



I'd like to start ... with a digression.



One of my favourite pieces of poetry was delivered, not by a beat poet in Greenwich Village or by a 19th Century Romantic,



but by one Donald Rumsfeld, then Secretary of Defense, from the Pentagon press briefing room, on February 12, 2002.



Hart Seeley later formed the Secretary's words into a poem which was published in Harper's Magazine in June 2003 as "The Unknown" As we know, There are known knowns. There are things we know we know. We also know There are known unknowns. That is to say We know there are some things We do not know. But there are also unknown unknowns, The ones we don't know We don't know.



I've also found the same sentiments expressed less elegantly, but more forcefully in diagram form, showing: The stuff we know we know; The stuff we know we don't know; And the vast expanse of things we don't know we don't know.

It is scary that the largest category by far is one we definitionally cannot comprehend stuff we don't know we don't know.



Of course, this is an epistemological diagram of *all* knowledge, so we *can* constrain it, a bit, by noting that, for practical purposes, we are really only concerned with the **stuff we *should* know**. Unfortunately the stuff we *should* know still falls in all three categories.

And as IT managers, there is, amongst the stuff we should know, the stuff we should know about open source software



So, why do I think I have something useful to say on this topic? My bona fides to talk about open source and management are as a developer of open source software, a hacker, a programmer, in particular on the PostGIS spatial database, which I have been involved in for 10 years and as a manager of a small consulting company, Refractions Research, which I ran for 10 years.



My goal today is to modify your internal epistemological diagrams a little, maybe to something more like this.



And not because I expect you all to rush back home and implement open source, but because when you know more about open source, it becomes an OPTION. who what where when why how

To understand why open source is an option, it helps to have some background. So I am going to begin with a quick rundown the journalistic who, what where when and why.



And fortunately, I can squeeze the Who, What, Where and When into a preliminary story, and then we can deal with Why and How at our leisure afterwards.



So, the story...



Once upon a time, there was a young man with wild ideas about freedom, who took on the established order of things, appeared to lose, but in the end changed the world forever (though perhaps in ways he might not approve of).



Actually, not that young man, though there is a striking resemblance...

In 1980, Richard Stallman was a programmer



at the MIT Artificial Intelligence lab. Some of the best minds in the

the best minds in AI

AI field worked together

sharing ideas and code

and shared ideas and implementations of those ideas in code.

golden age of hacker collaboration

It was, to hear Stallman tell it, a brief golden age of collaboration and intellectual ferment.



Then one day, and don't all horror stories start this way, one day, the lab got a new printer (a xerox 9700). Unlike the printer it was replacing, the new printer came with a binary-only printer driver; the source code was not included.



Stallman had modified the previous driver to send a message to users when the printer jammed.



With the new binary driver he couldn't do that. The situation was *inconvenient*, it was a pain. Why couldn't Xerox just share their code? Everyone would be happier!



Most people might have shrugged. But for Stallman it was a galvanizing moment. Over the past five years working in the AI lab, he had grown used to sharing code and ideas with other programmers.

But now the atmosphere in computing was changing.

symbolics
symbolics.com registered in 1985

It wasn't just the printer driver.

A private corporation had started recruiting his colleagues in the lab. Once hired, they were no longer allowed to exchange code with him. (Completely unrelated nerd trivia, but, the company doing the hiring, symbolics, was also the first company to register a .com domain name, in 1985.)



The old hardware in the lab, and by extension the software that ran on it, was becoming obsolete. The new systems being purchased by the lab included operating systems that were locked down: you had to sign nondisclosures just to use them. "the first step in using a computer was to promise **not to help** your neighbor"

It was the death of the old collaborative community. Stallman worried that "the first step in using a computer was to promise *not* to help your neighbour" by accepting a license agreement. "was there a program or programs that I could write, so as to make a **community possible** once again?"

As a highly talented and idealistic computer programmer, Stallman wanted his work to serve a larger purpose. The financial promise of working in the growing proprietary software industry was not enough, nor was the sterile intellectual amusement of continuing his work alone in academia.

Facing the death of his old intellectual community, Stallman asked himself "was there a program or programs that I could write, so as to make a *community possible* once again?"



free By "free", Stallman meant you should be free to run it, you should be free to modify it, you should be free to share it,

to share your modifications

to run it

to modify it

to share it

You can't use a computer without an operating system. So Stallman decided that he needed to write an operating system. It had to be portable to many computer platforms, it should be compatible with the popular new UNIX operating system to make it easy for people to run their existing programs on it, and most importantly it should be **free**.

you should be free to share your modifications



That is, you should have freedom, in how you use your software.

In a latinate language like French, Spanish or Italian it's more obvious, Stallman isn't talking about **logiciel gratuis**, he's talking about [fr] logiciel libre, [sp] software libre, [it] il software libero.

He's talking about liberated software: the key addition is liberty.



These core freedoms, to modify and share software, also make up the "open source definition", which was popularized in 2000 at the end of the dotcom boom.

GNU's Not UNIX

So, rather than join a computing industry that he considered morally bankrupt, Stallman decided to basically start a new one from scratch. It was an audacious plan.

Stallman called his new system GNU, which stands (recursively) for "GNU's Not UNIX?" (See the recursion?)

GNU's 'S Not UNIX NOT UNIX UNIX

'S GNU's Not UNIX 's Not UNIX Not UNIX



Let me just take a very minor diversion here to add some extra flavor.



In order to ensure GNU remained free, and did not get subsumed into a proprietary system in the future, Stallman released his work using a scheme he called "copyleft".

©	public domain
all rights reserved	no rights retained

Generally speaking, intellectual works (books, movies, songs, computer programs) are either under copyright or public domain. The author either retains full control over the work, "all rights are reserved", or no control, "no rights are retained".



Copyleft, and open source licenses in general, use the copyright system to selectively grant permission and exert control over software through *licensing*.

The copyleft license grants permission to all recipients of the code to use, modify and redistribute the work in any way they wish,

GNU GENERAL PUBLIC LICENSE Version 2, June 1991 Copyright (C) 1989, 1991 Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed. Preamble The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software -- to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Lesser General Public License instead.) You can apply it to your programs, too. When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

with one exception.

The license requires that any redistribution of the work or derived products include the source code, and be subject to the same license.



The legal language can get complex, but the principle is hardly foreign. Share and share alike. Do unto others as you would have them do unto you. Or else.



OK, back on the highway.



So, in 1984 Stallman quits his job at MIT and starts working on GNU full time. No visible means of support, this is a labor of love.

But where to start? From a blank canvas, you want a completely free software ecosystem, what do you do first?

If you wanted to build a 100% all hand crafted house, you would start by hand crafting your tools. Stallman did the *same* thing, with GNU versions of software development tools.



He starts by writing a text editor (GNU Emacs), so he can write his free system using free tools.

The Emacs editor proves so popular (and internet access is still so rare) that he is able to earn a small living selling tape copies of the code (distributed under copyleft of course).



Then he writes a compiler, GCC, so he can build the code into executables. You can still find GCC in every Linux and also in Mac OSX.



Stallman lives like a monk, works like a demon, attracts some followers and helpers, who formalize the project in a foundation.

By 1990 they have most of the components of an operating system.



Most importantly, they have a full programming toolchain: shells, compilers, debuggers, editors, core libraries, and so on. All the things you need to write complex software.



What they don't have, is a UNIX kernel, the piece of software that talks directly to the hardware.

At this point, all their free tools are still being run on proprietary UNIX!





In 1991, a Finnish computer science student named Linus Torvalds buys a new computer, an Intel 386.

As a student at the university, he has access to UNIX systems, and he wants to run UNIX on his 386 at home.



This is not possible.

The good implementations for the 386 cost more than the computer itself. The cheap implementation, Minix, is quite limited.

So Linus writes his own kernel. He uses Stallman's GNU tools to write and compile it. And in August of 1991 he posts the following on an internet discussion list. Hello everybody out there using minix -

I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) among other things).

I've currently ported bash(1.08) and

does anyone want to play?

Underneath the technical language, note the subtextual bits: the humility (just a hobby...); the interest in other people's ideas (what do you like/ dislike in minix...).

The posting is an invitation.

Does anyone else want to come out and play? Does anyone? They do.

"Tell us more! Does it need a MMU? How much of it is in C?"

Within 15 minutes, he has a reply. "Tell us more! Does it need a MMU? (memory management unit) How much of it is in C?"



Within 24 hours, he has replies from Finland, Austria, Maryland, and England.

In a month the code is on a public FTP server. Within four months, it is so popular that an FAQ document has been written to handle the common questions.

Linus Torvalds tapped a seam of enthusiasm just dying to express itself.



People who loved computers and computing and just wanted to play together. And through the medium of the internet, using only the simplest computing tools (diff, patch, ftp, e-mail), he built a community of thousands of contributors, and



together they built a usable operating system.



Something important changed between the time Stallman started the GNU project and Torvalds released Linux. The values of collaboration were the same, but the opportunity to exercise those values was greater, via the internet.

When Stallman started GNU in 1984, there were 1000 hosts on the internet. When Torvalds started Linux in 1991, there were over 400,000. The pool of potential collaborators was in the middle of a huge expansion.



Let's take a quick detour and talk about Star Wars.



In particular, let's look at a web site called Star Wars uncut.

Star Wars Uncut has taken the original movie and chopped it into 473 fifteen second scenes. Each scene is then separately claimed and re-enacted by site members, and uploaded.



The result looks like this.



Seems pretty frivolous, right, but break it down. How is this (frivolous) collaboration possible? And, why is it only happening now, not 10 years ago? There were just as many Star Wars nerds 10 years ago as there are now.



First, this activity requires easy access to video recording and editing tools, and until recently cameras and video editors were very expensive. And it requires enough bandwidth to download and upload video, and until recently people didn't have that kind of bandwidth in their homes. And finally it requires Star Wars geeks.



Generalizing, then... To build a large collaborative product, you need tools freely (or very cheaply) available and you need sufficient connectivity between participants. Combine that basic infrastructure, with community, collaboration and love for the subject matter, and magic happens.





There are many, many more examples of this kind of group collaboration, the academics call it "commonsbased peer production". If you are interested in the topic I strongly suggest reading "Cognitive Surplus" or "Here Comes Everybody" by Clay Shirky.



So this is less of a detour than it seemed at the start,



Because, Open source software in general and the Linux project in particular are some of the earliest examples of internet-mediated commons-based peer production.



The universal access to tools was provided by the original GNU project components. Editor, compiler, lexer, libraries, were all available equally to all participants.



The medium of communication was just e-mail. The work they were sending around was source code, snippets of text, no problem even for the dial-up internet connections of the early 1990s.



And why do open source programmers do it? What is the core motivation. It isn't money. Fundamentally they code because they love it.



It's the same reason: Star Wars geeks re-shoot 35 year old films; why food geeks post restaurant reviews; why car geeks re-build '68 Cameros. It's an avocation.

At least, it starts that way. But open source software has a wider utility than restaurant reviews and vintage muscle cars. So as projects have expanded, they have at each stage become more and more integrated into the wider economy.


Linux is a good example. Start with Linus and the early group of enthusiasts in 1991. These are individuals working in their spare time. They are doing it for love.

By 1992, you get "distributors", packaging up the Linux kernel with collections of GNU and other tools to form full working operating systems. First they do it for love, helping other Linux lovers, but soon they are covering their cost and time, selling CDROMs for 550. So programmers are earning livings with small Linux businesses within a couple years of the project start.

And then the corporate world starts to see the value in a new, free operating system.

In 1994, DEC sends Linus a "free" Alpha workstation in the hopes he will port Linux to the Alpha chip. He does. Simultaneously David Miller ports Linux to the Sun Sparc processor. Linux is now competing with "real" UNIX on corporate "big iron". Over the next couples years, the makers of these machines start to hire Linux programmers of their own.

So three years in, major corporations are seeing the competitive advantage in Linux, and start hiring experts in it.

In 1995, Red Hat Linux is formed, a company which will eventually grow to an \$8B Linux support enterprise.

In 1996, Los Alamos National Laboratory builds the first Linux cluster for simulating atomic shock waves. It costs 10% of a comparable supercomputer, and on start-up becomes the 315th most powerful supercomputer in the world.

By 1998, the explosion of the internet into general public use is underpinned by thousands of commodity servers running Linux as their operating system.



Microsoft is drafting strategy memos about how to counter Linux, and Linus Torvalds is featured on the front page of Forbes magazine.

Linux is no longer a hobbyist activity. It is deeply embedded in the economy at multiple levels. This is in 1998, just *seven years* after that first newsgroup post.



Fast forward to the present. The NSA employs Linux programmers to make their systems secure. NASA employs Linux programmers to run it on their space mission hardware. Google employs Linux programmers to optimize their massive compute clusters. Oracle employs Linux programmers to support their Oracle-optimized Linux. IBM employs Linux programers to ensure it runs on their SystemZ mainframes. Microsoft employs Linux programmers to add kernel support for Windows virtualization. And so on, and so on, and so on,



So, here's a question I get asked a lot: um, how do you make a living writing free software?



Referring back to the previous slide... Hopefully it would be obvious.



I make my living the same way my dentist, barber and plumber make their livings. I sell my very specialized expert services to people who need them. And in a globalized, internet connected world, there are plenty of people who need them.



I could talk for another half hour about the different ways open source projects are deriving support from the general economy, but unfortunately, then, I wouldn't have time to talk about why you, as managers, should be looking at open source as an option.



We have covered the Who (Richard Stallman, Linus Torvalds and thousands of others) What (Freedom to redistribute and modify) Where (Mostly on the internet) and When (From 1985 initially, but really starting from the early 1990s, growing rapidly right up to the present) of open source.



Now let's take a look at the "Why". And not the "why" of Stallman (the moral imperative to share), but the "why" of an information technology manager.

Here are five good reasons to consider open source in your enterprise

Cloud Readiness also known as Scaling also known as Rapid Deployment; License Liability or actually the Lack Of same; Flexibility and its kissing cousin Heterogeneity; Staff Development and Recruitment; and most importantly Market Power.



So, first of all Technical Superiority

Did I forget to mention this one?

• fewer bugs

- more modularity
- better security
- faster release cycles
- better performance

There are open source advocates who will claim, straight up, no hedging, that open source software is just technically superior to proprietary software. They will say that the open development model results in code with fewer bugs per 1000 lines, higher levels of modularity, better security due to wider peer review and faster release cycles, and better performance.



I am not one of those advocates.



At least not for most projects. For a project like Linux, perhaps. Linux has concentrated an incredibly large number of very high quality technical contributors

into one code base, more people than any one company could ever employ.

But most open source projects, and certainly those in the geospatial realm, operate with at most a few dozen contributors, they aren't out of the league of corporate development teams.



(Although I'm pretty sure there are more people working on PostGIS right now than are working on SQL Server Spatial.



Which is weird.)



David Wheeler

dwheeler.com/ oss_fs_why.html

If you are interested in the topic of technical superiority, David Wheeler has a 2007 paper "Why Open Source Software: Look at the Numbers!" that brings together all the research in one, very, very long page, which is well worth reading. **#1**: cloud readiness scaling rapid deployment

Moving on, Reason #1, cloud readiness, also known as scaling, also known as rapid deployment. It looks like I'm squeezing three topics into one, but I'm not. These three benefits are all aspects of the same open source attribute:



the zero-dollar capital cost of deployment.



Last year, your keynote speaker was Peter Batty, and he warned you of the technical trends ahead. More and more computing tasks will be delegated "clouds" of computers hosted in huge data centers "somewhere" on the internet. More users will expect direct access to data through web services.



More mobile devices will consume those services with every passing year. All that new user demand adds up to potentially unconstrained load on services, and growth curves that transition very quickly from horizontal to vertical, as services move into the mainstream.





One of the things I have noticed about users of geospatial open source over the past years, is that the most enthusiastic adopters have been start-up companies.



GlobeXplorer based their satellite image service on the PostGIS open source database, choosing it over Oracle and Informix.



Zonar Systems developed a fleet tracking system on top of PostGIS and the MapServer web mapping software.



RedFin started their real estate information site on MySQL and moved it to PostGIS for performance reasons.



The Google Analytics for the PostGIS site show that California is the state with highest interest and



inside California it is San Francisco and Silicon Valley that have the highest interest.



The reason start-ups love open source is because it removes a category of potential constraints on their growth. The cost of computing hardware falls dramatically year over year. The cost of proprietary software does not follow the same curve.



If you are using software licensed per CPU or core, that means that software cost is a primary driver of scaling cost.



The math can be brutal even before you start scaling horizontally.

This Dell T710 with dual quad-core CPUs, 36Gb of memory and 2TB of RAID 10 storage will set you back \$6,953.



OK, now let's put Oracle Enterprise on our fancy new server.

We have 8 cores, multiply that by a 0.5 "processor core factor", times the per processor price of Oracle Enterprise, add in Spatial because we are GIS folks (and remember, you need Enterprise to run Spatial) and the grand total is, a cool \$260,000.



Or as Larry Ellison calls it, a quarter.



Just contemplate the numbers for moment. Hmm.

The exact same unpleasant math applies to GIS map serving, and it gets worse and worse the more you scale up.

	proprietary	open source
server	\$5,000	\$5,000
software	\$30,000	\$0
training	\$0	\$30,000
total	\$35,000	\$35,000

Let's compare scaling an open source and a proprietary map service.

At initial roll-out, the load is small, so we buy one server for \$5,000, and one copy of the software for \$30,000. To be fair (or perhaps unfair) let's assume the staff is already fully trained in the proprietary software, but requires an immense amount of expensive training or learning time to adopt the open source. So there's our sub-total for the first server, \$35,000.

	proprietary	open source
server	\$5,000	\$5,000
software	\$30,000	\$0
training	\$0	\$30,000
3x server	\$15,000	\$15,000
3x software	\$90,000	\$0
training	\$0	\$0
total	\$140,000	\$50,000

Now great news, the citizens love the map service, maybe someone built a cool iPhone app around it, and suddenly the load on the machine quadruples. What does it cost? Add three more servers. Add three more licenses. We don't need more training, the software is the same. And he more you scale, the worse the totals on the left become.



Now, it is possible you are already so highly evolved that you run your public services in the cloud. So there are no capital costs for "servers". But the math in the cloud remains just as unpleasant:



ouch! per-instance proprietary software licensing dwarfs the per-instance hardware cost. The only difference is that the hardware costs are spread out more evenly over time instead of being concentrated in big capital-intensive bursts.



The final reason start-ups love open source is that they don't have to ask permission to fire up those new servers, so they can respond to crises and new opportunities very very quickly. Any software that requires a license or a license manager can potentially slow a deployment by days. If you are suddenly enjoying a surge in traffic, the last thing you want to offer your new customers is a slow customer experience because you haven't been able to deploy enough servers. **#2**: license liability (lack of)

So, I used to run a consulting company, and at our peak we had 30 staff. A small company.



But that still meant 30 workstations under 30 desks, running 30 licensed copies of Windows and 30 licensed copies of Microsoft Office.

At least, that was the theory.

In practice the company had grown very quickly over two years and software, particular application software, had been installed wherever it was needed whenever it was needed. So when we finally got around to counting up the difference between what we were using and what we owned, it was a bit shocking.



We had 10 licenses for Windows and 5 for Office.

Coming into compliance would cost almost \$20,000. Not coming into compliance would risk hundreds of thousands of dollars in fines.

We were one disgruntled employee away from a big cash crunch.

- Developers used Java development tools
- BAs used word processing and spreadsheets
- Everyone used e-mail and web browsing

So we examined what we wanted our software to do. Our developers needed Java development environments, our BAs needed document processing, our managers needed some word processing, everyone needed e-mail.



So we switched to open source. Everyone got OpenOffice for word processing. Email and web browsing was with Firefox and Thunderbird. Some developers switched to Linux as their operating system. And we bought enough extra Windows licenses to fill the desktop operating system gap.

It was all surprisingly easy. Two things to keep in mind

First, if we had been more disciplined about using open source in the first place, we wouldn't have built up the liability problem we did. On the server side we had always been a pure Linux and open source shop, so we never built up a problem.

Second, once we *got* the open source discipline, our potential *future* liability problems were reduced. There were just a *lot fewer licenses* remaining to keep track of.

ok, smart guy, what about GIS software?

So we replaced our office automation side without much trouble, what about the GIS side?



Some tools were too specialized and ingrained in our workflow to be replaced. We simply worked to manage our license load. We put our FME licenses on a shared system with remote desktop, for example.



Other things had just gotten out of hand...

ArcView 3 is just so darned easy to copy. Isn't it?

How many of you have copies of ArcView 3 floating around your offices or... sssshhh, your homes?

If I listen very carefully, I can hear a license compliance managers teeth grinding somewhere in the back.



Our story ended with removing all the unlicensed ArcView copies, since they weren't being used, and with those who needed a viewer/editor using the JUMP project.

That was several years ago. If it were happening today, we might use gvSIG or uDig, but probably the most complete ArcView replacement available right now is QGIS.



I've started to see a growing number of requests for proposals that specify QGIS in what would formerly have been the ArcView 3 role: a data entry platform with some custom tools in a field situation.



Here's a few screenshots of QGIS, it looks eerily familiar doesn't it? Simple UI, basic scripting language, simple printing capability. It fills a need.





It fills a need.



But the core point here is *not* that proprietary software is replaceable (though it is), it is that proprietary software adds a layer of legal liability that needs to be managed. And that takes time and effort.

Because software gets copied. A lot.



Why wouldn't it, you make a perfect copy with two keystrokes, it gets copied. Ctrl-C. Ctrl-V.

And if that software is proprietary, each of those keystrokes digs a compliance hole for the organization. Click, click, click, deeper and deeper and deeper. And you don't realize how deep that hole is, until you fall into it.



This is a bit of a geeky argument, but bear with me



 flexible components are easy to connect and adapt

First, flexible components are easy to connect to each other and to adapt. You can use flexible components from multiple vendors to build a heterogeneous system. "heterogeneity" is an attribute of the system
heterogeneous systems use parts from many sources

A heterogeneous system incorporates components from multiple sources.

Flexibility is great, but usually you have to trade some ease-of-use to get get it.



Which toolbox would you rather work with? The hextool: convenient, easy, fits in the palm of your hand, three sizes. Or The socket set: modular, extendable, 64 sizes, metric and imperial.

One's easy, one's flexible.



Here is a practical example of the value of flexibility and heterogeneity.

The British Columbia government built their web mapping infrastructure using ArcIMS for their internal web servers and web applications, and using MapServer for their external WMS services.

Both web mapping servers pull their data from a central ArcSDE instance. So they have a flexible tool (in MapServer) and a heterogeneous infrastructure (using both ArcIMS and MapServer).

A few years ago, the infrastructure team applied a minor, minor, teeny weeeny, sliver of a service patch to the Oracle database that hosted ArcSDE. To their surprise, the minor patch shut locked up SDE. Which meant their web services (that depended on SDE) were also down.



The WMS services were brought back up in three days, after a long process of loading the raw data into a temporary PostGIS database. Because MapServer could read from PostGIS just as easily as ArcSDE, this was no problem.



The ArcIMS services remained offline for the duration of the outage, which was 28 long days until a patch to ArcSDE was made available.



As a general proposition, proprietary product lines talk well to other systems from the same vendor, and less well to systems from other vendors. Competitive advantage dictates this arrangement, but, it puts the interests of the customer in interoperability below the interests of the vendor in promoting lock-in.



As a general proposition, open source products talk well to all other systems.

The reason why is less obvious, but it has to do with the practical motivations of the developers.



Once a project moves past the "for fun" stage, the developers are working on it because it is a workplace tool, they need it to "do something". And the "something" they need it to do, is usually in the context of other software.



So, as a developer, if you like MapServer's GML support, but you work in an environment where most of the data resides in ArcSDE, a reasonable thing to do is write code to connect the two.



Each of these practical interconnections increases the overall value of the project, bringing in more developers in, who bring in their own unique interconnection requirements.



An example of the end state of this process is the Minnesota MapServer supported format list, which is one of the most extensive in the industry. MapServer started out in 2000 with just one format: shape files.



Short digression



This is the boreal forest around Prince George, British Columbia, where I grew up. In the mature forest, out of the creek valleys, over 80% of the trees are pine and spruce.



In the late 1990s an infestation of the mountain pine beetle



began in Wells Grey Park in north western British Columbia.



The local infestation turned into an epidemic over the next few years.



The epidemic has been "uncontrolled" for a decade now and is only forecast to abate by the middle of this decade,


when the population of mature lodgepole pine has been completely digested.









Here's a graph of the number of hectares affected over time. The pine beetle has been so ..."successful" for lack of a better word, not just because climate change has reduced the number of cold winters that kill beetle populations, but also because of the good luck in finding a huge homogeneous area of mature boreal forest ready to consume, the product of 50 years of successful forest fire fighting.



Just a little digression on the digression...



Computer worms are pieces of code that self replicate, kind of like beetles. They start from a host, scan for other vulnerable hosts, then copy their children to the new host, where the process continues.





If you're a network administrator, you might recognize this log entry, the signature of an attempted Code Red infection: You can actually still see them in logs from time to time, very rarely now, like marooned Japanese soldiers who don't know the war is over.



OK, let's get back on the road.

homogeneous systems and single-vendor strategies can be convenient

Homogenous systems, and single-vendor strategies, are usually convenient. But there is a trade-off.

but, lack flexibility for fast adaptation

They lack flexibility, which can make it hard to adapt them to unexpected purposes.



And, they present reliability risk, an increased vulnerability to population catastrophes, issues that are capable of shutting down your whole infrastructure in one go. **#4**: staff development and recruiting

One of the most gratifying things I have heard over my career of teaching about open source GIS software is this:



"that talk you gave last year totally changed my life".



Saying this about a software talk.







The exhilaration of learning what was in that box, and the freedom to use that knowledge to make cool things, without external constraints (like licenses) on what they could make, was deeply empowering for them.

These are very special people, they are the kind of people you want to hire.



I recently came across a diagram which explains it all in one page.

Take the personality traits of intelligence, obsession, and social ineptitude.

People with intelligence and obsession are geeks. Inept smart ones are dweebs, and the inept obsessives are dorks. Those with all three traits, in the middle, are the nerds.



As GIS managers, building out new systems and pushing the envelope, you probably want smart folks with a mapping technology obsession, geogeeks ideally, but you can settle for geo-nerds.

So, how do you get those geeks and nerds to work for you? Offer something *interesting*. Remember, they are technology obsessives.



Paul Graham is a Silicon valley entrepreneur and a major league nerd, who tells this story about building an e-commerce engine he eventually sold to Yahoo! in the late 1990s.



For personal technical reasons, they wrote their engine in LISP, which was a rare choice, since most mainstream use of LISP had disappeared by the late 1990s.



But, using LISP had an odd side-effect, which was that when they advertised job openings, they got these amazing resumes, rock-star candidates, and when they interviewed them, they all mentioned their interest in LISP.



By the 1990s, LISP was really mostly used in academic settings, but also retained a prominent role as a customization language in... wait for it... Emacs, Richard Stallman's text editor for uber-programmers.

So the uber-nerd programmers who obsessed over Emacs LISP macros were intrigued by the chance to do web development in LISP.



OK, you're not going to build your web sites in LISP. I am not recommending that.



But you might build them in Python. Or Ruby. You might run them on Linux. You might serve them with MapServer or GeoServer. You might store your data in PostGIS and PostgreSQL. You might build your web pages with JavaScript and OpenLayers and GeoExt.



This does work in the real world.



The city of Northglenn, Colorado wrote a report about their experience with open source, and they cited some of the motivations I've already talked about, but in the section on "Unobvious Motivations for Adoption" there is this quote: "Contrast an open-source implementation position with a 'defined skill set' job..."

"...where the first diagnostic action is to reboot the server..."

"...and the second is to call the vendor and wait in a telephone hold queue..." "It is easy to understand why open-source jobs are prized."

City of Northglenn, CO



Finally, Market Power.

I chose not to give a deeply technical talk today, so I haven't really run through the panoply of open source GIS software that is available to you. Let me just quickly do that for effect.



For databases, you have PostGIS, MySQL, Ingres and SpatiaLite.



For map and feature servers, you have GeoServer, MapServer, Mapnik, TinyOWS, SharpMap, and others.



For tile caching you have TileCache, GeoWebCache, TileStache, and others.



For web interfaces, you have OpenLayers and OpenScales, GeoExt, Polymaps and others.



On the desktop you have gvSIG, uDig, QGIS, OpenJUMP, MapWindow, and others.



Underneath it all are libraries like GEOS, GDAL, OGR, Proj4, JTS, and GeoTools, which can be leveraged with scripting languages like Python, Perl, Ruby, Groovy, ASP.Net and others.



Sounds complex, yes?

"open source offers too many choices"

I give a talk, which started five years ago as a 20 minute talk and has now expanded into a 90 minute marathon, where I cover all these options in detail, and afterwards exhausted people come up to me and say, "open

source offers too many choices",

"it's easier with just one vendor"

"it's easier with just one vendor".



Which is odd, because we deal with lots of choice in all the other markets we navigate every day.

Theres lots of kinds of cars, lots of kinds of jeans, lots of kinds of coffee.



And we have a good idea of what a market with just one vendor looks like. We actually have laws against it.



Proprietary software has a dirty little secret, and it is a secret that lives in plain sight.

Even in otherwise competitive markets, the effect of proprietary licensing is to create an instant de facto monopoly.



How many companies provide support for your proprietary software: one. How many companies provide upgrades: one.

How many companies provide upgrades. one.

ORACLE is suing SAP for competing on support
ORACLE [®] profit margin on support: 800%

Proprietary companies guard their aftermarket monopoly zealously.

Oracle is currently suing SAP's TomorrowNow division for the crime of selling Oracle support to SAP customers.

And there's good reason Oracle is suing: the profit margin on Oracle support is 800%.



It is all about market power. Open source vests the market power in the software user, not the vendor.

As a manager, you probably don't care about tinkering with the internals of your software source code, but you SHOULD care about holding on to your market power as a customer.



Bob Young, the founder of Red Hat Linux, asks this question of customers:



Would you buy a car with its hood welded shut?



No, right? So ask the follow-up question:

"What do you know about modern internal combustion engines?"



What do you know about modern internalcombustion engines?



And the answer for most of us is "not much".

"We demand the ability to open the hood of our cars because it gives us, the consumer, control over the product we have bought, and takes it away from the vendor."

We demand the ability to open the hood of our cars because it gives us, the consumer, control over the product we've bought and takes it away from the vendor. We can take the car back to the dealer; if he does a good job, doesn't overcharge us and adds the features we need, we may keep taking it back to that dealer. But if he overcharges us, won't fix the problem we are having or refuses to install that musical horn we always wanted -- well, there are 10,000 other car-repair companies that would be happy to have our business.

this is not a good negotiating position



Making an enterprise commitment to a single vendor puts you permanently into the worst negotiating position possible.

You go into every negotiation with no alternative position, no other store to storm off to.

The only leverage you have left is the threat to buy nothing at all. Which isn't much of a threat.



Speaking of market power, does anyone else see the resemblance between these images?







OK, I can't bring that segue back on topic. Not a good detour perhaps.



To maintain market power, to provide our staff with new growth opportunities, to build heterogenous systems, to lower license liability, to be ready to scale, for all those reasons it makes sense to have open source as an option.



But how to start?



- 2007, 100% ESRI
- limited budget
- talented staff
- interest in new ideas

Way back in 2007, Pierce County, in Washington State, was a 100% ESRI and Microsoft shop, with a limited budget for new software acquisition.

It did have some talented technical staff, and a GIS manager, Linda Gerull, who was interested in new ideas.



In the fall of that year, she learned that the international open source GIS conference was being held just to the north in Victoria, and took the opportunity to send several of her staff.

"Keep an eye out for alternatives", she told them.



When they came back, they had lots and lots of alternatives, they were very excited.

But they couldn't just tear down their infrastructure and start again, they had to maintain service continuity.



The team started experimenting by duplicating some existing services that were built using old MapObjects technology and slated to be replaced anyways.

Some of them were very simple services with minimal user interface, like the Critical Areas query form.





800	Critical Areas			
↓ + Attp://yakima.co.pierce.wa.	us/CriticalAreas/AreaAnalysis.cfm?ParceINum=6175000	0 0	Q. Google	
Checklist for Tax Parcel 6175000081 D			Date: 10/04/2010	
About the Parcel:				
Site Address	1011 3RD AV NW		Tell Me More	
Range, Township, Section	Range 04E Township 20N Section 28		Tell Me More	
Jurisdiction	Puyallup		Tel Me More	
School District	SD #003 PUYALLUP		Tell Me More	
Fire District	FPD #006 CENTRAL PIERCE		Tell Me More	
Community Plan Area	No		Tell Me More	
Current Zoning	Puyallup		Tel Me More	
Urban Growth Area	Yes		Tell Me More	
About the Parcel's Natural Envir	ronment:			
Aquifer Recharge Area	Yes		Tel Me More	
Development Moratorium	No		Tel Me More	
Marine Shoreline Salmon Habitat	No		Tell Me More	
Potential for Erosion Hazard	No		Tell Me More	
Potential for Fish and Wildlife	No		Tel Me More	
Potential for Flood	No		Tell Me More	
Potential for Floodway	No		Tel Me More	
Potential for Landslide	No		Tell Me More	
Potential for Mine Hazard	No		Tell Me More	
Potential for Seismic Hazard	Yes		Tell Me. More	
Potential for Wetlands	No		Tell Me More	
Resource Lands	No		Tell Me More	
Charallan	No		Test Mrs. Mines	

MapObjects was unstable, ArcIMS was too slow, but open source (PostGIS in this case) was just right.The form didn't change at all, just the backend.

As their confidence in the tools grew, they looked at migrating core bits of infrastructure.



Most recently, they replaced their SQL Server database with PostgreSQL and PostGIS.

The key here is that they are continuing to run ArcSDE on top. This allows them to use their existing data management tools, like ArcGIS,



but use a pure open source web services stack directly against PostGIS.

So the changes are incremental, and exploratory.



Pierce County still runs ESRI software and ArcGIS desktops, but the number of options they have for deploying new systems is much higher, and the number of licenses they require is going down, not up. Budget flexibility is increasing.



At the same time, the staff has enjoyed learning the new technology.

This is the conclusion slide from a presentation Pierce County's Jared Erickson, gave at the Washington State GIS conference this spring.

Open source and ESRI can work together. Open source provides a diverse range of options.

FOSS4G 2011 Denver, Colorado September 12-16

(For Pierce County an important catalyst was the proximity of the FOSS4G conference in 2007, so I would be remiss if I failed to note that the next FOSS4G conference will be in Denver in September of 2011. It has not been in North America since 2007, so take the opportunity to go or send staff. It won't be back again until 2014.)


Pierce County experimented with a limited number of open source components: PostGIS, PostgreSQL, GeoServer, OpenLayers.



But as we saw earlier, there are a lot of choices.

So how should you choose open source components?



David Wheeler

dwheeler.com/ oss_fs_eval.html

David Wheeler, who I mentioned previously in the context of open source versus proprietary, also has a very complete document on "How to Evaluate Open Source Software" like COTS evaluation but some differences

Much of the evaluation is the same as with proprietary COTS (commercial off-the-shelf software), but some key things are different.



First, there will not be sales reps or sales material.



You may have to download and test the software yourself.



Hopefully this will give you a reality-based understanding of capabilities, rather than a marketing-based understanding. beware feature check-lists open source may have fewer features... but adding features is an option!

Second, you have to beware of feature check-list comparisons. Open source often has fewer features than proprietary software, but is also has a far faster and cheaper pipeline for getting features added.



For example, if you want to get a feature added to Oracle Spatial, you have to (a) become the Department of Defence and (b) lean on Larry Ellison while yachting.



If you want to get a feature added to PostGIS, you just have to (a) give me 2-10K and (b) wait 2-6 weeks.



Adding a required feature to an open source project should be an easy decision process, but it's just not one we're used to.

This feature is mission critical. Oracle has it. PostGIS doesn't. It'll cost \$500K to deploy Oracle. It'll cost \$50K to get this feature added to PostGIS. Ergo. We should choose _____



Amazingly, this decision is still not a slam dunk for open source, probably because our brains just are not quite used to thinking this way yet.



Open source is the collision of geeky love of computing with a political philosophy of freedom with a global community enabled by the internet. open source is a practical **option** for organizations

Open source provides organizations with a new software OPTION when building systems.



Organizations should consider open source for the scalability, the lack of license liability, the flexibility, the empowerment of staff, and the market power of the organization.



Start small, build a prototype, join the communities around the software you use, and always always try to have fun.



References

http://oreilly.com/catalog/opensources/book/stallman.html http://en.wikipedia.org/wiki/Copyleft http://www.linux.org/people/linus_post.html http://groups.google.com/group/comp.os.minix/browse_thread/thread/76536d1fb451ac60 http://www.oracle.com/us/corporate/pricing/technology-price-list-070617.pdf http://www.iwar.org.uk/comsec/resources/worms/cumulative-ts.gif http://gekpadshow.com/files/2009/09/nerd-venn-diagram-9420-1252236207-2.jpg http://icma.org/en/icma/knowledge_network/documents/kn/Document/301435/ Software_Selection_in_an_OpenSourceOriented_Government_IT_Department http://www.waurisa.org/conferences/2010/Agenda.html http://www.waurisa.org/conferences/2010/presentations/ 316_Jared_Erickson_Open_Source_GIS_at_Pierce_County.pdf http://yakima.co.pierce.wa.us/CriticalAreas/AreaAnalysis.cfm? ParcelNum=6175000081&CFID=196536&CFTOKEN=25584369