# MAR GREGORIOS COLLEGE OF ARTS & SCIENCE

Block No.8, College Road, Mogappair West, Chennai – 37

Affiliated to the University of Madras
Approved by the Government of Tamil Nadu
An ISO 9001:2015 Certified Institution

## DEPARTMENT OF COMPUTER APPLICATION

**SUBJECT NAME: OPEN SOURCE TECHNOLOGIES**

**SEMESTER: IV**

**PREPARED BY: PROF.T.INDHUMATHI / PROF.K.RAJALAKSHMI**

**OBJECTIVES:**

 To provide a basic idea of Open source technology, their software

development process to

understand the role and future of open source software in the industry alongwith the impact

of legal, economic and social issues for such software.OUTCOMES:

 To recognize the benefits and features of Open Source Technology and to interpret, contrast

and compare open source products among themselvesUNIT- I

Introduction – Why Open Source – Open Source –Principles, Standards Requirements,

Successes – Free Software – FOSS – Internet Application ProjectsUNIT- II

Open source – Initiatives, Principles, Methodologies, Philosophy, Platform,Freedom, OSSD,

Licenses – Copy right, Copy left, Patent, Zero Marginal Technologies, Incomegeneration

opportunities, Internalization

UNIT- III

Case Studies – Apache, BSD, Linux, Mozilla (Firefox), Wikipedia, Joomla,GCC, Open Office.

UNIT- IV

Open Source Project –Starting, Maintaining –Open Source – Hardware, Design,Teaching &

Media

UNIT- V

Open Source Ethics – Open Vs Closed Source – Government – Ethics – Impactof Open source

Technology – Shared Software – Shared SourceTEXT

BOOK:

1. Kailash Vadera, Bhavyesh Gandhi, ―Open Source Technology‖, Laxmi Publications Pvt Ltd

2012, 1st Edition.

REFERENCE BOOK:

1. Fadi P. Deek and James A. M. McHugh, ―Open Source: Technology and Policy‖, Cambridge

Universities Press 2007.

WEB REFERENCES:

 Coursera online course – Open Source Software Development Methods -

https://www.coursera.org/learn/open-source-software-development-methods

# UNIT I

## Introduction

Open-source technology is a growing trend in GIS, Open-source software is software in which the source code used to create the program is freely available for the public to view, edit, and redistribute. The term "open source" refers to something people can modify and share because its design is  publicly accessible.

The term originated in the context of software development to designate a specific approach to creating computer programs. Today, however, "open source" designates a broader set of value what we call "the open-source way." Open-source projects, products, or initiatives embrace and  celebrate principles of open exchange, collaborative participation, rapid prototyping, transparency, meritocracy, and community-oriented development.

## Open Source

What is open-source software?

Open-source software is software with source code that anyone can inspect, modify, and enhance."Source code" is the part of software that most computer users don't ever see; it's the code computer programmers can manipulate to change how a piece of software a "program" or "application" works. Programmers who have access to a computer program's source code can improve that program by adding features to it or fixing parts that don't always work correctly.

**What's the difference between open-source software and other types of software?**

Some software has source code that only the person, team, or organization who created and maintains exclusive control over it can modify. People call this kindof software "proprietary" or "closed source" software.

Only the original authors of proprietary software can legally copy, inspect, and alter that software. And in order to use proprietary software, computer users must agree (usually by signing a license displayed the first time they run this software) that they will not do anything with the software that the software's authors have not expressly permitted. Microsoft Office and Adobe Photoshop are examples of proprietary software.

Open-source software is different. Its authors make its source code available to others who would like to view that code, copy it, learn from it, alter it, or share it. LibreOffice and the GNU Image Manipulation Program are examples of open-source software.

As they do with proprietary software, users must accept the terms of a license when they use open-source software, but the legal terms of open-source licenses differ dramatically from those of proprietary licenses.

Open-source licenses affect the way people can use, study, modify, and distribute software. In general, open-source licenses grant computer users permission to use open-source software for any purpose they wish. Some open- source licenses some people call "copyleft" licenses stipulate that anyone who releases a modified open-source program must also release the source code for that program alongside it. Moreover, some open-source licenses stipulate that anyone who alters and shares a program with others must also share that program's source code without charging a licensing fee for it.

By design, open-source software licenses promote collaboration and sharing because they permit other people to make modifications to source code and

incoporate those changes into their own projects. They encourage computer programmers to access, view, and modify open-source software whenever they like, as long as they let others do the same when they share their work.

Because early inventors built much of the Internet itself on open-source technologies like the Linux operating system and the Apache Web server application anyone using the Internet today benefits from open-source software.

Every time computer users view web pages, check email, chat with friends, stream music online, or play multiplayer video games, their computers, mobile phones, or gaming consoles connect to a global network of computers using open-source software to route and transmit their data to the "local" devices they have in front of them. The computers that do all this important work are typically located in faraway places that users don't actually see or can't physically access, some people call these computers "remote computers."

More and more, people rely on remote computers when performing tasks they might otherwise perform on their local devices. For example, they may use online word processing, email management, and image editing software that they don't install and run on their personal computers. Instead, they simply access these programs on remote computers by using a Web browser or mobile phone application. When they do this, they're engaged in "remote computing."

Some people call remote computing "cloud computing," because it involves activities (like storing files, sharing photos, or watching videos) that incorporate not only local devices but also a global network of remote computers that form an "atmosphere" around them.

Cloud computing is an increasingly important aspect of everyday life with Internet-connected devices. Some cloud computing applications, like Google Apps, are proprietary. Others, like own Cloud and Next cloud, are open source.

Cloud computing applications run "on top" of additional software that helps them operate smoothly and efficiently, so people will often say that software running "underneath" cloud computing applications acts as a "platform" for those applications. Cloud computing platforms can be open source or closed source. OpenStack is an example of an open-source cloud computing platform.

**Why do people prefer using open-source software?**

People prefer open-source software to proprietary software for a number of reasons, including:

**Control.** Many people prefer open-source software because they have more control over that kind of software. They can examine the code to make sure it's not doing anything they don't want it to do, and they can change parts of it they don't like. Users who aren't programmers also benefit from open-source software, because they can use this software for any purpose, they wish not merely the way someone else thinks they should.

**Training.** Other people like open-source software because it helps them become better programmers. Because open-source code is publicly accessible, students can easily study it as they learn to make better software. Students can also share their work with others, inviting comment and critique, as they develop their skills. When people discover mistakes in programs' source code, they can share those mistakes with others to help them avoid making those samemistakes themselves.

**Security.** Some people prefer open-source software because they consider it more secure and stable than proprietary software. Because anyone can view and modify open-source software, someone might spot and correct errors or omissions that a program's original authors might have missed. And because so many programmers can work on a piece of open-source software without askin for permission from original authors, they can fix, update, and upgrade open- source software more quickly than they can proprietary software.

**Stability.** Many users prefer open-source software to proprietary software for important, long-term projects. Because programmers publicly distribute the source code for open-source software, users relying on that software for critical tasks can be sure their tools won't disappear or fall into disrepair if their original creators stop working on them. Additionally, open-source software tends to bothincorporate and operate according to open standards.

Open-source software programmers can charge money for the open-source software they create or to which they contribute. But in some cases, because an open-source license might require them to release their source code when they sell software to others, some programmers find that charging users money for software services and support (rather than for the software itself) is more lucrative. This way, their software remains free of charge, and they make money helping others install, use, and troubleshoot it.

While some open-source software may be free of charge, skill in programming and troubleshooting open-source software can be quite valuable. Many employers specifically seek to hire programmers with experience working on open-source software.

**Open-Source Principles**

5 Reasons to Use Open Source Software for Your Business

### 1. Its Cost Efficient

Between the cost of the software, licensing, virus protections and ongoingupgrade expenses, the cost of proprietary systems add up quick. Additionally, the software still contains flaws and limits your abilities. With an open sourcesystem, you can side line these costs, all while getting a customized product thatwill ensure growth and productivity.

### 2. It Allows Flexibility

After you make the investment in the proprietary software that you feel best suits your business, you‘re then locked into a system that is concrete, rigid, constantly needs upgrades and may contain unspecified bugs. Open source programs keep an open code so you can constantly go in, rewrite the code so as your business changes and adapts, so will your software system.

### 3. It’s More Secure

With proprietary software no one outside of the company knows how many bugs the program contains. Bugs in open source software tend to get fixed immediately. Versus a program like Microsoft, which typically takes weeks if not months to patch vulnerabilities?

### 4. Problems? No Problem.

With the popularity of open source software, there is plenty of support through forums, and live support chats. For businesses that want extra assurance, there are now paid support options on most open source packages at prices that still fall far below what most proprietary vendors will charge. Providers of commercial support for open source software tend to me more responsive since support is where their revenue is focused.

## 5. A Product You’re Proud Of

Open source allows you to tweak the software to suit your needs. With its open code, it‘s simply a matter of modifying it to add the functionality you want. It puts you in a unique position. This customization allows you to develop the applications quickly, reliably and economically to grow with the expansion of your business.

**Open source drives innovation**

Digital disruption is the norm in today's tech-centric era. Within the technology space, open source is now pervasive, and in 2018, it will be the driving force behind most of the technology innovations.

**Open Source Standards Requirements**

### 1. OpenStack gains increasing acceptance

OpenStack is essentially a cloud operating system that offers admins the ability to provision and control huge compute, storage, and networking resources through an intuitive and user-friendly dashboard.

### 2. Progressive Web Apps become popular

Progressive Web Apps (PWA), an aggregation of technologies, design concepts, and web APIs, offer an app-like experience in the mobile browser.

### 3. Rust to rule the roost

Most programming languages come with safety vs. control trade offs. Rust is an exception. The language co-opts extensive compile-time checking to offer 100% control without compromising safety. The last Pwn2Own competition threw up much serious vulnerability in Firefox on account of its underlying C++language. If Firefox had been written in Rust, many of those errors would have manifested as compile-time bugs and resolved before the product rollout stage.

Rust's unique approach of built-in unit testing has led developers to consider it aviable first-choice open source language. It offers an effective alternative to languages such as C and Python to write secure code without sacrificing expressiveness. Rust has bright days ahead in 2018.

### 4. R user community grows

The R programming language, a GNU project, is associated with statistical computing and graphics. It offers a wide array of statistical and graphicaltechniques and is extensible to boot. It starts where S ends. With the S language already the vehicle of choice for research in statistical methodology, R offers a viable open source route for data manipulation, calculation, and graphical display. An added benefit is R's attention to detail and care for the finer nuances.

### 5. The Internet of Things connects more things

At its core, the Internet of Things (IoT) is the interconnection of devices through embedded sensors or other computing devices that enable the devices (the "things") to send and receive data. IoT is already predicted to be the nextbig major disruptor of the tech space, but IoT itself is in a continuous state of flux.

One innovation likely to gain widespread acceptance within the IoT space is Autonomous Decentralized Peer-to-Peer Telemetry (ADEPT), which is propelled by IBM and Samsung. It uses a block chain-type technology to deliver a decentralized network of IoT devices. Freedom from a central control system facilitates autonomous communications between "things" in order to manage software updates, resolve bugs, manage energy, and more.

### Open Source Successes

The open source program office is an essential part of any modern company with a reasonably ambitious plan to influence various sectors of software ecosystems. If a company wants to increase its influence, clarify its open source

messaging, maximize the clout of its projects, or increase the efficiency of its product development, a multifaceted approach to open source programs is essential. Having viewed the operations of many such teams, I have summarized six common characteristics of successful open-source programs:

1. Marketing is important. Never underestimate the power of a solid marketing plan and branding strategy.
2. Strategically invest in open source communities and ecosystems. Some communities are more in keeping with your technology goals than others.
3. Get strong legal counsel. Without the right legal counsel, an open-source program office can end up placing undue risk on company management. They can also stifle innovation, so strike the right balance.

4. Align with product strategy. If your open source program office is not helping your product strategy, then it's probably a wasted effort.
5. Formulate and communicate your end-user and developer community support strategies and guidelines. Anyone in your company who wants to start or participate in an existing project should understand what a well- run community looks like.
6. Hire a believer. Open source pragmatists are everywhere, but your innovative, forward-thinking, ambitious open source advocate is an extremely valuable rarity. Hire them to run your open source programs if you want to make a difference.

In this article, I'll look at the evolution of the open source program office.

**Defining open source success**

Back when "open source" was a new thing, there was a rush to understand the ramifications of its success. From developers to admins to enterprise executives, everyone was struggling to make sense of a world in which code was givenaway for free.
At the time, some companies started creating departments designed to plot their open source strategies, most notably Google in 2004, although there were precedents at other companies, including IBM, Intel, Oracle, and others. Back then, I assumed that the need for these departments would go away once open source became mainstream. After all, who needs an open source vision or strategy when everyone's using open source software?

That view turns out to have been a bit naive. Despite the popularity of open source, a dearth of experience remains with the ins and outs of open source development and ecosystems in the executive class at most tech companies, including start-ups. Most of the executives at tech companies never cut their teeth in open source communities. They still don't understand many of the motivations for participants, nor do they understand the nuanced differences in licensing models, various types of productization and business models, or how proprietary and open source software can be used in conjunction to create a better product line. Many of them still have the dim (debunked) view that open source projects are used to get software development for free, without paying anyone. They might have an inkling that it's useful for procuring top talent, but they don't quite get that there's a quid pro quo involved: To get something, you have to give something. They don't understand that open-source ecosystems are managed ecosystems with rules that members abide by for the sake of creating a level playing field.

As turns out, in 2016 companies need open-source program offices more than ever. Such an office serves as the nexus for several ongoing simultaneous efforts that require an understanding of open source processes:

**Legal:** Many companies have a mix of licenses, embedded or OEM third-party tools, a patent portfolio, and several trademarks and copyrighted works. This mesh of intellectual property requires planning and forethought when considering adopting a company-wide legal framework that balances immediate company interests (i.e., defending ownership of said IP) with long-term goals ofopen-source initiatives.

**Marketing:** Some people will tell you that open source initiatives don't require marketing. They are wrong. However, the type of marketing required is different from traditional corporate marketing.

**Product management:** Everything you release, whether open source or not, is a product, regardless of whether it drives direct revenue. Better to make sure that the open source products you release augment your overall product portfolio and strategy.

**Engineering:** Do your engineering groups understand the requirements and rules for participating in open source communities? Do they have legal clearance to contribute code? Do they know not to send large patches as their first contribution?

**Customer support:** Your products will require thought devoted to how you support them after release, regardless of whether they directly or indirectly lead to revenue. The open source products you release will need a reliable support model, even if it's "self-serve."

**Community development:** How will you encourage others to participate in your open-source communities? This is not just some Q&A forum, althoughthat is certainly part of it. What's the best governance model? Are you sure that incoming users and developers feel welcome?

**Ecosystem development:** Not to be confused with the above, how will your open source efforts interact with other communities? Do you exist on an island unto yourself, or do you intend your communities to be part of aninterconnected, holistic approach?

These and other categories of activities are only a few things to consider when evaluating how to execute open source initiatives. As open source software becomes more important in your product portfolio, as it almost certainly will better to ensure that it augments your overall company strategy and leads to a magnifying effect.

**FOSS**

he two political camps in the free software community are the free software movement and open source. The free software movement is a campaign for computer users' freedom; we say that a non-free program is an injustice to itsusers. The open source camp declines to see the issue as a matter of justice tothe users, and bases its arguments on practical benefits only.

To emphasize that ―free software‖ refers to freedom and not to price, we sometimes write or say ―free (libre) software,‖ adding the French or Spanish word that means free in the sense of freedom. In some contexts, it works to use just ―libre software.‖

A researcher studying practices and methods used by developers in the free software community decided that these questions were independent of the developers' political views, so he used the term ―FLOSS,‖ meaning ―Free/Libre and Open Source Software,‖ to explicitly avoid a preference between the two political camps. If you wish to be neutral, this is a good way to do it, since this makes the names of the two camps equally prominent.

Others use the term ―FOSS,‖ which stands for ―Free and Open Source Software.‖ This is meant to mean the same thing as ―FLOSS,‖ but it is less clear, since it fails to explain that ―free‖ refers to *freedom*. It also makes ―free software‖ less visible than ―open source,‖ since it presents ―open source‖ prominently but splits ―free software‖ apart.

―Free and Open Source Software‖ is misleading in another way: it suggests that
―free and open source‖ names a single point of view, rather than mentioning two different ones. This conceptualization of the field is an obstacle to understanding the fact that free software and open source are different political positions that disagree fundamentally.

Thus, if you want to be neutral between free software and open source, and clear about them, the way to achieve that is to say ―FLOSS,‖ not ―FOSS.‖ We in the free software movement don't use either of these terms, because we don't want to be neutral on the political question.

As the complexity of software applications grew, this led to greater software package development efforts, making software licensing a market trend. Eventually, developers found techniques to avoid multicomputer software use, such as use of product keys and Internet activation. With the widespread use of the Internet, these techniques became essential for developers to regain profit from their efforts. FOSS surfaced as a result of a need for free, collaborative effort in complicated and expensive projects. Today, many FOSS projects are available for active developers.

**Internet Application Projects**

A web developer needs to upgrade his/her skills consistently with time. The best way to keep improving one's coding skills is by taking up small web app project ideas and developing on the side on weekends.

Such small web app projects are good to try for several reasons like:

- Such projects take very less time
- It‘s a fun way to upgrade your skills
- Learn new development tools and scripts
- You get to try out new categories of web apps
- Such initiatives will look good on your work resume
- You can try new, unconventional development methods
- Be your own boss and work without a safety net on the project
- If the web app is really good, it could turn into a profitable venture

Now you know why such side projects web app ideas are useful for a web developer like you. It‘s time to learn what are you doing wrong while coming up with web application ideas for the project.

The issue is that you unknowingly restrict your capabilities of thinking new web app ideas. For example, we may unintentionally limit the types of projects that we feel are _worthy‘ as side projects or underestimate our skills and remove good web app ideas from the list.

So, I will try to rectify this mistake and give a fair chance to you as well as the best web application ideas that I found after doing detailed research for about
2.5 weeks.

**Best 10 Ideas for Web App Project Development In 2020**

Here are the 10 best web app project ideas of 2020 that you must consider for your web app development side project.

## Lifestyle Web App Ideas

### 1. YouTube Radio Web App

YouTube is the biggest video content platform, with 500 hours of video getting uploaded on the platform every minute. There are so many genres and topics of video content being uploaded that searching what you want to see  becomes more difficult each day.

The YouTube Radio web app solves this issue by providing acustomized/personalized video playlist to its users. The app will study theviewing habits and video searches made by the user and then create a personalized video playlist for the users.

What you as a developer need to do is build a program that recognizes the videocategories and the video keywords and find the most viewed and commented videos matching with the categories and keywords and present them in  a playlist.

### 2. AI-Browser Cookies

Life is so easy due to Google, but what if we can make Google more personalized and life even easier!

What if Google would know what you wish to know even before you ask? That is what AI-browser cookies will help you achieve. It‘ll be a plugin web app that will monitor the users' Google search and daily web habits in detail and create a persona with the results.

With an AI-powered Browser Cookie web app, the users will have a single browser experience across all the devices they own. So, all your browser historyand preferences from various devices will be saved into a single web profile.So, you can continue reading the ―Top 10 Web App Ideas Blog‖ in your home PC which you found on your office system.

### 3. Memes & GIFs Portal

GIFs and Memes are an integral part of internet culture. Everyone from kids to elders indulges in viewing and sharing memes and GIFs with their friends, families, and colleagues. You must have already guessed what I am proposing here.

Develop a web app portal where people can search for Memes and GIFs and getaccurate results. You can either build a repository of memes and GIFs or createa search engine that finds and shows results only for such materials.

### 4. Machine Learning Astrology

Are you thinking of making something simpler than the ones mentioned above? How about an ML-based astrology web app.? It‘s really simple to build, and there is a high potential user base for such a web app.

You may think people don‘t believe in horoscopes and astrology anymore, but you are far from the truth. Almost all the Asian countries follow some sort of astrology in their culture, and there are millions of staunch believers in these countries. And, I haven‘t even added the people from other continents like Europe, Africa, America, and Australia.

The concept is quite simple if you know the basics of Machine Learning. You need to create an algorithm that takes in the information of daily horoscope, findthe pattern of predictions based on the astrological phenomenon‘s and voila! You have a precise ML-based astrology web app.

**Security Web Apps**

### 5. Family Location Tracking App

In present times when the crime rate has risen drastically, the safety of one's family and friends is of utmost importance. Giving a solution to this fear, you can provide a location tracking app that families and friends can use to make sure their loved ones are always safe.

The app may seem invasive to some, but it is a perfect app for families with young kids or elders. With such an app, only the listed contacts can know the location of each other. You can also add an SOS feature using which the user can share their location with a distress call to chosen contacts and local police authorities.

GPS integration is the biggest development module of the app. Once you have cracked the live-location sharing feature, everything else is a piece of cake from thereon.

### 6. Crime Alert Web App

We just talked about crime rates, so here is another web app idea that ensuresthe safety of the general citizens. A crime alert web app works similar to asocial media app like Facebook or Instagram but for crime alerts.

People witnessing a crime in their locality can post about the incident in real- time on this platform, and everyone registered to the app in that location will getnotified about the crime. You can also add features like live share location, pictures, and videos on the web app platform.

This would be a lengthy project, but if done correctly, you can launch this as a startup and begin your career as a technology entrepreneur.

### 7. Social Media Threat Alert

In digital times, digital crimes are also rocketing. So, people need a way to keep themselves safe from online threats like sexual predators, fraud accounts, internet trolls, etc. What you can do is create a web app plugin that warns itsusers whenever a ―threatening account‖ tries to contact them.Imagine it like the truecaller app but for social media accounts. This will be a perfect web app for kids and women who face the most threat online. The app can be built as a director of accounts that people mark as ―threats‖ whenever they face unpleasant encounters with.

## Fun Web Apps

### 8. Review Web App like IMDb

If you don't know what IMDb is (which is almost impossible!), it's the Internet Movie Database (IMDb). General people and movie critiques rate and review movies on this platform. IMDb already exists, but you can create a similar web app platform for something else

Some ideas for a review web app:

- Home Electronics Products
- Restaurants
- Books
- Paintings & Art Pieces
- Salons
- Hotels & Resorts
- Movie Theatre
- Cameras/Mobiles/Laptops
- Songs & YouTube Videos

This are just a few drops of ideas, if you think hard and research well; you'llfind a whole lake of ideas.

### 9. Mood Analyzer App

It may sound like a funny and unreal web app idea, but it's doable. The app, as the name suggests, will detect the mood of the user by tracking their mobile anddesktop activity.

The logic is simple; the app user can add their location, activities, events, etc. which took place in the day and based on their entries and the mobile activity log the web app can deduce the person's mood. Based on the mood, the web app will provide suggestions to the users on how to have a good day.

### 10. Sleep Inducer App

Insomnia is a serious problem that affects 30% of the world population. People try different things to sleep like taking cold showers, drinking warm milk and cookies before sleep, etc. But the most proven technique is sleep-inducingsound waves and hypnotic relaxation music.

So, you will be making a web app that makes white noises like rain, beach, wind, etc. or slow trance music to induce sleep. You can also create such an appfor children in which instead of music, you would play children‘s bedtime stories. The web app‘s UI should be such that with time the lights start to dim, and the sound waves or background music changes

## UNIT 2

### The Open Source Initiative

Open source is a term describing a means of developing and distributing software that ensures software is available for use, modification, and redistribution by anyone. Generally, anyone can download open source softwarefor free or a small fee, and can use, share, borrow, or change it without restriction. Open source practice promotes software reliability and quality by supporting independent peer review and rapid evolution of source code.

The Open Source Initiative (OSI) is a non-profit corporation whose goal is to promote the use of open source software in the commercial world. To accomplish this goal, OSI maintains and promotes the Open Source Definition and offers the OSI Certified Open Source Software Certification Mark and Program. To be OSI certified, the software must be distributed under a license that guarantees the right to read, redistribute, modify, and use the software freely. The Open Source Definition provided by OSI contains the following elements:

- Free redistribution
- Source code
- Derived works
- Integrity of the author's source code
- No discrimination against persons or groups
- No discrimination against fields of endeavor

- Distribution of license
- License must not be specific to a product
- License must not restrict other software
- License must be technology-neutral

You may be attracted to open source software for the following reasons:

- Rapid turnaround with regard to security patches
- Free availability
- Online access to software and source code without a large investment intime or money
- The opportunity to modify and improve source code

**Open Source Methodology**:

Open-source software presents an approach that challenges traditional, closed- source approaches. Post your company‘s source code on the Internet for everyone to see? It seems crazy. But does the open-source approach work? No question about it. It already has worked on Linux, Apache, Perl, send mail, and other programs, and, according to open-source advocates, the approach continues to work marvellously. They will tell you that the software it produces is more reliable than closed-source programs, and defect fix times are remarkably short. Large companies such as Dell, IBM, Intel, Oracle, and SAP seem to agree. They have embraced open source‘s most famous program, Linux, and the Linux development community in particular sets an energetic example for the rest of the world to follow.

Considering that open source is an obvious success, the most interesting software engineering questions are directed toward open source‘s future. Will the open-source development approach scale up to programs the size of Windows NT (currently at least four times as large as the largest estimate forLinux)? Can it be applied to horizontal-market desktop applications as effectively as it has been applied to systems programs? Should you use it for your vertical-market applications? Is it better than typical closed-source approaches? Is it better than the best closed-source approaches? After a little analysis, the answers will become clear.

**The Source of Open Source’s Methodology**

Open-source software development creates many interesting legal and business issues, but in this column, I‘m going to focus on open source‘s software development methodology.

Methodologically, open source‘s best-known element is its use of extensivepeer review and decentralized contributions to a code base. A key insight is that
–given enough eyeballs, all bugs are shallow.‖ The methodology is driven mainly by Linus Torvalds‘ example: Create a kernel of code yourself; make it available on the Internet for review; screen changes to the code base; and, when the code base becomes too big for one person to manage, delegate responsibilityfor major components to trusted lieutenants.

The open-source methodology hasn‘t been captured definitively in writing. The single best description is Eric Raymond‘s –The Cathedral and the Bazaar‖ paper, and that is sketchy at best (http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar.html).
The rest of open source‘s methodology resides primarily in the informal legend, myth, and surrounding specific projects like Linux.

**Bug Me Now or Bug Me Later**

In *Open Sources: Voices from the Open Source Revolution* (O‘Reilly, 1999), Paul Vixie points out that open-source projects use extensive field testing and unmatched code-level peer review. According to Vixie, open-source projects typically have sketchy marketing requirements, no system-level design, little detailed design, virtually no design documentation, and no system-level testing. The emphasis on code-level peer review gives the typical open-source project a leg up on the average closed-source project, which uses little or no review. But considering how ineffective the average project is, comparing open-source projects to the –average‖ closed-source project sets a pointless standard of comparison. Leading-edge organizations use a combination of practices that produce better quality, shorter schedules, and lower development costs than average, and software development effectiveness at that level makes a more useful comparison.

One of the bedrock realities of software development is that requirements and design defects cost far more to correct at coding or system testing time than theycost to correct upstream. The software industry has collected reams of data on this phenomenon: generally, you can expect to spend from 10 to 100 times as much to correct an upstream defect downstream as you would spend to fix the same defect upstream. (It‘s a lot easier to change a line on a design diagram than it is to change a module interface and all the code that uses that module.)As Vixie points out, open source‘s methodology focuses on fixing all bugs atthe source code level—in other words, downstream. Error by error, without upstream reviews, the open-source project will require more total effort to fix each design error downstream than the closed-source project will require to fix it upstream. This cost is not readily perceived because the downstream effort on an open-source project is spread across dozens or hundreds of geographically distributed people.

The implications of open source‘s code-and-fix approach might be more significant than they at first appear. By the time Linux came around, requirements and architecture defects had already been flushed out during the development of many previous generations of Unix. Linux should becommended for its reuse of existing designs and code, but most open-source projects won‘t have such mature, predefined requirements and architecture at their disposal. To those projects, not all requirements and architecture bugs will be shallow.

Open-source advocates claim that giving users the source code reduces the time needed for downstream defect correction the person who first experiences the problem can also debug it. But they have not published any data to support their assertion that this approach reduces overall defect correction costs. For this open source approach to work, large numbers of users have to be both interestedin and capable of debugging source code (operating system code, if the systemin question is Linux), and obviously doesn‘t scale beyond a small cadre ofhighly motivated programmers.

By largely ignoring upstream defect removal and emphasizing downstreamdefect correction, open source‘s methodology is a step backwards—back to Code and Fix instead of forward to more efficient, early defect detection and correction. This bodes poorly for open source‘s ability to scale to projects the size of Windows NT or to brand-new technologies on which insufficient upstream work can easily sink a project.

**Not All Eyeballs Are Shallow**

Open-source advocates emphasize the value of extensive peer review. Indeed, peer reviews have established themselves as one of the most useful practices in software engineering. Industry-leading inspection practices usually limit the number of reviewers to five or six, which is sufficient to produce software with close to zero defects on closed-source projects (Watts Humphrey, *Managing the oftware Process*, Addison Wesley Longman, 1989). The question for open source is, How many reviewers is enough, and how many is too many? Open source‘s typical answer is, –Given enough eyeballs, all bugs are shallow.‖ The more the merrier.

About 1,200 programmers have contributed bug fixes and other code to Linux. What this means in practice is that if a bug is reported in Linux, a couple dozen programmers might begin looking for it, and many bugs are corrected within hours. From this, open-source advocates conclude that large numbers of reviewers lead to ―efficient‖ development.

This answer confuses ―fast‖ and ―effective‖ with ―efficient.‖ To one of those people, the bug will turn out to be shallow. To the rest, it won‘t be shallow, but some people will spend time looking for it and trying to fix it nonetheless. That time isn‘t accounted for anywhere because many of those programmers are donating their time, and the paid programmers don‘t track their effort in any central location. Having several dozen people all looking for the same bug may indeed be fast and effective, but it is not efficient. Fast is having two dozen people look for a bug for one day for a total cost of 24 person-days. Efficient is having one person look for a bug eight hours a week for a month for a total cost of four person-days.

**Economic Shell Game**

A key question that will determine whether open source applies to development of more specialized applications (for example, vertical-market applications) is, Does the open-source methodology reduce development costs overall, or does it just push effort into dark economic corners where it‘s harder to see? Is it abetter mousetrap or an economic shell game?

Considering open source‘s focus on downstream defect correction with significantly redundant peer reviews, for now the approach looks more like a shell game than a better mousetrap. It is appealing at first glance because so many people contribute effort that is free or unaccounted for. The results of this effort are much more visible than the effort itself. But when you add up the total effort contributed—both seen and unseen—open source‘s use of labour looks awfully inefficient.

Open source is most applicable when you need to trade efficiency for speed and efficacy. This makes it applicable to mass-distribution products like operating systems where development cost hardly matters and reliability is paramount. But it also suggests that open source will be less applicable for vertical-market applications where the reliability requirements are lower, profit margins are slim enough that development cost does matter, and it‘s impossible to find 1,200 people to volunteer their services in support of your application.

**One Hit Wonder or Formidable Force?**

The open-source movement has not yet put its methodology under the open- source review process. The methodology is currently so loosely defined that it can hardly even be called a ―methodology.‖ At this time, the strength of the open-source approach arises largely from its massive code-level peer review, and little else. For open source to establish itself as a generalizable approach that applies to more than a handful of projects and that rises to the level of the most effective closed-source projects, it needs to fix four major problems:

1. Create a central clearinghouse for the open-source methodology so it can befully captured and evolved.

2. Kick its addiction to Code and Fix.

3. Focus on eliminating upstream defects earlier.

4. Collect and publish data to support its claims about the effectiveness of theopen-source development approach.

None of these weaknesses in open source‗s current development practices are fatal in principle, but if the methodology can‗t be evolved beyond its current kludgy practices, history will record open source‗s development approach as a one-hit wonder. If open source can focus the considerable energy at its disposal into defining and using more efficient development practices, it will be a formidable force indeed.

**Open Source Philosophy**

Open Source philosophy has been around for a very long time. There are books dedicated to its history, so I will not go into too much detail, as this is not the focus of this document. But basically, it started with hackers such as Richard Stallman, spending huge amounts of time writing software, but instead of selling it for financial gain they wanted to share their work with fellow users (Source: Kidd 2006). They wanted people to learn from what they had made, and improve upon it. Out of respect, any changes that someone makes should begiven back to the hacker community, so everyone can learn more from the additions, therefore improving their skills. This sharing of one‗s ideas and creation was purely based on good moral principles – money, fame and glory did not come into the equation.

The actual term, ―open source‖ however first surfaced on the 3rd of February 1998, during a strategy session in California, after Netscape (creators of an internet browser called Netscape Navigator which sparked the browser wars with Microsoft in the 90‗s), decided to release the code to their software to the world (Source: OSI 2006). They decided that the term ―free software‖ was too confrontational, and ―open source‖ was the best thing they could come up with at the time. So although open source philosophy has existed for many, many years (as stated previously, the open source definition is based on the Debian Free Software Guidelines), it was only in 1998 that it became a recognisable term.

**Popular Open Source Examples**

Even though you may be unaware of it, chances are you make use of open source technology every day. To get you thinking, here are two of the more popular open source examples:

At the time of writing Mozilla Firefox is currently one of the most popular open source software projects going around. With features such as tabbed browsers, integrated search, live bookmarks and a generally faster user experience, it is a lot more feature packed and friendly than proprietary products such as Internet Explorer. Despite the fact that Microsoft and Apple have multimillion dollar development budgets, their browser software is currently being beaten by something developed by the people.

Wikipedia, an open source encyclopaedia is also taking the World Wide Web by storm. Containing over five million articles in a large number of languages, the site is claimed to be one of the top twenty most visited sites on the Internet (Source: Alexa Internet 2006). Anyone can modify the content on Wikipedia, which is one of the key ingredients to its success.

Other Markets are open source philosophy has grown across a large number of different markets not just the computer industry. For example, in the agriculture industry, open source beer has been created, such as Vores Øl and Free Beer
The recipes for both of these beers are freely available on the Internet, released under a Creative Commons License. In the health world, organizations such as the Tropical Disease Initiative have been founded investigating open source pharmaceutical development. Content is also a big area, with websites such as Wikipedia (an open source encyclopaedia) and Yellowikis (an open source version of the Yellow Pages) growing bigger and stronger every day. Wehave already discussed open source software in a fair degree of detail, but open source hardware is also existent. For example, the designs of microchips have been released under open source licensing agreements.

**The Free Software Platform**

Open source refers to something that can be modified and shared as it is accessible for everyone. It originated from a software development term, which implied an approach towards creating computer programs. Today, open-source is more expanded, covering projects, services, products, platforms, and other constituents. Open-source software or platform is the one with source code thata user or other developer can inspect, change, and improve.

"Source code" is the software part invisible for most computer users; the code can be manipulated to change the way a program or application works. It differsfrom other software in the following way:

Some of them have a source code that can be modified by creators. In other words, it‘s a proprietary or closed source software. Only original authors can copy, and change it legally. In order to use a proprietary app, a user must

–agree‖ no to do anything with it unless permitted by the authors. For instance, Microsoft Office is a striking example of proprietary software. On the other hand, in open source software, authors of the source code available to everyone who wants to view, copy, learn, change, or share it. LibreOffice is one of them. There are also terms of use, but they differ dramatically from the first.

The four essential freedoms

A program is free software if the program's users have the four essential freedoms:

- The freedom to run the program as you wish, for any purpose (freedom 0).
- The freedom to study how the program works, and change it so it does your computing as you wish (freedom 1). Access to the source code is aprecondition for this.
- The freedom to redistribute copies so you can help others (freedom 2).
- The freedom to distribute copies of your modified versions to others (freedom 3). By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.

A program is free software if it gives users adequately all of these freedoms. Otherwise, it is nonfree. While we can distinguish various nonfree distribution schemes in terms of how far they fall short of being free, we consider them all equally unethical.

In any given scenario, these freedoms must apply to whatever code we plan to make use of, or lead others to make use of. For instance, consider a program A which automatically launches a program B to handle some cases. If we plan to distribute A as it stands, that implies users will need B, so we need to judge whether both A and B are free. However, if we plan to modify A so that it doesn't use B, only A needs to be free; B is not pertinent to that plan.

―Free software‖ does not mean ―noncommercial‖. On the contrary, a free program must be available for commercial use, commercial development, and commercial distribution. This policy is of fundamental importance without this, free software could not achieve its aims.

We want to invite everyone to use the GNU system, including businesses and their workers. That requires allowing commercial use. We hope that free replacement programs will supplant comparable proprietary programs, but they can't do that if businesses are forbidden to use them. We want commercial products that contain software to include the GNU system, and that would constitute commercial distribution for a price. Commercial development of free software is no longer unusual; such free commercial software is very important. Paid, professional support for free software fills an important need.

Thus, to exclude commercial use, commercial development or commercial distribution would hobble the free software community and obstruct its path to success. We must conclude that a program licensed with such restrictions does not qualify as free software.

A free program must offer the four freedoms to any would-be user that obtains acopy of the software, who has complied thus far with the conditions of the free license covering the software in any previous distribution of it. Putting some of the freedoms off limits to some users, or requiring that users pay, in money or inkind, to exercise them, is tantamount to not granting the freedoms in question, and thus renders the program nonfree.

You may have paid money to get copies of a free program, or you may have obtained copies at no charge. But regardless of how you got your copies, you always have the freedom to copy and change the software, even to sell copies.

**The freedom to run the program as you wish**

The freedom to run the program means the freedom for any kind of person or organization to use it on any kind of computer system, for any kind of overall job and purpose, without being required to communicate about it with the developer or any other specific entity. In this freedom, it is the *user's* purpose that matters, not the *developer's* purpose; you as a user are free to run the program for your purposes, and if you distribute it to someone else, she is then free to run it for her purposes, but you are not entitled to impose your purposes on her.

The freedom to run the program as you wish means that you are not forbiddenor stopped from making it run. This has nothing to do with what functionality the program has, whether it is technically capable of functioning in any given environment, or whether it is useful for any particular computing activity.

For example, if the code arbitrarily rejects certain meaningful inputs—or even fails unconditionally that may make the program less useful, perhaps even totally useless, but it does not deny users the freedom to run the program, so it does not conflict with freedom 0. If the program is free, the users can overcome the loss of usefulness, because freedoms 1 and 3 permit users and communities to make and distribute modified versions without the arbitrary nuisance code.

–As you wish‖ includes, optionally, –not at all‖ if that is what you wish. So there is no need for a separate –freedom not to run a program.‖

**The freedom to study the source code and make changes**

In order for freedoms 1 and 3 (the freedom to make changes and the freedom to publish the changed versions) to be meaningful, you need to have access to the source code of the program. Therefore, accessibility of source code is a necessary condition for free software. Obfuscated –source code‖ is not real source code and does not count as source code.

Freedom 1 includes the freedom to use your changed version in place of the original. If the program is delivered in a product designed to run someone else's modified versions but refuse to run yours a practice known as –tivoization‖ or–lockdown‖, or (in its practitioners' perverse terminology) as –secure boot‖ freedom 1 becomes an empty pretense rather than a practical reality. These binaries are not free software even if the source code they are compiled from is free.

One important way to modify a program is by merging in available free subroutines and modules. If the program's license says that you cannot merge in a suitably licensed existing module for instance, if it requires you to be the copyright holder of any code you add — then the license is too restrictive to qualify as free.

Whether a change constitutes an improvement is a subjective matter. If your right to modify a program is limited, in substance, to changes that someone else considers an improvement, that program is not free.

One special case of freedom 1 is to delete the program's code so it returns after doing nothing, or make it invoke some other program. Thus, freedom 1 includes the –freedom to delete the program.‖

**The freedom to redistribute if you wish: basic requirements**

Freedom to distribute (freedoms 2 and 3) means you are free to redistribute copies, either with or without modifications, either gratis or charging a fee for distribution, to anyone anywhere. Being free to do these things means (among other things) that you do not have to ask or pay for permission to do so.

You should also have the freedom to make modifications and use them privately in your own work or play, without even mentioning that they exist. If you do publish your changes, you should not be required to notify anyone in particular, or in any particular way.

Freedom 3 includes the freedom to release your modified versions as free software. A free license may also permit other ways of releasing them; in otherwords, it does not have to be a copyleft license. However, a license that requires modified versions to be nonfree does not qualify as a free license.

The freedom to redistribute copies must include binary or executable forms of the program, as well as source code, for both modified and unmodified versions.(Distributing programs in runnable form is necessary for conveniently installable free operating systems.) It is OK if there is no way to produce a binary or executable form for a certain program (since some languages don't support that feature), but you must have the freedom to redistribute such forms should you find or develop a way to make them.

**Copyleft**

Certain kinds of rules about the manner of distributing free software are acceptable, when they don't conflict with the central freedoms. For example, copyleft (very simply stated) is the rule that when redistributing the program, you cannot add restrictions to deny other people the central freedoms. This rule does not conflict with the central freedoms; rather it protects them.

In the GNU project, we use copyleft to protect the four freedoms legally for everyone. We believe there are important reasons why it is better to use copyleft. However, noncopylefted free software is ethical too. See Categories of Free Software for a description of how ―free software,‖ ―copylefted software‖ and other categories of software relate to each other.

**Rules about packaging and distribution details**

Rules about how to package a modified version are acceptable, if they don't substantively limit your freedom to release modified versions, or your freedom to make and use modified versions privately. Thus, it is acceptable for the license to require that you change the name of the modified version, remove a logo, or identify your modifications as yours. As long as these requirements are not so burdensome that they effectively hamper you from releasing your changes, they are acceptable; you're already making other changes to the program, so you won't have trouble making a few more.

Rules that ―if you make your version available in this way, you must make it available in that way also‖ can be acceptable too, on the same condition. An example of such an acceptable rule is one saying that if you have distributed a modified version and a previous developer asks for a copy of it, you must send one. (Note that such a rule still leaves you the choice of whether to distribute your version at all.) Rules that require release of source code to the users for versions that you put into public use are also acceptable.

A special issue arises when a license requires changing the name by which the program will be invoked from other programs. That effectively hampers you from releasing your changed version so that it can replace the original when invoked by those other programs. This sort of requirement is acceptable only if there's a suitable aliasing facility that allows you to specify the original program's name as an alias for the modified version.

**Export regulations**

Sometimes government export control regulations and trade sanctions can constrain your freedom to distribute copies of programs internationally. Software developers do not have the power to eliminate or override theserestrictions, but what they can and must do is refuse to impose them as conditions of use of the program. In this way, the restrictions will not affect activities and people outside the jurisdictions of these governments. Thus, free software licenses must not require obedience to any nontrivial export regulations as a condition of exercising any of the essential freedoms.

Merely mentioning the existence of export regulations, without making them a condition of the license itself, is acceptable since it does not restrict users. If an export regulation is actually trivial for free software, then requiring it as a condition is not an actual problem; however, it is a potential problem, since a later change in export law could make the requirement nontrivial and thusrender the software nonfree.

**Legal considerations**

In order for these freedoms to be real, they must be permanent and irrevocableas long as you do nothing wrong; if the developer of the software has the power to revoke the license, or retroactively add restrictions to its terms, without your doing anything wrong to give cause, the software is not free.

A free license may not require compliance with the license of a nonfree program. Thus, for instance, if a license requires you to comply with the licenses of ―all the programs you use‖, in the case of a user that runs nonfree programs this would require compliance with the licenses of those nonfree programs; that makes the license nonfree.

It is acceptable for a free license to specify which jurisdiction's law applies, or where litigation must be done, or both.

**Contract-based licenses**

Most free software licenses are based on copyright, and there are limits on what kinds of requirements can be imposed through copyright. If a copyright-based license respects freedom in the ways described above, it is unlikely to have some other sort of problem that we never anticipated (though this does happen occasionally). However, some free software licenses are based on contracts, and contracts can impose a much larger range of possible restrictions. That means there are many possible ways such a license could be unacceptably restrictive and nonfree.
We can't possibly list all the ways that might happen. If a contract-based license restricts the user in an unusual way that copyright-based licenses cannot, and which isn't mentioned here as legitimate, we will have to think about it, and we will probably conclude it is nonfree.

**Use the right words when talking about free software**

When talking about free software, it is best to avoid using terms like ―give away‖ or ―for free,‖ because those terms imply that the issue is about price, not freedom. Some common terms such as ―piracy‖ embody opinions we hope you won't endorse. See Confusing Words and Phrases that are Worth Avoiding for a discussion of these terms. We also have a list of proper translations of ―free software‖ into various languages.
**How we interpret these criteria**

Finally, note that criteria such as those stated in this free software definition require careful thought for their interpretation. To decide whether a specific software license qualifies as a free software license, we judge it based on these criteria to determine whether it fits their spirit as well as the precise words. If a license includes unconscionable restrictions, we reject it, even if we did not anticipate the issue in these criteria. Sometimes a license requirement raises an issue that calls for extensive thought, including discussions with a lawyer, before we can decide if the requirement is acceptable. When we reach a conclusion about a new issue, we often update these criteria to make it easier to see why certain licenses do or don't qualify.

**Get help with free licenses**

If you are interested in whether a specific license qualifies as a free software license, see our list of licenses. If the license you are concerned with is not listed there, you can ask us about it by sending us emailat <licensing@gnu.org>.

If you are contemplating writing a new license, please contact the Free Software Foundation first by writing to that address. The proliferation of different free software licenses means increased work for users in understanding the licenses; we may be able to help you find an existing free software license that meetsyour needs.

If that isn't possible, if you really need a new license, with our help you can ensure that the license really is a free software license and avoid various practical problems.

**Beyond Software**

Software manuals must be free, for the same reasons that software must be free, and because the manuals are in effect part of the software.

The same arguments also make sense for other kinds of works of practical use — that is to say, works that embody useful knowledge, such as educationalworks and reference works. Wikipedia is the best-known example.

Any kind of work *can* be free, and the definition of free software has beenextended to a definition of free cultural works applicable to any kind of works.

**Open Source?**

Another group uses the term ―open source‖ to mean something close (but not identical) to ―free software‖. We prefer the term ―free software‖ because, once you have heard that it refers to freedom rather than price, it calls to mind freedom. The word ―open‖ never refers to freedom.

**History**

From time to time we revise this Free Software Definition. Here is the list of substantive changes, along with links to show exactly what was changed.

- Version 1.169: Explain more clearly why the four freedoms must apply to commercial activity. Explain why the four freedoms imply the freedom not to run the program and the freedom to delete it, so there is no need to state those as separate requirements.
- Version 1.165: Clarify that arbitrary annoyances in the code do not negate

freedom 0, and that freedoms 1 and 3 enable users to remove them.

- Version 1.153: Clarify that freedom to run the program means nothing stops you from making it run.
- Version 1.141: Clarify which code needs to be free.
- Version 1.135: Say each time that freedom 0 is the freedom to runthe program as you wish.
- Version 1.134: Freedom 0 is not a matter of the program'sfunctionality.
- Version 1.131: A free license may not require compliance with anonfree license of another program.
- Version 1.129: State explicitly that choice of law and choice offorum specifications are allowed. (This was always our policy.)
- Version 1.122: An export control requirement is a real problem if the requirement is nontrivial; otherwise, it is only a potential problem.
- Version 1.118: Clarification: the issue is limits on your right to modify, not on what modifications you have made. And modifications are not limited to ―improvements‖
- Version 1.111: Clarify 1.77 by saying that only retroactive *restrictions* are unacceptable. The copyright holders can always grant additional *permission* for use of the work by releasingthe work in another way in parallel.
- Version 1.105: Reflect, in the brief statement of freedom 1, the point (already stated in version 1.80) that it includes really using your modified version for your computing.
- Version 1.92: Clarify that obfuscated code does not qualify as source code.
- Version 1.90: Clarify that freedom 3 means the right to distribute copies of your own modified or improved version, not a right to participate in someone else's development project.
- Version 1.89: Freedom 3 includes the right to release modified versions as free software.
- Version 1.80: Freedom 1 must be practical, not just theoretical; i.e., no tivoization.
- Version 1.77: Clarify that all retroactive changes to the license are

unacceptable, even if it's not described as a complete replacement.

- Version 1.74: Four clarifications of points not explicit enough, or stated in some places but not reflected everywhere:
    - "Improvements" does not mean the license can substantively limit what kinds of modified versions you can release. Freedom 3 includes distributing modified versions, not just changes.
    - The right to merge in existing modules refers to those that are suitably licensed.
    - Explicitly state the conclusion of the point about export controls.
    - Imposing a license change constitutes revoking theold license.
- Version 1.57: Add "Beyond Software" section.
- Version 1.46: Clarify whose purpose is significant in the freedom to run the program for any purpose.
- Version 1.41: Clarify wording about contract-based licenses.
- Version 1.40: Explain that a free license must allow to you use other available free software to create your modifications.
- Version 1.39: Note that it is acceptable for a license to require you to provide source for versions of the software you put into public use.
- Version 1.31: Note that it is acceptable for a license to require you to identify yourself as the author of modifications. Other minor clarifications throughout the text.
- Version 1.23: Address potential problems related to contract-based licenses.
- Version 1.16: Explain why distribution of binaries is important.
- Version 1.11: Note that a free license may require you to send a copy of versions you distribute to previous developers on request.

There are gaps in the version numbers shown above because there are other changes in this page that do not affect the definition or its interpretations. For instance, the list does not include changes in asides, formatting, spelling, punctuation, or other parts of the page. You can review the complete list of changes to the page through the cvsweb interface.

**software licenses you need to understand**

Different types of software licenses require you to meet certain obligations if you want to reuse the code. Here are 5 common types of software licenses.

If you write code, you also reuse code, including code snippets, libraries, functions, frameworks, and entire applications. All software code comes with certain rights and obligations if you want to add it to your codebase. Free and open source software (FOSS) is free of cost, but you aren‘t free to use it as you wish. Even unlicensed code snippets copied from Stack Overflow have obligations for reuse. But formally developed code usually comes with  a specific software license.

There are many different types of software licenses, and the penalties for license noncompliance can be harsh. If you reuse a component without following the obligations of its license, the licensor might sue, and you might be forced to publish your own source code. To protect your code and your organization, you need to understand these software licenses before using any code, including libraries and frameworks, you didn‘t write yourself. See our list of the top open source licenses and their potential legal risks.

**What are the different types of software licenses?**

Here are five types of common software license models you should know about. Four are examples of open source licenses (which allow you to reuse code to some extent), and one disallows any reuse whatsoever.

**Public domain.** This is the most permissive type of software license. When software is in the public domain, anyone can modify and use the software without any restrictions. But you should always make sure it‘s secure before adding it to your own codebase. Warning: Code that doesn‘t have an explicit license is NOT automatically in the public domain. This includes code snippets you find on the internet.

**Permissive.** Permissive licenses are also known as ―Apache style‖ or ―BSD style.‖ They contain minimal requirements about how the software can be modified or redistributed. This type of software license is perhaps the most popular license used with free and open source software. Aside from the Apache License and the BSD License, another common variant is the MIT License.

**LGPL.** The GNU Lesser General Public License allows you to link to open source libraries in your software. If you simply compile or link an LGPL- licensed library with your own code, you can release your application under anylicense you want, even a proprietary license. But if you modify the library or copy parts of it into your code, you‘ll have to release your application under similar terms as the LGPL.

**Copyleft.** Copyleft licenses are also known as reciprocal licenses or restrictive licenses. The most well-known example of a copyleft or reciprocal license isthe GPL. These licenses allow you to modify the licensed code and distribute new works based on it, *as long as* you distribute any new works or adaptations under the same software license. For example, a component‘s license might say the work is free to use and distribute for personal use only. So, any derivative you create would also be limited to personal use only. (A derivative is any new software you develop that contains the component.)

The catch here is that the users of your software would also have the right to modify the code. Therefore, you‘d have to make your own source code available. But of course, exposing your source code may not be in your best interests.

**Proprietary.** Of all types of software licenses, this is the most restrictive. The idea behind it is that all rights are reserved. It‘s generally used for proprietary software where the work may not be modified or redistributed.

**How do I know what licenses apply to the code in my codebase?**

Before you can determine which licenses govern any reused code in yourcodebase, you need to create a software bill of materials, or a list of all the components in your code. And the fastest way to generate that list is witha software composition analysis tool. A good SCA tool will be able to find full components as well as code snippets, and it‘ll tell you which licenses apply to each piece of code and whether you might be using licenses that have conflicts.

**What is Zero Marginal Cost?**

The cost to serve one additional customer is basically zero.

Businesses often fall under 1 of 2 categories:

1. Linear Business Model
2. Platform Business Model

Platform business models are the next wave of billion dollar companies.

1. A linear business model has increased costs when there is increased demand for their product or service.
2. A platform business model often has zero marginal cost when there is increased demand for its product or service.

**Why do linear business models have increased cost with increased demand?**

Linear business models often create a product or service for distribution. There is a cost associated for the product, production, and distribution. As demand for product grows, so does cost of business.

**Why do platform business models have near zero marginal cost?**

Platform business models facilitate the exchange of value between two or more user groups, typically a consumer and a producer. One of the most powerful aspects of platform business models is their ability to scale without increasing costs. The cost to serve one additional customer is basically zero.

**Why Platforms Scale Better than Linear Businesses:**

To understand how platform business models scale, we need to start with the economics. To create a mobile app, it might cost $250,000 to produce the original version, but creating a copy of that app for user #2 will cost next to nothing. In the language of economics, the app has near-zero marginal cost.

**Example**

Amazon is a Unicorn and this is why:

- Amazon has combined it's platform business model with it's linearbusiness model.
- Amazon linear business model vs. Amazon platform business model:

Read below to learn how Amazon leverages both and see how other companies are benefiting from their platform models too. These are real examples of platform businesses with zero marginal cost advantages. Coincidentally these businesses have achieved unicorn status.

**Amazon's Platform business model:**

Amazon has built an online platform for others to buy, sell and distribute product. Adding products, buyers and sellers to the Amazon platform is an example of zero marginal cost. Amazon accrues almost zero cost to add buyers, sellers and products to it's platform.

More than 80% of all items listed on Amazon are from third-party sellers. Meaning Amazon's online marketplace acts as a platform facilitating the exchange between two or more user groups, typically a consumer and a producer.

**Amazon’s Linear business model:**

Amazon builds and operates numerous distribution facilities for both offline and online commerce. Each facility is equipped with personnel, inventory and machinery. As demand for Amazon‘s business offerings increase so does their operational costs.

Amazon fulfilment services allow sellers to house their inventory in Amazon warehouses and let Amazon handle shipping when a customer places an order. Sellers are charged a fulfilment fee per item and monthly inventory storage cost for each item stored in an Amazon warehouse.

**Income generation opportunities in FOSS**

Opportunities for the companies[edit] Other possible ways of getting money from FOSS include: Dual license model, when the same code is provided both under aggressive FOSS licence (usually GPL) and "commercial" license that does not demand sharing of the derived works.

**Technology growth opportunities**

The number of FOSS projects has grown exponentially. GitHub hosts more than
100 million projects from more than 40 million contributors. Only a tiny percentage of these projects are suitable to be adopted for production use in business-critical systems; millions have been abandoned by their creator(s). Perhaps 0.01% (10,000) of them would satisfy the needs of someone building product-quality software. There are ample evidence and general agreement that the best FOSS code is equal in quality to, if not better than, closed proprietary code.

But there's a lot of room for technology growth. First, there's growing adoption of FOSS technology from programming languages to specialized libraries and from infrastructure software to end-user applications particularly among developers. Next, the size of communities around successful projects (such as Python) is growing quickly, not only with new versions of projects and numbers of maintainers but also with organizations that provide support services, including documentation, translation, and extensions and add-ons, for these technology projects. In some cases, these extensions and add-ons are proprietary, yielding an "open core" approach, where the core retains its FOSS status but customers must pay for some of the add-ons.

Software developers have been on the leading edge of this technology growth. Expensive commercial developer tools have largely been displaced by FOSS tools. An excellent example is the Eclipse environment, supported by contributors to the Eclipse Foundation, which was derived from IBM's commercial VisualAge tools. Microsoft's Visual Studio once sold for as much as US$ 2,000 per user. Today's tools for coding, testing, continuous integration, DevOps, and collaboration are often free, and developers, particularly in start-ups, have chosen them over other options.

There's also a place for new FOSS projects as technology evolves and traditional products become increasingly software-dependent. Automobiles and medical devices fall into this category. Newer automobiles are highly dependent on software in virtually every aspect of their operation, not just for autonomous operation but also for overall efficiency, self-detection of faults, and over-the- air software updates. Today, much of that software is proprietary, but it contains vast amounts of FOSS. For example, many of the entertainment systems offered by commercial airlines are built on Linux. It's possible that regulatory agencies or public sentiment will eventually require life-critical applications to be open.

The venture capitalist Mark Andreesen is known (among other things) forsaying "software is eating the world." That means not only more software but also more societal dependence on that software's secure and proper functioning. There's a great impact on FOSS as user expectations for usability, reliability, and overall quality continue to grow. The role of open source foundations isalso important here because the larger foundations host more projects, involve more contributors in their projects, and maintain governance over those projects, all of which give users greater confidence in the quality and long-term viability of these FOSS projects.

**Employment growth**

These trends point clearly to the need for a big jump in jobs for professionals interested in working with open source. While developers are an obvious need, the range of employment opportunities is much larger and includes quality assurance (QA) and release engineers, engineering managers, support engineers,consultants, service providers, executives, and even legal experts to help with licenses and contracts that involve FOSS. For example, many companies will need to establish an open source project office (OSPO) to keep track of their useof FOSS code and staff it with FOSS-knowledgeable people who can work withinternal groups on the company's use of FOSS and their contributions to various external FOSS projects and organizations, such as the Apache Foundation.Another example is the growth of information services related to FOSS, including publications, conferences, newsletters, blogs, and consultancies.

These requirements suggest the need for expanded educational programs related to FOSS. Many programs are emerging to encourage young people to learn howto code, and it is a straightforward extension to introduce FOSS at an early stage to create a growing pool of talent coming through secondary and university- level educational programs to meet employment needs. This also increases the demand for experienced FOSS-aware professionals to teach technology and related topics to students.

**Business growth**

The growth of FOSS use implies growth opportunities for businesses that develop FOSS and provide FOSS-related services, including project hosting, system integration, and commercial support (e.g., training and QA). As noted above, companies and governments will need people within their organizations and in the broader community around the projects that are most important to their businesses.

The economies of the world have natural business cycles, with newer companies growing and often displacing incumbents. For example, early database systems were replaced by relational database systems such as Oracle and DB2, which are now competing for market share with non-relational database applications, many of them open source, such as Apache CouchDBand Neo4j. The former are mature businesses, while the latter are growing at a rapid rate. Employment applicants seek out these growing companies since they not only provide greater internal career advancement opportunities but also the chance to work on leading-edge technology and potentially to benefit from the increased value of such companies as they grow. For example, GitHub and Red Hat employees benefited when these companies were acquired by Microsoft andIBM, respectively, earlier this year.

Technology customers reinforce these patterns since few want to spend their money on the trailing edge of technology. Unless they regularly update their systems, they will fall behind competitors that make more effective use of technology. Walmart, one of the world's largest retailers, ascribes much of its historic growth to the software technology used to manage its supply chain; by contrast, the international retail clothing chain Forever21 recently filed for bankruptcy, with analysts noting that the company made very poor use oftechnology.

These corporate technology buying decisions often create situations where a small number of software vendors come to dominate the market. For a long time, those decisions favored proprietary technology vendors, in part because industry analysts recommended them over FOSS. Now, however, FOSS companies are receiving greater attention, not only because they offer a lower total cost of ownership for their customers, but also because these commercially-focused FOSS businesses have added support services and service level agreements to match traditional vendors. These developments suggest that the leading FOSS companies and products will continue to grow as technology buyers become more comfortable with these newer entries in the software market.

**Investment growth**

With the topic of "growth opportunities" as the Open Source India panel's theme, one natural aspect for discussion involved opportunities for angel investors, business accelerators, venture capitalists, and others to benefit financially from funding commercial FOSS companies or investing in publicly traded companies with a significant role in FOSS.

The OSS Capital website has a tab labelled COSSCI (for Commercial Open Source Software Company Index) that lists more than 40 FOSS companies with annual revenue exceeding US$ 100 million and valuations in excess of US$ 1 billion. All but three of the companies on the list have received venture capital funding averaging US$ 240 million. At that level, not all of the FOSS companies will provide a return to their investors, but the overall data shows that a growing number of knowledgeable investors are acting on the belief that FOSS companies will continue to grow their revenue and profitability, whether they are entries in a new market segment or capturing market share in market segments historically dominated by proprietary vendors.

It's not that technology customers will abandon the systems on which they manage their businesses, but rather that decision-makers in up-and-coming companies will be more likely to choose FOSS solutions than their counterparts in legacy businesses. One can also view IBM's acquisition of Red Hat in this light, using Red Hat's products as a FOSS alternative to IBM's previous generations of software products.

It's clear that such investments of capital can assist companies from their earliest stages through their lifetimes, often resulting in their acquisition or publicoffering. Companies need funding to grow beyond their initial development, especially to generate awareness of their products and services through marketing programs, create auxiliary support services, and invest in further product development in response to customer requirements, changing markets, and evolving platforms. The founders of such companies spend a disproportionate amount of time seeking investment funding and negotiating theterms of such funding since their growth potential depends on their ability to hire and adequately compensate their employees, as well as to launch their products and build the sales and marketing channels to establish themselves in the market.

Over time, though, these companies must generate revenue and profits from customer purchases so that they are no longer dependent on investor funding. Some companies, such as WhatsApp, are acquired while they are very small, as larger companies see their potential, and are then in a position to fund their ongoing growth. Nonetheless, many start-ups companies fail to achieve "lift off" and join the large percentage of start-ups that don't make it past the initial development stage. These companies tend to fail from being underfunded, from poor management decisions related to hiring, and from an inability to identify a market need that will attract customers.

## Unit 3

**Case Studies Apache**

Top 7 Apache Flink Use Cases

Let‘s discuss top 7 real life case studies of Apache Flink-

### a. Bouygues Telecom – Third largest mobile provider in France

The Bouygues Group ranks in Fortune‘s ―Global 500.‖ Bouygues uses Flink for real-time event processing and analytics for billions of messages per day in a system that is running 24/7.

Bouygues chose Apache Flink because it supports true streaming at the API andat the runtime level, thus providing low latency that company was looking for.

In addition, their systems were up and running in shortest duration with Flink ascompared to other solutions (**follow this Flink vs Spark vs Hadoop comparison guide** for more details), which helped them in expanding the business logic in the system.

Bouygues wanted to get real-time insights about customer experience, what is happening globally on the network, and what is happening in terms of network evolutions and operations. For this, its team built a system to analyse network equipment logs to identify indicators of the quality of user experience.

The system handles 2 billion events per day (500,000 events per second) with a required end-to-end latency of fewer than 200 milliseconds (including message publication by the transport layer and data processing in Flink).

This was achieved on a small cluster reported to be only 10 nodes with 1 gigabyte of memory each.

Planning was to use Flink‘s stream processing for transforming and enriching data and pushing back the derived stream data to the message transport system for analytics by multiple consumers.

Flink‘s stream processing capability allowed the Bouygues team to complete the data processing and movement pipeline. While meeting the latency requirement and with high reliability, high availability, and ease of use.

The Apache Flink framework, for instance, is ideal for debugging, and it can switch to local execution. Flink also supports program visualization to help understand how programs are running. Furthermore, the Flink APIs are attractive to both developers and data scientists.

### b. King – The creator of Candy Crush Saga

King – the leading online entertainment company has more than 200 games in more than 200 countries and regions.

Any stream analytics use case becomes a real technical challenge when more than 300 million monthly users generate more than 30 billion events every day from the different games and systems.

To handle these massive data streams using data analytics while keeping maximal flexibility was a great challenge that has been overcome by Apache Flink.

Flink allows data scientists at King to get access to these massive data streamsin real time. Even with such a complex game application, Flink is able to provide out of the box solution.

### c. Zalando – Leading E-commerce Company in Europe

Zalando has more than 16 million customers worldwide and uses Apache Flink for real-time process monitoring.

A stream-based architecture nicely supports a microservices approach being used by Zalando, and Flink provides stream processing for business process monitoring and continuous Extract, Transform and Load (ETL)

### d. Otto Group – World’s second largest online retailer

Otto Group BI Department was planning to develop its own streaming engine for processing their huge data. As none of the open source options were fitting its requirements.

After testing Flink, the department found it fit for crowdsourcing user-agent identification and identifying a search session via stream processing.

**e. ResearchGate – Largest academic social network**

ResearchGate is using Flink since 2014 as one of its primary tools in the data infrastructure for both batch and stream processing. It uses Flink for its network analysis and near duplicate detection to enable flawless experience to its members.

**f. Alibaba Group – World's largest retailer**

Alibaba works with buyers and suppliers through its web portal. Flink‘s variation (called Blink) is being used by the company for online recommendations. Apache Flink provides it the feature to take into consideration the purchases.

That are being made during the day while recommending products to users.This plays a key role on special days (holidays) when the activity is unusually high. This is an example where efficient stream processing plays over batch processing.

**g. Capital One – Fortune 500 financial services company**

Being a leading consumer and commercial banking institution, the company hadthe challenge to monitor customer activity data in real time. They wanted this to detect and resolve customer issues immediately and enable flawless digital enterprise experience.

Current legacy systems were quite expensive and offered limited capabilities to handle this. Apache Flink provided a real time event processing system. It was cost effective and future proof to handle growing customer activity data.

Flink provided advanced analytics on data streams like advanced windowing, event correlation, event clustering, anomaly detection etc.

**Case Study on BSD**

BSD, or Berkeley Software Distribution, is a UNIX operating system developed at the University of California, Berkeley. Started in 1977, it introduced many technologies still in use by operating systems today. It was the first OS of its kind to implement libraries supporting the Internet Protocol, allowing for easy read/write across a network.

Although the project died in 1995, it gave birth to multiple popular derivatives such as FreeBSD, NetBSD, and OpenBSD.we provide on-line coverage of three other systems. The FreeBSD system is another UNIX system. However, whereas Linux combines features from several UNIX systems, FreeBSD is based on the BSD model of UNIX. FreeBSD source code, like Linux source code, is freely available. The Mach operating system is a modern operating system that provides compatibility with BSD UNIX. Windows is another modern operating system from Microsoft for Intel ...

Companies throughout the world are using and contributing to FreeBSD. Take moment to read more about how these companies are using FreeBSD to great success. Interested in showcasing how your company uses FreeBSD?

- **Mellanox**
- **Netflix**

**Case Study on Linux**

• Unix V6, released in 1975 became very popular. Unix V6 was free and wasdistributed with its source code.

• In 1983, AT&T released Unix System V which was a commercial version.

• Meanwhile, the University of California at Berkeley started the developmentof its own version of Unix. Berkeley was also involved in the inclusion of TransmissionControl Protocol/Internet Protocol (TCP/IP) networking protocol.

• The following were the major mile stones in UNIX history early 1980's

• AT&T was developing its System V Unix.

• Berkeley took initiative on its own Unix BSD (Berkeley Software Distribution) Unix.

• Sun Microsystems developed its own BSD-based Unix called SunOS and laterwas renamed to Sun Solaris.

• Microsoft and the Santa Cruz operation (SCO) were involved in anotherversion of UNIX called XENIX.

• Hewlett-Packard developed HP-UX for its workstations.

• DEC released ULTRIX.

• In 1986, IBM developed AIX (Advanced Interactive eXecutive).

The most popular Linux distributions are:

- Ubuntu Linux
- Linux Mint
- Arch Linux
- Deepin
- Fedora
- Debian
- openSUSE.

And don‘t think the server has been left behind. For this arena, you can turn to:

- Red Hat Enterprise Linux
- Ubuntu Server
- CentOS
- SUSE Enterprise Linux.

Some of the above server distributions are free (such as Ubuntu Server and CentOS) and some have an associated price (such as Red Hat Enterprise Linux and SUSE Enterprise Linux). Those with an associated price also include support Components of Linux System.

Linux Operating System has primarily three components

**Kernel** − Kernel is the core part of Linux. It is responsible for all major activities of this operating system. It consists of various modules and it interacts directly with the underlying hardware. Kernel provides the required abstraction to hide low level hardware details to system or application programs.

**System Library** − System libraries are special functions or programs using which application programs or system utilities accesses Kernel's features. These libraries implement most of the functionalities of the operating system and do not requires kernel module's code access rights.

**System Utility** − System Utility programs are responsible to do specialized, individual

level tasks.

**Kernel Mode vs User Mode**

Kernel component code executes in a special privileged mode called kernel mode with full

access to all resources of the computer. This code represents a single process, executes in single address space and do not require any context switch and hence is very efficient and fast. Kernel runs each process and provides system services to processes, provides protected

access to hardware to processes.

Support code which is not required to run in kernel mode is in System Library. User programs and other system programs works in User Mode which has no access to system hardware and kernel code. User programs/ utilities use System libraries to access Kernel functions to get system's low-level tasks.

**Case studies/Mozilla Foundation**

The Mozilla Foundation is a non-profit organization that exists to support and provide leadership for the open source Mozilla project. The organization sets the policies that govern development, operate key infrastructure and control trademarks and other intellectual property. It owns two taxable for-profit subsidiaries: the Mozilla Corporation, which employs several Mozilla developers and coordinates releases of the Mozilla Firefox web browser, and Mozilla Messaging, Inc., which primarily develops the Mozilla Thunderbird email client. The Mozilla Foundation is based in the Silicon Valley city of Mountain View, California, USA.

The Mozilla Foundation describes itself as "a non-profit organization dedicated to preserving choice and promoting innovation on the Internet". Mozilla Europe,Mozilla Japan and Mozilla China are non-profit organizations whose mission is to help promote and deploy Mozilla products and projects. They are independent of, but affiliated with, the Mozilla Foundation.

**Mozilla Corporation**

On August 3, 2005, the Mozilla Foundation launched a wholly owned subsidiary called the Mozilla Corporation to continue the development and delivery of Mozilla Firefox and Mozilla Thunderbird. The Mozilla Corporation takes responsibility for release planning, marketing and a range of distribution- related activities. It also handles relationships with businesses, many of which generate income. Unlike the Mozilla Foundation, the Mozilla Corporation is a taxable entity, which gives it much greater freedom in the revenue and business activities it can pursue.

**Financing**

The Mozilla Foundation accepts donations as a source of funding. Along with AOL's initial $2 million donation, Mitch Kapor gave $300,000 to the organization at its launch. The group has tax-exempt status under IRC 501(c)(3) of the U.S. tax code, though the Mozilla Corporation subsidiary is taxable.

In 2006 the Mozilla Foundation received $66.8 million in revenues, of which $61.5 million is attributed to "search royalties". The foundation has an ongoing deal with Google to make Google search the default in the Firefox browser search bar and hence send it search referrals; a Firefox themed Google search site has also been made the default home page of Firefox. A footnote in Mozilla's 2006 financial report states "Mozilla has a contract with a search engine provider for royalties. The contract originally expired in November 2006, however Google renewed the contract until November 2008 and has now renewed the contract through 2011. Approximately 85% of Mozilla‘s revenue for 2006 was derived from this contract."; this equates to approximately US$56.8 million.

## Case Study on famous Open Source Projects

### Wikipedia

**Wikipedia** is no less than an encyclopedia available free of cost to the public nowadays. If you want to write a passage, know about some famous person orthing you are just one click away from your desired article.

**Wikipedia** got registered with domain Wikipedia.com on January 12, 2001, andwith the domain Wikipedia.org on January 13, 2001, before it launched on January 15, 2001. Jimmy Wales and Larry Sanger were the founders of Wikipedia. Wikimedia is the founding organization of Wikipedia.

The name **Wikipedia** holds great meaning in itself, In Hawaiian language wikimeans quick and pedia means learning. The world-famous Alexa from amazonhas ranked Wikipedia on the 5th position in the most visited website category. Before Wikipedia, many such concepts were introduced but none of them worked for Nupedia, the first one to come up with the concept of an online encyclopedia. They developed a free encyclopedia online which was English- language based whose articles were jotting down by industry experts and wentunder review before publication on the internet.

**Wikipedia** being an Open Source took up the lead as it allowed everyone usingit to contribute for the same. Wikipedia touched the milestone of more than 42Million distinct visitors on its site in the year of 2007 and featured for the very first time in the top10 most visited websites and has been there since then. Wikipedia showed constant growth and was appreciated by each individual whoused it. In the years 2004 and 2005, Wikipedia won numerous awards and got a lot of acknowledgment. In 2005, it turned into a top100 site as per Alexa.com.
Wikipedia's one-millionth article was released in March of 2006 and only threeyears that number would develop to an aggregate of thirteen million articles accessible in multiple languages.

**Wikimedia** made Wikipedia a not for profit non-commercial organization justto provide people a platform where they can search for anything. Wikipedia founder pledged never to have promotions or any kind of advertisements on itssite, which is a pledge they have kept. Wikipedia highlights a simple search engine and a reliable format.

**Wikipedia** utilizes MediaWiki software to run its site. Its primary U.S. server farm, comprising of around 300 servers, is situated in Tampa, Florida.
Wikipedia additionally has a European server farm in Amsterdam, EvoSwitch, where they have around 50 servers. There are additionally provisional designs to verify another server farm in Virginia. Wikipedia as of now utilizes Ubuntu Linux 2.6 as its OS with a standard LAMP bundle.

## **Joomla**

Joomla is an award winning CMS used for developing different applications on the web (Rahmel, 2009). Joomla make way into the web development software in 2005 when the name Joomla was announced by Open Source Matters-a non-profit organisation. Joomla is an English spelling of a Swahili word jumla which means ―As a whole‖ or ―altogether‖.
Joomla is used to build small and large web applications; it offers nice and extensible web site functionalities. Joomla has been adopted internationally by web developers for building corporate web sites, corporate intranets and extranets, news publishing, ecommerce, and NGO web sites. Joomla ease of use provides features such as: built-in user polling, automatic full text searching, and accessibility option for disabled, and plug-ins for e-commerce.

*Real World Examples of What Joomla Can Create?*

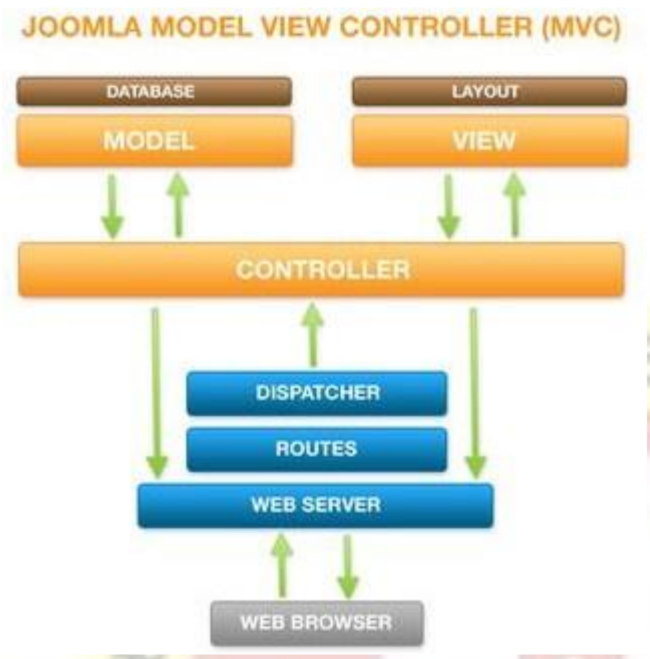
- Corporate web sites or portals
- Corporate intranets and extranets
- Online magazines, newspapers, and publications
- E-commerce and online reservations
- Government applications
- Small business web sites
- Non-profit and organizational web sites
- Community-based portals
- School and religious web sites
- Personal or family homepages

***I need to build a site for a client. How will Joomla! help me?***

oomla! is designed to be easy to install and set up even if you're not an advanced user. Many Web hosting services offer a single-click install or autoinstallers, getting your new site up and running in just a few minutes.

Since Joomla is so easy to use, as a Web designer or Developer, you can quickly build sites for your clients. Then, with a minimal amount of instruction,you can empower your clients to easily manage their own sites themselves.

If your clients need specialized functionality, Joomla! is highly extensible andthousands of extensions (most for free under the GPL license) are available inthe Joomla! Extensions Directory



The core Joomla framework enables developers to quickly and easily build:

- Inventory control systems
- Data reporting tools
- Application bridges
- Custom product catalogs
- Integrated e-commerce systems
- Complex business directories
- Reservation systems
- Communication tools

**GCC-OPENSOURCE**

GCC stands for GNU Compiler Collection. It is a compiler system supporting various Programming languages. It has played an important role in the growthof free software, as both a tool and an example.

GCC open source package is a complete debugger and open source C/C++ compiler toolchain for building and debugging embedded applications based on MSP430 microcontrollers. This compiler supports all MSP430 devices without code size limitations. This compiler can be used standalone from the command-line or within Code Composer Studio v6.0 or later. Get started today whether you are using Windows®, Linux® or Mac OS X® environments.

## *Finding errors in a simple program*

As mentioned above, compiler warnings are an essential aid when programming in C and C++. To demonstrate this, the program below contains a subtle error: it uses the function printf incorrectly, by specifying a floating-point
format '%f' for an integer value:

```
#include <stdio.h>

int
main (void)
{
  printf ("Two plus two is %f\n", 4);return
  0;
}
```

This error is not obvious at first sight, but can be detected by the compiler if thewarning option -Wall has been enabled. Compiling the program above, 'bad.c', with the warning option -Wall producesthe following message:

```
$ gcc -Wall bad.c -o bad
bad.c: In function `main':
bad.c:6: warning: double format, differenttype
arg (arg 2)
```

This indicates that a format string has been used incorrectly in the file 'bad.c' atline 6. The messages produced by GCC always have the form *file:line- number:message*. The compiler distinguishes between *error messages*, which prevent successful compilation, and *warning messages* which indicate possibleproblems (but do not stop the program from compiling).

In this case, the correct format specifier should be '%d' for an integer argument.The allowed format specifiers for printf can be found in any general book on C, such as the *GNU C Library Reference Manual* (see section Further reading).

Without the warning option -Wall the program appears to compile cleanly, but produces incorrect results:

```
$ gcc bad.c -o bad
$ ./bad
Two plus two is 2.585495          (incorrect output)
```

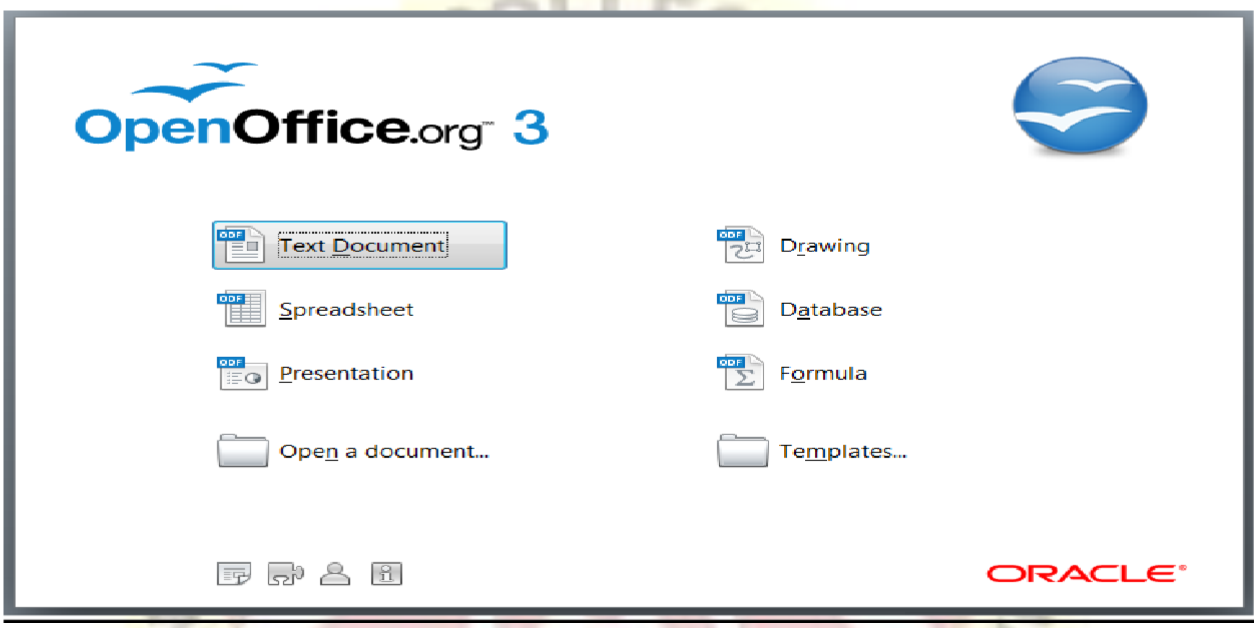
The incorrect format specifier causes the output to be corrupted, because the function printf is passed an integer instead of a floating-point number. Integersand floating-point numbers are stored in different formats in memory, and generally occupy different numbers of bytes, leading to a spurious result. The actual output shown above may differ, depending on the specific platform andenvironment.

Clearly, it is very dangerous to develop a program without checking for compiler warnings. If there are any functions which are not used correctly theycan cause the program to crash or produce incorrect results. Turning on the compiler warning option -Wall will catch many of the commonest errors whichoccur in C programming.

OpenOffice.org

OpenOffice included a word processor (Writer), a spreadsheet (Calc), a presentation application (Impress), a drawing application (Draw), a formula editor (Math), and a database management application (Base).[9] Its default file format was the OpenDocument Format (ODF), an ISO/IEC standard, which originated with OpenOffice.org. It could also read a wide variety of other file formats, with particular attention to those from Microsoft Office.

OpenOffice.org was primarily developed for Linux, Microsoft Windows and Solaris, and later for OS X, with ports to other operating systems.It was distributed under the GNU Lesser General Public License version 3 (LGPL); early versions were also available under the Sun Industry Standards Source License (SISSL).



Why Apache OpenOffice

*Why should I use Apache OpenOffice?*

Apache OpenOffice is the leading **open-source office software suite** for **wordprocessing**, **spreadsheets**, **presentations**, **graphics**, **databases** and more. It isavailable in **many languages** and works on all **common computers**. It stores all your data in an **international open standard format** and can also read andwrite files from other common office software packages. It can be downloadedand used completely **free of charge** for **any purpose**.

- Great software

  Apache OpenOffice is the result of over twenty years‘ software engineering. Designed from the start as a single piece of software, it has aconsistency other products cannot match. A completely open development process means that anyone can report bugs, request new features, or enhance the software. The result: Apache OpenOffice does everything you want your office software to do, the way you want it to.

- Easy to use

  Apache OpenOffice is easy to learn, and if you‘re already using anotheroffice software package, you‘ll take to OpenOffice straight away. Our world-wide native-language community means that OpenOffice is probably available and supported in your own language. And if you already have files from another office package - OpenOffice will probably read them with no difficulty.

- and it‘s free

  Best of all, Apache OpenOffice can be downloaded and used entirely **free** of any license fees. Like all Apache Software Foundation software, Apache OpenOffice is free to use. Apache OpenOffice is released under the Apache 2.0 License. This means you may use it for any purpose - domestic, commercial, educational, public administration.You may install it on as many computers as you like. You may make copies and give them away to family, friends, students, employees - anyone you like.

*Where is Apache OpenOffice currently used?*

- Governments
- Education
- Businesses
- Not for profits
- IT Businesses
- F/OSS advocates

## UNIT IV

### Open Source Project

**How to Get Started With Open Source Contribution?**

As the name suggests, **Open-Source** is something that is open for allwhich means there is no genuine owner of it. Open source is defined as *software or project for which the original source code is made freely available which may be redistributed and modified/re-modified according tothe requirement of the user*.

Sounds interesting, right? It is pretty much interesting. So Open **sourcesoftware** is basically software with source code that anyone can **inspect, modify, update, and enhance** as per his/her necessity or work.

**Questions to Ask Before Starting an Open Source Project**

1. Can we financially sponsor the project? Do we have an internal executive champion?
2. Is it possible to join efforts with an existing open source project?
3. Can we launch and maintain the project using the OSS model?
4. What constitutes success?
How do we measure it?
5. Will the project be able to attract outside enterprise participation (from the start)?
6. Is there enough external interest to form and grow a developer community?

**Important Terminologies and File Names**

**Open source contribution** requires you to know basic git commands and also knowing how to navigate your way on Github. You can check out Git Hub Guides, to get the basics. Below are some basic terminologies that may help you to contribute to an open-source project.

- **Author**: The person/s who created the project.
- **Owner**: The person/s who has administrative ownership of the project.
- **Contributors**: The person/s who contribute to the project.

**Common File Names in Open Source Repositories**

- **LICENSE:** It is important for a project to have an open-source license. Without a license, a project is not open-source.
- **README:** It‘s an instruction manual that welcomes new community members to the project. Here you will get the explanation that why the project is useful and how to get started.
- **CONTRIBUTING:** Contributing docs help people contribute to the project. Here you will get the explanation that what types of contributionsare needed and how the process works.
- **CODE_OF_CONDUCT**: The code of conduct sets ground rules for participants‘ behavior associated and helps to facilitate a friendly, welcoming environment.

Here are a few ways in which you can contribute to an open-source project…

- You can submit a bug fix.
- You can add a new feature.
- You can update the documentation.
- You can answer or give suggestions.
- You can review code.
- You can create a new issue

Firstly you need to find some open source projects to contribute. Search for **GitHub** projects that are open-source and have issues that are been labeled good-first issues, beginners-friendly, easy, e.t.c.

## How to start an Open-Source Software Projects

- **Determine the goals**

Every open source project solves a specific problem. Talk with colleagues, chats, forums, and share your idea. It all helps you on the first steps to understand important things, like which solutions already exist, and to hear criticism. Talk with people who already have open source projects. They can give you very valuable advice, so don't be afraid to ask and take the initiative.

One important bit of advice which I got at that stage is to pay attention in thefirst place on the documentation of the project. You can have a very good project, but no one will spend the time to understand how it works.

The most important aspect, without which further steps are impossible, is motivation. The idea of the project should inspire you primarily. Most often people get used to the tools with which they work and fall into a comfort zone,so external opinions may be ambiguous.

- **Planning**

The choice of a certain task manager is a matter of taste. It should have a clearpicture of the tasks and stages of your project.

Divide tasks into sub-tasks. Ideally, if one task does not take more than 3–4 hours, it is important to enjoy the implementation of small tasks. This will helpto avoid burnout and loss of motivation.

Use pivotal tracker. The main advantage is a free version for open source projects where you can sort tasks by type (feature, bug, chore, release), andgroup them into releases and determined deadlines.

- **Documentation**

Every open source project should contain these things:

- README
- Open Source license
- Contributing guidelines
- Changelog

- The README file not only explains how to use your project, but also thepurpose of your project. If you do not know how to properly write a README file, you can look at other known open source projects or use
  a template.
- The license guarantees that others can use, copy and modify the source code of the project. You need to add this file to each repository with youropen source project. MIT and Apache 2.0 GPLv3 are the most popular licenses for open source projects. If you are not sure what to choose, youcan use this convenient service.
- The CONTRIBUTING file will help other developers contribute to theproject. At the first steps of the project, it is not necessary to pay close attention to this file. You can use the already prepared template from another project.

- Changelog contains a supported, chronologically-ordered list of significant changes for each version. As with the CONTRIBUTING file, Ido not advise paying special attention to this at an early stage.

- **Versioning**

To track important changes for users and contributors, there is a semantic version. The version number contains numbers and adheres to the followingpattern X.Y.Z.

- X major release
- Y minor release
- Z patch release

- **Continuous integration / Continuous delivery**

To automatically run tests and build, use Travis CI. It‘s also a good idea to addbadges to display the successful assembly of the build in the wizard, the test coverage (Codecov), and the documentation (Inch CI).

## Maintain Open-Source Software Projects

### Introduction

When you maintain an open-source software repository, you‘re taking on aleadership role. Whether you‘re the founder of a project who released it to the public for use and contributions, or you‘re working on a team and are maintaining one specific aspect of the project, you are going to be providing animportant service to the larger developer community.

While open-source contributions through pull requests from the developer community are crucial for ensuring that software is as useful as it can be for end users, maintainers have a real impact on shaping the overall project. Repository maintainers are extremely involved in the open-source projects they manage, from day-to-day organization and development, to interfacing with the public and providing prompt and effective feedback to contributors.

- *Write Useful Documentation*

Documentation that is thorough, well-organized, and serves the intended communities of your project will help expand your user base. Over time, youruser base will become the contributors to your open-source project.

Since you‘ll be thinking through the code you are creating anyway, and mayeven be jotting down notes, it can be worthwhile to incorporate documentation as part of your development process while it is fresh in your mind. You may even want to consider writing the documentation before the code, following thephilosophy of a documentation-driven development approach that documents features first and develops those features after writing out what they will do.

Along with your code, there are a few files of documentation that you‘llwant to keep in your top-level directory:

- README.md file that provides a summary of the project and your goals.
- CONTRIBUTING.md file with contribution instructions.
- License for your software, which can encourage more contributions. Readmore about choosing an open-source license here.

Documentation can come in many forms and can target different audiences. As part of your documentation, and depending on the scope of your work, you may decide to do one or more of the following:

- A **general guide** to introduce users to the project
- **Tutorials** to walk people through different use cases
- **FAQs** to address frequently asked questions that users may have
- **Troubleshooting guides** to help users resolve problems
- An **API reference** to provides users with a quick way to look up APIinformation
- **Release notes** with known bugs to let users know what to expect in eachrelease
- **Planned features** to keep track of and explain what is coming up in thefuture
- **Video walkthroughs** to provide users with a multimedia approach toyour software

Your project may be better-suited to certain kinds of documentation than others, but providing more than one approach to the software will help your userbase better understand how to interact with your work.

### *Organize Issues*

**Issues** are typically a way to keep track of or report bugs, or to request newfeatures to be added to the code base. Open-source repository hosting services like GitHub, GitLab and Bitbucket will provide you with an interface for yourself and others to keep track of issues within your repository. When releasing open-source code to the public, you should expect to have issues opened by the community of users. Organizing and prioritizing issues will give you a good road map of upcoming work on your project.

Issues should represent concrete tasks that need to be done on the source code, and you will need to prioritize them accordingly. You and your team will have an understanding of the amount of time and energy you or contributors candevote to filed issues, and together you can work collaboratively to makedecisions and create an actionable plan.

Working to organize issues as best you can will keep your project up to date and relevant to its community of users. Remove issues that are outside of the scope of your project or become stale, and prioritize the others so that you are able to make continuous progress.

### *Make Contributing Rewarding*

The more you welcome contributors to your project and reward their efforts, the more likely you‘ll be to encourage more contributions. To get people started, you‘ll want to include a CONTRIBUTING.md filein the top-level of your repository, and a pointer to that file in your README.md file.

A good file on contributing will outline how to get started working on the project as a developer. You may want to offer a step-by-step guide, or provide a checklist for developers to follow, explaining how to successfully get their code merged into the project through a pull request.

### *Build Your Community*

By empowering users through documentation, being responsive to issues, and encouraging them to participate, you are already well on your way to building out the community around your open-source project. Users that you keep happy and who you treat as collaborators will in turn promote your software.

Additionally, you can work to promote your project through various avenues:

- Blogging
- Releasing overview or walkthrough videos
- Maintaining a mailing list
- Being active on social media channels
- Collaborating with similar or related projects and cross-promoting them

## Open-source hardware development

Open-source hardware is proposed as a bridge for the technological, educational and cultural gaps between developing and developed countries. Open-source hardware organizations are introduced with statistics of activities.

Open source hardware is based on publishing all necessary data about the hardware. The design specification, HDL files, simulation test benches, synthesis results, utilization instructions and interfaces to other systems should be documented [6]. The openness of necessary design documentation and its disclosure to the public should be governed by the terms of GPL like licenses [7].

All information is disclosed for free, according to the terms of GPL-like licenses. The EDA tools used to develop open hardware should also be open. Openness of resources is a must to allow the community to reuse, develop andimprove open designs.

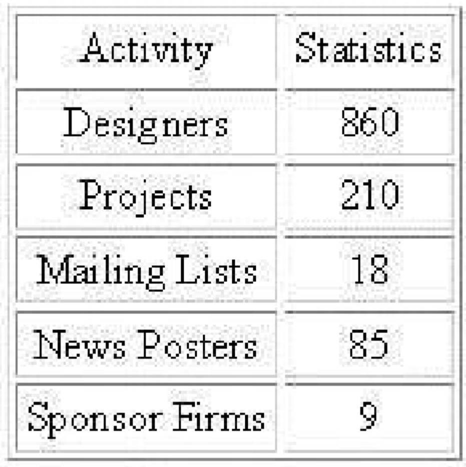
Open source hardware has a set of business models that include the following:

- **Design distribution** — Companies can pack sets of designs and sell thedistribution just like Linux distributions. The OpenTech CD-ROM is anexample of this method.
- **Design technical support** — Experts can give support for Open designs. Asics.ws is a company that follows this model by releasing IPcores and charging customers for technical support.
- **Design implementation** — Companies can implement the designs, sell them and pay royalties to original designers, according to their release license.
- **Releasing** — the release of open designs under the control of GPL-compatible licenses can occur whenever a silicon implementation is considered commercially.

Open source can help build a high-tech base in these regions due to thefollowing considerations:

- Open-source hardware is an open resource for industrial and academicfields.
- Lack of high-tech activities and firms.
- Lack of expertise in the high-tech field.
- Open-source allows interaction with a wide spectrum of experts in high-tech fields all over the world.
- HDL designs that address programmable logic devices based implementation platforms can be affordable, as chip manufacturing isbeyond economic capabilities.
- Open source might help focus the spotlight on high-tech talents and qualifications in developing countries that are hidden due to market constraints.

This section introduces organizations that adopted the open source hardware model. It provides statistics of activities that run under the , OpenCores , and Opencollector organizations.



| Activity | Statistics |
|---|---|
| Designers | 860 |
| Projects | 210 |
| Mailing Lists | 18 |
| News Posters | 85 |
| Sponsor Firms | 9 |

**Open Source Software in Education**

*Open source* refers to both the concept and practice of making program source code openly available. Users and developers have access to the core designing functionalities that enable them to modify or add features to the source code andredistribute it. Extensive collaboration and circulation are central to the open source movement.

❖ ***Learning and Digitization***

The digitization of education is a relatively new phenomenon that has transformed the education sector. Corporations and academic institutions have joined forces to further explore the potential for digitizing education through

- Virtual universities
- Online courses
- Education portals
- Courseware

*Virtual universities* are the best-known form of online education. Accredited virtual universities such as the University of Phoenix offer degrees in mainly professional courses taught largely by part-time faculty members from differentuniversities. Online consortia of academic institutions integrate related courses into programs delivered via a single virtual university.

*Online courses* are offered in a variety of forms by various sources. Some courses are offered by subsidiaries of renowned traditional universities, although many such courses are not accredited. The parent universities' namesact as a powerful draw for online students. Courses are also offered by organizations that create digital collections of study material culled from different academic sources.

*Education portals*, although not directly connected to the curriculum, have become an integral part of education. Since the late 1990s, some U.S. universities have outsourced e-mail and other Web services, site administrativefunctions, courseware, and other computer administrative services to software development and application companies.

*Courseware* is used in both the academic and corporate sectors, with development often outsourced to companies that provide study material for both online and offline purposes. Many companies use sophisticated computerized courses in their employee training programs.

The Internet offers opportunities to combine educational and economic goals on a common, globally accessible platform. This requires extensive technical support to create and sustain the software infrastructure on which digital education primarily depends. Most universities rely on software vendors to support, for instance, virtual learning environments and learning management systems that deliver online learning components. This puts considerable strain on their already overburdened finances.

Following a period of intense competition, the higher education software domain is dominated by a few major vendors,[4] with the risk of monopolization in the future.[5] This leaves academic institutions with one obvious option: to develop in-house systems to fulfill their IT requirements. Unfortunately, such projects often are isolated endeavors riddled with flaws or prohibitively expensive—or both.

Another option is to adopt the collaborative model of open source software development, which enables educational institutions to pool their financial andtechnical resources. In addition, a huge user community provides a variety of testing environments for the new software.

Open source software products tend to be more reliable and benefit from continuous development. This is one reason to invest liberally in developingopen source application software—to work out a more cost-effective way ofmeeting e-learning software challenges.

## Open Source Media Framework

**Open Source Media Framework** (OSMF) is a free, open-source, developmentframework for building video experiences on the web and desktop. OSMF is a pure ActionScript 3.0 based framework and is created by Adobe Systems.

OSMF is designed for content publishers, developers, and Adobe Flash platformecosystem partners—anyone who is incorporating video on their website.
OSMF simplifies development by providing out-of-the-box support for multiplemedia types, including video, audio, images, and SWF files. The extensible plug-in architecture enables additional features from third-party services, such as advertising insertion, rendering, tracking, and reporting for analytics,
and content delivery network (CDN) authentication.

OSMF supports RTMP and HTTP streaming, progressive download, sequentialand parallel compositions of video and other media, and layout in and outside the video player.

Adobe is actively working with many industry participants to achieve thefollowing goals:

- Improve video experiences
- Extend video and media experiences to mobile devices
- Help customers to simultaneously:
  - Shorten development time for media applications
  - Improve monetization of content
  - Reduce cost of development

# UNIT- V

## Open Source Ethics

### Ethical issues

Ethics are moral principles, or rules, which govern a person's attitudes andbehaviour.

Ethics apply to the use of computers as much as they do to other things in life.Ethical issues in computing include:

- ensuring public safety
- security of data

***Ensuring public safety***

Ensuring public safety is paramount. As new technologies are introduced, theybring safety concerns.

For example, driverless cars may soon be on the roads in the UK. The designersof driverless cars have not only had to ensure the safety of passengers, but also of other drivers and pedestrians. Ethics apply here as a situation may occur where the car's software has to decide who has safety priority, the passengers orother road users.

***Data security***

Personal data is precious and needs to be kept safe. Unfortunately, there arepeople that attempt to hack systems in order to gain access to other people's data. Social media accounts, phone mailboxes and networks that computers connect to are all prone to hacking.

Some people may also use malware to obtain data. Recent times have seen theincreased use of a type of malware known as ransomware. People who write ransomware do it to extort money from unsuspecting users. Once the ransomware infects a computer it encrypts data on it, denying users access unless a ransom is paid.

## Comparing Open Source Software vs Closed Source Software

For better understanding the peculiarities of open source software and closed source software, have a comparison of five basic aspects: pricing, security, support, source availability, and usability.

### *#1 Price Policy*

Open source often referred as free of cost software. It can, however, have costs for extras like assistance, additional services or added functionality. Thus, you may still pay for a service with OSS.

Closed source software is usually a paid software. The costs can vary depending on the complexity of the software. While the price can be higher, what you get is a better product, full support, functionality and innovation. However, most companies provide free trials to convince the purchaser that their software is theright fit.

### *#2 Security*

The question of security is very controversial as each software has two sides of the coin. The code of open source software can be viewed, shared and modifiedby the community, which means anyone can fix, upgrade and test the broken code. The bugs are fixed quickly, and the code is checked thoroughly after eachrelease. However, because of availability, the source code is open for hackers topractice on.

On the contrary, closed source software can be fixed only by a vendor. If something goes wrong with the software, you send a request and wait for the answer from the support team. Solving the problem can take much longer thancompared to OSC.

When it comes to choosing the most secure software, the answer is that each ofthem has its pros and cons. Thus, it is often a challenge for firms that work in aparticular industry.

***#3 Quality of Support***

Comparing open source and closed source software support, it is obvious that CSS is predominant in this case. The costs for it include an option to contact support and get it in one business day in most cases. The response is well organized and documented.

For open source software, such an option is not provided. The only supportoptions are forums, useful articles, and a hired expert

However, it is not surprising that using such kind of service you will not receivea high level of response.

***#4 Source Code Availability***

Open source software provides an ability to change the source code without anyrestrictions. Individual users can develop what they want and get benefits from innovation developed by others within the user community. As the source code is easily accessible, it enables the software developers to improve the already existing programs.

Closed source software is more restricted than open source software because thesource code cannot be changed or viewed. However, such limitation is what may contribute to CSS security and reliability.

***#5 Usability***

Usability is a painful subject of open source software. User guides are written for developers rather than to layperson users. Also, these manuals are failing toconform to the standards and structure.

For closed source software usability is one of the merits. Documentation is usually well-written and contains detailed instructions.

***Best Examples of OSS and CSS Shopping Carts***

The market is full of open source and closed source shopping carts. The basic difference lies in the price. Open source shopping cart systems are free, whereasfor closed source programs you will have to pay. With payment, you get customer support and confidence. Because open source shopping carts are free, they don't have such an option. However, their community on different forums is very active and always ready to help.

The benefits of open source solutions are primarily flexibility and scalability. You have full control over every aspect of your site's design, thanks to the open source code. When your business expands, and your monthly sales increase, youcan embrace it without being charged more for increased sales volume.

Closed source software is easier to work with for beginners or those who don'tknow how to code. Also, closed source websites are easier and faster to set upout of the box.

The top open source shopping carts are **Magento** and **OpenCart**,
and **BigCommerce** and **Shopify** are popular closed source platforms.

## Impact of Open Source

Open source software has spread far beyond Linux and is gaining enormousmomentum, according to a newly released IDC study.

The study, which analyzed IDC surveys from over 5,000 developers in 116 countries, finds the use of open source is increasing at a such a rate that it represents the most significant all-encompassing and long-term trend that thesoftware industry has seen since the early 1980s.

IDC believes that open source will eventually play a role in the life-cycle ofevery major software category, and will fundamentally change the value proposition of packaged software for customers.

Dr. Anthony Picardi, senior vice-president of global software research at IDC,said the use of open source beyond Linux is pervasive, used by almost three- quarters of organizations and spanning hundreds of thousands of projects.
Although open source will significantly reduce the industry opportunity over thenext 10 years, the real impact of open source is to sustain innovations in mature software markets, thus extending the useful life of software assets and saving customers money.

The study, entitled *Open Source in Global Software: Market Impact, Disruption, and Business Models*, finds that of the 5,000 survey respondents, open source software is being used by 71 per cent of the developers in the worldand is in production at 54 per cent of their organizations. In addition, half of theglobal developers claim that the use of open source is increasing in their organizations.
IDC identifies the following developments in the open source phenomenon:

- Over the next ten years, open source will extract a toll on the industry in thelow double digits, percentage wise, led by vicious price competition;

• Price effects are a less important impact of open source adoption than theeffect of open source on the entire life-cycle of software invention and innovation;

• Despite the proliferation of open source license forms, only three business models are important from an industry and an individual vendor success pointof view: the software revenue model, the public collective model, and the service broker model;

• Competitive success among vendors‘ open source markets will be determinedby a different set of core competencies than those required to invent and marketa new product.

The study examines the future impact of open source in the software life-cycle,the emerging business models for open source software markets, and the potential for market disruption. It also investigates the open source business models of Microsoft, IBM, Oracle, SAP, Sun Microsystems, CA, AOL, Amazon, and Perot Systems.

There‘s a discussion of –networked intelligence‖ and the key ingredients of open source sustained innovation business models.

## Open-source governance

**Open-source governance** (also known as **open politics**) is a politicalphilosophy which advocates the application of the philosophies of the open-source and open-content movements to democratic principles to enable any interested citizen to add to the creation of policy, as with a wiki document. Legislation is democratically opened to the general citizenry, employing their collective wisdom to benefit the decision-making process and improvedemocracy.[1]

Theories on ho to constrain, limit or enable this participation vary. Accordingly, there is no one dominant theory of how to go about authoring legislation with this approach. There are a wide array of projects and movements which are working on building open-source governance systems.[2]

Many left-libertarian and radical centrist organizations around the globe have begun advocating open-source governance and its related political ideas asa reformist alternative to current governance systems. Often, these groups have their origins in decentralized structures such as the Internet and place particular importance on the need for anonymity to protect an individual's right to free speech in democratic systems. Opinions vary, however, not least because theprinciples behind open-source government are still very loosely defined.[3]

*****