Writing Good Software
Lessons Learned The Hard Way

In 1986, I was one of the founders of the Advanced Software Engineering Education and Training (ASEET) team. We were composed of members from all four services, plus civilian contractors. Our mission was to spread the word about software engineering and how it was much more than just programming. ASEET is long gone, but I still teach, present, and consult. After years of working with universities, military organizations and DoD contractors, I started to learn (often the hard way) what the elements of good software engineering really are.

I boil it down to four attributes—software must be reliable, understandable, modifiable, and efficient.

You can easily add other items; affordable, usable, delivered on time, fault tolerant, and more. However, I have found that the above framework covers the attributes I expect in "good" code. When I teach software engineering, I use the above list. I find it so important that I put some form of question about this list on every test I give. This list is not my creation. I got this framework from *Software Engineering with Ada*, by Grady Booch, back in 1983. Mr. Booch was Chief Scientist for Rational Computers, one of the developers of the Unified Modeling Language, and now is with the IBM Thomas J. Watson Research Center, serving as Chief Scientist for Software Engineering. If this list was good enough for him, it is good enough for me. (As a side note, he also taught at the USAF Academy. I brag that I was the last person to hold the "Booch Chair of Software Engineering" when I taught at the Academy. It was not really a formal position. One day, somebody mentioned to me that I was using Grady Booch's old chair and office. The chair was replaced while I was there—so nobody else held the chair after me.)

I want to include 15 lessons I have learned over the years. Some lessons were easy to learn; some are ones I paid dearly for. Some are my own; some are lessons that I was wise enough to learn from others.

1. Always follow the Attributes of Good Software listed above.
2. You need a coding standard and the self-discipline to follow it.
3. Document why, not what.
4. Code as if the next person to maintain your code is a homicidal maniac who knows where you live. (John F. Woods)
5. A software engineering expert is a person who knows enough about what is really going on to be scared. (Adapted from P. J. Plauger)
6. The foolish learn from experience. The wise learn from the experience of others.
7. Want to be a good programmer? Maintain your own code after letting it sit for a few months.
8. Want to be a really good programmer? Maintain somebody else's code (and everybody else's code will always be "bad".)
9. Want to make your code foolproof? Not likely. They are breeding a better quality of foolish users at an amazing rate.
10. Need the code really bad? We can deliver it that way if you do not quit rushing us. (Courtesy of many friends –thanks to Gene, Les, Lindy, and all of the other founding members of the ASEET team. And thanks to all of the members during the 15 years the ASEET existed.)
11. You cannot have too many backups.
12. If you can overload basic operations (such as "+", "-", etc.), then your language values "cool" over maintainability. I do not want to see "x = x+1" and have to debug for hours to find out that it is not adding 1, but instead invoking a user-defined "+" function with bizarre side effects. "DoSomeWeirdOperation(x)" makes debugging and maintenance a lot easier. (Discovered independently by legions of Ada and C++ programmers. In one program I was debugging, I eventually discovered "a < b" was calling a user-defined function, but "a>b" was not. Given 12 programmers for a jury, I thought I could get off with "justifiable homicide.")
13. In many languages "float_num = integer_num1 / integer_num2" does integer division, truncates, and then converts to a float. "y = 3/2;" is always going to be 1, even if "y" is a float. (This is why I like strongly typed languages, which flag it a syntax error. I really learned this the hard way!)
14. Indexing strings and arrays? If you switch from languages that start indexing at 0 to languages that start at 1 (or vice versa) you are going to write loops with an "off by 1" error. (Relearned every time I switch from Ada to C/C++ to Ada…….)
15. Always carry a short extension cord when you travel. When you need to charge your computer, tablet and phone all at the same time, inevitably the single outlet in the hotel room will be behind the TV stand. (Nothing to do with software, but truly a lesson I have learned the hard way.)

E-mail me your own Lessons Learned The Hard Way. I will try and publish them in a later column.

David A. Cook, Ph.D.
Stephen F. Austin State University
cookda@sfasu.edu