Does IP matter with Open Source Software?

Licensing and best practices

Introduction

Hello, my name is Paul McAdam, director of Source Code Control. In the previous section we looked at some of the implications of open source licensing. And now in this section, we're going to take a little bit of a deeper dive.

So, first of all, let's break some myths. These are some of the things that people get wrong when you think about open source and licensing. Freeware is regularly confused with open source licensing, but it is actually a proprietary licence, where the software is being made available at very little or no cost. Similarly, shareware is a proprietary licence. It's usually where the software is offered for a limited period of time at very little or no cost. Once that expires, the intention is that the software company will follow up and offer the customer a paid for licence. We also have the concept of non-commercial software, and this quite often bleeds into the world of open source – technically, it's not an open source licence, but it often used for non-commercial use. The best example is probably the creative commons set of licences. We saw in one of the earlier modules a piece of software that was available on the CC-BY-NC licence, NC meaning not for commercial use. We also have public domain software which is not in any way protected by law. The software is simply put out for the public to consume. I would encourage not to get involved in either the publishing of public domain software, or the consumption of public domain software because there is no indemnification and there's no limitation on the warranties. And then finally we have the morality licences, which attempts to limit the field of use in order to support something that the software publisher holds dear. So for example, we have the HESSLA licence, which attempts to prevent the use of the software in fields which contravene human rights. We also have the Xineo licence, which attempts to prohibit the use of the software for military purposes.

Another couple of interesting licences. Firstly the EXT JS licence, where the software publisher is trying to support animal rights and within the licence excludes any use to which the software could be put almost in conjunction with animals of any sort. And secondly, very popular, the JSON licence, which states quite clearly the software will be used for good and not evil. Unfortunately, although well intended, the licence is very subjective as not everybody agrees on what the definition of evil would be.

Now the next set of licences are definitely what you'd perhaps call an anti-licence, people demonstrating that a licence can be used for a multitude of different purposes.

So, first of all, we have the Beer-ware licence, which clearly says as long as you retain this notice, you can do whatever you want with this stuff. But if we meet some day and you think that this stuff is good, then you can buy me a beer. I can certainly applaud the sentiment in that one.

And then we have probably the more complex one, which is well worth looking up on YouTube, the Chicken Dance licence, which requires, for every use of one thousand units, then somebody has to perform the chicken dance. Over 20,000 units, you have to record on video performing the chicken dance, and then finally, any employee is not allowed to use the word plinth in public. That's a little bit restrictive, I think.

Now the next one is the WTFPL, it's very much an anti-licence and we saw a use of this in the 'Rate my Sandwich' application in an earlier module. Although the sentiment is good, and what they're trying to do is to make a statement saying, 'licences are overly complicated,' clearly this isn't helpful for the consumer. And also I would like to point out that there is no limitation of warranties and there is no indemnification applied for this licence.

Now we are clearly advocating management of good licences and understanding the licences which are attached to each of the components. On a more serious note, one of the things you do have to be careful with is license compatibility. You can get into the situation where actually you've adopted a couple of different components with different licences and you can't achieve your aims. In the example on your screen, my intention is to publish under the Apache 2.0 licence, but because I have a single component which has the GPL licence attached to it, I can't achieve my desired outcome. If you remember from the previous module, this inclusion of a single GPL component requires the publication of the final output to be also covered under the GPL licence – and therefore I wouldn't be able to achieve what I wanted to achieve and publish under Apache.

Another consideration is multi-licensing or dual licensing. Now this isn't to be confused with when software is published with multiple licenses attached to it. In that scenario, you have to adhere to each of the licences. This is where the consumer has a choice as to which licence path they follow.

So here's a great example of dual licensing from a company called LexPredict. If you follow the decision tree on your screen, if the customer wants to resell this software to third parties, then they have to follow the next choice, if they are not going to do that, then there are no fees due. If they are going to sell the software to third parties, then they are asked the next question: will you keep the derived work open? If they are going to continue on down the open source path, then, again, there are no fees due. If they are not going to keep their work open, then an annual subscription fee is due with half of the money going to a trust, which then goes on to grant money for open source projects, and half the money goes to the publishing organization. The idea being, the organization firmly believes in open source software and it is trying to encourage and proliferate that for the future.

So when you are considering your own software publishing and the components included in your solution, you need to think about the licensing triggers and there are four key licensing triggers. Distribution, linking, incorporation and modification.

Probably the most common trigger which you will recognise is that of distribution. Now distribution is the dissemination of material to an outside entity. That could be an application which is downloaded to a machine or a mobile device. But it could also be using the software within JavaScript, within a web client, or any of the other wrappers which are receiving the software on a user's machine. Now just be aware that this has been a source of much debate in the open source community. Many of the commentators believe the original licensing, which was written some years ago now left a gap in the terms which allowed software to be triggered by running it on a server. It was therefore not covered under the licence.

So here you have in front of you an example from the GPL version 3 licence, and it clearly states if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients that they too can receive the source code. So very clearly terms relating to the distribution of the software.

I referred to earlier the debate in the open source community about accessing code remotely on another server. The AGPL, or the Affero GPL licence was introduced to try and address that gap. And you can see again on the terms in front of you, it specifically deals with the scenario where you are accessing it remotely through a computer network. This issue is quite commonly called the SaaS issue within open source licences, and many of the internet companies find it very difficult to adhere to this licence. And for that very reason, you can see the example on the right hand side, Google have a policy that the AGPL licence cannot be used within their software.

The next licensing trigger is linking, and this is where a developer might decide to link or join a component to other components within their software solution. Other phrases you might hear them use which should trigger that licensing point in your brain would be static or dynamic linking, paring, combining, utilising, packaging or creating an interdependency.

Now this is quite a complex field and there are many different types of linking, as you have already seen. Now the LGPL licence, or the Lesser GPL licence, again attempts to try and out some definition around that. So you have the example of the LGPL in front of you. It's the lesser GPL licence, but sometimes it's also called the 'library' GPL licence, because of the use to which it is quite often put. You can see that what it's trying to do is to draw a distinction between software which is statically linked and that which is dynamically linked, which is quite often the scenario in terms of linking a piece of software to a library. You just have to be careful of that distinction drawn in the LGPL licence when you are architecting your solution.

And then we have two final licensing triggers. Modification and incorporation. Now these two triggers are the most common scenarios you would expect to see as you build out your development and they'll be covered in a lot more detail in the next module.