SCL TVG FOSS: Neil's notes

(These are the notes I put together before the presentation, to give me an idea as to what I was going to cover. I may have deviated from these on the night. They are licensed under CC BY SA 2.0)

Introduction

Ladies and gentleman, good evening. I'm delighted to have the opportunity of speaking to you tonight, especially since there are so many of you!

As Sam has said, my name is Neil, and I split my time between senior counsel to Vodafone, advising on a range of telecoms and technology issues, and reading for my Ph.D in communications regulations. I have been interested in Free software for around 10 years now, and, as well as being an avid user of Free software, I have written a fair amount about it, including contributing a chapter on Free software in business to Oxford University Press's publication.

I am going to talk briefly through what I see as the three key points when considering advice to business on FOSS and, inevitably, since we have kept speaking time down to just 20 minutes, it is going to be a brief introduction to the issues. For those who might be interested in a little more detail, there is, of course, the book, and, on my website, I have a reasonably detailed pack of notes on some aspects of this, which accompanied a longer session I taught over the summer. Feel free to help yourselves. Unsurprisingly, they are available under a very permissive Creative Commons licence.

The are two areas on which I am going to spend a few minutes tonight are:

- developing a FOSS strategy and policy: understanding how your business intends to use FOSS is the first step in being able to advise on it informedly, and we will talk about how you get to a clear understanding of that strategy. The FOSS policy is the underpinning of the effective and safe use of FOSS within an organisation, and I will discuss what I consider to be the most pertinent points in building a solid base

- licensing own code under FOSS / using third party FOSS: I will talk about why a company might wish to license its own code under FOSS terms, and the other side of the coin, using third party FOSS in your own products. I make an assumption in these sections that you are advising a commercial organisation, rather than a not-for-profit or a foundation focussed on Free software — the considerations for these would be really rather different, and the focus would probably be on the foundation governance rather than on the use of FOSS *per se*

FOSS strategy and policy

Now, you are in a complete safe and friendly environment — I'm not sure that anyone has ever said that about a room full of lawyers before, actually — but, in the spirit, I'm going to ask a question, and, if the answer is yes, please just stick your hand in the air.

It is a very simple question: does your organisation use FOSS?

I suspect that, if you were to ask your CEO and your CTO that question, the CEO is likely to say "no", and the CTO "of course". These days, I think you would be very hard pressed to find an organisation which does not use FOSS in some way, shape or form — whether that's an Android handset, Linux running as router or a web server, or code actually baked into your products.

FOSS strategy

A company may use FOSS for many different reasons, or without a reason. At one end of the spectrum there are companies using FOSS without knowing it; perhaps they have been supplied with products which contain FOSS components, or perhaps their developers have built a product or platform using their favourite FOSS code and never thought to mention it. At the other end are companies for which the use of FOSS is a measured business decision; perhaps they use FOSS to reduce their time to market, or to avoid reinventing the wheel, thereby allowing them to spend money on differentiating elements of their product or service.

The difference between using FOSS without realising it and using FOSS in a way which furthers the company's business objectives is the existence of a FOSS strategy.

Determining the FOSS strategy and policy

Understanding a company's FOSS strategy is the starting point for developing a FOSS policy. Without a clear statement of how the company intends to use FOSS, any policy document exists in a vacuum, and is likely to have little value.

A FOSS strategy is a simple explanation of what a company intends to achieve through the use of FOSS. In developing its strategy, a company is looking to complete the statement '*We use FOSS to* ...' The aim of this is to get a considered understanding of the company's rationale for using FOSS; for example, 'We use FOSS to keep our costs down,' or, better, 'We use FOSS to allow us to focus spending on differentiating elements of our product.'

Importantly, the company is not looking to complete 'We use FOSS *because* ...'. In trying to answer this question, the company is likely to produce a list of generic statements: we use FOSS because it is cheaper, we use FOSS because it is more stable and so on. Whilst these may or may not be true, they do not help understand the goal the company is trying to reach in the use of FOSS.

Who should be involved in setting the strategy? All sorts of people, not least your engineers. You need buy-in, else it will just be a largely pointless piece of corporate governance.

A FOSS strategy gives a clear direction on use of FOSS within the company. However, without operational guidance on delivering this strategy, the business will need to make ad hoc decisions on use of FOSS, which is likely to lead to inconsistent or ill-considered use of FOSS, as different business units attempt to understand what the strategy means to their operations. The FOSS strategy should thus be seen as the underpinning of a FOSS policy, which sets out the business the how, what and where of FOSS usage.

The FOSS policy

There are, in my view, five main areas which a FOSS policy should look to cover:

- Use of third party FOSS in products and services
- Use of third party FOSS other than in products and services
- FOSS in the inbound supply chain
- Releasing code as FOSS / contributing to FOSS projects
- Handling exceptions

The most critical part of a FOSS policy is a clear statement as to which licences are permitted for use in the business. This may be an absolute statement, or else the choice of licence might be dictated by the use to which the FOSS code is to be put, such as a restriction on the use of code licensed under the Affero GPL or LPGL in hosted service components. Whilst a simple approach may be attractive from a clarity and compliance perspective, a more purposive approach may attain better results for the business.

The main consideration for a purposive approach is whether the code in question is to be used in a differentiating or non-differentiating manner. More on that later, but, as you have just heard from Max, there is a whole spectrum of licences, so more permissive than others. Generally speaking, a broader range of licences is likely to be acceptable in non-differentiating code than in the part of your product or service which actually makes you stand out.

Use of third party FOSS on there than in products and services: are you happy if your staff use, say, the Firefox browser, or your office router uses Linux?

FOSS in the inbound supply chain: the question you are trying to solve is "what, if any, FOSS am I happy to accept in products or services supplied to me, and how to control this?" If there is enough positive feedback or requests, we could pick this up in a follow-up session.

Releasing code as FOSS / contributing to FOSS projects: I will pick this up in the next theme

Handling exceptions: who is going to answer questions when they come up, or make decisions as to what is, or is not, acceptable if it does not comply with the policy. You probably want someone knowledgeable about the strategy side, someone with some legal knowledge, and someone who understands the technical implications of decisions. That could be just one person, or it could be a board.

Why pick a FOSS licence for own software?

Why might someone choose to license their own software under FOSS terms? To copyright lawyers used to the idea of maximising restrictions, and licensing out on the most prohibitive terms possible, the idea that someone might create something and then give it away might seem alien.

As discussed in the introduction to Free software, above, for some, ensuring software freedom is a political movement and statement — that non-Free software is an evil which should be abolished. Some, perhaps many, Free software developers write software under FOSS licence terms to give effect to this belief, or because they seen no value in restricting others' access to their work. I, for example, often license my notes under CC BY 4.0 (attribution only) terms, to remove unnecessary copyright restrictions which would prevent modification or simple redistribution.

Conversely, others may choose to license their own software under FOSS terms for commercial or business, rather than political or ideological, reasons. Three of these are explored briefly, as food for thought.

To build a community

You may have a great piece of software, but not have the resource, or other capability, to take it where you want it to go on your own — you need the involvement of others. You could take a more traditional route, and hire (and pay) third party developers. Alternatively, you could make your code base available to anyone with the skill and inclination to work on it, and benefit from the changes which they might wish to make.

There would be no guarantee, of course, that anyone did want to work on your project, and it is unlikely that, in the early days, at least, it would be the same kind of "work" that you might direct of a hired developer: rather than someone building what you want, they are more likely to make changes which do something which they want. You may end up with a richer, more capable piece of software, but it might not be what you had originally envisaged.

[Alternatively, focus on development on non-differentiated elements as part of building market segment — for example, a VR headset manufacturer, which makes money from sales of hardware, and which might consider open sourcing its enabling drivers on the host computer, and encouraging community development / porting.]

To undercut a rival's investment

Consider the situation in which you have spent months developing a new piece of software to do something amazing — something for which there is nothing else quite like it on the market. And, a couple of months before you are ready to launch, someone else puts something substantially similar on the market. They win customers immediately, eating heavily into your projected market space.

You have choices:

You could start selling your software immediately, before it is quite ready, and try to win a chunk of the market, but at the risk that your software fails to perform adequately.

You could wait until your software is ready, but then enter a market where someone already has the benefit of the first-to-market advantage — you might be able to win customers who have not already bought the other company's solution, but the size of the market is much smaller; you are unlikely to attract customers who have just bought the other company's solution unless yours is much better, or theirs is failing.

You could try to undercut your rival's investment, and regain the focus. One mechanism to do this might be to release your software as FOSS, and, additionally, not charge for it — assuming that your software is comparable and not materially worse, new customers have the choice of either buying your rival's software,

or taking yours without charge. By making your software available on both gratis and FOSS terms, someone who has already invested in your rival's solution may take the opportunity of downloading yours and experimenting with it, where, had they had to pay, they may not have done so.

Likewise, you could offer the core of your software on FOSS terms, and charge for proprietary modules and plug-ins; this may be particularly suitable if you are able to build a community around the software, and encourage third party module development, to broaden the appeal and reach of your software. This approach is commonly known as "open core".

This is not without risk, of course, and it might require a shift in business model, to providing support and maintenance services, or to offering custom development and consultancy, but it might keep you in a market which otherwise might not have supported you.

As a desirable sales pitch

One of the biggest costs of any project involving software, alongside licensing costs themselves, is support and maintenance — that, in order to continue to receive patches and updates, and to have someone knowledgeable about the software available to support, you need to continue paying. This form of tie-in is very attractive to many software companies, which seen the ongoing provision of support and maintenance as a continual revenue stream.

To a company looking at purchasing a particular solution, the ongoing costs of support and maintenance may be substantial. Worse, a company may feel tied-in to that vendor: even if their support is poor, or the ongoing costs high, there may be little or no alternative.

You may find it highly marketable to be able to explain to such a company that, whilst you can provide support and maintenance, and, as the author of the code base, are very well placed to do so, since you make the code available under FOSS terms, anyone can look at the code base, and make the modifications that the company might want in the future: they can use you if they wish, but that there is no compulsion to do so.

In this way, whilst you might be introducing competition which you might not have otherwise had, you may open up potential business which you might not otherwise have won: if a company feels that they will be able to support the software if they no longer wish to deal with you, either through in-house support or through engagement of a consultancy of their choice, using your software, and taking your services for as long as they wish, may be desirable.

Why might a business use third party FOSS software?

Unlike a decision to license one's own code base under FOSS terms, the rationale for the use of third party FOSS is straightforward: to enable the business to focus on differentiating elements, rather than recreating components which already exist.

Take, for example, a company which wishes to offer a novel new web service. The company could, if it so wished, build its own infrastructure from the ground upwards — it could write its own operating system, and web server, and supporting infrastructure.

However, it is likely that perfectly good code for these functions exists already, and effort spend on developing these components, which are unlikely to differentiate the company's product from others, takes resource away from spending on areas which really could add value.

By using FOSS code for these non-differentiating elements, not only can resource be focussed on developing differentiating components, changes to the supporting infrastructure can be made easily, and by a wider range of consultants than may be the case in respect of proprietary software: the source code is available for inspection by anyone with sufficient skills.

In a similar vein, in the event of problems, you are not dependent on a third party deciding to work on the problem and issue a patch: you can look to patch the problem yourself, pay someone to do it for you, or see whether someone else in the user community has already done so. In doing this, reliance on a single point of failure is reduced.

Conclusion

You have heard about two main areas of FOSS in business tonight.

First, the importance of having a FOSS strategy, with a FOSS policy sitting underneath it. The aim of the strategy is to answer the question "We use FOSS to…", and the policy provides an operational manual for the business.

Second, we have talked about business reasons as to why you might release your own software as FOSS, and why you might use third party FOSS — perhaps most importantly of all, there may be very good reasons for doing either or both of these things which are not purely ideological, but because they may simply make good business sense.

There are a number of topics which are very pertinent to FOSS in business, which the time available just doesn't permit us to address tonight. For example:

- handling in-bound supply chain
- compliance, development and code-scanning
- FOSS enforcement, and dealing with violations

I am sure that, if you had particular questions, we will be delighted to answer them afterwards, and, if you would like us to run a session on these issues, or, indeed, any other FOSS issues, do just let one of us know, and we will see what we can put together.