Controversial Ideas about Programming

Gene Michael Stover

created Saturday, 2003 January 11
updated Wednesday, 2007 August 8

Cite: Please cite with a form similar to that of the entry for this article in its own bibliography ([**?**]). Please do not deep-link into this document. Use just the URL from that citation.[1]

Contents

[1] I will not threaten legal action if you deep-link, but I cannot guarrantee that the nodes within this document will have constant names (URLs). I ensure only that the URL in the citation will remain. People & companies who threaten legal action for deep-linking are incontinent bozos.

1 Introduction

Here are some of my more controversial beliefs about programming. Why? 'Cause I feel like writing them down.

I may or may not write just a few sentences about it, leaving the idea cryptic & up to the imagination. What? You don't have enough imagination to fill-in the holes? Life is pretty rough sometimes. Sorry about that.

If you don't think they are controversial, good for you. You & I think alike. (Maybe the world has finally seen the wisdom in my ways & caught up with me.)

I freely admit that a more accurate title for this document might be "Gene Bitches about Programming".

2 Macros

definition: A *macro* is a function whose arguments are not evaluated but whose return value is evaluated.

Exercise for the reader: Consider C macros, Lisp macros, M4 macros, & keystroke macros (such as in a spreadsheet) in the light of this definition of macro.

3 Object-Orientation

Object-orientation isn't the only way to architect a program. It's not the best way, though maybe there are no better ways.

You see objects where you look for them. You create objects if that's the way you think.

You see other structures & patterns where you look for them. You create them, not objects, if that's the way you think.

There is no best way to write a program. Some programs work better with a non-object-oriented design.

Look at it this way: It's possible to write crap even though it's object oriented, right? So isn't it possible that some program might be good even though it's not object oriented?

4 The "Singleton" Design Pattern Does More Harm Than Good

Just because it's object oriented doesn't mean it's good design. I've never seen a singleton that had a right to exist. If you think you need a singleton, you are probably wrong. Others have discussed it so I don't need to.

- "Conversations: Once Is Not Enough: Don't let Singletons degenerate into global variables", by Jim Hyslop and Herb Sutter, http://www.cuj.com/articles/2003/0303/0303i/0303i.h
- "Singletons are evil, and other Design Pattern Dangers", http://coding.strangesearch.net/archives/00001
- "Design Patterns Considered Harmful", by `dws` on Dec 19, 2001, http://perlmonks.thepen.com/133399.htr

I'm sure there are some good reasons for using singletons, but they must occur as frequently as winning Lotto.

5 C++ & Java Are Bad Object Oriented Languages

Static typing has no place in a truly object oriented language, at least in one that wants to support true polymorphism. Templates, which C++ has had for years & which Java has recently added, are just a band-aid on the deficiency of compile-time types. Besides providing inferior polymorphism, the syntax of templates is just horrid. C++ is probably the programming language I know best, but it's the only one in which I have to fight the compiler to make things work.

Besides that, C++'s syntax is so complex that it's just crap. It's more complex than Ada or Algol, for Christ's sake.

To get a taste of real object-oriented programming, try Smalltalk (which defined the technique), Self (which is proud to have no class), or CLOS (an object-oriented library for Lisp). Heck, even Perl does a decent job of providing real object-oriented features.

6 Multi-dispatch

Programmers who use C++, Java, & certain other languages are talking lately about the revolutionary new idea of "multi-methods", methods which are selected by the actual class of more than just a single target object.

Lisp programmers have had it for, oh, 20 years I guess. I believe it was originally part of LOOPS, & it is certainly part of CLOS now.[2]

7 Ada Is a Good Programming Language

One of Ada's design criteria was that the average programmer could do more good than harm with it.

Since most programmers are average by definition, Ada is a good language.

Ada has a lot of nice features. It also has simple syntax, & it uses mostly alphabetic characters, few special characters, so it's easy to type.

Ada rocks.

8 XML

XML is a family of languages that share the same lexer & whose parsers could share some parts.

XML is the data-definition side of Lisp, but without the parenthesis. Instead of parenthesis, it has begin-tags & end-tags, thereby increasing the likelihood of a syntax error.

9 SOAP Is RPC

SOAP is just Remote Procedure Calls (RPC) encoded in XML & transported by HTTP. Nothing new here, folks. Move along.

If SOAP is so cool, why did you ignore Sun RPC when it was the only RPC in town?

10 Files Are Interprocess Communication

Files are a form of interprocess communication (IPC) that does not require the end-points (processes) to exist at the same time.

11 "Methodology" Is a Stupid Word

When you say that Extreme Programming or Waterfall are methodologies, you're wrong. They are software development processes. Possibly, you could call them methods.

Methodology is the study of methods.

[2] You probably didn't know Lisp has object-oriented features, huh?

12 Extreme Programming Is Nothing New

Extreme Programming & the other light-weight software development processes are nothing new.

Test-driven design? I've been doing it since I discovered it on my own in the early 1990s, & I'm certain I'm not the only one.

Programming pairs & code reviews? They're old ideas that are good when used in moderation. When used to the extreme, they suck. Especially pair programming. What, programming is a social exercise?

Code is maleable? Sure is. Obviously. In fact, it's only gotten more maleable over the years when you consider that "code" on ENIAC was jumper cables. Nothing new here, folks. Move along.

Lower-casing the first letter of a name & upper-casing the second letter is stupid.

13 Software Development Processes Will Never Solve the Software Crisis

Good software development depends on good software developers: programmers, analysts, managers, testers, & others. Hell, it even depends on good customers.

Processes can remove only so much of the risk of software development. Once you've removed all the risk you can, you must rely on the skills of your people, & they will need to do some hacking.

Hacking is experimental programming to test theories & algorithms, and to work your way through libraries, APIs, protocols, & other tools. There will be kinks in the works for which you cannot plan. The only way to find them is to hack through it. Bottom-up implementation. That's hacking, & you can't develop most programs without it.

The only program which you can design before you code is the program you've written so many times that you know everything it needs to do. (Nope, you can't always put it in a reusable library. Sometimes, you have to write from scratch.) That's the only program you can design completely before you code. And why is that? It's because you've done all your hacking on your previous implementations!

For what it's worth, I don't believe the software development crisis is a crisis. It's just a fact that most software development these days is bad software development. If we never come to the day when most software development is good software development, life as we know it will still continue. Trust me; I know these things.

14 Understand Your Requirements

Programmers too often don't understand their requirements. Has it occurred to you that when you have those design meetings, you are usually just getting together to understand the requirements? It's *great* (necessary, in fact) to understand your requirements, but if that's what we're doing, let's call it that & not call it designing.

Yes, the processes of understanding your requirements & of designing your software overlap.

15 Word Processors

Word processors are bad. I don't mean that the popular word processors are buggy. I mean that word processors, as a concept, are bad.

You should use a non-graphical document-description language. Yes, even if you're not a programmer.

- It improves the quality of your writing by allowing you to concentrate on composition. You consider presentation later (& sometimes you don't have to consider it at all).
- It's easier to learn, easier to use.
- It's portable.
- It's more suitable for archiving.
- The file format isn't proprietary, so other programs can generate documents in those languages. In other words, document-description languages are more flexible than word processors.

LaTeX is probably the best document-description language for documents written in English & some other natural languages, but you could do worse than use HTML.

By the way, LaTeX is a better type-setter than Microsoft Word.

16 Virtual Machines & Portability

Run-time virtual machines, like Java & Microsoft Dot-Net use, are the wrong way to achieve portability in most cases. It would be better to compile the source code to a portable subset of C, then compile that C to native at installation time.

I wrote an essay about this ([**?**]).

17 Corba

Corba never delivered. Sure, sure, technically, they delivered the standard & companies have implemented it. In practice, though, it was too little, too complex, & too late. Too bad, too. The world could have used an open, language-independent remote object protocol.

18 How to Pronounce URL

If ever there was an acronym that begged to be pronounced as a single word, it was URL. Pronounce it like the pre-existing English word, "earl".

If URL is pronounced like "you are ell", then COBOL is "see oh bee oh ell", FORTRAN is "eff oh are tee are ay in", & BASIC is "bee ay ess eye see". Don't forget ASCII, DOS, RAM, & ROM.

References