Why ‘Producing Production Quality Software’

Why did I decide to teach this course? In Fall 2002, for the first time as an NYU professor, I decided to read all of the code students wrote for an assignment. In the past I’d just graded student code with functional tests. (I run a grading program that provides input to the student code, and receives its output. If the student code produces the correct output for the input, then I assumed the student code was working properly.) I was horrified by the poor quality of my student’s code. (Of course, none of you were in the class!)

Several points became clear. Many students did not know how to do a good job at the following activities, all of them crucial to developing good software.

1. Design
   a. Verify that they understood the specification for the software they were to write
   b. Design reasonably efficient algorithms
   c. Wisely use libraries and other existing tools
   d. Handle synchronization, messaging and other issues of concurrency and interprocess (and thread) communications

2. Implement
   a. Structure their software, that is, clearly organize the type declarations, code and documentation in multiple files
   b. Write well-organized software, that is, do a good job at layout, variable naming, block structure, etc., within one file
   c. Write general purpose code

3. Test
   a. Write comprehensive test cases
   b. Test a program’s I/O with test driving programs

I hope that this course will teach you how to do these activities well.

Learning Attitude

It will be challenging to adapt and maintain a successful learning attitude in this course. I (and the TA if we have someone) will be reading a lot of your code, and criticizing it. (If we cannot find anything to criticize, then you know everything I’m teaching already and you’re wasting your time and money taking this course!)

People have varying responses to criticism. At one end of the spectrum, people strongly dislike being criticized, because they feel personally attacked. They typically respond by defending their actions. At the other end of the spectrum, some people are able to accept criticism as an instructive discussion of their actions, and not a personal attack, and respond to the criticism by reevaluating their action and an effort to improve their action. I hope you can adopt the later attitude in this course.

I will try to be aware that people can be attached to their software, and sensitive to criticism of it. We will try to make our criticism ‘constructive’.
Multiple Good Approaches Exist

Multiple good approaches exist for many of the problems we’ll attack. I may mention only a few approaches. For example we’ll talk about top-down and bottom-up design, and structured and object-oriented design. You may think of or know about others. Please feel free to mention them. There’s more than one way to skin a cat.

Software Development And Engineering

How does this course fit into the overall picture of software development and software engineering?

Roughly speaking, developing good software involves the following activities.1

1. Specification
   a. Identify a problem that custom software could solve
   b. Determine a set of desirable features the software might have
   c. Write the specification, which specifies, in detail, the features and properties that the software to be built will have

2. Design
   a. Design the software’s architecture, that is, its basic structure and relationships to other software and users
   b. Design the software’s algorithms

3. Implementation
   a. Write test cases
   b. Code the software
   c. Test the software with the test cases, and return to when the tests fail
   d. Write more test cases
   e. Test the software with the additional test cases, and return to when the tests fail
   f. Read the software carefully, looking for places where it fails to meet the specification

4. Deployment
   a. Finish writing documentation
   b. Combine together multiple modules, and conduct system tests
   c. Deploy the system

5. Maintenance

A small program might involve only one module, and this process would proceed linearly. In this course we’ll focus on Design and Implementation. However, we won’t try to design clever, complex algorithms, as that expertise is the province of people knowledgable in the application domain.

A large program would involve many modules, and the steps from 2.b through 4 would proceed in parallel for multiple modules.

We’ll pay scant attention to some major issues, such as the following:

- Specifications
- How large groups of people (needed to write large programs) work on multiple modules in parallel during design and implementation.

---

1 The ‘implementation’ section describes a model of development which designs tests before the code is written. Certain unit testing philosophies advocate this approach. Most programmers do not follow it, however.