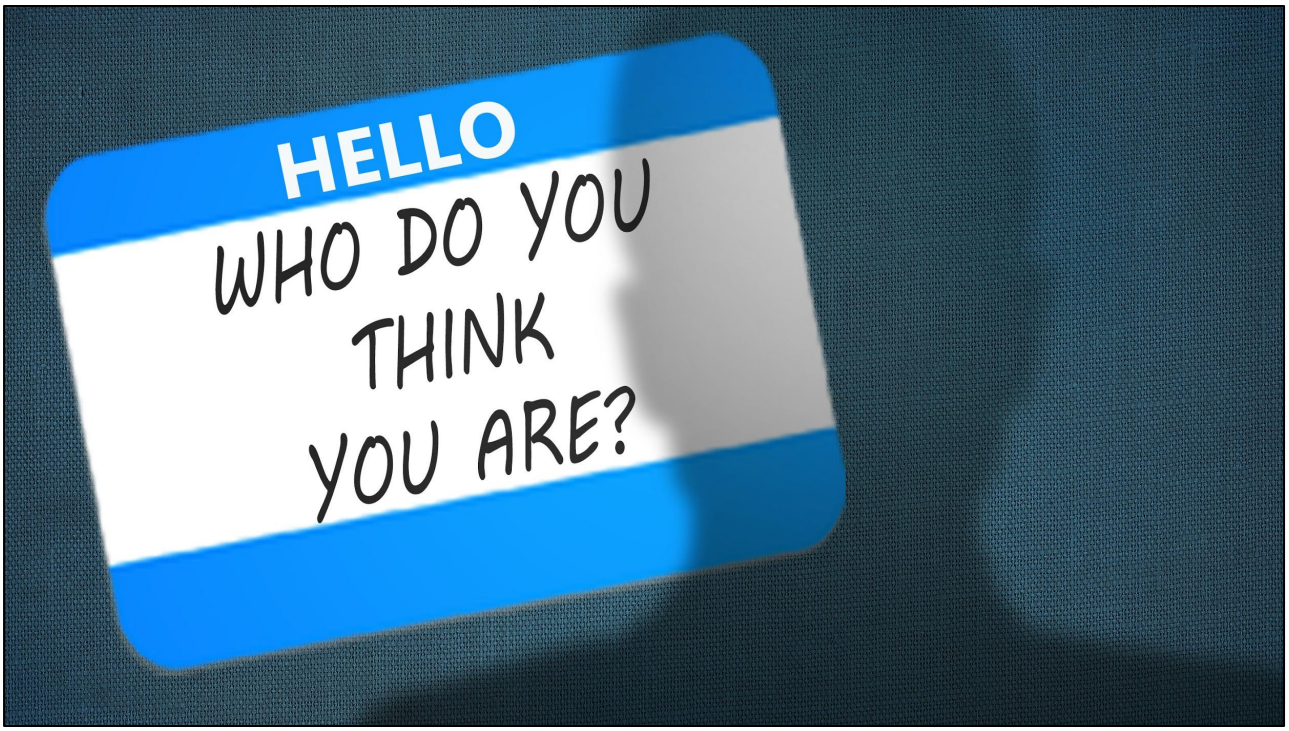
So, I've given this talk the title "Let's get small", in honor of Steve Martin.
I've always loved the early comedy of Steve Martin, the juxtaposition of raw absurdity and his precise use of language, the way he can play his own straight man.
And honestly,
it's hard to talk about enterprise IT without some turn to the absurd.
How did we end up at this place? How did using computers get so complicated?

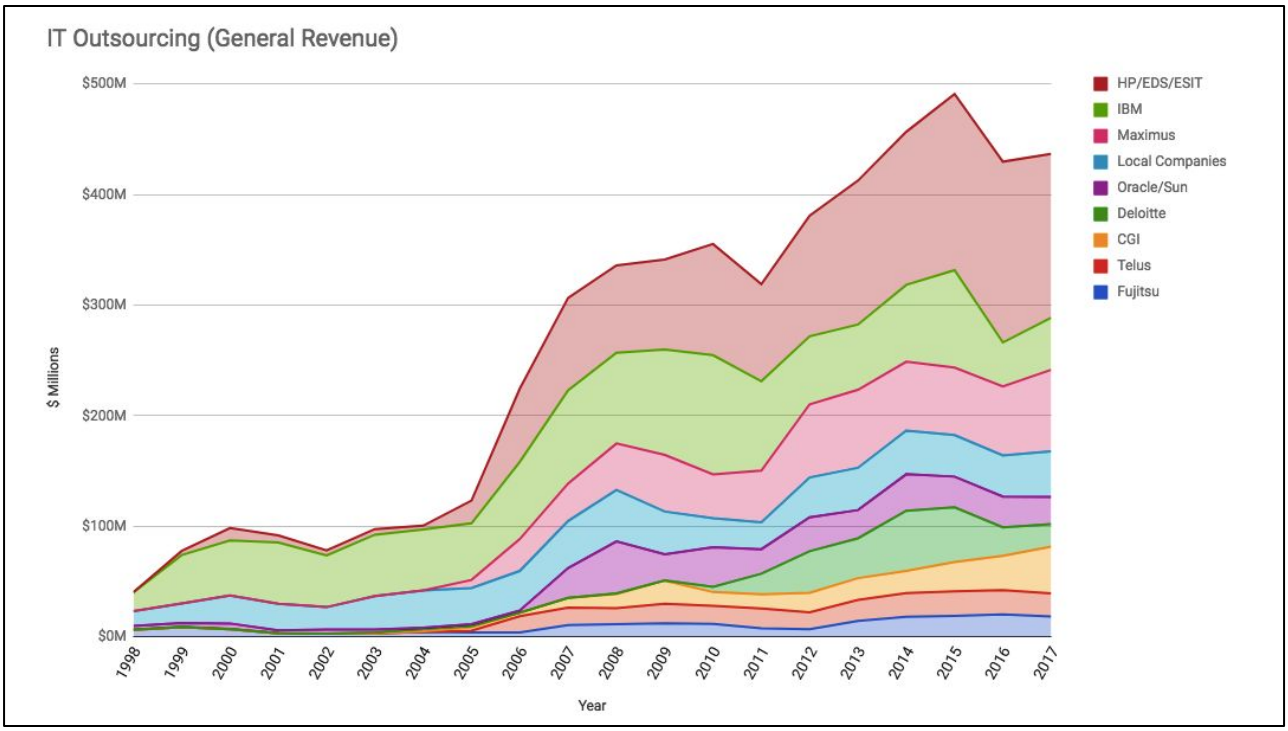


So, what standing to do I have to talk about this stuff?
From 1998 to 2008 I built a small systems integration company from myself to a staff of 30, and in that time I got to play most of the roles in the orchestra: developer, team lead, architect, project manager, the vision guy, the sales guy.
I also got to work all the sides of the procurement dance, responding to RFPs, drafting them, teaming up with other vendors from big regional to the international firms, and with software vendors large and small.

However, my consulting career in BC was quite a few years ago now, so if you know me at all it's either from my work on open source software, or from my writing about BC government IT over the past five years.
Maybe "writing" isn't the right word. Maybe more "primal howling".

	472,265	HUSKY OIL MARKETING COMPANY	165,812
	79,103	HYDE PROJECT MANAGEMENT SERVICES	
	44,285	LIMITED	89,036
		HYDRA HELICOPTERS INC.	149,463
	159,021	HYPERION INVESTMENTS LIMITED	604,420
	225,793	HY–RIDGE HELICOPTERS LTD.	48,376
	56,306	I.T. FALLING LTD.	255,512
	1,989,820	IACOBUCCI, FRANK	30,968
	2,765,700	IAG ENTERPRISES LTD.	187,365
	167,171	IAN H. PITFIELD INC.	34,744
	47,173	IBEX CONSULTING INC.	389,807
RIN		IBI GROUP	865,163
	67,405	IBI GROUP ARCHITECTS (CANADA) INC.	525,016
	35,966	IBM CANADA LIMITED	41,060,139
	68,135	IBOX PACKAGING LTD.	31,179
	75,943	ICE DEVELOPMENT LTD.	275,071
	92,832	ICE PEAK SOLUTIONS	26,911
	146,294	ICL PERFORMANCE PRODUCTS CANADA LTD.	4,798,797
	40,000	ICLEI LOCAL GOVERNMENTS FOR	
	132,768	SUSTAINABILITY MANAGEMENT INC.	52,172

This is the initial research that set me off, the “holy cow” moment.
I’ve read the public accounts annually since I was a consultant, it’s a way of tracking who is up and who is down in the business.
And seeing the IBM number go up and up year after year, I wondered, what does this data look like longitudinally. So I pulled every public accounts document available online and compiled the data for every identifiable IT vendor.



And this is what I saw, I think 2010 was the year I first compiled this graph.
So, starting in 1998, around $50M per year, and over a decade going up by a factor of six.
Clearly a sea change in how IT was being delivered by and to government was taking place, very quietly and with not a lot of people looking at it.
We almost got to half a billion last year, and if the Health Authorities are included it gets north of $700M.
It's a lot of money, it's big.

Major Drivers of Outsourcing Expenditure

Operational Outsourcing

- Maximus
- EDS / HPAS / ESIT
- IBM
- Telus

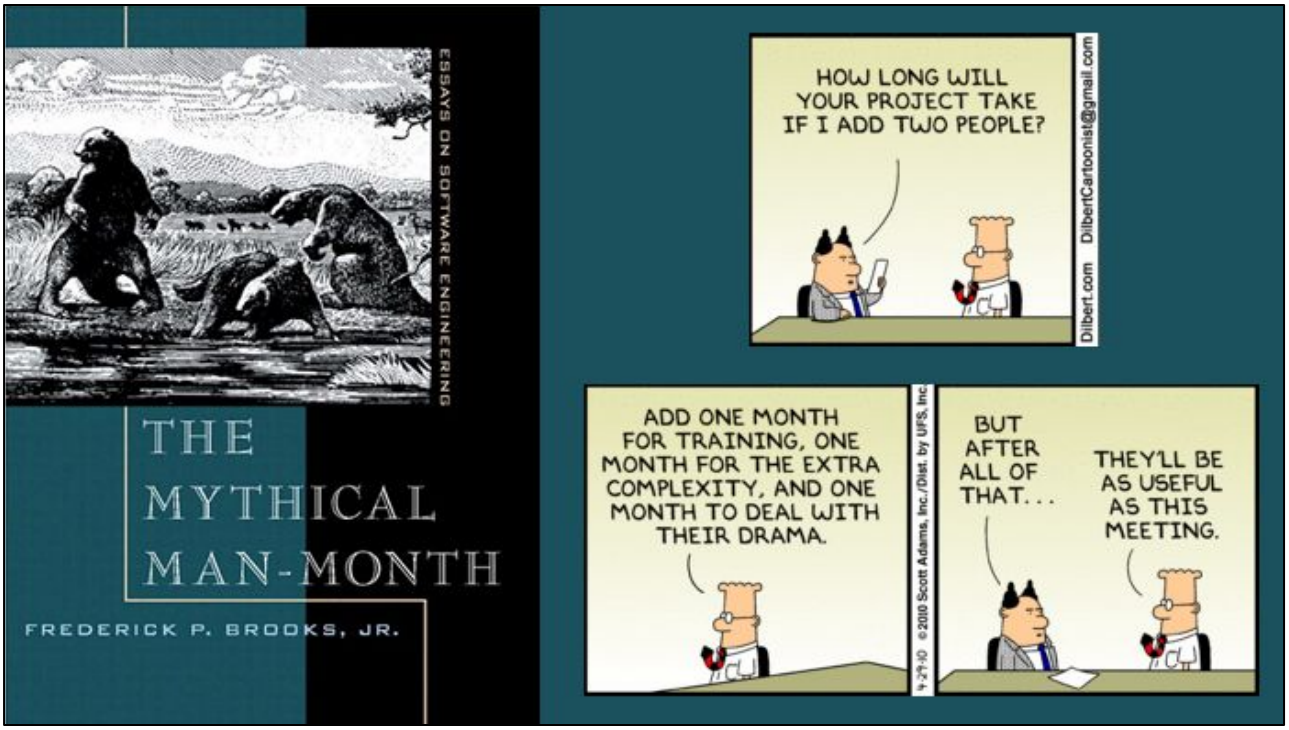
Major Capital Projects

- ICM
- BCeSIS
- JUSTIN
- PANORAMA
- NRPP
- EHealth

The growth in contracting dollars has been driven by outsourcing of operations, and by outsourcing of major capital builds, and just by the sheer size of contracts allocated to those activities.
Since this is a room of architects, I'm going to assume that the operational outsourcing is less interesting, except insofar as limitations in those contract vehicles have in turn limited the options available for you to for systems design.
I'm sure everyone has an HPAS horror story.
Anyways, the capital projects, the projects on the right, all have a three things in common.

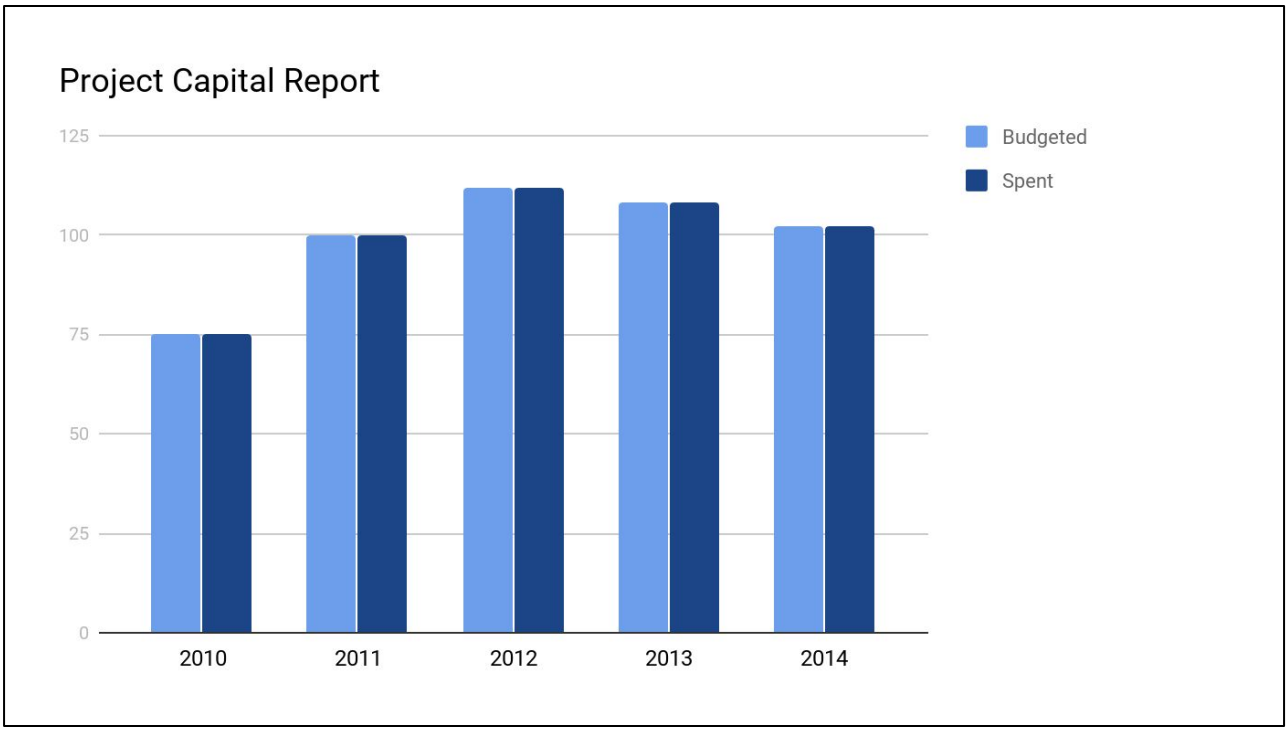
First, they all managed to deliver a poor enough product that they rose to the level of political notice.
They were mentioned during debates in the Legislature.
The were mentioned in the press.
All publicity is not **necessarily** good publicity, and political IT messes are about as fun to clean up as sewer system failures.

Second, these projects have in the range of eight zeros, give or take a few tens or hundreds of millions among friends.
That's enough money to run a company of 100 **very** well paid staff for **five years**. It's a huge amount.
It's a lot more than what was considered a "big" project just ten years ago, when a $10M project would be in the upper range.
And all that money is being spent in about the least effective possible way, in a big, time-boxed lump.



We know big projects are risky in **theory**, because we've all read or at least heard of the Mythical Man Month.
We know that the more people we add to a knowledge project, the higher the cost of coordination gets.
Reporting, scheduling, status updates, all the meta-work required to coordinate a large organization.
Modern tooling and electronic communications have vastly improved software since Brooks wrote, but there's still no doubt that coordination costs increase dramatically with team size.

We also know **empirically** that big projects are risky.
Every year the Standish Group updates its CHAOS research on IT project failure, and every year one of the strongest correlates to failure is project size.
Big projects just tend to fail more often than smaller ones. And by “fail” they don’t mean total write-offs, but failing to meet user needs, or failing to achieve the initial stated objectives, or just blowing the budget, or all of those at once.
Another major correlate is lack of user involvement, which we’ll get to when talking about small project organization.



So, the third thing all these projects have in common…
This is a fake chart, I just made it up, it's a placeholder for when I can find the ICM capital plan again.
Anyways, I was floored when I first saw the ICM capital plan because this $180M project, where they had to build out the capital plan for five whole years, had perfectly predicted capital spending for **every year**. For five years!
So, either the ICM planners were the best estimators in the **history** of estimating, or… something else.

- **On time**
- **On budget**

So, the third thing all these big, bad, failed projects have in common… is that they were all on time and on budget.

- **On time**
- **On budget**
- **Unsatisfactory**
  - Did not meet user needs
  - Lacked promised functionality

But they were also unsatisfactory.
They ended up so deficient in one way or another that they had newspaper articles written about them.
Panorama, the exception to prove the rule, managed to be unsatisfactory AND late AND way over budget.

Promise	Delivery	Delivery
		
	Failure	**Still a Failure**

Now,
If I promise you a mustang and I deliver nothing... that's a pretty obvious failure.
But it's ***also*** a failure when I promise a mustang and deliver an Edsel.
That still counts as a failure.
Not many projects come up **completely** empty, but there **are** a startling number of Mustangs promised and Edsels delivered.

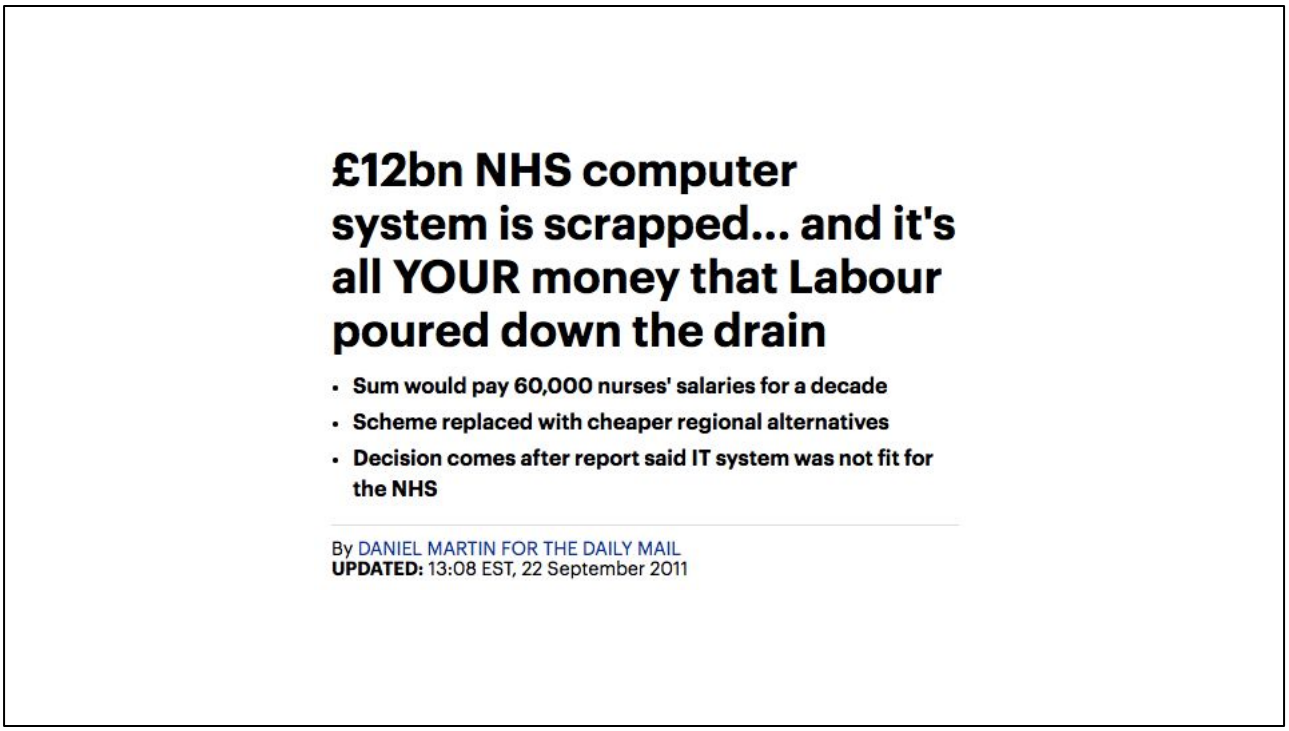
This situation, of having a recent history of major project failure, and a big contractual exposure to international “tier one” IT consulting companies is not special to BC.
We are not unique and special snowflakes.
It’s not even unique to government, there’s no shortage of Fortune 100 companies with similar tales.
It’s not tied to any particular political philosophy, left or right.

Fads of the 2000s

Outsourced
Waterfall
Transformation
Initiatives

Along with crocs and livestrong bracelets,
big outsourcing and waterfall transformation projects appear to have been a flavour of the moment,
a fad that swept through organizations in the late 90s and 2000s,

So, around 2010,
we started to see the chickens come home to roost for government in a visible way.
For example, the UK had a rash of expensive, high profile IT failures.

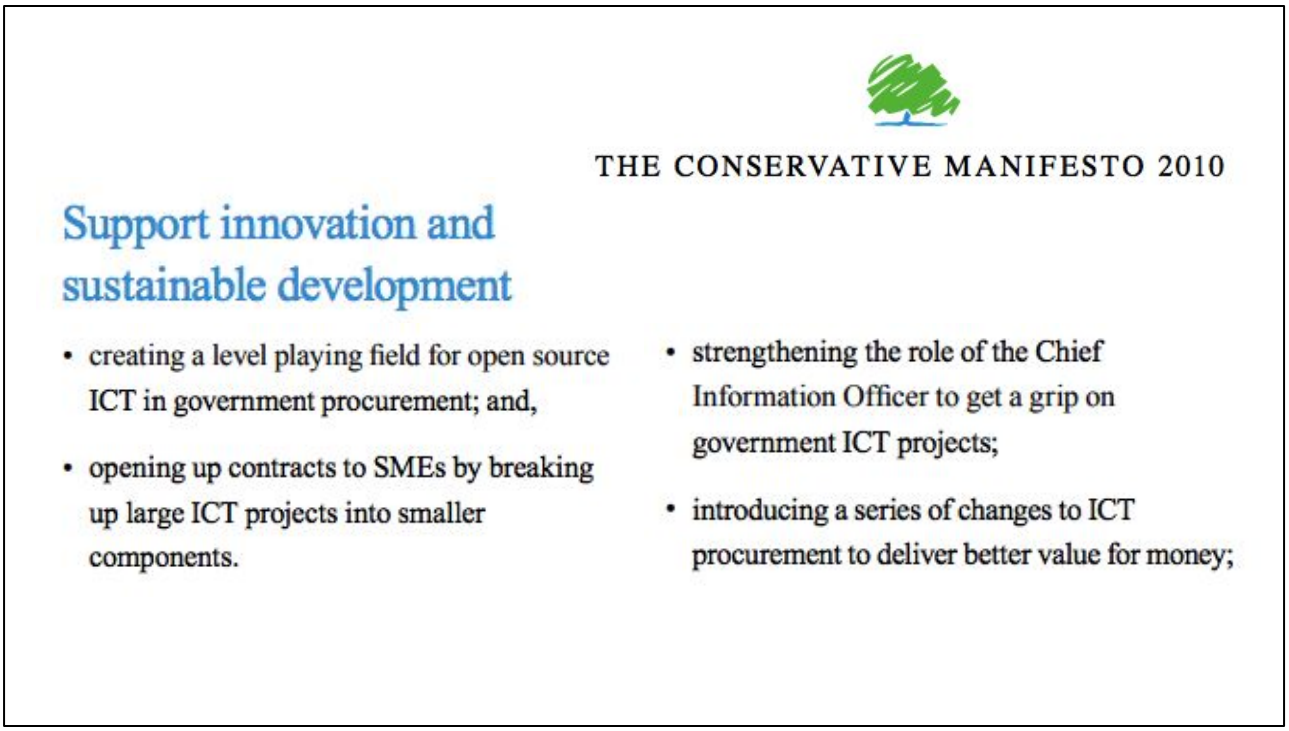


£12bn NHS computer system is scrapped... and it's all YOUR money that Labour poured down the drain

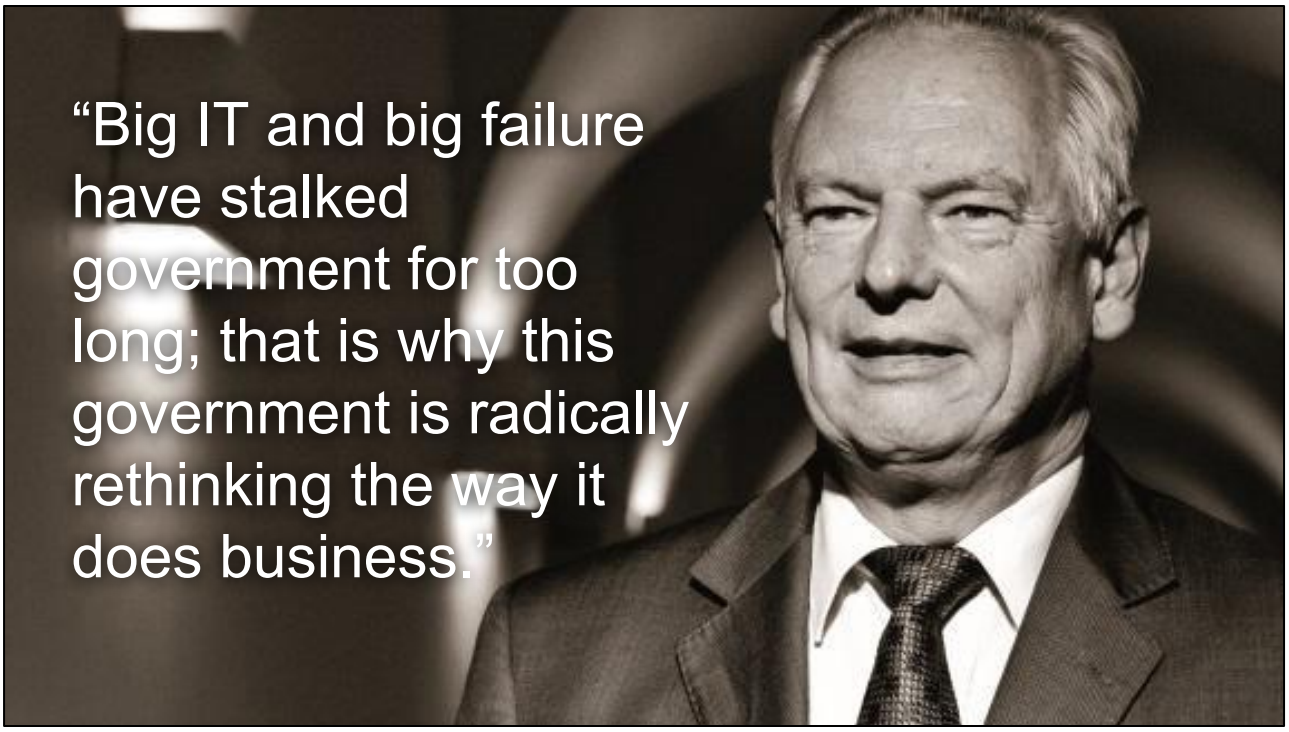
- **Sum would pay 60,000 nurses' salaries for a decade**
- **Scheme replaced with cheaper regional alternatives**
- **Decision comes after report said IT system was not fit for the NHS**

By DANIEL MARTIN FOR THE DAILY MAIL
UPDATED: 13:08 EST, 22 September 2011

IT rarely rises to the level of a political issue,
because mostly people just put up with the Edsels,
but as the delivery gets worse and the cost goes up,
eventually people do notice,
and they wonder what kind of computer program can possibly cost 12 billion pounds.

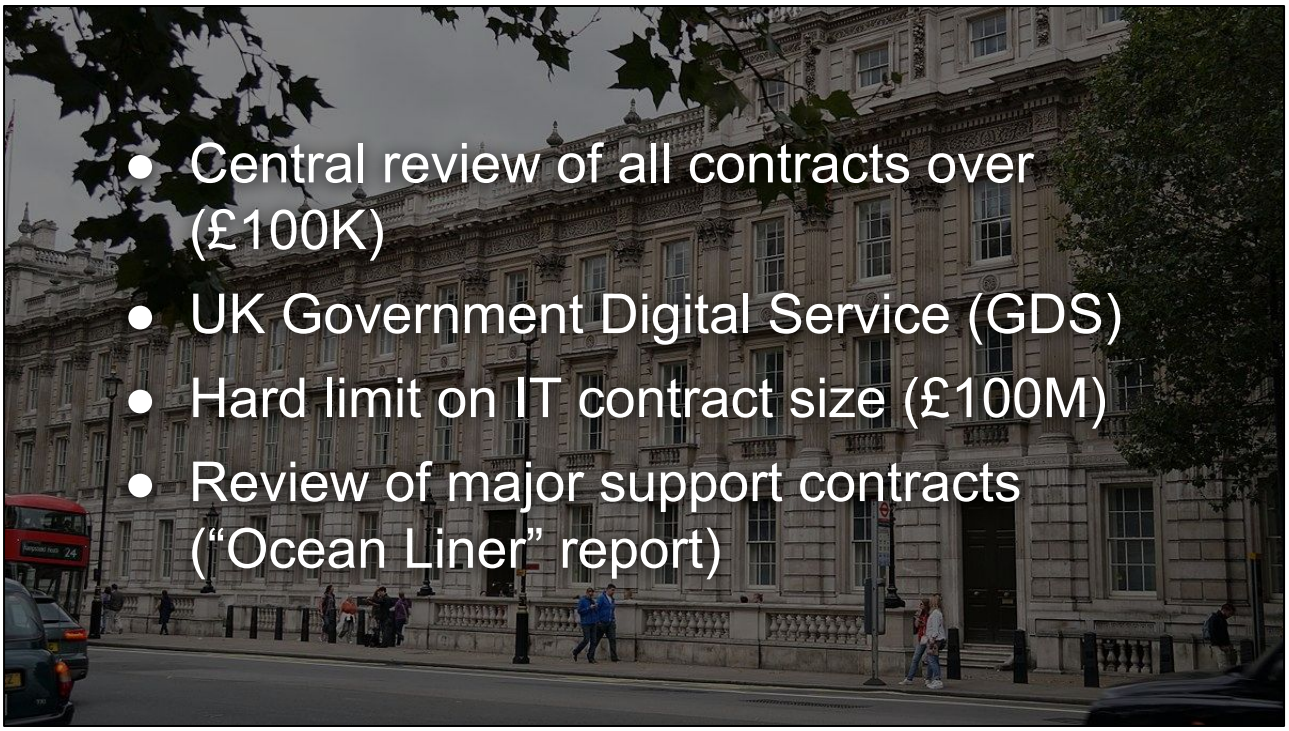


So in the 2010 UK election,
the Conservative party made some **surprisingly specific** promises
regarding information technology in government.
Bringing in open source,
breaking up large contracts, and generally “getting a grep” on IT.



In 2010, the Conservatives won government, in a coalition with the Liberal Democrats,
and they brought in some big changes that have echoed around the world of government IT.
The Minister of the Cabinet Office, Sir Francis Maude,
made IT reform a top priority and put some genuine **political** power behind the push for change.

Maude used his clout right away to get cabinet to support central review, by his Ministry, of all major IT capital projects.
He also set up the first high level government “digital service”,
and the one most frequently cited and copied around the world



After the first term, they eventually settled on a permanent hard cap of £100M and a lighter process for reviewing all large initiatives.
They also reviewed all their existing outsourced support contracts, with an eye to changing the overall culture of IT in government.
(Hence the reference to changing the direction of an ocean liner.)
But the UK wasn't the only government grappling with traditional enterprise IT.

Hop back across the pond, a couple years after the UK started their changes and set up GDS, the US also had a “big IT” moment.
The web site that was supposed to handle ObamaCare insurance registration for for half the population couldn’t handle more than a few thousand applications per day.

And so the US public learned about a “billion dollar website” and they too wondered how a website can cost a billion dollars.
And the policy elite got a little education on how IT is delivered inside government, what a system integrator is, what a waterfall development project is, and why going over the waterfall sometimes hurts.

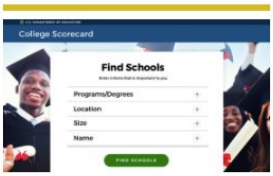
But because healthcare.gov was tied to the marquee policy achievement of the Obama Presidency, a bunch of programmers from the famed Obama campaign IT team descended on Washington and volunteered to fix the thing.
The Time magazine article is a great read, Steven Brill is a good writer, but the fixing isn't the important bit.
The important bit is that **after** fixing it, a bunch of them stayed on, because in fixing healthcare.gov they got a glimpse at the internals of government IT and thought they could help make it better.

THE U.S. DIGITAL SERVICE

Our Work

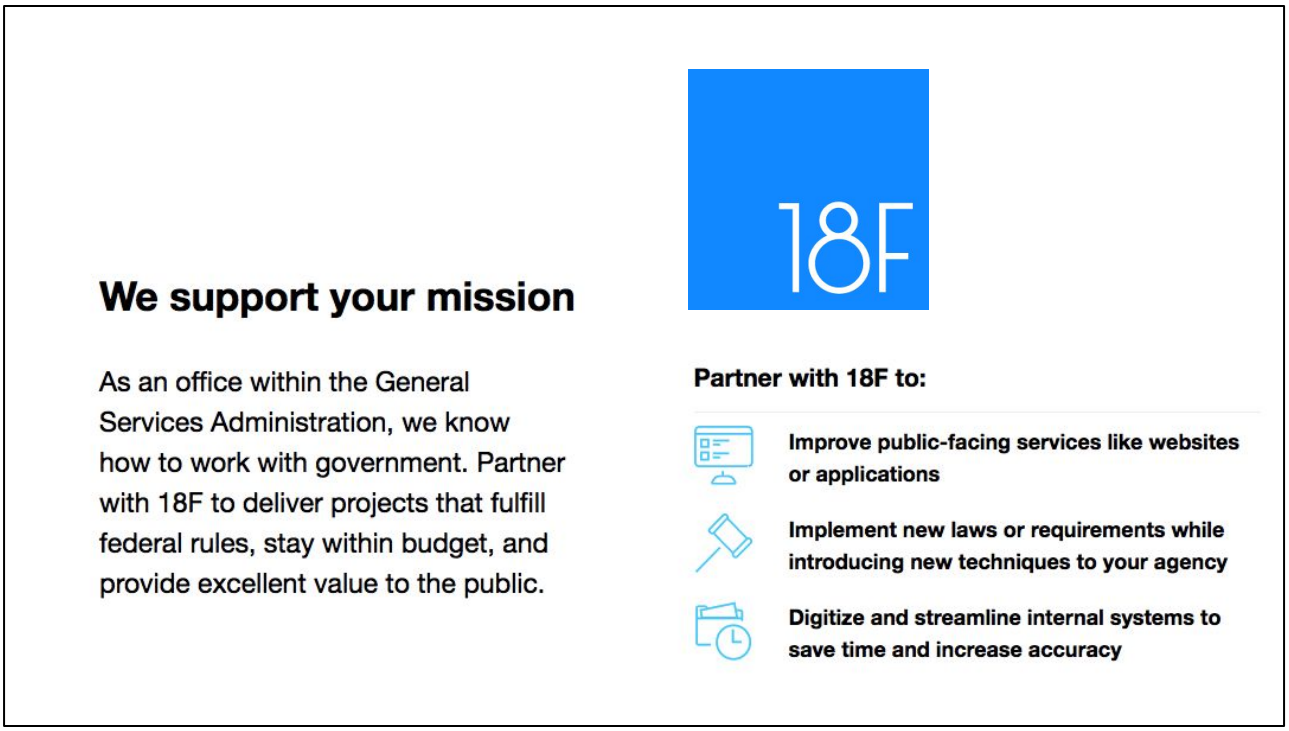
Vets.gov

Digital Immigration



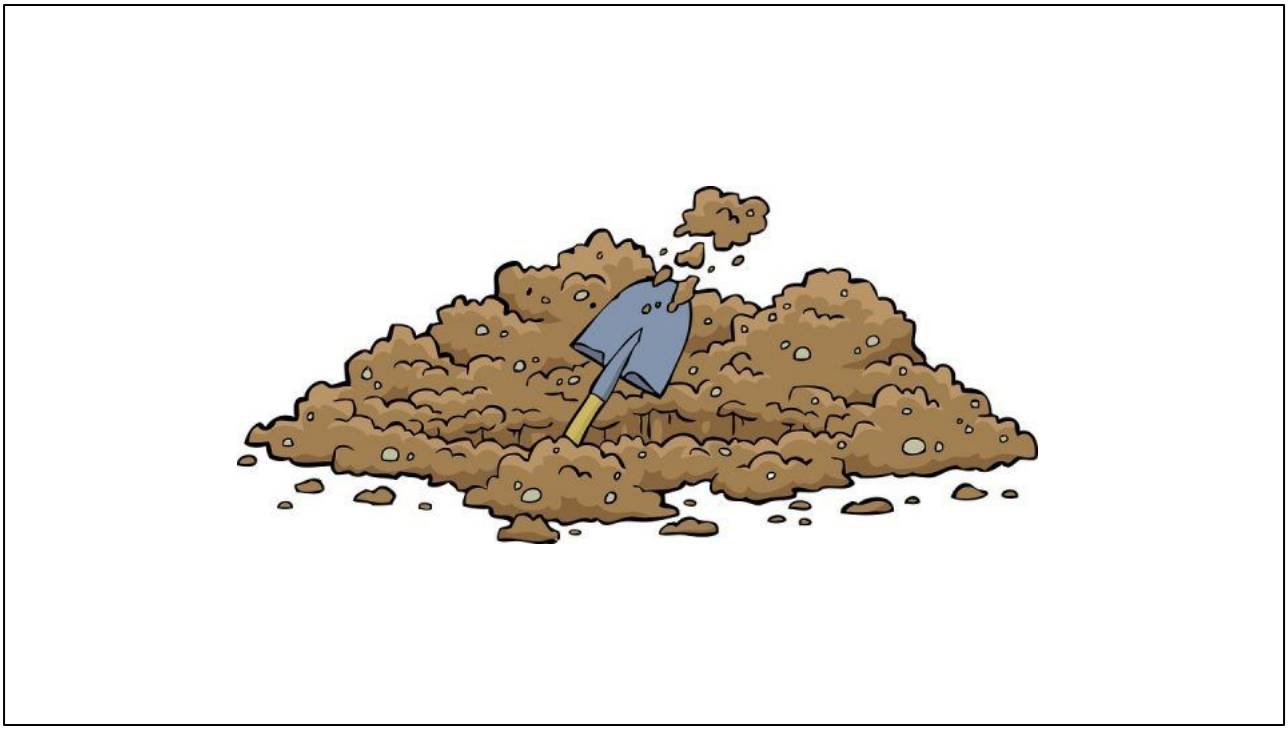
College Scorecard

And from that nucleus two organizations were born, the US Digital Service, run out of the White House, and 18F run out of the General Services Administration.
Both are much smaller than GDS, neither has the kind of budget or central authority that GDS commands, because the US system is much more decentralized, and despite the success of the healthcare.gov rescue large scale IT reform is not really on the agenda of the executive.
You can see by the web site that the USDS has been about picking particular problems and bringing tiger teams to them, often in departments, like the VA, that have been traditionally challenged.



18F has taken a lower key approach, they don't send in teams usually, they co-locate 18F staff and try to build teams more organically inside departments.
The goal in both cases is to amass small wins, and slowly prove to decision makers that IT problems aren't necessarily solved through the application of more money, or through the outsourcing of risk.

Unfortunately, the arrival of the Trump administration has had a really negative effect on the the executive, the sense of optimism about government, that public service can effect positive change, has been lost.
So it's possible the USDS might not survive in its current form, it's run out of the White House Office of Science and Technology, which has been gutted, and there's been a lot of staff loss.
18F is deeper in the bureaucracy, and it may well survive and continue to push modern development methods from the inside.



All this really, to say that, BC is not the first jurisdiction to sort of look at the state of their so-called “enterprise” IT, and think “it’s time to start following the first law of holes”, which is “when you find yourself in a hole, stop digging”.
The UK spending reviews, GDS, USDS, 18F, the Australian Digital Transformation Authority, these are different ways government have responded to the question of “OK, so there’s a problem, what’s the way out”?

In your role as Minister of Citizens' Services I expect that you will make substantive progress on the following priorities:

- Institute a cap on the value and the length of government IT contracts to save money, increase innovation, improve competition and help our technology sector grow.

- Ensure government IT and software development procurement work better for companies that hire locally and have a local supply chain.

BRITISH COLUMBIA

Which brings us to this little nugget.

...

For the sake of brevity, I hope that even if you disagree ideologically with a policy of favoring local procurement, you can accept that other folks might nonetheless want to make it a priority for their own misguided reasons.

That **other** bullet though: Why would an incoming government want to commit to such a blunt policy instrument as a **cap on IT contract size and length**?

You cannot make a $100,000,000 loss if you do not make a $100,000,000 bet.

The bluntest political calculus is just that if you want to stop making big mistakes you have to stop making big bets.
Half the Ministers in the current cabinet got to grill Ministers of the **previous** government over major IT project failures.
I think it's fair to assume that they are not interested in being on the **receiving** end of that process.

Adrian Dix, Minister of Health
December 2017

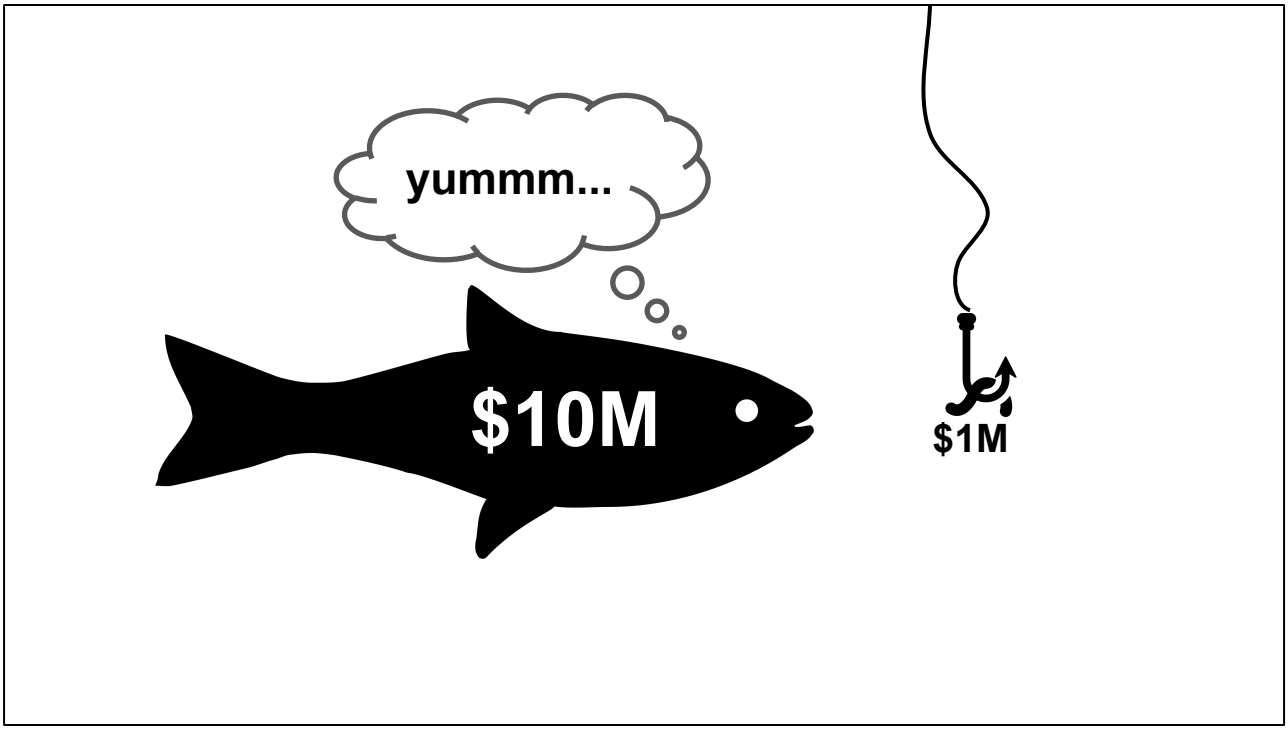
"I think at its core, these IT projects are too big, and not gated enough.

What we need are smaller spends.

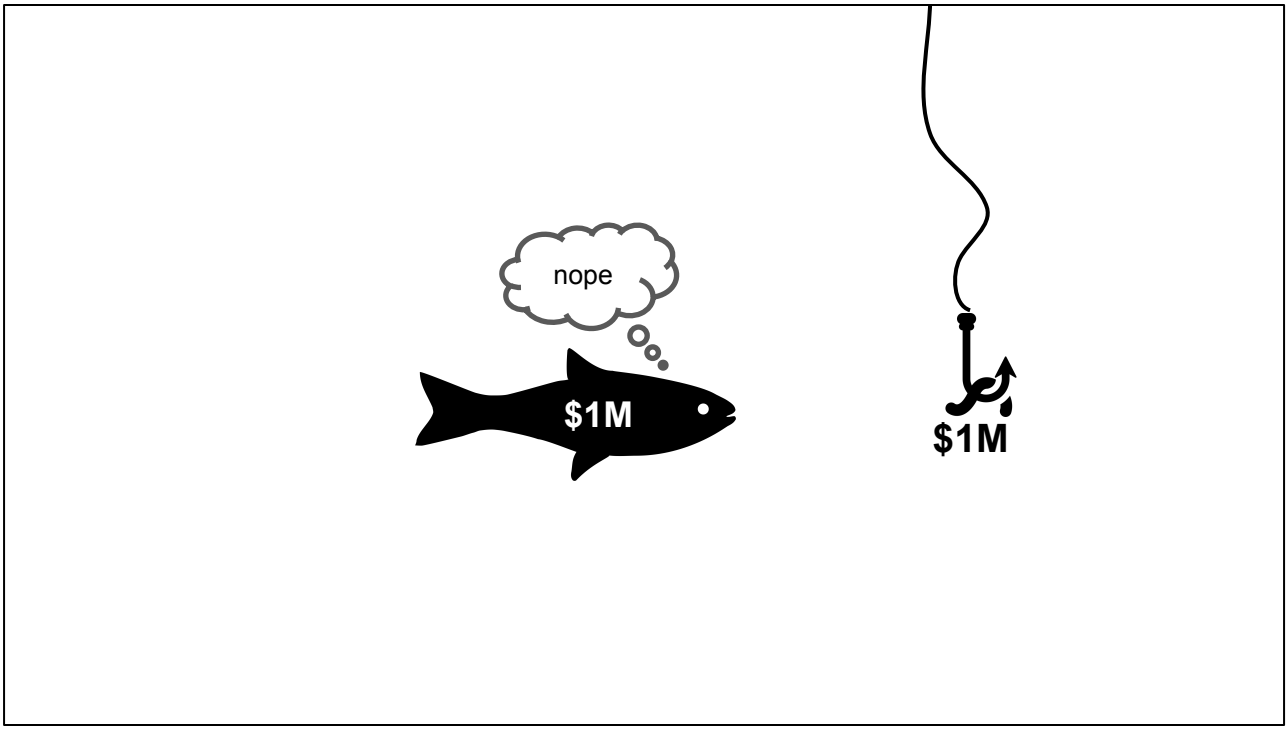
And if the smaller spend doesn't work, sure you have to write off some money and it doesn't look good, but it's a smaller spend.

When you start and approve projects with these $500-600-800 million in costs, and go ahead and spend a couple million you are stuck into those systems before you know that they work."

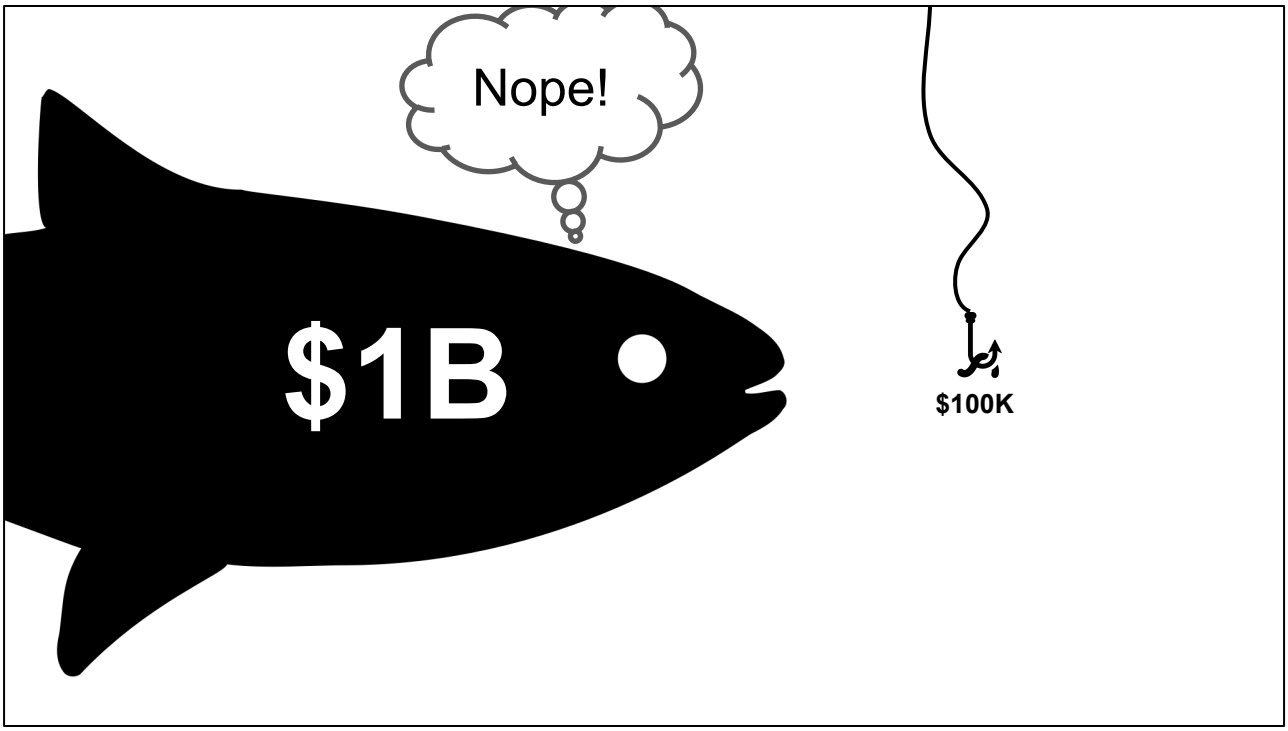
When Minister Dix changed the leadership at the Coastal Health Electronic Health Record project, he said this to the Vancouver Sun.
"I think at its core, these IT projects are too big, and not gated enough.
What we need are smaller spends.
And if the smaller spend doesn't work, sure you have to write off some money and it doesn't look good, but it's a smaller spend.
When you start and approve projects with these $500-600-800 million in costs, and go ahead and spend a couple million you are stuck into those systems before you know that they work."



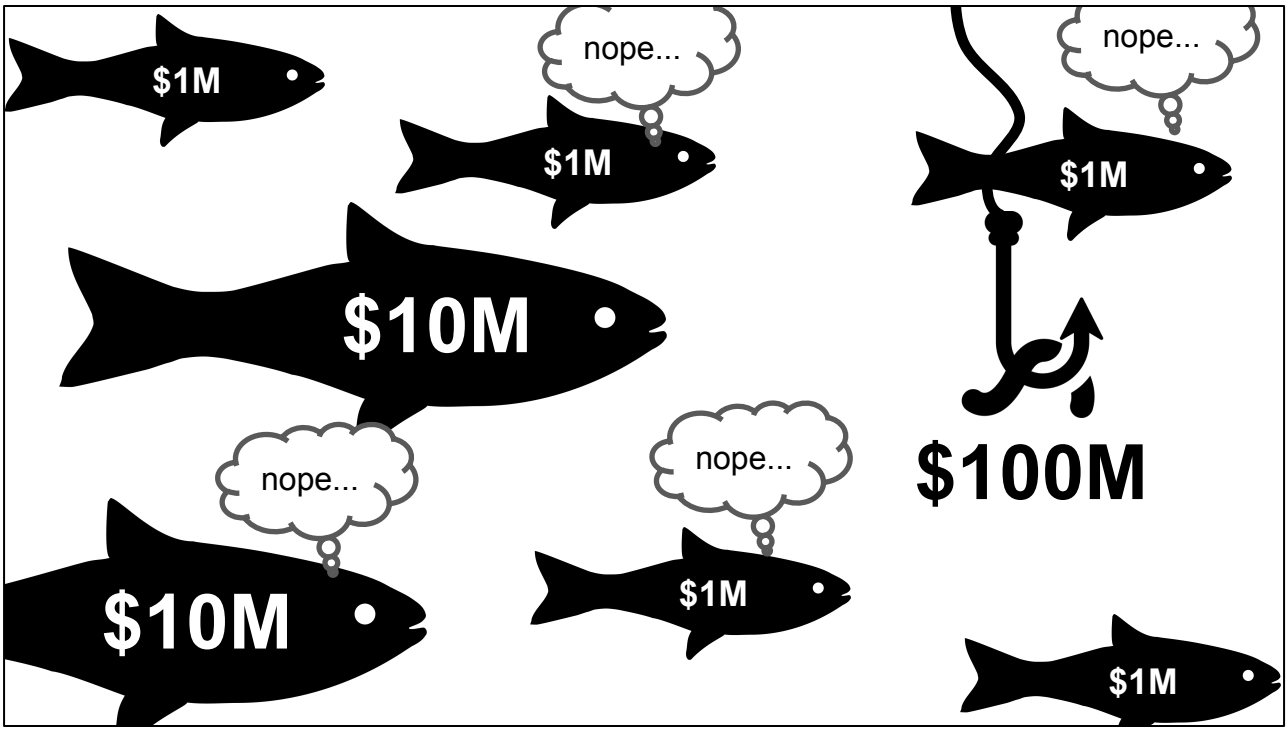
The second reason for establishing a cap is more subtle, but it comes out of fairly cursory examination of the population of companies capable of competing for any given opportunity.
Basically, a sane systems integrator won't pursue an opportunity worth more than 10% of their revenue.



Holding a contract worth more than 10 or 20% of revenue is too much exposure to a single customer. It’s just too risky.



Conversely, **smaller** opportunities are not financially viable for larger systems integrators. The overhead of bidding and planning for a small opportunity renders them uneconomic.



Now, look at the population of local system integration firms.
Are there any local $1B firms? Not really.
Sizing opportunities for $1B firms basically guarantees the awards go to non-local firms.
The contract value cap serves a triple purpose of mitigating failure risk, opening up opportunities to local bidders and discouraging large foreign bidders.

Long term contracts legally bind the government to aging solutions to swiftly changing problems.

Finally, the cap on contract **length**.
There may be a strong argument to be made for signing 10 year deals for highway maintenance.
I hope, after the last 10 years of change, there a no arguments left to make for 10 year deals for IT management or hardware or networks.
Things change.
Technology changes 10 times faster.

So.
Now what?

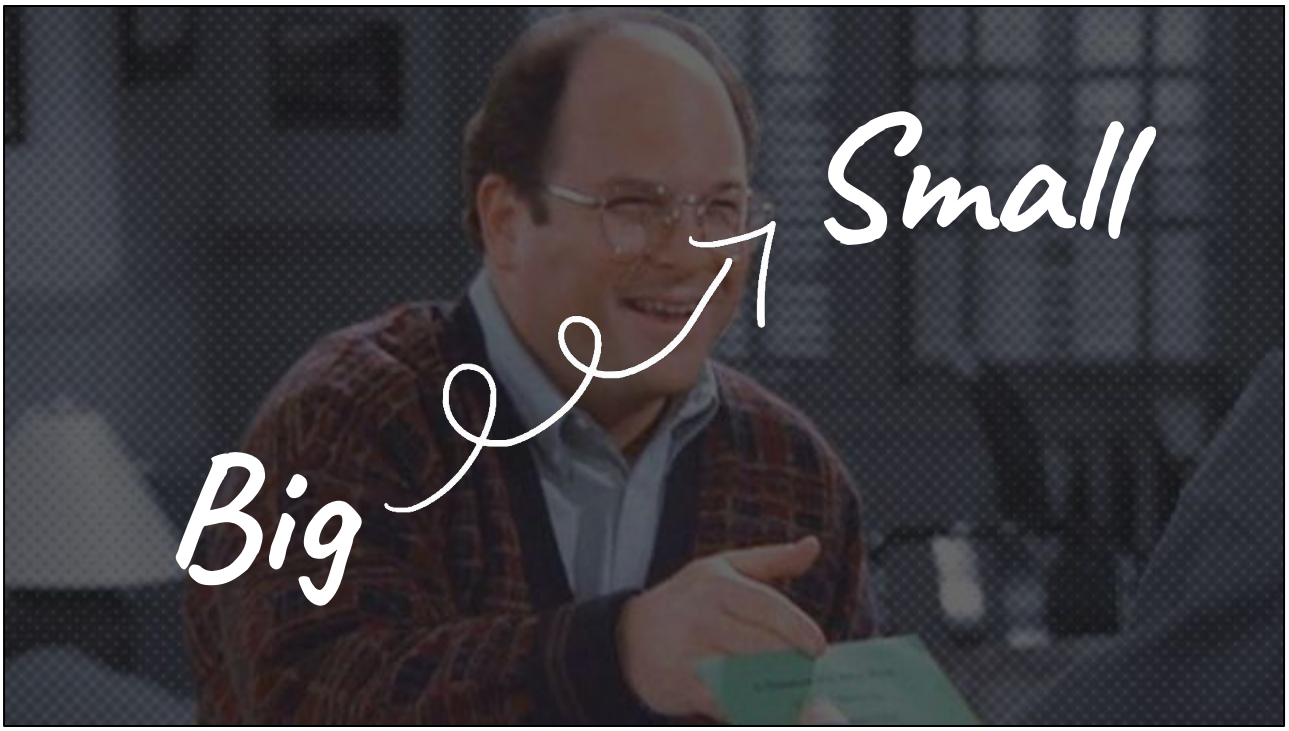
OK.
So other governments have been here. They've started down their own roads of change.
Our new government has a blunt policy that mitigates against continuing on as we have in the past.
It **doesn't** say what your should do instead.
So, what should the future look like?

There's this episode of Seinfeld…
It starts with George and Jerry and Elaine in the coffee shop, and George is complaining about his life,
No job, no prospects, living with his parents.
"Every decision I've ever made, about anything… has been wrong, Jerry."

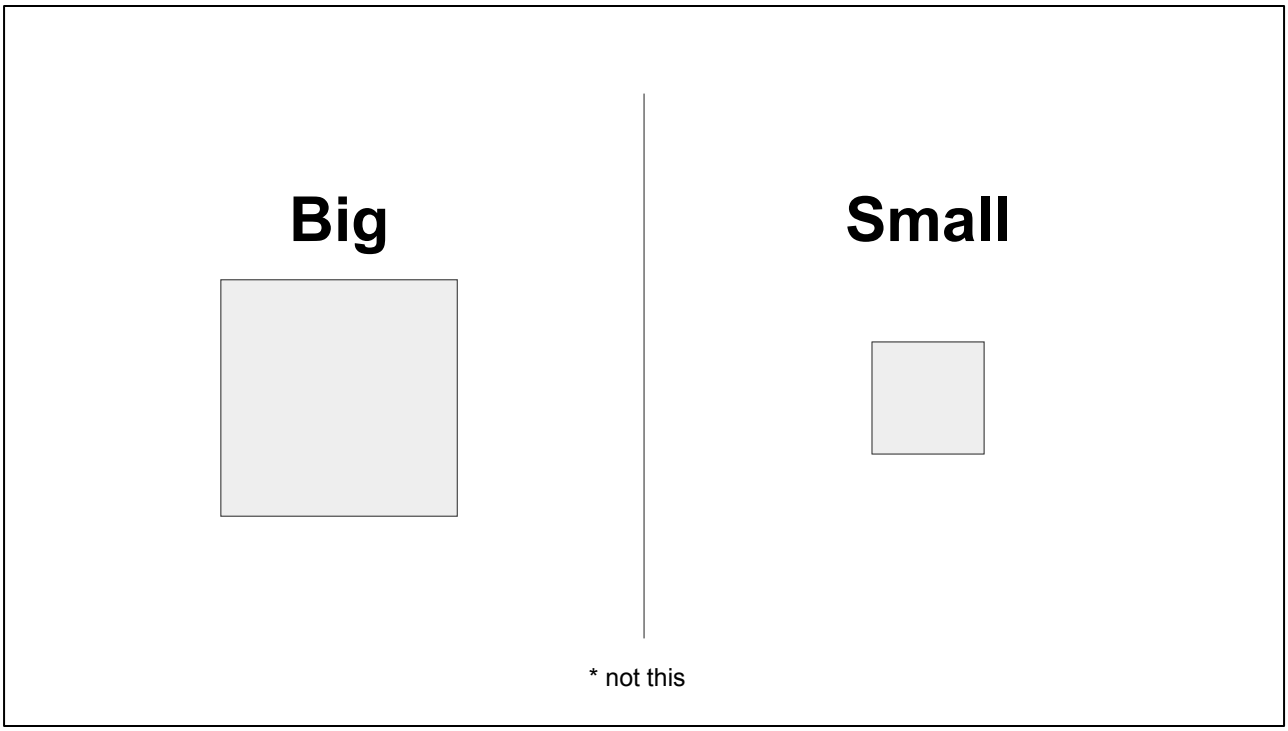
And Jerry replies,
“If every instinct you have is **wrong**, then the **opposite**, would have to be **right**.”

And so George goes on through the rest of the episode, doing the opposite of this initial instinct, and it's all a screaming success.
One of the best Seinfeld episodes ever, in my book.
So, if we take the past as a guide, and we follow George, and do the opposite, what should be future look like?

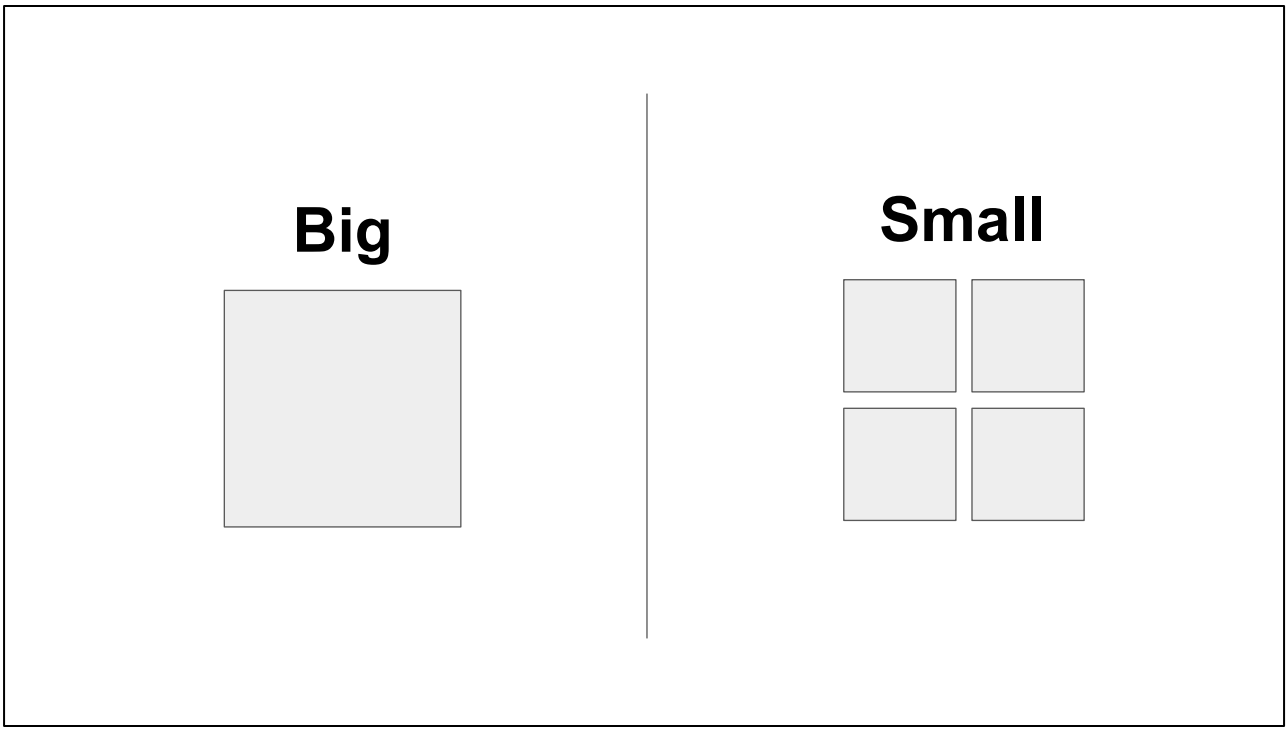
I give you, the Costanza Plan.
If past projects have been big, proprietary, opaque waterfall projects, carried out by outsourced teams with bias towards replacing existing systems wholesale,
Then if you do the opposite, In the future,
projects will be small, open source, transparent agile projects, building knowledge in in-house teams, with a bias towards enhancing existing systems incrementally.



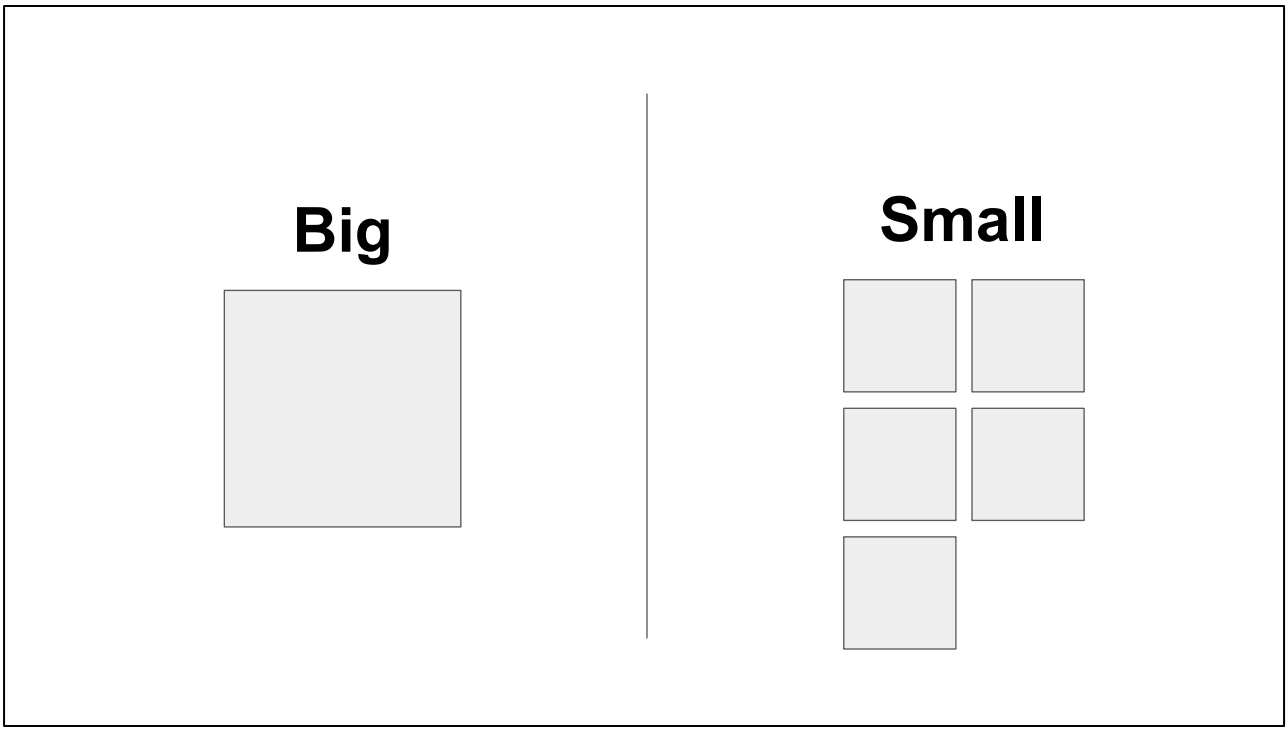
So first, big versus small.
This is the hardest part, really, and it’s good this room has lots of architects, because its you folks that will bear a lot of the burden of breaking big plans into smaller ones that can be executed incrementally.



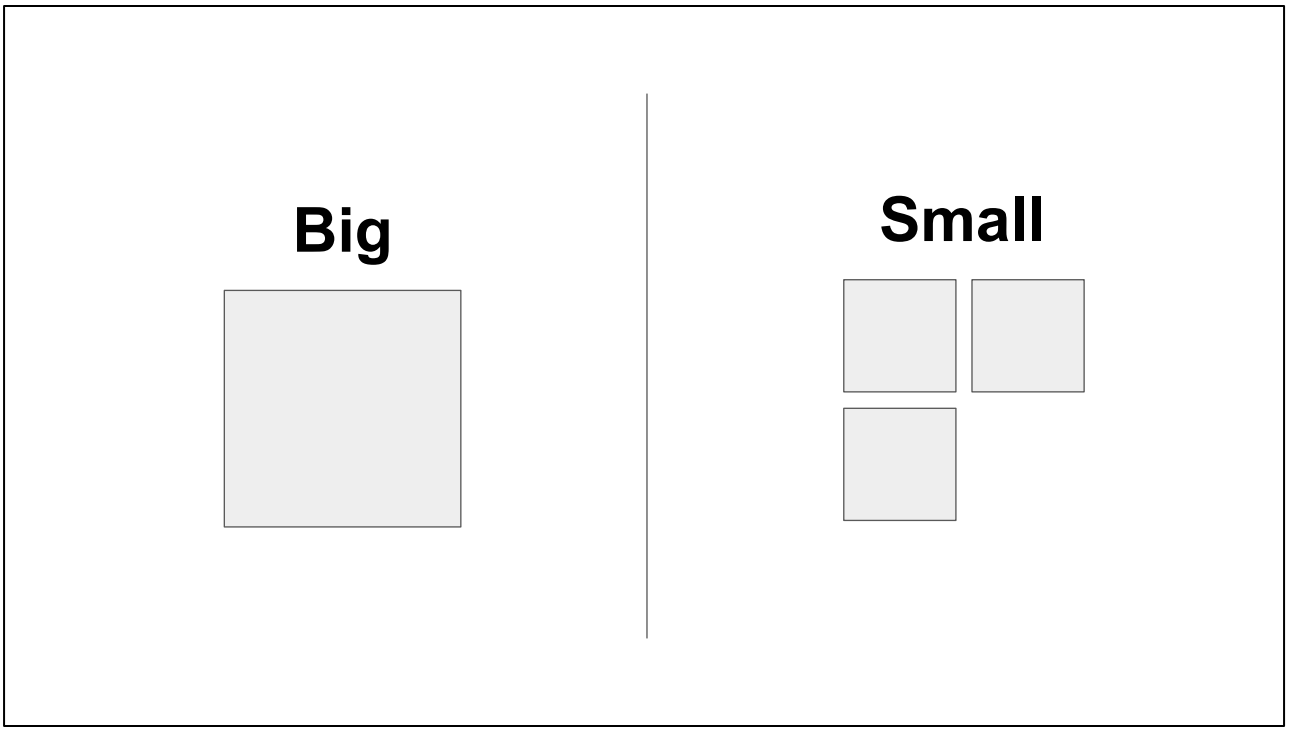
Because, just to reiterate, a cap on the length and value of IT contracts doesn't mean replacing one big thing with one small thing. It's not a "do more with less" mandate.



It’s a “do the same thing in smaller parts” mandate.



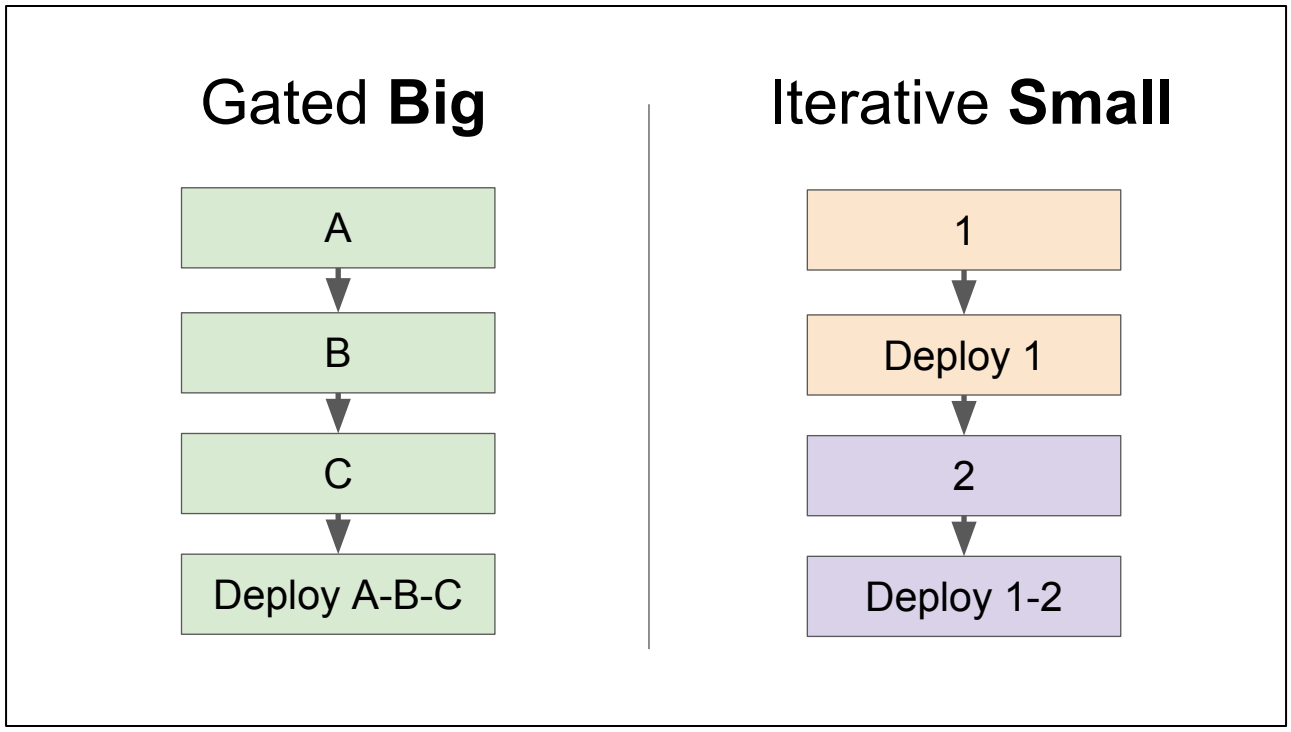
It might even involve more parts. It might be more expensive.



If you’re lucky, it could take less parts, but regardless, it will proceed in smaller chunks.
Not one big, multi-year contract with a systems integrator, no false promises about the “vendor taking on the risk”.
Instead a set of smaller engagements, each of which delivers a new piece of **independently valuable** functionality.
That’s the key, and the hardest part, breaking a big project into **real incremental pieces**.

> Fortunately, there's a very simple test to determine if you're falling prey to the False Incrementalism: if after each increment, an Important Person were to ask your team to drop the project right at that moment, would the business have seen some value? That is the gold standard.
>
> **Dan Milstein**
>
> @danmil

Breaking a project into incremental pieces isn't hard, but it **is hard** to do it well.
If the pieces don't have independent business value, you haven't really de-risked anything at all,
that's False Incrementalism.
"If after each increment, an important person were to ask your team to drop the project, would the business have seen some value?"
Beware False Incrementalism.



While a gated contract that doesn't start B until A is completed in good order is better than an uncontrolled sprint to ABC, it's still not as good as a process that starts delivering business value as soon as possible.
The iterative, small approach may promise less functionality in a given time period, but it's more likely to **keep** its promises,
and more likely to adjust its deliveries to meet real requirements exposed during the process.

All that said, there's no hard and fast rules for this stuff,
decomposing each problem is going to be different, and difficult in its own way,
so all I can add is, May the Force be with you.

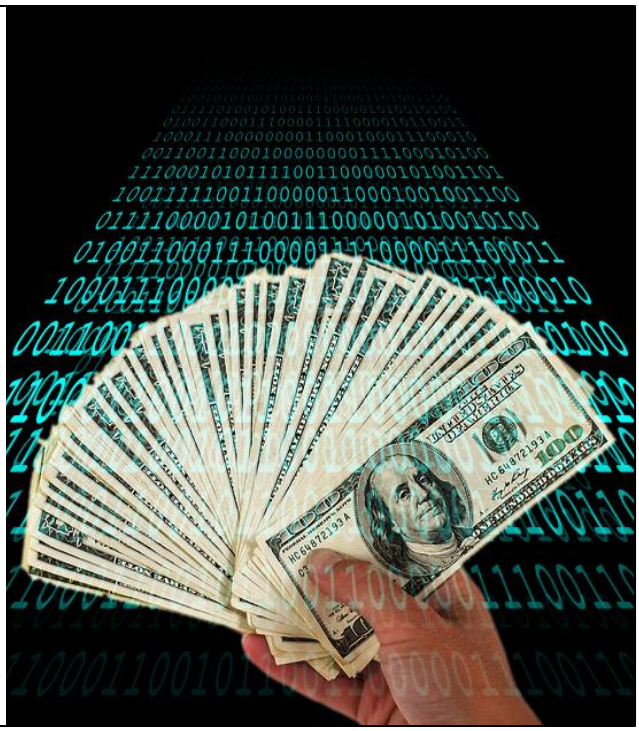
Next, open source versus proprietary.
I went back and forth on whether to include this one, since I’m such an obvious open source guy, and I don’t want to seem even more preachy than I already do.
So let me caveat up front by saying, and this goes for all these points, I’m not talking in absolutes here.
This is not either/or guidance, this is “bias in favor of” advice.

Using open source components

Releasing open source code

There’s actually two aspects to the change from a proprietary first mindset to an open source first mindset.
The first is the more common one, which is just making more use of open source components.
Architecturally, a database is a database, an app server is an app server, an OS is an OS, if you’re building up an architecture, fill in more of the boxes with open source options.
It’s already happening in government, it could happen more.

“Enterprise”
is a synonym for
“expensive”



First, straight up, dollars and cents, a lot of the kit that is deployed in the name of the “enterprise” is ludicrously expensive for what it does.
I know software licensing cost is never going to be more than 5% of a build, but even so, that’s 5% you could use for something else.
Licensing and then maintenance fees are also a cost over which you basically have no control.
Once you’ve committed, that part of the cost structure moves into the hands of the vendor.
And when operating dollars start to get tight, licensing cost avoidance can lead to anti-patterns, like my favourite, and there’s a number of these in the government:

The Licensing Oriented Architecture, or LOA.

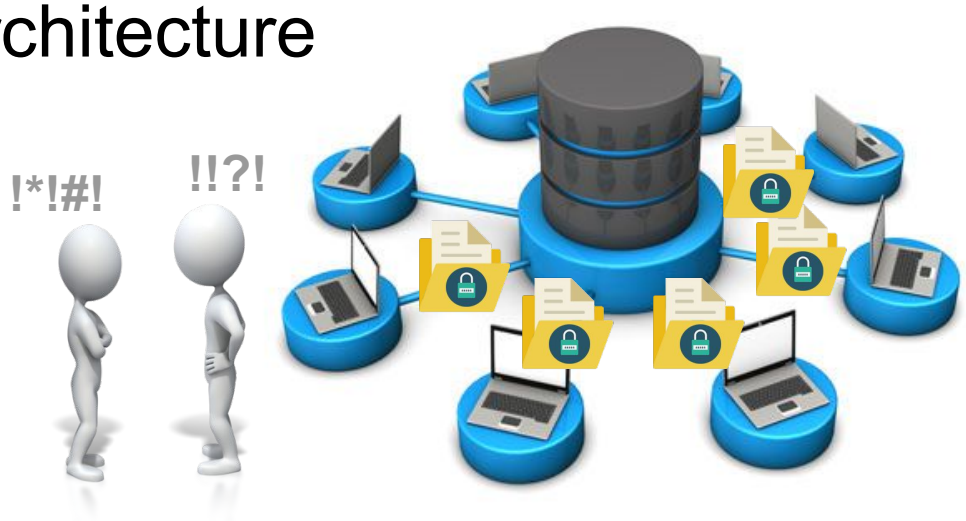
Licensing **O**riented **A**rchitecture

You know you're working in a LOA shop when every app is hosted in the same database instance, because that way they can all share the license. Or run in the same middleware server. Or share the same ETL host. Or, or, or.
It's all great at first, the system administrator has only one thing to manage, she's happy.

Licensing **O**riented **A**rchitecture

But as the number of apps pile up, the fact that the organization has this single point of failure, starts to make the operations folks squirrely. Any one app can exhaust resources for everyone else, or crash the whole organization. So new controls, and extra process, start to build up around deploys and resource access.

Licensing **O**riented **A**rchitecture



And then inevitably some new, and business critical app requires a new system feature, so the system has to be upgraded.
But older apps aren't compatible with the new version.
So the **whole business** is now on a synchronized upgrade cycle!
And why, again, did we design this way? Oh right, to limit license liability, to save money.

Bugs in black boxes break beautiful buildings

There’s lots of other reasons to use open source components, but from my time as a systems integrator, the one that kept repeating in practice was avoiding the black box. SI’s string together components to make systems, but the connections are never quite perfect. There’s always some combination of context and data that causes a bug.

Bugs in black boxes break beautiful buildings

Fix flexible free foundations fast

And when you hit a bug like that, if you can trace the problem into the component, using the open source code, you can often put a fix into just the right place and keep your system neat and clean.
When you cannot trace in, you have to work around, instead, and the system gets hairy and ugly, right from the start.
There’s an elegance and a satisfaction to finding the right fix, making your system work right, and sending that fix back to the open source project, rather that just hacking in a workaround.

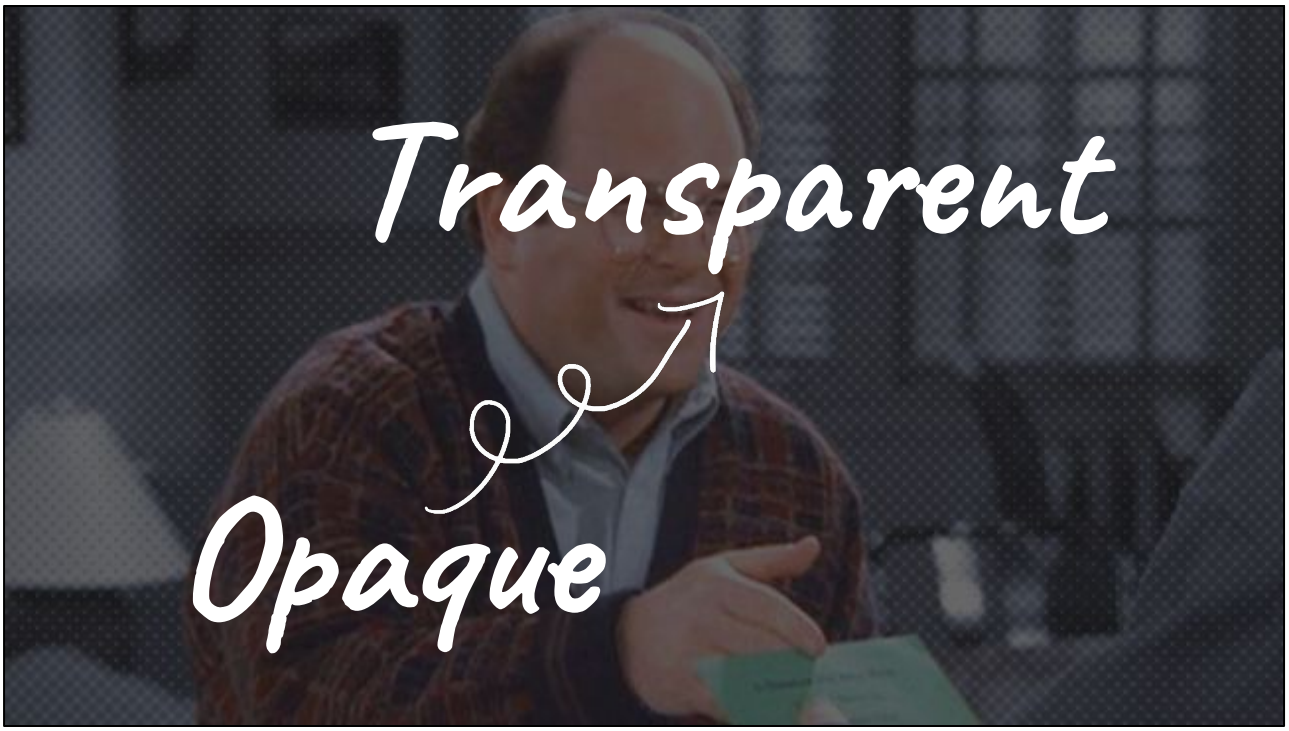
Using open source components

Releasing open source code

The second aspect of the proprietary/open shift, is releasing your own code as open source.

Releasing your code maximizes the public good.

At a minimum, the public will get value from your code because you'll run it and it'll solve a government problem. That's easy. That's the value you **can** predict.
Releasing your code opens it up to create value in all the ways that you **cannot** predict.
Some other jurisdiction could find it useful.
Some other part of the government could find some portion of it useful.
Some business could find it useful.
If you're lucky someone might even improve it and send you a patch, but **even if that never happens**, opening your code opens up the **possibility** for it to generate more value than just keeping it closed and running it yourself.



Opening up **code** is actually only step one on the road from an opaque project mindset to a transparent one.
Traditionally, the circle of people who know about and work on a project is pretty small: the sponsor, some users, the project team, the vendor if there is one. If it's really really big it might get one page on gov.bc.ca.
Let me start with **what** should be transparent and then move on to the **why**.

Everything should be transparent.

Everything should be transparent. Everything.
But let me be more specific.

Everything should be transparent

- Source code repository online
- Continuous integration status online
- Project mailing list with open archives online
- Every project artefact online
- Issue tracker online
- Developer quickstart guide with deployment instructions

Source code, continuous integration status, project mailing archives, all the documents and artefacts, the issue tracker, and developer quickstarts with a deployment guide, should all be online and public.
You can tell I'm "of a certain age" because I mention email archives, but everything else is just what you get by default hosting a project at github.
The **tools** are easy to come by, the trick is getting the whole project team, from UX to design to data to code, to buy in. And of course management. Always management.

Why put everything online, isn’t that just asking for trouble?
No, it’s ensuring the intellectual edifice you are building, which consists of a lot more than just source code, is explorable and comprehensible. To put it into one word,

On-boarding

* not a real word

On-boarding.
The documents provide context,
the deployment guide gets new team members up to speed quickly,
the continuous integration safeguards against early mistakes and ensures the deployment has been automated,
the mailing list archives provide historical reference to past decisions and rationales,
the issue tracker provides current status and future goals,
and keeping it all public means they can all easily hyperlink with each other.

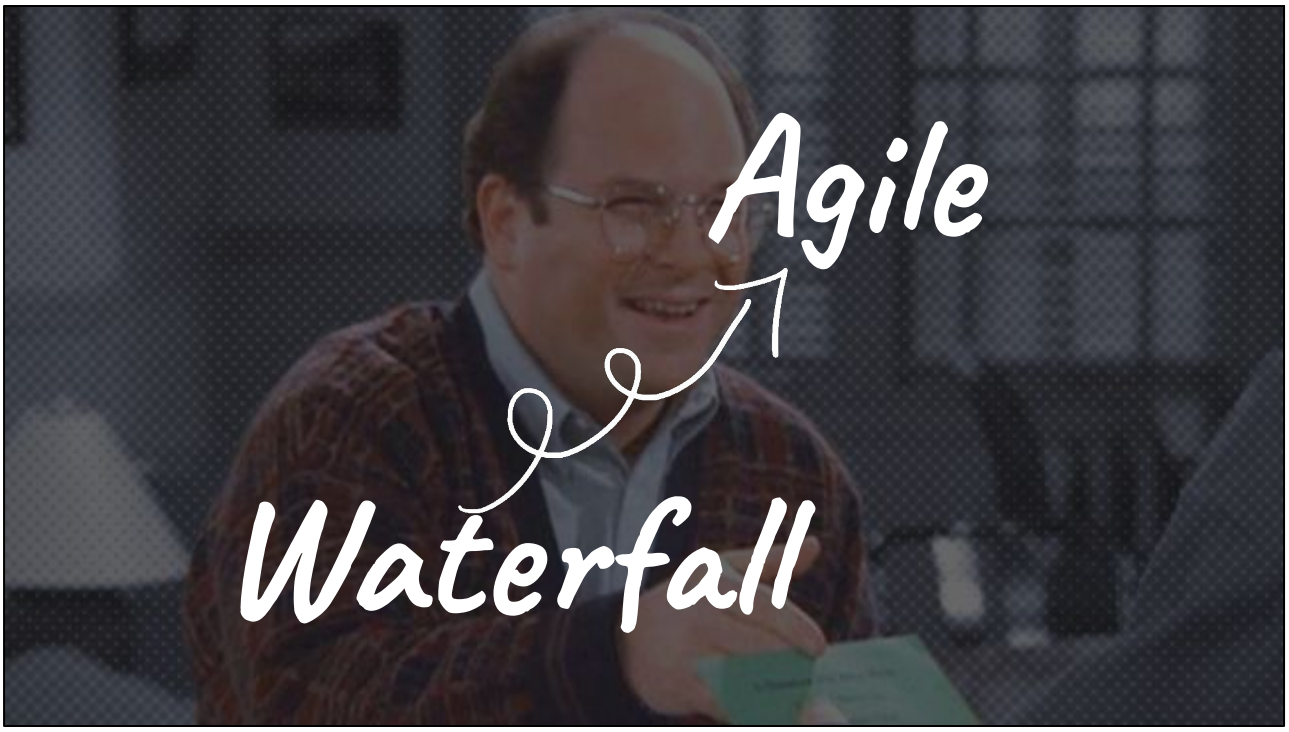
That explains **maintaining** the corpus, but it doesn't explain the "making it public" part. Why make it **public**?

On-boarding

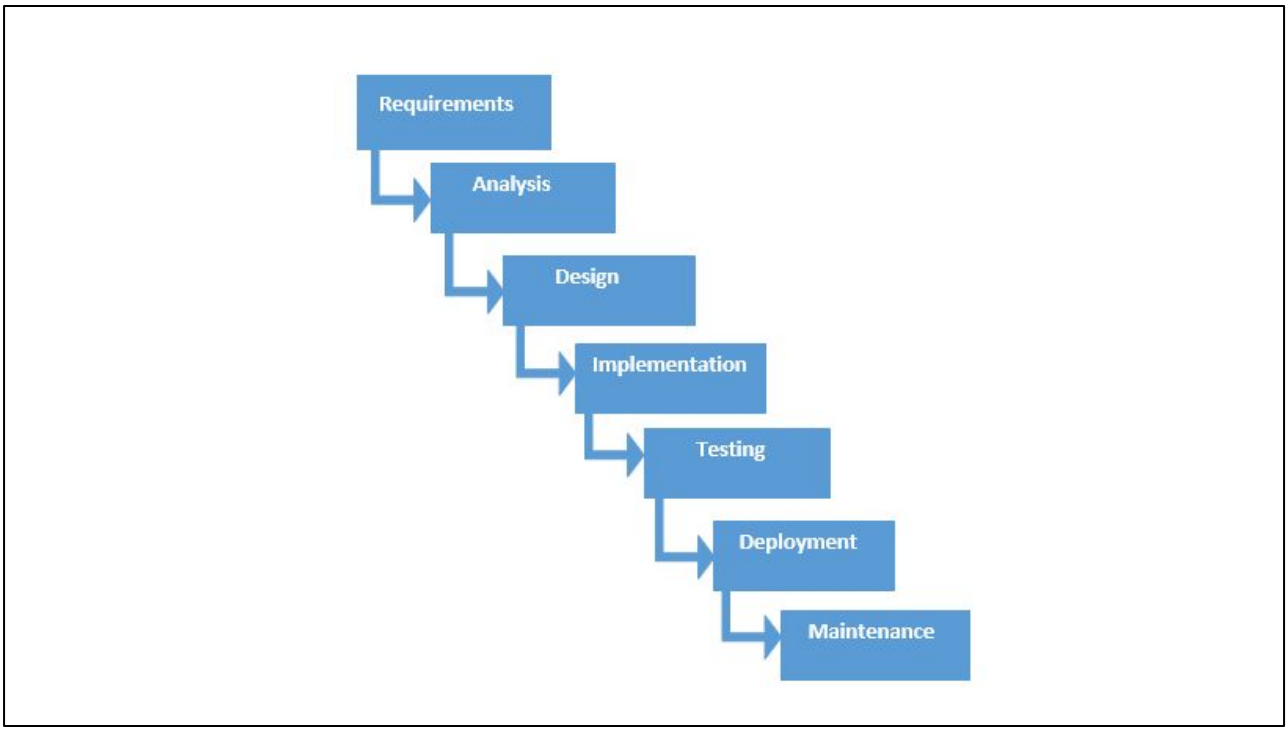
* not a real word

is not just for your team members.

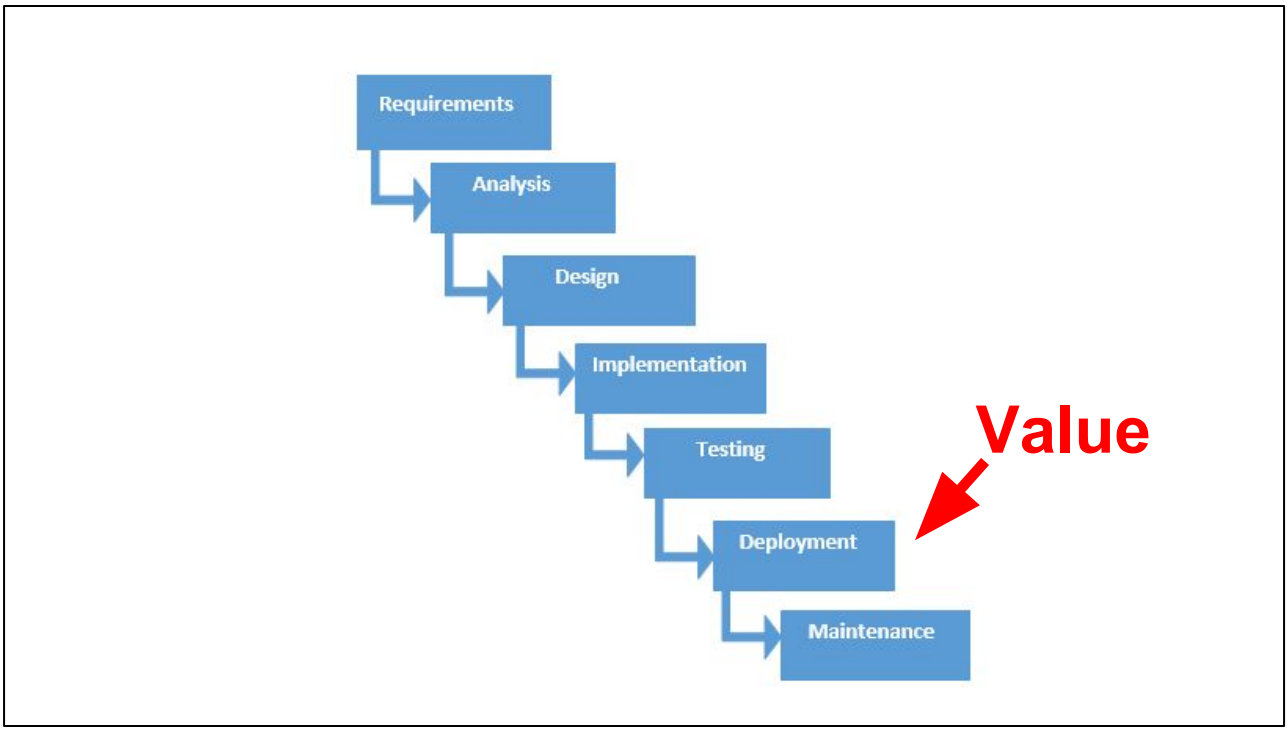
Because when people who aren't on your team can quickly get up to speed with your project, you unlock a bunch of new possibilities.
You can farm out individual tickets to outside developers, who can easily access the context they need to complete the work.
You can bring other organizations to the table and they can see if there is reality behind your claims.
You can easily have third parties evaluate the work of your team.
You can go out to bid for a new vendor and know that all bidders are in fact on an equal footing.
As a system integrator, I almost never bid maintenance contracts, because the incumbent always had an insuperable advantage in terms of knowing the scope and technology.
Going open is a statement of confidence and competence, it's a statement worth making.



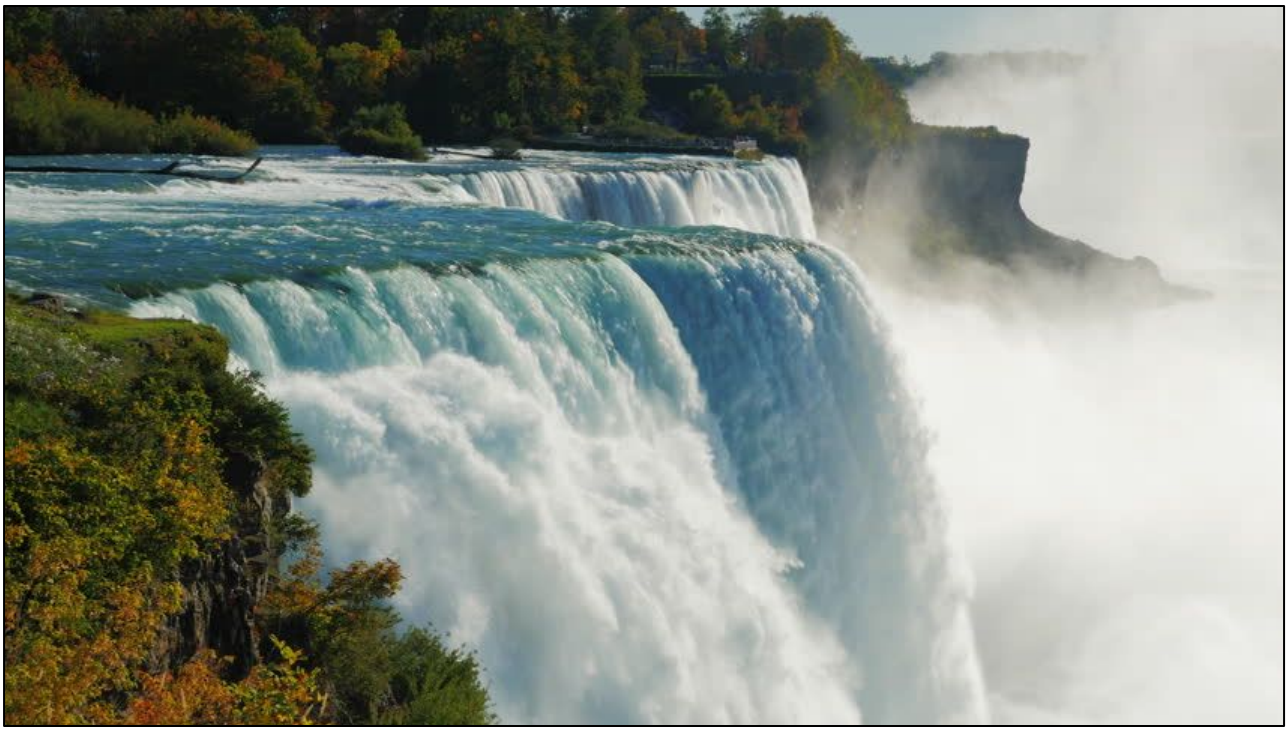
Next, moving from waterfall to agile.
You all have been made lean over the last few years, now you get to be agile too.
Agile is finding it's way into large enterprises, finally, but at this point it might feel like yet another buzzword.
It's not. It's a reasonable, fairly simple, way to build software. It's good for environments where the underlying business assumptions change frequently, which is certainly true for software.



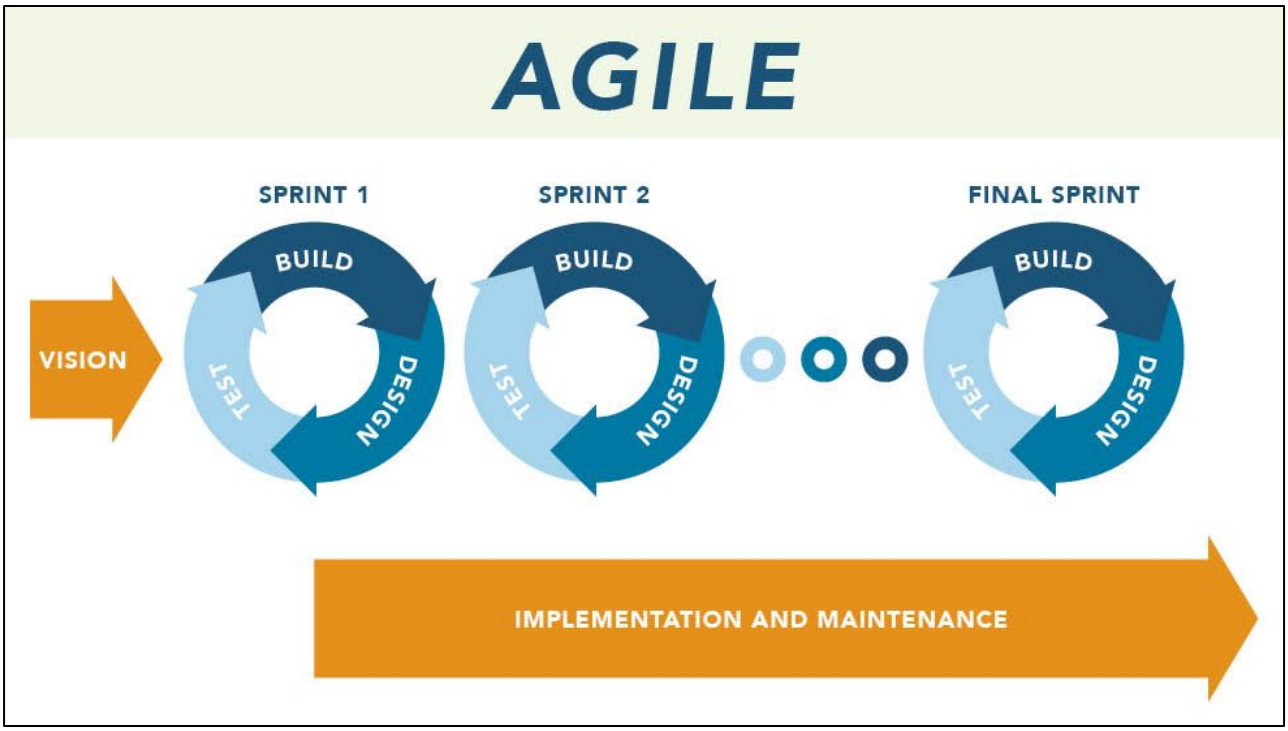
So, when I say “waterfall” what I mean is a classic “system development life cycle”, where you figure out what a system is going to do, design it, actually build it, make sure it works, and then finally delivery it to the customer.
If you remember the admonishment about “false incrementalism” earlier, you can see the fatal flaw in the waterfall methodology,



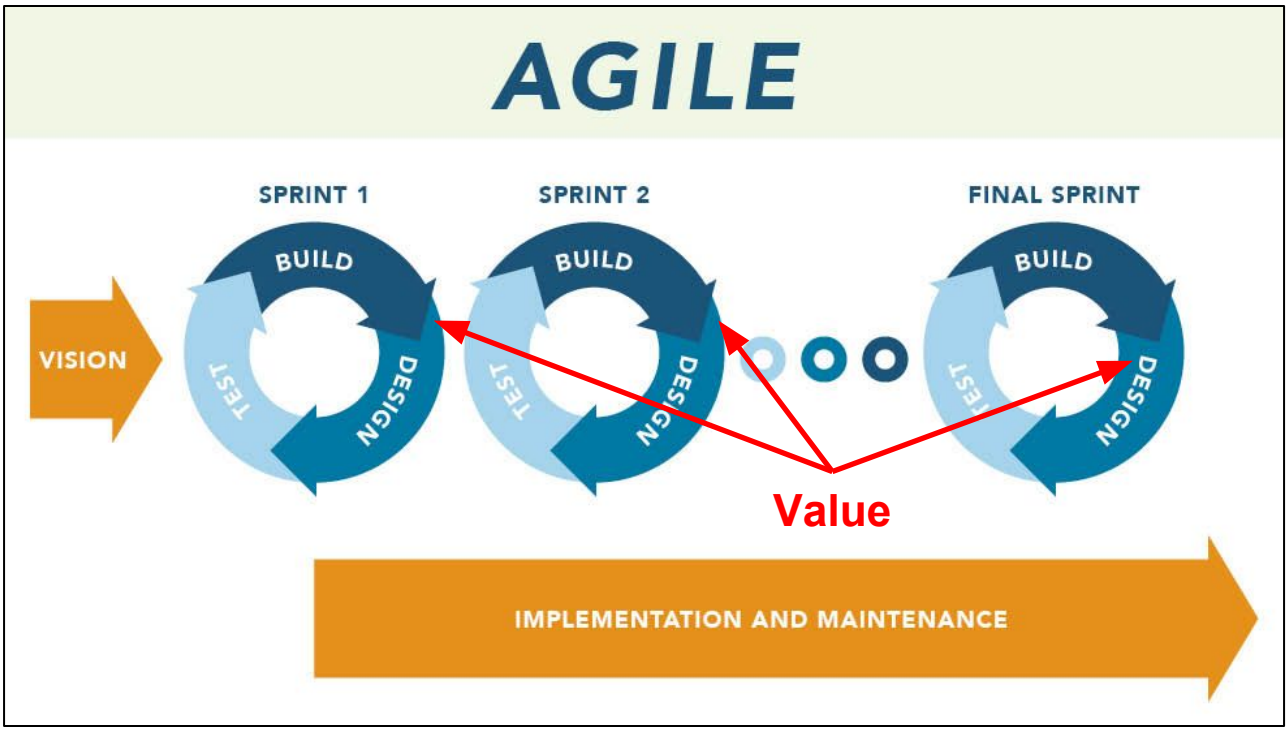
...which is that none of the business value is delivered until the final step.
Up until then, from the point of view of the business, you're just wanking, you're taking up their time and delivering nothing, so you'll slowly bleed away goodwill.
And If at any point along the line in earlier steps you have to stop, you've created nothing of business value.
It also assumes you'll always do everything right, that you won't discover new requirements along the way.



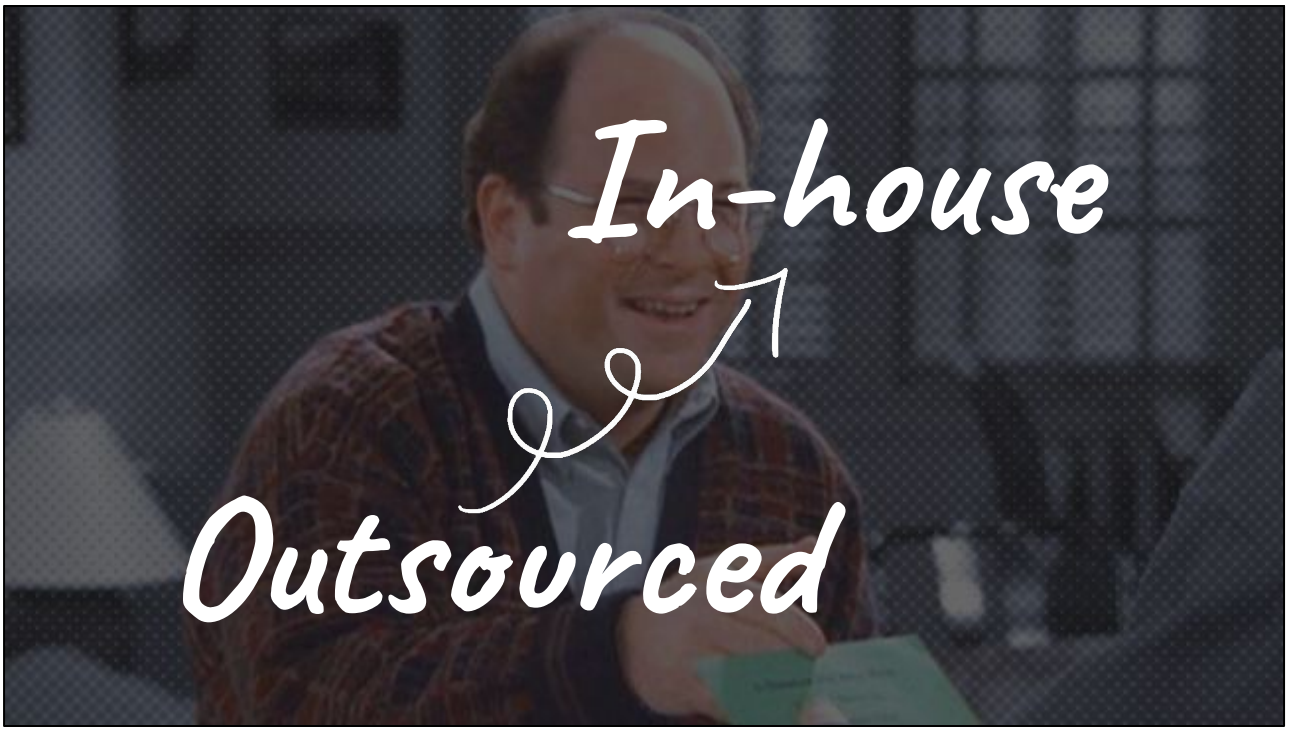
Like a real waterfall, the process is assumed to be basically one-way.
Each step leads naturally to the next.
Waterfall methodologies work, but they work best in fairly static operational environments.
If things don’t change much, in terms of requirements, or methods of implementation, or expectations for the final product, a waterfall process can be fine.
We build highways and bridges using waterfall methodologies, and it works fine.



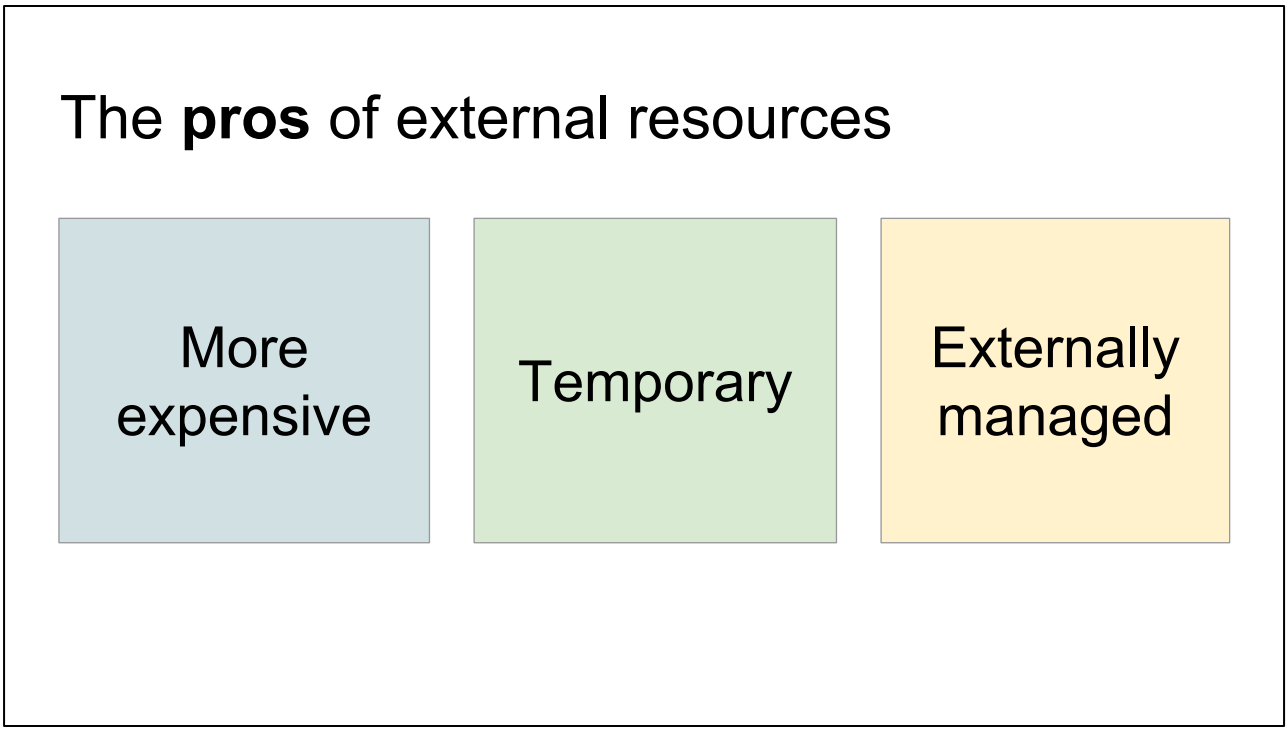
In an agile project, the goal is to get a working product in front of the users as soon as possible.
In the start-up world it's called a "minimum viable product", it does something useful and it's worth using.
From there, changes become truly incremental and are based on real feedback from users who actually use the product.
In a waterfall requirements process, users have to imagine they will need, and business value is delivered at the end.



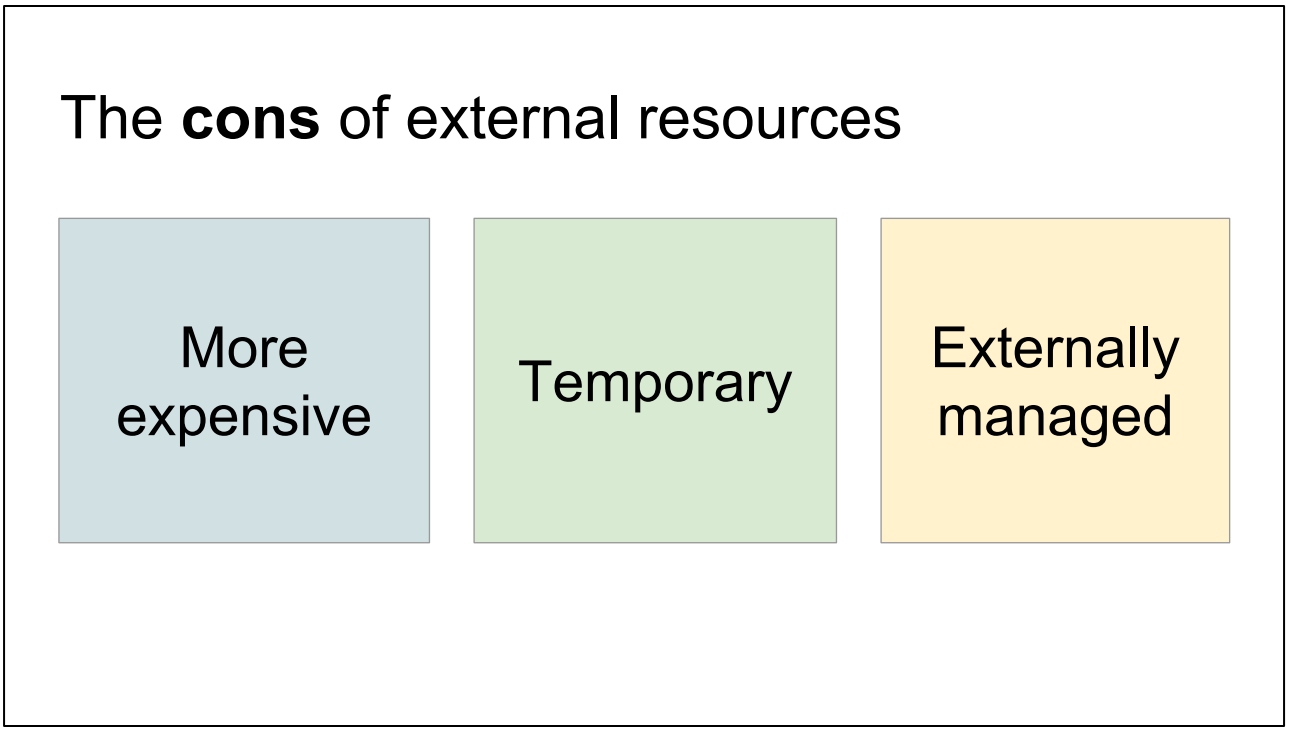
In an agile process, users experience what they are missing, and business value is delivered at each iteration.
User feedback helps inform what things are important **now**, and what can be done **later**.
There's lots of folks here who can speak far better than I about the benefits of agile methodology, it's already used in lots of projects in government, the challenge here is just to do it **more**, and make it the accepted default.



Next, outsourced or in-house.
There’s no doubt outsourced staff have some short term advantages over internal staff,



So, they are more expensive, which means you can pay more for “better” resources.
And, they are temporary, so you can scale your teams up and down to match your present needs.
And, they are externally managed, so hiring and firing and HR is relatively easy.
But it’s important to bear in mind, that they are also



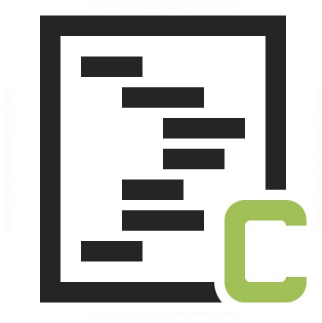
...more expensive, temporary and externally managed.

More expensive

- More dollars per hour means fewer hours per dollar
- Premium prices do not guarantee premium resources
- Everybody will take their cut

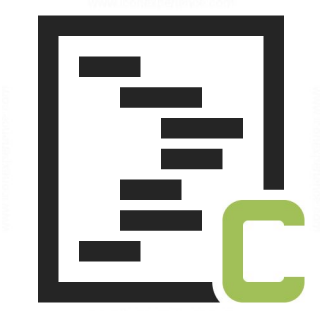
So, just simplistically if you pay more for time, you can get less time out of a fixed budget.
Which is fine, if you're guaranteed that in fact you're always getting the creme-de-la-creme, but brand name SI's do not always deliver on that promise.
And finally, a great deal of the premium price of external resources is eaten up by the middle-man. You may be paying 3x the internal cost of a resource in order to access a resource who is only being paid a 10% premium over the internal cost.
Yes, external resources are often paid more competitively, but rarely in proportion to the extra amount government pays to access them.

Temporary



Temporary.
When you form or hire a team, there's a tendency to look at their work in a very instrumental way.
They come in, they learn the problem, and they figure out a solution in the form of the system, that they deliver.
In that mental model, all the value gets deposited in the system. It's up and running, it works, so the team can leave and you retain all that value.

Temporary



A more realistic model recognizes that during system development and maintenance a great deal of the value is built up in the brains of the team members and is never deposited anywhere.
The understanding of the business, the surprising special cases,
the sociological aspects of relationships with other members of your organization,..
there is a huge store of intellectual capital related to your business that takes a lot of time (and therefore costs a lot of money) to build up...

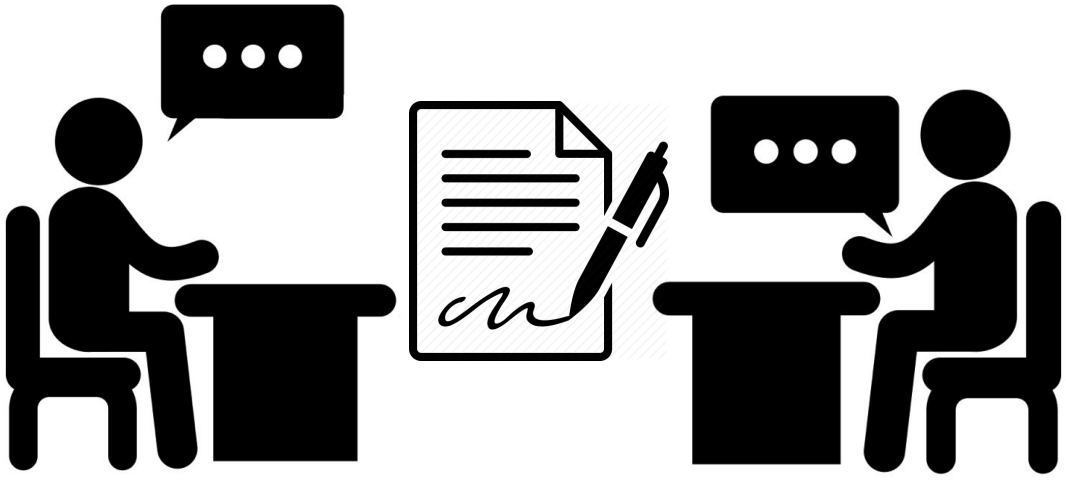
Temporary

...and then that value just walks out of the door when the temporary resources move on to their next project.
Any new resources brought in later have to be taught the things you **know** they need to know;
like the technology and the code which hopefully are all documented and transparent.
And then they have to learn all the things you **don't know** they need to know:
the relationships, the business quirks, the code quirks, the decisions that were make 3 years ago for reasons that made sense at the time.
That costs. And it costs in ways that are never captured in a capital budget request.

Externally managed

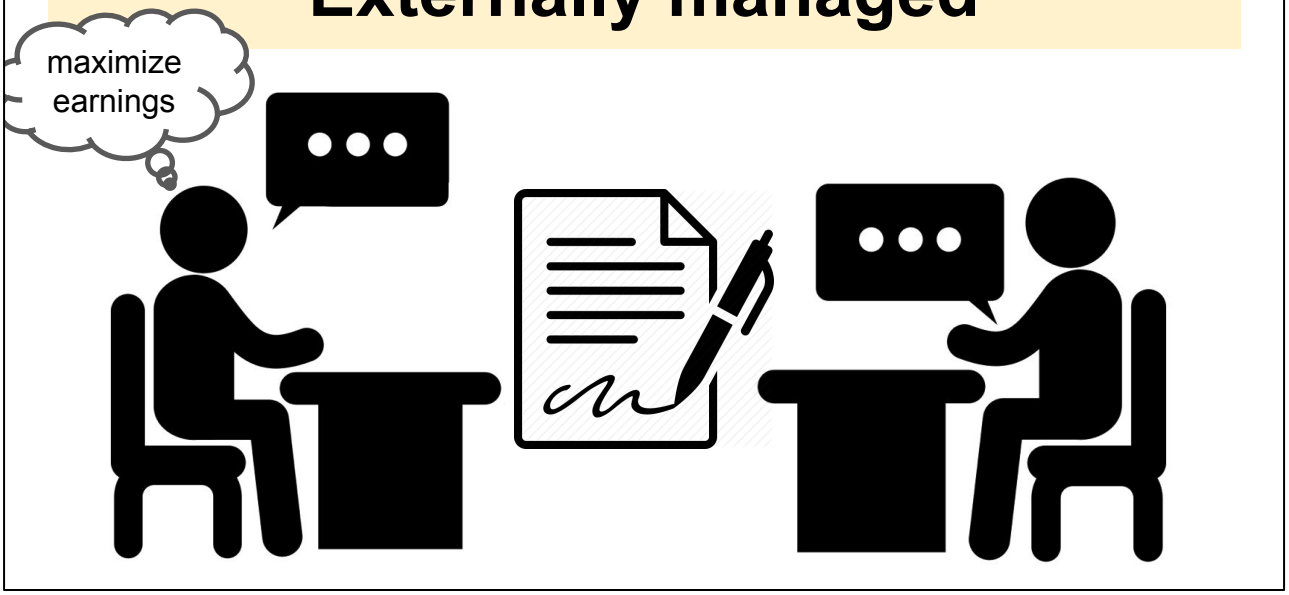
Finally, when the team is externally managed,
and in particular when the whole project is outsourced,
in a traditional design/build contract,
the communications between the people building the system,
and the people who will eventually receive delivery and have to use the system,

Externally managed

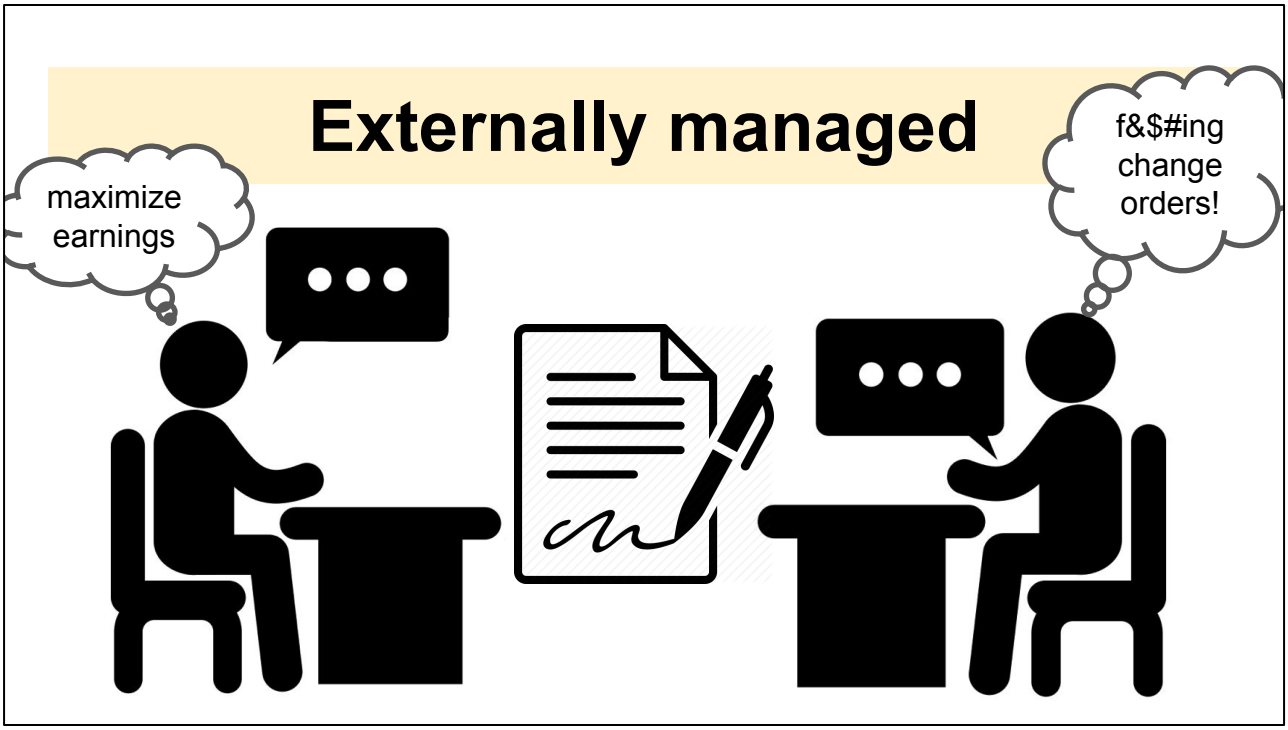


... all end up mediated by the project contract.
Everyone can try as hard as they want to pull together and deliver the best project possible,
but the initiative is always going to be strained by the fact that one side of the table

Externally managed

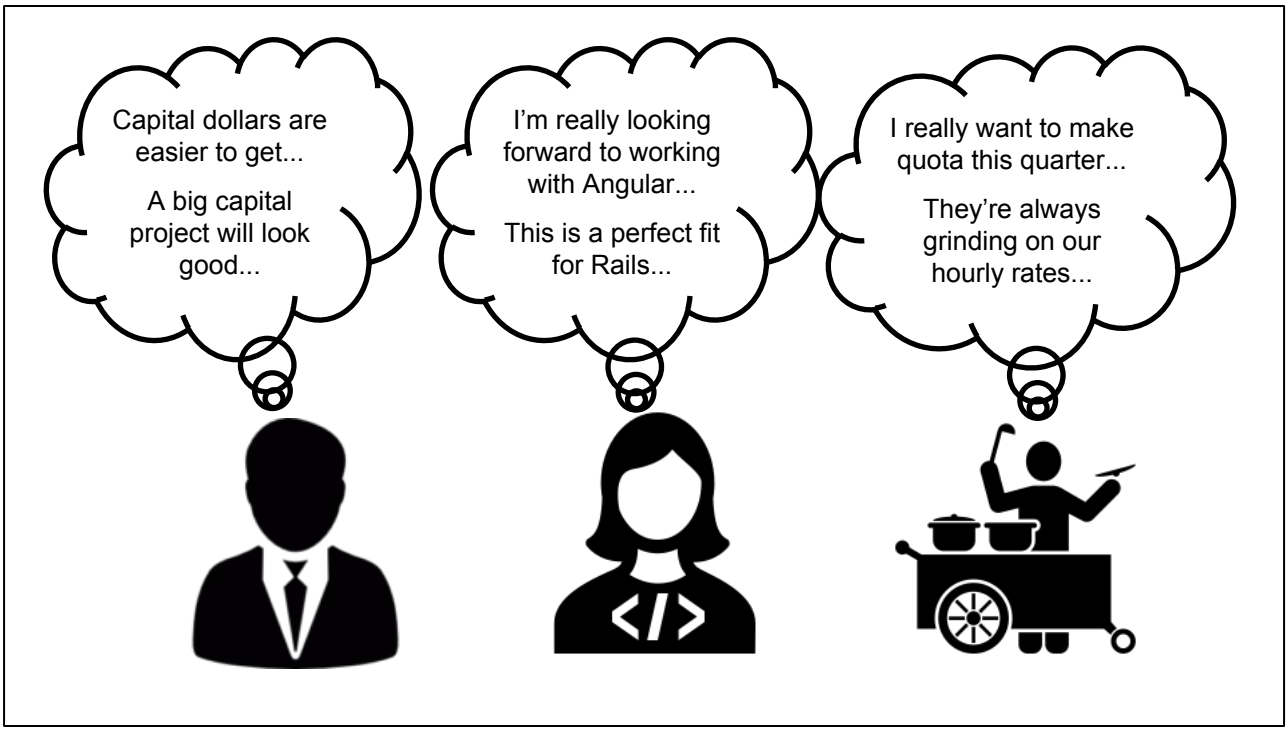


... is an organization whose primary duty is to maximize earnings.
Now, it's possible to maximize earnings with great and timely delivery, but it's also possible, and often easier, to maximize earnings through The aggressive use of change orders and contract leverage.



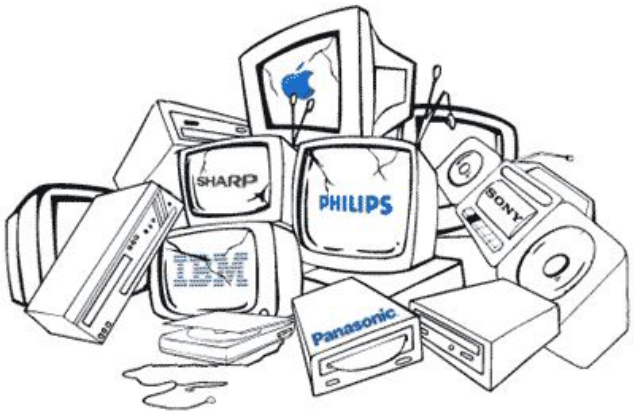
There's a reason why "change order" is a dirty word for almost anyone who has had to manage a large outsourced build.
Also from a personal perspective, one of the reasons I quit systems consulting was the sense that the path to success was paved with clever customer relationship management, not with great technical architecture or delivery.

Finally, changing the default prescription for aging systems from “replace” to “enhance”.
Oddly, I think this could be one of the hardest to achieve.
All the incentives in our system are lined up for replace



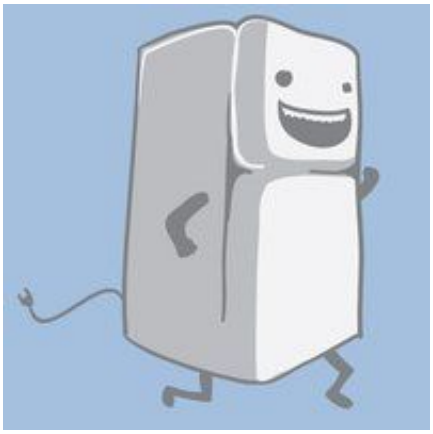
Capital dollars are easier to get than operating dollars,
There is managerial **glory** in accessing large capital envelopes,
Career-wise, a short stint building a system and then walking away when it's "done" is nicer than the long term grind of improvement,
The technology folks **always** want to work with the latest software and frameworks,
The vendors prefer to sell **new** software, and they prefer the murkier **pricing** opportunities that new builds provide,
So, yeah, **nobody** in the process is all that interested in "enhance".

Your “legacy” system...



We assign almost no value to our old systems, they’ve been paid for and depreciated,
they run on some crappy system **someone else** bought, before we arrived,
running software someone **stupider** than us wrote;
they’re monuments to **other people’s** dumb choices,
not our **own** bright shiny **correct** choices.
However, there is lots of value in that old system,
it just doesn’t show up on the balance sheet.

Your “legacy” system...



… is running right now.

- Reliability and predictability have value
- Already meeting a large fraction of your organizational needs
- Years of bug fixes and handling of surprise corner cases

First, it’s doing useful work right now. It probably has been for years.
Almost by definition, a long-lived system is already doing some good work for the organization.
And old code contains the bug fixes and enhancements of years.
The value of that experience, embodied in the code, is usually discounted when looking at old systems.

Your “legacy” system...

… is understood by users.

- They have muscle memory for it
- They know the strengths and limitations
- They know how to maximize their value from it

Second, the legacy system has a great deal of value stored in the heads of its users. Their knowledge of how to use and get the most from the system gets flushed away in a full replacement.
Product organizations that value their customers cater to that, in varying degrees. Things change, but slowly, incrementally.

This is a Bloomberg terminal from 1985.
A trading floor would pay $20,000 per year, per trader to put one in front of each trader.
Look at the keyboard. Note the color code function keys..
You can’t see it, but at the top, is an input field for typing in commands.

This is a contemporary Bloomberg terminal.
Still $20,000 per seat per year.
Look at the keyboard.
The terminal still has the same input field and command language, because the traders demanded it.
Mice are nice for leisurely data exploration, but for high speed retrieval of things you already know, nothing beats a keyboard.
Compared to a modern UX design, it’s ugly. But it’s functional, because it’s evolved over time with very demanding users.
No re-writes for the Bloomberg terminal.

Your “legacy” system...

… is full of valuable data.

- Rewrites tend to focus on code, data will “be copied in later”
- Data is most valuable part of any system

There’s a tendency when looking for a new shiny,
to ignore the data that resides in the old system,
or to wave away the need to understand the data.
“At the end we’ll just migrate the old data.”
There’s more value in the old data than there will be in any new system.
Information systems are just conveniences for managing data,
so retaining and protecting that data should be the central concern.

Your “legacy” system...

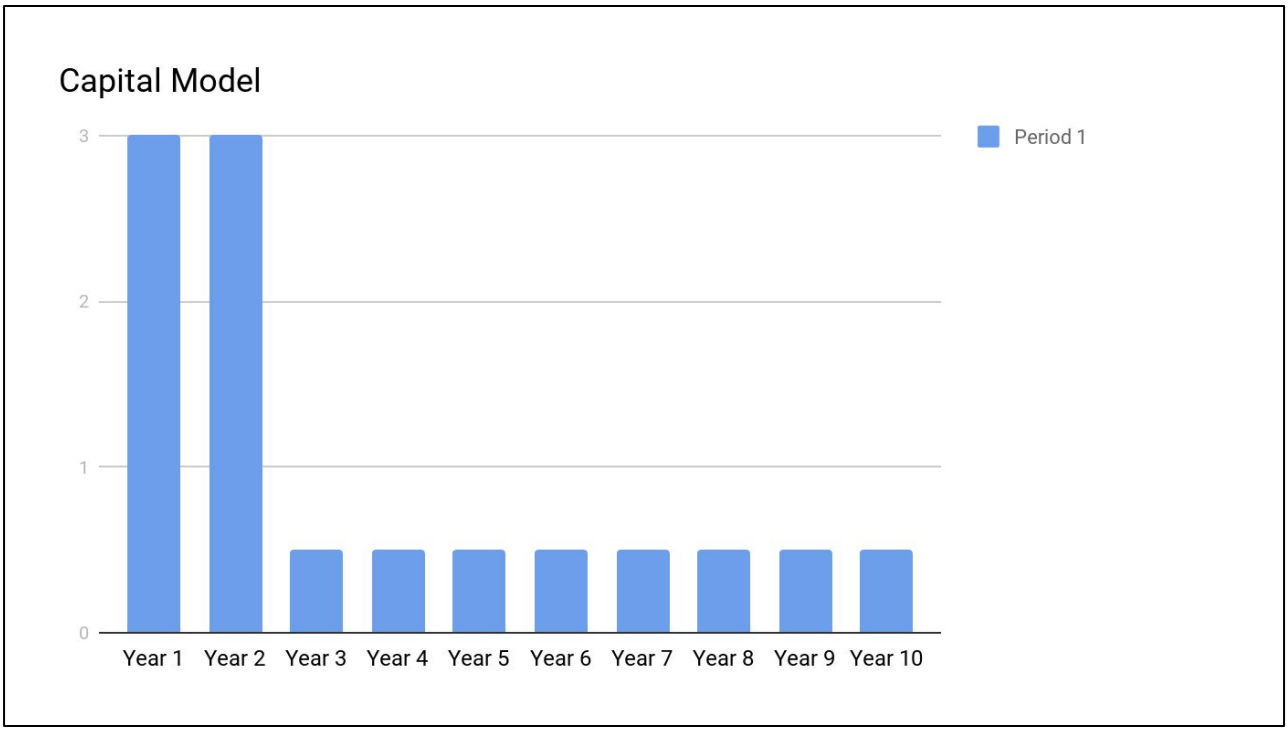
… can be incrementally improved.

- Legacy system is the MVP, already installed and running
- Rewriting feels good for techs, does it add business value?

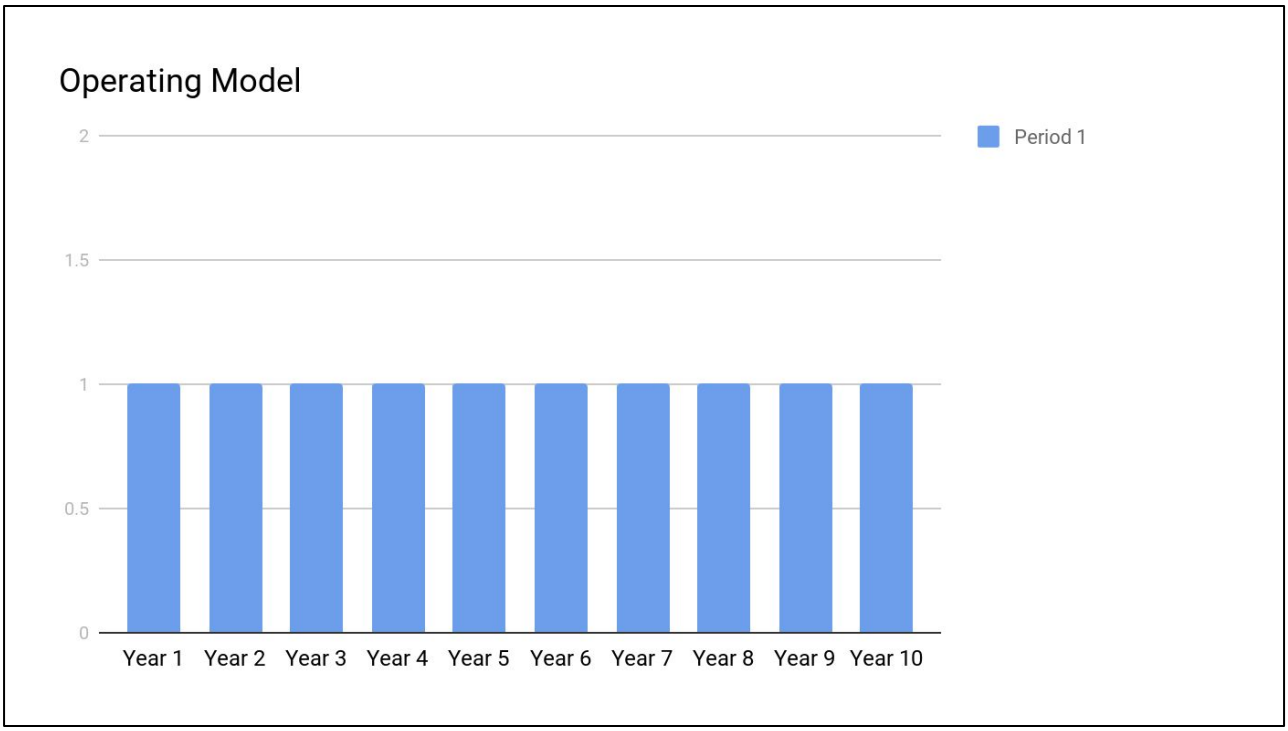
Finally, a legacy system offers a perfect jumping off point for incremental improvement.
It’s a Minimal Viable Product, already stood up and deployed.
What needs to be added, what needs to be changed?
I talk about enhance versus replace, but it’s almost a proxy term,

Replace	Enhance
Capital	Operating
Intermittent	Continuous

Where replacement is associated with the intermittent application of capital dollars to the problem of making a better system.
And enhancement is associated with the continuous application of operating dollars to the problem of making a better system.
And what I keep coming back to is, if I have $10M devoted to having a good IT system available to my business over 10 years.

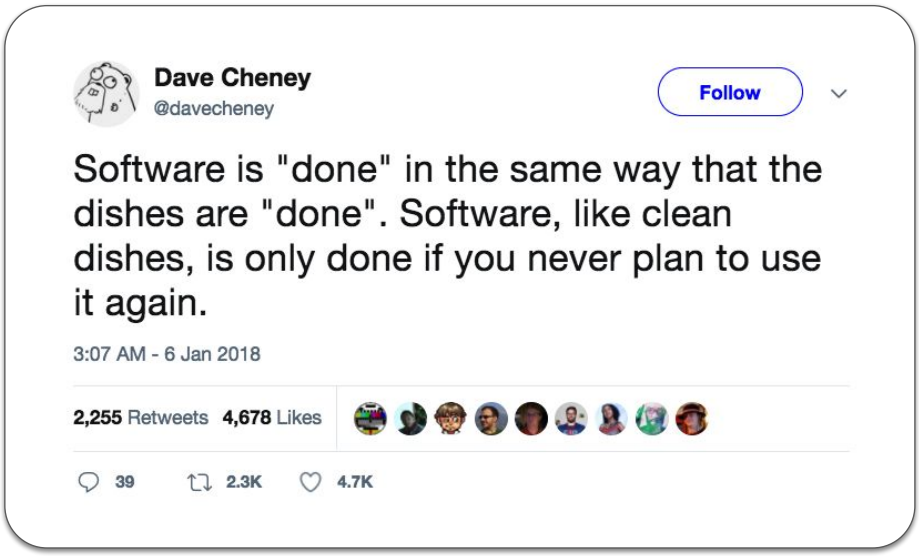


What model would be most likely to lead to a sustainable and flexible system? Spending $6M up front in a design and build, and then cutting back to a skeleton operations team for the remaining years?



Or just maintaining a team large enough to both operate and improve the system as a constant expenditure over 10 years.

If we can generally agree that the up-front model is riskier on almost every axis, that it sets systems up for failure, that as IT practitioners, we'd **rather not do it…** then that's a good sign that the capital model is broken for IT systems, that maybe IT systems aren't "capital assets" after all, and we should be talking to the accountants about how to get out of this mess.

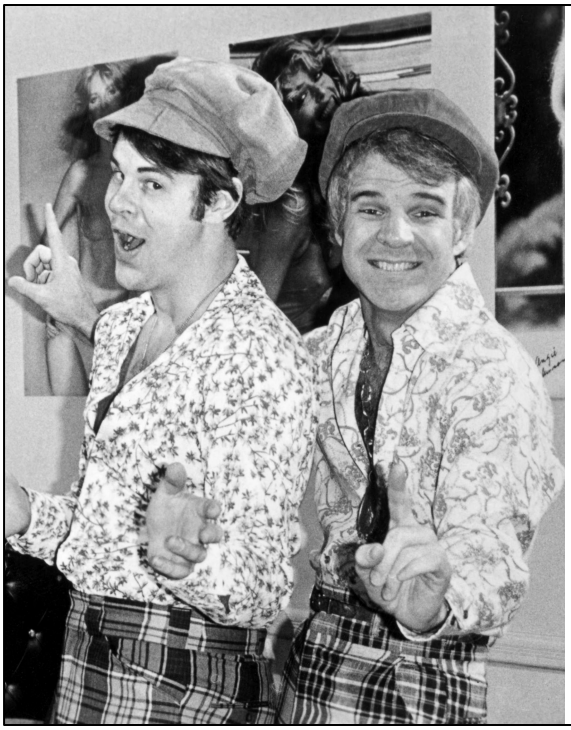


The idea that software is rolled off the loading dock, plugged in, and then quietly depreciated for a decade is really weird, and I don't know how the accountants ever got it into their heads in the first place. Maybe the shrink wrapped vendors promoted the idea, so they could make sales into capital budgets, but for most business systems, it seems like a very bad fit.

Anyhow, that's the Costanza plan, not a sudden switch, but a transition,
To smaller, open source, transparent agile projects, building knowledge in in-house teams, with a bias towards enhancing existing systems incrementally.

So.
Now what?

Well, now you have to figure out how to actually do that.
Steve Martin has some advice on front.

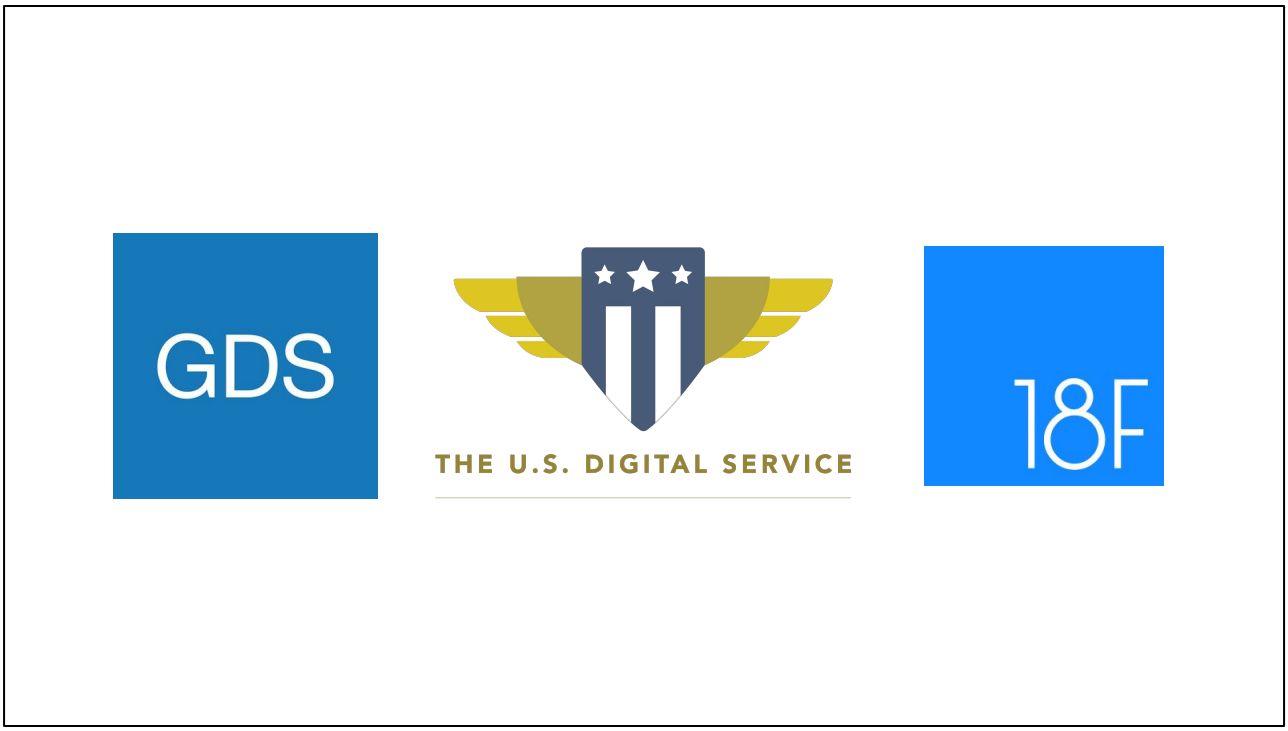


- Be **wild** and **crazy** guys!
  - There are **other** wild and crazy guys in government
  - There is **strength** in numbers

First, be wild and crazy guys.
Own your wildness, own your craziness.
You are not alone, there's folks already doing this stuff.
Centrally there's the Continuous Service Improvement lab, which is already using these methodologies and working to formalize the "right way to do this in government without breaking rules".
Each Ministry has groups doing pieces of this, open source, agile methodologies, in-house teams.

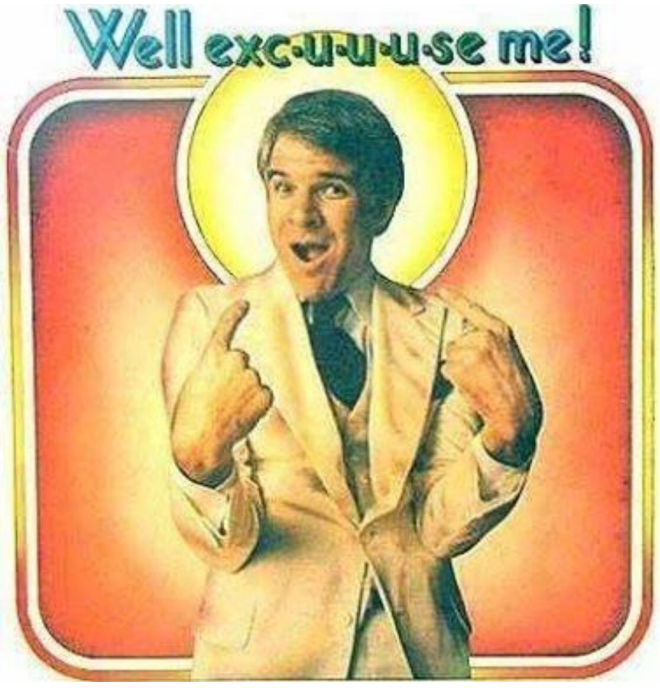
- Get **happy feet**!
    - Dance around and **tell people** how fun dancing around is
    - **Culture change** is more important than org charts

If you talk to each other about what you are doing, you can cross-pollinate your success stories, and learn from your failures.
One of the frustrations I have with government IT is how insular the various business silos tend to be.
You're all solving the same fundamental problems, there should be a lot more movement, of staff and ideas, back and forth.
Why aren't Health IT professionals doing time in Natural Resources and vice versa?



By all means, look to GDS, USDS and others for materials on systems design and continuous improvement and user experience methodologies, but don't get hung up by the fact that other governments have chosen to build centralized organizations to promote change.

Central agencies are great at concentrating enthusiasm and focussing it out like a laser, it's very exciting, but in doing that concentrating they also can reduce enthusiasm for change at operational levels. There's a potential trade-off and it's not clear there's any one right way to introduce these ideas.



- Be polite!
- If your insistence on change offends people, be sure to say, “**excuuuse me**”.

Finally, be persistent, and polite. But remember that you’re in the right.
Things as they stand aren’t too great, there’s a need for change, and there’s a need for people to stand up and say so.
And if you don’t think so, well excuuuuuuuse me.

Thanks.
There are links to this presentation and to a bibliography of interesting reading, at the link above.