Software Processes? How *Bohr*ing!

After 12 years in the software industry, I recently returned to teaching. I enjoy it immensely, and especially enjoy teaching the "Intro to Computer Science Principles." I find it refreshing to give students their first taste of computer science. I am always amused by their initial first attempts at programming. Just like watching a newborn calf take its first teetering steps, watching a "new mind" learn to write programs is both rewarding and entertaining.

In my graduate courses, however, I expect a somewhat higher standard. I am pretty strict about expecting my students to follow "good" programming practices—good documentation, readable code, etc. My standards are pretty high, and once the students learn what I expect, they eventually see how clear, easy-to-read code makes debugging and maintenance easier (either that, or they humor me until the end of the semester, after grades have been assigned).

In a recent grad class, I was lecturing on some subtle point of operating systems. I was using Unix as an example, and I had pulled up an example of Unix network code from a handy reference manual. As my class was going over the example, several of the students were shocked by some of the "poor" code used in the operating system. Poorly documented. Bad variable names. Heavens, even global variables! A couple of the students pointed out that I would have quickly given them a grade of zero for writing code like that.

My response to them involved the International Obfuscated C Code Contest (IOCCC). The purpose of this contest is to award to most creatively obfuscated code (where obfuscated means to make things as confusing or difficult to understand as possible). A quick example, in the sidebar, is the code that will generate the entire 12 verses of The 12 Days of Christmas. No misprints—I compiled and checked it to make sure! See <http://en.wikipedia.org/wiki/Obfuscated_code>. At first glance, it looks like the random typing of a drunken cat as it trods across the keyboard.

Why do we celebrate poorly written code? Mostly, of course, for the humor. According to the IOCCC Web site <www1.us.ioccc.org/main.html>, it "shows the importance of programming style, in an ironic way."

In the IOCCC, I am sure that every one of the entries was written by an expert coder. They already know how to write very good code—and this contest is a safe forum poking fun at themselves. If my students wrote code like this? First of all, they would never be able to debug it, so grading would be *much* simpler. And I could always hope for a hammer to spring from a keyboard and crack a knuckle if a student tries to write such code. I teach my students to follow the rules, because it allows me to hold them to a standard. However, an experienced programmer who understands the system can safely break a few rules—as long as their experience allows them to do it safely. There are exceptions to every rule, and an expert knows when it's safe to break a rule. But you have to understand the processes *before* you start breaking them.

One reason I love the CMM/CMMI is the emphasis on "repeatable processes." A long time ago, I had a friend try and teach me how to golf. He said I was a perfect student—I consistently made the same mistakes over and over (and over and ...). It's easy to fix a problem when the problem is repeatable.

When designing software, you need to be able to understand a flawed process—and make it better. In software engineering, I teach about heisenbugs (named after the Heisenberg Uncertainly Principle) and bohrbugs (named after the well-defined Bohr atom model) which are bugs that are respectively difficult to reproduce *and* easy to reproduce. A heisenbug can't be reliably duplicated, so it's very hard to find and fix. A bohrbug, however, can be reliably reproduced, making it somewhat easy (or at least easier) to trace and fix.

A process model allows you to create bohrbugs rather than heisenbugs. While I'm not saying it is perfectly OK to create mistakes, I AM saying that if you want to improve your process, you have to at least screw up in a repeatable, predictable manner. And be willing to improve. And promise to NEVER write code like the 12 Days program.

How Not to Write Code
12days.c

```
#include <stdio.h>
main(t,_,a)char
*a;{return!0<t?t<3?main(-79,-
13,a+main(-87,1-_,
main(-
86,0,a+1)+a)):1,t<_?main(t+1,_,a):3,mai
n(-94,-27+t,a)&&t==2?_<13?
main(2,_+1,"%s %d %d\n"):9:16:t<0?t<-
72?main(_,t,
"@n'+,#'/*{}w+/w#cdnr/+,{}r/*de}+,/*{*+
,/w{%+,/w#q#n+,/#{l,+,/n{n+,/+#n+,/#\
;#q#n+,/+k#;*+,/'r :'d*'3,}{w+K
w'K:'+}e#';dq#'l \
q#'+d'K#!/+k#;q#'r}eKK#}w'r}eKK{nl]'/#;
#q#n')){}#}w'){}{nl]'/+#n';d}rw' i;# \
){nl]!/n{n#'; r{#w'r nc{nl]'/#{l,+'K
{rw' iK{;[{nl]'/w#q#n'wk nw' \
iwk{KK{nl]!/w{%'l##w#' i;
:{nl]'/*{q#'ld;r'}{nlwb!/*de}'c \
;;{nl'-
{}rw]'/+,}##'*}#nc,',#nw]'/+kd'+e}+;#'r
dq#w! nr'/ ') }+}{rl#'{n' ')# \
}'+}##(!!/")
:t<-50?_==*a?putchar(31[a]):main(-
65,_,a+1):main((*a=='/')+t,_,a+1)
:0<t?main(2,2,"%s"):*a=='/'||main(0,mai
n(-61,*a,
"!ek;dc i@bK'(q)-
[w]*%n+r3#l,{}:\nuwloca-O;m
.vpbks,fxntdCeghiry"),a+1);}.vpbks,fxnt
dCeghiry"),a+1);}
```

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