

I Miss Lisp

by Philip Chu

Table of contents

1 Publication Information.....	2
2 What I Miss About Lisp.....	2
3 Why Didn't Lisp Become Java?.....	4

1. Publication Information

Copyright ©2006-2008 by Philip Chu All rights reserved.



My favorite language is Lisp. What happened?

2. What I Miss About Lisp

2.1. Simplicity

A common complaint against Lisp is the parentheses. Lots of Interspersed Spurious Parentheses. That's pretty funny, but it's wrong. The parentheses are not spurious - each one

serves a purpose, and if your program compiles, you have exactly the number you need. And there is no question about operator precedence or scope. I've never heard Lisp haters say they're confused by parentheses - they just find them annoying.

But a combination of curly braces, parentheses, brackets, periods and commas is simpler and more succinct? No one has ever had to pop open a Lisp book to remember its syntax (although I did wear out my CLtL2 referencing all the built-in functions). Think how many hours have been lost to Java and C/C++ operator precedence bugs, forgetting to add brackets when expanding a single-line if clause (and Eclipse will nag me about unnecessary brackets if I try to play it safe) and the endless debates among programmers about their favorite bracket usage and placement conventions.

2.2. Elegance

A related complaint is that Lisp is too strange and hard to understand. There is a misguided notion that programming languages should resemble English. Which has given us COBOL, BASIC and AppleScript. (I have fond memories of BASIC, and Microsoft had a pretty good run with it, but no one's going to write AI programs or even web server applications with it). English expression of computer programs is as efficient and clear as writing mathematical formulas in English.

Simplicity lends itself to elegance. You know you're using an elegant language if you're not distracted by discussions of proper style. I don't recall wasting time on coding style debates when working in Lisp. But after moving to Java and C++, nearly every organization I joined made a big deal about adherence to a coding standard, which usually involved some variation of the the abomination known as Hungarian notation. (Although Sun has defined a perfectly readable and simple style exemplified in the Java standard class libraries, and Hungarian notation is a monster when applied to object-oriented languages). A foolish consistency is the hobgoblin of little minds, the saying goes. So is a mandated coding style. When you're programming, you should be thinking more about the problem than the language.

2.3. Flexibility

The syntax makes possible a unique feature of Lisp, one that every satisfied Common Lisp user seems to appreciate - macros. "Macros" seems like such a mundane name - C macros or Excel macros are hardly in the same league. What other language allows you to implement new constructs that can even define new control flow, in a way that blends in with the built-in syntax? It's a small price to pay for parentheses.

After using Flavors, Object Lisp, C++ and Java, I still have to say the Common Lisp Object System is the best. Generic functions, eql parameters, multiple inheritance (yes, it's a good

thing), a metaobject protocol that lets you customize the object system - nothing else matches it.

2.4. Optimization

A fallacy about Lisp is that it's slow. Of course, it's easy to write slow code in any language, and even easier in Lisp because 1) it's big, and tends to be slow just to start up, 2) automatic memory management and a run-time type system has an overhead and 3) Lisp is easy to learn - any idiot can learn it well enough to produce a complicated, messy prototype of something that looks sophisticated (although it takes an idiot with taste to appreciate it).

But it's really the best of both worlds. You can prototype an application quickly and clearly, and then optimize - at the algorithmic and data structure level as with any language, but also at the language level, e.g. by adding declarations that turn off run-time type checking. This avoids a scenario like the application I worked on that was prototyped in Tcl - and stayed in Tcl for two years while the development tried to figure out how to migrate it to another language.

It's painful to watch Java follow in Lisp's footsteps and yet apparently not learn all its lessons. I worked on a high-performance 3D content creation tool in Lisp - I still cannot imagine that a 3D Java application can perform as well.

2.5. Culture

Lisp had a cool culture. Much like AI labs, it attracted groupies of dubious ability but also some of the most brilliant programmers around. And I mean brilliant in a worldly, literate, sense. Lisp programmers don't make lists of design patterns, although Richard Gabriel did a lot of writing on how design patterns were originally applied to architecture. Among the well-known names involved in the origination, development and popularization of Lisp - Paul Graham, Peter Norvig, Guy Steele, Marvin Minsky, Rod Brooks, Richard Stallman, James Gosling....

3. Why Didn't Lisp Become Java?

3.1. A Hook

Whether it was a brilliant marketing move or stroke of luck, the introduction of Java initially for browser applets was crucial. It's not quite so popular for that, now (another essay - Why Didn't Java Become Flash?) but it was a great introduction that made Java an everyday word. Java hasn't exactly dominated the desktop and has been hit and miss in various areas, but has been targeted successfully in the web server and mobile phone arenas.

Common Lisp never had such a hook. It was associated with AI and large, expensive Lisp workstations. Now it is the first choice for, what?

3.2. It Wasn't Free

If you want everyone to adopt a standard, it has to be free. Netscape understood it and Microsoft understood it when they wanted to crush Netscape. Everyone who wanted to get into this cool thing called the World Wide Web was able to download Java and play around with it.

There are free implementations of Common Lisp (GCL, CMU Common Lisp, CLISP), but they came late and incomplete. The commercial vendors have been slow and late to release free versions of their products and still do only under restrictive licenses (evaluation/educational/personal use). The open source versions don't have everything you really need, e.g. multithreading.

3.3. Class Libraries

Common Lisp is big, but doesn't include as a standard libraries that are important for real applications - multithreading, networking, user interface. Commercial Lisp implementations have their own versions of these libraries, but they're all different. For developers, the appealing part of Java's vaunted write-once-run-everywhere philosophy is not so much the bytecode portability - any polished app will have different installers for different platforms, anyway - but the fact that you can code to a single set of standard class libraries.

And Java has great class libraries. Not only do they supply just about everything you need and extend them as industry needs advance (without the lame Microsoft tactic of just adding new functions with "Ex" appended), they are typically well-designed - they express good API design, the standard Sun Java coding style, and model the underlying functionality well. For example, serious network programmers should read classic texts like Stevens, but just learning the Java network class libraries will give you a reasonable idea of what's really going on. And anyone concerned about coding style should just emulate conventions used in the Java API's.

3.4. Native Integration

Real applications also have to look like they belong on the machine they're running on. Java faced up to that in Swing - running with the Metal look and feel everywhere wasn't acceptable to anyone shipping a commercial product, so now you can switch to a Windows, Mac or Motif look and feel.

3.5. Too Big

It's easy to write "Hello, World" in Lisp. It just takes a while to load.

3.6. Too Good

The unique features of Common Lisp are largely unappreciated, even by Lisp devotees. I've seen macros used a lot, but mostly to inline code - few Lisp programmers think abstractly enough to define their own language within a language. I think CLOS is still the best object system out there, but most Lisp programmers I've seen are content to use lists. A West Coast Lisp programmer, during an interview for a startup using Dandelion isp machines, opined that "object-oriented programmers are born, not made". I'd have to agree - I've met many programmers complain about "too many objects", whether it's C programmers moving to C++ or Lisp programmers encountering CLOS.

In a sense, Lisp was too easy. A programmer who couldn't deal with C pointers could still hack together code in Lisp. I've seen plenty of prototypes that demoed well enough to get funding and turn into serious projects, then stall because the software was held together with string and bailing wire. And Lisp was too cool - it attracted a lot of groupies who weren't that great at coding but liked the culture and cachet (particularly in AI surroundings). So it's no surprise that Lispers come off as wine snobs to the beer-drinking masses out there.

3.7. Too Smart

It's a VB world out there. Most programmers don't really want to program - they want an IDE that will do as much work as possible for them, like graphically connect components and generate skeleton code. And they want the security of knowing they're using the same language, tools, API, design patterns, methodology...as everyone else.

Personally, I was happy running Lisp in Emacs, but if there was the equivalent of Eclipse for Lisp (or at least a VB-style environment), there might be a larger following.

3.8. Panic

While developing a commercial product in Lisp, I often heard, "No one will buy our product if they know it's written in Lisp", the implication being that we shouldn't tell any customers about Lisp, despite the obvious extensibility benefits, and that we should consider porting the whole thing to C++. For some reason, I never heard "We're having trouble selling the product because our salespeople aren't informed about the product, the company as a whole is uninformed about the target market, and we released the product with a lot of bugs."

I never heard any user complain about the implementation being in Lisp, but I did see one customer complaint about being "treated like a two-dollar whore" and I was called to a customer site to field angry tirades - I wasn't even on the project anymore at that point, but the regular sales support people couldn't take the abuse, anymore.

The attempts I've seen to move a project away from Lisp didn't solve anything and sometimes made things worse. Lucid doubled their size by buying a C++ company and then went bankrupt. A lesser-known CAD system that I worked on, called DROID and running on TI Explorer machines, had their own Lisp vs. C++ battle and somehow compromised on Smalltalk (now, what was the point of that?). Business people like to say that the product and its technology is secondary to smart business strategy and marketing, but when they screw up, it must be the technology.