Software Engineering Courses," the 24th Annual Lilly Conference on College Teaching, Marcum Conference Center, Miami University, Oxford, Ohio, November 18-21, 2004.

[12] Chang Liu and Christine Wolfe, "Project Selection in Software Project Courses," abstract accepted to the 2005 American Society of Engineering Education (ASEE) North Central Section Spring Conference Hosted by Ohio Northern University, April 7 & 8, 2005.

[13] Chang Liu, "Adopting Open-Source Software Engineering in Computer Science Education," the 3rd Workshop on Open Source Software Engineering: Taking Stock of the Bazaar, held in conjunction with the 2003 International Conference on Software Engineering, Portland, Oregon, USA, May 3, 2003.

[14] Neil Fleming, VARK: A Guide to Learning Styles, http://vark-learn.com.