Notes from the Mystery Machine Bus

I've spent the past eight years (starting back in June 2004) writing elaborate rants about a bunch of vaguely related software engineering issues.

I was doing all that ranting because I've been genuinely perplexed by a set of "bizarre" world-views held dear by -- as far as I can tell -- about half of all programmers I encounter, whether online or in person.

Last week, after nearly a decade of hurling myself against this problem, I've finally figured it out. I know *exactly* what's been bothering me.

In today's essay I'm going to present you with a new conceptual framework for thinking about software engineering. This set of ideas I present will be**completely obvious** to you. You will probably slap yourself for not having thought of it yourself. Or you might slap the person next to you. In fact you probably *have* thought of it yourself, because it is so blindingly obvious.

But in my thirty-odd years as a programmer I'm pretty sure this way of thinking about things, if it already existed, has never been mainstream. That assertion is backed by what has to be at least ten Google searches that turned up nothing. So I'm pretty confident.

I'm going to make it mainstream, right now. Watch!

And I suspect this conceptual framework I'm giving you will immediately become, and forever remain, one of the most important tools in your toolkit for talking with -- and about -- other programmers.

The punch line, a.k.a. TL;DR

I won't keep you in suspense. Here is the thesis of this looooong essay. It is the root cause that motivated over half of my ranting all these years, starting at Amazon and continuing here at Google.

(Note: I Do Not Speak For My Employer. This should be patently obvious. When employers want someone to speak for them, they hire a person like the Mouth of Sauron, to make absolutely sure everyone knows they are speaking for the Employer.)

My thesis:

1) **Software engineering has its own political axis, ranging from conservative to liberal.**

(Note: Technically, you could stop reading right here and be at pretty much 90% comprehension. In case you care.)

2) The notions of "conservative" and "liberal" on this political axis are specialized to software engineering. But they exhibit some strong similarities to their counterparts in real-world politics.

3) *Everyone* in the software industry who does stuff related to programming computers falls somewhere fairly precise on this political spectrum, whether they realize it or not.

Put another way, YOU are either a liberal or a conservative software engineer. You may be more of a centrist, or maybe an extremist, but you fall *somewhere*on that left/right spectrum.

Just as in real-world politics, software conservatism and liberalism are radically different world views. Make no mistake: they are at odds. They have opposing value systems, priorities, core beliefs and motivations. These value systems clash at design time, at implementation time, at diagnostic time, at recovery time. They get along like green eggs and ham.

I think it is important for us to recognize and understand the conservative/liberal distinction in our industry. It probably won't help us *agree* on anything, pretty much by definition. Any particular issue only makes it onto the political axis if there is a fundamental, irreconcilable difference of opinion about it. Programmers probably won't -- or maybe even can't -- change their core value systems.

But the political-axis framework gives us a familiar set of ideas and terms for identifying areas of fundamental disagreement. This can lead to faster problem resolution. Being able to identify something quickly as a well-defined political issue means we can stop wasting time trying to convince the other side to change their minds, and instead move directly into the resolution phase, which (just as in politics) generally boils down to negotiation and compromise. Or, you know, Watergate.

Are real politics and software politics correlated?

Does being a political conservative make you more likely to be a software conservative? I think all we have are hunches and intuition at this point. We'd need studies for an answer with any scientific basis.

But my hunch tells me that they are only loosely correlated, if at all. I suspect that the four possible combinations of political left/right and software left/right each contain around a quarter of the programming population, plus or minus maybe 8%. But that's a totally nonscientific hunch, and I'd love to see the real numbers.

The reason I suspect they're only loosely correlated is that I think a person's software-political views are largely formed by two forces:

1) Their personal early/formative experiences as a programmer -- in particular, what kinds of decisions either burned them or made them wildly successful.

2) The software-political core values of their teachers, professors, mentors, role models, and coworkers.

But there are some factors that could increase correlation between political orientation and software orientation. One is geographic/regional influence -- for instance, big tech universities being located primarily in red or blue states. Another might be basic personality traits that may incline people toward traditionally liberal or conservative value systems.

Regardless of how correlated they are, there are going to be a lot of programmers out there who are mildly shocked to find that they are political conservatives but software liberals, or vice-versa.

Isn't the 2-party system a flawed model?

Well, yeah. Mapping something as nuanced as political viewpoints onto a one-dimensional axis is a well-known gross oversimplification. For instance, some U.S. voters may self-identify as fiscally conservative but socially liberal, and as a result may wind up not having a strong tie to either majority political party. These "swing votes" tend to be the reason that political moderates more often wind up winning elections, even though studies have shown that people tend to assign more credibility to extreme opinions.

The influence of centrists or moderates is just as important in the software engineering world, and we'll explore it more in a bit.

But having a single axis still allows for a wide distribution on both the right and the left. As in real politics, a moderately right-leaning viewpoint may seem dishearteningly liberal to someone who lies even further toward the extreme right. I will illustrate with some well-known real-world programming language scenarios in the examples section.

All in all, despite its oversimplifications, I think 2-party model is a good starting point for educating people about the politics of software. It'll be a good frame of reference for refining the model down the road, although that's out of scope for this essay.

So what's a Software Liberal slash Conservative?

If you ignore specific real-world issues, and just focus on the underlying traits and values of real-world conservatives and liberals, it boils down to just a few themes. I'll argue that those underlying themes are also the basis for software politics.

It's easiest to talk first about conservatives, and then define liberals in terms of what conservatives are not. This is because conservatives tend to have a unified and easily-articulated value system, whereas liberals tend to be more weakly organized and band together mostly as a reaction to conservatism. This applies to both real-world and software-world politics.

So we'll start with an operational definition of conservatism, from Jost et al.:

"We regard political conservatism as an ideological belief system that is significantly (but not completely) related to motivational concerns having to do with the psychological management of uncertainty and fear."

This theory is explored, re-tested and affirmed in a 2008 study from Garney et. al, "The Secret Lives of Liberals and Conservatives: Personality Profiles, Interaction Styles, and the Things They Leave Behind".

I hope you'll agree that this definition is minimally controversial. After all, the adjective "conservative" is more or less synonymous with caution and risk aversion. Financial conservatism is frequently (and intuitively) associated with age and with wealth. Companies tend to grow more conservative with age as they weather the storms of lawsuits, technical fiascoes, dramatic swings of public opinion, and economic downturns. We even have fables about ants and grasshoppers to teach us about conserving food for the upcoming winter.

Conservatism, at its heart, is really about risk management.

Similarly, liberal views are often associated with youth, with idealism, with naivete. In the corporate world, we think of startups as being prototypically liberal -- in part because they're setting out to change the world in some way (and liberalism is traditionally associated with change), and in part because they have to go all-out in order to hit their scheduled funding milestones, which can justify cutting corners on software safety.

Liberalism doesn't lend itself quite as conveniently to a primary root motivation. But for our purposes we can think of it as a belief system that is motivated by the desire above all else to effect change. In corporate terms, as we observed, it's about changing the world. In software terms, liberalism aims to maximize the speed of feature development, while simultaneously maximizing the*flexibility* of the systems being built, so that feature development never needs to slow down or be compromised.

To be sure, conservatives think that's what they're maximizing too. But their approach is... well, conservative. Flexibility and productivity are still motivators, but they are not the *primary* motivators. *Safety* always trumps other considerations, and *performance* also tends to rank very highly in the software-conservative's value system.

The crux of the disagreement between liberals and conservatives in the software world is this: how much focus should you put on safety? Not just compile-time type-safety, but also broader kinds of "idiot-proofing" for systems spanning more than one machine.

Let's characterize this core disagreement with some example statements that would be rated with much higher importance by conservatives than by liberals:

1. *Software should aim to be bug free before it launches.* (Banner claim: "Debugging Sucks!") Make sure your types and interfaces are all modeled, your tests are all written, and your system is fully specified before you launch. Or else be prepared for the worst!

2. *Programmers should be protected from errors.* Many language features are inherently error-prone and dangerous, and should be disallowed for all the code we write. We can get by without these features, and our code will be that much safer.

3. *Programmers have difficulty learning new syntax.* We should limit the number of languages used at our company, so that nobody has to learn a new syntax when a system goes down in the middle of the night on Christmas Eve. And we should *never* permit features that allow defining new syntax, nor changing the semantics of existing syntax. (Common examples: no operator overloading, and NO metaprogramming!)

4. *Production code must be safety-checked by a compiler.* Any code that cannot be statically checked should in general be avoided. In specific cases where it is strictly necessary, uses of it must be approved by a central committee. (Examples: eval, dynamic invocation, RTTI).

5. *Data stores must adhere to a well-defined, published schema.* Relational databases must be in third-normal form and adhere to a UML or equivalent schema definition. XML should have a DTD. NoSQL databases and name/value stores -- both of which should be avoided in general -- *must* have a separate physical schema that defines all permissible keys and their corresponding value types.

6. *Public interfaces should be rigorously modeled.* Data should never be stored in strings or untyped collections. All input and output entities should be thorougly and explicitly specified via statically-checkable, ideally object-oriented models.

7. *Production systems should never have dangerous or risky back-doors.* It should never be possible to connect to a live production system via a debugger, telnet shell, nor any other interface that allows the developer to manipulate the runtime operation of the code or data. The only ports into a production system should be read-only monitoring channels.

8. *If there is ANY doubt as to the safety of a component, it cannot be allowed in production* -- no matter how teams may cry and wail that they need it to make forward progress. (I'm talkin' to you, FUSE).

9. *Fast is better than slow.* Everyone hates slow code. Code should perform well. You should engineer all your code for optimum speed up front, right out of the box. Otherwise it might not be fast enough. Avoid using languages or DSLs or libraries that have a reputation for being slow. Even if they're fast enough for your current purposes, the

requirements (or callers) could change, and suddenly the software would be too slow!

Of course these examples are meant to be illustrative rather than comprehensive. And obviously not all of them may necessarily be espoused by everyone who self-identifies as a conservative. But barring minor exceptions, they are all very common Conservative viewpoints.

To decide whether a system or a technology is liberal or conservative, just think of a bunch of independent "safety checkboxes". If a majority of them are checked, then the system is conservative as a whole. The more that are checked, the more conservative it is.

By way of comparison, here are the Liberal versions of the nine examples above. It might help to think of Scooby-Doo characters. The examples above are all from Fred, and the examples below are from Shaggy.

1. *Bugs are not a big deal.* They happen anyway, no matter how hard you try to prevent them, and somehow life goes on. Good debuggers are awesome pieces of technology, and stepping through your code gives you insights you can't get any other way. Debugging and diagnosing are difficult arts, and every programmer should be competent with them. The Christmas Eve Outage scenario never, *ever* happens in practice -- that's what code freeze is for. **Bugs are not a big deal!** (This belief really may be the key dividing philosophy between Conservative and Liberal philosophies.)

2. *Programmers are only newbies for a little while.* The steady state for a programmer's career is being smart, knowledgeable, creative, resourceful and experienced. Putting a bunch of rules in place to protect newbies from doing harm (or being harmed) is incorrectly optimizing for the transient case instead of the steady state.

3. *Programmers figure stuff out amazingly fast when their jobs depend on it.*People learn to read sheet music, braille, sign language, and all sorts of other semiotic frameworks. Hell, even gorillas can apparently do all that. Programmers don't need *protection* from syntax. They just need documentation and a little slack time to read up on it.

4. *Succinctness is power.* Code should be kept small. Period. If your static checking tools can't reason about the code, then the checking needs to be made smarter (e.g. by incorporating runtime data) rather than making the code dumber.

5. *Rigid schemas limit flexibility and slow down development.*Lightweight/partial/optional schemas are a better tradeoff. Moreover, the schema is often not well-understood until a lot of data is collected and a lot of use cases are thorougly exercised. So the schema should follow the code rather than precede it.

6. *Public interfaces should above all else be simple, backward-compatible, and future-compatible.* Rigorous modeling is just guessing at how the interface will need to evolve. It makes both forward- and backward-compatibility almost impossible, resulting in interface churn and customer unhappiness. Public interfaces should always do the simplest thing that could possibly work, and grow only as needed.

7. *System flexibility can mean the difference between you getting the customer (or contract) vs. your competitor nabbing it instead.* Security and safety risks in runtime production systems can be mitigated and controlled by logging, monitoring and auditing. There are plenty of existence-proofs of large systems with root-access backdoors and shells (e.g. RDBMS, online game servers) whose risk is controlled while still giving them world-class runtime flexibility.

8. *Companies should take risks, embrace progress, and fiercely resist ossification.* It doesn't matter how big your

business is: it must grow or die. If you want to stay competitive, you have to make a conscious, often painful effort to take risks. Which means you'll need good recovery techniques for the inevitable disasters. *But you need those even if you don't take risks*. So take risks!

9. *Premature optimization is the root of all evil.* Get the code working first, focusing on correctness over performance, and on iterative prototyping over correctness. Only when your customers list latency as the top priority should you begin performing profiler-driven optimizations.

There you have it: Shaggy vs. Fred.

Just as in real-world politics, software liberals are viewed by conservatives as slovenly, undisciplined, naive, unprincipled, downright "bad" engineers. And liberals view conservatives as paranoid, fearmongering, self-defeating bureaucrats.

Once again, although I don't think the two camps will ever agree, I do think that mutual understanding of the underlying value systems may help the camps compromise.

Or at the very least, the conservative/liberal classification should help the two camps steer clear of each other. I think it is probably better to have a harmonious team of all-liberals or all-conservatives than a mixed team of constantly clashing ideologies. It's a lot like how vehicle-driving philosophies can differ regionally -- it's OK if everyone drives in some crazy way, as long as they ALL drive that way.

So Security Engineers are the most conservative, then, right?

Wrong! The reality here goes against intuition, and I mention it to show how poorly our intuition fares when it comes to interpolating software-political views from job descriptions.

Security engineers are deeply concerned with risk assessment and attack-surface management, so you might well guess that they would naturally tend towards software conservatism.

In practice, however, security engineers tend to be keenly aware of the tradeoffs between caution and progress, since they have to spend big parts of their day meeting with software teams who are often too busy (not to mention woefully underinformed) to spend a lot of time on security. Security engineers learn quickly to make well-informed and *practical* risk-management decisions, rather than conservatively always trying to be as safe as humanly possible.

Hence many security engineers are in fact software liberals. Some just swing that way.

Many of the security engineers at Google happen to be fans of Ruby -- both as an intrinsically secure language, and also as a nice, expressive language for writing auditing scripts and other security analysis tools. It wound up being fairly easy for me to get security sign-off for using Ruby in production for my first project at Google. In contrast, it was almost career-endingly difficult for me to get the same sign-off from our highly conservative systems programmers.

The reality is that almost any programming specialization is going to be divided into liberal and conservative camps. There are left- and right-wing versions of web developers, persistence engineers, protocol engineers, serving-system engineers, and most other sub-genres of programming.

I'm hard-pressed to think of any domains that tend to be mostly-liberal or mostly-conservative, with the sole

exception of Site Reliability Engineering, which is sort of by definition a conservative role. For most of the other domains that initially came to mind -- for instance, data analysts, which at first I thought were uniformly liberal -- I've been able to think of specific teams or subdomains composed entirely of one or the other.

So in the end I think it comes down to personal preference. That's it. I don't think it's as domain-driven as much as it is personality-driven. Some people are just liberal, and some are just conservative, and that's how they are.

BIG FAT DISCLAIMER

Before I continue, I will now issue the important disclaimer that I am a hardcore software liberal, bordering on (but not quite) being a liberal extremist.

This fact, by and large, is what has driven me to pen many of my most infamous and controversial rants, including (among others) "Execution in the Kingdom of Nouns", "Portrait of a N00b", "Rhinos and Tigers", "Code's Worst Enemy", "Scheming is Believing", "Tour de Babel", "Dynamic Languages Strike Back", "Transformation", "Haskell Researchers Announce Discovery of Industry Programmer Who Gives a Shit", and many others besides.

Sure, I sometimes write about other stuff. My in-progress "Programmers View of the Universe" series is motivated by an entirely different bugbear that has nothing at all to do with software politics. My "Universal Design Pattern" article is about a technique that transcends software politics and can be applied with equal effectiveness in either a conservative or a liberal manner. And I've written about video games, Japanese anime and random other stuff.

But I've come to understand that liberalism underlies a tremendous amount of my thinking. Even when I'm writing about management, I see that I am a liberal manager rather than a conservative one. And despite being both relatively old and relatively wealthy, I am also a political liberal -- both socially and fiscally.

Nevertheless I am going to try to represent both sides fairly and accurately in this essay. You know, mostly. I think it's most important for you to buy in to the left/right distinction itself. I think it's far less important whether you happen to agree with my particular side.

I'll consider this essay a success if a sufficient number of you agree that the liberal/conservative distinction is valid for software engineering, and that my particular left/right classification of various technologies and philosophies below seems intuitively reasonable. I'm fine with declaring success if we disagree on a few small details, as long as we agree on the overall picture.

Soooo... static typing enthusiasts are conservative, right?

Why yes. Yes, they are. Static typing is unquestionably one of the key dividing software-political issues of our time. And static typing is a hallmark of the conservative world-view.

In the conservative view, static typing (whether explicit or inferred) is taken on faith as an absolute necessity for modern software engineering. It is not something that one questions. It is a non-issue: a cornerstone of what constitutes the very definition of Acceptable Engineering Practice.

In the liberal's view, static typing is analogous to Security Theater. It exists solely to make people *feel* safe. People (and airports) have proven time and again that you can be just as statistically secure without it. But some people need it in order to feel "safe enough".

That's a pretty big difference of opinion -- one I'm sure you can relate to, regardless of how you feel about it.

I'm not going to try to defend my view here, since that's what all those old blog posts were for. If I haven't convinced you by now, then there's not much point in continuing to try. I respect your opinion. Well, *now.* And I hope you now have a little better understanding of mine.

I should, however, mention that there is an unrelated (i.e. politically-neutral) point on which both camps agree: namely, that static types yield better toolchain support. This is undeniably true today, and I have made it my life's work to ensure that it is not true tomorrow.

I have spent the last four years championing an initiative within Google called the "Grok Project", one that will at some point burst beyond our big walled garden and into your world. The project's sole purpose in life is to bring toolchain feature parity to all languages, all clients, all build systems, and all platforms.

(Some technical details follow; feel free to skip to the next section heading...)

My project is accomplishing this lofty and almost insanely ambitious goal through the (A) normative, language-neutral, cross-language definitions of, and (B) subsequent standardization of, several distinct parts of the toolchain: (I) compiler and interpreter Intermediate Representations and metadata, (II) editor-client-to-server protocols, (III) source code indexing, analysis and query languages, and (IV) fine-grained dependency specifications at the level of build systems, source files, and code symbols.

OK, that's not the *whole* picture. But it's well over half of it.

Grok is not what you would call a "small" project. I will be working on it for quite some time to come. The project has gone through several distinct lifecycle phases in its four years, from "VC funding" to "acceptance" to "cautious enthusiasm" to "OMG all these internal and even external projects now depend critically on us." Our team has recently doubled in size, from six engineers to twelve. Every year -- every quarter -- we gain momentum, and our code index grows richer.

Grok is not a confidential project. But we have not yet talked openly about it, not much, not yet, because we don't want people to get over-excited prematurely. There is a lot of work and a lot of dogfooding left to do before we can start thinking about the process for opening it up.

For purposes of this essay, I'll assert that at some point in the next decade or so, static types will not be a prerequisite for world-class toolchain support.

I think people will still argue heatedly about type *systems*, and conservatives may never be able to agree amongst themselves as to which type system approach is best for modeling program behavior. But at least I will have helped that discussion be ONLY about representations. The quality of the corresponding developer tools should no longer be a factor in the discussions.

Dividing up the Space

I'm going to go through and toss a bunch of random technologies, patterns, designs and disciplines each into one of six buckets: "apolitical", "conservative", "centrist", "liberal" buckets, plus two buckets that start Centrist and head Left or Right in the presence of overuse. One bucket per thingy. Hey, it's not an exact science. But it should help set the rough foundation for the more detailed use cases below.

Non-political Stuff: Algorithms, data structures, concrete mathematics, complexity analysis, information theory, type

theory, computation theory, and so on. Basically all CS theory. These disciplines occasionally inspire tempest-in-a-teapot butthurtedness in academia, but when it happens, it's just similar fish in too small a tank biting on each other. It's to be expected. Overall, these essentially mathematical disciplines are timeless, and they are all equally applicable to both the Liberal and Conservative programming worlds. Yes, even type theory.

Conservative Stuff: Provably sound type systems. Mandatory static type annotations. Nonpublic symbol visibility modifiers (private/protected/friend/etc.). Strict, comprehensive schemas. all-warnings-are-errors. Generics and templates. Avoidance of DSLs (XPath, regexps) in favor of explicit DOM manipulation and hand-rolled state machines. Build dependency restrictions. Forced API deprecation and retirement. No type equivalence (i.e. no automatic conversions) for numeric types. Checked exceptions. Single-pass compilers. Software Transactional Memory. Type-based function overloading. Explicit configuration in preference to convention. Pure-functional data structures. Any kind of programming with the word "Calculus" in it.

Centrist (or flat-out Neutral) Stuff: Unit testing. Documentation. Lambdas. Threads. Actors. Callbacks. Exceptions. Continuations and CPS. Byte-compilation. Just-in-time compilation. Expression-only languages (no statements). Multimethods. Declarative data structures. Literal syntax for data structures. Type dispatch.

Liberal Stuff: Eval. Metaprogramming. Dynamic scoping. all-errors-are-warnings. Reflection and dynamic invocation. RTTI. The C preprocessor. Lisp macros. Domain-specific languages (for the most part). Optional parameters. Extensible syntax. Downcasting. Auto-casting. reinterpret_cast. Automatic stringification. Automatic type conversions across dissimilar types. Nil/null as an overloaded semantic value (empty list, empty string, value-not-present). Debuggers. Bit fields. Implicit conversion operators (e.g. Scala's implicits). Sixty-pass compilers. Whole-namespace imports. Thread-local variables. Value dispatch. Arity-based function overloading. Mixed-type collections. API compatibility modes. Advice and AOP. Convention in preference to explicit configuration.

Centrist Stuff that Becomes Conservative If Taken Far Enough: Type modeling. Relational modeling. Object modeling. Interface modeling. Functional (i.e., side-effect-free) programming.

Centrist Stuff that Becomes Liberal if Taken Far Enough: Dynamic class loading and dynamic code loading. Virtual method dispatch. Buffer-oriented programming.

Woah, that exercise was surprisingly fun! It's far from complete, but hopefully you get the idea.

Some natural themes arise here:

-- *implicit* is generally liberal; *explicit* is generally conservative.

-- *performance-oriented* is generally conservative; *late-optimized* is generally liberal.

-- *compile-time binding* is generally conservative; *runtime/late binding* is generally liberal.

-- concurrency and parallelism in general seem to be politically charged topics, but the disagreement is orthogonal to the liberal/conservative camps.

I'd love to keep going with the classification. But I'll stop here, since we've got higher-level stuff to discuss.

Examples and Case Studies

I'll walk you through a bunch of examples to show you how just widespread and deep-rooted this political

phenomenon is.

Example 1: Languages

Here are some very rough categorizations. Note that within each language camp there are typically liberal and conservative sub-camps. But as a whole, language usage tends to be dominated by what the language makes possible (and easy), so the culture tends to follow the features.

This list is just a few representative examples to give you the flavor. I'm only listing general-purpose languages here, since DSLs and query languages are typically feature-restricted enough to be hard to categorize.

Assembly language: Batshit liberal.

Perl, Ruby, PHP, shell-script: Extremist liberal.

JavaScript, Visual Basic, Lua: Hardcore liberal.

Python, Common Lisp, Smalltalk/Squeak: Liberal.

C, Objective-C, Scheme: Moderate-liberal.

C++, Java, C#, D, Go: Moderate-conservative.

Clojure, Erlang, Pascal: Conservative.

Scala, Ada, OCaml, Eiffel: Hardcore conservative.

Haskell, SML: Extremist conservative.

These are my own categorizations based on my own personal experiences with these languages and their respective communities. Your mileage may vary. However, I'd be quite surprised if you chose to move any of these languages more than a step or two away from where I've positioned it.

One thing that jumps out is that a language doesn't have to be statically-typed or even strongly-typed in order to be conservative overall. More on that in a bit.

The next thing you might notice from the list is that the liberal and moderate languages are all pretty popular, and that popularity declines sharply as languages head into conservative territory.

I think this has a simple explanation: It's possible to write in a liberal language with a conservative accent, but it's very hard (and worse, discouraged) to write in a conservative language with a liberal accent.

For instance, it's straightforward to write JavaScript code in a way that eschews reflection, eval, most automatic type casting, prototype inheritance, and other dynamic features. You can write JavaScript that plods along as unadventurously as, say, Pascal. It doesn't have all the static type annotations, but you can replace them with assertions and unit tests and stereotypically stolid code organization.

But if you try writing your Haskell code with a bunch of dynamic features, well, you're in for a LOT of work. Haskell enthusiasts have managed to implement dynamic code loading and a ton of other ostensibly dynamic features, but

it was only through herculean effort.

What's more, if you write your liberal-language code in a conservative way, people will just look at it and say: "Well, it's kinda boring, and you could have saved a lot of coding by using some dynamic features. But I guess it gets the job done. LGTM."

Whereas if you write your conservative-language code in a liberal way, you run the risk of being ostracized by your local language community, because... why are you doing all that dangerous dynamic stuff in the first place? I'll explore this cultural phenomenon further when I talk about Clojure below.

The last big, interesting observation from the list is that a lot of the most popular languages out there are only moderately conservative -- even if they think of themselves as *quite* conservative compared to their ultra-dynamic cousins.

I've said it before, and it bears repeating here: the reason C++, C# and Java have been *particularly* successful in the marketplace is that -- just like effective politicians -- they know how to play both sides.

C++ allows liberal-biased programmers to program in straight C, and it allows conservative-biased programmers to layer in arbitrary amounts of static type modeling, depending on how much work they want to expend in order to feel secure. Java? Pretty much the same story.

Playing to both the fast-and-loose and lock-your-doors mindsets has proven to be a key ingredient to market success. Also marketing, but it helps a LOT to be viewed as philosophically friendly by both the liberal and conservative camps.

There is a new crop of languages on the horizon (for instance, Google's Dart language, but also new specs for EcmaScript) that are deliberately courting the centrist crowd -- and also delicately playing to grab both the liberals and conservatives -- by offering *optional* static types. In principle this is a sound idea. In practice I think it will come down to whether the marketing is any good. Which it probably won't be.

Language designers always seem to underestimate the importance of marketing!

Example 2: Tech corporations

Just for fun, let's contrast four similar-ish tech companies in their software-political outlook.

1) **Facebook** -- *Diagnosis: Extremist Liberal.* Despite their scale, they are still acting like a startup, and so far they've been getting away with it. They use primarily C++ and PHP, and they're prone to bragging about how their code calls back and forth from PHP to C++ and back into PHP, presumably bottoming out somewhere. Their datastore is memcached: just name-value pairs. No schema. They dump the data and logs into a backend Hive store and run Hadoop mapreduces for offline data analysis. They still hold all-night hackathons every other week or so, which will remain feasible for them as long as the majority of their programmers are very young males (as was the case last time I toured there) and their stock continues to promise great riches (as was not so much the case last I checked.) As a company they are tightly knit and strongly biased for action, placing a high value on the ability of individual programmers to launch features to their website with little to no bureaucracy or overhead. This is pretty remarkable for a company as big as they are, with as many users as they have. Conservatives no doubt regard them with something between horror and contempt. But Facebook is proving that programmers of*any* world-view can get a hell of a lot accomplished when they gang up on a problem.

2) **Amazon.com** -- *Diagnosis: Liberal.* Which is surprising, given how long they've been in business, how much money is at stake, how mature their Operations division is, and how financially conservative they are. But "Liberal" is actually quite a retreat compared to their early days. Back in 1998-1999 they were almost exactly like Facebook is today, with the sole exception that they put everything in relational databases and did a ton of up-front relational data modeling. Well, except in Customer Service Apps, where we used a name/value store just to be flexible enough to keep up with the mad chaotic scramble of the business launches. All part of my multi-decade indoctrination as a Liberal. In any case, despite many corporate improvements with respect to work-life balance (which happened after several stock plunges and years of significant double-digit turnover in engineering), Amazon has retained its liberal, startup-like engineering core values. Every team owns their own data and makes their own decisions, more or less like independent business units. Amazon still launches and executes faster than just about anyone else out there, because they're still willing to take real risks (incurring occasional huge outages), and to make hard decisions in favor of launching early and often. Above all else, Amazon has proven conclusively that after fifteen years, they can still innovate like nobody else. They've still got it.

3) **Google** -- *Diagnosis: Conservative.* They began life as slightly liberal and have grown more conservative ever since. Google was only software-liberal in the very very early days, back when the search engine itself was written in Python. As they grew, they quickly acquired a software conservatism driven entirely by the engineers themselves. Manifestos were written about the dangers of using multiple languages, and strict style guides were put in place to severely limit "risky" or "hard to read" language features of the few languages they did allow. Google's JavaScript code is written in an extremely conservative style with extensive static type annotations, and eval is forbidden. The Python style guide forbids metaprogramming and other dynamic features, which makes their Python look a lot like untyped Java. And they have severely limited the use of many C++ language features, with C++11 support rolling out literally one feature every few weeks. (There are over five hundred new features in C++11.) In internal surveys, Google engineers commonly cite bureaucracy, churn and complexity as core obstacles to feature advancement and rapid launches. Google has made serious attempts on several occasions to reduce this bureacracy, but they always get pushback from -- surprise -- the engineers themselves, who have grown so staunchly conservative that they actively (and even more often, passively) resist the introduction of more flexible stacks and technologies. Most of the major technological shifts within Google over the past half-decade have been overtly conservative. For a liberal like me, it has been a very sad process to observe. But at least I've found myself a niche that's widely regarded (by both camps) as valuable, and within my own org we can still be pretty liberal and get away with it.

4) **Microsoft** -- *Diagnosis: Batshit Conservative.* Microsoft has two geese that lay golden eggs: Office and Windows. Microsoft has been reduced to a commercial farmer protecting the geese from all incursions. The golden eggs still have value, because customers are locked into the platform by the cost-ineffectiveness of retraining their fleets. But Microsoft can no longer innovate in Office or Windows precisely because of those corporate fleet retraining costs. Their OEMs are stretched as thin as they can go. Apple is dominating the handheld markets, and Microsoft is actively stifling their own innovation in Windows Phone because they're afraid it will cannibalize their core Windows business. Microsoft has not had a successful product-level *innovation* in fifteen, maybe twenty years. All of their successful products have been copies of competitors' products: IE, XBox, C#, .NET, Bing, Windows Phone, and so on ad infinitum. All great implementations of someone else's ideas. Microsoft's playbook is to embrace, extend, and leverage their brand to crush the competition -- or at least it was, until the goverment put an end to that circa 2002. Now the company genuinely doesn't know what the fuck to do with themselves, and what's more, instead of Bill Gates they now have a lunatic in charge. Employees are leaving in droves, all citing the same internal "existential crisis" and unbearable corporate politics caused by competing business units actively sabotaging one another. Microsoft has turned into a caricature of right-wing corporatism: sitting on their front porch with a shotgun cursing at passers-by, waiting for their government bribes to give them another few years of subsidies and shelters while they wait to die. I've personally chatted with close to four hundred current and ex-Microsoft employees over the past seven years. Oh, the stories I could tell you... someday, maybe.

5) Bonus company: **Apple**. Diagnosis: no idea, but they're so good at marketing that it's almost irrelevant. Would love to have more insight into their internal software culture, though. Any takers? Throwaway accounts? AMA?

OK, *that* was a fun exercise too. But we need to move on! Almost done now.

Specific Case Study: The Clojure Language

I've been meaning to follow up on Clojure for a while now. Over a year, at least. But until recently I didn't have the conceptual tools to explain what I wanted to say about it.

Now I do!

Clojure is a new-ish Lisp dialect that runs on the JVM and .NET, and I was honored to write the Foreward to "The Joy of Clojure" a while back. For a few years I had been really excited to start learning Clojure, and my initial experiences with it were quite positive.

However, I eventually learned that the Clojure community is extremely conservative. That is is pretty unusual for a Lisp dialect. Lisp is widely regarded as one of the most liberal language families in existence. And Clojure has the superficial appearance of being a laissez-faire kind of language. It is quite expressive, including a -- ahem -- *liberal* dose of new syntax. And it eschews static type annotations and strong type modeling in favor of a small set of highly regular, composable data types and operations -- not unlike, say, Scheme or Python.

But the resemblance to a liberal language ends there. Clojure's community came pre-populated with highly conservative programmers from the pure-functional world: basically Haskell/ML types (lots of puns today!) who happen to recognize the benefits of Lisp's tree syntax. So under its expressive covers, everything about Clojure is *strongly* conservative, with a core overriding bias towards protecting programmers from mistakes.

And the community follows suit. At a Clojure conference last year (or was it two years ago? time flies so fast these days...), there was a key presenter doing a talk on how Macros were basically harmful and should be avoided in modern Clojure code.

I trust that if you know anything about Lisp, your blood is basically boiling at this point. I know mine was.

But his argument is perfectly valid from the classic software-conservative's viewpoint. Macros allow you to invent domain-specific language abstractions. Those require documentation in order for users to figure out what they do, and what they mean. That means you can know Clojure and not *really* know enough to read someone's Clojure code without some documentation handy. Which is sort of the definition of a newbie. And there you have it. In a very real sense, conservatives fear being turned -- in the blink of an eye -- from masters into newbies by the application of macros.

It's sort of scary if you have a lot of your personal identity invested in knowing some shit. And wouldn't you know it, real-world politics conservatives are shown in study after study to be "staunch" in sticking to their world views rather than compromising. That means they have a lot of identity tied up in those views.

So while I liked a lot of what I saw in Clojure, as a hardcore software liberal I was inevitably and inexorably driven away from the language. And that's good for me, and it's good for Clojure. I mean, why should they compromise?

I think that my conceptual framework gives us an "out" -- a way to avoid being emotional about these subjects. Casting the problem as a clash between Liberalism and Conservsatism gives us the ultimate ticket for "agreeing to

disagree".

Hopefully it will also help language designers and communities do a better job of targeted marketing. Right now just about every language out there makes a claim along the lines of "This language is the best choice for *everybody*!" But now we know that is very unlikely to be true -- or if it is, we can at least be assured that they're centrists, and they run the risk of being equally distasteful to everybody.

In the conservative/liberal framework, language designers can make more accurate, less bait-and-switchy claims; for instance: "Haskell is the best choice for *every* radical extremist conservative programmer!"

Well, we can work on the wording. But you get the idea.

Wrap-Up

I was thinking of going through a bunch more examples and stuff, but I see that I'm on my third *(Editor's Note: fourth)* glass of wine, which means my typing is about to give out any minute.

So let's wrap it up!

There's one kinda key point I wanted to get across, but didn't see a good place for it. That point is this: please do not be alarmed that I am calling you a (software) Conservative.

I worry that politically left-leaning programmers will hear the term "conservative" and will immediately associate it with all of the... uh, politically-charged connotations associated with far right-wing conservatism in the United States political arena today. You know, racism, sexism, religious fundamentalism, homophobia, warmongering, bear-shooting, that kind of thing.

I'm not saying they're *bad*, at least not in this essay. I'm just saying nobody in their right mind wants to be associated even remotely with those embarrassing wingnuts. See what fine-grained, nuanced distinctions three *(Editor's Note: four)*glasses of wine can produce? But I'm not saying their views are *bad*. No. Not here. I'm just observing that they're heavily politically charged viewpoints which have, for better or worse, recently come to be associated with the term "conservatism" in US politics.

So please do me a favor and try to dissociate those specific agendas and real-world political viewpoints from the generic term "Conservative", which here really just means "risk averse".

It's perfectly OK, and normal, to be a programming conservative. You don't have to shoot any bears. I would actually like to see the terms "liberal" and "conservative" to become badges of honor in the programming world. People should stand behind their beliefs. I mean, we do already, I think, so it shouldn't be much of a stretch to allow our beliefs to be given convenient labels.

Ah, me. I can see the Euphemism Treadmill rearing its ugly head already. We'll see.

Anyway, tell me what you think! I welcome any and all viewpoints and comments. Even from bears!

Special thanks to Writer's Block Syrah for reviewing this post for career suicide.

(Special Note to my fellow Googlers: Yes, I meant to post this externally. BOTH times. No, I am not the Mouth of Sauron.)