OPEN SOURCE FOR MOBILE APPS

DISCLAIMER

This document is intended as an overview and guide but is not written with any specific set of circumstances in mind. No action should be taken, nor decision made not to take action, based on the content of this document, without taking skilled legal advice in the relevant jurisdictions.

Prepared by Orcro Limited in partnership with Moorcrofts LLP
orcro.co.uk and moorcrofts.com
November 2021

TABLE OF CONTENTS

1. MOBILE APP DEVELOPMENT PROCESS AND WHY OPEN SOURCE IS IMPORTANT

Mobile apps have become an integral part of people's lives, app popularity having increased exponentially with smartphone uptake. The market value of the mobile app economy has also grown exponentially, driven by a huge community of software developers.

Statistics abound on how mobile apps pervade every walk of life, from health, entertainment, banking and financial services, to just about anything developers can imagine. Mobile apps have become an essential tool through which a company's customers access products and services. Creative app development is a global phenomenon, found in most countries worldwide. It is, therefore, essential for companies to remain within the legal frameworks of all the jurisdictions in which they operate,[1] as frameworks differ among countries.

The tool of the World Intellectual Property Organization (WIPO) outlined in the current document is intended to complement other WIPO materials that pertain to mobile applications and Intellectual Property (IP), including the recent handbook on *Key Contracts for Mobile Applications – a developer's perspective*.[2] The tool described here enables a deeper dive into the topic of open source software in mobile apps.

Open source code (and its similar relative, free software) is software available for use, modification or distribution by anyone free of charge, enabling companies to share the development burden of foundational technologies. Much of the software currently under development around the world falls into this category, to such an extent that most new software cannot be developed without some open source content.

While freely available, open source software must still be used with care, for it remains subject to licenses and, therefore, conditions that, if breached, could result in developers losing the right to use it. Developers could also be subject to injunctions preventing them from distributing their apps at all, and possibly a claim for damages. Typical license conditions include the requirement to retain any copyright notices, license notices and disclaimers, all of which are

---

[1] WIPO – Intellectual Property and Mobile Applications - January 2018 - https://www.wipo.int/export/sites/www/ip-development/en/agenda/pdf/ip_and_mobile_applications_study.pdf

[2] Key Contracts for Mobile Apps - a developer's perspective (wipo.int)

relatively easy to meet. More complex are the obligations to provide source code for the code, or possibly the entire app, and to license all of it on the same open source terms. This latter obligation can be disastrous if not properly managed, as it could drain all value from the developer's app and expose it to legal claims.

There are also potential issues with license compatibility, where licenses for components of the same project are incompatible. We cover this issue briefly below. The situation highlights the importance for developers to ensure a process for identifying the open source code being used in their apps, so that developers can ensure compliance with open source licenses.

Ideally, any agencies developing code will have both frameworks demonstrating that they understand open source obligations and mechanisms for fulfilling and documenting those obligations.[3]

[3] One such framework is the Linux Foundation's OpenChain compliance program.

2. WHAT IS OPEN SOURCE?

2.1 The OSI Definition

Open source software is software provided on license terms that allow it to be used, modified and distributed freely. It could be subject to conditions – for example, requiring that attribution notices, disclaimers, notice files, and copies of license text are retained when the software is distributed. There is sometimes the additional requirement that the source code, including that of any other linked software, be made available to any recipient of the code (usually under the same license terms). Technically, open source software is any software licensed in compliance with the Open Source Definition (OSD).[2] The OSD is administered by the Open Source Initiative (see opensource.org).

[2] https://opensource.org/osd

The Open Source Definition

Introduction

Open source doesn't just mean access to the source code. The distribution terms of open source software must comply with the following criteria:

1. Free Redistribution

The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

2. Source Code

The program must include source code and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost, preferably downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

3. Derived Works

The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

4. Integrity of The Author's Source Code

The license may restrict source code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

5. No Discrimination Against Persons or Groups

The license must not discriminate against any person or group of persons.

6. No Discrimination Against Fields of Endeavor

The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

7. Distribution of License

The rights attached to the program must apply to all to whom the program is redistributed, without the need for execution of an additional license by those parties.

8. License Must Not Be Specific to a Product

The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the

program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

9. License Must Not Restrict Other Software

The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open source software.

10. License Must Be Technology-Neutral

No provision of the license may be predicated on any individual technology or style of interface.[3]

The Free Software Foundation (fsf.org) administers a definition of free software that is similar in practice to the open source definition (if software qualifies as open source, it usually will also qualify as free software, and vice versa). From a compliance perspective, it makes little difference if a software component is classified as open source, free software or both, because, ultimately, the aim is to comply with the licensing terms for the software component in question, and in respect of the software component's intended use.

2.2 Examples of where and how open source software is used

Open source is everywhere -- from tiny internet-of-things devices costing a few dollars, to every one of the world's top 500 supercomputers.[4] Every Android mobile phone, tablet and other device runs an open source operating system. Apple uses a great deal of open source in both its MacOS and iOS operating systems. Google's search engine is based on open source code, and open source runs most of the infrastructure of the internet. In 2001, Steve Ballmer, then CEO of Microsoft, described Linux as "a cancer that attaches itself in an intellectual property sense to everything it touches". [5] Now, Microsoft is a fierce advocate of open source; has based its latest web browser, Microsoft Edge, on the open source Chromium project; and is a major contributor to the Linux kernel itself. Why did Microsoft change its mind?

2.3 Why has open source software been so successful?

It is easy to see why companies use open source software. There is an abundance of high quality, well-respected software available that can be used freely by companies, with no charge

[3] The Open Source Definition | Open Source Initiative. Reproduced by permission. Copyright opensource.org. The text of the OSD is licensed under a Creative Commons Attribution 4.0 International License.

[4] https://www.top500.org/statistics/details/osfam/1/

[5] https://www.theregister.com/2001/06/02/ballmer_linux_is_a_cancer/

and minimal restrictions. The more difficult question is why those companies themselves release the code under an open source license. Why does it make sense for those companies to give software away?

There are two possible answers. In some cases, the companies have no choice. If they are building on code under a “copyleft” license (see below), this means that, when the companies distribute the software they have developed, they are required by the underlying copyleft license to release their own code under the same license. However, companies frequently still release open source code under an open source license even when not required to do so by the licensing model. This is typically because the companies want to develop a community of users to share the burden of developing, maintaining and supporting the code base. Used in this way, open source software provides an excellent way for organizations and individuals to participate in collaborative research and development.

Traditional joint development processes tend to require a complex collaboration agreement that governs the obligations for which each of the participants is liable. A collaboration agreement covers such issues as who will finance various parts of development; who will own the intellectual property; and how it may be exploited. Open source development processes are less complicated. They are often based on not much more than the underlying open source license that has been selected for the project, and some governance concerning who can submit and accept changes to the project.

A key driver of the above trend has been the availability of collaborative development platforms such as GitHub and GitLab. Both services are free to use for open source projects, and they enable anyone with a computer and internet connection to participate in collaborative software development. Another important driver has been that the vast majority of software tools needed to develop software are now themselves available free of charge as open source software or are available at no or very low cost. For example, Apple’s XCode environment is free of charge, but joining the developer network costs $99 a year. In fact, many developers develop software using no proprietary software at all. In many cases, you can even use an open source operating system on the computer you use, usually one of the many variations of Linux.

Open source works well for non-differentiating software such as general-purpose databases, operating systems, tools and libraries, plus a great deal of infrastructure. “Non-differentiating” means that a company will not gain a competitive advantage by keeping the technology to itself. No one chooses a bank, for example, because its underlying accounting systems use a particular brand of database. It is, therefore, in a bank’s interest to participate in a joint development project, because that will lower its development costs (by sharing development

and support with other banks and financial institutions) without causing the bank to lose a competitive advantage. Customers might, however, be attracted to a bank that has a particularly easy to use and highly functional mobile app, even though many of the app's components are likely to be open source. For this reason, the bank may decide to collaborate on the open source project implementing its accounting systems, but not open source (or develop a project around) the user interface to its app.

2.4 How open source can be used in app development and why it should be

Open source components can be incorporated into code easily. There is a huge variety of code components available, as well as development environments, tooling and test suites. When selecting a component, consider factors such as performance, functionality, compatibility, reputation, the maturity of the component, and whether it has any vulnerabilities or security issues. You also need to establish whether there is an active community around the component keeping it updated, patched and debugged, and whether the component's roadmap is compatible with the roadmap for your own product.

If you're not using open source, it's very likely that your competitors are. You'll also be spending a lot of time writing code from scratch where you could have downloaded an open source version.

However, even if you find the perfect open source component for your needs, you can't necessarily go ahead and use it without any more thought. You always need to ensure that the component is available under a license that enables you to use it in the way you envisage. To put this in context, see the next section.

Table: 1 – Advantages and Disadvantages of Open Source Software

Advantages of Open Source Software	Disadvantages of Open Source Software
• Easily incorporated into your app. • Easily available (e.g., code, development environments, tooling, test suites). • Reduction in development time/go-to-market time. • Reduction in development costs. • Access to developer communities. • The fact that developers like working in open source. • Not locked into one supplier.	• Third-party support might not be available. • You still must check the code's quality and performance. • Compliance with open source licenses can be complicated. • If the code is not used correctly, it could lead to breaches, enforcement claims or remedial measures that might require the disclosure of trade secrets.

3. OPEN SOURCE — A DEEPER DIVE

3.1 How components are licensed through open source licenses

By definition, open source licenses make the source code of a component available under terms that allow for use, modification and redistribution. There are hundreds of different open source licenses. Each type of license has different requirements. Common requirements include "attribution", meaning that the license requires providing details of the original author or developer of the component if you distribute it; providing a copyright statement preservation; and/or providing a copy of the source code. However, the vast majority of open source code is released under a fairly small subset of licenses.

Many of the most common open source licenses are approved by the Open Source Initiative (OSI) based on its Open Source Definition (OSD) referred to earlier. A complete list of OSI approved licenses is available at http://www.opensource.org/licenses.

These licenses can be grouped broadly into two different categories – permissive and copyleft.

3.2 The types of licenses

3.2.1 Permissive licenses

"Permissive" is a term often used to describe minimally restrictive open source licenses. Take the example of the BSD 3 Clause license. The BSD license allows unlimited redistribution of a component licensed under it, for any purpose, in source or object code form, so long as its copyright notices and the license's disclaimers of warranty are maintained. There is no requirement to provide the source code of the component. The component can be incorporated into any other code, and licensed under any other license (including a proprietary license), provided that the relevant copyright notices and disclaimers are maintained.

By "proprietary", we mean the type of license with which software users are more familiar. For example, the license you buy when you buy a copy of Microsoft Word is a proprietary license. In contrast to an open source license, a proprietary license will contain more restrictions and obligations. These could include payment requirements, restrictions on reverse engineering and transfer of the software, and restrictions on use (such as a restriction on use for educational purposes only). Apple's iOS, for example, is released under a proprietary license. Apple, entirely legally, has taken numerous open source components released under permissive licenses and incorporated them into iOS. If you have an iPhone, go to Settings>General>Legal

& Regulatory>Legal Notices, and you will see the huge number of notices that Apple has provided to comply with the underlying open source license in iOS.

BSD 3 Clause also restricts using the names of contributors, without specific permission, for endorsement of a derived work. A company is not allowed, for example, to advertise a program it has developed containing IBM-developed components licensed under BSD 3 Clause by saying "endorsed by IBM". That would almost certainly be false advertising, in any case.

Other examples of permissive licenses are MIT, ISC and Apache-2.0.

Permissive licenses are also sometimes called "academic", because this licensing model is widely used in academia. Two of the most prominent licenses were developed by the University of California, Berkeley (BSD), and the Massachusetts Institute of Technology (MIT) respectively.

3.2.2 Copyleft licenses

Some licenses require that, if derivative works of a component are distributed, each derivative work must be licensed under the same terms as the original work (or, sometimes, a different specified license). A copy of the source code to the derivative work must also be made available.

This is referred to as a "copyleft" or "reciprocal" effect. An example of a license term establishing reciprocity from the GNU General Public License version 2.0 is:

You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed […] under the terms of this License.

Licenses that include reciprocity or copyleft clauses include all versions of the GPL, LGPL, AGPL, MPL and CDDL.

The extent to which copyleft applies is called the "license scope". Although the traditional way of describing scope is in terms of "strong copyleft" or "weak copyleft", Simon Phipps, a past president of the Open Source Initiative, helpfully distinguishes between "project-scoped" and "file-scoped" copyleft (equivalent to strong and weak copyleft respectively). Broadly, if your app (the project) contains any strong copyleft code, you will be required to release the whole app, and all the components contained within it, under the same strong copyleft component. If it contains any weak copyleft code, you will have to release that component (including any modifications you have made) under the same license and provide access to the source. That

will apply only to the component specified, however, not to the whole app. There are subtle differences between licenses in terms of the detail of the copyleft effect. For example, LGPL is described as weak copyleft, but it contains stringent requirements to allow reverse engineering of the app to which the LGPL component is linked. Therefore, it is essential to understand the license in its entirety in order fully to understand the scope.

There are strong copyleft licenses that have an even stronger effect than that described above. License conditions in open source licenses normally have a significant effect only once the code is distributed. However, some licenses extend this effect so that it applies also when the applicable component is part of the server-side code and its functionality made available over a network -- for example, in a Software as a Service (SaaS). In that case, the component is not being distributed to the end user, but the use by a third party in this case is deemed a form of distribution. AGPL, for example, is similar to GPL but, if you are running a SaaS using an AGPL component, and if you have modified that component, you will need to make the source of it (and derivatives) available to anyone who uses the SaaS. The Open Software License has a similar effect but, in the case described here, it applies even if the underlying component has not been modified.

The main two issues with copyleft licenses are that (1) they could require you to release the source code of your own software developed for your app and license them under the license concerned; and (2) it could be impossible to combine code under different licenses into the same app if the licenses conflict with one another. See the compliance section below for more details.

3.2.3 False friends: licenses similar to open source licenses

Components are sometimes available under a license that has similar characteristics to an open source license but does not meet the open source definition.

This could be because the license does not comply with the strict OSD in a way that does not have any practical effect. For example, the JSON license (used for many components that process the JSON data format) appears to be a liberal open source license. However, it contains the following sentence: “The Software shall be used for Good, not Evil.” This statement is an impermissible field-of-use restriction under a strict interpretation of the OSD (if you accept that “evil” can be regarded as a field of use), so code licensed under the JSON license is not technically open source. However, this stipulation has not stopped JSON licensed code from being widely used. It seems unlikely in practice that any copyright holder of JSON code would start legal proceedings against a licensee for being evil, given that it would be difficult for a court to assess what “evil” means in a particular context.

Another example is the “non-commercial” license. This is a clearer breach of the field-of-use restriction and could have a practical effect. The Creative Commons organization publishes a suite of licenses with slightly different effects. Although not designed for software, they are sometimes used as software licenses. An organization seeking to use code under one of the Creative Commons licenses in a commercial context must avoid any code licensed under a non-commercial license. This is true even if the code is not being distributed (for example, used only internally or to provide a SaaS).

In contrast, code under a Creative Commons license that has a no-derivatives requirement can be used (unless it has a non-commercial restriction), so long as the code is not modified. This is true even though no-derivatives variants are not open source licenses, because they do not allow the licensed materials to be modified.

There are other license categories similar to open source licenses that do not meet the strict definition of open source licenses. Before making use of components, ensure that you are complying with the license restrictions. With open source code, you know that (subject to conditions) you may use the software, modify it and distribute it. With similar but non open source licenses, you might not be able to exercise these rights, even subject to conditions.
Other examples are as follows.

“Freeware” is software distributed under a proprietary license at no cost. You might or might not be provided with source code, and there could be restrictions on redistribution and modification. For example, it is common for major software suppliers to provide freeware code that can be incorporated into your apps to enable them to integrate or communicate with the software supplier’s proprietary product, such as a database engine. There is usually no issue with using this software if you comply with the relevant license terms.

There also exist “source-available” licenses, which explicitly make the source code available on terms that fail to meet all the open source requirements. These licenses can impact use on the server side, even if code is not distributed. Code under a license such as the Commons Clause (which is a small modification to the OSI-approved Apache 2.0 license) cannot be used to provide a commercial SaaS that derives much of its functionality from the licensed code. Other source-available licenses include the MariaDB Business Source License and the Redis Source Available License.

3.2.4 Public domain

You will occasionally come across a component labelled “public domain”. An author’s intention in this instance is to make that component available for use without any restrictions, even such basic restrictions as retaining attribution notices. They are attempting to abandon their copyright in the code entirely. In theory, this means that you can use these components without any compliance obligations. However, the ability of authors to declare voluntarily that their work is in the public domain depends on local law in different countries. It is possible in the United States, but not in the United Kingdom, Germany and many other countries. This places the status of “public domain” code in legal limbo. However, the risk that people declaring their software in the public domain and then taking advantage of a different country’s legal system to claim copyright ownership seems small.

Creative Commons recognized that a more legally sound basis for allowing authors to release their software with as many unconditional rights as possible would be helpful. As a result, it developed the CC0 license, which acts as a public domain dedication in those countries that accept such a thing, and as an extremely permissive license in those that don’t. Software released under CC0 is not technically open source but can generally be treated as being released under an extremely permissive license, with no conditions (for example, no attribution requirements).

The table below summarizes the various types of licenses discussed above.

Table 2 – Summary of licenses

Types of open-source licenses	Meaning	Examples	Description (restrictions, obligations, etc.)
Permissive licenses	Open source licenses with minimal restrictions	BSD 3 Clause license	Unlimited redistribution of a component licensed under it: - for any purpose, - in source or object code form, so long as its copyright notices and license disclaimers of warranty are maintained. No requirement to provide the component's source code. The component can be incorporated into any other code and licensed under any other license (including a proprietary license), provided that relevant copyright notices and disclaimers are maintained. Restriction on use of the names of contributors for endorsement of a derived work without specific permission.
		Apache 2.0	Allows distribution of unmodified, modified or aggregated work; code can be used in works licensed under proprietary licenses. On distribution, any attributions etc. and any NOTICE files must be provided, together with a copy of the license text.
		Other examples of permissive licenses include MIT and ISC.	
Copyleft licenses	If derivative works of a component are distributed, that derivative work must be licensed under the same terms as the original work (or, sometimes, a	All versions of the GPL, LGPL, AGPL, MPL and CDDL.	Distinguish between strong copyleft and weak copyleft: - If your app (the project) contains any strong copyleft code, you will be required to release the whole app, and all the components contained within it, under the same strong copyleft component (GPL, AGPL). - If it contains any weak copyleft code, you must release that component (including any modifications you have made) under the same license and provide access to the source (LGPL, MPL, CDDL).

	different specified license) A copy of the source code to the derivative work must also be made available		- For example, LGPL is described as weak copyleft but it contains stringent requirements to allow reverse engineering of the app to which the LGPL component is linked (LGPL). Some strong copyleft licenses apply even when the applicable component is part of the server-side code and its functionality made available over a network (for example, in a SaaS). (AGPL if you have modified the AGPL-licensed code)
False friends: licenses similar to open source licenses	Components are sometimes available under a license that has similar characteristics to an open source license but does not meet the open source definition	JSON license	"The Software shall be used for Good, not Evil" is an impermissible field-of-use restriction under a strict interpretation of the OSD, so code licensed under the JSON license is not technically open source.
		Non-commercial license	For example, Creative Commons licenses: - For code intended to be used in a commercial context, avoid any code licensed under a non-commercial license (even if no distribution takes place). - However, code under a Creative Commons license that has a no-derivatives requirement can be used (unless it has a non-commercial restriction), so long as the code is not modified. This is true even though no-derivatives variants are not open source licenses, because they do not allow the licensed materials to be modified.
		Freeware	Freeware is software distributed under a proprietary license at no cost. You might or might not be provided with source code, and there could be restrictions on redistribution and modification.
		"Source available" licenses	These explicitly make the source code available on terms that fail to meet all the open source requirements.
		Public Domain	The component is available for use without restrictions, even such basic restrictions as retaining attribution notices. Check local laws and the Creative Commons CC0 license. In practice, the risk is likely to be fairly low.

4. COMPLIANCE

4.1 The importance of generating a software bill of materials (SBOM)

Developers using open source software must establish a process, not only for disclosing the use of open source code, but also for providing, upon delivery, a complete list of all components used and, crucially, a set of compliance materials, including all text and other information required for the licenses. The list of software components is called a “bill of materials”; the compliance materials (including notice files, license texts, attribution notices and disclaimers) are called “compliance artifacts”. It is difficult to undertake any form of compliance exercise without a complete and trustworthy bill of materials.

4.2 Complete software bill of materials

As the software is developed, keep a comprehensive list of every component, its name, source, version number, and the license attached to it. The SBOM is the starting point for any compliance exercise. If a third party is developing the software for you, ensure that that party provides you with a current SBOM for your software, and that the SBOM is updated every time a new version is released.

The SBOM should incorporate, or be accompanied by, a text file containing all the compliance artifacts distributed with the software as required under open source and third-party licenses. Compliance artifacts include notice files, copyright notices, relevant open source license text and disclaimers. A standard format for defining a SBOM, called SPDX, can be found at https://spdx.dev/.

4.3 What you need to do to comply

4.3.1 Provide attributions

Almost all open source licenses require that copyright notices and similar attributions be retained in source code, and that a copy of them be provided with the code if distributed in binary form. The SBOM should provide any necessary attributions. How the attributions are delivered depends on the license and the mode of distribution. There could be different options. A typical example is in the iPhone, where you can find a set of attributions for the open source code used in iOS by going to Settings>General>Legal & Regulatory>Legal Notices. Other options include printing required artifacts in an accompanying paper instruction manual; distributing a text document containing them alongside the distribution mechanism for the app installer; or providing a persistent URL to a text file containing the attribution notices for a particular software release. Which of these is most

suitable will depend on how the software is distributed and the specific requirements of the licenses covering the various software components.

A copyright notice is a notice in a form such as:

```
Copyright © 2021 Alex Developer
```

Because developers could be employed by a company and write open source code as part of that job, the copyright in the code the developers are writing generally will transfer to the company automatically. In that case, the copyright notice would refer to the company, but the company might also allow the developer to be acknowledged as an author, even though the developer has no ownership rights. A copyright and attribution notice might look like:

```
Written by Alex Developer. Copyright © 2021 Softco Inc.
```

The terms of the license might require copyright notices as well as attributions to be preserved. Even if the license requires only copyright notices to be preserved, it is courteous to acknowledge the developer as an author if the developer was listed in the source code or an applicable notice file.

4.3.2 Notices, license texts and disclaimers

More broadly, open source components might contain “notice” files, which could, according to the applicable license (for example, Apache 2.0) be required to be preserved completely (unless the underlying code has been modified so much that part of the notice file become inapplicable). As well as containing attributions and copyright notices, the notice file might also contain other information about the software (such as whether it has been modified from the original version) and might contain a copy of the applicable license text.

4.3.3 Source code (and offers to provide source code)

Copyleft licenses typically require that the source code to the relevant component be made available. How that source code must be provided depends on the underlying license. Weak copyleft licenses require only that you provide the source code of the relevant component, including any modifications that you might have made. Typically, you would provide access to the source code by supplying a persistent link to it. That link could lead to the GitHub or GitLab site where the underlying project is hosted (assuming that the component in question has not been modified) or to your own repository (where the component has been modified). For GPL family licenses, the most common option is to provide an offer, valid for at least three years from the date the code was

most recently distributed, to provide the source code. You might also wish to provide a download link to the relevant source code. Thus, you will have complied by making the offer, but it is unlikely that you will have to provide the source specifically to the person requesting it, as that person almost always will have downloaded the source from the link. If there is a technical issue with compliance for that source, a link gives you an opportunity to remedy the problem without being in breach.

The source code must be provided in an editable format, and you should not deliberately try to make it difficult to use (for example, by minifying or obfuscating it). You should retain any comments in the code as well. Particularly in the case of GPL family licences, you might also need to provide additional information, such as scripts that control compilation and installation.

4.3.4 License compatibility

License compatibility means that the license terms of all components in a software distribution do not conflict. If one license requires you to do something and another prohibits it (or makes it impossible to do), the licenses conflict and are not compatible if the combination of the two software modules triggers the obligations under a license. For example, GPL-2.0 and EPL-1.0 each extend their obligations to “derivative works” that are distributed. If a GPL-2.0 module is combined with an EPL-1.0 module and the merged module is distributed, that module must (according to GPL-2.0) be distributed under GPL-2.0 only and (according to EPL-1.0) under EPL-1.0 only. The distributor cannot satisfy both conditions at the same time, so the software cannot be distributed. This does not prevent the software from being used internally. In the example given above, the incompatibility of two components arises upon distribution of the software. If the software is not distributed, the question of incompatibility does not arise.
Incompatibility might also arise when the app is incompatible with the terms imposed by a distribution medium. That could apply when you decide to distribute an app (which you must be able to distribute directly, with no compatibility or compliance issues) through an app store.
There are several ways you can resolve a licensing incompatibility. The software component might be available under multiple licenses. If one of the available licenses does not involve incompatibility, that is the license to use.

Another option is to review the software architecture being used and see if the component causing the problem can be replaced with a compliant component. Sometimes, you might even find that some components can be removed because they are unnecessary. Alternatively, you might find a component with equivalent functionality that has a more permissive license. Occasionally, you might find that a different version of the same component (which still contains the appropriate

functionality) is licensed under more satisfactory terms. Other options include contacting copyright holders to ask if they are prepared to grant a specific license that would resolve the incompatibility. However, doing that would flag that you are considering using one of copyright holder's components in breach of the license that was applied to it. You could find that you must pay for a new license, but it is also possible that the copyright holder might be prepared to grant a more liberal license without charge.

In some cases, it could be worth carrying out some detective work on the specific component. It could be the case that the component you are using is a slightly modified version of another component available under a more permissive license. In that case, you might be able to take the earlier, more liberally licensed component and, if necessary, make modifications yourself to enable the component to work with your application.

Finally, you might need to rewrite a particular component from scratch, to ensure that you own all of the copyright in it. Care must be taken when doing that, because you must ensure that you are not inadvertently copying the original component. Essentially, you would have to describe a functional specification of the component in question and then get a developer to rewrite code to that specification without any reference to the original code (and preferably without ever having seen the original code).

Table 3 – Potential remedial measures

Problem	**Solution**
Compliance artifacts incomplete or inaccurate	Prepare and supply correct compliance artifacts.
License incompatible	If the component is multi-licensed, choose a compatible license.
	Select a different component with similar functionality to the incompatible license.
	Substitute a version of the component causing incompatibility with a component that has a compatible license.
	Ask if the copyright owner will grant you a specific license (for which you might need to pay).
	Rewrite the component from scratch without infringing copyright of the original component.

4.3.5 A note on app stores

If developing an app to be distributed through an app store, you should be aware that many app store agreements have provisions that restrict the extent to which open source components can be used. It is, therefore, important to ensure that the components used in a developer's app do not contain code licensed under terms that could cause problems under the app store agreement or be incompatible with the license the developer is using under its customer agreement.

4.3.6 End User License Agreements

An end user license agreement (EULA) is an agreement into which the end user is required to enter before using the software. Typically, when you run an app for the first time as a user, you will be presented with a box where you must register, and that might require you to agree to a license agreement. (Most people do not read the text.) When releasing your own app, it is important to ensure that you do not breach any requirements specified in any of the open source licenses of the components you are using. The GPL family of licenses does not allow you to impose additional restrictions on your obligations. MPL and other licenses stipulate that you are not allowed to license the code under a license that conflicts with its own terms. This means that, depending on the licenses, you must not add terms restricting such things as ability to reverse engineer the code or copy/redistribute it. You might also need to ensure that the EULA contains terms recognizing that, by distributing open source code, you are not implying that the developers of that code are responsible for its performance or other characteristics.

4.4 Other uses for the SBOM

Once you have a software bill of materials, it is easier to comply with several other obligations you might have.

4.4.1 Export control

Many countries have requirements limiting the extent to which certain types of software components can be distributed and exported. This applies particularly to software that includes cryptographic functions. Your SBOM will, in effect, be a database that will enable you to establish whether any of the components could cause issues regarding local law on export control. You might decide that you need to modify your application in order to make compliance easier. That could involve removing components entirely or changing them to different ones that make compliance easier (for example, by using a lower grade of encryption).

4.4.2 Safety and security

Once you have a complete list of all the components contained within your application, you will be able to determine, using publicly available databases, whether any of these components have potential vulnerability or other security issues. Two of these are CVE (Common Vulnerabilities and Exposures) and NVD (National Vulnerability Database), maintained by the National Institute of Standards and Technology (NIST), a physical sciences laboratory and non-regulatory agency of the Department of Commerce in the United States of America. There are other databases, maintained by proprietary organizations, and made available for a cost, which might contain additional vulnerabilities not disclosed in the two databases mentioned above.

4.5 Engaging a developer

Many organizations seeking to develop an app will, rather than developing the software themselves, employ an external developer. Developers will almost always make use of open source software. You must ensure that your developer has a procedure for disclosing the use of open source code and for providing, upon delivery, a complete list of all components used. Crucially, the developer must also provide a set of compliance artifacts, including all text and other information required for the licenses (including notice files, license texts, attribution notices and disclaimers).

When selecting developers, ensure that they are aware of their obligations: that they understand open source licensing and licensing models, and that they must provide a complete SBOM. The SBOM must list any components that have been modified, with the modifications specified. External code might not comprise individual components only but could comprise lines or sections of source code cut and pasted from existing third-party code. The SBOM must specify whether any code snippets have been incorporated into the code being delivered.

Table 2 - Compliance checklist

- Are you using open source? Check the OSI definition.
- Ensure that you require any third-party developer you are using to disclose any open source used.
- Obligate a third party developer to defend any open source related claim.
- Obtain a complete SBOM.
- Ensure compliance with open source license terms.
    - Provide required attributions, notices, license text, disclaimers and source code.
    - Ensure license compatibility among components under different licenses.
    - Comply with app store terms.
    - Ensure that EULA complies with open source licenses.
    - Consider if export-control laws apply and, if so, comply with relevant laws.
    - Consider and address any safety and security vulnerabilities.

- Obtain all compliance artifacts (attributions, license texts, source, etc. as required by the license. Use the Bill of Materials as a guide).

4.6 What can go wrong?

4.6.1 Claims and enforcement

Perhaps the biggest problem distributing an app is the possibility of third parties claiming that the app infringes their rights. In the case of open source software, claims might arise if code is used or distributed in breach of license terms. Such claims can be costly, time consuming and dire from a business perspective if a court issues an injunction preventing an app's sale until the issue is resolved.

Undertaking a compliance exercise -- ensuring that you know what code you are using, what licenses each of the components are under, and what is necessary to comply with each of those licenses -- is intended to avoid the possibility of a claim in the first place. However, no compliance exercise is likely to be perfect, and whether you are developing the code internally or outsourcing development to a third party, there remains the possibility, however slight, of an infringement claim. If you employ a third-party developer, ensure that the contract specifies that the developer is responsible for defending any infringement claims. The contract also must state that the developer is responsible for any claims that arise as a result of the developer's failure to take reasonable care in following applicable practices and procedures.

Note that you are not guaranteed that, just because a component is provided as being under a particular licence, the person distributing it is actually legally permitted to distribute it under that

licence. They may have incorporated other open source (or other) code failed to do their own compliance properly. This is fairly rare, and the chances are that if you are using open source components from a well-established, well-governed project, any potential issues will have been considered and addressed.

4.6.2 How to handle an enforcement claim

Generally, parties seeking to enforce their rights under an open source license aim to make the alleged infringer comply. The parties are less likely to be interested in claiming significant damages (although they might seek to recover legal costs). There are situations where individuals have sought to use a sequence of enforcement claims to generate a financial benefit, but that is relatively rare. No enforcement claim should be handled without taking appropriate legal advice. However, the first step should be to research the person or organization alleging infringement. For example, the goal of the Free Software Foundation or the Software Freedom Conservancy almost certainly will be to obtain compliance. You need to consider all licenses applicable to your code to ensure compliance with all relevant conditions.

If a compliance failure relates to an error or incompleteness in compliance artifacts distributed alongside your app, you will need to refresh them. Note that this could include ensuring that you provide the appropriate source code, together with any required additional information such as scripts that control compilation or installation. If you are unable to comply, possibly because you discover a license incompatibility, you might have to redesign your app to avoid the component causing the incompatibility. That could resolve the situation in terms of future releases but, if the claimant is persistent (and particularly if seeking monetary damages rather than compliance), you might still find yourself liable for prior infringements and, in the worst-case scenario, even find that your license to use the original component is terminated. That would prevent you from using the component in a future release, even if become compliant after legal resolution. These are complex questions that should be addressed with appropriately skilled legal advice.

5. ENGAGING WITH OPEN SOURCE COMMUNITIES

Many open source projects, particularly more complex ones, have developed because of the activities of a community of individuals and organizations that provide development, testing, debugging and governance. If you are using such a component, especially if it is a significant part of your app, you might want to become involved with the development community. Benefits include better understanding of how the code works and what it does; knowing how to correct bugs and other issues that you might find; suggesting additional features and functionality for the component; and accessing community members who can help to support your use of the code. Ultimately, you could become involved in the governance of the code, in such a way that you could influence its feature set and roadmap. Many developers, on an individual level, enjoy getting involved in open source communities. Many communities are extensively supported by conferences and other events such as those organized by the Linux Foundation, in addition to virtual presence and support.

Developers and organizations can participate in an open source community at any level. One of the beauties of open source development is that there is no obligation to become deeply involved. A single bug fix will be welcomed if it does genuinely fix a bug. A note of caution – if a project is made available under a permissive license, you are entitled to take it and modify it without feeding back any of the modifications you make. However, if do this, you are likely to find that your relationship with the community sours, and that the community will be less willing to assist you in future.

A secondary issue is that, if you take a permissively licensed open source project and start making your own private modifications, after a while you will find that you have a separate version from the original. This will mean that you will have to update your code manually with any additional bug fixes and other modifications that have been incorporated into the official version. This is called “forking”. You are likely to find that the effort needed to keep your own fork supported is greater than had you submitted your changes into the official code base and had them accepted. This is called “upstreaming” and is one of the reasons why open source communities still form around software released under permissive licenses, and why businesses are likely to upstream modifications they have developed themselves rather than try to maintain their own forks.

6. OPEN SOURCE AND DIFFERENT-USE CASES

When considering mobile app development, people typically focus on the app itself (the specific piece of code downloaded and run on the mobile device).

However, an app frequently will be supported by a backend server, and the server could also provide some functionality through what is, in effect, an embedded browser. Such an app should, therefore, be considered as the combination of code running on the device plus any code running on a backend server to provide data to the app running on the mobile device, and also potentially a SaaS provision displayed through a browser running on the mobile device. The compliance requirements in each case are slightly different

6.1.1 On the backend (server)

Because the software running on your server is not being distributed to anyone else, and because the conditions contained in almost all open source licenses apply only when the software is distributed, the compliance obligations applicable to services running on your server are relatively light. Some components will still have a compliance obligation even if they are running on a server (for example, those components subject to the AGPL licence if they have been modified). Such situations are discussed above. You might also choose to use components under licenses similar to open source licenses, even if they are not technically defined as open source (see false friends above). Those licenses frequently do have compliance obligations that apply even if you are not distributing the code. You might wish to distribute the code in the future -- for example, if you sell your business, or if you want to license the whole system to another company. For these and other reasons, it is always a good idea to develop and maintain a SBOM. That will enable you to check if all components have been assigned licenses and to undertake such activities as vulnerability checking. It is possible, with care, to combine components that would be incompatible in licensing terms were you to distribute them.

6.1.2 On the device (app)

Any software running on the device will have been distributed, by definition. Therefore, the full terms of all open source licenses applicable to every component contained within the app will need to be compliant. You will not be able to distribute the app if it contains components that have been licensed under incompatible licenses, and you will need to ensure that you create, and make available, a full list of compliance artifacts, which would include appropriate source code and installation instructions.

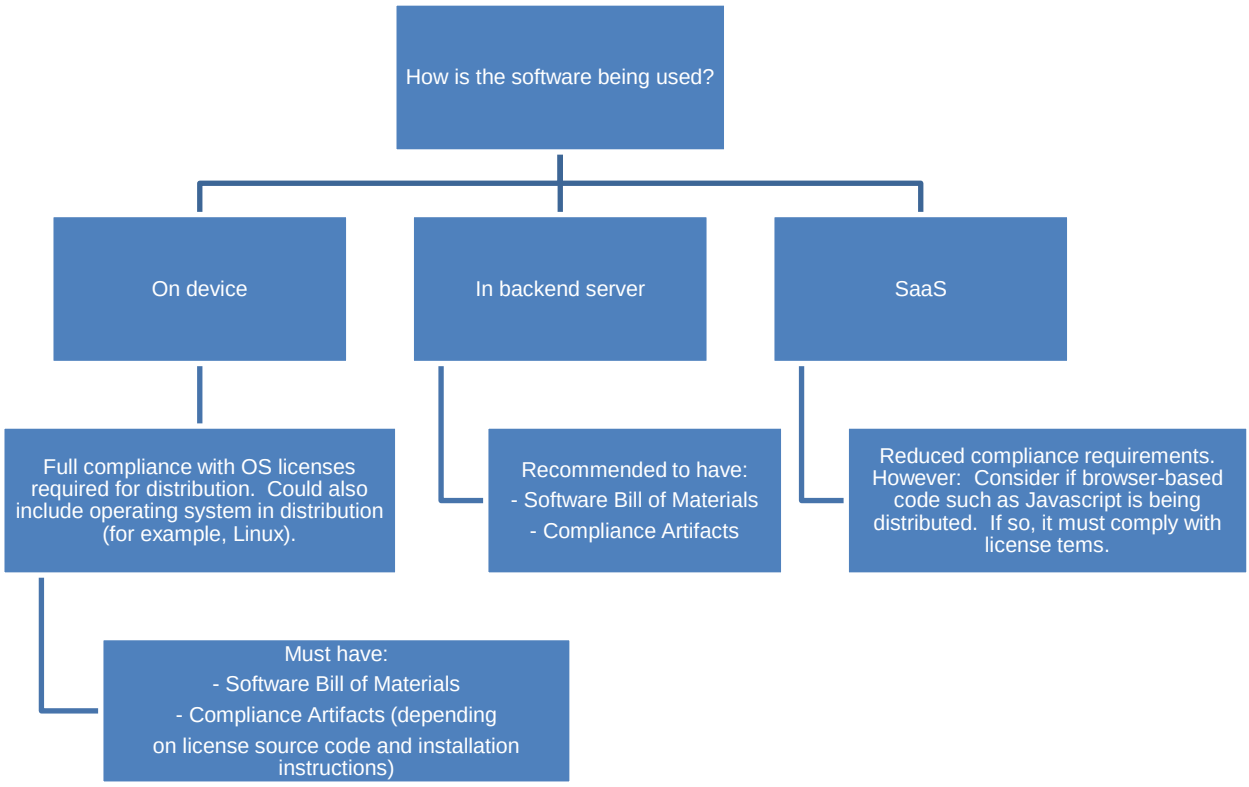
6.1.3 Providing software on a service basis (SaaS)

Even if you provide your software's functionality on a software as a service (SaaS) basis, where the code runs on a server, you might be providing some of the functionality using browser-based code, such as JavaScript. In this case, even though almost all the code would be running server side and would, therefore, be subject to the reduced compliance requirements mentioned above, any code such as JavaScript running on the browser must be treated as distributed code and dealt with accordingly. In many cases, JavaScript is already provided as source code (which could be verified using the browser's "view source" function). However, if that is not the case, it should be regarded as object code and made compliant.

6.1.4 A note on containers

Increasingly, software (particularly that running on the server side) is developed and utilized using container technology, such as Docker. Compliance in this context is beyond the scope of the current document, but it should be noted that, when a container is used, it typically will cause many components to be downloaded automatically from one or more repositories. Any compliance process must ensure that all components are identified and meet requirements.

Flowchart -- Various levels of compliance difficulty



7. OPEN SOURCE AND APP STORES

7.1.1 Overview

Many app store agreements have provisions that restrict the extent to which open source components can be used. It is, therefore, important to ensure that app components do not contain code licensed under terms that could cause problems under the app store agreement or be incompatible with the license the developer is using under its customer agreement.

7.1.2 iOS

Compliance analysis under iOS can be complex. You almost certainly would need specialist advice to ensure compliance with various license obligations, especially when using components released under licenses other than permissive ones.

This is because the tight integration of Apple's XCode development environment with the Apple App Store means that it is not always clear whether the application received by the end user is, essentially, the same application the developer uploaded or if the app has been processed further by Apple. Part of the reason for this is that Apple is rolling out a system called Bitcode, which is designed to lower the footprint of apps when downloaded to different Apple devices, such as iOS running on iPhones and iPads, TVOS running on Apple TVs, and WatchOS running on Apple watches. Bitcode works by downloading only the relevant components tailored to work with the specific device on which the app is to be run. For this reason, in many cases it will be Apple, not the app developer, compiling, packaging and distributing the app to the end user.

Other issues can arise from the use of code under licenses such as LGPL, which require the provision of installation scripts. LGPL has the specific requirement that, if you distribute an app with an LGPL library, any recipient of the app must be able to access the source code to the library, modify it, re-compile the modified source, and re-link that component back into the app. Providing a mechanism to do all this is complex and likely to require that end users have access to their own copies of XCode.

7.1.3 Android

In the case of Android, compliance is reasonably straightforward where the app contains only permissively licensed code. However, once code under various copyleft licenses is incorporated, compliance becomes more complex. The Google Play Store implements some processes to streamline the download and installation progress for Android apps. However, because it is easier

for a user to install an app on an Android device through means other than going through the Google Play Store, the ability to re-compile and re-install a modified version of the app tends to be more straightforward. Different manufacturers of phones and other devices sometimes distribute software through their own app stores, which have their own policies and procedures.

8. OTHER TYPES OF OPEN LICENSES

8.1 Data

Mobile applications frequently make use of data feeds for various purposes. For example, they might use geographic data obtained from databases such as openstreetmap. There are many freely available databases, often offered under similar terms to those for open source licenses. Open source software licenses often are offered for use with databases, although this is not ideal, for various reasons.

Similar issues apply in other related scenarios: you could need to provide attribution for the databases that you use; the databases could be subject to copyleft licenses (normally called sharealike in this context); and you must avoid using databases in a commercial context if they are governed by a non-commercial license.

There are licenses designed specifically to deal with data. Frequently used examples include the Open Data Commons Open Database License and the Linux Foundation's Community Database License. The Creative Commons suite of licenses apply to many databases. Some databases are declared to be in the public domain or subject to CC0, as explained above.

The intellectual property rules that apply to databases vary significantly from jurisdiction to jurisdiction. For example, the European Union and the United Kingdom have implemented a specific database right. The United States of America does not, in general, recognize any intellectual property rights in databases as such. For that reason, it is important to be aware of the specific rights and obligations that attach to the use of databases in the different jurisdictions you intend to use the app and its backend services.

8.2 Content (images, text, music and video)

Many apps use a wide variety of content, including images, text, music and video. This content could be available under an open source-like license. The most common suite of licenses in this case is published by Creative Commons. Avoid using content licensed under a non-commercial license in a commercial context. There probably will be compliance requirements that must be respected, such as making attribution notices available. Often, this cannot be dealt with within the same compliance documentation for open source software provided as part of the app. The equivalent of copyleft in this context is sharealike.

There are various additional rules that need to be considered, such as the requirements of collecting societies, which could require paying royalties, even where the underlying content is available under a license that appears to offer free use. There are rights called “moral rights”, which can apply irrespective of what the license terms stipulate. These rights could require content users to ensure proper attribution. The extent to which these various rules varies significantly from country to country.

9. PATENTS

Whereas copyright applies automatically as soon as anyone writes a piece of software (or creates content such as documentation, images, music and video), some intellectual property rights, such as patents, come into effect only once they have been applied for and registered. A patent is an intellectual property right in an invention. A patent can, in certain circumstances, be granted in relation to inventions that have been implemented in software. For example, various audio and video compression mechanisms have been subject to patent protection. People using any of these mechanisms in an app might need to seek a license from the patent holders, in addition to complying with the license of the underlying software providing video and audio compression/decompression.

The relationship between patents and open source software is complex and outside the scope of the current document.

10. COMPLIANCE STANDARDS — OPENCHAIN

One of the best ways to ensure continued compliance with open source licenses is to adopt an open source development compliance standard such as OpenChain (ISO/IEC 5230:2020), a Linux Foundation project (openchainproject.org). An OpenChain-compliant development program should ensure that developers are aware of all the components contained within the software they are developing. They must be aware of all licenses applicable, understanding the effect of any conditions and preparing the necessary compliance artifacts. Developers must be appropriately trained and due diligence undertaken in the case of third parties used. A policy covering the selection and use of code must be implemented. Appropriate governance mechanisms must be introduced and followed. Necessary resources must be made available. Appropriate records must be kept.

The organization must have a policy for participating in open source communities. Pathways must be developed and implemented to ensure a robust mechanism for fielding queries from internal and external sources. An organization with an Openchain-compliant program will be able to confirm that the software it ships is compliant. Even if your organization does not plan to implement the entire OpenChain specification, it makes sense to use it as a checklist for open source compliance.

If third parties are developing code for your organization, consider making OpenChain compliance a requirement.

11. SUMMARY AND CONCLUSION

Software development makes extensive use of open source code, including the development of mobile apps. This trend is likely to continue because of the undeniable benefits offered by open source components, frameworks, tools and modules. Using open source software significantly reduces development costs, time, and go-to-market time. Other advantages include access to and participation in communities, the benefits of influencing code development and the increased ease with which open source-friendly companies can hire skilled employees.

While open source is freely available and can provide the above benefits, it must be adopted conscientiously, to ensure no breach(es) of underlying licenses or, as a result, potential infringement claims. The process of unravelling the issues can prove time consuming and could delay the app or even require its suspension until the issues are resolved. Such issues can have a significant negative impact on the credibility (and uptake) of the app, and of the company providing it.
Compliance with the license terms of the open source code is, therefore, crucial. Compliance measures should be implemented as early as possible, to ensure that any potential breaches can be identified and rectified in the context of the intended use and licensing model.

The first step towards compliance is to generate a complete software bill of materials (SBOM) detailing all the components (including open source) within the app. The next step is check that the components are available under compatible licenses, given the app's intended distribution mode. Relevant compliance artifacts (including notices, attributions and source code) must be generated and made available. The open source code must be fit for distribution on the chosen app store. The end user license agreement (EULA) must not breach the open source license terms.

Other considerations include understanding if the software is being exported and, if so, if export complies with applicable export legislation. Vulnerabilities and security issues can be addressed once the list of codes is known. Having a complete SBOM is essential.

When engaging a third-party developer, conduct due diligence and ensure that relevant obligations are imposed in terms of disclosure of all open source code used, compliance with respective license terms, plus the provision of an SBOM and relevant compliance artifacts.

Where the app is likely to be valuable to your organization, you might wish to consider following a formally recognized compliance program such as OpenChain.

Many of the issues around open source compliance can be highly technical in legal as well as engineering terms. We hope that the current document is helpful. We also recommend that appropriate legal advice be sought when needed.

Further Recommended Reading

The following websites contain further information that could be helpful.

1. Open Source Initiative: https://opensource.org/
2. Free Software Foundation: https://www.fsf.org/
3. Linux Foundation: https://linuxfoundation.org/fsf.org
4. OpenChain Project: https://www.openchainproject.org/
5. SPDX: https://spdx.dev/

About the Authors

Andrew Katz

Andrew Katz is the CEO of Moorcrofts LLP and Orcro Limited. Moorcrofts LLP was founded in 2000 as a boutique law firm focused on tech business. Orcro Limited is a sister company to Moorcrofts that provides consulting services, particularly for the issues surrounding software supply chain compliance and the Linux Foundation's OpenChain open source compliance program.

Andrew, who also heads the Moorcrofts Technology Department, has been practicing technology law for over 20 years, having previously been a programmer and accredited NeXT developer. He has focused in particular on cloud computing and free and open source software, including the intellectual property issues arising from the incorporation of open source software into software released through the Apple and Google (Android) app stores. He studied science and law at Cambridge University, qualified as a barrister, and subsequently re-qualified as a solicitor in England and Wales. He is also a solicitor (non-practicing) in Ireland and lectures and works extensively worldwide. His commentary, on issues such as the interface between intellectual property rights and software development, has been published by Oxford University Press, Edinburgh University Press and others. He is a visiting researcher at the University of Skövde, Sweden, where he has co-authored several papers, one used as the basis for the Swedish government's procurement policy. Andrew has also co-authored with Usha Guness a handbook on 'Key contracts for mobile apps - from a developer's perspective', for the WIPO in October 2020.

Usha Guness

Usha Guness is a dual-qualified barrister and solicitor working in the technology department at Moorcrofts. She has over fifteen years' experience in the commercial and technology field, including private practice, as well as international experience working for several companies, including a major global telecommunications company. She holds a Master's degree and recently obtained a certificate in US Copyright Law from the Berkman Center at Harvard University. She has co-authored with Andrew Katz a handbook on 'Key contracts for mobile apps - from a developer's perspective', for the WIPO in October 2020.

Acknowledgments

We would like to thank Mr. Dimiter Gantchev, of WIPO, for his meaningful insights on the subject of this note and for his comments and suggestions throughout the drafting process.