Re: Computer Programming Career

Source: http://coding.derkeiler.com/Archive/General/comp.programming/2004−03/1869.html

---

From: Michael Wojcik (*mwojcik_at_newsguy.com*)
Date: 03/23/04

```
Date: 23 Mar 2004 22:15:22 GMT
```

In article <c3l6lk$hc6$1@newsg4.svr.pol.co.uk>, "Malcolm" <malcolm@55bank.freeserve.co.uk> writes:
>
> *"Michael Wojcik" <mwojcik@newsguy.com> wrote in message*
> >
> > *The abysmal quality of most*
> > *software today is ample evidence that understanding basic*
> > *operations and algorithms is not sufficient to produce decent*
> > *software, since the vast majority of bugs are in implementation or*
> > *insufficiently−mature design, and not in choice of algorithm.*
> >
> *I think you have two classes of software problem. The first is where you are*
> *trying to get the computer to do something which is inherently difficult,*
> *like reading car registration plates. Here the algorithm is all. The second*
> *is where you are trying to get the computer to do something which is*
> *well−understood, such as sending bills to everyone caught in the congestion*
> *zone without a ticket, but is difficult to manage (because you've got so*
> *many terminals, so many people with legitimate excuses for not paying,*
> *foreign cars and inevitable mis−reads, and so on).*

I think that distinction is rarely useful in commercial software development. (I'd say it often is useful in some other contexts, such as when writing one−off data transformation programs, which in many cases never have to deal with malformed input, for example.)

In your first example, a great image−recognition algorithm won't be of any help if, for example, a careless programmer does a double free of a temporary buffer and corrupts the allocation system such that the entire program hangs at unpredictable intervals. Real programs operating in the real world suffer when they aren't robust, regard− less of how good the underlying algorithms are.

In any case, I don't see how this contradicts my original point: most extant bugs are design or implementation issues, which are often language−specific, not algorithm issues, which are not. So language− specific issues dominate software quality (or lack thereof).

> > *Knowing these will not produce a good C programmer. As*

> > evidence I offer the vast stretches of extant crap C code publically
> > available which were written by people who knew all of them.
> >
> "crap code" can be a very subjective definition.

Let me make it more specific. I'm referring to C code that invokes undefined behavior, including but not limited to in the presence of malformed input, or suffers from egregious design failures, such as unsafe use of temporary files.

Actually, the first is more than enough. Most of the publicly−available C code I've seen (and most of the proprietary C code I've seen, for that matter) invokes undefined behavior either when presented with malformed input or in the course of its normal operation.

Pick some C project at random off Sourceforge, say, and I bet there's an example of UB in it somewhere. Some projects have been reviewed for such errors and are in much better shape than the average (OpenBSD, for example), but most are still a mess.

This problem is not limited to C, of course, but that's the language under discussion here.

> There's code which performs
> on spec, there's code which does the right thing most of the time but not
> for some degenerate inputs, and then there's code which is definitely
> bugged.

In most cases code which breaks on invalid input is buggy. Contexts where such code is safe are relatively few.

*> > There are loads of C programmers who have used the language for *years**
> > and still make fatal pointer errors. Why do you think van der
> > Linden devotes so much of _Expert C Programming_ to clarifying
> > pointer and array confusion?
>
> It shouldn't be confusing.

Yet it is, as demonstrated daily in comp.lang.c.

> Indirection is a fundamental programming concept.

Arithmetic is a fundamental programming concept. Yet somehow many C programmers are confused by the subtleties of arithmetic in C, as demonstrated daily in comp.lang.c.

> C syntax is a little grotty, but it isn't so bad that it is unusable.

I never said it was. Nearly all of my work is in C, and I understand the difference between pointers and arrays in C. Many C programmers

do not.

I don't understand the argument you're trying to mount here. It
doesn't matter whether indirection in C *should be* confusing;
there's plenty of evidence that it *is* confusing. It doesn't matter
whether the C *syntax* for indirection is difficult; it's the
semantics that many people get wrong.

> > "[K]nock[ing] out a competent 'twenty questions'" does not qualify
> > as professional programming where I come from.
> >
> Programming isn't a profession.

Ah. I'll tell Micro Focus to stop paying me, then.

> If you can write "twenty questions" in C
> then you can write a working program,

Yes, "twenty questions". Which proves nothing about your ability
to write production code.

> and you can easily extend your knowledge to donkeywork problems.

This assumption is another of the major causes of low−quality code.
There is a world of difference between toy programs and production
applications.

> > > It depends where she wants to work. Most of the more
> > > interesting jobs will require C.
> >
*> > What's your evidence for *this* claim?*
> >
> Well a book called "Graphics Gems" was published and people in the computer
> graphics community invited to submit code. Any language was accepted, but
> every single contributor chose C. That tells you something about computer
> graphics.

It tells me something about the people who responded to the call for
submissions to one book.

> Graphics isn't the only area where this applies.

So you say. Again, where's the evidence that "most of the more
interesting jobs will require C"? Unless you're going to engage in
circular reasoning about what constitutes an "interesting job", I
think that's pretty much idle speculation.

> If the
> programming problem is to devise an algorithm to do something, then you can
> expect all your sources to be written in C, unless the problem is so
> specialised that a general−purpose language isn't useful.

More unsubstantiated claims.

I'm happy to grant that C remains popular with algorithm developers
and the like, language lawyers, and hackers. But it's certainly not
the sole language of choice for expressing new algorithms. There are
things that standard C does poorly or not at all, such as coroutines
or dynamic closures. Many algorithm developers are simply more
comfortable with another language, such as Fortran. And often
algorithms are initially implemented in another language for reasons
of portability or compact expression, because that increases the
chance that other people will give them a try.

> > Outside of c.l.c, I generally cringe when I read C source; nearly all
> > of it is bug−ridden, nonstandard, poorly documented, and sloppy.
> >
> And many of these people are earning a living as programmers.

Yes, that's the problem.

> Knowing enough
> C to write programs in it and being a really good C programmer are two
> different things. You have to go through stage 1 to rise to ascended master.

Yes, and we'd all be better off if fewer people who aren't really good
C programmers were writing C code for other people to use.

If we're going to encourage people to learn C, we ought to be
realistic about it. Judging from the average quality of C code, very
few people are capable of quickly becoming "really good". And as far
as I'm concerned, they should be "really good" before they're
producing code for public consumption. We're awash in software that
crashes and produces incorrect results and permits abuse because far
too many people think mostly OK is good enough. It isn't.

> But C isn't the only language like that − for instance C++ lends itself far
> more readily to abuse and horrible kludges.

Agreed. I have nothing against C. It's my favorite procedural
language, if I had to pick one, and the one I do most of my work in.
But most people who write C do so poorly.

This is not a religious argument about C. It's a religious argument
about attitudes toward software quality and the factors that diminish
it. And one of those is encouraging programmers to work in areas
where they lack sufficient expertise.

```
--
Michael Wojcik                  michael.wojcik@microfocus.com
Unlikely predition o' the day:
Eventually, every programmer will have to write a Java or distributed
object program.
   -- Orfali and Harkey, _Client / Server Programming with Java and CORBA_
```