



Published on Free Software Magazine (<http://www.freesoftwaremagazine.com>)

# The seven sins of programmers

## Fixing bugs in the coder, not the code

By Steven Goodwin

Programmers. The system administrators worship their bit twiddling capabilities. The users exchange vast quantities of beer for new features and tools. And the project managers sell their souls when they make the magic work. But inside the average programmer's psyche are several demons that need exorcising.

## Pride

This is all too common in programmers. Instead of asking whether a particular function exists, or for the best way to retrieve data from the system, a proud programmer is likely to write their own. Especially when faced with a large, or unfamiliar, code base. By re-inventing the wheel there are now two very similar routines. Not only does this increase the code size, it doubles the amount of maintenance required, creates another opportunity for bugs, and adds inconsistency. Later, when another programmer sees these two functions, they will have to choose between them. Which one will depend on their mood (are they also too proud to seek help?), who's on holiday or who's outside smoking, at the time! This can equally be applied to duplicate constants, member variables or structures.

### **Code reviews... must focus on the code, not the coder**

Code reviews with a senior team member can help quell a developer's pride, and guide the other developers in the appropriate direction. This is basic employee training, and one that is simple to implement. Reviews also force the developer to reason each decision made, and justify the creation of new utility routines, perhaps explaining to the lead why existing standard code was not used. To be constructive the review must focus on the code, not the coder, and support free flowing ideas, regardless of the relative seniority of the reviewers. Remember that if the developer is too proud they'll be closed to ideas, or won't suggest improvements.

## Envy

Programmers should improve themselves by learning from others, not blindly emulating them. All coding methodologies (be they syntactical or design-based) come with caveats. A programmer might inline a function with the pre-processor because he's seen it done elsewhere, unaware of potential side effects, as in this classic example.

```
#define SQUARE(x)    x*x
```

Similar evils occur when programmers move between languages. An experienced developer will typically have a specific style that he tries to crowbar into every other language. Perl written like C. Java written like Ruby. We've all seen the examples. Naturally, you should use the best tool for the job, and work to the strengths of that language. By trying to fit idioms from one language into another highlights the fact you understand neither. It's a development fact of life that some languages are better suited to some tasks, so adapt

## The seven sins of programmers

to it, and close those envious eyes that look at the language you'd rather use. Remember the oft-quoted saying, "When all you have is a hammer, everything looks like a nail".

### Gluttony

Not an evening at the all-you-can-eat buffet, but the problem of writing too much code. Every hour spent coding will require an extra hour of testing, and possibly two more of maintenance. And guess who gets lumbered with the maintenance?

Worse still, this does not scale. A two-minute feature (or one line bug fix) may also take an hour to test. Whatever your gluttonous reasons are—attempts to impress, an under-staffed team, late night camaraderie, or personal pride—curb them. Spending the whole morning trying to unravel last night's two-minute feature is like the buffet—you get stuffed afterwards!

### Lust

Programmers crave pleasure; they love to "scratch their own itches". If unchecked, some developers will write fantastic, innovative, original... and completely unnecessary, code. It could be a new graphics shader, a faster search algorithm or an entire processing engine! If you're working on anything other than a pet project, you will probably have a schedule that must be adhered to, and a set of pre-determined priorities. These affect the whole project, and not just the wanton lust of an individual developer. Unless you are called Spock, the needs of the many, outweigh the needs of the few... or the one.

Make sure any code written is actually needed. Will it get used? Or will it distract the users who, in turn, add a new plug-in system, just to make use of the code? One very common example of this is premature optimization. Without profiling the code it is impossible to tell exactly where the bottlenecks will occur. Granted, most developers can look at a function and give several reasons why it is slower than it could be, but very few can tell you what percentage of the total running time that function will occupy. Unless it's known to be on the critical path, then it's probably not worth optimizing at this time.

The desire to write new code can also exclude the possibility of introducing middleware as a viable solution. The cliché is then of "Not Invented Here". Programmer X once said,

"I'm not using middleware since I don't understand it; it'll be just as quick to write it myself. How can I be expected to maintain something I don't understand?"

He was erroneous for many reasons. Firstly, the initial period of evaluation would give him experience with the code base, and should be a pre-requisite when introducing any new code to a project. Secondly, he's only considered the development time in terms of coding, when in actuality much of it is taken up with testing—something any sensible middleware product would have in abundance—and as we've already seen, writing code doesn't scale to the subsequent testing and maintenance phases. And finally, there are other trade-offs between write and buy, such as access to other developer forums, paid consultancy, and third party contracts which determine whether the purchase of middleware is a good idea *for your specific project*.

Curbing programmer lust in this way allows more time spent on the important tasks. This would include writing original, novel, parts of the software and learning the middleware solution itself. After all, it is usually quicker to read, than to write.

## Anger

Do not code out of anger. Do not ignore good ideas, regardless from where they come. If a more junior programmer has a solution to the problem in hand—discuss it. If it works—use it! The engine programmer should not be allowed to implement his own solution just because “He’s the engine programmer”.

**Anger leads to hate. Hate leads to suffering. To bad code does that lead**

Do not code out of spite. Lead programmers: love your coders. Code reviews, for example, should raise the ability of the whole team, and not be used for leads to show off, introduce new jargon, demonstrate obfuscated syntax or exhibit other prima donna characteristics.

Do not code out of fury. Programmers: love your leads. They distribute work because it needs doing. Don’t work on somebody else’s tasks because you’re more suited, or believe it should have been yours. If you want to move into other areas of programming, talk to your lead. Develop prototypes at home. Employing enthusiasm in this manner will win more brownie points than ignoring the task list and schedule.

## Sloth

Don’t procrastinate! If a particular piece of code is uninteresting or difficult (like an obscure crash bug), more interesting tasks should be available to compensate. Look forward to those tasks, but don’t daydream about them. If you stop every five minutes for a coffee and chat (or more likely, a whinge) then the task will take much longer, and it will become a self-fulfilling prophecy. Instead, begin with a cup of coffee, a bag of sweets, and your favorite MP3s. Then lose yourself and knuckle down to the task in hand. It won’t be as bad as you think as even dull work makes time pass quickly if you become engrossed in it.

Also, make sure all the tasks are clear, consistent and given from one manager. Opposing requests from different managers will make one of them unhappy, and starting such a doomed task is no fun for anybody.

## Greed

There are a couple of places where developers suffer greed. We have already touched on one, and this is a programmer’s innate desire to do too much. The proverbial “biting off more than you can chew” scenario leads to an exponential increase in testing, and a swell of code paths to verify. It can also lead to a lower quality of code since the problem domain may not be well understood by the developer assigned, and the increased workload limits their opportunity to learn.

The final greedy point is directed more towards management, as everyone should feel valued—financially and metaphorically. This is especially true during crunch-time, when even the most junior programmer can work out that their hourly pay packet could be improved by working on the cold meat counter at Tesco! Paying for late night food goes without saying (I hope!), but an occasional pub evening also helps. This gets everyone out the office, and shows that management aren’t greedy with an employee’s time, either. Many ambitious people, regardless of salary, always want more. So even the lead programmer will start questioning their role and think “I’m worth more than this” when they feel unappreciated. How many times do you hear “I’m so overpaid”, compared to “I’m so underpaid”?

Leads should check for warning signs, like “funny” comments in the code—“I’ll do this properly when I get a fcking pay rise!!!”. Management should be wary of stopping or limiting low-cost company perks (such as

## The seven sins of programmers

free soda) since the loss in productivity and willingness to do overtime is undoubtedly greater than the few dollars spent. Follow Dilbert. Less money for staff does not mean more for management.

Naturally, the lead programmer should help prevent such sinful practices. But as responsible professionals, we should all try and curb the devil inside first. Of course, not everyone is Beelzebub incarnate, so score yourself honestly out of ten for each category, and ask a colleague to do the same for you. Then compare. I score 4, 2, 6, 1, 1, 2 and 3 on the categories above, but pride prevents me from letting you in on which scores are for which categories...

### Biography

Steven Goodwin (/user/39" title="View user profile.): When builders go down to the pub they talk about football. Presumably therefore, when footballers go down to the pub they talk about builders! When Steven Goodwin goes down the pub he doesnâ€™t talk about football. Or builders. He talks about computers. Constantly... He is also known as the angry man of open source. Steven Goodwin (<http://www.bluedust.com/blog/>) a blog that no one reads that, and a beer podcast (<http://www.thebeercrate.com>) that no one listens to :)

### Copyright information

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is available at <http://www.gnu.org/copyleft/fdl.html>.

---

#### Source URL:

[http://www.freesoftwaremagazine.com/articles/the\\_seven\\_sins\\_of\\_programmers](http://www.freesoftwaremagazine.com/articles/the_seven_sins_of_programmers)

---