Why Open Software Matters for Government & Civic Tech

[and how to support it]

by

Rufus Pollock[1]

OPEN KNOWLEDGE

Does free/open source software matter for government and civic tech? Matter in the sense that it should have a deep and strategic role in government IT and policy rather than just being a "nice to have" or something "we use when we can"?

The answer is yes, open source software matters for government and civic tech – and, conversely, government matters for open source. This report shows how and why, covering:

- Why open software is especially important for government and civic tech
- Why open software needs special support and treatment by government
- What specific actions can be taken to provide this support for open software

We also discuss how software is different from other things that government traditionally buy or fund. This difference is why government cannot buy software like it buys office furniture or procures the building of bridges – and why buying open matters so much.

[1] President and Founder of Open Knowledge and Associate at the Centre of Intellectual Property and Information Law, University of Cambridge. I'm grateful to many people with whom I've had the opportunity to discuss these topics over the years. I also wish to acknowledge funding from Open Society Foundations that supported the drafting of the first version of this report. First draft March 2015, this version July 2016.

Table of Contents

Executive Summary

This executive summary consists of two parts. First, a summary of why open software is important in government and why it should be explicitly supported and promoted. Second, suggestions and recommendations for the steps that government and funders can take to promote open software in government and civic tech.

Why Open Software

We begin with four facts about software and government which are a basis for the conclusions and recommendations that follow.

1. **The economics of software: software has high fixed costs and low (zero) marginal costs and it is also incremental in that new code builds on old**. The cost structure creates a fundamental dilemma between finding ways to fund the fixed cost, e.g. by having proprietary software and raising prices; and promoting optimal access by setting the price at the marginal cost level of zero. In resolving this dilemma, proprietary software models favour the funding of fixed costs but at the price of inefficiently raised pricing and hampering future development, whilst open source models favour efficient pricing and access but face the challenge of funding the fixed costs to create high quality software in the first place. The incremental nature of software sharpens this dilemma and contributes to technological and vendor lock-in.
2. **Switching costs are significant: it is (increasingly) costly to switch off a given piece of software once you start using it**. This is because you make "asset (software) specific investments": in learning how to use the software, integrating the software with your systems, extending and customizing the software, etc. These all mean there are often substantial costs associated with switching to an alternative later.
3. **The future matters and is difficult to know**: software is used for a long time – whether in its original or upgraded form. Knowing the future is therefore especially important in purchasing software. Predictions about the future in relation to software are especially hard because of its complex nature and adaptability; behavioural biases mean the level of uncertainty and likely future change are underestimated. Together these mean lock-in is under-estimated.
4. **Governments are bad at negotiating, especially in this environment, and hence the lock-in problem is especially acute for Government**. Government are generally poor decision-makers and bargainers due to the incentives faced by government as a whole and by individuals within government. They are especially weak when having to make trade-offs between the near-term and the more distant future. They are even weaker when the future is complex, uncertain and hard to specify contractually up front. Software procurement has all of these characteristics, making it particularly prone to error compared to other government procurement areas.

The Logic of Support

Note: numbers in brackets e.g. (1) refer to one of the four observations of the previous section.

A. Lock-in to Proprietary Software is a Problem

Incremental Nature of Software (1) + Switching Costs (2)
imply ...
Lock-in happens for a software technology, and, if it is proprietary, to a vendor

Zero Marginal Cost of Software (1) + Uncertainty about the Future both user needs and technology changes (3) + Governments are Poor Bargainers (4)
imply ...
Lock-in to proprietary software is a problem
Lock-in has high costs and is under-estimated - and especially so for government

B. Open Source is a Solution

Lock-in is a problem
imply ...
Strategies that reduce lock-in are valuable

Economics of Software (1)
imply ...
Open-source is a strategy for government (and others) to reduce future lock-in
Why? Because it requires the software provider to make an up-front commitment to making the essential technology available both to users and other technologists at zero cost, both now ***and*** *in the future*

Together these two points
imply ...
Open source is a solution
And a specific commitment to open source in civic tech is important and valuable

C. Open Source Needs Support

And Government / Civic Tech is an area where it can be provided effectively

Software has high fixed costs and a challenge for open source is to secure sufficient support investment to cover these fixed costs (1 - Economics)
+
Governments are large spenders on IT and are bureaucratic: they can make rules to pre-commit

up front (e.g. in procurement) and can feasibly coordinate whether at local, national or, even, international levels on buying and investment decisions related to software.

imply …

Government is especially well situated to support open source
AND
Government *has* the tools to provide systematic support
AND
Government *should* provide systematic support

Promoting Open Software for Civic Tech

We have established in the previous section that there is a strong basis for promoting open source for civic tech. This section focuses on some specific strategic and tactical suggestions for achieving this goal of promoting open source. There are four proposals that we summarize here. Each of these is covered in detail in the main section below. We especially emphasize the potential of the third option as it does not require up-front participation by government and can be boot-strapped with philanthropic funding.

1. Recognize and reward open source in IT procurement.

Give open source explicit recognition and beneficial treatment in procurement. Specifically, introduce into government tenders: EITHER an explicit requirement for an open source solution OR a significant points value for open source in the scoring for solutions (more than 30% of the points on offer).

2. Make government IT procurement more agile and lightweight.

Current methodologies follow a "spec and deliver" model in which government attempts to define a full spec up front and then seeks solutions that deliver against this. The spec and deliver model greatly diminishes the value of open source - which allows for rapid iteration in the open, and more rapid switching of provider - and implicitly builds lock-in to the selected provider whose solution is a black-box to the buyer. In addition, whilst theoretically shifting risk to the supplier of the software, given the difficulty of specifying software up front it really just inflates upfront costs (since the supplier has to price in risk) and sets the scene for complex and cumbersome later negotiations about under-specified elements.

3. Develop a marketing and business development support organization for open source in key markets (e.g. US and Europe).

The organization would be small, at least initially, and focused on three closely related activity areas (in rough order of importance):

1. General marketing of open source to government at both local and national level: getting in front of CIOs, explaining open source, demystifying and derisking it, making the case etc. This is not specific to any specific product or solution.
2. Supporting open source businesses, especially those at an early-stage, in initial business development activities including: connecting startups to potential customers ("opening the rolodex") and guidance in navigating the bureaucracy of government procurement including discovering and responding to RFPs.
3. Promoting commercialization of open source by providing advice, training and support for open source startups and developers in commercializing and marketing their technology. Open source developers and startups are often strong on technology and weak on marketing and selling their solutions and this support would help address these deficiencies.

4. Open Offsets: establish target levels of open source financing combined with a "offsets" style scheme to discharge these obligations.

An "Open Offsets" program would combine three components:

1. Establish target commitments for funding open source for participants in the program who could include government, philanthropists and private sector. Targets would be a specific measurable figure like 20% of all IT spending or $5m.
2. Participants discharge their funding commitment either through direct spending such as procurement or sponsorship **or** via purchase of open source "offsets". "Offsets" enable organizations to discharge their open source funding obligation in an analogous manner to the way carbon offsets allow groups to deliver on their climate change commitments.
3. Administrators of the open offset fund distribute the funds to relevant open source projects and communities in a transparent manner, likely using some combination of expert advice, community voting and value generated (this latter based on an estimate of the usage and value of created by given pieces of open software).

5. "Choose Open": a grass-roots oriented campaign to promote open software in government and government run activities such as education.

"Choose Open" would be modelled on recent initiatives in online political organizing such as "Move On" in the 2004 US Presidential election as well as online initiatives like Avaaz. It would combine central provision of message, materials and policy with localized community participation to use these to drive change.

Key Concepts

Civic tech

Civic tech is digital technology (websites, apps and platforms) related to “civic” life. As defined, it includes most notably all government IT (whether in-house or procured) but it also includes technology created by others directly for citizens - from apps that let you register to vote to those that let you email your elected representative.

These two streams of "civic tech" are increasingly close from the perspective of end users – for citizens the distinction between a "where to vote" app produced by a third party such as Google or a non-profit and an app produced officially may be in minimal. However, from a policy, funding and management perspective the difference between "government IT" – whether built, bought in or outsourced – and all other civic technology is large.[2]

Thus, for our purposes, we distinguish the two areas. Moreover, our focus will largely be on the government area, given its vastly larger size and its systemic policy and process role.

Free / Open Source Software

Throughout I will use open source software or open software as shorthand for free/open source software (or "libre" software).

Open source software is defined precisely by the Open Source Definition at http://opensource.org/. In summary, it is software that everyone has freedom to use, modify and share without need to seek permission or make payment – whoever they are and whatever their purpose.

Informal open sharing has been a constant feature of the software community from its earliest days in the 1950s. However, it was only in the 1980s that free and open source software developed as a formal formal idea and the first free and open source licenses were created (the GNU Public License).

Open source has now spread rapidly beyond its roots in academia, and today it dominates major areas such as Internet tools and it has seen widespread adoption in business. Most people in their daily browsing of the web use an open source web browser such as Firefox or Chrome and the majority of smartphones run on the open source Android operating system.

[2] Government is still running or buying the vast majority of the "civic tech" even in the biggest pipe-dreams of the "government as a platform" evangelists.

All that said, open source still struggles in many areas, and uptake in government remains relatively low. At a policy level, there is also limited support for open source: few, if any, governments have an explicit open source preference in procurement. In addition, government and philanthropic funding of software often fails to require open sourcing the outputs.

Software Services and Open Services

Increasingly we do not purchase a given software application but we interact with an online "service" in the “cloud”. Ignoring online services and focusing only on open "software" in a narrow sense would exclude a large and growing area of information technology and especially civic tech. Thus, whenever you see open source or open source software mentioned, consider it to include online applications and services.

We should therefore be clear what we mean by "open" (open source) service. An open service, as per the Open Service Definition – http://opendefinition.org/ossd/ – is a service whose entire application code is open source and where any user's data can be extracted, and migrated, quickly, easily and at no charge to another instance of the same application (though not necessarily other, different applications).

Why Open Source

Introduction

Open source software is free today, free in future, and provides freedom of choice today and in the future regarding both vendor and mode of implementation (e.g. on premise vs cloud, self-managed vs vendor managed etc).

By contrast, proprietary software is often expensive, has clear future purchase costs and commits one to a given vendor both now and into the future.

Given this it would seem obvious that open source is always preferable. However, there are problems with such a simple comparison.

First, open source software is "free" only in a limited sense. It is free in that, unlike proprietary software, there is no license fee. However, this is only part of the story. The cost of software is not just the licensing fee: there is the cost of setup and installation, the cost of maintenance (bug-fixing, upgrades), costs of training, etc. The full price for software is some combination of all these costs and the term "total cost of ownership" is often used. In this sense, the price of open source is not zero, since it will still have these ancillary costs even if there is no license fee.

Second, we need to account for quality (features, reliability etc). Two different pieces of software may differ substantially in quality even when performing very similar functions. In that sense, we need to consider quality and price together – one piece of software may be more expensive than another, but it may also be much better.

In theory, we could try to estimate the cost and quality for open source solutions and compare them with proprietary ones. Long debates have been had over the "total cost of ownership" for open source vs alternatives. The issue is that, unlike the license fee which is well-defined, the other costs that make up the total-cost of ownership are often hard to estimate and subject to a significant degree of judgment (and debate).

However, there are substantial uncertainties associated with this kind of empirical analysis, and it necessarily focuses on only a few particular pieces of software or technology. Thus, rather than pursue this route, we will instead focus on principled approach based on a few key facts that together yield some clear conclusions.

Four Facts and the Logic of Open Source

We proceed from four facts about the way in which software and government work that are important for government IT policy. These are as follows:

1. **The economics of software: software has high fixed costs and low (zero) marginal costs and it is also incremental in that new code builds on old**. The cost structure creates a fundamental dilemma between finding ways to fund the fixed cost, e.g. by having proprietary software and raising prices; and promoting optimal access by setting the price at the marginal cost level of zero. In resolving this dilemma, proprietary software models favour the funding of fixed costs but at the price of inefficiently raised pricing and hampering future development, whilst open source models favour efficient pricing and access but face the challenge of funding the fixed costs to create high quality software in the first place. The incremental nature of software sharpens this dilemma and contributes to technological and vendor lock-in.
2. **Switching costs are significant: it is (increasingly) costly to switch off a given piece of software once you start using it**. This is because you make "asset (software) specific investments": in learning how to use the software, integrating the software with your systems, extending and customizing the software, etc. These all mean there are often substantial costs associated with switching to an alternative later.
3. **The future matters and is difficult to know**: software is used for a long time – whether in its original or upgraded form. Knowing the future is therefore especially important in purchasing software. Predictions about the future in relation to software are especially hard because of its complex nature and adaptability; behavioural biases mean the level of uncertainty and likely future change are underestimated. Together these mean lock-in is under-estimated.
4. **Governments are bad at negotiating, especially in this environment, and hence the lock-in problem is especially acute for Government**. Government are generally poor decision-makers and bargainers due to the incentives faced by government as a whole and by individuals within government. They are especially weak when having to make trade-offs between the near-term and the more distant future. They are even weaker when the future is complex, uncertain and hard to specify contractually up front. Software procurement has all of these characteristics, making it particularly prone to error compared to other government procurement areas.

Each of these will be elaborated on in detail below. First, however we look at the conclusions that follow:

Note: numbers in brackets e.g. (1) refer to one of the four observations of the previous section.

A. Lock-in to Proprietary Software is a Problem

Incremental Nature of Software (1) + Switching Costs (2)
imply ...
Lock-in happens for a software technology, and, if it is proprietary, to a vendor

Zero Marginal Cost of Software (1) + Uncertainty about the Future both user needs and technology changes (3) + Governments are Poor Bargainers (4)
imply …
Lock-in to proprietary software is a problem
Lock-in has high costs and is under-estimated - and especially so for government

B. Open Source is a Solution

Lock-in is a problem
imply …
strategies that reduce lock-in are valuable

Economics of Software (1)
imply …
Open-source is a strategy for government (and others) to reduce future lock-in
Why? Because it requires the software provider to make an up-front commitment to making the essential technology available both to users and other technologists at zero cost, both now ***and*** *in the future*

Together these two points
imply …
Open source is a solution
And a specific commitment to open source in civic tech is important and valuable

C. Open Source Needs Support

And Government / Civic Tech is an area where it can be provided effectively

Software has high fixed costs and a challenge for open source is to secure sufficient support investment to cover these fixed costs (1 - Economics)
+
Governments are large spenders on IT and are bureaucratic: they can make rules to pre-commit up front (e.g. in procurement) and can feasibly coordinate whether at local, national or, even, international levels on buying and investment decisions related to software.

imply …

Government is especially well situated to support open source
AND
Government has the tools to provide systematic support

Economics of Software

There are three key aspects of the economics of software that we highlight:

1. High fixed, low (zero) marginal costs. Software is digital and therefore approximately costless to reproduce (once you have a first copy).[3] At the same time the cost of producing the first copy can be very high.
2. Software systems are often platforms on which other systems build or to which they interconnect. This - plus the first point - creates (indirect) network effects for software: a user of a certain software is benefitted by the use of that software by others (in the same way that with direct network effects a user of a classic network like a telephone system gets benefit from other users on their network).
3. Software is incremental: you can build on what you already did, improving and extending it.

From these observations derive the following implications.

1. Open vs Closed Dilemma

First, the low marginal / high fixed cost nature of software presents us with a dilemma: if we supply (efficiently) at marginal cost how will we fund the fixed cost?

The open source option resolves this one way: supply software at marginal cost (zero). However, the challenge then is to find sufficient funds to cover the fixed costs and we risk “under-investment” – software not being created or of being of lower quality then we wish.

[3] Software is just digital bits that tell a computer "what to do". Digital bits are costless to copy. At the same time, software has large up-front fixed costs to create the first copy. So, software has large fixed costs together with very low "marginal" costs (that is the cost of producing an additional copy or serving an additional user). Put more simply: if I develop a new application, the costs of that development are the same whether I serve 1 user or 1 million. Implications include:

- initial software development is risky especially if you want a reasonably polished product - you have to invest quite a lot before you get something useful and you bear those costs before you have many users.
- If you are successful you then make a lot of money as the costs of additional users is very low so what each new user pays is pretty much pure profit. Software development is thus similar to the film or music industry which also have large up front costs (making the record, making the film) and low marginal costs for each user. Like those industries software is therefore "hit-oriented" - most things never make it in a big way, and a few go massive. It is also why there is so much VC (venture capital) in software compared to, say, ice-cream parlours. Software needs investors to bear the up-front investment risk in return for a share of a, possible, massive win.
- Once you have software it is *feasible* - and, in fact, economically "optimal" - to provide it to every potential user (since costs of provision are zero). This makes software quite different compared to "normal" goods like cars or food or even laptops. Economists call things like software "nonrival" - as my use of it (making a copy) does not affect your use of it - our use is not "rival". Traditional, physical goods are "rival".

Finally, what about software "services"? Services are a bit different. With services we do not have just the software: we also have the hardware and support staff to run the service. There are still "economies of scale": one support staff member plus a server can provide for, say, 10,000 users and so the service costs do not vary between 1 and 10,000 users. Nevertheless, software services are more hybridized and, on the non-software side, have more traditional economic characteristics.

The proprietary option resolves this the other way: use copyright (and patents) to create "excludability" limiting access and making production less efficient but generating more money for investment. Funding production is easier and so more software, of a higher quality, may be created. However, this comes at the cost of higher prices which mean that some users never get to use the software (even though they could be supplied). In addition, exclusion prevents reuse by future innovators and creators retarding innovation.

2. Path Dependence and Lock-In
The second implication is around path dependence and lock-in. Points 2 (platforms) and 3 (incrementality) imply there is path dependence and increasing (technological) lock-in[4] to previously selected software: as time goes by increasingly hard to enter a given software market as existing software has a huge lead.[5]

3. The "Market" Does Not Help (that much)
Third, these economic characteristics provide an immediate response to market fundamentalists who argue that we can just let the "market" sort it out -- that is: market forces will produce an optimal outcome. The economics of software mean that a simple "market-based" outcome will never be optimal.[6] In a perfect world, you would find a way to fund the fixed costs of the software (that you wanted) and then supply it to everyone openly for free (this would essentially be open source with efficient up front funding).

The Challenge for Open Source
Given these economics, open source has a challenge. Being free to use and free to copy, open source suppliers are able to capture less of the value generated by their product (users get more of it!). In addition, open source is more competitive – if you create an open source product anyone else can come in and supply it too. This means the present value of all income, present and future, for open source is much lower because income is both lower and more risky.[7] This means open source has a much harder time obtaining funding compared to proprietary options - whether that funding is out of its own cash flows or from investors willing to make up-front

[4] Technological lock-in is different from switching-costs lock-in but the two reinforce. Strictly, switching costs lock-in is between a particular user and the specific software they have chosen. Technological lock-in is about the fact that when a particular software or technology is chosen by people today it increases its likelihood of being chosen (by anyone) tomorrow.
[5] The switching costs section (see the next section) and this point about lock-in here are distinct but they reinforce each other: path dependence reduces choice in the future and mean that the power/impact of proprietary control of a successful software platform are very large.
[6] For the economists out there, classic market arguments rely on "convexity" of production functions; the production function is very non-convex as you have large fixed costs and then zero marginal costs. In addition, as we have detailed these markets clearly are incomplete and participants have incomplete information plus various behavioural biases.
[7] A riskier income stream means a higher discount rate for future income. Open source future cash flows are riskier for an investor because the open source nature means that, even if the software is successful, there is significant risk that other firms can enter (the software is open source!) and provide solutions based on that software in competition with the original developers.

payments in exchange for rights to future income.

Of course, open source also has commercial benefits. For example: because the source code is visible, it is much easier for third parties other than the original developers to contribute and extend. It is also more attractive to third-parties to do so as there is no hold-up problem. This both lowers the cost of production of open source software for the original developers and increases opportunity for commercial reuse and exploitation.[8] However, the disadvantage on the revenue side almost certainly outweighs the advantages on the cost side - at least, at the key early stages of software development.[9]

This leads to a common dynamic where users choose a proprietary option early on, since it is better as a result of all that up front investment that proprietary providers can afford. But later, users find themselves then locked in and tend to have underestimated the extent of this lock-in. Later, when open source may be comparable in quality – or even better, it has to overcome this high level of lock-in.

Switching Costs and Lock-in

The basic idea here is obvious: as you use software, you make investments that gradually increase your switching costs to an alternative. This makes it increasingly difficult to switch to an alternative and you end up "locked-in" to the existing software solution and vendor. Note that technology and vendor lock-in are often considered the same: that is you are locked-in both to a given piece of software and the vendor who provides it. This is certainly the case for proprietary software. However, for open source the two are different: because the software is open and therefore freely available to anyone - including other vendors - there is no vendor lock-in.[10]

This lock-in is especially significant when software is a "platform" - where you - or others - plan build on it. Platforms are especially strong on lock-in because you or others make substantial investments in other software or tools that build on (are "complementary") to that platform. A good example is operating systems like Windows: an operating system is really useful for the software that runs on that operating system - the operating system is a platform. In these cases, lock-in tends to evolve and get stronger and stronger. As you - or others - build on the platform

[8] This is the “yin and yang” of software production. Be proprietary: you get more money but your costs are higher and people are less likely to extend and build on what you create. Conversely, if you go open source you get less direct money but your costs of production are lower.

[9] The people most likely to contribute (or to pay others to contribute) to development of (open source) software are actual users of the software (it is the users who are getting the direct benefits). However, when software is at early stages it is likely not very usable - and it may be unclear whether it ever will be usable. Thus, at this stage, open source has all the costs of development but few of the benefits in terms of additional contributors helping out.

[10] With open source, lock-in still occurs at the technology level. Ultimately, lock-in to a technology or platform is unavoidable given the nature of software. However, open source still reduces the level of technological lock-in, as its open nature makes it easier to adapt and evolve the technology.

it becomes more and more useful, so more and more people build on it, making it ever harder to switch to an alternative.

Anticipating that they extract value from locked-in users later, vendors are willing discount software and services early on to build their market share. For example, Twitter charges nothing for its use, is conservatively estimated to have "lost" over a $1 billion and is yet to see profit. It is thus pricing its service well below cost. The assumption motivating its backers is that, once firmly established, it will have a form of lock-in – its audience and hence advertisers – sufficient for it to charge high fees to advertisers (or others) that recoup those losses.[11][12]

Pricing and marketing strategies like these, which rely on high switching-costs, might be termed the "crack" model: a vendor give its users some "crack" (software) for free in the hope that it is addictive enough (i.e. has high enough switching costs) that later on when they raise the price the users will not be able to "stop" (switch to something else).

The analogy is useful beyond conveying the basic concept. Drug addiction is a concern because we assume that users fail to correctly anticipate the costs of addiction on themselves and others: if they did anticipate them correctly, we could argue that addiction was "rational" and there would be little reason to intervene.[13][14]

Similarly, our concern about "addiction" in software (lock-in from high switching costs) is that users incorrectly anticipate future costs. As with drugs, they inflict these future costs not just on themselves but on others: because of the high fixed costs / low marginal cost and platform nature of software, my choice in software affect yours, because there is a strong incentive to cluster on a "successful" software platform. This success is determined by which software is chosen not just by me but by others.

The Future is Important and Difficult to Know

That knowing the future is hard is true in many areas of our life. However, in software it is especially important – and difficult for several reasons.

[11] The pricing dynamics of software, especially low up-front prices, are not purely a result of switching costs but also interact with the cost model of software (see below) where scale (i.e. lots of users) is key.
[12] In a world of perfect foresight, switching costs and lock-in would not necessarily be a problem: the switching costs (and hence lock-in) later would be perfectly anticipated up front and fully discounted into the initial price. However, as we will discuss, we are far from this perfect world. Moreover, in the world of software where technology has multiple and complex interdependencies, it is very hard to understand the implications and costs of future lock-in – and hence the discount you should get today.
[13] The point would essentially be that we do not wish to dispute others' preferences: de gustibus non est disputandam. After all, there may be many areas where I think you make mistaken choices or vice versa - e.g. my watching X-Factor, or your liking for classical music.
[14] There is, interestingly, a rich economic literature on "rational" addiction models in which users correctly anticipate their changing future preferences (i.e. growing addiction). The main interest there is on the differing impact of policy actions e.g. price regulation when addiction is rational vs irrational and on the unusual behaviour of path-dependent preferences.

First, software has a very long life compared to many we buy. For example, it is not unusual to hear of corporations using and evolving software systems that are decades old – that is rare for a car or a refrigerator. At the individual level, we stay with trusted and familiar programs for years, even after they have been superseded by superior alternatives (or even newer versions of the same software). This fact is directly related to and driven by the lock-in point: once one has learned or built on a given system, one is loath to surrender all that work and use a new system.

Second, software technology evolves incredibly quickly and often unpredictably. This means that it is hard to predict which technology options will be available in a few years time.

Third, user needs also evolve rapidly and in unpredictable ways (often driven by the experience of using the software). This is especially important because software is so easy to alter and adapt. If I start using my car and I decide I do not like its dashboard layout it is very expensive to change. By contrast, if I decide I do not like the layout of my software application it is quite easy to tweak and modify.

Together this means that: a) my needs (as purchaser) will almost certainly evolve in ways I did not anticipate b) it will (often) be feasible to adapt my existing software to those needs.

Finally, humans **underestimate** the degree of change for two reasons. First, we find it difficult to conceive of the multiplicity of possible futures and weight them accurately – we focus on one or two outcomes and exclude low probability options. Second, we have an "anchoring" and framing bias based in the present and we overweight futures that are close to the present because it is easier to extrapolate from it.

Together, these exacerbate the lock-in problem: rapidly changing technology and user needs imply I will need to see software developed rapidly in future. I will need to get upgrades regularly to my existing system or switch to another system. Switching costs will therefore "matter" on a regular basis.[15] Furthermore, the incremental nature of software will confer a major cost advantage on existing software (and especially my choice of software) as it can be incrementally evolved (rather than having new code written) thus increasing lock-in over time. Finally, and most importantly, for behavioural reasons, we will underestimate these changes and the level of lock-in they generate.

Government Incentives and Bargaining

[15] In addition, it is worth emphasizing that rapid change and uncertainty about the future mean that initial contracts will be very "incomplete" regarding the future performance and development of the software. This makes it hard for the buyer of software to specify at the time of initial purchase how the software should be in future periods. This weakens the buyer relative to the seller and increases the risk of hold-up. Combined with the behavioural issues this increases lock-in and reduces the ability of the buyer to bargain efficiently up front with the seller.

Governments are bad at negotiating generally and are especially bad in this environment. As a result, the lock-in problem is especially acute for Government. Government struggles here for several reasons:

1. The principal-agent problem. The bureaucrats who actually buy software for government are themselves the agents for the government. The government itself is , in turn, an agent for “the people”. Thus, we have a two-stage principal agent problem which is problematic for effective supervision and incentive alignment.
2. Governments, and their bureaucratic representatives, over-discount the future compared to the present. Crudely: in ten years time they may not be in office but the software they bought will probably still be in use . This means they underweight lock-in.
3. Government bureaucrats in charge of procurement are usually less experienced and less competent compared with the private-sector parties with which they deal.

The principal-agent problem
Software procurement by government is ultimately for the benefit of citizens - either because they use the software (e.g. it runs the country's healthcare site) or because it makes government run better. However, the people responsible are individual bureaucrats who a) may never use the system b) do not bear the costs (or get the savings) resulting from a given system. In the terminology of the principal-agent model, citizens are the principal whilst the procurement bureaucrat is the agent.[16] Even worse, software is complex and hard to evaluate, so it is difficult for the principal (citizens) to assess whether the agent (the bureaucrat) has done a good job. Second, citizens are diffuse and poorly organized so it is hard for them to monitor and sanction the bureaucrats (of course, citizens can elect "management" - i.e. politicians - to monitor the bureaucrats but that just adds another level to the principal-agent setup).

The response to these issues is twofold. First, governments introduce elaborate rules and procedures for procurement to follow and focus on compliance with those rules rather than the actual outcome (good software). This is because compliance is auditable and visible. Second, and more significantly, bureaucrats face extremely skewed incentives: if something visibly goes wrong they will be punished severely; if something goes wrong that is not obvious (e.g. you paid $20m for a working system that you could have got for $10m) nothing will happen; if something goes right in ways that are visible someone else will take the credit; if something goes right in ways that are hard to see no-one will care (e.g. the system works better than expected, or is cheaper than anticipated).

This makes bureaucrats extremely risk-averse as well as strongly inclined to do the standard

[16] In fact the problem here is worse than a simple principal agent problem as there are two levels. Government bureaucrats in charge of procuring software are usually at the far end of a two-level principal-agent problem: citizens elect politicians (level 1), and then these politicians oversee the bureaucrats who actually procure the software (level 2).

thing ("no-one ever got fired for buying IBM").[17]It also makes them innovation-averse, as innovation has a high risk component: it pairs a small probability of a large win with a large probability of small loss (you lose all or part of your initial investment).

It should be clear that these points have major implications for software. First, software is especially complex and hard to assess, so the "monitoring" and reward problems inherent in the principal-agent problem are especially severe. Second, open source is perceived as "risky" for bureaucrats because it is a) (still) less standardized b) its development model makes it harder for bureaucrats to (appear to) transfer risk to the contractor, for example: around legal compliance or security.[18] Third, open source fits less well with bureaucratic processes around procurement: open source is about adaptability whilst bureaucratic procurement is about spec and deliver; open source is often less-developed up front but can be more easily adapted and extended; and, finally, open source firms are smaller and are less able to afford the red-tape involved in bureaucratic procurement. Fourth, and most importantly, the future is often entirely ignored in procurement because it is complex and does not fit well with a rigid process - it is not a feature you can define and it is hard to include in the price. Thus, **the costs of lock-in - and the benefits of reducing it - are often not considered at all in the procurement process**.

Governments over-discount the future

Governments, or, at least, bureaucrats have short time horizons, at least relative to the lifetime of software. Elected governments often last only a few years, and inside those governments bureaucrats move around frequently. Thus, a bureaucrat who buys a particular piece of software will rarely still be in the same post when the consequences of that decision are felt, especially if we are thinking in terms of later upgrades and customization.

Thus, government software buyers care much more about today than tomorrow and will quite happily trade, for example, a $5m saving in buying software today for a $50m higher cost of software three years down the road. This means that “future features” such as ease of customization or switching are given much less weight than “today’s features” such as a cheap price or more options.

[17] Risk aversion means that gains are weighed much less heavily than losses. Note that gains here include *saving* money, so bureaucrats are quite insensitive to poor outcomes that comply with the rules. For example, where the bureaucrat pays $100m for a software system for which they could have paid $50m, but where the selected system was lowest bid against the procurement specification and to get the $50m option required a small change in the specification or slightly relaxed qualifications for your bidders.

[18] Note that the proprietary model does not actually handle these risks any better, but they hide them better from the bureaucratic process. Remember, what matters to the bureaucrat is compliance with the process and the appearance - not the actuality - of safety and security. Open source, unlike the proprietary option, does not hide these issues nearly as effectively (for example, closed source software may have risks around actual legal ownership of code, or lack of infringement, or indemnities for security issues but these can be hidden better than in the open source setup).

Government lack of experience and competence relative to private sector counterparts
In software negotiations, government bureaucrats are often less competent and less experienced than their private sector counterparts. Government IT is a multi-billion dollar business. Governments often do large deals – in large part, for some of the bureaucratic reasons discussed earlier: procurement is painful so doing it in big chunks is attractive.

Thus, you will often have a bureaucrat on a $50k salary up against private sector sales personnel on ten times that salary on a deal for software worth tens or hundreds of millions of dollars. In such circumstances, it is not surprising that the government comes off worse. It is also no surprise that future costs (lock-in) get neglected and that open source, which cannot afford the high-priced sales teams, loses out.

How To Promote Open Source in Civic Tech

We have established in the previous section that there is a strong basis for promoting open source for civic tech and specifically government. This section focuses on some strategic and tactical suggestions for achieving this goal. There are four specific proposals:

1. Recognize and reward open source in IT procurement
2. Make government IT procurement more agile and lightweight
3. Develop a marketing and business development support organization for open source in key markets (e.g. US and Europe).
4. Open Offsets: establish a flexible permits-style trading scheme for funding open source software similar to carbon offsets. The analogy is based on the fact that many public and private entities benefit from open software but do not "offset" that use with matching funding of open source. Here, public and private actors could purchase "open offsets" with to discharge their funding obligations with the money raised allocated to funding open source projects.

Recognize and Reward Open Source in Procurement

Give open source explicit recognition and beneficial treatment in procurement. Specifically, introduce into government tenders: EITHER an explicit requirement for an open source solution OR a significant points value for open source in the scoring for solutions. Significant points value would be, say, greater than 30% of the points on offer being allocated for the solution being open source.

It is important to ensure here that open source is explicitly defined as per the Open Source Definition, that the open source requirement cover **all** components of the solution.

As an aside, we note that a challenge for pursuing this approach is the existence of powerful existing proprietary software vendors who will oppose this change. Ways to address this include adopting the second option (points-scoring as opposed to fixed requirement) as that still allows for proprietary solutions to be offered (and to win if they are sufficient superior). Second, but less attractive (since it is vulnerable to expansion as an exception), is to focus more on areas that are relatively new and temporarily neglect areas in which lock-in is already large, and vendors well entrenched.

Agile Procurement

Alter procurement methodologies to favour agile approaches over "spec and deliver". Specifically, current methodologies follow a "spec and deliver" model in which government attempts to define a full spec up front and then seeks solutions that deliver against this.

The spec and deliver approach greatly diminishes the value of open source - which allows for rapid iteration in the open, and more rapid switching of provider - and implicitly builds lock-in to the selected provider whose solution is a black-box to the buyer.

In addition, whilst theoretically shifting risk to the supplier of the software, given the difficulty of specifying software up front, it really just inflates upfront costs (since the supplier has to price in risk) and sets the scene for complex and cumbersome later negotiations about under-specified elements.

Instead, create an agile procurement stream in which, rather than detailed requirements being set up front, you secure estimated budget for an initial phase and seeks bids on X number of sprints (with ability to end after any sprint). This model requires acceptance of some budget uncertainty: the total cost of delivery of software may not be fully known in advance. However, one can, alternatively, fix a budget and accept some uncertainty over features delivered. We emphasize that this limitation is not a limitation of open source but of software in general. As the maxim goes: in software development you can have any two of features, time and budget – but not all three. Traditional tendering with its fixed requirements, fixed timeframes – and implicitly fixed costs – exists in an illusory world where one can have all three and implicitly imagines that buying software is like buying traditional goods like chairs whose features, usage and cost are all well-known up front.

The agile model should be paired with a requirement that the software produced is open source. This is the only way to get real value as it ensures the government purchaser retains freedom to switch from a given vendor in future. This preserves a good bargaining position and the ability to exit the relationship if vendor performance is inadequate. Moreover, software development will end at some point and the buyer should be left with software they can freely use -- as well as adapt and build on in the future if they need to.

To move to more agile approach requires that procurement become less costly and risky inside government. Currently the high cost and risk mean that bureaucrats have strong incentives to bundle procurement into as large chunks as possible.

To counter this we propose that government officials have an explicit threshold below which they are allowed to engage in a lightweight, fast-track process. One could even have multiple thresholds with reduced bureaucracy below each threshold. Suggested thresholds would be $250,000, $50,000, $20,000 (each lower threshold having a lighter-weight approach).

Finally, we note that a light-weight procurement methodology that supports agile software development will benefit government generally and will also support more procurement from (local) SMEs which is something many governments wish to do.

Open Source Solutions - a marketing and business development agency for government open source

The proposal is to develop a marketing and business development support organization for open source in key markets (e.g. US and Europe). The organization would be small, at least initially, and focused on three closely related activity areas (in rough order of importance):

1. General marketing of open source to government at both local and national level: getting in front of CIOs, explaining open source, demystifying and derisking it, making the case etc. This is not specific to any specific product or solution.
2. Supporting open source businesses, especially those at an early-stage, in initial business development activities including: connecting startups to potential customers ("opening the rolodex") and guidance in navigating the bureaucracy of government procurement including discovering and responding to RFPs.
4. Promoting commercialization of open source by providing advice, training and support for open source startups and developers in commercializing and, especially, marketing their technology. Open source developers and startups are often strong on technology and (very) weak on marketing and selling their solutions - the support here helps address these deficiencies

Open source has particular "collective action" challenges around marketing. First, open source is a general concept that needs to be promoted to potential users. However, being everyone's problem, this is also no-one's problem. Second, open source firms tend to be smaller naturally because they have more competition and lower margins. By contrast, proprietary firms (successful ones) become huge monopolists with marketing and lobbying budgets to match. Their size and profitability generates large amounts of money to invest in marketing their own specific products and advocating for proprietary solutions generally (especially where threatened by open source). They are also able to solve the collective action problem, at least partially, thanks to their size relative to the market.

This demonstrates the need to market the open source concept. What is needed is something like an industry association but (initially) smaller and more dynamic. This marketing would focus on targeted engagement with decision-makers in government, rather than public activities or mass marketing. The model here would almost be a sales one, but focused on selling a concept rather than selling a product.[19]

Another, related, challenge for open source is that open source companies struggle at the individual level with marketing and business development. In part, this this is about attitude and experience. The bigger part, though, arises from the fact that open source companies lack the up-front capital that proprietary firms have. Marketing and sales is a major "up-front" cost where you have to invest (significantly) now for a future return. As discussed previously, the open source model has difficulty generating this sort of up-front cash-flow. In addition, open source civic tech firms are all selling to the same customer (government) whilst being relatively small and fragmented and therefore can especially benefit from coordinated support in this area.

The proposal here addresses this challenge by providing specific business development support including:

- Opening the "rolodex": connecting a startup with a product to the relevant people (getting them a demo or a phone call)
- Guidance on navigating the government procurement system including finding and responding to RFPs or obtaining relevant certifications[20]

The suggestion here is to provide this business development function in the same organization as the general marketing; the logic being that the marketing is more low-key, one-to-one sales-style and therefore has a natural overlap with the business development work - both in the "rolodex" aspect and understanding the government system.

In terms of actual funding and structure, we suggest organizing this entity as a for-profit both in order to attract the right kind of talent - e.g. experienced sales and marketing professionals - and to provide a path for sustainability. Initial start-up capital can be relatively small e.g. $2m - the lack of any existing investment even here means even relatively small initial support could have a large impact. The suggestion would be that this investment would be provided from philanthropic sources. Once established, the organization could sustain itself by establishing a membership model for open source businesses based around support for both marketing and business development. In addition, it could engage in some consultancy and training around business development. The actual organization could initially be relatively small: just a few

[19] This marketing effort would focus on "evangelism" - i.e. building awareness, understanding and support - rather than "lobbying" for specific policy change.
[20] Government procurement often has all kinds of restrictions on who can supply, e.g. you must have over X revenue, have certification Y, be based in location Z. Open source startups are often especially small and lack the kind of resources to allow them to do this. Help here can either be in getting these certifications, or, more likely, in connecting them to relevant organizations who have the certification and are able to act as "sponsors".

experienced sales and marketing staff with a small support team.

Open Offsets

Open Offsets is a model for funding open source analogous to the carbon offsets model for funding actions to reduce carbon pollution that contributes to climate change. It is based on establishing target levels of open source financing combined with a flexible permits-style trading scheme for public and private actors to discharge these obligations. The essential components are:

- Tracking direct government, philanthropic and private funding for open source (either through procurement or sponsorship)
- Establishing target commitments for spending on open source (these may be formal or informal targets)
- Establishing a route for those whose spending is insufficient to pay into an "offsets" fund: those who have "under-bought" or "under-funded" open source relative to their commitments have to buy "open offsets" with those funds used to support open source development.

The background to this approach is understanding open source as a similar (but opposite) "commons" problem to environmental "commons" problems like global warming. Specifically, open source has the problem that, being freely shared, creators generate positive benefits for others that they are not paid for - i.e. "positive externalities". With global warming the issue is that creators of carbon dioxide create costs for others in the form of global warming that they do not fully pay for - i.e. "negative externalities".

Just as carbon offsets (or carbon prices) seek to internalise the negative externality of carbon emissions by getting emitters to pay for their emissions (and offset them) so, with open source, we want to internalise some of benefits by getting major beneficiaries like governments to pay for the benefits they receive directly and indirectly from open source.

Specifically, governments - and other groups - would commit to certain levels of funding for open source. However, rather than a rigid system whereby funding has to be in a specific form (for example, buying specific open source software), it is better to combine targets with a flexible, permits-style, approach to meeting the target.[21] In particular, participants in the program can discharge their funding target via several routes; for example, purchase of open source solutions or general funding and support of open source software development. Second, those who have underspent against their target can make up their deficit by spending into a

[21] This takes direct inspiration from environmental regulation where this kind of approach is common. It also takes inspiration from work on alternative (non-IP based) funding models for innovation and especially medical innovation. See e.g. http://blog.okfn.org/2004/10/30/the-medical-innovation-convention-a-new-global-framework-for-healthcare-research-and-development/.

common fund, whilst those who overspent can reclaim from the fund (within limits).

This schema would be targeted at government and other relevant entities such as philanthropies and publicly-funded research organizations. It can also be run formally or informally: informally would mean that there were no contractual commitments to particular targets. As well as increasing funding, the schema provides increased transparency, accountability and coordination around open source support.

Choose Open

"Choose Open": a grass-roots oriented campaign to promote open software in government and government run activities such as education. "Choose Open" would be modelled on recent initiatives in online political organizing such as "Move On" in the 2004 US Presidential election as well as online initiatives like Avaaz. It would combine central provision of message, materials and policy with localized community participation to use these to drive change.

The strength of the open source is its large set of people who are deeply committed and supportive to it. Many of these people might be willing to volunteer to promote its use. However, to be effective this community needs direction and coordination. Using community-organizing techniques "Choose Open" should be able to convert this group which is currently informal, diffuse and poorly organized into a proper activist network by providing it with direction and process.

Choose Open would focus initially on more local change: e.g. at the city or district level as this will be the most favourable for driving change given the grassroots make-up of the supporter base the political dynamics of national decision making. In addition, tactical choices can be made in terms of whether to focus at the policy level (e.g. a city Mayor introducing an open source preference), the discretionary level (a school switching to open source) or even just the individual level - more people who know and care about open source.

In terms of how this would proceed we anticipate something like the following

- A "Choose Open" campaign would be set up with a small paid staff
- Identification of overall mission and specific potential asks
- Recruitment phase to identify and select local volunteer "leads". Seek individuals who are both committed but also reasonable and credible – pragmatic rather than fanatic. For example, if you were targeting increasing use of open source in schools you would want someone who was a parent and respected within the community. Aim would be to have one leader per "area" at the lowest level with perhaps one or two volunteer coordinators at each level above that.
- Start a campaign. Look to have one campaign at a time nationwide. We think schools might be a great initial focus as there is a strong case on several levels to be using and teaching with open source.

Because of its activist nature, “Choose Open” would obviously not be funded by government. Instead we propose it would be funded either by philanthropists or by open source businesses.

Appendix

Why Open Standards or Open APIs mean little (relative to open source)

Sometimes it is argued - often by purveyors of proprietary software solutions - that the focus should be on open standards (and "open APIs") rather than open source.

Here, I want to briefly make clear why I think open open standards (and, even more so, open APIs) are no substitute for open source.

The logic of standards is usually clear: a standard is almost always about coordinating on some kind of technology "platform" (conceived in the broad sense); in particular, specifying the interface between the provider(s) of the platform and users of the platform and allowing these two groups to operate independently.

For example, there are standards for the width of railways, or for electric plug sockets. The railway standard means that people can build tracks (the platform) whilst others can make trains that run on those tracks without having to coordinate (and, even better, different people can make different parts of the track and they will be consistent). Similarly, people can build houses and install electric sockets and you can go out and purchase electrical equipment and know that the equipment and sockets will work together. Since information technology has platforms everywhere, standards are a big deal.

Our interest here is not in standards per se, since they are obviously very valuable, but the role of (open) standards in avoiding lock-in to a given vendor. The core of our concerns about open standards on their own is that:

- Open standards without running code to implement them mean little. If only one vendor can implement an "open standard" due to complexity or other factors, the fact the standard is open delivers little in terms of reduced lock-in or competitiveness. The real test of an open standard is a running open source solution and delivers full compatibility
- Standards are often adapted and deviated from because software is so malleable. Again this means what really matters is the actual implementation - the running code - not the standard as written in some official document. Once again, it means we should focus on the openness of software which represents the standard "in reality" rather than the

openness of the standard.
- Standards are usually behind the curve technologically due to slow pace at which standardization moves. This means, similar to the previous point, that the standard does not represent what is actually being used. Once again it is running software which matters - and whose openness we should focus on.

Open APIs are even less meaningful.[22] An API is an "application programming interface". Originally, APIs were about connection points between different pieces of software. With the growth of online activity we are increasingly talking about online APIs.

An "open" API as the term is conventionally used simply means and API which is publicly documented and which can be ipmlemented by others free of charge. However, that "open API" is still owned and controlled by the vendor implementing the system exposing the API and it can be changed at any time at the whim of that vendor. Competitors who seek to implement the API themselves will always be playing catch-up and have no guarantee that the API will remain stable or even properly documented. As such, the benefit of an open API in avoiding or reducing lock-in is negligible.

Furthermore, for online APIs making something "open" just means making it public. However, online APIs are already de facto public -- no-one is going to make an online API and then keep it secret and try get people to pay to know about it (they may charge you to use it but that's different). As such, making the API open is an empty statement: it already was open. The commitment to an open API would really only mean something if the API was shared and standardized across multiple vendors in which case it would be a standard and we would be back to our previous discussion of standards.

In sum "open" APIs are a form of sham openness. They do not deliver any of the real benefits of openness and serve only to distract uninformed buyers from more meaningful forms of openness such as open source.

[22] We made similar points several years ago in https://blog.okfn.org/2006/09/04/open-apis-dont-equal-open-knowledge/