**Handouts of Lecture 17 Professional Practices (IT)**

**Lecture Title: Intellectual Property Rights (Cont.)**

**Open Source Software (Cont.)**

**Beneficial Consequences of Open-Source Software**

Advocates of open-source software describe five beneficial consequences of open-source licensing.

The first benefit of open source is that it gives everyone using a program the opportunity to improve it. People can fix bugs, add enhancements, or adapt the program for entirely new uses. Software evolves more quickly when more people are working on it.

Rapid evolution of open-source software leads to the second benefit: new versions of open-source programs appear much more frequently than new versions of commercial programs. Users of open-source programs do not have to wait as long for bug fixes and patches.

A third benefit of open source is that it eliminates the tension between obeying copyright law and helping others. Suppose you legally purchased a traditional license to use a program and your friend asks you for a copy. You must choose between helping your friend and conforming to the license agreement. If the program had an open-source license, you would be free to distribute copies of it to anyone who wanted it.

The fourth benefit is that open-source programs are the property of the entire user community, not just a single vendor. If a vendor selling a proprietary program decides not to invest in further improvements to it, the user community is stuck. In contrast, a user community with access to the source code to a program may continue its development indefinitely.

The fifth benefit of open source is that it shifts the focus from manufacturing to service, which can result in customers getting better support for their software. If source code were distributed freely, companies would make money by providing support, and the companies that provided the best support would be rewarded in the marketplace.

**Examples of Open-Source Software**

Open-source software is a key part of the Internet’s infrastructure, and an increasing number of open-source applications are reaching the desktop. Here are a few examples of highly successful programs distributed under open-source licenses:

BIND provides DNS (domain name service) for the entire Internet.

Apache runs about half of the world’s Web servers.

The most widely used program for moving email about the Internet is the opensource program send mail.

The Android operating system is the world’s best-selling smartphone platform. Firefox and Chrome are the world’s second and third most popular Web browsers, respectively.

OpenOffice.org is an office application suite supporting word processing, spreadsheets, databases, and presentations.

Perl is the most popular Web programming language.

Other popular open-source programming languages and tools are Python, Ruby, TCL/TK, PHP, and Zope. Programmers have long recognized the high quality of the GNU compilers for C, C++, Objective-C, Fortran, Java, and Ada.

**GNU Project and Linux**

The GNU Project and Linux are important success stories in the history of the opensource movement. (GNU is pronounced “guhnew” with the accent on the second syllable. It’s a tradition among hackers to invent recursive acronyms; GNU stands for “GNU’s Not Unix.”) Richard Stallman began the GNU Project in 1984. The goal of the project was ambitious: to develop a complete Unix-like operating system consisting entirely of open-source software.

In order to be fully functional, a modern operating system must include text editors, command processors, assemblers, compilers, debuggers, device drivers, mail servers, and many other programs. During the late 1980s, Stallman and others developed most of the necessary components. The GNU Project also benefited from open-source software previously developed by others, notably Donald Knuth’s TEX typesetting system (used to typeset this book) and MIT’s X Window System. Most of the software developed as part of the GNU Project is distributed under the GNU Public License, an example of an open-source license. (For technical reasons some programs have been distributed as open-source software under other licenses.) In 1991 Linus Torvalds began work on a Unix-like kernel he named Linux. (The kernel is the software at the very heart of an operating system.) He released version 1.0 of the kernel in 1994. Because the other major components of a Unix-like operating system had already been created through the GNU Project, Torvalds was able to combine all of the software into a complete, open-source, Unix-like operating system. To the obvious chagrin of Stallman, Linux has become the commonly accepted name for the opensource operating system based on the Linux kernel. (Stallman urges people to refer to the entire system as GNU/Linux.)

**Impact of Open-Source Software**

In 1998 Andrew Leonard summarized the impact of Linux this way: “Linux is subversive. Who could have thought even five years ago that a world-class operating system could coalesce as if by magic out of part-time hacking by several thousand developers scattered all over the planet, connected only by the tenuous strands of the Internet?”. Linux has become a viable alternative to proprietary versions of Unix. Many companies adopted Linux as a way to cut costs during the recession of 2008–2009. A survey conducted in June 2013 revealed that 95 percent of the world’s 500 fastest supercomputers were running the Linux operating system.

**Critique of the Open-Source Software Movement**

The open-source movement has many detractors. They have raised the following criticisms of the open-source model of software development.

First, if a particular open-source project does not attract a critical mass of developers, the overall quality of the software can be poor.

Second, without an “owner,” there is always the possibility that different groups of users will independently make enhancements to a software product that are incompatible with each other. The source code to a single program may fork into a multitude of irreconcilable versions.

Third, open-source software as a whole tends to have a relatively weak graphical user interface, making it harder to use than commercial software products. This is one explanation why to this point open-source systems have made greater inroads as servers than as desktop systems.

Fourth, open source is a poor mechanism for stimulating innovation. Currently, corporations invest billions of dollars developing new software products. By removing the financial reward for creating new software, companies would sharply curtail or even eliminate research and development. They would no longer be a fountain of new programs. The open-source movement has proven it is able to produce alternatives to proprietary programs (for example, StarOffice instead of Microsoft Office), but it has not demonstrated its ability to innovate completely new products.

**Legitimacy of Intellectual Property Protection for Software**

Licenses for proprietary software usually forbid you from making copies of the software to give or sell to someone else. These licenses are legal agreements. If you violate the license, you are breaking the law.

**Rights-based Analysis**

Not everyone can write good computer programs, and programming is hard work. Programmers who write useful programs that are widely used by others should be rewarded for their labor. That means they should own the programs they write. Ownership implies control. If somebody creates a piece of software, he or she has the right to decide who gets to use it. Software owners ought to be able to charge others for using their programs. Everybody ought to respect these intellectual property rights. This line of reasoning is a variation of Locke’s natural rights argument we discussed at the beginning of the chapter. It is based on the Lockean notion that mixing your labor with something gives you an ownership right in it. Here are two criticisms of the “just deserts”1 argument.

First, why does mixing your labor with something mean that you own it? Doesn’t it make just as much sense to believe that if you mix your labor with something you lose your labor? Robert Nozick gives this example: If you own a can of tomato juice and pour it in the ocean, mixing the tomato juice with the salt water, you do not own the ocean. Instead, you have lost your can of tomato juice. Certainly it would be unjust if someone else could claim ownership of something you labored to produce, but if there were no notion of property ownership and everybody understood when they mixed their labor with something they lost their labor, it would be just. Of course, we do live in a society that has the notion of ownership of tangible property. How can we justify giving a farmer the right to the crop he labors to produce while failing to give a programmer the right to the accounting program he produces for the benefit of the farmer? Still, if we do want to give ownership rights to those who produce intellectual property, we run into the problems we discussed at the beginning of the chapter.

The second criticism of the “just deserts” argument is that Locke’s natural rights argument does not hold up well when extended to the realm of intellectual property. There are two crucial differences between intellectual property and tangible property. Each piece of intellectual property is unique, and copying intellectual property is different from stealing something physical.

**Utilitarian Analysis**

A second argument in favor of providing intellectual property protection for software producers is based on consequences. Failing to provide this protection would have net harmful consequences. The argument goes like this:

When software is copied, it reduces software purchases. If less software is purchased, less money flows to the producers of software. As a result, less new software is produced. As a whole, new software titles benefit society. When the number of new titles drops, society is harmed. Therefore, when software is copied, society is harmed. Copying software is wrong. Copying software causes software sales to drop, which causes the software industry to decline, which causes fewer products to be released, which causes society to be harmed. Logically, all the links in the chain must be strong in order for the argument to be convincing. We will look at each of the links in turn, and we’ll see that none of them are strong. The first claim is that copying software results in reduced sales of software. When talking about software piracy, the computer industry cites the dollar value of the copied software as if each instance of copying represents a lost sale. Obviously this is an exaggeration. Not everyone who gets a free copy of a computer game has the money or the desire to purchase the game for $50. In fact, sometimes software copying may lead to a sale. A person may not have been interested in buying a particular program. After trying it out for free, the person may decide it is so useful she is willing to buy a copy of the program in order to get access to all of the documentation, the technical support line, or another service provided to registered users of the program. It is fair to say that copying software sometimes results in reduced sales of software, but it is not always the case. Hence it is incorrect to make a universal statement.

The second claim is that reduced sales of software result in a decline in the software industry. An argument against this claim is the continued success of Microsoft, despite the fact that software counterfeiting is prevalent in some countries. A better argument against the claim is that it makes a strong cause-and-effect connection between the creation of software and financial remuneration. However, the open-source movement demonstrates many people are willing to create software without being rewarded financially. Some people write programs because they find it fun. Others are motivated by the desire to gain a good reputation by writing a program many people find useful. Advocates of open-source software, including Richard Stallman, suggest that the best way to stimulate innovation is to allow a free exchange of ideas and source code. From this point of view, allowing software producers to control the distribution of their code stifles, rather than promotes, innovation in the software industry.

Finally, the second claim assumes that software customers are solely responsible for the health of the software industry. In reality, other groups want to ensure that there are plenty of new software titles released. Intel, for example, makes its money from selling CPU chips. Every year the chips are faster. If a person owns a computer fast enough to run his current programs, he has little motivation to upgrade the hardware. However, if that same person purchases a new program that requires additional CPU cycles, he may be motivated to upgrade his computer. Hence it is in Intel’s interest to encourage the development of ever more computationally intensive computer programs. Software customers are not solely responsible for promoting the growth of the software industry.

The third claim is that new software packages benefit society. This is a difficult claim to prove. Certainly some programs benefit society more than others. Hence it’s not the number of different programs that matters; it’s what they can be used for. The utility of new software titles must be weighed against the utility of letting people give away copies of programs that would help their friends.

**Creative Commons**

Creative Commons Web site (www.creativecommons.org), which allows you to choose between six different licenses, depending upon your responses to two questions (quoted verbatim):

Allow commercial uses of your work?

Yes

No

Allow modifications of your work?

Yes

Yes, as long as others share alike

No

After you answer these two questions, the Web site creates HTML code containing the appropriate Creative Commons license. You can copy the HTML code and paste it into the appropriate Web page along with your photograph. Visitors to your Web site will be able to see a human-readable summary of the license you have chosen. Commercial artists may choose to use Creative Commons licenses to increase exposure to their work. For example, suppose you take a great photograph of the Golden Gate Bridge. You post it on your Web site with a Creative Commons license indicating the photograph may be used for noncommercial purposes as long as the user gives attribution to you. People from around the world think the image is stunning, and they copy it to their own personal Web sites, giving you credit for the photo. A travel agent in a foreign country sees the image and wants to put it on a travel poster. Since this is a commercial purpose, she must gain your permission before using the image. At that time you can negotiate a fair price for its use. Without the widespread distribution of the image through a Creative Commons license, the travel agent might never have seen it.

The computer-readable versions of the licenses are designed to make it easier for search engines to identify content based upon the particular criteria. For example, a history professor might use a search engine hoping to find an image of the Coliseum in Rome that he could include on his Web site. His purpose is noncommercial, and he is happy to credit the photographer, but he does not want to have to pay to display the image or write a letter asking for the photographer’s permission. A search engine could return only those images that meet these criteria. By 2008 about 130 million different pieces of intellectual property had been distributed using Creative Commons licenses. In 2009 the Creative Commons Attribution Share Alike license became the principal content license for Wikipedia.

***Reference***

***Lecture 17 slides: Intellectual Property Rights(cont.)***

***Gao, Y. (2012). Ethics for the Information Age by Michael J. Quinn. World Libraries, 20(1).***