Beating the Averages

April 2001, rev. April 2003(This article is derived from a talk given at the 2001 Franz  
Developer Symposium.)  
In the summer of 1995, my friend Robert Morris and I  
started a startup called   
Viaweb.   
Our plan was to write  
software that would let end users build online stores.  
What was novel about this software, at the time, was  
that it ran on our server, using ordinary Web pages  
as the interface.A lot of people could have been having this idea at the  
same time, of course, but as far as I know, Viaweb was  
the first Web-based application. It seemed such  
a novel idea to us that we named the company after it:  
Viaweb, because our software worked via the Web,  
instead of running on your desktop computer.Another unusual thing about this software was that it  
was written primarily in a programming language called  
Lisp. It was one of the first big end-user  
applications to be written in Lisp, which up till then  
had been used mostly in universities and research labs. [1]The Secret WeaponEric Raymond has written an essay called "How to Become a Hacker,"  
and in it, among other things, he tells would-be hackers what  
languages they should learn. He suggests starting with Python and  
Java, because they are easy to learn. The serious hacker will also  
want to learn C, in order to hack Unix, and Perl for system  
administration and cgi scripts. Finally, the truly serious hacker  
should consider learning Lisp:  
  
 Lisp is worth learning for the profound enlightenment experience  
 you will have when you finally get it; that experience will make  
 you a better programmer for the rest of your days, even if you  
 never actually use Lisp itself a lot.  
  
This is the same argument you tend to hear for learning Latin. It  
won't get you a job, except perhaps as a classics professor, but  
it will improve your mind, and make you a better writer in languages  
you do want to use, like English.But wait a minute. This metaphor doesn't stretch that far. The  
reason Latin won't get you a job is that no one speaks it. If you  
write in Latin, no one can understand you. But Lisp is a computer  
language, and computers speak whatever language you, the programmer,  
tell them to.So if Lisp makes you a better programmer, like he says, why wouldn't  
you want to use it? If a painter were offered a brush that would  
make him a better painter, it seems to me that he would want to  
use it in all his paintings, wouldn't he? I'm not trying to make  
fun of Eric Raymond here. On the whole, his advice is good. What  
he says about Lisp is pretty much the conventional wisdom. But  
there is a contradiction in the conventional wisdom: Lisp will  
make you a better programmer, and yet you won't use it.Why not? Programming languages are just tools, after all. If Lisp  
really does yield better programs, you should use it. And if it  
doesn't, then who needs it?This is not just a theoretical question. Software is a very  
competitive business, prone to natural monopolies. A company that  
gets software written faster and better will, all other things  
being equal, put its competitors out of business. And when you're  
starting a startup, you feel this very keenly. Startups tend to  
be an all or nothing proposition. You either get rich, or you get  
nothing. In a startup, if you bet on the wrong technology, your  
competitors will crush you.Robert and I both knew Lisp well, and we couldn't see any reason  
not to trust our instincts and go with Lisp. We knew that everyone  
else was writing their software in C++ or Perl. But we also knew  
that that didn't mean anything. If you chose technology that way,  
you'd be running Windows. When you choose technology, you have to  
ignore what other people are doing, and consider only what will  
work the best.This is especially true in a startup. In a big company, you can  
do what all the other big companies are doing. But a startup can't  
do what all the other startups do. I don't think a lot of people  
realize this, even in startups.The average big company grows at about ten percent a year. So if  
you're running a big company and you do everything the way the  
average big company does it, you can expect to do as well as the  
average big company-- that is, to grow about ten percent a year.The same thing will happen if you're running a startup, of course.  
If you do everything the way the average startup does it, you should  
expect average performance. The problem here is, average performance  
means that you'll go out of business. The survival rate for startups  
is way less than fifty percent. So if you're running a startup,  
you had better be doing something odd. If not, you're in trouble.Back in 1995, we knew something that I don't think our competitors  
understood, and few understand even now: when you're writing  
software that only has to run on your own servers, you can use  
any language you want. When you're writing desktop software,  
there's a strong bias toward writing applications in the same  
language as the operating system. Ten years ago, writing applications  
meant writing applications in C. But with Web-based software,  
especially when you have the source code of both the language and  
the operating system, you can use whatever language you want.This new freedom is a double-edged sword, however. Now that you  
can use any language, you have to think about which one to use.  
Companies that try to pretend nothing has changed risk finding that  
their competitors do not.If you can use any language, which do you use? We chose Lisp.  
For one thing, it was obvious that rapid development would be  
important in this market. We were all starting from scratch, so  
a company that could get new features done before its competitors  
would have a big advantage. We knew Lisp was a really good language  
for writing software quickly, and server-based applications magnify  
the effect of rapid development, because you can release software  
the minute it's done.If other companies didn't want to use Lisp, so much the better.  
It might give us a technological edge, and we needed all the help  
we could get. When we started Viaweb, we had no experience in  
business. We didn't know anything about marketing, or hiring  
people, or raising money, or getting customers. Neither of us had  
ever even had what you would call a real job. The only thing we  
were good at was writing software. We hoped that would save us.  
Any advantage we could get in the software department, we would  
take.So you could say that using Lisp was an experiment. Our hypothesis  
was that if we wrote our software in Lisp, we'd be able to get  
features done faster than our competitors, and also to do things  
in our software that they couldn't do. And because Lisp was so  
high-level, we wouldn't need a big development team, so our costs  
would be lower. If this were so, we could offer a better product  
for less money, and still make a profit. We would end up getting  
all the users, and our competitors would get none, and eventually  
go out of business. That was what we hoped would happen, anyway.What were the results of this experiment? Somewhat surprisingly,  
it worked. We eventually had many competitors, on the order of  
twenty to thirty of them, but none of their software could compete  
with ours. We had a wysiwyg online store builder that ran on the  
server and yet felt like a desktop application. Our competitors  
had cgi scripts. And we were always far ahead of them in features.  
Sometimes, in desperation, competitors would try to introduce  
features that we didn't have. But with Lisp our development cycle  
was so fast that we could sometimes duplicate a new feature within  
a day or two of a competitor announcing it in a press release. By  
the time journalists covering the press release got round to calling  
us, we would have the new feature too.It must have seemed to our competitors that we had some kind of  
secret weapon-- that we were decoding their Enigma traffic or  
something. In fact we did have a secret weapon, but it was simpler  
than they realized. No one was leaking news of their features to  
us. We were just able to develop software faster than anyone  
thought possible.When I was about nine I happened to get hold of a copy of The Day  
of the Jackal, by Frederick Forsyth. The main character is an  
assassin who is hired to kill the president of France. The assassin  
has to get past the police to get up to an apartment that overlooks  
the president's route. He walks right by them, dressed up as an  
old man on crutches, and they never suspect him.Our secret weapon was similar. We wrote our software in a weird  
AI language, with a bizarre syntax full of parentheses. For years  
it had annoyed me to hear Lisp described that way. But now it  
worked to our advantage. In business, there is nothing more valuable  
than a technical advantage your competitors don't understand. In  
business, as in war, surprise is worth as much as force.And so, I'm a little embarrassed to say, I never said anything  
publicly about Lisp while we were working on Viaweb. We never  
mentioned it to the press, and if you searched for Lisp on our Web  
site, all you'd find were the titles of two books in my bio. This  
was no accident. A startup should give its competitors as little  
information as possible. If they didn't know what language our  
software was written in, or didn't care, I wanted to keep it that  
way.[2]The people who understood our technology best were the customers.  
They didn't care what language Viaweb was written in either, but  
they noticed that it worked really well. It let them build great  
looking online stores literally in minutes. And so, by word of  
mouth mostly, we got more and more users. By the end of 1996 we  
had about 70 stores online. At the end of 1997 we had 500. Six  
months later, when Yahoo bought us, we had 1070 users. Today, as  
Yahoo Store, this software continues to dominate its market. It's  
one of the more profitable pieces of Yahoo, and the stores built  
with it are the foundation of Yahoo Shopping. I left Yahoo in  
1999, so I don't know exactly how many users they have now, but  
the last I heard there were about 20,000.  
The Blub ParadoxWhat's so great about Lisp? And if Lisp is so great, why doesn't  
everyone use it? These sound like rhetorical questions, but actually  
they have straightforward answers. Lisp is so great not because  
of some magic quality visible only to devotees, but because it is  
simply the most powerful language available. And the reason everyone  
doesn't use it is that programming languages are not merely  
technologies, but habits of mind as well, and nothing changes  
slower. Of course, both these answers need explaining.I'll begin with a shockingly controversial statement: programming  
languages vary in power.Few would dispute, at least, that high level languages are more  
powerful than machine language. Most programmers today would agree  
that you do not, ordinarily, want to program in machine language.  
Instead, you should program in a high-level language, and have a  
compiler translate it into machine language for you. This idea is  
even built into the hardware now: since the 1980s, instruction sets  
have been designed for compilers rather than human programmers.Everyone knows it's a mistake to write your whole program by hand  
in machine language. What's less often understood is that there  
is a more general principle here: that if you have a choice of  
several languages, it is, all other things being equal, a mistake  
to program in anything but the most powerful one. [3]There are many exceptions to this rule. If you're writing a program  
that has to work very closely with a program written in a certain  
language, it might be a good idea to write the new program in the  
same language. If you're writing a program that only has to do  
something very simple, like number crunching or bit manipulation,  
you may as well use a less abstract language, especially since it  
may be slightly faster. And if you're writing a short, throwaway  
program, you may be better off just using whatever language has  
the best library functions for the task. But in general, for  
application software, you want to be using the most powerful  
(reasonably efficient) language you can get, and using anything  
else is a mistake, of exactly the same kind, though possibly in a  
lesser degree, as programming in machine language.You can see that machine language is very low level. But, at least  
as a kind of social convention, high-level languages are often all  
treated as equivalent. They're not. Technically the term "high-level  
language" doesn't mean anything very definite. There's no dividing  
line with machine languages on one side and all the high-level  
languages on the other. Languages fall along a continuum [4] of  
abstractness, from the most powerful all the way down to machine  
languages, which themselves vary in power.Consider Cobol. Cobol is a high-level language, in the sense that  
it gets compiled into machine language. Would anyone seriously  
argue that Cobol is equivalent in power to, say, Python? It's  
probably closer to machine language than Python.Or how about Perl 4? Between Perl 4 and Perl 5, lexical closures  
got added to the language. Most Perl hackers would agree that Perl  
5 is more powerful than Perl 4. But once you've admitted that,  
you've admitted that one high level language can be more powerful  
than another. And it follows inexorably that, except in special  
cases, you ought to use the most powerful you can get.This idea is rarely followed to its conclusion, though. After a  
certain age, programmers rarely switch languages voluntarily.  
Whatever language people happen to be used to, they tend to consider  
just good enough.Programmers get very attached to their favorite languages, and I  
don't want to hurt anyone's feelings, so to explain this point I'm  
going to use a hypothetical language called Blub. Blub falls right  
in the middle of the abstractness continuum. It is not the most  
powerful language, but it is more powerful than Cobol or machine  
language.And in fact, our hypothetical Blub programmer wouldn't use either  
of them. Of course he wouldn't program in machine language. That's  
what compilers are for. And as for Cobol, he doesn't know how  
anyone can get anything done with it. It doesn't even have x (Blub  
feature of your choice).As long as our hypothetical Blub programmer is looking down the  
power continuum, he knows he's looking down. Languages less powerful  
than Blub are obviously less powerful, because they're missing some  
feature he's used to. But when our hypothetical Blub programmer  
looks in the other direction, up the power continuum, he doesn't  
realize he's looking up. What he sees are merely weird languages.  
He probably considers them about equivalent in power to Blub, but  
with all this other hairy stuff thrown in as well. Blub is good  
enough for him, because he thinks in Blub.When we switch to the point of view of a programmer using any of  
the languages higher up the power continuum, however, we find that  
he in turn looks down upon Blub. How can you get anything done in  
Blub? It doesn't even have y.By induction, the only programmers in a position to see all the  
differences in power between the various languages are those who  
understand the most powerful one. (This is probably what Eric  
Raymond meant about Lisp making you a better programmer.) You can't  
trust the opinions of the others, because of the Blub paradox:  
they're satisfied with whatever language they happen to use, because  
it dictates the way they think about programs.I know this from my own experience, as a high school kid writing  
programs in Basic. That language didn't even support recursion.  
It's hard to imagine writing programs without using recursion, but  
I didn't miss it at the time. I thought in Basic. And I was a  
whiz at it. Master of all I surveyed.The five languages that Eric Raymond recommends to hackers fall at  
various points on the power continuum. Where they fall relative  
to one another is a sensitive topic. What I will say is that I  
think Lisp is at the top. And to support this claim I'll tell you  
about one of the things I find missing when I look at the other  
four languages. How can you get anything done in them, I think,  
without macros? [5]Many languages have something called a macro. But Lisp macros are  
unique. And believe it or not, what they do is related to the  
parentheses. The designers of Lisp didn't put all those parentheses  
in the language just to be different. To the Blub programmer, Lisp  
code looks weird. But those parentheses are there for a reason.  
They are the outward evidence of a fundamental difference between  
Lisp and other languages.Lisp code is made out of Lisp data objects. And not in the trivial  
sense that the source files contain characters, and strings are  
one of the data types supported by the language. Lisp code, after  
it's read by the parser, is made of data structures that you can  
traverse.If you understand how compilers work, what's really going on is  
not so much that Lisp has a strange syntax as that Lisp has no  
syntax. You write programs in the parse trees that get generated  
within the compiler when other languages are parsed. But these  
parse trees are fully accessible to your programs. You can write  
programs that manipulate them. In Lisp, these programs are called  
macros. They are programs that write programs.Programs that write programs? When would you ever want to do that?  
Not very often, if you think in Cobol. All the time, if you think  
in Lisp. It would be convenient here if I could give an example  
of a powerful macro, and say there! how about that? But if I did,  
it would just look like gibberish to someone who didn't know Lisp;  
there isn't room here to explain everything you'd need to know to  
understand what it meant. In   
Ansi Common Lisp I tried to move  
things along as fast as I could, and even so I didn't get to macros  
until page 160.But I think I can give a kind of argument that might be convincing.  
The source code of the Viaweb editor was probably about 20-25%  
macros. Macros are harder to write than ordinary Lisp functions,  
and it's considered to be bad style to use them when they're not  
necessary. So every macro in that code is there because it has to  
be. What that means is that at least 20-25% of the code in this  
program is doing things that you can't easily do in any other  
language. However skeptical the Blub programmer might be about my  
claims for the mysterious powers of Lisp, this ought to make him  
curious. We weren't writing this code for our own amusement. We  
were a tiny startup, programming as hard as we could in order to  
put technical barriers between us and our competitors.A suspicious person might begin to wonder if there was some  
correlation here. A big chunk of our code was doing things that  
are very hard to do in other languages. The resulting software  
did things our competitors' software couldn't do. Maybe there was  
some kind of connection. I encourage you to follow that thread.  
There may be more to that old man hobbling along on his crutches  
than meets the eye.Aikido for StartupsBut I don't expect to convince anyone   
(over 25)   
to go out and learn  
Lisp. The purpose of this article is not to change anyone's mind,  
but to reassure people already interested in using Lisp-- people  
who know that Lisp is a powerful language, but worry because it  
isn't widely used. In a competitive situation, that's an advantage.  
Lisp's power is multiplied by the fact that your competitors don't  
get it.If you think of using Lisp in a startup, you shouldn't worry that  
it isn't widely understood. You should hope that it stays that  
way. And it's likely to. It's the nature of programming languages  
to make most people satisfied with whatever they currently use.  
Computer hardware changes so much faster than personal habits that  
programming practice is usually ten to twenty years behind the  
processor. At places like MIT they were writing programs in  
high-level languages in the early 1960s, but many companies continued  
to write code in machine language well into the 1980s. I bet a  
lot of people continued to write machine language until the processor,  
like a bartender eager to close up and go home, finally kicked them  
out by switching to a risc instruction set.Ordinarily technology changes fast. But programming languages are  
different: programming languages are not just technology, but what  
programmers think in. They're half technology and half religion.[6]  
And so the median language, meaning whatever language the median  
programmer uses, moves as slow as an iceberg. Garbage collection,  
introduced by Lisp in about 1960, is now widely considered to be  
a good thing. Runtime typing, ditto, is growing in popularity.  
Lexical closures, introduced by Lisp in the early 1970s, are now,  
just barely, on the radar screen. Macros, introduced by Lisp in the  
mid 1960s, are still terra incognita.Obviously, the median language has enormous momentum. I'm not  
proposing that you can fight this powerful force. What I'm proposing  
is exactly the opposite: that, like a practitioner of Aikido, you  
can use it against your opponents.If you work for a big company, this may not be easy. You will have  
a hard time convincing the pointy-haired boss to let you build  
things in Lisp, when he has just read in the paper that some other  
language is poised, like Ada was twenty years ago, to take over  
the world. But if you work for a startup that doesn't have  
pointy-haired bosses yet, you can, like we did, turn the Blub  
paradox to your advantage: you can use technology that your  
competitors, glued immovably to the median language, will never be  
able to match.If you ever do find yourself working for a startup, here's a handy  
tip for evaluating competitors. Read their job listings. Everything  
else on their site may be stock photos or the prose equivalent,  
but the job listings have to be specific about what they want, or  
they'll get the wrong candidates.During the years we worked on Viaweb I read a lot of job descriptions.  
A new competitor seemed to emerge out of the woodwork every month  
or so. The first thing I would do, after checking to see if they  
had a live online demo, was look at their job listings. After a  
couple years of this I could tell which companies to worry about  
and which not to. The more of an IT flavor the job descriptions  
had, the less dangerous the company was. The safest kind were the  
ones that wanted Oracle experience. You never had to worry about  
those. You were also safe if they said they wanted C++ or Java  
developers. If they wanted Perl or Python programmers, that would  
be a bit frightening-- that's starting to sound like a company  
where the technical side, at least, is run by real hackers. If I  
had ever seen a job posting looking for Lisp hackers, I would have  
been really worried.  
Notes[1] Viaweb at first had two parts: the editor, written in Lisp,  
which people used to build their sites, and the ordering system,  
written in C, which handled orders. The first version was mostly  
Lisp, because the ordering system was small. Later we added two  
more modules, an image generator written in C, and a back-office  
manager written mostly in Perl.In January 2003, Yahoo released a new version of the editor   
written in C++ and Perl. It's hard to say whether the program is no  
longer written in Lisp, though, because to translate this program  
into C++ they literally had to write a Lisp interpreter: the source  
files of all the page-generating templates are still, as far as I  
know, Lisp code. (See Greenspun's Tenth Rule.)[2] Robert Morris says that I didn't need to be secretive, because  
even if our competitors had known we were using Lisp, they wouldn't  
have understood why: "If they were that smart they'd already be  
programming in Lisp."[3] All languages are equally powerful in the sense of being Turing  
equivalent, but that's not the sense of the word programmers care  
about. (No one wants to program a Turing machine.) The kind of  
power programmers care about may not be formally definable, but  
one way to explain it would be to say that it refers to features  
you could only get in the less powerful language by writing an  
interpreter for the more powerful language in it. If language A  
has an operator for removing spaces from strings and language B  
doesn't, that probably doesn't make A more powerful, because you  
can probably write a subroutine to do it in B. But if A supports,  
say, recursion, and B doesn't, that's not likely to be something  
you can fix by writing library functions.[4] Note to nerds: or possibly a lattice, narrowing toward the top;  
it's not the shape that matters here but the idea that there is at  
least a partial order.[5] It is a bit misleading to treat macros as a separate feature.  
In practice their usefulness is greatly enhanced by other Lisp  
features like lexical closures and rest parameters.[6] As a result, comparisons of programming languages either take  
the form of religious wars or undergraduate textbooks so determinedly  
neutral that they're really works of anthropology. People who  
value their peace, or want tenure, avoid the topic. But the question  
is only half a religious one; there is something there worth  
studying, especially if you want to design new languages.