Teaching How to Write Hospitable Computer Code

Victor Norman
Calvin College, Computer Science Department
September 2011

Introduction

Prior to coming to Calvin College, I worked as a software development engineer in three different companies for a total of 14 years. I loved, and still love, to write computer code, as I find writing code to be a creative outlet. At about the eighth year of my time at the first company, I began to notice I was repeatedly assigned to work on products that went to completion, but never sold any units. That is, the software and hardware were designed carefully, implemented, tested thoroughly, packaged, and marketed, but the initial design given to us software engineers resulted in a product that none of our customers wanted. This realization made me think about my role as a software engineer, and why I spent so much time and effort working so hard to create software that no one would ever use. If no one was going to use this product, why bother working so hard? Why bother working so diligently to get the design correct? Why bother reviewing the code? Why bother writing, reviewing, and implementing test plans, and then fixing bugs for weeks and months on end? Why bother reviewing my own code to make it "perfect", with excellent documentation, excellent naming, and excellent design?

From these musings, I came to two conclusions. First, as a Christian I was called to do my work diligently as an offering to God, even if no human being would ever run my code or use the product I was helping create. Second, I had to continue to create the best code I could, write the best documents I could, and document the code the best I could, even though the code may never benefit any human being.

Now I, as a Christian professor, try to teach my students these same lessons. First, the students must learn not only how to write code that produces the correct output, but also create what I now call *hospitable code*. In other words, I teach that not only the function, but also the form of the code matters, to me and to God. I explain to students that this is code written for two "consumers": the computer that will execute the code, and also others who will come later to review, understand, modify, borrow, and extend the code.

You may ask, Who looks at computer code after it is written? To answer this question, one needs to know about a typical software lifecycle.

Software Lifecycle

There is a saying in the software development community: Code is written once, but read a thousand times. This saying illustrates an important point: a programmer, when writing code, needs to remember that he or she is not just communicating with the computer, but also communicating with those that come later, who will have to read and understand the code.

In my experience in software development, after code is written, the programmer himself must read and debug the code. Then, the programmer's team holds a formal "code review" in which the team reviews all code in the project, looking for bugs, poor coding style, missing documentation, etc. Then, during final integration testing, stress testing, and release testing, a larger group of developers and testers may need to read and debug the code.

Even after the code has been released as a product, it will likely need to be read again. (The lifetime of a typical program is 20 years or so.) In most companies, software goes through multiple revisions – 1.0, 2.0, 2.1, 3.0, etc. These revisions differ in two ways: new features are added, and existing defects are fixed. In the latter case, when code is being fixed, software developers spend many hours reading the existing code, looking for the defects. However, even when new features are added, it is often the case that programmers borrow and alter large portions of existing code to add a new feature.

So, code is read often after it has been written, and thus, the readability (or "form") of the code matters. But, what does it mean to write *hospitable code*? I address this question next.

Hospitable Code

Most people, when expecting guests, try to make their home hospitable by cleaning it, lighting it properly, making it comfortable, and warm. They prepare food and drink, and perhaps, entertainment. In short, they make the space welcoming. I teach my students that hospitable code gives the reader of the code this same sense of being welcomed, a sense of warmth, and a confidence that the code was created with care. Hospitable code welcomes the reader to come in and be comfortable, to enjoy the cleanliness of the code, to feel at home, and to see that the space has been carefully prepared with guests in mind.

How does one do this with computer code? A programmer has many choices to make when writing code, many of which affect the readability of the code. Let me give three examples of choices the programmer has which can affect the hospitality of the code.

- Clear and descriptive variable names. A line of code as simple as

  ```
  a = 92
  ```

  can be improved and be made more hospitable to the reader simply by instead being written:

  ```
  minAGrade = 92
  ```

  When a reader encounters the first line, the reader may not immediately understand the purpose of the code. This may make the reader anxious that he is already losing understanding of this code. However, if the reader instead encounters the second line, he can intuit immediately that the code uses this variable to indicate the minimum score that is an A grade. Thus, the reader is left more confident that he understands the code so far.
- Proper in-code documentation (i.e., comments). All programming languages (that I know of) allow comments to be written in the code. These are lines that are not executed by the computer, but are written only to communicate with a human reader of the code. The proper level of documentation in the code explains to the reader any tricky or non-obvious steps or structures in the code.
- Consistent indentation. All modern programming languages contain control structures that cause the code to execute code conditionally or repeatedly. The coding structures can become nested within one another. For example, here is a piece of code that is not indented:

  ```
  foreach elem in aList {
  if (elem.score < 60)
  {  newList.append(elem);
  aList.remove(elem);
  }
  }
  ```

  Understanding this code is difficult. However, if I indent the code consistently and carefully, it becomes much easier to see that the code removes all elements from `aList` have scores less than 60, and adds them to `newList`.

  ```
  foreach elem in aList {
     if (elem.score < 60) {
         newList.append(elem);
         aList.remove(elem);
     }
  }
  ```

  If one does not indent the code consistently, then the reader will find it difficult to determine how control flows through the code. This difficulty can undermine the reader's confidence in understanding the code's logic.

Why Christians Should Write Hospitable Code

In my classroom, I emphasize to students that they must get in the habit of writing hospitable code, because it is the Christian thing to do (and it is the only way to get a good grade in my class). I explain that a Christian should write hospitable code for these reasons:

Hospitable code is code written with others in mind.

I have argued above that a programmer writes code not only for a computer to execute, but also for others to read, modify, and re-use. Thus, the Christian computer programmer should write code keeping in mind that this code needs to serve others in the community. The programmer should have a servant's attitude, looking toward the needs of others. This is a Christian attitude, clearly demonstrated by Jesus Christ, who came not to be served, but to serve. The temptation for many programmers is to think that the code just needs to have the correct functionality, and its form does not matter. The Christian should remember that both functionality and form matter to those people who come later to use this code.

Hospitable code is code can be written to serve God.

A Christian can serve God by writing hospitable code, because the Christian is doing his or her best to create code that is readable, correct, and looks to serve the needs of others (1 Peter 4:8-11). I remind my students that writing computer code is a creative effort. (In fact, creating computer code is my personal creative outlet – it is one thing I enjoy doing in my "off hours.") In creating code, we emulate God in his acts of creation. In fact, the first characteristic we learn about God is that he is a creative being (Gen. 1:3). However, to truly emulate God's acts of creation, we must create things that are good. I teach my students that one way to create "good" programs is to write hospitable code.

Creating hospitable code demonstrates integrity and God's lordship over all.

I emphasize to my students that function and form both matter, to me and to God. God judges us not only by what we do, but also by who we are. Similarly, we need to create code that is "good" throughout the creative process. We don't want to be pharisaical in our creations, creating code that is a "whitewashed tomb" – beautiful on the outside, but ugly on the inside (Matt. 23:27). I believe considering the form of computer code to be important is a truly reformed attitude, and I believe it is also acknowledges God's lordship over all.