[These are the thoughts on which I based my thoughts for the SCL session on FOSS — this was written for me, rather than for publication, so please excuse any typos.]

"Demystifying Open Source" — FOSS in practice[1]

Introduction

Good evening — my name's Neil. For those of you I haven't met yet, I'm an independent academic, looking at the overlaps of law, technology and society. Frankly, I'm a legally qualified geek. Alongside my academic life, I'm currently employed by Vodafone as legal counsel, doing all sorts of geeky technology and law stuff. I've worked with open source software for many years now, both on the technical and legal side, and also on the business side of things.

Tonight, I'm going to give you some introductory thoughts, as predominantly a legal audience, on issues around Free and open source software in practice.

I'm going to use the term "FOSS" throughout, to cover any software under Free or open source licences. I don't mean this to be a political statement; it's just a convenient shorthand for me. If you want to ask questions, please do, and if you want to call it something other than "FOSS," that's absolutely fine.

Two key points

If you take nothing else away from what I'm going to say tonight, it's these two things:

FOSS licensing is nothing special. It's not special, it's not magic, and it's nothing to be scared about. I've come across many lawyers who seem scared by FOSS, and I think there's some kind of self-reinforcing mysticism around it. Just think of it as software licensing.

---

[1] Neil Brown (http://neilzone.co.uk | neil@neilzone.co.uk)

I don't say that it's "nothing special" to undermine it — I'll put my hand up, and say that I'm not a fan of the current system of copyright, and I think that FOSS, Creative Commons, public domain declarations (to the extent that they exist), and other more liberal approaches to knowledge sharing are highly desirable for achieving certain social goals. I say it's nothing special because I don't want people to shy away from these sort of licensing models through fear that they are dealing with the unknown. It's software licensing. It's nothing special.

The second thing that I want you take away is that you don't have to be a geek to give competent advice on FOSS. That's really important, and, in some senses, goes back to the point about FOSS being shrouded in mysticism. If you want to get engaged in some of the more specific debates then, yes, technical awareness is probably going to help quite a lot, but, generally speaking, you don't need to be a geek. So, for those of you who aren't, please don't give up hope.

With the scene now set, let's think about FOSS in practice.

What is the driver?

The impact of FOSS on your legal advice will depend, not unsurprisingly, on your role — what is it that you are being paid to do?

Is it procurement? M&A activity? Product development?

Each of these has rather different issues to think about, but I'd say that there's a commonality to each, and that's the driver behind the company's — your client's — use of FOSS.

[What is the driver?]

Perhaps the most important question to understand is "how does FOSS fit into the company's strategy?" Or, perhaps, why is the company using FOSS?

Sometimes a company is going to have FOSS policy. I'd see a strategy and a policy as being two separate things — the policy is basically the operational manual for delivering the strategy — but, if a company has a policy, hopefully at least they've done some thinking about the strategy they're hoping the policy will deliver.

If the company doesn't have a ready answer for the "Why use FOSS" question, you may need to pry a little. Now, of course, the person coming to you for advice may not be the right person or even in the right group of people to be making the determination as to why their company uses FOSS. So, if there isn't a clear strategy as to why the company is using FOSS, you'll need to find the right people to determine that strategy — who that is, or who makes up that group, will depend on the governance model of the company in question.

So you've found the right people, and you're trying to understand why the company is using FOSS. And you're really looking to get to the position where you can complete the sentence:

[We use FOSS to ...]

"We use FOSS to ..."

I stress this, because there's another similar sentence, which I don't think is as helpful at all:

[We use FOSS because ...]

"We use FOSS because ..."

If they are answering the question "because", you'll get a whole load of rather generic statements — we use FOSS because it's cheap. Because it's more stable. Because it's better.

Now, all of those may be true — or none of them. It doesn't matter at this point, but what does matter is knowing what some people think about FOSS doesn't help understand the company strategy.

Instead, by asking:

[We use FOSS to ...]

"We use FOSS to ..."

You'll hopefully get to a position where you know the company's reasoning.

"We use FOSS to keep our costs down," for example.

Now you know a strategic driver, not just a fact.

Better still:

"We use FOSS to allow us to focus spending on differentiating elements."

Great — you know where you are standing. You've still got a bit of a way to go, to understand whether it's important that the differentiating element is kept closed source or not, but you're in a pretty good position to start from.

Whether you are looking at buying something in, acquiring a company or whatever, you know what the driver is.

I'm going to break down the next few minutes into six topics — we'll look at procurement and FOSS in the supply chain. We'll then look about outbound licensing and the triggers for obligations. We'll touch briefly on product development and FOSS in M&A. And we'll close off by looking at determining compliance, and what happens when it all goes wrong.

It's going to be a busy twenty minutes!

FOSS in the supply chain

Let's start off by thinking about FOSS in the supply chain.

[Supply chain]

Let's look at a standard supply chain — you're buying stuff from suppliers, you're potentially doing something to it, adding your own value, and then delivering it to your customers, whether that's to end users or another downstream supplier. You've got the inbound, and you've got the outbound.

And they are pretty tightly intertwined — if you're looking to sell beef burgers, it's no good if your procurement team is buying batteries. And it's the same when it comes to dealing with FOSS — what you are willing and able to accept on the inbound leg depends on what you are looking to deliver outbound.

[Where and when — or not]

Now that you know what the driver is, you know what will work and what won't. If you know your client wants to keep costs down for non-differentiating elements of the product, but wants to keep the differentiating element to themselves, then, in your

contract with the supplier, you're going to need to spell out exactly where FOSS can be used, and where it must not be used.

[Provenance]

If your supplier is going to be including FOSS code, or, frankly any non-FOSS code with particular licensing provisions attracting to it, I'd want to know what the code was, what licence it's under, and how it is to be used — you want to know the provenance of the code. There is another argument, of course, that says that, if you are not actually going to do anything with the information, why bother gathering it in the first place but, in the event of a problem down the line, having this information could be very helpful, rather than just being stuck with a binary and no real knowledge as to what's in it.

If you are planning on doing this, I'd suggest that you have a look at SPDX.

[http://spdx.org/]

SPDX is the acronym for software package data exchange, and it's a standard developed by the FOSS community for sharing information about modules and their licensing. It's a nicely structured, easy to read, bill of materials for a piece of software. It's well worth looking at.

[Appropriate warranties]

One of the supply situations I see most often is someone coming to me and saying that the supplier has struck out all the warranties in the contract, saying that you don't get warranties for open source. I've also seen the opposite, where the purchaser has been insisting on a warranty of non-infringement of IPR, backed up by an uncapped indemnity, for FOSS which the purchaser has specified that the supplier must use. Now, if your supplier is an insurance company, and you're willing to pay for their due diligence and audit, and the risk they are taking on, fair enough, but that seems somewhat unreasonable to me more generally. A warranty against knowing infringement, and that the supplier will comply with all licensing conditions, seems far more reasonable.

But, again, working out which party is best suited to bearing the risk is nothing particular to FOSS — working out what's

reasonable in the circumstances, and what each party can actually perform, is not rocket science. Again, FOSS is nothing special.

I'm not going to talk through all the different possibilities and arguments and counter-arguments, you'll be pleased to hear. Instead, I'm going to recommend that you take a look at something called the Risk Grid.

[The Risk Grid]

The Risk Grid is a document produced by members of the Free Software Foundation Europe's European Legal Network, with extensive experience being both supplier and purchaser of FOSS, trying to demonstrate the kind of issues which come up, with suggestions as to where the risk should properly sit.

If you do lots of procurement work around FOSS and have not already seen this document, I'd suggest that the first thing you do tomorrow is download and take a look. I can't promise that you'll agree with it, but hopefully it forms a good starting point. Again, the principle of finding the right party to bear the risk is nothing special, but it might help appreciate at least one view of the situation when it comes to FOSS-related issues.

It's a bit convoluted to find where the Risk Grid is hosted, so instead just visit

neilzone.co.uk/riskgrid.html

[neilzone.co.uk/riskgrid.html]

and it will direct you to the page automatically.

[blank]

One thing which will obvious to a lot of you, of course, is that paperwork has very little effect in the real world — having words on paper does not equate to guarantee of a compliant code base. We'll come back to checking for compliance.

[Outbound / obligations]

Outbound / obligations

The other side of the procurement activity is the outbound side of things — making sure that you get things right when you are providing the product to your customers.

There are two points I want to note here.

Firstly, I've equated "outbound" with "obligations." The reason I've done this is that few licences have much of an impact when you are just using FOSS code for your own internal purposes. Generally speaking, obligations arise when you are making the code available to a third party.

[Distribution]

Secondly, the issue of what is "making available" or "distributing" (and I use these terms casually) depends on the licence — under GNU GPL 2.0, it's "distribution;" GNU GPL 3.0, it's conveyance. Under each, you envisage the transfer of a copy of the code from one party to another, or at least the second party making a copy of the code hosted by the licensor. Running the software — for example, a web server — in a such a way that third parties interact with it, but don't download any part of it, is unlikely to constitute distribution, and GNU GPL 3.0 expresses calls out that mere interactive use is not "conveyance."

If you don't have an understanding of the technology, the question you'll be wanting to ask at this point is exactly what is happening — is it mere interactive use, or is code being transferred to the customer?

The web server is, most probably, something with which your users simply interact. But, if you've got a Java applet on your website, when the user loads the page, the applet is transferred from the server to the client, and runs locally — in this case, you've got an act of distribution or conveyance, at least for the purposes of each of the GNU GPLs, and so you'll need to comply with the requirements on binary distribution in terms of that code. So, where the main compliance obligations are triggered on distribution, you need to understand what your product / software is doing, and whether it is distributed.

[Network interaction]

As you can't have failed to notice, we're seeing an increase in online interactive services, where the user may never receive a copy of the code in question but merely interact with it. In these cases, other than some perhaps unusual technical use cases, there's no distribution here.

Now, some developers see this change in usage pattern as being damaging to the principles of Free software, in that the code is not being made available to users, and we're seeing an increase in the number of projects which use network interactive use as the trigger instead of distribution. So, whilst GNU GPL 3.0 states expressly that mere interactive use is not "conveyance," the Affero version of the licence — the AGPL — adds an extra condition, that users "interacting with [the software] remotely through a computer network" must be offered the source code.

This goes back to the key message here, which is understanding the company's drivers, and ensuring that you manage your upstream or inbound licensing appropriately — if you are developing the next greatest web application, and the key to the success of that application is that the source code remains secret, it would be unfortunate to find that, because there were insufficient controls in place, the code needed to be licensed as AGPL.

[blank]

So the two points again:

Firstly, that, for many licences, "outbound" is the trigger for obligations, and internal use tends to be low risk.

Secondly, whilst "mere interactive use" is not conveyance, some licences do consider network interaction as the trigger for obligations, so check carefully, and make sure you know what is happening technically?

I'm going to talk about detecting violations as a means of checking compliance, but let me touch quickly on two other topics — in-house development, and M&A.

[In-house development]

In-house development

You've got a team of developers, and they are happily putting together your next big product — perhaps they are writing all the code themselves, but, equally, perhaps they can see opportunities to save time by using pre-existing FOSS code. Can they use it?

The question here is not really different to inbound licensing — the licences which your client is willing to accept will depend on what they want to do with the outbound licensing of the final product.

But the key difference here, to my mind, is that the supplier is not the one making it all hang together — you are not being supplied with a product which is being warranted as comply with all FOSS licensing obligations.

Instead, if you're buying in components, and you or another third party is doing the aggregation, then, depending on the degree and form of integration, you're going to need to make sure that the licences of the various components work together. As I say, there's no quick and easy answer to this sort of question, as not all licences are interoperable, but, depending on the degree of interaction between the components, this might not be an issue.

Generally, you're going to be sitting down with the architecture team, or the developer, or whoever's responsible for the overall design of the product, to work out what needs to be done. If your client has particular packages in mind — they want a particular web server, or want busybox for their routers and so on — then you're going to start building these in as dependencies for your project, and fleshing out the licensing from there. You might also need to explore technical ways to achieve what you want whilst also ensuring you are compliant.

Make sure you budget enough time for this, and that, as far as you able to control this, try to make sure that you get involved in the project as early as possible, rather than as a last minute "legal sign off."

FOSS in M&A

The main issue to touch on in a M&A context, as with most things to do with M&A, is to understand what you are buying, both in terms of the asset base, and also the risk that you might be taking on.

Hopefully you'll know a bit about any company that you are looking to acquire, but, in any case, if any of the value of the company comes from software, whether in a physical product or otherwise, I'd be wanting to find out whether the company used FOSS and, if so, how.

The key, again, is to understand the client's driver — it may be absolutely fine that the majority of the target's product is FOSS. But, if the fact that it is FOSS would impede what your client is planning on doing with the target, it's potentially a big deal, and so finding out exactly what's the situation is important. It's a key part of the due diligence process.

You might also want to consider including warranties in the share purchase agreement that the company has disclosed all use of FOSS, or something similar. "All," of course, is very wide, so, if you are the target, you might want to try to limit it — are you warranting just the current release of the code, or all future releases, or all branches of the code, or just the externally deployed code rather than code used within your business and so on.

[blank]

So, in all of these contexts — M&A, procurement, own development — you are going to need to be comfortable that what you are shipping is compliant, and the starting point for this is provenance — you need to know what code forms your code base, what licences are involved, and how the various modules interact. Even if you are pushing risk upstream contractually, you may well be liable for any copyright infringement arising from failing to comply with a licence requirement.

[Checking for compliance]

Checking for compliance

So how do you check for compliance?

I can think of three ways of doing this, or a combination of any or indeed all of them.

First off, you could hire an external compliance advisor, and pay them to check that what you are shipping is compliant. You're going to need to find an expert in the area and, the bigger the project is, the longer they are likely to need to become familiar with the code base. Obviously, you are going to have to allow sufficient time for their work.

Secondly, you could sift through the code yourself, and work out what the obligations are. Now, this might work if the project consists of a few modules, and you are happy to work through understanding what came from where, what each bit does and so on, but I don't know that many lawyers who have that kind of time on their hands — I've done that once in seven years, for a very small project.

Thirdly, you could look to use a scanning tool, to help you understand the make-up of the code base. Two things to note — firstly, there are not a lot of scanning tools, and so you are likely going to be limited in the options you have available. If you are looking at scanning source code, then you are really looking at either Black Duck or Fossology. If you need to scan binaries, then then you'll be wanting to use the Binary Analysis Tool.

Whatever tool you use, it does not do all the analysis for you — they generate information, which needs to read and understood and acted on. You may also need to check whether any of the flags raised are false positives, and so on. But, if you've got a binary, and you've no idea what's in it, you can either install the binary and poke around and see what's going on, you could look for strings in it and do some analysis on those, or you could use the Binary Analysis Tool and have a lot of this done for you.

But what happens if it doesn't all go to plan? What happens if you ship a product, and it contains FOSS, and you are not complying with the licence?

[Tackling infringement]

What happens if it goes wrong?

Chances are, the first you'll hear about it is from either an upset user, or else from Software Freedom Law Center or gpl-

violations.org, depending on the complaint and the copyright holder.

One thing I would suggest is monitoring the gpl-violations.org mailing list, as this is a good heads-up for allegations of GPL violations.

You've got the complaint in — what do you do?

Make sure the complaint goes to the right place, and is handled promptly — you may need to train your customer service desk to ensure that, when someone complains about FOSS, it gets put through to someone who can deal with it. If the complainant feels that they are being taken seriously, and that they are not being fobbed off, life can be a lot easier, particularly since FOSS complainants are often very technically savvy, and are happy to share their experiences.

Make sure you get a clear description of the alleged problem, so you are not on a wild goose chase, or else looking for a needle in a haystack, depending on your preferred metaphor.

At this point, if you have a warranty in the contract with a supplier, you may need to tell the supplier of the complaint — if your contractual protection is linked to telling the supplier within [x] days of receiving notice of a complaint, make sure that you give the notice.

Work out whether it's valid or not — in some cases, this is easy — perhaps your file hosting is not working as it should, and is not making the source code available for download. Or perhaps its something harder, that someone thinks you are required to make code available which you are not doing. Most of the complaints that I have seen over the years have been pretty basic issues of the code not being available, or not even mentioned.

But let's imagine that you've done all this, and you've found that there is a problem — that you are not complying with the licence. That you are infringing copyright.

You're can generally in one of two situations. You can fix it, so that you become compliant, or the problem is such that you simply cannot fix it.

Perhaps you can't fix it because you do not have what is required of you — for example, that someone upstream has used FOSS, not declared it, and you have no way of getting the source code, no way for your supplier to get the source code, and so on. Especially in the embedded electronics world, with a long supply chain and minimal margins at each step, this can be a real problem.

Perhaps you can't fix it because fixing it would undermine your business model — for example, releasing source code which you had not expected to release might give away the inherent value in your product, such that it is easy to replicate and you lose your advantage.

Well, if you can fix it, great — if you can do it quickly, such that people overlooked the past infringement, brilliant.

But if you can't fix it, what are your options?

You could ignore the problem, and hope it goes away — you can hope that the complainant does not have standing to litigate, or does not have the funds even if they did have the standing. You could hope that the adverse publicity would be minimal, and that it will not affect sales seriously.

You could do what you can to redress the problem, short of taking the problematic step, and ask if that will suffice, whilst you re-engineer the product to remove the infringement. But, of course, you might get pushed to withdraw the product, although even this doesn't help you rectify the past infringement.

Or perhaps you are going to have to enter into a formal settlement, perhaps paying damages, and promising to comply in future.

But can you be forced to release source code? Opinions differ on this, but my view is that it would be unlikely that a court would order someone owning copyright in software to make that code available free of charge to remedy a licensing defect. I would have thought that a court might order removal of the product from sale, and perhaps to pay damages. I'm not aware of any case in which source code release was ordered by a court. Of course, if you are going down the settlement route, then you might be pressurised into release the source code, but that's not quite the same as being ordered to do so.

The best strategy, of course, is not to be in that position in the first place.

[Conclusion]

Conclusion

You can find most of what I've spoken about tonight online in one shape or form — it might not be brought together so nicely, but it's probably out there, and plenty more beside. I'm going to sum up by repeating to two points that I made right at the beginning as, for me, these are the two most important points:

Firstly, FOSS is nothing special. It's just software licensing, with some peculiarities around some of the detail.

Secondly, you don't have to be a geek to give competent FOSS advice. It's a fascinating area of work, it's a growing area of work, there's a great community of lawyers involved in it, and I look forward to meeting many of you in FOSS contexts in the future.